Course Automated Reasoning and Formal Verification
Module I: Automated Reasoning
Ch. 02: **Satisfiability Modulo Theories (SMT)**

Roberto Sebastiani

DISI, Università di Trento, Italy – `roberto.sebastiani@unitn.it`
URL: `https://disi.unitn.it/rseba/DIDATTICA/fm2023/`
Teaching assistant: Giuseppe Spallitta – `giuseppe.spallitta@unitn.it`

M.S. in Computer Science, Mathematics, & Artificial Intelligence Systems
Academic year 2022-2023

last update: Friday 21st February, 2025

Outline

1. Introduction
   - Basics on First-order Logic
   - What is a Theory?
   - Satisfiability Modulo Theories
   - Motivations and Goals of SMT

2. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for Theories of Interest (hints)
   - SMT for Combinations of Theories

3. Beyond Solving: Advanced SMT Functionalities
   - Proofs and Unsatisfiable Cores
   - All-SMT & Predicate Abstraction (hints)
   - SMT with Optimization (Optimization Modulo Theories)

Outline

1. Introduction
    - Basics on First-order Logic
    - What is a Theory?
    - Satisfiability Modulo Theories
    - Motivations and Goals of SMT

2. Efficient SMT solving
    - Combining SAT with Theory Solvers
    - Theory Solvers for Theories of Interest (hints)
    - SMT for Combinations of Theories

3. Beyond Solving: Advanced SMT Functionalities
    - Proofs and Unsatisfiable Cores
    - All-SMT & Predicate Abstraction (hints)
    - SMT with Optimization (Optimization Modulo Theories)

Outline

1. Introduction
   - Basics on First-order Logic
   - What is a Theory?
   - Satisfiability Modulo Theories
   - Motivations and Goals of SMT
2. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for Theories of Interest (hints)
   - SMT for Combinations of Theories
3. Beyond Solving: Advanced SMT Functionalities
   - Proofs and Unsatisfiable Cores
   - All-SMT & Predicate Abstraction (hints)
   - SMT with Optimization (Optimization Modulo Theories)

First-Order Logic (FOL)

- PL assumes world contains facts
  - atomic events
- FOL is structured: a world/state includes objects, each of which may have attributes of its own as well as relationships to other objects
- FOL assumes the world contains:
  - Objects:
    e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
  - Relations:
    e.g., red, round, bogus, prime, tall ...,
    brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - Functions:
    e.g., father of, best friend, one more than, end of, ...
- Allows to quantify on objects
  - ex: “All man are equal”, “some persons are left-handed”, ...

First-Order Logic (FOL)

- PL assumes world contains facts
  - atomic events
- FOL is structured: a world/state includes objects, each of which may have attributes of its own as well as relationships to other objects
- FOL assumes the world contains:
  - Objects:
    e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
  - Relations:
    e.g., red, round, bogus, prime, tall ...,
    brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - Functions:
    e.g., father of, best friend, one more than, end of, ...
- Allows to quantify on objects
  - ex: “All man are equal”, “some persons are left-handed”, ...

First-Order Logic (FOL)

- PL assumes world contains facts
    - atomic events
- FOL is structured: a world/state includes objects, each of which may have attributes of its own as well as relationships to other objects
- FOL assumes the world contains:
    - Objects:
      e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
    - Relations:
      e.g., red, round, bogus, prime, tall ...,
      brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
    - Functions:
      e.g., father of, best friend, one more than, end of, ...
- Allows to quantify on objects
    - ex: “All man are equal”, “some persons are left-handed”, ...

First-Order Logic (FOL)

- PL assumes world contains facts
    - atomic events
- FOL is structured: a world/state includes objects, each of which may have attributes of its own as well as relationships to other objects
- FOL assumes the world contains:
    - Objects:
      e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
    - Relations:
      e.g., red, round, bogus, prime, tall ...,
      brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
    - Functions:
      e.g., father of, best friend, one more than, end of, ...
- Allows to quantify on objects
    - ex: “All man are equal”, “some persons are left-handed”, ...

First-Order Logic (FOL)

- PL assumes world contains facts
    - atomic events
- FOL is structured: a world/state includes objects, each of which may have attributes of its own as well as relationships to other objects
- FOL assumes the world contains:
    - Objects:
      e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
    - Relations:
      e.g., red, round, bogus, prime, tall ...,
      brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
    - Functions:
      e.g., father of, best friend, one more than, end of, ...
- Allows to quantify on objects
    - ex: “All man are equal”, “some persons are left-handed”, ...

First-Order Logic (FOL)

- PL assumes world contains facts
  - atomic events
- FOL is structured: a world/state includes objects, each of which may have attributes of its own as well as relationships to other objects
- FOL assumes the world contains:
  - Objects:
    e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
  - Relations:
    e.g., red, round, bogus, prime, tall ...,
    brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - Functions:
    e.g., father of, best friend, one more than, end of, ...
- Allows to quantify on objects
  - ex: “All man are equal”, “some persons are left-handed”, ...

First-Order Logic (FOL)

- PL assumes world contains facts
  - atomic events
- FOL is structured: a world/state includes objects, each of which may have attributes of its own as well as relationships to other objects
- FOL assumes the world contains:
  - Objects:
    e.g., people, houses, numbers, theories, Jim Morrison, colors, basketball games, wars, centuries, ...
  - Relations:
    e.g., red, round, bogus, prime, tall ...,
    brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
  - Functions:
    e.g., father of, best friend, one more than, end of, ...
- Allows to quantify on objects
  - ex: “All man are equal”, “some persons are left-handed”, ...

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \land, \lor, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: “$=$” (also “$\neq$” s.t. “$a \neq b$” shortcut for “$\neg(a = b)$”)
- Quantifiers: “$\forall$” (“forall”), “$\exists$” (“exists”, aka “for some”)
- Punctuation Symbols: “,”, “(”, “)”

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

Syntax of FOL: Basic Elements

- Constant symbols: KingJohn, 2, UniversityofTrento,...
- Predicate symbols: Man(.), Brother(.,.), $(. > .)$, AllDifferent(...),...
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Brother(.,.)) or infix (e.g. $(. > .)$)
- Function symbols: Sqrt, LeftLeg, MotherOf
  - may have different arities (1,2,3,...)
  - may be prefix (e.g. Sqrt(.)) or infix (e.g. $(. + .)$)
- Variable symbols: x, y, a, b, ...
- Propositional Connectives: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$
- Equality: "$=$" (also "$\neq$" s.t. "$a \neq b$" shortcut for "$\neg(a = b)$")
- Quantifiers: "$\forall$" ("forall"), "$\exists$" ("exists", aka "for some")
- Punctuation Symbols: ",", "(", ")"

- Constants symbols are 0-ary function symbols
- Propositions are 0-ary predicates $\Longrightarrow$ PL subcase of FOL
- Signature: the set of predicate, function & constant symbols

FOL: Syntax

- Terms:
  - constant or variable or *function*(*term*$_1$, ..., *term*$_n$)
  - ex: KingJohn, x, LeftLeg(Richard), (z*log(2))
  - denote objects in the real world (aka domain)
- Atomic sentences (aka atomic formulas):
  - $\top$, $\bot$
  - *proposition* or *predicate*(*term*$_1$, ..., *term*$_n$) or *term*$_1$ $=$ *term*$_2$
  - $(Length(LeftLeg(Richard)) > Length(LeftLeg(KingJohn)))$
  - denote facts
- Non-atomic sentences/formulas:
  - $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, $\alpha \leftrightarrow \beta$, $\alpha \oplus \beta$,
    $\forall x.\alpha$, $\exists x.\alpha$ s.t. x (typically) occurs in α
  - Ex: $\forall y.(Italian(y) \rightarrow President(Mattarella, y))$
    $\exists x \forall y.President(x, y) \rightarrow \forall y \exists x.President(x, y)$
    $\forall x.(P(x) \wedge Q(x)) \leftrightarrow ((\forall x.P(x)) \wedge (\forall x.Q(x)))$
    $\forall x.(((x \geq 0) \wedge (x \leq \pi)) \rightarrow (sin(x) \geq 0))$
  - denote (complex) facts

FOL: Syntax

- Terms:
  - constant or variable or $function(term_1, \ldots, term_n)$
  - ex: KingJohn, x, LeftLeg(Richard), (z*log(2))
  - denote objects in the real world (aka domain)
- Atomic sentences (aka atomic formulas):
  - $\top, \bot$
  - $proposition$ or $predicate(term_1, \ldots, term_n)$ or $term_1 = term_2$
  - $(Length(LeftLeg(Richard)) > Length(LeftLeg(KingJohn)))$
  - denote facts
- Non-atomic sentences/formulas:
  - $\neg\alpha, \alpha \wedge \beta, \alpha \vee \beta, \alpha \rightarrow \beta, \alpha \leftrightarrow \beta, \alpha \oplus \beta$,
    $\forall x.\alpha, \exists x.\alpha$ s.t. x (typically) occurs in α
  - Ex: $\forall y.(Italian(y) \rightarrow President(Mattarella, y))$
    $\exists x \forall y.President(x, y) \rightarrow \forall y \exists x.President(x, y)$
    $\forall x.(P(x) \wedge Q(x)) \leftrightarrow ((\forall x.P(x)) \wedge (\forall x.Q(x)))$
    $\forall x.(((x \geq 0) \wedge (x \leq \pi)) \rightarrow (sin(x) \geq 0))$
  - denote (complex) facts

FOL: Syntax

- Terms:
  - constant or variable or $function(term_1, ..., term_n)$
  - ex: KingJohn, x, LeftLeg(Richard), (z*log(2))
  - denote objects in the real world (aka domain)
- Atomic sentences (aka atomic formulas):
  - $\top$, $\bot$
  - $proposition$ or $predicate(term_1, ..., term_n)$ or $term_1 = term_2$
  - $(Length(LeftLeg(Richard)) > Length(LeftLeg(KingJohn)))$
  - denote facts
- Non-atomic sentences/formulas:
  - $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, $\alpha \leftrightarrow \beta$, $\alpha \oplus \beta$,
    $\forall x.\alpha$, $\exists x.\alpha$ s.t. x (typically) occurs in α
  - Ex: $\forall y.(Italian(y) \rightarrow President(Mattarella, y))$
    $\exists x \forall y.President(x, y) \rightarrow \forall y \exists x.President(x, y)$
    $\forall x.(P(x) \wedge Q(x)) \leftrightarrow ((\forall x.P(x)) \wedge (\forall x.Q(x)))$
    $\forall x.(((x \geq 0) \wedge (x \leq \pi)) \rightarrow (sin(x) \geq 0))$
  - denote (complex) facts

FOL: Syntax

- Terms:
  - constant or variable or *function*($term_1, ..., term_n$)
  - ex: KingJohn, x, LeftLeg(Richard), (z*log(2))
  - denote objects in the real world (aka domain)
- Atomic sentences (aka atomic formulas):
  - $\top$, $\bot$
  - *proposition* or *predicate*($term_1, ..., term_n$) or $term_1 = term_2$
  - $(Length(LeftLeg(Richard)) > Length(LeftLeg(KingJohn)))$
  - denote facts
- Non-atomic sentences/formulas:
  - $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, $\alpha \leftrightarrow \beta$, $\alpha \oplus \beta$,
    $\forall x.\alpha$, $\exists x.\alpha$ s.t. x (typically) occurs in α
  - Ex: $\forall y.(Italian(y) \rightarrow President(Mattarella, y))$
    $\exists x \forall y.President(x, y) \rightarrow \forall y \exists x.President(x, y)$
    $\forall x.(P(x) \wedge Q(x)) \leftrightarrow ((\forall x.P(x)) \wedge (\forall x.Q(x)))$
    $\forall x.(((x \geq 0) \wedge (x \leq \pi)) \rightarrow (sin(x) \geq 0))$
  - denote (complex) facts

Truth in FOL: Intuitions

- Sentences are true with respect to a model
    - containing a domain and an interpretation
- The domain contains ≥ 1 objects (domain elements) and relations and functions over them
- An interpretation specifies referents for
    - variables $\rightarrow$ objects
    - constant symbols $\rightarrow$ objects
    - predicate symbols $\rightarrow$ relations
    - function symbols $\rightarrow$ functional relations
- An atomic sentence $P(t_1, ..., t_n)$ is true in an interpretation iff the objects referred to by $t_1, ..., t_n$ are in the relation referred to by P

Truth in FOL: Intuitions

- Sentences are true with respect to a model
  - containing a domain and an interpretation
- The domain contains ≥ 1 objects (domain elements) and relations and functions over them
- An interpretation specifies referents for
  - variables $\rightarrow$ objects
  - constant symbols $\rightarrow$ objects
  - predicate symbols $\rightarrow$ relations
  - function symbols $\rightarrow$ functional relations
- An atomic sentence $P(t_1, \ldots, t_n)$ is true in an interpretation iff the objects referred to by $t_1, \ldots, t_n$ are in the relation referred to by P

Truth in FOL: Intuitions

- Sentences are true with respect to a model
  - containing a domain and an interpretation
- The domain contains ≥ 1 objects (domain elements) and relations and functions over them
- An interpretation specifies referents for
  - variables $\rightarrow$ objects
  - constant symbols $\rightarrow$ objects
  - predicate symbols $\rightarrow$ relations
  - function symbols $\rightarrow$ functional relations
- An atomic sentence $P(t_1, ..., t_n)$ is true in an interpretation iff the objects referred to by $t_1, ..., t_n$ are in the relation referred to by P

Truth in FOL: Intuitions

- Sentences are true with respect to a model
  - containing a domain and an interpretation
- The domain contains $\geq$ 1 objects (domain elements) and relations and functions over them
- An interpretation specifies referents for
  - variables $\rightarrow$ objects
  - constant symbols $\rightarrow$ objects
  - predicate symbols $\rightarrow$ relations
  - function symbols $\rightarrow$ functional relations
- An atomic sentence $P(t_1, ..., t_n)$ is true in an interpretation iff the objects referred to by $t_1, ..., t_n$ are in the relation referred to by P

FOL: Semantics

FOL Models (aka possible worlds)

- A model $\mathcal{M}$ is a pair $\langle\mathcal{D},\mathcal{I}\rangle$ ($\langle$*domain*, *interpretation*$\rangle$)
- Domain $\mathcal{D}$: a non-empty set of objects (aka domain elements)
- Interpretation $\mathcal{I}$: a (non-injective) map on elements of the signature
  - constant symbols $\longmapsto$ domain elements:
    a constant symbol C is mapped into a particular object $[C]^{\mathcal{I}}$ in $\mathcal{D}$
  - predicate symbols $\longmapsto$ domain relations:
    a k-ary predicate $P(...)$ is mapped into a subset $[P]^{\mathcal{I}}$ of $\mathcal{D}^k$
    (i.e., the set of object tuples satisfying the predicate in this world)
  - functions symbols $\longmapsto$ domain functions:
    a k-ary function f is mapped into a domain function $[f]^{\mathcal{I}} : \mathcal{D}^k \longmapsto \mathcal{D}$ ($[f]^{\mathcal{I}}$ must be total)

(we denote by $[.]^{\mathcal{I}}$ the result of the interpretation $\mathcal{I}$)

An Interpretation $\mathcal{I}$ is extended to assign domain values to variables, domain values to terms and truth values to formulas.

