Towards Extensible Symbolic Formal Methods

José Meseguer

University of Illinois at Urbana-Champaign

Motivation

The use of decision procedures for theories axiomatizing data structures and commonly used functions is currently one of the most effective methods at the heart of state-of-the art

Motivation

The use of decision procedures for theories axiomatizing data structures and commonly used functions is currently one of the most effective methods at the heart of state-of-the art

- model checkers; and

Motivation

The use of decision procedures for theories axiomatizing data structures and commonly used functions is currently one of the most effective methods at the heart of state-of-the art

- model checkers; and
- theorem provers

Motivation

The use of decision procedures for theories axiomatizing data structures and commonly used functions is currently one of the most effective methods at the heart of state-of-the art

- model checkers; and
- theorem provers

It can represent infinite sets of states symbolically as states satisfying certain decidable constraints.

Motivation

The use of decision procedures for theories axiomatizing data structures and commonly used functions is currently one of the most effective methods at the heart of state-of-the art

- model checkers; and
- theorem provers

It can represent infinite sets of states symbolically as states satisfying certain decidable constraints.

In this way it can scale up verification efforts to handle large systems used in industrial practice.

Motivation II

This is great.

Motivation II

This is great. But what are the current limitations?

Motivation II

This is great. But what are the current limitations?

One important limitation is lack of extensibility.

Motivation II

This is great. But what are the current limitations?

One important limitation is lack of extensibility. For example:

Motivation II

This is great. But what are the current limitations?

One important limitation is lack of extensibility. For example:

- A satisfiability modulo T (SMT) solver has a (usually small) library of decidable theories

Motivation II

This is great. But what are the current limitations?

One important limitation is lack of extensibility. For example:

- A satisfiability modulo T (SMT) solver has a (usually small) library of decidable theories and can only support combinations of theories in the library, but no others.

Motivation II

This is great. But what are the current limitations?

One important limitation is lack of extensibility. For example:

- A satisfiability modulo T (SMT) solver has a (usually small) library of decidable theories and can only support combinations of theories in the library, but no others.
- A unification modulo T (UMT) solver has a (usually small) library of T-unification algorithms

Motivation II

This is great. But what are the current limitations?

One important limitation is lack of extensibility. For example:

- A satisfiability modulo T (SMT) solver has a (usually small) library of decidable theories and can only support combinations of theories in the library, but no others.
- A unification modulo T (UMT) solver has a (usually small) library of T-unification algorithms and can only support combinations of algorithms in its library, but no others.

Motivation II

This is great. But what are the current limitations?

One important limitation is lack of extensibility. For example:

- A satisfiability modulo T (SMT) solver has a (usually small) library of decidable theories and can only support combinations of theories in the library, but no others.
- A unification modulo T (UMT) solver has a (usually small) library of T-unification algorithms and can only support combinations of algorithms in its library, but no others.

Solving this extensibility problem is an eminently practical matter:

Motivation II

This is great. But what are the current limitations?

One important limitation is lack of extensibility. For example:

- A satisfiability modulo T (SMT) solver has a (usually small) library of decidable theories and can only support combinations of theories in the library, but no others.
- A unification modulo T (UMT) solver has a (usually small) library of T-unification algorithms and can only support combinations of algorithms in its library, but no others.

Solving this extensibility problem is an eminently practical matter: the more tasks we can automate,

Motivation II

This is great. But what are the current limitations?

One important limitation is lack of extensibility. For example:

- A satisfiability modulo T (SMT) solver has a (usually small) library of decidable theories and can only support combinations of theories in the library, but no others.
- A unification modulo T (UMT) solver has a (usually small) library of T-unification algorithms and can only support combinations of algorithms in its library, but no others.

Solving this extensibility problem is an eminently practical matter: the more tasks we can automate, the more can we scale up to solve harder and bigger problems.

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.
2. **SMT Solving**: infinite sets of states are represented as

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.
2. **SMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$,

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.
2. **SMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with t a term and ϕ a formula in a decidable theory T.

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.
2. **SMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with t a term and ϕ a formula in a decidable theory T.
3. **UMT Solving**: infinite sets of states are represented as

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.
2. **SMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with t a term and ϕ a formula in a decidable theory T.
3. **UMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$,

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.
2. **SMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with t a term and ϕ a formula in a decidable theory T.
3. **UMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with ϕ a positive QF formula in

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.
2. **SMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with t a term and ϕ a formula in a decidable theory T.
3. **UMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with ϕ a positive QF formula in an equational theory T having a unification algorithm.

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.
2. **SMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with t a term and ϕ a formula in a decidable theory T.
3. **UMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with ϕ a positive QF formula in an equational theory T having a unification algorithm.

Note that:

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.
2. **SMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with t a term and ϕ a formula in a decidable theory T.
3. **UMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with ϕ a positive QF formula in an equational theory T having a unification algorithm.

Note that:

- automata-based methods are less extensible; and

Symbolic Methods for Representing Infinite State Sets

The most common symbolic representation methods are:

1. **Automata-Based Methods**: infinite sets of states are represented and manipulated as languages $L(\mathcal{A})$ accepted by a certain kind of automaton $\mathcal{A}$.
2. **SMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with t a term and ϕ a formula in a decidable theory T.
3. **UMT Solving**: infinite sets of states are represented as constrained patterns $t \mid \phi$, with ϕ a positive QF formula in an equational theory T having a unification algorithm.

Note that:

- automata-based methods are less extensible; and
- UMT solving is an important special case of SMT solving.

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**,

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**,

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T,

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution,

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.
2. **SMT-Based**:

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.
2. **SMT-Based**:
   - **model checking**,

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.
2. **SMT-Based**:
   - **model checking**, e.g., tuple-based, array-based, and rewriting modulo SMT, model checkers.

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.
2. **SMT-Based**:
   - **model checking**, e.g., tuple-based, array-based, and rewriting modulo SMT, model checkers.
   - **theorem proving**,

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.
2. **SMT-Based**:
   - **model checking**, e.g., tuple-based, array-based, and rewriting modulo SMT, model checkers.
   - **theorem proving**, e.g., traditional general-purpose,

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.
2. **SMT-Based**:
   - **model checking**, e.g., tuple-based, array-based, and rewriting modulo SMT, model checkers.
   - **theorem proving**, e.g., traditional general-purpose, programming-language theorem proves,

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.
2. **SMT-Based**:
   - **model checking**, e.g., tuple-based, array-based, and rewriting modulo SMT, model checkers.
   - **theorem proving**, e.g., traditional general-purpose, programming-language theorem proves, and recent general-purpose theorem provers.

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.
2. **SMT-Based**:
   - **model checking**, e.g., tuple-based, array-based, and rewriting modulo SMT, model checkers.
   - **theorem proving**, e.g., traditional general-purpose, programming-language theorem proves, and recent general-purpose theorem provers.

Note that:

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.
2. **SMT-Based**:
   - **model checking**, e.g., tuple-based, array-based, and rewriting modulo SMT, model checkers.
   - **theorem proving**, e.g., traditional general-purpose, programming-language theorem proves, and recent general-purpose theorem provers.

Note that:

- All of these methods will benefit from greater extensibility.

Symbolic Formal Methods

Besides automata-based infinite-state model checking, the following symbolic formal methods are used:

1. **UMT-Based**:
   - **model checking**, e.g., narrowing-based model checkers.
   - **theorem proving**, e.g., superposition modulo T, higher-order resolution, and inductionless induction.
2. **SMT-Based**:
   - **model checking**, e.g., tuple-based, array-based, and rewriting modulo SMT, model checkers.
   - **theorem proving**, e.g., traditional general-purpose, programming-language theorem proves, and recent general-purpose theorem provers.

Note that:

- All of these methods will benefit from greater extensibility.
- UMT methods and SMT methods should be combined.

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable.

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available from tool implementers.

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available from tool implementers. To achieve this, the key distinction is between:

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available from tool implementers. To achieve this, the key distinction is between:

- **Theory-Specific** procedures, that work for a single theory T,

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available from tool implementers. To achieve this, the key distinction is between:

- **Theory-Specific** procedures, that work for a single theory T, e.g., AC unification, linear arithmetic, bit vectors, etc., and

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available from tool implementers. To achieve this, the key distinction is between:

- **Theory-Specific** procedures, that work for a single theory T, e.g., AC unification, linear arithmetic, bit vectors, etc., and
- **Theory-Generic** procedures, that work for an infinite class of theories.