FOL: Semantics

FOL Models (aka possible worlds)

- A model $\mathcal{M}$ is a pair $\langle \mathcal{D}, \mathcal{I} \rangle$ ($\langle$*domain*, *interpretation*$\rangle$)
- Domain $\mathcal{D}$: a non-empty set of objects (aka domain elements)
- Interpretation $\mathcal{I}$: a (non-injective) map on elements of the signature
  - constant symbols $\longmapsto$ domain elements:
    a constant symbol C is mapped into a particular object $[C]^{\mathcal{I}}$ in $\mathcal{D}$
  - predicate symbols $\longmapsto$ domain relations:
    a k-ary predicate $P(...)$ is mapped into a subset $[P]^{\mathcal{I}}$ of $\mathcal{D}^k$
    (i.e., the set of object tuples satisfying the predicate in this world)
  - functions symbols $\longmapsto$ domain functions:
    a k-ary function f is mapped into a domain function $[f]^{\mathcal{I}} : \mathcal{D}^k \longmapsto \mathcal{D}$ ($[f]^{\mathcal{I}}$ must be total)

(we denote by $[.]^{\mathcal{I}}$ the result of the interpretation $\mathcal{I}$)

An Interpretation $\mathcal{I}$ is extended to assign domain values to variables, domain values to terms and truth values to formulas.

FOL: Semantics

FOL Models (aka possible worlds)

- A model $\mathcal{M}$ is a pair $\langle \mathcal{D}, \mathcal{I} \rangle$ ($\langle$*domain*, *interpretation*$\rangle$)
- Domain $\mathcal{D}$: a non-empty set of objects (aka domain elements)
- Interpretation $\mathcal{I}$: a (non-injective) map on elements of the signature
  - constant symbols $\longmapsto$ domain elements:
    a constant symbol C is mapped into a particular object $[C]^{\mathcal{I}}$ in $\mathcal{D}$
  - predicate symbols $\longmapsto$ domain relations:
    a k-ary predicate $P(...)$ is mapped into a subset $[P]^{\mathcal{I}}$ of $\mathcal{D}^k$
    (i.e., the set of object tuples satisfying the predicate in this world)
  - functions symbols $\longmapsto$ domain functions:
    a k-ary function f is mapped into a domain function $[f]^{\mathcal{I}} : \mathcal{D}^k \longmapsto \mathcal{D}$ ($[f]^{\mathcal{I}}$ must be total)

(we denote by $[.]^{\mathcal{I}}$ the result of the interpretation $\mathcal{I}$)

An Interpretation $\mathcal{I}$ is extended to assign domain values to variables, domain values to terms and truth values to formulas.

FOL: Semantics

FOL Models (aka possible worlds)

- A model $\mathcal{M}$ is a pair $\langle \mathcal{D}, \mathcal{I} \rangle$ ($\langle$*domain*, *interpretation*$\rangle$)
- Domain $\mathcal{D}$: a non-empty set of objects (aka domain elements)
- Interpretation $\mathcal{I}$: a (non-injective) map on elements of the signature
  - constant symbols $\longmapsto$ domain elements:
    a constant symbol C is mapped into a particular object $[C]^{\mathcal{I}}$ in $\mathcal{D}$
  - predicate symbols $\longmapsto$ domain relations:
    a k-ary predicate $P(...)$ is mapped into a subset $[P]^{\mathcal{I}}$ of $\mathcal{D}^k$
    (i.e., the set of object tuples satisfying the predicate in this world)
  - functions symbols $\longmapsto$ domain functions:
    a k-ary function f is mapped into a domain function $[f]^{\mathcal{I}} : \mathcal{D}^k \longmapsto \mathcal{D}$ ($[f]^{\mathcal{I}}$ must be total)

(we denote by $[.]^{\mathcal{I}}$ the result of the interpretation $\mathcal{I}$)

An Interpretation $\mathcal{I}$ is extended to assign domain values to variables, domain values to terms and truth values to formulas.

FOL: Semantics [cont.]

Interpretation of terms

$\mathcal{I}$ maps terms into domain elements

- Variables are assigned domain values
  - variables $\longmapsto$ domain elements:
    a variable x is mapped into a particular object $[x]^{\mathcal{I}}$ in $\mathcal{D}$
- A term $f(t_1, ..., t_k)$ is mapped by $\mathcal{I}$ into the value $[f(t_1, ..., t_k)]^{\mathcal{I}}$ returned by applying the domain function $[f]^{\mathcal{I}}$, into which f is mapped, to the values $[t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}}$ obtained by applying recursively $\mathcal{I}$ to the terms $t_1, ..., t_k$:
  - $[f(t_1, ..., t_k)]^{\mathcal{I}} = [f]^{\mathcal{I}}([t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}})$
  - Ex: if "Me, Mother, Father" are interpreted as usual, then "Mother(Father(Me))" is interpreted as my (paternal) grandmother
  - Ex: if "$+, -, \cdot, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(3 - 1) \cdot (0 + 2)$" is interpreted as 4

FOL: Semantics [cont.]

Interpretation of terms

$\mathcal{I}$ maps terms into domain elements

- Variables are assigned domain values
  - variables $\longmapsto$ domain elements:
    a variable x is mapped into a particular object $[x]^{\mathcal{I}}$ in $\mathcal{D}$
- A term $f(t_1, ..., t_k)$ is mapped by $\mathcal{I}$ into the value $[f(t_1, ..., t_k)]^{\mathcal{I}}$ returned by applying the domain function $[f]^{\mathcal{I}}$, into which f is mapped, to the values $[t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}}$ obtained by applying recursively $\mathcal{I}$ to the terms $t_1, ..., t_k$:
  - $[f(t_1, ..., t_k)]^{\mathcal{I}} = [f]^{\mathcal{I}}([t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}})$
  - Ex: if "Me, Mother, Father" are interpreted as usual, then "Mother(Father(Me))" is interpreted as my (paternal) grandmother
  - Ex: if "$+, -, \cdot, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(3 - 1) \cdot (0 + 2)$" is interpreted as 4

FOL: Semantics [cont.]

Interpretation of terms

$\mathcal{I}$ maps terms into domain elements

- Variables are assigned domain values
  - variables $\longmapsto$ domain elements: a variable x is mapped into a particular object $[x]^{\mathcal{I}}$ in $\mathcal{D}$
- A term $f(t_1, ..., t_k)$ is mapped by $\mathcal{I}$ into the value $[f(t_1, ..., t_k)]^{\mathcal{I}}$ returned by applying the domain function $[f]^{\mathcal{I}}$, into which f is mapped, to the values $[t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}}$ obtained by applying recursively $\mathcal{I}$ to the terms $t_1, ..., t_k$:
  - $[f(t_1, ..., t_k)]^{\mathcal{I}} = [f]^{\mathcal{I}}([t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}})$
  - Ex: if “Me, Mother, Father” are interpreted as usual, then “Mother(Father(Me))” is interpreted as my (paternal) grandmother
  - Ex: if “$+, -, \cdot, 0, 1, 2, 3, 4$” are interpreted as usual, then “$(3 - 1) \cdot (0 + 2)$” is interpreted as 4

FOL: Semantics [cont.]

Interpretation of formulas

$\mathcal{I}$ maps formulas into truth values

- An atomic formula $P(t_1, ..., t_k)$ is true in $\mathcal{I}$ iff the objects into which the terms t_1,...t_k are mapped by $\mathcal{I}$ comply to the relation into which P is mapped
  - $[P(t_1, ..., t_k)]^{\mathcal{I}}$ is true iff $\langle [t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}} \rangle \in [P]^{\mathcal{I}}$
  - Ex: if "Me, Mother, Father, Married" are interpreted as traditon, then "Married(Mother(Me),Father(Me))" is interpreted as true
  - Ex: if "$+, -, >, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 0) > (1 + 2)$" is interpreted as true
- An atomic formula $t_1 = t_2$ is true in $\mathcal{I}$ iff the terms t_1, t_2 are mapped by $\mathcal{I}$ into the same domain element
  - $[t_1 = t_2]^{\mathcal{I}}$ is true iff $[t_1]^{\mathcal{I}}$ same as $[t_2]^{\mathcal{I}}$
  - Ex: if "Mother" is interpreted as usual, Richard, John are brothers, then "Mother(Richard)=Mother(John))" is interpreted as true
  - Ex: if "$+, -, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 1) = (1 + 2)$" is interpreted as true
- $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$ interpreted by $\mathcal{I}$ as in PL

FOL: Semantics [cont.]

Interpretation of formulas

$\mathcal{I}$ maps formulas into truth values

- An atomic formula $P(t_1, ..., t_k)$ is true in $\mathcal{I}$ iff the objects into which the terms t_1,...t_k are mapped by $\mathcal{I}$ comply to the relation into which P is mapped
  - $[P(t_1, ..., t_k)]^{\mathcal{I}}$ is true iff $\langle [t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}} \rangle \in [P]^{\mathcal{I}}$
  - Ex: if "Me, Mother, Father, Married" are interpreted as traditon, then "Married(Mother(Me),Father(Me))" is interpreted as true
  - Ex: if "$+, -, >, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 0) > (1 + 2)$" is interpreted as true
- An atomic formula $t_1 = t_2$ is true in $\mathcal{I}$ iff the terms t_1, t_2 are mapped by $\mathcal{I}$ into the same domain element
  - $[t_1 = t_2]^{\mathcal{I}}$ is true iff $[t_1]^{\mathcal{I}}$ same as $[t_2]^{\mathcal{I}}$
  - Ex: if "Mother" is interpreted as usual, Richard, John are brothers, then "Mother(Richard)=Mother(John))" is interpreted as true
  - Ex: if "$+, -, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 1) = (1 + 2)$" is interpreted as true
- $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$ interpreted by $\mathcal{I}$ as in PL

FOL: Semantics [cont.]

Interpretation of formulas

$\mathcal{I}$ maps formulas into truth values

- An atomic formula $P(t_1, ..., t_k)$ is true in $\mathcal{I}$ iff the objects into which the terms t_1,...t_k are mapped by $\mathcal{I}$ comply to the relation into which P is mapped
  - $[P(t_1, ..., t_k)]^{\mathcal{I}}$ is true iff $\langle [t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}} \rangle \in [P]^{\mathcal{I}}$
  - Ex: if "Me, Mother, Father, Married" are interpreted as traditon, then "Married(Mother(Me),Father(Me))" is interpreted as true
  - Ex: if "$+, -, >, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 0) > (1 + 2)$" is interpreted as true
- An atomic formula $t_1 = t_2$ is true in $\mathcal{I}$ iff the terms t_1, t_2 are mapped by $\mathcal{I}$ into the same domain element
  - $[t_1 = t_2]^{\mathcal{I}}$ is true iff $[t_1]^{\mathcal{I}}$ same as $[t_2]^{\mathcal{I}}$
  - Ex: if "Mother" is interpreted as usual, Richard, John are brothers, then "Mother(Richard)=Mother(John))" is interpreted as true
  - Ex: if "$+, -, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 1) = (1 + 2)$" is interpreted as true
- $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$ interpreted by $\mathcal{I}$ as in PL

FOL: Semantics [cont.]

Interpretation of formulas

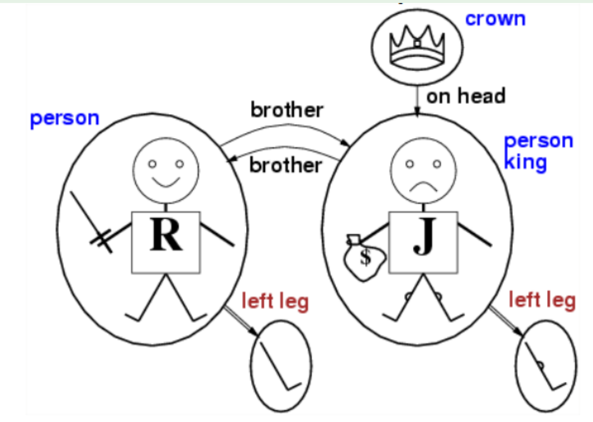
$\mathcal{I}$ maps formulas into truth values

- An atomic formula $P(t_1, ..., t_k)$ is true in $\mathcal{I}$ iff the objects into which the terms t_1,...t_k are mapped by $\mathcal{I}$ comply to the relation into which P is mapped
  - $[P(t_1, ..., t_k)]^{\mathcal{I}}$ is true iff $\langle [t_1]^{\mathcal{I}}, ..., [t_k]^{\mathcal{I}} \rangle \in [P]^{\mathcal{I}}$
  - Ex: if "Me, Mother, Father, Married" are interpreted as traditon, then "Married(Mother(Me),Father(Me))" is interpreted as true
  - Ex: if "$+, -, >, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 0) > (1 + 2)$" is interpreted as true
- An atomic formula $t_1 = t_2$ is true in $\mathcal{I}$ iff the terms t_1, t_2 are mapped by $\mathcal{I}$ into the same domain element
  - $[t_1 = t_2]^{\mathcal{I}}$ is true iff $[t_1]^{\mathcal{I}}$ same as $[t_2]^{\mathcal{I}}$
  - Ex: if "Mother" is interpreted as usual, Richard, John are brothers, then "Mother(Richard)=Mother(John))" is interpreted as true
  - Ex: if "$+, -, 0, 1, 2, 3, 4$" are interpreted as usual, then "$(4 - 1) = (1 + 2)$" is interpreted as true
- $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$ interpreted by $\mathcal{I}$ as in PL

Models for FOL: Example

Richard Lionhearth and John Lackland

- $\mathcal{D}$: domain at right
- $\mathcal{I}$: s.t.
  - $[Richard]^{\mathcal{I}}$: Richard the Lionhearth
  - $[John]^{\mathcal{I}}$: evil King John
  - $[Brother]^{\mathcal{I}}$: brotherhood
- $[Brother(Richard, John)]^{\mathcal{I}}$ is true
- $[LeftLeg]^{\mathcal{I}}$ maps any individual to his left leg
- ...

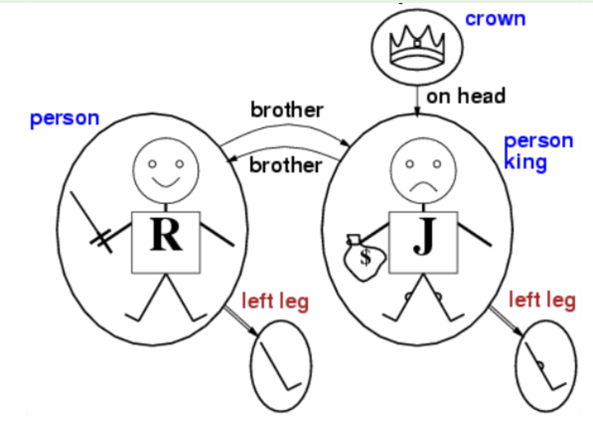


(© S. Russell & P. Norwig, Artificial Intelligence: A Modern Approach, III Ed., Pearson)

Models for FOL: Example

Richard Lionhearth and John Lackland

- $\mathcal{D}$: domain at right
- $\mathcal{I}$: s.t.
  - $[Richard]^{\mathcal{I}}$: Richard the Lionhearth
  - $[John]^{\mathcal{I}}$: evil King John
  - $[Brother]^{\mathcal{I}}$: brotherhood
- $[Brother(Richard, John)]^{\mathcal{I}}$ is true
- $[LeftLeg]^{\mathcal{I}}$ maps any individual to his left leg
- ...

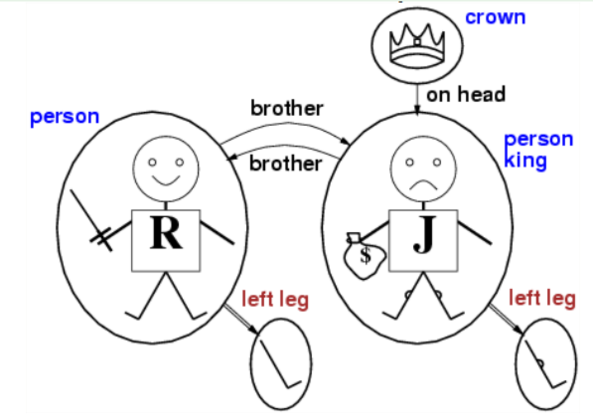


(© S. Russell & P. Norwig, Artificial Intelligence: A Modern Approach, III Ed., Pearson)

Models for FOL: Example

Richard Lionhearth and John Lackland

- $\mathcal{D}$: domain at right
- $\mathcal{I}$: s.t.
  - $[Richard]^{\mathcal{I}}$: Richard the Lionhearth
  - $[John]^{\mathcal{I}}$: evil King John
  - $[Brother]^{\mathcal{I}}$: brotherhood
- $[Brother(Richard, John)]^{\mathcal{I}}$ is true
- $[LeftLeg]^{\mathcal{I}}$ maps any individual to his left leg
- ...