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available from tool implementers. To achieve this, the key distinction is between:

- **Theory-Specific** procedures, that work for a single theory T, e.g., AC unification, linear arithmetic, bit vectors, etc., and
- **Theory-Generic** procedures, that work for an infinite class of theories. For example, folding variant narrowing is a theory-generic unification algorithm.

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available from tool implementers. To achieve this, the key distinction is between:

- **Theory-Specific** procedures, that work for a single theory T, e.g., AC unification, linear arithmetic, bit vectors, etc., and
- **Theory-Generic** procedures, that work for an infinite class of theories. For example, folding variant narrowing is a theory-generic unification algorithm.

Note that:

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available from tool implementers. To achieve this, the key distinction is between:

- **Theory-Specific** procedures, that work for a single theory T, e.g., AC unification, linear arithmetic, bit vectors, etc., and
- **Theory-Generic** procedures, that work for an infinite class of theories. For example, folding variant narrowing is a theory-generic unification algorithm.

Note that:

- in a theory-generic procedure the theory is easily defined by the user as an input to the procedure.

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available from tool implementers. To achieve this, the key distinction is between:

- **Theory-Specific** procedures, that work for a single theory T, e.g., AC unification, linear arithmetic, bit vectors, etc., and
- **Theory-Generic** procedures, that work for an infinite class of theories. For example, folding variant narrowing is a theory-generic unification algorithm.

Note that:

- in a theory-generic procedure the theory is easily defined by the user as an input to the procedure.
- a theory-generic SMT solving procedure would be very useful.

Theory-Specific vs. Theory-Generic Procedures

The goal of extensible formal methods is that decision procedures should be easily user-definable. Instead, now they are only available from tool implementers. To achieve this, the key distinction is between:

- **Theory-Specific** procedures, that work for a single theory T, e.g., AC unification, linear arithmetic, bit vectors, etc., and
- **Theory-Generic** procedures, that work for an infinite class of theories. For example, folding variant narrowing is a theory-generic unification algorithm.

Note that:

- in a theory-generic procedure the theory is easily defined by the user as an input to the procedure.
- a theory-generic SMT solving procedure would be very useful. Variant-based satisfiability is such a procedure.

Plan of This Tutorial

In this tutorial I will:

Plan of This Tutorial

In this tutorial I will:

1. briefly review folding variant narrowing

Plan of This Tutorial

In this tutorial I will:

1. briefly review folding variant narrowing for theories satisfying the finite variant property (FVP)

Plan of This Tutorial

In this tutorial I will:

1. briefly review folding variant narrowing for theories satisfying the finite variant property (FVP) as a theory-generic finitary unification algorithm.

Plan of This Tutorial

In this tutorial I will:

1. briefly review folding variant narrowing for theories satisfying the finite variant property (FVP) as a theory-generic finitary unification algorithm.

2. show how folding variant narrowing can be extended to

Plan of This Tutorial

In this tutorial I will:

1. briefly review folding variant narrowing for theories satisfying the finite variant property (FVP) as a theory-generic finitary unification algorithm.

2. show how folding variant narrowing can be extended to variant-based satisfiability,

Plan of This Tutorial

In this tutorial I will:

1. briefly review folding variant narrowing for theories satisfying the finite variant property (FVP) as a theory-generic finitary unification algorithm.
2. show how folding variant narrowing can be extended to variant-based satisfiability, a theory-generic SMT solving procedure.

Plan of This Tutorial

In this tutorial I will:

1. briefly review folding variant narrowing for theories satisfying the finite variant property (FVP) as a theory-generic finitary unification algorithm.

2. show how folding variant narrowing can be extended to variant-based satisfiability, a theory-generic SMT solving procedure.

3. explain how folding variant narrowing supports formal verification tools such as:

Plan of This Tutorial

In this tutorial I will:

1. briefly review folding variant narrowing for theories satisfying the finite variant property (FVP) as a theory-generic finitary unification algorithm.
2. show how folding variant narrowing can be extended to variant-based satisfiability, a theory-generic SMT solving procedure.
3. explain how folding variant narrowing supports formal verification tools such as:
   - Maude-NPA Protocol Analyzer

Plan of This Tutorial

In this tutorial I will:

1. briefly review folding variant narrowing for theories satisfying the finite variant property (FVP) as a theory-generic finitary unification algorithm.
2. show how folding variant narrowing can be extended to variant-based satisfiability, a theory-generic SMT solving procedure.
3. explain how folding variant narrowing supports formal verification tools such as:
   - Maude-NPA Protocol Analyzer
   - Maude's Symbolic LTL Model Checker.

Acknowledgements

The work on:

Acknowledgements

The work on:

1. Folding Variant Narrowing is joint with S. Escobar and R. Sasse;

Acknowledgements

The work on:

1. Folding Variant Narrowing is joint with S. Escobar and R. Sasse;
2. Variant-Based satisfiability is joint with S. Skeirik and R. Gutiérrez;

Acknowledgements

The work on:

1. Folding Variant Narrowing is joint with S. Escobar and R. Sasse;
2. Variant-Based satisfiability is joint with S. Skeirik and R. Gutiérrez;
3. Maude-NPA is joint with C. Meadows, S. Escobar, and Ph.D. students at Univ. of Illinois at Urbana-Champaign, Technical University of Valencia, and University of Oslo;

Acknowledgements

The work on:

1. Folding Variant Narrowing is joint with S. Escobar and R. Sasse;
2. Variant-Based satisfiability is joint with S. Skeirik and R. Gutiérrez;
3. Maude-NPA is joint with C. Meadows, S. Escobar, and Ph.D. students at Univ. of Illinois at Urbana-Champaign, Technical University of Valencia, and University of Oslo;
4. Maude's Symbolic LTL Model Checker is joint with K. Bae and S. Escobar;

Acknowledgements

The work on:

1. Folding Variant Narrowing is joint with S. Escobar and R. Sasse;
2. Variant-Based satisfiability is joint with S. Skeirik and R. Gutiérrez;
3. Maude-NPA is joint with C. Meadows, S. Escobar, and Ph.D. students at Univ. of Illinois at Urbana-Champaign, Technical University of Valencia, and University of Oslo;
4. Maude's Symbolic LTL Model Checker is joint with K. Bae and S. Escobar;

Variants in a Nutshell

Consider an equational theory $(\Sigma, E \cup B)$,

Variants in a Nutshell

Consider an equational theory $(\Sigma, E \cup B)$, with B a set of axioms and E equations oriented as confluent, terminating and coherent rewrite rules.

Variants in a Nutshell

Consider an equational theory $(\Sigma, E \cup B)$, with B a set of axioms and E equations oriented as confluent, terminating and coherent rewrite rules.

Can think of a Σ-term t with variables as a functional expression

Variants in a Nutshell

Consider an equational theory $(\Sigma, E \cup B)$, with B a set of axioms and E equations oriented as confluent, terminating and coherent rewrite rules.

Can think of a Σ-term t with variables as a functional expression to be symbolically evaluated with E modulo B.

Variants in a Nutshell

Consider an equational theory $(\Sigma, E \cup B)$, with B a set of axioms and E equations oriented as confluent, terminating and coherent rewrite rules.

Can think of a Σ-term t with variables as a functional expression to be symbolically evaluated with E modulo B.

The Comon-Delaune notion of the E, B-variants of t

Variants in a Nutshell

Consider an equational theory $(\Sigma, E \cup B)$, with B a set of axioms and E equations oriented as confluent, terminating and coherent rewrite rules.

Can think of a Σ-term t with variables as a functional expression to be symbolically evaluated with E modulo B.

The Comon-Delaune notion of the E, B-variants of t describes the different symbolic results to which t can be evaluated.

Variants in a Nutshell

Consider an equational theory $(\Sigma, E \cup B)$, with B a set of axioms and E equations oriented as confluent, terminating and coherent rewrite rules.

Can think of a Σ-term t with variables as a functional expression to be symbolically evaluated with E modulo B.

The Comon-Delaune notion of the E, B-variants of t describes the different symbolic results to which t can be evaluated.