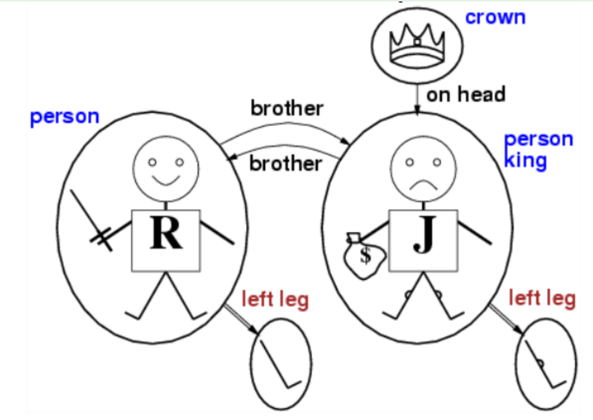


(© S. Russell & P. Norwig, Artificial Intelligence: A Modern Approach, III Ed., Pearson)

Models for FOL: Example

Richard Lionhearth and John Lackland

- $\mathcal{D}$: domain at right
- $\mathcal{I}$: s.t.
  - $[Richard]^{\mathcal{I}}$: Richard the Lionhearth
  - $[John]^{\mathcal{I}}$: evil King John
  - $[Brother]^{\mathcal{I}}$: brotherhood
- $[Brother(Richard, John)]^{\mathcal{I}}$ is true
- $[LeftLeg]^{\mathcal{I}}$ maps any individual to his left leg
- ...



(© S. Russell & P. Norwig, Artificial Intelligence: A Modern Approach, III Ed., Pearson)

Equality

- Equality is a special predicate: $t_1 = t_2$ is true under a given interpretation if and only if t_1 and t_2 refer to the same object
  - Ex: $1 = 2$ and $x * x = x$ are satisfiable (!)
  - Ex: $2 = 2$ is valid
- Ex: definition of *Sibling* in terms of *Parent*
  $\forall x, y.\ (Siblings(x, y) \leftrightarrow [\neg(x = y) \land \exists p_1, p_2.\ (\neg(p_1 = p_2) \land Parent(p_1, x) \land Parent(p_2, x) \land Parent(p_1, y) \land Parent(p_2, y)]))$

Equality

- Equality is a special predicate: $t_1 = t_2$ is true under a given interpretation if and only if t_1 and t_2 refer to the same object
  - Ex: $1 = 2$ and $x * x = x$ are satisfiable (!)
  - Ex: $2 = 2$ is valid
- Ex: definition of *Sibling* in terms of *Parent*
  $\forall x, y.\ (Siblings(x, y) \leftrightarrow [\neg(x = y) \wedge \exists p_1, p_2.\ (\neg(p_1 = p_2) \wedge Parent(p_1, x) \wedge Parent(p_2, x) \wedge Parent(p_1, y) \wedge Parent(p_2, y)]))$

Outline

1. Introduction
    - Basics on First-order Logic
    - What is a Theory?
    - Satisfiability Modulo Theories
    - Motivations and Goals of SMT
2. Efficient SMT solving
    - Combining SAT with Theory Solvers
    - Theory Solvers for Theories of Interest (hints)
    - SMT for Combinations of Theories
3. Beyond Solving: Advanced SMT Functionalities
    - Proofs and Unsatisfiable Cores
    - All-SMT & Predicate Abstraction (hints)
    - SMT with Optimization (Optimization Modulo Theories)

FOL Theories

Traditional Definition (FOL)

Given a FOL signature Σ, a Σ-Theory $\mathcal{T}$ (hereafter simply “theory”) is a (possibly infinite) set of FOL closed formulas (axioms)

- Typically used to provide some intended interpretation to the symbols in the signature Σ
- FOL formulas deduces from these axioms via inference rules
- Definition used by logicians,
- Very low practical use in AR & Formal Verification

FOL Theories

Traditional Definition (FOL)

Given a FOL signature Σ, a Σ-Theory $\mathcal{T}$ (hereafter simply “theory”) is a (possibly infinite) set of FOL closed formulas (axioms)

- Typically used to provide some intended interpretation to the symbols in the signature Σ
- FOL formulas deduces from these axioms via inference rules
- Definition used by logicians,
- Very low practical use in AR & Formal Verification

FOL Theories

Traditional Definition (FOL)

Given a FOL signature Σ, a Σ-Theory $\mathcal{T}$ (hereafter simply "theory") is a (possibly infinite) set of FOL closed formulas (axioms)

- Typically used to provide some intended interpretation to the symbols in the signature Σ
- FOL formulas deduces from these axioms via inference rules
- Definition used by logicians,
- Very low practical use in AR & Formal Verification

FOL Theories

Traditional Definition (FOL)

Given a FOL signature Σ, a Σ-Theory $\mathcal{T}$ (hereafter simply “theory”) is a (possibly infinite) set of FOL closed formulas (axioms)

- Typically used to provide some intended interpretation to the symbols in the signature Σ
- FOL formulas deduces from these axioms via inference rules
- Definition used by logicians,
- Very low practical use in AR & Formal Verification

FOL Theories

Traditional Definition (FOL)

Given a FOL signature Σ, a Σ-Theory $\mathcal{T}$ (hereafter simply "theory") is a (possibly infinite) set of FOL closed formulas (axioms)

- Typically used to provide some intended interpretation to the symbols in the signature Σ
- FOL formulas deduces from these axioms via inference rules
- Definition used by logicians,
- Very low practical use in AR & Formal Verification

Example: A FOL Theory of Positive Integer Numbers (aka “Peano Arithmetic”, $\mathcal{P}$)

- Signature
  - (basic) unary predicate symbol: NatNum (“natural number”)
  - (basic) unary function symbol: S (“successor”)
  - (basic) constant symbol: 0
  - (derived) binary function symbols: +,* (infix)
  - (derived) constant symbols: 1,2,3,4,5,6,...
- Axioms
  1. $NatNum(0)$
  2. $\forall x.(NatNum(x) \rightarrow NatNum(S(x)))$
  3. $\forall x.(NatNum(x) \rightarrow (0 \neq S(x)))$
  4. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow ((x \neq y) \rightarrow (S(x) \neq S(y))))$
  5. $\forall x.(NatNum(x) \rightarrow (x = (0 + x)))$
  6. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow (S(x) + y) = S(x + y))$
  7. $1 = S(0), 2 = S(1), 3 = S(2), ...$
- Formulas deduced
  - ex: $\mathcal{P} \vdash NatNum(25)$
  - ex: $\mathcal{P} \vdash \forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow ((x + y) = (y + x)))$

Example: A FOL Theory of Positive Integer Numbers (aka “Peano Arithmetic”, $\mathcal{P}$)

- Signature
  - (basic) unary predicate symbol: NatNum (“natural number”)
  - (basic) unary function symbol: S (“successor”)
  - (basic) constant symbol: 0
  - (derived) binary function symbols: +,* (infix)
  - (derived) constant symbols: 1,2,3,4,5,6,...
- Axioms
  1. $NatNum(0)$
  2. $\forall x.(NatNum(x) \rightarrow NatNum(S(x)))$
  3. $\forall x.(NatNum(x) \rightarrow (0 \neq S(x)))$
  4. $\forall x, y.((NatNum(x) \land NatNum(y)) \rightarrow ((x \neq y) \rightarrow (S(x) \neq S(y))))$
  5. $\forall x.(NatNum(x) \rightarrow (x = (0 + x)))$
  6. $\forall x, y.((NatNum(x) \land NatNum(y)) \rightarrow (S(x) + y) = S(x + y))$
  7. $1 = S(0), 2 = S(1), 3 = S(2), \ldots$
- Formulas deduced
  - ex: $\mathcal{P} \vdash NatNum(25)$
  - ex: $\mathcal{P} \vdash \forall x, y.((NatNum(x) \land NatNum(y)) \rightarrow ((x + y) = (y + x)))$

Example: A FOL Theory of Positive Integer Numbers (aka “Peano Arithmetic”, $\mathcal{P}$)

- Signature
  - (basic) unary predicate symbol: NatNum (“natural number”)
  - (basic) unary function symbol: S (“successor”)
  - (basic) constant symbol: 0
  - (derived) binary function symbols: +,* (infix)
  - (derived) constant symbols: 1,2,3,4,5,6,...
- Axioms
  1. $NatNum(0)$
  2. $\forall x.(NatNum(x) \rightarrow NatNum(S(x)))$
  3. $\forall x.(NatNum(x) \rightarrow (0 \neq S(x)))$
  4. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow ((x \neq y) \rightarrow (S(x) \neq S(y))))$
  5. $\forall x.(NatNum(x) \rightarrow (x = (0 + x)))$
  6. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow (S(x) + y) = S(x + y))$
  7. $1 = S(0)$, $2 = S(1)$, $3 = S(2)$, ...
- Formulas deduced
  - ex: $\mathcal{P} \vdash NatNum(25)$
  - ex: $\mathcal{P} \vdash \forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow ((x + y) = (y + x)))$

Example: A FOL Theory of Positive Integer Numbers (aka "Peano Arithmetic", $\mathcal{P}$)

- Signature
  - (basic) unary predicate symbol: NatNum ("natural number")
  - (basic) unary function symbol: S ("successor")
  - (basic) constant symbol: 0
  - (derived) binary function symbols: +,* (infix)
  - (derived) constant symbols: 1,2,3,4,5,6,...
- Axioms
  1. $NatNum(0)$
  2. $\forall x.(NatNum(x) \rightarrow NatNum(S(x)))$
  3. $\forall x.(NatNum(x) \rightarrow (0 \neq S(x)))$
  4. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow ((x \neq y) \rightarrow (S(x) \neq S(y))))$
  5. $\forall x.(NatNum(x) \rightarrow (x = (0 + x)))$
  6. $\forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow (S(x) + y) = S(x + y))$
  7. $1 = S(0), 2 = S(1), 3 = S(2), ...$
- Formulas deduced
  - ex: $\mathcal{P} \vdash NatNum(25)$
  - ex: $\mathcal{P} \vdash \forall x, y.((NatNum(x) \wedge NatNum(y)) \rightarrow ((x + y) = (y + x)))$

FOL Theories (cont.)

SMT Definition

Given a FOL signature Σ, a Σ-Theory $\mathcal{T}$ (hereafter simply "theory") is one (or more) model(s) constraining the interpretations of Σ

- Provides an intended interpretation to the symbols in Σ
  - constants mapped into domain elements
    - ex: "1" mapped into the number one
  - predicate symbols mapped into relations on domain elements
    - ex: "$. < .$" mapped into the arithmetical relation "less then"
  - function symbols mapped into functions on domain elements
    - ex: "$S(.)$" mapped into the arithmetical function "successor of"

  These symbols are called interpreted
- Compliant with previous definition: model(s) satisfying all axioms
- Ad hoc "$\mathcal{T}$-aware" decision procedures for reasoning on formulas
- Very effective in practical applications

FOL Theories (cont.)

SMT Definition

Given a FOL signature Σ, a Σ-Theory $\mathcal{T}$ (hereafter simply “theory”) is one (or more) model(s) constraining the interpretations of Σ

- Provides an intended interpretation to the symbols in Σ
  - constants mapped into domain elements
    - ex: “1” mapped into the number one
  - predicate symbols mapped into relations on domain elements
    - ex: “$. < .$” mapped into the arithmetical relation “less then”
  - function symbols mapped into functions on domain elements
    - ex: “$S(.)$” mapped into the arithmetical function “successor of”

  These symbols are called interpreted
- Compliant with previous definition: model(s) satisfying all axioms
- Ad hoc “$\mathcal{T}$-aware” decision procedures for reasoning on formulas
- Very effective in practical applications

FOL Theories (cont.)

SMT Definition

Given a FOL signature Σ, a Σ-Theory $\mathcal{T}$ (hereafter simply “theory”) is one (or more) model(s) constraining the interpretations of Σ

- Provides an intended interpretation to the symbols in Σ
  - constants mapped into domain elements
    - ex: “1” mapped into the number one
  - predicate symbols mapped into relations on domain elements
    - ex: “$. < .$” mapped into the arithmetical relation “less then”
  - function symbols mapped into functions on domain elements
    - ex: “$S(.)$” mapped into the arithmetical function “successor of”

  These symbols are called interpreted
- Compliant with previous definition: model(s) satisfying all axioms
- Ad hoc “$\mathcal{T}$-aware” decision procedures for reasoning on formulas
- Very effective in practical applications

FOL Theories (cont.)

SMT Definition

Given a FOL signature Σ, a Σ-Theory $\mathcal{T}$ (hereafter simply “theory”) is one (or more) model(s) constraining the interpretations of Σ

- Provides an intended interpretation to the symbols in Σ
  - constants mapped into domain elements
    - ex: “1” mapped into the number one
  - predicate symbols mapped into relations on domain elements
    - ex: “$. < .$” mapped into the arithmetical relation “less then”
  - function symbols mapped into functions on domain elements
    - ex: “$S(.)$” mapped into the arithmetical function “successor of”

  These symbols are called interpreted
- Compliant with previous definition: model(s) satisfying all axioms
- Ad hoc “$\mathcal{T}$-aware” decision procedures for reasoning on formulas
- Very effective in practical applications

FOL Theories (cont.)