Symbolic evaluation is performed by

Variants in a Nutshell

Consider an equational theory $(\Sigma, E \cup B)$, with B a set of axioms and E equations oriented as confluent, terminating and coherent rewrite rules.

Can think of a Σ-term t with variables as a functional expression to be symbolically evaluated with E modulo B.

The Comon-Delaune notion of the E, B-variants of t describes the different symbolic results to which t can be evaluated.

Symbolic evaluation is performed by narrowing t with rules E

Variants in a Nutshell

Consider an equational theory $(\Sigma, E \cup B)$, with B a set of axioms and E equations oriented as confluent, terminating and coherent rewrite rules.

Can think of a Σ-term t with variables as a functional expression to be symbolically evaluated with E modulo B.

The Comon-Delaune notion of the E, B-variants of t describes the different symbolic results to which t can be evaluated.

Symbolic evaluation is performed by narrowing t with rules E modulo axioms B.

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \rightsquigarrow_{E,B} t'$

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \rightsquigarrow_{E,B} t'$ is defined iff there is:

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \rightsquigarrow_{E,B} t'$ is defined iff there is:

- a non-variable position $p \in Pos(t)$;

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \leadsto_{E,B} t'$ is defined iff there is:

- a non-variable position $p \in Pos(t)$;
- a rule $l \to r$ in E; and

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \rightsquigarrow_{E,B} t'$ is defined iff there is:

- a non-variable position $p \in Pos(t)$;
- a rule $l \to r$ in E; and
- a B-unifier σ

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \rightsquigarrow_{E,B} t'$ is defined iff there is:

- a non-variable position $p \in Pos(t)$;
- a rule $l \to r$ in E; and
- a B-unifier σ such that $\sigma(t|_p) =_B \sigma(l)$, and $t' = \sigma(t[r]_p)$.

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \leadsto_{E,B} t'$ is defined iff there is:

- a non-variable position $p \in Pos(t)$;
- a rule $l \to r$ in E; and
- a B-unifier σ such that $\sigma(t|_p) =_B \sigma(l)$, and $t' = \sigma(t[r]_p)$.

A complete set of variants of t

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \leadsto_{E,B} t'$ is defined iff there is:

- a non-variable position $p \in Pos(t)$;
- a rule $l \rightarrow r$ in E; and
- a B-unifier σ such that $\sigma(t|_p) =_B \sigma(l)$, and $t' = \sigma(t[r]_p)$.

A complete set of variants of t can be computed as those t' such that $t \leadsto^{*}_{E,B} t'$ and t' is in E, B-normal form.

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \leadsto_{E,B} t'$ is defined iff there is:

- a non-variable position $p \in Pos(t)$;
- a rule $l \to r$ in E; and
- a B-unifier σ such that $\sigma(t|_p) =_B \sigma(l)$, and $t' = \sigma(t[r]_p)$.

A complete set of variants of t can be computed as those t' such that $t \leadsto^*_{E,B} t'$ and t' is in E, B-normal form.

Folding variant narrowing

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \leadsto_{E,B} t'$ is defined iff there is:

- a non-variable position $p \in Pos(t)$;
- a rule $l \to r$ in E; and
- a B-unifier σ such that $\sigma(t|_p) =_B \sigma(l)$, and $t' = \sigma(t[r]_p)$.

A complete set of variants of t can be computed as those t' such that $t \leadsto^*_{E,B} t'$ and t' is in E, B-normal form.

Folding variant narrowing is a strategy to compute a complete set of most general variants.

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \rightsquigarrow_{E,B} t'$ is defined iff there is:

- a non-variable position $p \in Pos(t)$;
- a rule $l \to r$ in E; and
- a B-unifier σ such that $\sigma(t|_p) =_B \sigma(l)$, and $t' = \sigma(t[r]_p)$.

A complete set of variants of t can be computed as those t' such that $t \rightsquigarrow^*_{E,B} t'$ and t' is in E, B-normal form.

Folding variant narrowing is a strategy to compute a complete set of most general variants.

$(\Sigma, E \cup B)$ has the finite variant property (FVP)

Equational Narrowing in a Nutshell

For $(\Sigma, E \cup B)$ as above, the narrowing relation $t \rightsquigarrow_{E,B} t'$ is defined iff there is:

- a non-variable position $p \in Pos(t)$;
- a rule $l \rightarrow r$ in E; and
- a B-unifier σ such that $\sigma(t|_p) =_B \sigma(l)$, and $t' = \sigma(t[r]_p)$.

A complete set of variants of t can be computed as those t' such that $t \rightsquigarrow^*_{E,B} t'$ and t' is in E, B-normal form.

Folding variant narrowing is a strategy to compute a complete set of most general variants.

$(\Sigma, E \cup B)$ has the finite variant property (FVP) iff any term t has a finite set of most general variants.

An Example: Presburger Arithmetic is FVP

Let $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ be the Presburger arithmetic FVP two-sorted equational specification with:

An Example: Presburger Arithmetic is FVP

Let $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ be the Presburger arithmetic FVP two-sorted equational specification with:

- $\Sigma = \{0, 1, +, >, \top, \bot\}$,

An Example: Presburger Arithmetic is FVP

Let $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ be the Presburger arithmetic FVP two-sorted equational specification with:

- $\Sigma = \{0, 1, +, >, \top, \bot\}$,
- E two equations, defining $>$, oriented as rewrite rules $m + n + 1 > n \to \top$ and $n > n + m \to \bot$, and

An Example: Presburger Arithmetic is FVP

Let $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ be the Presburger arithmetic FVP two-sorted equational specification with:

- $\Sigma = \{0, 1, +, >, \top, \bot\}$,
- E two equations, defining $>$, oriented as rewrite rules $m + n + 1 > n \to \top$ and $n > n + m \to \bot$, and
- ACU the axioms of associativity commutativity (AC) and unit 0 (U) for +.

An Example: Presburger Arithmetic is FVP

Let $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ be the Presburger arithmetic FVP two-sorted equational specification with:

- $\Sigma = \{0, 1, +, >, \top, \bot\}$,
- E two equations, defining $>$, oriented as rewrite rules $m + n + 1 > n \to \top$ and $n > n + m \to \bot$, and
- ACU the axioms of associativity commutativity (AC) and unit 0 (U) for +.

The initial algebra of $\mathcal{N}_{+,>}$ is the Presburger natural numbers.

An Example: Presburger Arithmetic is FVP

Let $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ be the Presburger arithmetic FVP two-sorted equational specification with:

- $\Sigma = \{0, 1, +, >, \top, \bot\}$,
- E two equations, defining $>$, oriented as rewrite rules $m + n + 1 > n \to \top$ and $n > n + m \to \bot$, and
- ACU the axioms of associativity commutativity (AC) and unit 0 (U) for +.

The initial algebra of $\mathcal{N}_{+,>}$ is the Presburger natural numbers.

Folding variant narrowing computes the following three most general variants of the term $x > y$:

An Example: Presburger Arithmetic is FVP

Let $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ be the Presburger arithmetic FVP two-sorted equational specification with:

- $\Sigma = \{0, 1, +, >, \top, \bot\}$,
- E two equations, defining $>$, oriented as rewrite rules $m + n + 1 > n \to \top$ and $n > n + m \to \bot$, and
- ACU the axioms of associativity commutativity (AC) and unit 0 (U) for +.

The initial algebra of $\mathcal{N}_{+,>}$ is the Presburger natural numbers.

Folding variant narrowing computes the following three most general variants of the term $x > y$:

- $x > y$ itself, with identity substitution

An Example: Presburger Arithmetic is FVP

Let $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ be the Presburger arithmetic FVP two-sorted equational specification with:

- $\Sigma = \{0, 1, +, >, \top, \bot\}$,
- E two equations, defining $>$, oriented as rewrite rules $m + n + 1 > n \to \top$ and $n > n + m \to \bot$, and
- ACU the axioms of associativity commutativity (AC) and unit 0 (U) for +.

The initial algebra of $\mathcal{N}_{+,>}$ is the Presburger natural numbers.

Folding variant narrowing computes the following three most general variants of the term $x > y$:

- $x > y$ itself, with identity substitution
- $\top$, with substitution $\{x \mapsto 1 + n + m, y \mapsto n\}$,

An Example: Presburger Arithmetic is FVP

Let $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ be the Presburger arithmetic FVP two-sorted equational specification with:

- $\Sigma = \{0, 1, +, >, \top, \bot\}$,
- E two equations, defining $>$, oriented as rewrite rules $m + n + 1 > n \to \top$ and $n > n + m \to \bot$, and
- ACU the axioms of associativity commutativity (AC) and unit 0 (U) for +.

The initial algebra of $\mathcal{N}_{+,>}$ is the Presburger natural numbers.

Folding variant narrowing computes the following three most general variants of the term $x > y$:

- $x > y$ itself, with identity substitution
- $\top$, with substitution $\{x \mapsto 1 + n + m, y \mapsto n\}$,
- $\bot$ with substitution $\{x \mapsto n, y \mapsto n + m\}$.

Variant Unification as Folding Variant Narrowing

Unification modulo Presburger Arithmetic $\mathcal{N}_{+,>}$ is computed by folding variant narrowing by just:

Variant Unification as Folding Variant Narrowing

Unification modulo Presburger Arithmetic $\mathcal{N}_{+,>}$ is computed by folding variant narrowing by just:

- adding a binary operator $\_ \equiv \_$ for solving equations and

Variant Unification as Folding Variant Narrowing

Unification modulo Presburger Arithmetic $\mathcal{N}_{+,>}$ is computed by folding variant narrowing by just:

- adding a binary operator $\_ \equiv \_$ for solving equations and
- the single rewrite rule $x \equiv x \to \top$.

Variant Unification as Folding Variant Narrowing

Unification modulo Presburger Arithmetic $\mathcal{N}_{+,>}$ is computed by folding variant narrowing by just:

- adding a binary operator $\_ \equiv \_$ for solving equations and
- the single rewrite rule $x \equiv x \to \top$.

Then the unifiers of two terms u, v modulo $\mathcal{N}_{+,>}$

Variant Unification as Folding Variant Narrowing

Unification modulo Presburger Arithmetic $\mathcal{N}_{+,>}$ is computed by folding variant narrowing by just:

- adding a binary operator $\_ \equiv \_$ for solving equations and
- the single rewrite rule $x \equiv x \to \top$.

Then the unifiers of two terms u, v modulo $\mathcal{N}_{+,>}$ are precisely the substitutions associated to the variants of the form $\top$ of the term $u \equiv v$.

Variant Unification as Folding Variant Narrowing

Unification modulo Presburger Arithmetic $\mathcal{N}_{+,>}$ is computed by folding variant narrowing by just:

- adding a binary operator $\_ \equiv \_$ for solving equations and
- the single rewrite rule $x \equiv x \to \top$.

Then the unifiers of two terms u, v modulo $\mathcal{N}_{+,>}$ are precisely the substitutions associated to the variants of the form $\top$ of the term $u \equiv v$.

Since $\mathcal{N}_{+,>}$ is FVP, there is a finite number of variants of $u \equiv v$,

Variant Unification as Folding Variant Narrowing

Unification modulo Presburger Arithmetic $\mathcal{N}_{+,>}$ is computed by folding variant narrowing by just:

- adding a binary operator $\_ \equiv \_$ for solving equations and
- the single rewrite rule $x \equiv x \to \top$.

Then the unifiers of two terms u, v modulo $\mathcal{N}_{+,>}$ are precisely the substitutions associated to the variants of the form $\top$ of the term $u \equiv v$.

Since $\mathcal{N}_{+,>}$ is FVP, there is a finite number of variants of $u \equiv v$, i.e., Presburger Arithmetic $\mathcal{N}_{+,>}$-unification is finitary.

Variant Unification as Folding Variant Narrowing

Unification modulo Presburger Arithmetic $\mathcal{N}_{+,>}$ is computed by folding variant narrowing by just:

- adding a binary operator $\_ \equiv \_$ for solving equations and
- the single rewrite rule $x \equiv x \to \top$.

Then the unifiers of two terms u, v modulo $\mathcal{N}_{+,>}$ are precisely the substitutions associated to the variants of the form $\top$ of the term $u \equiv v$.

Since $\mathcal{N}_{+,>}$ is FVP, there is a finite number of variants of $u \equiv v$, i.e., Presburger Arithmetic $\mathcal{N}_{+,>}$-unification is finitary.

For example, $x > y \equiv y > x$ has the single unifier $\{x \mapsto y\}$ modulo $\mathcal{N}_{+,>}$.

Constructor Variants and Constructor Unifiers

In $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ the predicate $>$ is a defined symbol: it evaluates to either $\top$ or $\bot$.

Constructor Variants and Constructor Unifiers

In $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ the predicate $>$ is a defined symbol: it evaluates to either $\top$ or $\bot$. Instead, the other operators $\Omega = \{0, 1, +, \top, \bot\}$ are constructor symbols.

Constructor Variants and Constructor Unifiers

In $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ the predicate $>$ is a defined symbol: it evaluates to either $\top$ or $\bot$. Instead, the other operators $\Omega = \{0, 1, +, \top, \bot\}$ are constructor symbols.

A constructor variant

Constructor Variants and Constructor Unifiers

In $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ the predicate $>$ is a defined symbol: it evaluates to either $\top$ or $\bot$. Instead, the other operators $\Omega = \{0, 1, +, \top, \bot\}$ are constructor symbols.

A constructor variant is variant that has constructor instances.

Constructor Variants and Constructor Unifiers

In $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ the predicate $>$ is a defined symbol: it evaluates to either $\top$ or $\bot$. Instead, the other operators $\Omega = \{0, 1, +, \top, \bot\}$ are constructor symbols.

A constructor variant is variant that has constructor instances. For example, $\top$ and $\bot$ are constructor variants of $x > y$,

Constructor Variants and Constructor Unifiers

In $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ the predicate $>$ is a defined symbol: it evaluates to either $\top$ or $\bot$. Instead, the other operators $\Omega = \{0, 1, +, \top, \bot\}$ are constructor symbols.

A constructor variant is variant that has constructor instances. For example, $\top$ and $\bot$ are constructor variants of $x > y$, but $x > y$ is not.

A constructor R, B-unifier of $u \equiv v$

Constructor Variants and Constructor Unifiers

In $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ the predicate $>$ is a defined symbol: it evaluates to either $\top$ or $\bot$. Instead, the other operators $\Omega = \{0, 1, +, \top, \bot\}$ are constructor symbols.

A constructor variant is variant that has constructor instances. For example, $\top$ and $\bot$ are constructor variants of $x > y$, but $x > y$ is not.

A constructor R, B-unifier of $u \equiv v$ is a B-unifier of $u' \equiv v'$ where u', v' are constructor variants of u, resp. v.

Constructor Variants and Constructor Unifiers

In $\mathcal{N}_{+,>} = (\Sigma, E \cup ACU)$ the predicate $>$ is a defined symbol: it evaluates to either $\top$ or $\bot$. Instead, the other operators $\Omega = \{0, 1, +, \top, \bot\}$ are constructor symbols.

A constructor variant is variant that has constructor instances. For example, $\top$ and $\bot$ are constructor variants of $x > y$, but $x > y$ is not.

A constructor R, B-unifier of $u \equiv v$ is a B-unifier of $u' \equiv v'$ where u', v' are constructor variants of u, resp. v. For example, $\{x \mapsto y\}$ is not a constructor unifier of $x > z \equiv y > z$.

OS-Compact Theories

An equational order-sorted theory (Ω, G) is OS-compact iff:

OS-Compact Theories

An equational order-sorted theory (Ω, G) is OS-compact iff:

1. G-unification is finitary, and

OS-Compact Theories

An equational order-sorted theory (Ω, G) is OS-compact iff:

1. G-unification is finitary, and
2. a conjunction of disequalities $\bigwedge_{1 \leq i \leq n} u_i \neq v_i$ where all variables have infinite sorts is satisfiable in $T_{\Omega/G}$ iff $u_i \neq_G v_i$, $1 \leq i \leq n$.

OS-Compact Theories

An equational order-sorted theory (Ω, G) is OS-compact iff:

1. G-unification is finitary, and
2. a conjunction of disequalities $\bigwedge_{1 \leq i \leq n} u_i \neq v_i$ where all variables have infinite sorts is satisfiable in $T_{\Omega/G}$ iff $u_i \neq_G v_i$, $1 \leq i \leq n$.