SMT Definition

Given a FOL signature Σ, a Σ-Theory $\mathcal{T}$ (hereafter simply "theory") is one (or more) model(s) constraining the interpretations of Σ

- Provides an intended interpretation to the symbols in Σ
  - constants mapped into domain elements
    - ex: "1" mapped into the number one
  - predicate symbols mapped into relations on domain elements
    - ex: "$. < .$" mapped into the arithmetical relation "less then"
  - function symbols mapped into functions on domain elements
    - ex: "$S(.)$" mapped into the arithmetical function "successor of"

  These symbols are called interpreted
- Compliant with previous definition: model(s) satisfying all axioms
- Ad hoc "$\mathcal{T}$-aware" decision procedures for reasoning on formulas
- Very effective in practical applications

Example: Linear Arithmetic on the Integers ($\mathcal{LIA}$)

- Domain: integer numbers
- Numerical constants interpreted as numbers
  - ex: "1", "1346231" mapped directly into the corresponding number
- function and predicates interpreted as arithmetical operations
  - "+" as addiction, "*" as multiplication, "$<$" as less-then, . etc.
- ILP solvers used to do logical reasoning
  - ex: $(3x - 2y \leq 3) \wedge (4y - 2z < -7) \models (6x - 2z < -1)$

Example: Linear Arithmetic on the Integers ($\mathcal{LIA}$)

- Domain: integer numbers
- Numerical constants interpreted as numbers
  - ex: “1”, “1346231” mapped directly into the corresponding number
- function and predicates interpreted as arithmetical operations
  - “$+$” as addiction, “$*$” as multiplication, “$<$” as less-then, . etc.
- ILP solvers used to do logical reasoning
  - ex: $(3x - 2y \leq 3) \wedge (4y - 2z < -7) \models (6x - 2z < -1)$

Example: Linear Arithmetic on the Integers ($\mathcal{LIA}$)

- Domain: integer numbers
- Numerical constants interpreted as numbers
  - ex: “1”, “1346231” mapped directly into the corresponding number
- function and predicates interpreted as arithmetical operations
  - “+” as addiction, “*” as multiplication, “<” as less-then, . etc.
- ILP solvers used to do logical reasoning
  - ex: $(3x - 2y \leq 3) \wedge (4y - 2z < -7) \models (6x - 2z < -1)$

Example: Linear Arithmetic on the Integers ($\mathcal{LIA}$)

- Domain: integer numbers
- Numerical constants interpreted as numbers
  - ex: “1”, “1346231” mapped directly into the corresponding number
- function and predicates interpreted as arithmetical operations
  - “+” as addiction, “*” as multiplication, “$<$” as less-then, . etc.
- ILP solvers used to do logical reasoning
  - ex: $(3x - 2y \leq 3) \wedge (4y - 2z < -7) \models (6x - 2z < -1)$

Satisfiability, Validity, Entailment (Modulo a Theory $\mathcal{T}$)

Definitions

- Idea: We restrict to models satisfying $\mathcal{T}$ ("$\mathcal{T}$-models")
- A formula is satisfiable in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-satisfiable") iff some $\mathcal{T}$-model satisfies also φ
  - ex: $(x < 3)$ satisfiable in $\mathcal{LIA}$
  - ex: $(1 = 2)$ not satisfiable in $\mathcal{LIA}$!
- A formula φ is valid in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-valid" or "$\models_{\mathcal{T}} \varphi$") iff all $\mathcal{T}$-models satisfy also φ
  - ex: $(x < 3) \to (x < 4)$ valid in $\mathcal{LIA}$
- A formula φ entails ψ in $\mathcal{T}$ (aka "φ $\mathcal{T}$-entails ψ" or "$\varphi \models_{\mathcal{T}} \psi$") iff all $\mathcal{T}$-models satisfying φ satisfy also ψ
  - ex: $(x < 3) \models_{\mathcal{LIA}} (x < 4)$

Properties

- φ is $\mathcal{T}$-valid iff $\neg\varphi$ is $\mathcal{T}$-unsatisfiable
- $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \to \psi$ is $\mathcal{T}$-valid

$\Longrightarrow$ $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \wedge \neg\psi$ $\mathcal{T}$-unsatisfiable

Satisfiability, Validity, Entailment (Modulo a Theory $\mathcal{T}$)

Definitions

- Idea: We restrict to models satisfying $\mathcal{T}$ ("$\mathcal{T}$-models")
- A formula is satisfiable in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-satisfiable") iff some $\mathcal{T}$-model satisfies also φ
  - ex: $(x < 3)$ satisfiable in $\mathcal{LIA}$
  - ex: $(1 = 2)$ not satisfiable in $\mathcal{LIA}$!
- A formula φ is valid in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-valid" or "$\models_{\mathcal{T}} \varphi$") iff all $\mathcal{T}$-models satisfy also φ
  - ex: $(x < 3) \rightarrow (x < 4)$ valid in $\mathcal{LIA}$
- A formula φ entails ψ in $\mathcal{T}$ (aka "φ $\mathcal{T}$-entails ψ" or "$\varphi \models_{\mathcal{T}} \psi$") iff all $\mathcal{T}$-models satisfying φ satisfy also ψ
  - ex: $(x < 3) \models_{\mathcal{LIA}} (x < 4)$

Properties

- φ is $\mathcal{T}$-valid iff $\neg\varphi$ is $\mathcal{T}$-unsatisfiable
- $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \rightarrow \psi$ is $\mathcal{T}$-valid

$\Longrightarrow$ $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \wedge \neg\psi$ $\mathcal{T}$-unsatisfiable

Satisfiability, Validity, Entailment (Modulo a Theory $\mathcal{T}$)

Definitions

- Idea: We restrict to models satisfying $\mathcal{T}$ ("$\mathcal{T}$-models")
- A formula is satisfiable in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-satisfiable") iff some $\mathcal{T}$-model satisfies also φ
  - ex: $(x < 3)$ satisfiable in $\mathcal{LIA}$
  - ex: $(1 = 2)$ not satisfiable in $\mathcal{LIA}$!
- A formula φ is valid in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-valid" or "$\models_{\mathcal{T}} \varphi$") iff all $\mathcal{T}$-models satisfy also φ
  - ex: $(x < 3) \rightarrow (x < 4)$ valid in $\mathcal{LIA}$
- A formula φ entails ψ in $\mathcal{T}$ (aka "φ $\mathcal{T}$-entails ψ" or "$\varphi \models_{\mathcal{T}} \psi$") iff all $\mathcal{T}$-models satisfying φ satisfy also ψ
  - ex: $(x < 3) \models_{\mathcal{LIA}} (x < 4)$

Properties

- φ is $\mathcal{T}$-valid iff $\neg\varphi$ is $\mathcal{T}$-unsatisfiable
- $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \rightarrow \psi$ is $\mathcal{T}$-valid

$\Longrightarrow$ $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \wedge \neg\psi$ $\mathcal{T}$-unsatisfiable

Satisfiability, Validity, Entailment (Modulo a Theory $\mathcal{T}$)

Definitions

- Idea: We restrict to models satisfying $\mathcal{T}$ (“$\mathcal{T}$-models”)
- A formula is satisfiable in $\mathcal{T}$ (aka “φ is $\mathcal{T}$-satisfiable”) iff some $\mathcal{T}$-model satisfies also φ
  - ex: $(x < 3)$ satisfiable in $\mathcal{LIA}$
  - ex: $(1 = 2)$ not satisfiable in $\mathcal{LIA}$!
- A formula φ is valid in $\mathcal{T}$ (aka “φ is $\mathcal{T}$-valid” or “$\models_{\mathcal{T}} \varphi$”) iff all $\mathcal{T}$-models satisfy also φ
  - ex: $(x < 3) \rightarrow (x < 4)$ valid in $\mathcal{LIA}$
- A formula φ entails ψ in $\mathcal{T}$ (aka “φ $\mathcal{T}$-entails ψ” or “$\varphi \models_{\mathcal{T}} \psi$”) iff all $\mathcal{T}$-models satisfying φ satisfy also ψ
  - ex: $(x < 3) \models_{\mathcal{LIA}} (x < 4)$

Properties

- φ is $\mathcal{T}$-valid iff $\neg\varphi$ is $\mathcal{T}$-unsatisfiable
- $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \rightarrow \psi$ is $\mathcal{T}$-valid

$\Longrightarrow$ $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \wedge \neg\psi$ $\mathcal{T}$-unsatisfiable

Satisfiability, Validity, Entailment (Modulo a Theory $\mathcal{T}$)

Definitions

- Idea: We restrict to models satisfying $\mathcal{T}$ ("$\mathcal{T}$-models")
- A formula is satisfiable in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-satisfiable") iff some $\mathcal{T}$-model satisfies also φ
  - ex: $(x < 3)$ satisfiable in $\mathcal{LIA}$
  - ex: $(1 = 2)$ not satisfiable in $\mathcal{LIA}$!
- A formula φ is valid in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-valid" or "$\models_{\mathcal{T}} \varphi$") iff all $\mathcal{T}$-models satisfy also φ
  - ex: $(x < 3) \rightarrow (x < 4)$ valid in $\mathcal{LIA}$
- A formula φ entails ψ in $\mathcal{T}$ (aka "φ $\mathcal{T}$-entails ψ" or "$\varphi \models_{\mathcal{T}} \psi$") iff all $\mathcal{T}$-models satisfying φ satisfy also ψ
  - ex: $(x < 3) \models_{\mathcal{LIA}} (x < 4)$

Properties

- φ is $\mathcal{T}$-valid iff $\neg\varphi$ is $\mathcal{T}$-unsatisfiable
- $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \rightarrow \psi$ is $\mathcal{T}$-valid

$\Longrightarrow$ $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \wedge \neg\psi$ $\mathcal{T}$-unsatisfiable

Satisfiability, Validity, Entailment (Modulo a Theory $\mathcal{T}$)

Definitions

- Idea: We restrict to models satisfying $\mathcal{T}$ ("$\mathcal{T}$-models")
- A formula is satisfiable in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-satisfiable") iff some $\mathcal{T}$-model satisfies also φ
  - ex: $(x < 3)$ satisfiable in $\mathcal{LIA}$
  - ex: $(1 = 2)$ not satisfiable in $\mathcal{LIA}$!
- A formula φ is valid in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-valid" or "$\models_{\mathcal{T}} \varphi$") iff all $\mathcal{T}$-models satisfy also φ
  - ex: $(x < 3) \rightarrow (x < 4)$ valid in $\mathcal{LIA}$
- A formula φ entails ψ in $\mathcal{T}$ (aka "φ $\mathcal{T}$-entails ψ" or "$\varphi \models_{\mathcal{T}} \psi$") iff all $\mathcal{T}$-models satisfying φ satisfy also ψ
  - ex: $(x < 3) \models_{\mathcal{LIA}} (x < 4)$

Properties

- φ is $\mathcal{T}$-valid iff $\neg\varphi$ is $\mathcal{T}$-unsatisfiable
- $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \rightarrow \psi$ is $\mathcal{T}$-valid

$\Longrightarrow$ $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \wedge \neg\psi$ $\mathcal{T}$-unsatisfiable

Satisfiability, Validity, Entailment (Modulo a Theory $\mathcal{T}$)

Definitions

- Idea: We restrict to models satisfying $\mathcal{T}$ ("$\mathcal{T}$-models")
- A formula is satisfiable in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-satisfiable") iff some $\mathcal{T}$-model satisfies also φ
  - ex: $(x < 3)$ satisfiable in $\mathcal{LIA}$
  - ex: $(1 = 2)$ not satisfiable in $\mathcal{LIA}$!
- A formula φ is valid in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-valid" or "$\models_{\mathcal{T}} \varphi$") iff all $\mathcal{T}$-models satisfy also φ
  - ex: $(x < 3) \rightarrow (x < 4)$ valid in $\mathcal{LIA}$
- A formula φ entails ψ in $\mathcal{T}$ (aka "φ $\mathcal{T}$-entails ψ" or "$\varphi \models_{\mathcal{T}} \psi$") iff all $\mathcal{T}$-models satisfying φ satisfy also ψ
  - ex: $(x < 3) \models_{\mathcal{LIA}} (x < 4)$

Properties

- φ is $\mathcal{T}$-valid iff $\neg\varphi$ is $\mathcal{T}$-unsatisfiable
- $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \rightarrow \psi$ is $\mathcal{T}$-valid

$\Longrightarrow$ $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \wedge \neg\psi$ $\mathcal{T}$-unsatisfiable

Satisfiability, Validity, Entailment (Modulo a Theory $\mathcal{T}$)

Definitions

- Idea: We restrict to models satisfying $\mathcal{T}$ ("$\mathcal{T}$-models")
- A formula is satisfiable in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-satisfiable") iff some $\mathcal{T}$-model satisfies also φ
  - ex: $(x < 3)$ satisfiable in $\mathcal{LIA}$
  - ex: $(1 = 2)$ not satisfiable in $\mathcal{LIA}$!
- A formula φ is valid in $\mathcal{T}$ (aka "φ is $\mathcal{T}$-valid" or "$\models_{\mathcal{T}} \varphi$") iff all $\mathcal{T}$-models satisfy also φ
  - ex: $(x < 3) \rightarrow (x < 4)$ valid in $\mathcal{LIA}$
- A formula φ entails ψ in $\mathcal{T}$ (aka "φ $\mathcal{T}$-entails ψ" or "$\varphi \models_{\mathcal{T}} \psi$") iff all $\mathcal{T}$-models satisfying φ satisfy also ψ
  - ex: $(x < 3) \models_{\mathcal{LIA}} (x < 4)$

Properties

- φ is $\mathcal{T}$-valid iff $\neg\varphi$ is $\mathcal{T}$-unsatisfiable
- $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \rightarrow \psi$ is $\mathcal{T}$-valid

$\Longrightarrow$ $\varphi \models_{\mathcal{T}} \psi$ iff $\varphi \wedge \neg\psi$ $\mathcal{T}$-unsatisfiable

Outline

1. Introduction
   - Basics on First-order Logic
   - What is a Theory?
   - Satisfiability Modulo Theories
   - Motivations and Goals of SMT

2. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for Theories of Interest (hints)
   - SMT for Combinations of Theories

3. Beyond Solving: Advanced SMT Functionalities
   - Proofs and Unsatisfiable Cores
   - All-SMT & Predicate Abstraction (hints)
   - SMT with Optimization (Optimization Modulo Theories)

Satisfiability Modulo Theories (SMT($\mathcal{T}$))

Satisfiability Modulo Theories (SMT($\mathcal{T}$))

The problem of deciding the satisfiability of (typically quantifier-free) formulas in some decidable first-order theory $\mathcal{T}$

- $\mathcal{T}$ can also be a combination of theories $\bigcup_i \mathcal{T}_i$.

SMT($\mathcal{T}$): Theories of Interest

Some theories of interest (e.g., for formal verification)

- Equality and Uninterpreted Functions ($\mathcal{EUF}$): $((x = y) \wedge (y = f(z))) \rightarrow (g(x) = g(f(z)))$
- Difference logic ($\mathcal{DL}$): $((x = y) \wedge (y - z \leq 4)) \rightarrow (x - z \leq 6)$
- UTVPI ($\mathcal{UTVPI}$): $((x = y) \wedge (y - z \leq 4)) \rightarrow (x + z \leq 6)$
- Linear arithmetic over the rationals ($\mathcal{LRA}$): $(T_\delta \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \wedge (\neg T_\delta \rightarrow (s_1 = s_0))$
- Linear arithmetic over the integers ($\mathcal{LIA}$): $(x = x_l + 2^{16} x_h) \wedge (x \geq 0) \wedge (x \leq 2^{16} - 1)$
- Arrays ($\mathcal{AR}$): $(i = j) \vee read(write(a, i, e), j) = read(a, j)$
- Bit vectors ($\mathcal{BV}$): $x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[8]}[3:0]$
- Non-Linear arithmetic over the reals ($\mathcal{NLA}(\mathbb{R})$) : $((c = a \cdot b) \wedge (a_1 = a - 1) \wedge (b_1 = b + 1)) \rightarrow (c = a_1 \cdot b_1 + 1)$
- ...

SMT($\mathcal{T}$): Theories of Interest

Some theories of interest (e.g., for formal verification)

- Equality and Uninterpreted Functions ($\mathcal{EUF}$):
  $((x = y) \land (y = f(z))) \rightarrow (g(x) = g(f(z)))$
- Difference logic ($\mathcal{DL}$): $((x = y) \land (y - z \leq 4)) \rightarrow (x - z \leq 6)$
- UTVPI ($\mathcal{UTVPI}$): $((x = y) \land (y - z \leq 4)) \rightarrow (x + z \leq 6)$
- Linear arithmetic over the rationals ($\mathcal{LRA}$):
  $(T_\delta \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \land (\neg T_\delta \rightarrow (s_1 = s_0))$
- Linear arithmetic over the integers ($\mathcal{LIA}$): $(x = x_l + 2^{16} x_h) \land (x \geq 0) \land (x \leq 2^{16} - 1)$
- Arrays ($\mathcal{AR}$): $(i = j) \lor read(write(a, i, e), j) = read(a, j)$
- Bit vectors ($\mathcal{BV}$): $x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[8]}[3:0]$
- Non-Linear arithmetic over the reals ($\mathcal{NLA}(\mathbb{R})$) :
  $((c = a \cdot b) \land (a_1 = a - 1) \land (b_1 = b + 1)) \rightarrow (c = a_1 \cdot b_1 + 1)$
- ...

SMT($\mathcal{T}$): Theories of Interest

Some theories of interest (e.g., for formal verification)

- Equality and Uninterpreted Functions ($\mathcal{EUF}$):
  $((x = y) \land (y = f(z))) \to (g(x) = g(f(z)))$
- Difference logic ($\mathcal{DL}$): $((x = y) \land (y - z \leq 4)) \to (x - z \leq 6)$
- UTVPI ($\mathcal{UTVPI}$): $((x = y) \land (y - z \leq 4)) \to (x + z \leq 6)$
- Linear arithmetic over the rationals ($\mathcal{LRA}$):
  $(T_\delta \to (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \land (\neg T_\delta \to (s_1 = s_0))$
- Linear arithmetic over the integers ($\mathcal{LIA}$): $(x = x_l + 2^{16} x_h) \land (x \geq 0) \land (x \leq 2^{16} - 1)$
- Arrays ($\mathcal{AR}$): $(i = j) \lor read(write(a, i, e), j) = read(a, j)$
- Bit vectors ($\mathcal{BV}$): $x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[8]}[3:0]$
- Non-Linear arithmetic over the reals ($\mathcal{NLA}(\mathbb{R})$) :
  $((c = a \cdot b) \land (a_1 = a - 1) \land (b_1 = b + 1)) \to (c = a_1 \cdot b_1 + 1)$
- ...

SMT($\mathcal{T}$): Theories of Interest

Some theories of interest (e.g., for formal verification)

- Equality and Uninterpreted Functions ($\mathcal{EUF}$):
  $((x = y) \wedge (y = f(z))) \rightarrow (g(x) = g(f(z)))$
- Difference logic ($\mathcal{DL}$): $((x = y) \wedge (y - z \leq 4)) \rightarrow (x - z \leq 6)$
- UTVPI ($\mathcal{UTVPI}$): $((x = y) \wedge (y - z \leq 4)) \rightarrow (x + z \leq 6)$
- Linear arithmetic over the rationals ($\mathcal{LRA}$):
  $(T_\delta \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \wedge (\neg T_\delta \rightarrow (s_1 = s_0))$
- Linear arithmetic over the integers ($\mathcal{LIA}$): $(x = x_l + 2^{16} x_h) \wedge (x \geq 0) \wedge (x \leq 2^{16} - 1)$
- Arrays ($\mathcal{AR}$): $(i = j) \vee read(write(a, i, e), j) = read(a, j)$
- Bit vectors ($\mathcal{BV}$): $x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[8]}[3:0]$
- Non-Linear arithmetic over the reals ($\mathcal{NLA}(\mathbb{R})$) :
  $((c = a \cdot b) \wedge (a_1 = a - 1) \wedge (b_1 = b + 1)) \rightarrow (c = a_1 \cdot b_1 + 1)$
- ...

SMT($\mathcal{T}$): Theories of Interest

Some theories of interest (e.g., for formal verification)

- Equality and Uninterpreted Functions ($\mathcal{EUF}$):
  $((x = y) \wedge (y = f(z))) \rightarrow (g(x) = g(f(z)))$
- Difference logic ($\mathcal{DL}$): $((x = y) \wedge (y - z \leq 4)) \rightarrow (x - z \leq 6)$
- UTVPI ($\mathcal{UTVPI}$): $((x = y) \wedge (y - z \leq 4)) \rightarrow (x + z \leq 6)$
- Linear arithmetic over the rationals ($\mathcal{LRA}$):
  $(T_{\delta} \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \wedge (\neg T_{\delta} \rightarrow (s_1 = s_0))$
- Linear arithmetic over the integers ($\mathcal{LIA}$): $(x = x_l + 2^{16} x_h) \wedge (x \geq 0) \wedge (x \leq 2^{16} - 1)$
- Arrays ($\mathcal{AR}$): $(i = j) \vee read(write(a, i, e), j) = read(a, j)$
- Bit vectors ($\mathcal{BV}$): $x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[8]}[3:0]$
- Non-Linear arithmetic over the reals ($\mathcal{NLA}(\mathbb{R})$) :
  $((c = a \cdot b) \wedge (a_1 = a - 1) \wedge (b_1 = b + 1)) \rightarrow (c = a_1 \cdot b_1 + 1)$
- ...

SMT($\mathcal{T}$): Theories of Interest

Some theories of interest (e.g., for formal verification)

- Equality and Uninterpreted Functions ($\mathcal{EUF}$):
  $((x = y) \wedge (y = f(z))) \rightarrow (g(x) = g(f(z)))$
- Difference logic ($\mathcal{DL}$): $((x = y) \wedge (y - z \leq 4)) \rightarrow (x - z \leq 6)$
- UTVPI ($\mathcal{UTVPI}$): $((x = y) \wedge (y - z \leq 4)) \rightarrow (x + z \leq 6)$
- Linear arithmetic over the rationals ($\mathcal{LRA}$):
  $(T_\delta \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \wedge (\neg T_\delta \rightarrow (s_1 = s_0))$
- Linear arithmetic over the integers ($\mathcal{LIA}$): $(x = x_l + 2^{16} x_h) \wedge (x \geq 0) \wedge (x \leq 2^{16} - 1)$
- Arrays ($\mathcal{AR}$): $(i = j) \vee read(write(a, i, e), j) = read(a, j)$
- Bit vectors ($\mathcal{BV}$): $x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[8]}[3:0]$
- Non-Linear arithmetic over the reals ($\mathcal{NLA}(\mathbb{R})$) :
  $((c = a \cdot b) \wedge (a_1 = a - 1) \wedge (b_1 = b + 1)) \rightarrow (c = a_1 \cdot b_1 + 1)$
- ...

SMT($\mathcal{T}$): Theories of Interest

Some theories of interest (e.g., for formal verification)

- Equality and Uninterpreted Functions ($\mathcal{EUF}$): $((x = y) \wedge (y = f(z))) \rightarrow (g(x) = g(f(z)))$
- Difference logic ($\mathcal{DL}$): $((x = y) \wedge (y - z \leq 4)) \rightarrow (x - z \leq 6)$
- UTVPI ($\mathcal{UTVPI}$): $((x = y) \wedge (y - z \leq 4)) \rightarrow (x + z \leq 6)$
- Linear arithmetic over the rationals ($\mathcal{LRA}$): $(T_\delta \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \wedge (\neg T_\delta \rightarrow (s_1 = s_0))$
- Linear arithmetic over the integers ($\mathcal{LIA}$): $(x = x_l + 2^{16} x_h) \wedge (x \geq 0) \wedge (x \leq 2^{16} - 1)$
- Arrays ($\mathcal{AR}$): $(i = j) \vee read(write(a, i, e), j) = read(a, j)$
- Bit vectors ($\mathcal{BV}$): $x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[8]}[3:0]$
- Non-Linear arithmetic over the reals ($\mathcal{NLA}(\mathbb{R})$) : $((c = a \cdot b) \wedge (a_1 = a - 1) \wedge (b_1 = b + 1)) \rightarrow (c = a_1 \cdot b_1 + 1)$
- ...

SMT($\mathcal{T}$): Theories of Interest

Some theories of interest (e.g., for formal verification)

- Equality and Uninterpreted Functions ($\mathcal{EUF}$):
  $((x = y) \land (y = f(z))) \rightarrow (g(x) = g(f(z)))$
- Difference logic ($\mathcal{DL}$): $((x = y) \land (y - z \leq 4)) \rightarrow (x - z \leq 6)$
- UTVPI ($\mathcal{UTVPI}$): $((x = y) \land (y - z \leq 4)) \rightarrow (x + z \leq 6)$
- Linear arithmetic over the rationals ($\mathcal{LRA}$):
  $(T_\delta \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \land (\neg T_\delta \rightarrow (s_1 = s_0))$
- Linear arithmetic over the integers ($\mathcal{LIA}$): $(x = x_l + 2^{16} x_h) \land (x \geq 0) \land (x \leq 2^{16} - 1)$
- Arrays ($\mathcal{AR}$): $(i = j) \lor read(write(a, i, e), j) = read(a, j)$
- Bit vectors ($\mathcal{BV}$): $x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[8]}[3:0]$
- Non-Linear arithmetic over the reals ($\mathcal{NLA}(\mathbb{R})$) :
  $((c = a \cdot b) \land (a_1 = a - 1) \land (b_1 = b + 1)) \rightarrow (c = a_1 \cdot b_1 + 1)$
- ...

Satisfiability Modulo Theories (SMT($\mathcal{T}$)): Example

Example: SMT($\mathcal{LIA} \cup \mathcal{EUF} \cup \mathcal{AR}$)

$$\varphi \overset{\text{def}}{=} (d \geq 0) \wedge (d < 1) \wedge$$
$$((f(d) = f(0)) \rightarrow (read(write(V, i, x), i + d) = x + 1))$$

- involves arithmetical, arrays, and uninterpreted function/predicate symbols, plus Boolean operators
    - Is it satisfiable?
    - No:

$$\begin{array}{ll} & \varphi \\ \Longrightarrow_{\mathcal{LIA}} & (d = 0) \\ \Longrightarrow_{\mathcal{EUF}} & (f(d) = f(0)) \\ \Longrightarrow_{Bool} & (read(write(V, i, x), i + d) = x + 1) \\ \Longrightarrow_{\mathcal{LIA}} & (read(write(V, i, x), i) = x + 1) \\ \Longrightarrow_{\mathcal{LIA}} & \neg(read(write(V, i, x), i) = x) \\ \Longrightarrow_{\mathcal{AR}} & \bot \end{array}$$

Satisfiability Modulo Theories (SMT($\mathcal{T}$)): Example

Example: SMT($\mathcal{LIA} \cup \mathcal{EUF} \cup \mathcal{AR}$)

$$\varphi \overset{\text{def}}{=} (d \geq 0) \wedge (d < 1) \wedge$$
$$((f(d) = f(0)) \rightarrow (read(write(V, i, x), i + d) = x + 1))$$

- involves arithmetical, arrays, and uninterpreted function/predicate symbols, plus Boolean operators
  - Is it satisfiable?
  - No:

$$\begin{array}{ll} & \varphi \\ \Longrightarrow_{\mathcal{LIA}} & (d = 0) \\ \Longrightarrow_{\mathcal{EUF}} & (f(d) = f(0)) \\ \Longrightarrow_{Bool} & (read(write(V, i, x), i + d) = x + 1) \\ \Longrightarrow_{\mathcal{LIA}} & (read(write(V, i, x), i) = x + 1) \\ \Longrightarrow_{\mathcal{LIA}} & \neg(read(write(V, i, x), i) = x) \\ \Longrightarrow_{\mathcal{AR}} & \bot \end{array}$$

Satisfiability Modulo Theories (SMT($\mathcal{T}$)): Example

Example: SMT($\mathcal{LIA} \cup \mathcal{EUF} \cup \mathcal{AR}$)

$$\varphi \stackrel{\text{def}}{=} (d \geq 0) \wedge (d < 1) \wedge$$
$$((f(d) = f(0)) \rightarrow (read(write(V, i, x), i + d) = x + 1))$$

- involves arithmetical, arrays, and uninterpreted function/predicate symbols, plus Boolean operators
  - Is it satisfiable?
  - No:

$$\begin{array}{ll} & \varphi \\ \Longrightarrow_{\mathcal{LIA}} & (d = 0) \\ \Longrightarrow_{\mathcal{EUF}} & (f(d) = f(0)) \\ \Longrightarrow_{Bool} & (read(write(V, i, x), i + d) = x + 1) \\ \Longrightarrow_{\mathcal{LIA}} & (read(write(V, i, x), i) = x + 1) \\ \Longrightarrow_{\mathcal{LIA}} & \neg(read(write(V, i, x), i) = x) \\ \Longrightarrow_{\mathcal{AR}} & \bot \end{array}$$

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for heavy Boolean reasoning combined with capabilities for reasoning in expressive decidable F.O. theories

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine SAT solvers with $\mathcal{T}$-specific decision procedures (theory solvers or $\mathcal{T}$-solvers)
  - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for heavy Boolean reasoning combined with capabilities for reasoning in expressive decidable F.O. theories

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine SAT solvers with $\mathcal{T}$-specific decision procedures (theory solvers or $\mathcal{T}$-solvers)
  - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for heavy Boolean reasoning combined with capabilities for reasoning in expressive decidable F.O. theories

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine SAT solvers with $\mathcal{T}$-specific decision procedures (theory solvers or $\mathcal{T}$-solvers)
  - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for heavy Boolean reasoning combined with capabilities for reasoning in expressive decidable F.O. theories

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine SAT solvers with $\mathcal{T}$-specific decision procedures (theory solvers or $\mathcal{T}$-solvers)
    - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for heavy Boolean reasoning combined with capabilities for reasoning in expressive decidable F.O. theories

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine SAT solvers with $\mathcal{T}$-specific decision procedures (theory solvers or $\mathcal{T}$-solvers)
  - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)

Notational remark (1): most/all examples in $\mathcal{LRA}$

For better readability, in most/all the examples of this presentation we will use the theory of linear arithmetic on rational numbers ($\mathcal{LRA}$) because of its intuitive semantics. E.g.:

$$(\neg A_1 \vee (3x_1 - 2x_2 - 3 \leq 5)) \wedge (A_2 \vee (-2x_1 + 4x_3 + 2 = 3))$$

Nevertheless, analogous examples can be built with all other theories of interest.

Notational remark (2): “constants” vs. “variables”

- Consider, e.g., the formula:
  $(\neg A_1 \vee (3x_1 - 2x_2 - 3 \leq 5)) \wedge (A_2 \vee (-2x_1 + 4x_3 + 2 = 3))$
- How do we call A_1, A_2?:
  (*a*) Boolean/propositional variables?
  (*b*) uninterpreted 0-ary predicates?
- How do we call x_1, x_2, x_3?:
  (*a*) domain variables?
  (*b*) uninterpreted Skolem constants/0-ary uninterpreted functions?
- Hint:
  (*a*) typically used in SAT, CSP and OR communities
  (*b*) typically used in logic & ATP communities

Hereafter we call A_1, A_2 “Boolean/propositional variables” and x_1, x_2, x_3 "domain variables” (logic purists, please forgive me!)

Notational remark (2): “constants” vs. “variables”

- Consider, e.g., the formula:
  $(\neg A_1 \vee (3x_1 - 2x_2 - 3 \leq 5)) \wedge (A_2 \vee (-2x_1 + 4x_3 + 2 = 3))$
- How do we call A_1, A_2?:
  (*a*) Boolean/propositional variables?
  (*b*) uninterpreted 0-ary predicates?
- How do we call x_1, x_2, x_3?:
  (*a*) domain variables?
  (*b*) uninterpreted Skolem constants/0-ary uninterpreted functions?
- Hint:
  (*a*) typically used in SAT, CSP and OR communities
  (*b*) typically used in logic & ATP communities

Hereafter we call A_1, A_2 “Boolean/propositional variables” and x_1, x_2, x_3 "domain variables” (logic purists, please forgive me!)

Notational remark (2): “constants” vs. “variables”

- Consider, e.g., the formula:
  $(\neg A_1 \vee (3x_1 - 2x_2 - 3 \leq 5)) \wedge (A_2 \vee (-2x_1 + 4x_3 + 2 = 3))$
- How do we call A_1, A_2?:
  - (a) Boolean/propositional variables?
  - (b) uninterpreted 0-ary predicates?
- How do we call x_1, x_2, x_3?:
  - (a) domain variables?
  - (b) uninterpreted Skolem constants/0-ary uninterpreted functions?
- Hint:
  - (a) typically used in SAT, CSP and OR communities
  - (b) typically used in logic & ATP communities

Hereafter we call A_1, A_2 “Boolean/propositional variables” and x_1, x_2, x_3 "domain variables” (logic purists, please forgive me!)