Theorem. If (Ω, G) is OS-compact, then satisfiability of QF Ω-formulas in $T_{\Omega/G}$ is decidable.

Remark. The notion of OS-compact theory and the above theorem generalize a similar notion and theorem by H. Comon.

OS-Compact Theories

An equational order-sorted theory (Ω, G) is OS-compact iff:

1. G-unification is finitary, and
2. a conjunction of disequalities $\bigwedge_{1 \leq i \leq n} u_i \neq v_i$ where all variables have infinite sorts is satisfiable in $T_{\Omega/G}$ iff $u_i \neq_G v_i$, $1 \leq i \leq n$.

Theorem. If (Ω, G) is OS-compact, then satisfiability of QF Ω-formulas in $T_{\Omega/G}$ is decidable.

Remark. The notion of OS-compact theory and the above theorem generalize a similar notion and theorem by H. Comon.

Theorem. (Ω, B) is OS-compact for any Ω with B any combination of associativity and/or commutativity and/or identity axioms, except associativity without commutativity.

Main Theorem Let $(\Sigma, E \cup B)$ be FVP with B having a finitary unification algorithm,

Variant-Based Satisfiability

Main Theorem Let $(\Sigma, E \cup B)$ be FVP with B having a finitary unification algorithm, and such that (Ω, E_Ω) specifies the Ω-reduct algebra of $T_{\Sigma/E \cup B}$

Main Theorem Let $(\Sigma, E \cup B)$ be FVP with B having a finitary unification algorithm, and such that (Ω, E_Ω) specifies the Ω-reduct algebra of $T_{\Sigma/E \cup B}$ (i.e., $T_{\Sigma/E \cup B}|_\Omega \cong T_{\Omega/E_\Omega}$)

Main Theorem Let $(\Sigma, E \cup B)$ be FVP with B having a finitary unification algorithm, and such that (Ω, E_Ω) specifies the Ω-reduct algebra of $T_{\Sigma/E\cup B}$ (i.e., $T_{\Sigma/E\cup B}|_\Omega \cong T_{\Omega/E_\Omega}$) and is OS-compact.

Variant-Based Satisfiability

Main Theorem Let $(\Sigma, E \cup B)$ be FVP with B having a finitary unification algorithm, and such that (Ω, E_Ω) specifies the Ω-reduct algebra of $T_{\Sigma/E \cup B}$ (i.e., $T_{\Sigma/E \cup B}|_\Omega \cong T_{\Omega/E_\Omega}$) and is OS-compact.

Then satisfiability of QF Σ-formulas in $T_{\Sigma/E \cup B}$ is decidable.

Variant-Based Satisfiability

Main Theorem Let $(\Sigma, E \cup B)$ be FVP with B having a finitary unification algorithm, and such that (Ω, E_Ω) specifies the Ω-reduct algebra of $T_{\Sigma/E \cup B}$ (i.e., $T_{\Sigma/E \cup B}|_\Omega \cong T_{\Omega/E_\Omega}$) and is OS-compact.

Then satisfiability of QF Σ-formulas in $T_{\Sigma/E \cup B}$ is decidable.

Algorithm: Given conjunction of literals $\bigwedge G \wedge \bigwedge D$,

Variant-Based Satisfiability

Main Theorem Let $(\Sigma, E \cup B)$ be FVP with B having a finitary unification algorithm, and such that (Ω, E_Ω) specifies the Ω-reduct algebra of $T_{\Sigma/E\cup B}$ (i.e., $T_{\Sigma/E\cup B}|_\Omega \cong T_{\Omega/E_\Omega}$) and is OS-compact.

Then satisfiability of QF Σ-formulas in $T_{\Sigma/E\cup B}$ is decidable.

Algorithm: Given conjunction of literals $\bigwedge G \wedge \bigwedge D$, with G equalities and D disequalities:

Variant-Based Satisfiability

Main Theorem Let $(\Sigma, E \cup B)$ be FVP with B having a finitary unification algorithm, and such that (Ω, E_Ω) specifies the Ω-reduct algebra of $T_{\Sigma/E\cup B}$ (i.e., $T_{\Sigma/E\cup B}|_\Omega \cong T_{\Omega/E_\Omega}$) and is OS-compact.

Then satisfiability of QF Σ-formulas in $T_{\Sigma/E\cup B}$ is decidable.

Algorithm: Given conjunction of literals $\bigwedge G \wedge \bigwedge D$, with G equalities and D disequalities:

1. compute constructor $E \cup B$-unifiers α of $\bigwedge G$,

Variant-Based Satisfiability

Main Theorem Let $(\Sigma, E \cup B)$ be FVP with B having a finitary unification algorithm, and such that (Ω, E_Ω) specifies the Ω-reduct algebra of $T_{\Sigma/E\cup B}$ (i.e., $T_{\Sigma/E\cup B}|_\Omega \cong T_{\Omega/E_\Omega}$) and is OS-compact.

Then satisfiability of QF Σ-formulas in $T_{\Sigma/E\cup B}$ is decidable.

Algorithm: Given conjunction of literals $\bigwedge G \wedge \bigwedge D$, with G equalities and D disequalities:

1. compute constructor $E \cup B$-unifiers α of $\bigwedge G$,
2. compute the constructor E, B-variants $\bigwedge D'$ of $\bigwedge D\alpha$, and

Variant-Based Satisfiability

Main Theorem Let $(\Sigma, E \cup B)$ be FVP with B having a finitary unification algorithm, and such that (Ω, E_Ω) specifies the Ω-reduct algebra of $T_{\Sigma/E \cup B}$ (i.e., $T_{\Sigma/E \cup B}|_\Omega \cong T_{\Omega/E_\Omega}$) and is OS-compact.

Then satisfiability of QF Σ-formulas in $T_{\Sigma/E \cup B}$ is decidable.

Algorithm: Given conjunction of literals $\bigwedge G \wedge \bigwedge D$, with G equalities and D disequalities:

1. compute constructor $E \cup B$-unifiers α of $\bigwedge G$,
2. compute the constructor E, B-variants $\bigwedge D'$ of $\bigwedge D\alpha$, and
3. for each $u' \neq v'$ in $\bigwedge D'$ check that $u' \neq_{E_\Omega} v'$.

Example of Variant-Based Satisfiability

Consider the quantifier-free formula:

$$head(l) > head(l') = \top \wedge head(l) > 1{+}1{+}1 = \bot \wedge \{(1{+}1); nil\} \subseteq \{l, l', \emptyset\} \neq tt$$

in the composition of: (i) Presburger arithmetic $\mathcal{N}_{+,>}$, (ii) the parameterized theory of lists $\mathcal{L}[X]$, and (iii) the parameterized theory of hereditarily finite (HF) sets $\mathcal{H}[Y]$. These three theories and their composition are FVP and have decidable satisfiability.

To decide satisfiability we:

1. first solve the sytem of equations $head(l) > head(l') = \top \wedge head(l) > 1 + 1 + 1 = \bot$ modulo the composed theory. There are six constructor unifiers. The first is: $\alpha = \{l \mapsto (1 + 1 + 1); l_1, l' \mapsto (1 + 1); l_2\}$.
2. This shows that the formula is satisfiable, because $\{(1 + 1); nil\} \subseteq \{(1 + 1 + 1); l_1, (1 + 1); l_2, \emptyset\} \neq tt$, is irreducible by the equations for $\subseteq$ modulo AGU.

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

1. HF sets do not seem to be supported by any of the SMT solvers in the Wikipedia SMT solver page,

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

1. HF sets do not seem to be supported by any of the SMT solvers in the Wikipedia SMT solver page, yet HF sets and the three theories are easily definable by rewrite rules.

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

1. HF sets do not seem to be supported by any of the SMT solvers in the Wikipedia SMT solver page, yet HF sets and the three theories are easily definable by rewrite rules.
2. Even if Presburger arithmetic, lists, and HF sets were available in a standard SMT solver, a Nelson-Oppen (NO) combination procedure would have been needed;

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

1. HF sets do not seem to be supported by any of the SMT solvers in the Wikipedia SMT solver page, yet HF sets and the three theories are easily definable by rewrite rules.
2. Even if Presburger arithmetic, lists, and HF sets were available in a standard SMT solver, a Nelson-Oppen (NO) combination procedure would have been needed; here we just take the union of the three theories:

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

1. HF sets do not seem to be supported by any of the SMT solvers in the Wikipedia SMT solver page, yet HF sets and the three theories are easily definable by rewrite rules.
2. Even if Presburger arithmetic, lists, and HF sets were available in a standard SMT solver, a Nelson-Oppen (NO) combination procedure would have been needed; here we just take the union of the three theories: no NO combination is needed.

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

1. HF sets do not seem to be supported by any of the SMT solvers in the Wikipedia SMT solver page, yet HF sets and the three theories are easily definable by rewrite rules.
2. Even if Presburger arithmetic, lists, and HF sets were available in a standard SMT solver, a Nelson-Oppen (NO) combination procedure would have been needed; here we just take the union of the three theories: no NO combination is needed.

Many other theories can be made decidable this way, including:

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

1. HF sets do not seem to be supported by any of the SMT solvers in the Wikipedia SMT solver page, yet HF sets and the three theories are easily definable by rewrite rules.
2. Even if Presburger arithmetic, lists, and HF sets were available in a standard SMT solver, a Nelson-Oppen (NO) combination procedure would have been needed; here we just take the union of the three theories: no NO combination is needed.

Many other theories can be made decidable this way, including: (i) any FVP theory whose constructor subspecification is OS-compact;

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

1. HF sets do not seem to be supported by any of the SMT solvers in the Wikipedia SMT solver page, yet HF sets and the three theories are easily definable by rewrite rules.
2. Even if Presburger arithmetic, lists, and HF sets were available in a standard SMT solver, a Nelson-Oppen (NO) combination procedure would have been needed; here we just take the union of the three theories: no NO combination is needed.

Many other theories can be made decidable this way, including: (i) any FVP theory whose constructor subspecification is OS-compact; (ii) all constructor-selector parameterized data types;

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

1. HF sets do not seem to be supported by any of the SMT solvers in the Wikipedia SMT solver page, yet HF sets and the three theories are easily definable by rewrite rules.
2. Even if Presburger arithmetic, lists, and HF sets were available in a standard SMT solver, a Nelson-Oppen (NO) combination procedure would have been needed; here we just take the union of the three theories: no NO combination is needed.

Many other theories can be made decidable this way, including: (i) any FVP theory whose constructor subspecification is OS-compact; (ii) all constructor-selector parameterized data types; (iii) sets, multisets and HF sets parameterized types;

Example of Variant-Based Satisfiability II

Although this is a simple example, it illustrates the extensible nature of variant-based satisfiability because:

1. HF sets do not seem to be supported by any of the SMT solvers in the Wikipedia SMT solver page, yet HF sets and the three theories are easily definable by rewrite rules.
2. Even if Presburger arithmetic, lists, and HF sets were available in a standard SMT solver, a Nelson-Oppen (NO) combination procedure would have been needed; here we just take the union of the three theories: no NO combination is needed.

Many other theories can be made decidable this way, including: (i) any FVP theory whose constructor subspecification is OS-compact; (ii) all constructor-selector parameterized data types; (iii) sets, multisets and HF sets parameterized types; (iv) various numeric functions; and (v) many cryptographic theories.

Rewriting Logic in a Nutshell

Rewriting logic is a flexible logical framework to specify concurrent systems and also logics.

Rewriting Logic in a Nutshell

Rewriting logic is a flexible logical framework to specify concurrent systems and also logics.

- A concurrent system is specified as rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ where:

Rewriting Logic in a Nutshell

Rewriting logic is a flexible logical framework to specify concurrent systems and also logics.

- A concurrent system is specified as rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ where:
  - Σ is signature defining the syntax of the system and of its states

Rewriting Logic in a Nutshell

Rewriting logic is a flexible logical framework to specify concurrent systems and also logics.

- A concurrent system is specified as rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ where:
    - Σ is signature defining the syntax of the system and of its states
    - $E \cup B$ is a set of equations defining system's states as an algebraic data type

Rewriting Logic in a Nutshell

Rewriting logic is a flexible logical framework to specify concurrent systems and also logics.

- A concurrent system is specified as rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ where:
  - Σ is signature defining the syntax of the system and of its states
  - $E \cup B$ is a set of equations defining system's states as an algebraic data type
  - R is a set of rewrite rules of the form $t \to t'$, specifying system's local concurrent transitions.

Rewriting Logic in a Nutshell

Rewriting logic is a flexible logical framework to specify concurrent systems and also logics.

- A concurrent system is specified as rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ where:
  - Σ is signature defining the syntax of the system and of its states
  - $E \cup B$ is a set of equations defining system's states as an algebraic data type
  - R is a set of rewrite rules of the form $t \to t'$, specifying system's local concurrent transitions.
- Rewriting logic deduction consists of applying rewriting rules R concurrently, modulo the equations $E \cup B$.

Rewriting Logic in a Nutshell

Rewriting logic is a flexible logical framework to specify concurrent systems and also logics.

- A concurrent system is specified as rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ where:
  - Σ is signature defining the syntax of the system and of its states
  - $E \cup B$ is a set of equations defining system's states as an algebraic data type
  - R is a set of rewrite rules of the form $t \to t'$, specifying system's local concurrent transitions.
- Rewriting logic deduction consists of applying rewriting rules R concurrently, modulo the equations $E \cup B$.

Maude provides several model checkers based on narrowing with rules R modulo and FVP theory $E \cup B$ such as:

Rewriting Logic in a Nutshell

Rewriting logic is a flexible logical framework to specify concurrent systems and also logics.

- A concurrent system is specified as rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ where:
  - Σ is signature defining the syntax of the system and of its states
  - $E \cup B$ is a set of equations defining system's states as an algebraic data type
  - R is a set of rewrite rules of the form $t \to t'$, specifying system's local concurrent transitions.
- Rewriting logic deduction consists of applying rewriting rules R concurrently, modulo the equations $E \cup B$.

Maude provides several model checkers based on narrowing with rules R modulo and FVP theory $E \cup B$ such as: Maude-NPA and

Rewriting Logic in a Nutshell

Rewriting logic is a flexible logical framework to specify concurrent systems and also logics.

- A concurrent system is specified as rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ where:
  - Σ is signature defining the syntax of the system and of its states
  - $E \cup B$ is a set of equations defining system's states as an algebraic data type
  - R is a set of rewrite rules of the form $t \to t'$, specifying system's local concurrent transitions.
- Rewriting logic deduction consists of applying rewriting rules R concurrently, modulo the equations $E \cup B$.

Maude provides several model checkers based on narrowing with rules R modulo and FVP theory $E \cup B$ such as: Maude-NPA and Maude's Symbolic LTL Model Checker.

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by:

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by: (i) representing sets of states as terms with variables, and

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by: (i) representing sets of states as terms with variables, and (ii) performing narrowing with rules R modulo $E \cup B$,

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by: (i) representing sets of states as terms with variables, and (ii) performing narrowing with rules R modulo $E \cup B$, where the narrowing relation $t \rightsquigarrow_{R/E \cup B} t'$

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by: (i) representing sets of states as terms with variables, and (ii) performing narrowing with rules R modulo $E \cup B$, where the narrowing relation $t \rightsquigarrow_{R/E \cup B} t'$ is defined iff there is:

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by: (i) representing sets of states as terms with variables, and (ii) performing narrowing with rules R modulo $E \cup B$, where the narrowing relation $t \rightsquigarrow_{R/E \cup B} t'$ is defined iff there is:

- a rule $l \rightarrow r$ in R; and

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by: (i) representing sets of states as terms with variables, and (ii) performing narrowing with rules R modulo $E \cup B$, where the narrowing relation $t \leadsto_{R/E \cup B} t'$ is defined iff there is:

- a rule $l \rightarrow r$ in R; and
- a $E \cup B$-variant unifier σ

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by: (i) representing sets of states as terms with variables, and (ii) performing narrowing with rules R modulo $E \cup B$, where the narrowing relation $t \leadsto_{R/E \cup B} t'$ is defined iff there is:

- a rule $l \to r$ in R; and
- a $E \cup B$-variant unifier σ such that $\sigma(t) =_{(E \cup B)} \sigma(l)$, and $t' = \sigma(r)$.