Outline

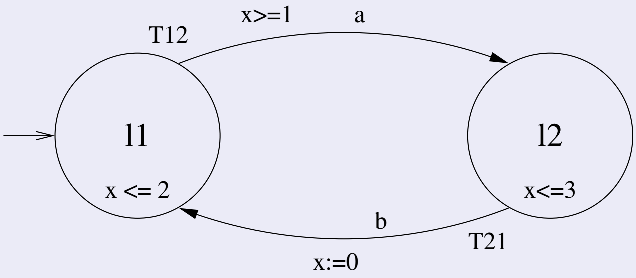
1. Introduction
   - Basics on First-order Logic
   - What is a Theory?
   - Satisfiability Modulo Theories
   - Motivations and Goals of SMT
2. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for Theories of Interest (hints)
   - SMT for Combinations of Theories
3. Beyond Solving: Advanced SMT Functionalities
   - Proofs and Unsatisfiable Cores
   - All-SMT & Predicate Abstraction (hints)
   - SMT with Optimization (Optimization Modulo Theories)

Some Motivating Applications

Interest in SMT triggered by some real-word applications

- Verification of Hybrid & Timed Systems
- Verification of RTL Circuit Designs & of Microcode
- SW Verification
- Planning with Resources
- Temporal reasoning
- Scheduling
- Compiler optimization
- ...

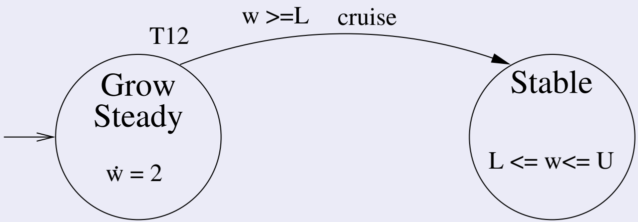
Verification of Timed Systems



- Model checking of Timed Systems [6, 35, 58], ...
- Timed Automata encoded into $\mathcal{T}$-formulas:
  - discrete information (locations, transitions, events) with Boolean vars.
  - timed information (clocks, elapsed time) with differences $(t_3 - x_3 \leq 2)$, equalities $(x_4 = x_3)$ and linear constraints $(t_8 - x_8 = t_2 - x_2)$ on $\mathbb{Q}$

$\Longrightarrow$ SMT on $\mathcal{DL}(\mathbb{Q})$ or $\mathcal{LRA}$ required

Verification of Hybrid Systems ...

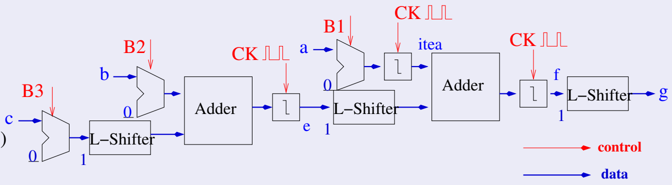


- Model checking of Hybrid Systems [5],...
- Hybrid Automata encoded into $\mathcal{L}$-formulas:
  - discrete information (locs, trans., events) with Boolean vars.
  - timed information (clocks, elapsed time) with differences $(t_3 - x_3 \leq 2)$, equalities $(x_4 = x_3)$ and linear constraints $(t_8 - x_8 = t_2 - x_2)$ on $\mathbb{Q}$
  - Evolution of Physical Variables (e.g., speed, pressure) with linear $(\omega_4 = 2\omega_3)$ and non-linear constraints $(P_1 V_1 = 4 T_1)$ on $\mathbb{Q}$
- Undecidable under simple hypotheses!

$\Longrightarrow$ SMT on $\mathcal{DL}(\mathbb{Q})$, $\mathcal{LRA}$ or $\mathcal{NLA}(\mathbb{R})$ required

Verification of HW circuit designs & microcode

g = 2*f
f' = itea+2*e
itea' = ITE(B1;a;0)
e' = ITE(B2;b;0)+2*ite(B3;c;0)



- SAT/SMT-based Model Checking & Equiv. Checking of RTL designs, symbolic simulation of μ-code [25, 22, 42]
- Control paths handled by Boolean reasoning
- Data paths information abstracted into theory-specific terms
  - words (bit-vectors, integers, $\mathcal{EUF}$ vars, ...): $\underline{a}[31:0]$, a
  - word operations: $(\mathcal{BV}, \mathcal{EUF}, \mathcal{AR}, \mathcal{LIA}, \mathcal{NLA}(\mathbb{Z})$ operators)
    $x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[8]}[3:0]$, $(a = a_L + 2^{16}a_H)$, $(m_1 = store(m_0, l_0, v_0))$,
    ...
- Trades heavy Boolean reasoning ($\approx 2^{64}$ factors) with $\mathcal{T}$-solving

$\Longrightarrow$ SMT on $\mathcal{BV}$, $\mathcal{EUF}$, $\mathcal{AR}$, modulo-$\mathcal{LIA}$ [$\mathcal{NLA}(\mathbb{Z})$] required

Verification of SW systems

```
...
10. i = 0;
11. acc = 0.0;
12. while (i<dim) {
13.   acc += V[i];
14.   i++;
15. }
...
```

....

$(pc = 10) \rightarrow ((i' = 0) \wedge (pc' = 11))$

$(pc = 11) \rightarrow ((acc' = 0.0) \wedge (pc' = 12))$

$(pc = 12) \rightarrow ((i < dim) \rightarrow \wedge(pc' = 13))$

$(pc = 12) \rightarrow (\neg(i < dim) \rightarrow \wedge(pc' = 16))$

$(pc = 13) \rightarrow ((acc' = acc + read(V, i)) \wedge (pc' = 14))$

$(pc = 14) \rightarrow (i' = i + 1) \wedge (pc' = 15))$

$(pc = 15) \rightarrow (pc' = 12))$

...

- Verification of SW code
  - BMC, K-induction, Check of proof obligations, interpolation-based model checking, symbolic simulation, concolic testing, ...

$\Longrightarrow$ SMT on $\mathcal{BV}$, $\mathcal{EUF}$, $\mathcal{AR}$, (modulo-)$\mathcal{LIA}$ [$\mathcal{NLA}(\mathbb{Z})$] required

Planning with Resources [80]

- SAT-bases planning augmented with numerical constraints
- Straightforward to encode into into SMT($\mathcal{LRA}$)

Example (sketch) [80]

$$\begin{array}{ll}
(\mathit{Deliver}) & \wedge\ \texttt{// goal} \\
(\mathit{MaxLoad}) & \wedge\ \texttt{// load constraint} \\
(\mathit{MaxFuel}) & \wedge\ \texttt{// fuel constraint} \\
(\mathit{Move} \to \mathit{MinFuel}) & \wedge\ \texttt{// move requires fuel} \\
(\mathit{Move} \to \mathit{Deliver}) & \wedge\ \texttt{// move implies delivery} \\
(\mathit{GoodTrip} \to \mathit{Deliver}) & \wedge\ \texttt{// a good trip requires} \\
(\mathit{GoodTrip} \to \mathit{AllLoaded}) & \wedge\ \texttt{// a full delivery} \\
\hline
(\mathit{MaxLoad} \to (\mathit{load} \leq 30)) & \wedge\ \texttt{// load limit} \\
(\mathit{MaxFuel} \to (\mathit{fuel} \leq 15)) & \wedge\ \texttt{// fuel limit} \\
(\mathit{MinFuel} \to (\mathit{fuel} \geq 7 + 0.5\mathit{load})) & \wedge\ \texttt{// fuel constraint} \\
(\mathit{AllLoaded} \to (\mathit{load} = 45)) & \ \texttt{//}
\end{array}$$

(Disjunctive) Temporal Reasoning & Scheduling [77, 2]

- Temporal reasoning problems encoded as disjunctions of difference constraints

$$\begin{array}{llll} ((x_1 - x_2 \leq 6) & \vee (x_3 - x_4 \leq -2)) & \wedge \\ ((x_2 - x_3 \leq -2) & \vee (x_4 - x_5 \leq 5)) & \wedge \\ ((x_2 - x_1 \leq 4) & \vee (x_3 - x_7 \leq -6)) & \wedge \\ \ldots & & \end{array}$$

- Straightforward to encode into into SMT($\mathcal{DL}$)

Goal

Provide an overview of standard “lazy” SMT:

- foundations
- SMT-solving techniques
- beyond solving: advanced SMT functionalities
- ongoing research

We do not cover related approaches like:

- Eager SAT encodings
- Rewrite-based approaches

We refer to [70, 10] for an overview and references.

Goal

Provide an overview of standard “lazy” SMT:

- foundations
- SMT-solving techniques
- beyond solving: advanced SMT functionalities
- ongoing research

We do not cover related approaches like:

- Eager SAT encodings
- Rewrite-based approaches

We refer to [70, 10] for an overview and references.

Outline

Outline

1. Introduction
   - Basics on First-order Logic
   - What is a Theory?
   - Satisfiability Modulo Theories
   - Motivations and Goals of SMT

2. **Efficient SMT solving**
   - **Combining SAT with Theory Solvers**
   - Theory Solvers for Theories of Interest (hints)
   - SMT for Combinations of Theories

3. Beyond Solving: Advanced SMT Functionalities
   - Proofs and Unsatisfiable Cores
   - All-SMT & Predicate Abstraction (hints)
   - SMT with Optimization (Optimization Modulo Theories)

Modern “lazy” SMT($\mathcal{T}$) solvers

A prominent “lazy” approach [45, 2, 80, 3, 8, 35] (aka “DPLL($\mathcal{T}$)”)

- a CDCL SAT solver is used to enumerate truth assignments μ_i for (the Boolean abstraction φ^p of) the input formula φ
  - the Boolean abstraction φ^p of φ maps theory atoms in φ into fresh Boolean variables
- a theory-specific solver $\mathcal{T}$*-solver* checks the $\mathcal{T}$-satisfiability of the set of $\mathcal{T}$-literals corresponding to each assignment

- Built on top of modern SAT CDCL solvers
  - benefit for free from all modern CDCL techniques (e.g., Boolean preprocessing, backjumping & learning, restarts,...)
  - benefit for free from all state-of-the-art data structures and implementation tricks (e.g., two-watched literals,...)
- Many techniques to maximize the benefits of integration [70, 10]
- Many lazy SMT tools available (Barcelogic, CVC5, MathSAT, OpenSMT, Yices, Z3, . . .)

Modern “lazy” SMT($\mathcal{T}$) solvers

A prominent “lazy” approach [45, 2, 80, 3, 8, 35] (aka “DPLL($\mathcal{T}$)”)

- a CDCL SAT solver is used to enumerate truth assignments μ_i for (the Boolean abstraction φ^p of) the input formula φ
  - the Boolean abstraction φ^p of φ maps theory atoms in φ into fresh Boolean variables
- a theory-specific solver $\mathcal{T}$*-solver* checks the $\mathcal{T}$-satisfiability of the set of $\mathcal{T}$-literals corresponding to each assignment

- Built on top of modern SAT CDCL solvers
  - benefit for free from all modern CDCL techniques (e.g., Boolean preprocessing, backjumping & learning, restarts,...)
  - benefit for free from all state-of-the-art data structures and implementation tricks (e.g., two-watched literals,...)
- Many techniques to maximize the benefits of integration [70, 10]
- Many lazy SMT tools available (Barcelogic, CVC5, MathSAT, OpenSMT, Yices, Z3, . . .)

Basic schema: example

$\varphi =$		$\varphi^p =$
c_1 :	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
c_2 :	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
c_3 :	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
c_4 :	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
c_5 :	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
c_6 :	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
c_7 :	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$

true, *false*

$$\begin{aligned}
\mu^p &= \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\} \\
\mu &= \{\underline{\neg(3v_1 - v_3 \leq 6)}, \underline{(v_3 = 3v_5 + 4)}, (v_2 - v_4 \leq 6), \\
&\quad \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), \underline{(v_1 - v_5 \leq 1)}\}
\end{aligned}$$

$\Longrightarrow$ unsatisfiable in $\mathcal{LRA}$ $\Longrightarrow$ backtrack

Basic schema: example

$\varphi =$		$\varphi^p =$
$c_1:$	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
$c_2:$	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
$c_3:$	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
$c_4:$	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
$c_5:$	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
$c_6:$	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
$c_7:$	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$

true, *false*

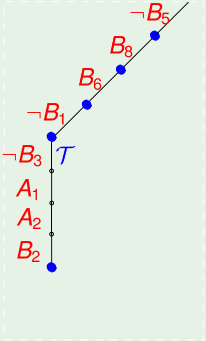
$$
\begin{aligned}
\mu^p &= \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\} \\
\mu &= \{\underline{\neg(3v_1 - v_3 \leq 6)}, \underline{(v_3 = 3v_5 + 4)}, (v_2 - v_4 \leq 6), \\
&\quad \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), \underline{(v_1 - v_5 \leq 1)}\}
\end{aligned}
$$

$\Longrightarrow$ unsatisfiable in $\mathcal{LRA}$ $\Longrightarrow$ backtrack

Basic schema: example

$\varphi =$		$\varphi^p =$
c_1 :	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
c_2 :	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
c_3 :	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
c_4 :	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
c_5 :	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
c_6 :	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
c_7 :	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$

true, *false*



$$\begin{aligned}
\mu^p &= \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\} \\
\mu &= \{\underline{\neg(3v_1 - v_3 \leq 6)}, \underline{(v_3 = 3v_5 + 4)}, (v_2 - v_4 \leq 6), \\
&\quad \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), \underline{(v_1 - v_5 \leq 1)}\}
\end{aligned}$$

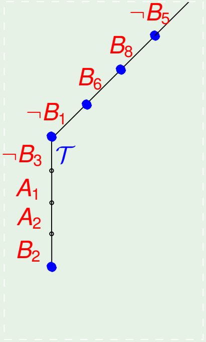
$\Longrightarrow$ unsatisfiable in $\mathcal{LRA}$ $\Longrightarrow$ backtrack

Basic schema: example

$\varphi =$
$c_1: \quad \neg(2v_2 - v_3 > 2) \vee A_1$
$c_2: \quad \neg A_2 \vee (v_1 - v_5 \leq 1)$
$c_3: \quad (3v_1 - 2v_2 \leq 3) \vee A_2$
$c_4: \quad \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$
$c_5: \quad A_1 \vee (3v_1 - 2v_2 \leq 3)$
$c_6: \quad (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$
$c_7: \quad A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$

true, *false*

$\varphi^p =$
$\neg B_1 \vee A_1$
$\neg A_2 \vee B_2$
$B_3 \vee A_2$
$\neg B_4 \vee \neg B_5 \vee \neg A_1$
$A_1 \vee B_3$
$B_6 \vee B_7 \vee \neg A_1$
$A_1 \vee B_8 \vee A_2$

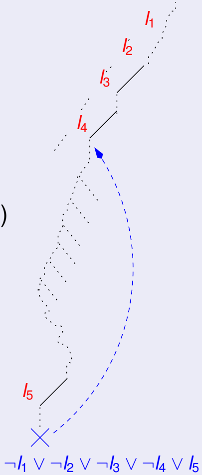


$$
\begin{aligned}
\mu^p &= \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\} \\
\mu &= \{\underline{\neg(3v_1 - v_3 \leq 6)}, \underline{(v_3 = 3v_5 + 4)}, (v_2 - v_4 \leq 6), \\
&\quad \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), \underline{(v_1 - v_5 \leq 1)}\}
\end{aligned}
$$

$\Longrightarrow$ unsatisfiable in $\mathcal{LRA}$ $\Longrightarrow$ backtrack

$\mathcal{T}$-Backjumping & $\mathcal{T}$-learning [50, 80, 3, 8, 35]

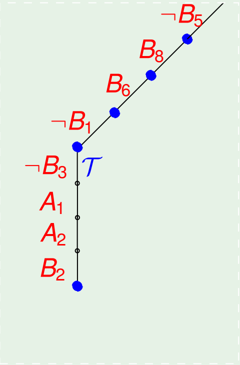
- Similar to Boolean backjumping & learning
- important property of $\mathcal{T}$-solver:
  - extraction of $\mathcal{T}$-conflict sets: if μ is $\mathcal{T}$-unsatisfiable, then $\mathcal{T}$*-solver* (μ) returns the subset η of μ causing the $\mathcal{T}$-unsatisfiability of μ ($\mathcal{T}$-conflict set)
- If so, the $\mathcal{T}$-conflict clause $C := \neg\eta$ is used to drive the backjumping & learning mechanism of the SAT solver
  $\Longrightarrow$ lots of search saved
- the less redundant is η, the more search is saved