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by: (i) representing sets of states as terms with variables, and (ii) performing narrowing with rules R modulo $E \cup B$, where the narrowing relation $t \rightsquigarrow_{R/E \cup B} t'$ is defined iff there is:

- a rule $l \rightarrow r$ in R; and
- a $E \cup B$-variant unifier σ such that $\sigma(t) =_{(E \cup B)} \sigma(l)$, and $t' = \sigma(r)$.

This method is complete for reachability analysis: an instance of the states described by t can reach an instance of those described by t' in the system specified by $\mathcal{R}$ iff

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by: (i) representing sets of states as terms with variables, and (ii) performing narrowing with rules R modulo $E \cup B$, where the narrowing relation $t \rightsquigarrow_{R/E \cup B} t'$ is defined iff there is:

- a rule $l \to r$ in R; and
- a $E \cup B$-variant unifier σ such that $\sigma(t) =_{(E \cup B)} \sigma(l)$, and $t' = \sigma(r)$.

This method is complete for reachability analysis: an instance of the states described by t can reach an instance of those described by t' in the system specified by $\mathcal{R}$ iff $t \rightsquigarrow_{R/E \cup B} t'$.

Rule Narrowing in a Nutshell

We can model check a concurrent system specified by a topmost rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ with $E \cup B$ FVP by: (i) representing sets of states as terms with variables, and (ii) performing narrowing with rules R modulo $E \cup B$, where the narrowing relation $t \rightsquigarrow_{R/E \cup B} t'$ is defined iff there is:

- a rule $l \to r$ in R; and
- a $E \cup B$-variant unifier σ such that $\sigma(t) =_{(E \cup B)} \sigma(l)$, and $t' = \sigma(r)$.

This method is complete for reachability analysis: an instance of the states described by t can reach an instance of those described by t' in the system specified by $\mathcal{R}$ iff $t \rightsquigarrow_{R/E \cup B} t'$.

Note that narrowing happens at two levels:

- with rules R modulo $E \cup B$ to perform symbolic transitions
- with oriented equations E modulo B to compute $E \cup B$-unifiers by folding variant narrowing.

The Maude-NPA Crypto Protocol Analyzer

The Maude-NPA tool of Escobar, Meadows and Meseguer, analyzes crypto protocols modeled as $\mathcal{P} = (\Sigma, G \cup B, R)$ by narrowing with rules R modulo FVP equations $G \cup B$.

The Maude-NPA Crypto Protocol Analyzer

The Maude-NPA tool of Escobar, Meadows and Meseguer, analyzes crypto protocols modeled as $\mathcal{P} = (\Sigma, G \cup B, R)$ by narrowing with rules R modulo FVP equations $G \cup B$.

Many protocols have been analyzed modulo non-trivial theories such as: (i) encryption-decryption; (ii) exclusive or; (iii) Diffie-Hellman exponentiation; (iv) homomorphic encryption, and combinations of such theories.

The Maude-NPA Crypto Protocol Analyzer

The Maude-NPA tool of Escobar, Meadows and Meseguer, analyzes crypto protocols modeled as $\mathcal{P} = (\Sigma, G \cup B, R)$ by narrowing with rules R modulo FVP equations $G \cup B$.

Many protocols have been analyzed modulo non-trivial theories such as: (i) encryption-decryption; (ii) exclusive or; (iii) Diffie-Hellman exponentiation; (iv) homomorphic encryption, and combinations of such theories.

Although Maude-NPA deals with unbounded sessions for which reachability is undecidable, its use of very effective symbolic state space reduction techiques often makes the state space finite, allowing full verification.

The Maude-NPA Crypto Protocol Analyzer

The Maude-NPA tool of Escobar, Meadows and Meseguer, analyzes crypto protocols modeled as $\mathcal{P} = (\Sigma, G \cup B, R)$ by narrowing with rules R modulo FVP equations $G \cup B$.

Many protocols have been analyzed modulo non-trivial theories such as: (i) encryption-decryption; (ii) exclusive or; (iii) Diffie-Hellman exponentiation; (iv) homomorphic encryption, and combinations of such theories.

Although Maude-NPA deals with unbounded sessions for which reachability is undecidable, its use of very effective symbolic state space reduction techiques often makes the state space finite, allowing full verification.

The tool is available at http://maude.cs.illinois.edu/w/index.php?title=Maude_Tools:_Maude-NPA

The Maude-NPA Crypto Protocol Analyzer (II)

Homomorphic encryption is challenging: the theories H and AGH are not FVP, and combining their unification algorithms with those of other theories is computationally expensive.

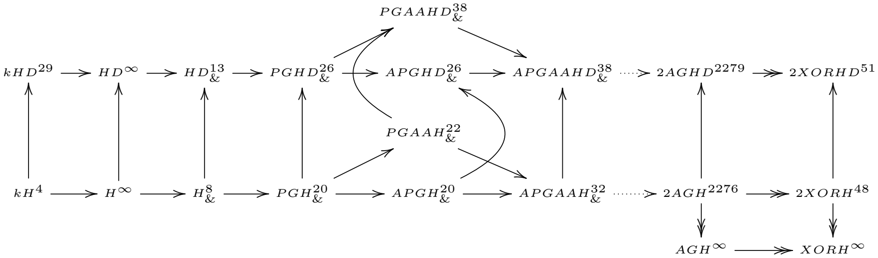
The Maude-NPA Crypto Protocol Analyzer (II)

Homomorphic encryption is challenging: the theories H and AGH are not FVP, and combining their unification algorithms with those of other theories is computationally expensive.

In joint work with Yang et al., several FVP theories of homomorphic encryption have been used with protocols in Maude-NPA by trading accuracy and variant complexity.

Homomorphic encryption is challenging: the theories *H* and *AGH* are not FVP, and combining their unification algorithms with those of other theories is computationally expensive.

In joint work with Yang et al., several FVP theories of homomorphic encryption have been used with protocols in Maude-NPA by trading accuracy and variant complexity.



Maude's Narrowing-Based LTL Model Checker

Many concurrent systems are infinite-state.

Maude's Narrowing-Based LTL Model Checker

Many concurrent systems are infinite-state.

The Maude Logical Bounded Model Checker tool developed by K. Bae with S. Escobar and J. Meseguer is an infinite-state model checker for LTL and LTLR properties supporting:

Maude's Narrowing-Based LTL Model Checker

Many concurrent systems are infinite-state.

The Maude Logical Bounded Model Checker tool developed by K. Bae with S. Escobar and J. Meseguer is an infinite-state model checker for LTL and LTLR properties supporting:

- Symbolic representation of states and transitions by narrowing with rules modulo FVP equations.

Maude's Narrowing-Based LTL Model Checker

Many concurrent systems are infinite-state.

The Maude Logical Bounded Model Checker tool developed by K. Bae with S. Escobar and J. Meseguer is an infinite-state model checker for LTL and LTLR properties supporting:

- Symbolic representation of states and transitions by narrowing with rules modulo FVP equations.
- State space reduction using state subsumption

Maude's Narrowing-Based LTL Model Checker

Many concurrent systems are infinite-state.

The Maude Logical Bounded Model Checker tool developed by K. Bae with S. Escobar and J. Meseguer is an infinite-state model checker for LTL and LTLR properties supporting:

- Symbolic representation of states and transitions by narrowing with rules modulo FVP equations.
- State space reduction using state subsumption
- Further reduction using equational abstractions

Maude's Narrowing-Based LTL Model Checker

Many concurrent systems are infinite-state.

The Maude Logical Bounded Model Checker tool developed by K. Bae with S. Escobar and J. Meseguer is an infinite-state model checker for LTL and LTLR properties supporting:

- Symbolic representation of states and transitions by narrowing with rules modulo FVP equations.
- State space reduction using state subsumption
- Further reduction using equational abstractions
- bounded model checking, which can detect a finite symbolic state space to provide full verification.

Maude's Narrowing-Based LTL Model Checker

Many concurrent systems are infinite-state.

The Maude Logical Bounded Model Checker tool developed by K. Bae with S. Escobar and J. Meseguer is an infinite-state model checker for LTL and LTLR properties supporting:

- Symbolic representation of states and transitions by narrowing with rules modulo FVP equations.
- State space reduction using state subsumption
- Further reduction using equational abstractions
- bounded model checking, which can detect a finite symbolic state space to provide full verification.

Lamport's Bakery Example

In Lamport's Bakery protocol for mutual exclusion each state with n processes:

Lamport's Bakery Example

In Lamport's Bakery protocol for mutual exclusion each state with n processes:

$$i\ ;\ j\ ;\ [k_1\,, m_1]\ldots[k_n, m_n]$$

- i: the current number in the bakery's number dispenser,

Lamport's Bakery Example

In Lamport's Bakery protocol for mutual exclusion each state with n processes:

$$i \; ; \; j \; ; \; [k_1, m_1] \ldots [k_n, m_n]$$

- i: the current number in the bakery's number dispenser,
- j: the number currently served,

Lamport's Bakery Example

In Lamport's Bakery protocol for mutual exclusion each state with n processes:

$$i \; ; \; j \; ; \; [k_1, m_1] \ldots [k_n, m_n]$$

- i: the current number in the bakery's number dispenser,
- j: the number currently served,
- $[k_l, m_l]$: a process k_l in a mode m_l,

Lamport's Bakery Example

In Lamport's Bakery protocol for mutual exclusion each state with n processes:

$$i \; ; \; j \; ; \; [k_1, m_1] \ldots [k_n, m_n]$$

- i: the current number in the bakery's number dispenser,
- j: the number currently served,
- $[k_l, m_l]$: a process k_l in a mode m_l, either `idle`, `wait(t)`, or `crit(t)`.

Lamport's Bakery Example

In Lamport's Bakery protocol for mutual exclusion each state with n processes:

$$i \; ; \; j \; ; \; [k_1, m_1] \ldots [k_n, m_n]$$

- i: the current number in the bakery's number dispenser,
- j: the number currently served,
- $[k_l, m_l]$: a process k_l in a mode m_l, either `idle`, `wait(t)`, or `crit(t)`.

Behaviors:

Lamport's Bakery Example

In Lamport's Bakery protocol for mutual exclusion each state with n processes:

$$i\ ;\ j\ ;\ [k_1, m_1] \ldots [k_n, m_n]$$

- i: the current number in the bakery's number dispenser,
- j: the number currently served,
- $[k_l, m_l]$: a process k_l in a mode m_l, either `idle`, `wait(t)`, or `crit(t)`.

Behaviors:

```
rl [wake]: N ; M ; [K, idle] PS => s N ; M ; [K, wait(N)] PS .
rl [crit]: N ; M ; [K, wait(M)] PS => N ; M ; [K, crit(M)] PS .
rl [exit]: N ; M ; [K, crit(M)] PS => N ; s M ; [K, idle] PS .
```

Lamport's Bakery Example

In Lamport's Bakery protocol for mutual exclusion each state with n processes:

$$i \; ; \; j \; ; \; [k_1, m_1] \ldots [k_n, m_n]$$

- i: the current number in the bakery's number dispenser,
- j: the number currently served,
- $[k_l, m_l]$: a process k_l in a mode m_l, either `idle`, `wait(t)`, or `crit(t)`.

Behaviors:

```
rl [wake]: N  ; M ; [K, idle] PS  => s N  ;  M ; [K, wait(N)] PS .
rl [crit]: N ; M ; [K, wait(M)] PS =>  N ;   M ; [K, crit(M)] PS .
rl [exit]: N ;  M  ; [K, crit(M)] PS =>  N ;  s M ; [K, idle] PS .
```

Mutual exclusion: $\Box ex?$ where:

Lamport's Bakery Example

In Lamport's Bakery protocol for mutual exclusion each state with n processes:

$$i\ ;\ j\ ;\ [k_1, m_1] \ldots [k_n, m_n]$$

- i: the current number in the bakery's number dispenser,
- j: the number currently served,
- $[k_l, m_l]$: a process k_l in a mode m_l, either `idle`, `wait(t)`, or `crit(t)`.

Behaviors:

```
rl [wake]: N  ; M ; [K, idle] PS   => s N  ;   M ; [K, wait(N)] PS .
rl [crit]: N ; M ; [K, wait(M)] PS =>   N ;    M ; [K, crit(M)] PS .
rl [exit]: N ;  M  ; [K, crit(M)] PS =>   N ;  s M ; [K, idle] PS .
```

Mutual exclusion: $\Box ex?$ where:

```
eq N ; M ; [K1, crit(M1)] [K2, crit(M2)] PS |= ex? = false .
```

Lamport's Bakery Example (II)

The commands below show the results of the bounded model checking with depth 10, and of full model checking using an equational abstraction, for an arbitrary number of processes.

Lamport's Bakery Example (II)

The commands below show the results of the bounded model checking with depth 10, and of full model checking using an equational abstraction, for an arbitrary number of processes.

```
Maude> (lmc [10] N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex? .)
logical model check in BAKERY-SAFETY-SATISFACTION :
  N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex?
result:
  no counterexample found within bound 10

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

Lamport's Bakery Example (II)

The commands below show the results of the bounded model checking with depth 10, and of full model checking using an equational abstraction, for an arbitrary number of processes.

```
Maude> (lmc [10] N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex? .)
logical model check in BAKERY-SAFETY-SATISFACTION :
  N:Nat ; N:Nat ; IS:ProcIdleSet |= [] ex?
result:
  no counterexample found within bound 10

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

The tool is available at

http://maude.cs.uiuc.edu/tools/lmc/

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and
- **Variant-Based Satisfiability**

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and
- **Variant-Based Satisfiability**

to make formal methods much more extensible.

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and
- **Variant-Based Satisfiability**

to make formal methods much more extensible.

I have also shown how symbolic model checkers such as:

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and
- **Variant-Based Satisfiability**

to make formal methods much more extensible.

I have also shown how symbolic model checkers such as: **Maude-NPA** and

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and
- **Variant-Based Satisfiability**

to make formal methods much more extensible.

I have also shown how symbolic model checkers such as: **Maude-NPA** and **Maude's Symbolic LTL Model Checker**

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and
- **Variant-Based Satisfiability**

to make formal methods much more extensible.

I have also shown how symbolic model checkers such as: **Maude-NPA** and **Maude's Symbolic LTL Model Checker** can benefit from narrowing with rules R modulo and FVP equational theory $E \cup B$ to verify infinite-state systems.

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and
- **Variant-Based Satisfiability**

to make formal methods much more extensible.

I have also shown how symbolic model checkers such as: **Maude-NPA** and **Maude's Symbolic LTL Model Checker** can benefit from narrowing with rules R modulo and FVP equational theory $E \cup B$ to verify infinite-state systems.

Variant satisfiability has already been implemented in Maude and has been applied to

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and
- **Variant-Based Satisfiability**

to make formal methods much more extensible.

I have also shown how symbolic model checkers such as: **Maude-NPA** and **Maude's Symbolic LTL Model Checker** can benefit from narrowing with rules R modulo and FVP equational theory $E \cup B$ to verify infinite-state systems.

Variant satisfiability has already been implemented in Maude and has been applied to deductive verification of concurrent systems

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and
- **Variant-Based Satisfiability**

to make formal methods much more extensible.

I have also shown how symbolic model checkers such as: **Maude-NPA** and **Maude's Symbolic LTL Model Checker** can benefit from narrowing with rules R modulo and FVP equational theory $E \cup B$ to verify infinite-state systems.

Variant satisfiability has already been implemented in Maude and has been applied to deductive verification of concurrent systems in the **Reachability Logic Theorem Prover** of S. Skeirik, A. Stefanescu, and J. Meseguer.

Conclusion: Towards Extensible Formal Methods

I have emphasized the importance of theory-generic symbolic methods such as:

- **Folding Variant Narrowing** and
- **Variant-Based Satisfiability**

to make formal methods much more extensible.

I have also shown how symbolic model checkers such as: **Maude-NPA** and **Maude's Symbolic LTL Model Checker** can benefit from narrowing with rules R modulo and FVP equational theory $E \cup B$ to verify infinite-state systems.

Variant satisfiability has already been implemented in Maude and has been applied to deductive verification of concurrent systems in the **Reachability Logic Theorem Prover** of S. Skeirik, A. Stefanescu, and J. Meseguer.

Using varian satisfiability in model checking remains ahead.