$\mathcal{T}$-Backjumping & $\mathcal{T}$-learning: example

$\varphi =$		$\varphi^p =$
$c_1:$	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
$c_2:$	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
$c_3:$	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
$c_4:$	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
$c_5:$	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
$c_6:$	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
$c_7:$	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$
$c_8:$	$(3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \ldots$	$B_5 \vee \neg B_8 \vee \neg B_2$

true, *false*

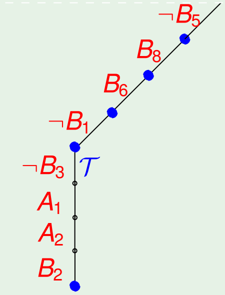


$$
\begin{aligned}
\mu^p &= \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\} \\
\mu &= \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2), \\
&\quad \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\} \\
\eta &= \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\} \\
\eta^p &= \{\neg B_5, B_8, B_2\}
\end{aligned}
$$

$\mathcal{T}$-Backjumping & $\mathcal{T}$-learning: example

	$\varphi =$	$\varphi^p =$
c_1 :	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
c_2 :	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
c_3 :	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
c_4 :	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
c_5 :	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
c_6 :	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
c_7 :	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$
c_8 :	$(3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \ldots$	$B_5 \vee \neg B_8 \vee \neg B_2$

true, *false*



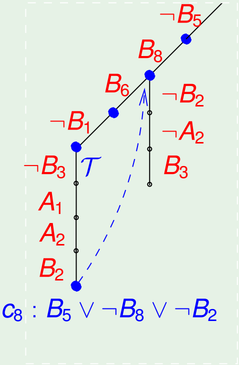
$c_8 : B_5 \vee \neg B_8 \vee \neg B_2$

$$\begin{aligned}
\mu^p &= \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\} \\
\mu &= \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2), \\
&\quad \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\} \\
\eta &= \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\} \\
\eta^p &= \{\neg B_5, B_8, B_2\}
\end{aligned}$$

$\mathcal{T}$-Backjumping & $\mathcal{T}$-learning: example

$\varphi =$		$\varphi^p =$
c_1 :	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
c_2 :	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
c_3 :	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
c_4 :	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
c_5 :	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
c_6 :	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
c_7 :	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$
c_8 :	$(3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \ldots$	$B_5 \vee \neg B_8 \vee \neg B_2$

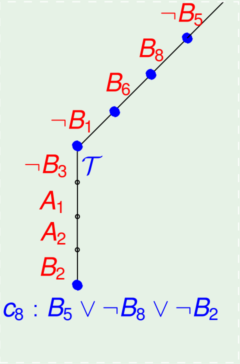
true, *false*



$$
\begin{aligned}
\mu^p &= \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\} \\
\mu &= \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2), \\
&\quad \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\} \\
\eta &= \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\} \\
\eta^p &= \{\neg B_5, B_8, B_2\}
\end{aligned}
$$

$\mathcal{T}$-Backjumping & $\mathcal{T}$-learning: example (2)

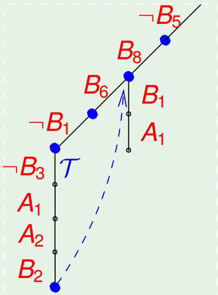
	$\varphi =$	$\varphi^p =$
c_1 :	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
c_2 :	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
c_3 :	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
c_4 :	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
c_5 :	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
c_6 :	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
c_7 :	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$
c_8' :	$(3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \ldots$	$B_5 \vee \neg B_8 \vee B_1$
c_8 :	$(3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \ldots$	$B_5 \vee \neg B_8 \vee \neg B_2$
	true, *false*	



$$\cfrac{\cfrac{\cfrac{\overbrace{B_5 \vee \neg B_8 \vee \neg B_2}^{c_8:\ \text{theory conflicting clause}} \qquad \overbrace{\neg A_2 \vee B_2}^{c_2}}{B_5 \vee \neg B_8 \vee \neg A_2}\ (B_2) \qquad \overbrace{B_3 \vee A_2}^{c_3}}{B_5 \vee \neg B_8 \vee B_3}\ (\neg A_2) \qquad \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_\mathcal{T}}}{\underbrace{B_5 \vee \neg B_8 \vee B_1}_{c_8':\ \text{mixed Boolean+theory conflict clause}}}\ (B_3)$$

$\mathcal{T}$-Backjumping & $\mathcal{T}$-learning: example (2)

$\varphi =$		$\varphi^p =$
c_1 :	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
c_2 :	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
c_3 :	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
c_4 :	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
c_5 :	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
c_6 :	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
c_7 :	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$
c'_8 :	$(3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$	$B_5 \vee \neg B_8 \vee B_1$
c_8 :	$(3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$	$B_5 \vee \neg B_8 \vee \neg B_2$
	true, *false*	

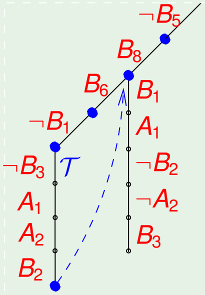


$c'_8 : B_5 \vee \neg B_8 \vee B_1$

$$\cfrac{\cfrac{\cfrac{\overbrace{B_5 \vee \neg B_8 \vee \neg B_2}^{c_8:\ \textit{theory conflicting clause}} \qquad \overbrace{\neg A_2 \vee B_2}^{c_2}}{B_5 \vee \neg B_8 \vee \neg A_2}\,(B_2) \qquad \overbrace{B_3 \vee A_2}^{c_3}}{B_5 \vee \neg B_8 \vee B_3}\,(\neg A_2) \qquad \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_{\mathcal{T}}}}{\underbrace{B_5 \vee \neg B_8 \vee B_1}_{c'_8:\ \textit{mixed Boolean+theory conflict clause}}}\,(B_3)$$

$\mathcal{T}$-Backjumping & $\mathcal{T}$-learning: example (2)

	$\varphi =$	$\varphi^p =$
c_1 :	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
c_2 :	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
c_3 :	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
c_4 :	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
c_5 :	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
c_6 :	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
c_7 :	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$
c_8' :	$(3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \ldots$	$B_5 \vee \neg B_8 \vee B_1$
c_8 :	$(3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \ldots$	$B_5 \vee \neg B_8 \vee \neg B_2$
	true, *false*	



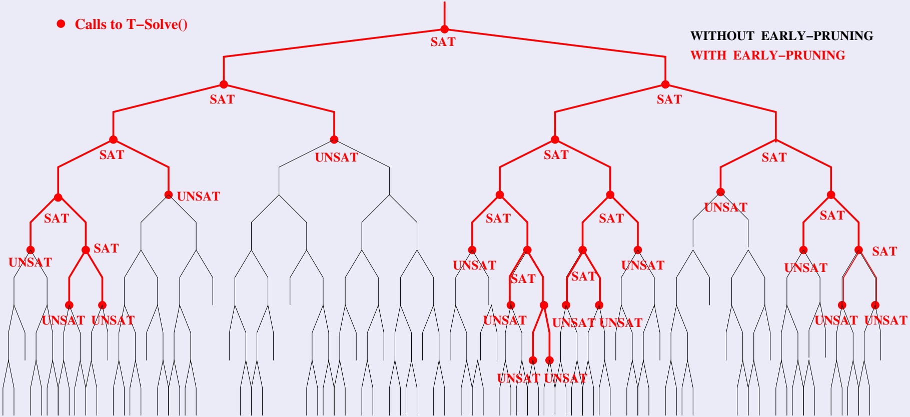
c_8' : $B_5 \vee \neg B_8 \vee B_1$

c_8 : $B_5 \vee \neg B_8 \vee \neg B_2$

$$
\cfrac{
  \cfrac{
    \cfrac{\overbrace{B_5 \vee \neg B_8 \vee \neg B_2}^{c_8:\ \textit{theory conflicting clause}} \qquad \overbrace{\neg A_2 \vee B_2}^{c_2}}{B_5 \vee \neg B_8 \vee \neg A_2}\ (B_2) \qquad \overbrace{B_3 \vee A_2}^{c_3}
  }{B_5 \vee \neg B_8 \vee B_3}\ (\neg A_2) \qquad \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_{\mathcal{T}}}
}{\underbrace{B_5 \vee \neg B_8 \vee B_1}_{c_8':\ \textit{mixed Boolean+theory conflict clause}}}\ (B_3)
$$

Early Pruning [45, 2, 80]

- Introduce a $\mathcal{T}$-satisfiability test on intermediate assignments: if $\mathcal{T}$*-solver* returns UNSAT, the procedure backtracks.
  - benefit: prunes drastically the Boolean search
  - Drawback: possibly many useless calls to $\mathcal{T}$*-solver*



Early pruning: example

$$\begin{aligned}\varphi = & \{\neg(2v_2 - v_3 > 2) \vee A_1\} \wedge \\ & \{\neg A_2 \vee (2v_1 - 4v_5 > 3)\} \wedge \\ & \{(3v_1 - 2v_2 \leq 3) \vee A_2\} \wedge \\ & \{\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1\} \wedge \\ & \{A_1 \vee (3v_1 - 2v_2 \leq 3)\} \wedge \\ & \{(v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg A_1\} \wedge \\ & \{A_1 \vee (v_3 = 3v_5 + 4) \vee A_2\}.\end{aligned}$$

$$\begin{aligned}\varphi^p = & \{\neg B_1 \vee A_1\} \wedge \\ & \{\neg A_2 \vee B_2\} \wedge \\ & \{B_3 \vee A_2\} \wedge \\ & \{\neg B_4 \vee \neg B_5 \vee \neg A_1\} \wedge \\ & \{A_1 \vee B_3\} \wedge \\ & \{B_6 \vee B_7 \vee \neg A_1\} \wedge \\ & \{A_1 \vee B_8 \vee A_2\}.\end{aligned}$$

- Suppose it is built the intermediate assignment:

$$\mu'^p = \neg B_1 \wedge \neg A_2 \wedge B_3 \wedge \neg B_5.$$

corresponding to the following set of $\mathcal{T}$-literals

$$\mu' = \neg(2v_2 - v_3 > 2) \wedge \neg A_2 \wedge (3v_1 - 2v_2 \leq 3) \wedge \neg(3v_1 - v_3 \leq 6).$$

- If $\mathcal{T}$-*solver* is invoked on μ', then it returns UNSAT, and DPLL backtracks without exploring any extension of μ'.

Early Pruning [45, 2, 80] (cont.)

- Different strategies for interleaving Boolean search steps and $\mathcal{T}$-*solver* calls
  - Eager E.P. [80, 11, 78, 44]): invoke $\mathcal{T}$-*solver* every time a new $\mathcal{T}$-atom is added to the assignment (unit propagations included)
  - Selective E.P.: Do not call $\mathcal{T}$-*solver* if the have been added only literals which hardly cause any $\mathcal{T}$-conflict with the previous assignment (e.g., Boolean literals, disequalities $(x - y \neq 3)$, $\mathcal{T}$-literals introducing new variables $(x - z = 3)$)
  - Weakened E.P.: for intermediate checks only, use weaker but faster versions of $\mathcal{T}$-*solver* (e.g., check μ on $\mathbb{R}$ rather than on $\mathbb{Z}$): $\{(x - y \leq 4), (z - x \leq -6), (z = y), (3x + 2y - 3z = 4)\}$

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$-*solvers*

- With early pruning, lots of incremental calls to $\mathcal{T}$-*solver*:

$\mathcal{T}$-*solver* (μ_1)	$\Rightarrow$ *Sat*	Undo μ_4, μ_3, μ_2	
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2)$	$\Rightarrow$ *Sat*	$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2')$	$\Rightarrow$ *Sat*
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow$ *Sat*	$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2' \cup \mu_3')$	$\Rightarrow$ *Sat*
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow$ *Unsat*	...	

$\Longrightarrow$ Desirable features of $\mathcal{T}$-*solvers*:

  - incrementality: $\mathcal{T}$-*solver*$(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}$-*solver*(μ_1) without restarting from scratch
  - backtrackability (resettability): $\mathcal{T}$-*solver* can efficiently undo steps and return to a previous status on the stack

$\Longrightarrow$ $\mathcal{T}$-*solver* requires a stack-based interface

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$-*solvers*

- With early pruning, lots of incremental calls to $\mathcal{T}$-*solver*:

$\mathcal{T}$-*solver* (μ_1)	$\Rightarrow$ *Sat*	Undo μ_4, μ_3, μ_2	
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2)$	$\Rightarrow$ *Sat*	$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2')$	$\Rightarrow$ *Sat*
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow$ *Sat*	$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2' \cup \mu_3')$	$\Rightarrow$ *Sat*
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow$ *Unsat*	...	

$\Longrightarrow$ Desirable features of $\mathcal{T}$-*solvers*:

- incrementality: $\mathcal{T}$-*solver*$(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}$-*solver*(μ_1) without restarting from scratch
- backtrackability (resettability): $\mathcal{T}$-*solver* can efficiently undo steps and return to a previous status on the stack

$\Longrightarrow$ $\mathcal{T}$-*solver* requires a stack-based interface

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$-*solvers*

- With early pruning, lots of incremental calls to $\mathcal{T}$-*solver*:

$\mathcal{T}$-*solver* (μ_1)	$\Rightarrow$ *Sat*	Undo μ_4, μ_3, μ_2	
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2)$	$\Rightarrow$ *Sat*	$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2')$	$\Rightarrow$ *Sat*
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow$ *Sat*	$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2' \cup \mu_3')$	$\Rightarrow$ *Sat*
$\mathcal{T}$-*solver* $(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow$ *Unsat*	...	

$\Longrightarrow$ Desirable features of $\mathcal{T}$-*solvers*:

  - incrementality: $\mathcal{T}$-*solver*$(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}$-*solver*(μ_1) without restarting from scratch
  - backtrackability (resettability): $\mathcal{T}$-*solver* can efficiently undo steps and return to a previous status on the stack

$\Longrightarrow$ $\mathcal{T}$-*solver* requires a stack-based interface

$\mathcal{T}$-Propagation [2, 3, 44]

- strictly related to early pruning
- important property of $\mathcal{T}$-*solver*:
  - $\mathcal{T}$-deduction: when a partial assignment μ is $\mathcal{T}$-satisfiable, $\mathcal{T}$-*solver* may be able to return also an assignment η to some unassigned atom occurring in φ s.t. $\mu \models_{\mathcal{T}} \eta$.
- If so:
  - the literal η is then unit-propagated;
  - optionally, a $\mathcal{T}$-deduction clause $C := \neg\mu' \vee \eta$ can be learned, μ' being the subset of μ which caused the deduction ($\mu' \models_{\mathcal{T}} \eta$)
  - lazy explanation: compute C only if needed for conflict analysis

$\Longrightarrow$ may prune drastically the search

Both $\mathcal{T}$-deduction clauses and $\mathcal{T}$-conflict clauses are called $\mathcal{T}$-lemmas since they are valid in $\mathcal{T}$

$\mathcal{T}$-Propagation [2, 3, 44]

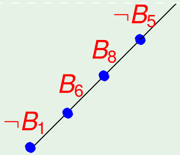
- strictly related to early pruning
- important property of $\mathcal{T}$-*solver*:
  - $\mathcal{T}$-deduction: when a partial assignment μ is $\mathcal{T}$-satisfiable, $\mathcal{T}$-*solver* may be able to return also an assignment η to some unassigned atom occurring in φ s.t. $\mu \models_{\mathcal{T}} \eta$.
- If so:
  - the literal η is then unit-propagated;
  - optionally, a $\mathcal{T}$-deduction clause $C := \neg\mu' \vee \eta$ can be learned, μ' being the subset of μ which caused the deduction $(\mu' \models_{\mathcal{T}} \eta)$
  - lazy explanation: compute C only if needed for conflict analysis

$\Longrightarrow$ may prune drastically the search

Both $\mathcal{T}$-deduction clauses and $\mathcal{T}$-conflict clauses are called $\mathcal{T}$-lemmas since they are valid in $\mathcal{T}$

$\mathcal{T}$-propagation: example

$$
\begin{array}{lll}
\varphi = & & \varphi^p = \\
c_1: & \neg(2v_2 - v_3 > 2) \vee A_1 & \neg B_1 \vee A_1 \\
c_2: & \neg A_2 \vee (v_1 - v_5 \leq 1) & \neg A_2 \vee B_2 \\
c_3: & (3v_1 - 2v_2 \leq 3) \vee A_2 & B_3 \vee A_2 \\
c_4: & \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1 & \neg B_4 \vee \neg B_5 \vee \neg A_1 \\
c_5: & A_1 \vee (3v_1 - 2v_2 \leq 3) & A_1 \vee B_3 \\
c_6: & (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1 & B_6 \vee B_7 \vee \neg A_1 \\
c_7: & A_1 \vee (v_3 = 3v_5 + 4) \vee A_2 & A_1 \vee B_8 \vee A_2
\end{array}
$$



true, *false*

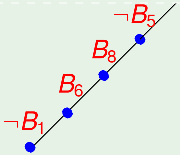
$$
\begin{array}{ll}
\mu^p & = \{\neg B_5, B_8, B_6, \neg B_1\} \\
\mu & = \{\underline{\neg(3v_1 - v_3 \leq 6)}, (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \underline{\neg(2v_2 - v_3 > 2)}\} \\
 & \models_{\mathcal{LRA}} \underbrace{\neg(3v_1 - 2v_2 \leq 3)}_{\neg B_3}
\end{array}
$$

$\Longrightarrow$ propagate $\neg B_3$ [and learn the deduction clause $B_5 \vee B_1 \vee \neg B_3$]

$\mathcal{T}$-propagation: example

	$\varphi =$	$\varphi^p =$
c_1 :	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
c_2 :	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
c_3 :	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
c_4 :	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
c_5 :	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
c_6 :	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
c_7 :	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$

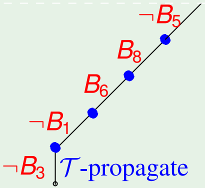
true, *false*



$$\begin{array}{ll} \mu^p & = \{\neg B_5, B_8, B_6, \neg B_1\} \\ \mu & = \{\underline{\neg(3v_1 - v_3 \leq 6)}, (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \underline{\neg(2v_2 - v_3 > 2)}\} \\ & \models_{\mathcal{LRA}} \underbrace{\neg(3v_1 - 2v_2 \leq 3)}_{\neg B_3} \end{array}$$

$\Longrightarrow$ propagate $\neg B_3$ [and learn the deduction clause $B_5 \vee B_1 \vee \neg B_3$]

$\mathcal{T}$-propagation: example



	$\varphi =$	$\varphi^p =$
c_1 :	$\neg(2v_2 - v_3 > 2) \vee A_1$	$\neg B_1 \vee A_1$
c_2 :	$\neg A_2 \vee (v_1 - v_5 \leq 1)$	$\neg A_2 \vee B_2$
c_3 :	$(3v_1 - 2v_2 \leq 3) \vee A_2$	$B_3 \vee A_2$
c_4 :	$\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$	$\neg B_4 \vee \neg B_5 \vee \neg A_1$
c_5 :	$A_1 \vee (3v_1 - 2v_2 \leq 3)$	$A_1 \vee B_3$
c_6 :	$(v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$	$B_6 \vee B_7 \vee \neg A_1$
c_7 :	$A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$	$A_1 \vee B_8 \vee A_2$

true, *false*

$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1\}$$

$$\mu = \{\underline{\neg(3v_1 - v_3 \leq 6)}, (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \underline{\neg(2v_2 - v_3 > 2)}\}$$

$$\models_{\mathcal{LRA}} \underbrace{\neg(3v_1 - 2v_2 \leq 3)}_{\neg B_3}$$

$\Longrightarrow$ propagate $\neg B_3$ [and learn the deduction clause $B_5 \vee B_1 \vee \neg B_3$]

Pure-literal filtering [80, 3, 17]

Property

If we have non-Boolean $\mathcal{T}$-atoms occurring only positively [negatively] in the original formula φ (learned clauses are not considered), we can drop every negative [positive] occurrence of them from the assignment to be checked by $\mathcal{T}$*-solver* (and from the $\mathcal{T}$-deducible ones).

- increases the chances of finding a model
- reduces the effort for the $\mathcal{T}$*-solver*
- eliminates unnecessary "nasty" negated literals
  (e.g. negative equalities like $\neg(3v_1 - 9v_2 = 3)$ in $\mathcal{LIA}$ force splitting: $(3v_1 - 9v_2 > 3) \vee (3v_1 - 9v_2 < 3)$).
- may weaken the effect of early pruning.

Pure-literal filtering [80, 3, 17]

Property

If we have non-Boolean $\mathcal{T}$-atoms occurring only positively [negatively] in the original formula φ (learned clauses are not considered), we can drop every negative [positive] occurrence of them from the assignment to be checked by $\mathcal{T}$*-solver* (and from the $\mathcal{T}$-deducible ones).

- increases the chances of finding a model
- reduces the effort for the $\mathcal{T}$*-solver*
- eliminates unnecessary “nasty” negated literals
  (e.g. negative equalities like $\neg(3v_1 - 9v_2 = 3)$ in $\mathcal{LIA}$ force splitting:
  $(3v_1 - 9v_2 > 3) \vee (3v_1 - 9v_2 < 3)$).
- may weaken the effect of early pruning.

Pure literal filtering: example

$$
\begin{array}{rl}
\varphi = & \{\neg(2v_2 - v_3 > 2) \vee A_1\} \wedge \\
& \{\neg A_2 \vee (2v_1 - 4v_5 > 3)\} \wedge \\
& \{(3v_1 - 2v_2 \le 3) \vee A_2\} \wedge \\
& \{\neg(2v_3 + v_4 \ge 5) \vee \neg(3v_1 - v_3 \le -2) \vee \neg A_1\} \wedge \\
& \{A_1 \vee (3v_1 - 2v_2 \le 3)\} \wedge \\
& \{(v_1 - v_5 \le 1) \vee (v_5 = 5 - 3v_4) \vee \neg A_1\} \wedge \\
& \{A_1 \vee (v_3 = 3v_5 + 4) \vee A_2\} \wedge \\
& \{(2v_2 - v_3 > 2) \vee \neg(3v_1 - 2v_2 \le 3) \vee (3v_1 - v_3 \le -2)\} \quad \textit{learned}
\end{array}
$$

$$\mu' = \{\neg(2v_2 - v_3 > 2), \neg A_2, (3v_1 - 2v_2 \le 3), \neg A_1, (v_3 = 3v_5 + 4), (3v_1 - v_3 \le -2)\}.$$

$\Longrightarrow$ Sat: $v_1 = v_2 = v_3 = 0$, $v_5 = -4/3$ is a solution

Note

- $(3v_1 - v_3 \le -2)$ "filtered out" from μ' because it occurs only negatively in the original formula φ
- $\mu' \cup \{(3v_1 - v_3 \le -2)\}$ is $\mathcal{LRA}$-unsatisfiable

Pure literal filtering: example

$$
\begin{array}{rl}
\varphi = & \{\neg(2v_2 - v_3 > 2) \vee A_1\} \wedge \\
& \{\neg A_2 \vee (2v_1 - 4v_5 > 3)\} \wedge \\
& \{(3v_1 - 2v_2 \leq 3) \vee A_2\} \wedge \\
& \{\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq -2) \vee \neg A_1\} \wedge \\
& \{A_1 \vee (3v_1 - 2v_2 \leq 3)\} \wedge \\
& \{(v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg A_1\} \wedge \\
& \{A_1 \vee (v_3 = 3v_5 + 4) \vee A_2\} \wedge \\
& \{(2v_2 - v_3 > 2) \vee \neg(3v_1 - 2v_2 \leq 3) \vee (3v_1 - v_3 \leq -2)\} \quad \textit{learned}
\end{array}
$$

$$\mu' = \{\neg(2v_2 - v_3 > 2), \neg A_2, (3v_1 - 2v_2 \leq 3), \neg A_1, (v_3 = 3v_5 + 4), (3v_1 - v_3 \leq -2)\}.$$

$\Longrightarrow$ Sat: $v_1 = v_2 = v_3 = 0, v_5 = -4/3$ is a solution

Note

- $(3v_1 - v_3 \leq -2)$ “filtered out” from μ' because it occurs only negatively in the original formula φ
- $\mu' \cup \{(3v_1 - v_3 \leq -2)\}$ is $\mathcal{LRA}$-unsatisfiable

Pure literal filtering: example

$$\begin{aligned}\varphi = \;& \{\neg(2v_2 - v_3 > 2) \vee A_1\} \wedge \\ & \{\neg A_2 \vee (2v_1 - 4v_5 > 3)\} \wedge \\ & \{(3v_1 - 2v_2 \leq 3) \vee A_2\} \wedge \\ & \{\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq -2) \vee \neg A_1\} \wedge \\ & \{A_1 \vee (3v_1 - 2v_2 \leq 3)\} \wedge \\ & \{(v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg A_1\} \wedge \\ & \{A_1 \vee (v_3 = 3v_5 + 4) \vee A_2\} \wedge \\ & \{(2v_2 - v_3 > 2) \vee \neg(3v_1 - 2v_2 \leq 3) \vee (3v_1 - v_3 \leq -2)\} \quad \textit{learned}\end{aligned}$$

$$\mu' = \{\neg(2v_2 - v_3 > 2), \neg A_2, (3v_1 - 2v_2 \leq 3), \neg A_1, (v_3 = 3v_5 + 4), (3v_1 - v_3 \leq -2)\}.$$

$\Longrightarrow$ Sat: $v_1 = v_2 = v_3 = 0, v_5 = -4/3$ is a solution

Note

- $(3v_1 - v_3 \leq -2)$ "filtered out" from μ' because it occurs only negatively in the original formula φ
- $\mu' \cup \{(3v_1 - v_3 \leq -2)\}$ is $\mathcal{LRA}$-unsatisfiable

Preprocessing atoms [45, 50, 4]

Source of inefficiency:

Semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]

$\Longrightarrow$ they may be assigned different [resp. identical] truth values.

$\Longrightarrow$ lots of redundant unsatisfiable assignment generated

Solution

Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.

Preprocessing atoms [45, 50, 4]

Source of inefficiency:

Semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]

$\Longrightarrow$ they may be assigned different [resp. identical] truth values.

$\Longrightarrow$ lots of redundant unsatisfiable assignment generated

Solution

Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.

Preprocessing atoms [45, 50, 4]

Source of inefficiency:

Semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]

$\Longrightarrow$ they may be assigned different [resp. identical] truth values.

$\Longrightarrow$ lots of redundant unsatisfiable assignment generated

Solution

Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.

Preprocessing atoms [45, 50, 4]

Source of inefficiency:

Semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]

$\Longrightarrow$ they may be assigned different [resp. identical] truth values.

$\Longrightarrow$ lots of redundant unsatisfiable assignment generated

Solution

Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.

Preprocessing atoms (cont.)

- Sorting: $(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \Longrightarrow (v_1 + v_2 - v_3 \leq 1))$;
- Rewriting dual operators:
  $(v_1 < v_2), (v_1 \geq v_2) \Longrightarrow (v_1 < v_2), \neg(v_1 < v_2)$
- Exploiting associativity:
  $(v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1) \Longrightarrow (v_1 + v_2 + v_3 = 1)$;
- Factoring $(v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \Longrightarrow (0.25v_1 + 0.5v_2 \leq 1.0)$;
- Exploiting properties of $\mathcal{T}$:
  $(v_1 \leq 3), (v_1 < 4) \Longrightarrow (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Preprocessing atoms (cont.)

- Sorting: $(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \Longrightarrow (v_1 + v_2 - v_3 \leq 1))$;
- Rewriting dual operators:
  $(v_1 < v_2), (v_1 \geq v_2) \Longrightarrow (v_1 < v_2), \neg(v_1 < v_2)$
- Exploiting associativity:
  $(v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1) \Longrightarrow (v_1 + v_2 + v_3 = 1)$;
- Factoring $(v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \Longrightarrow (0.25v_1 + 0.5v_2 \leq 1.0)$;
- Exploiting properties of $\mathcal{T}$:
  $(v_1 \leq 3), (v_1 < 4) \Longrightarrow (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Preprocessing atoms (cont.)

- Sorting: $(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \Longrightarrow (v_1 + v_2 - v_3 \leq 1))$;
- Rewriting dual operators:
  $(v_1 < v_2), (v_1 \geq v_2) \Longrightarrow (v_1 < v_2), \neg(v_1 < v_2)$
- Exploiting associativity:
  $(v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1) \Longrightarrow (v_1 + v_2 + v_3 = 1)$;
- Factoring $(v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \Longrightarrow (0.25v_1 + 0.5v_2 \leq 1.0)$;
- Exploiting properties of $\mathcal{T}$:
  $(v_1 \leq 3), (v_1 < 4) \Longrightarrow (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Preprocessing atoms (cont.)

- Sorting: $(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \Longrightarrow (v_1 + v_2 - v_3 \leq 1))$;
- Rewriting dual operators:
  $(v_1 < v_2), (v_1 \geq v_2) \Longrightarrow (v_1 < v_2), \neg(v_1 < v_2)$
- Exploiting associativity:
  $(v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1) \Longrightarrow (v_1 + v_2 + v_3 = 1)$;
- Factoring $(v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \Longrightarrow (0.25v_1 + 0.5v_2 \leq 1.0)$;
- Exploiting properties of $\mathcal{T}$:
  $(v_1 \leq 3), (v_1 < 4) \Longrightarrow (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Preprocessing atoms (cont.)

- Sorting: $(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \Longrightarrow (v_1 + v_2 - v_3 \leq 1))$;
- Rewriting dual operators:
  $(v_1 < v_2), (v_1 \geq v_2) \Longrightarrow (v_1 < v_2), \neg(v_1 < v_2)$
- Exploiting associativity:
  $(v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1) \Longrightarrow (v_1 + v_2 + v_3 = 1)$;
- Factoring $(v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \Longrightarrow (0.25v_1 + 0.5v_2 \leq 1.0)$;
- Exploiting properties of $\mathcal{T}$:
  $(v_1 \leq 3), (v_1 < 4) \Longrightarrow (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Static Learning [2, 4]

- Often possible to quickly detect a priori short and "obviously unsatisfiable" pairs or triplets of literals occurring in φ.
  - mutual exclusion $\{x = 0, x = 1\}$,
  - congruence $\{(x_1 = y_1), (x_2 = y_2), \neg(f(x_1, x_2) = f(y_1, y_2))\}$,
  - transitivity $\{(x - y = 2), (y - z \leq 4), \neg(x - z \leq 7)\}$,
  - substitution $\{(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)\}$
  - ...
- Preprocessing step: detect these literals and add blocking clauses to the input formula: (e.g., $\neg(x = 0) \vee \neg(x = 1)$)

$\Longrightarrow$ No assignment including one such group of literals is ever generated: as soon as all but one literals are assigned, the remaining one is immediately assigned false by unit-propagation.

Static Learning [2, 4]

- Often possible to quickly detect a priori short and “obviously unsatisfiable” pairs or triplets of literals occurring in φ.
    - mutual exclusion $\{x = 0, x = 1\}$,
    - congruence $\{(x_1 = y_1), (x_2 = y_2), \neg(f(x_1, x_2) = f(y_1, y_2))\}$,
    - transitivity $\{(x - y = 2), (y - z \leq 4), \neg(x - z \leq 7)\}$,
    - substitution $\{(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)\}$
    - ...
- Preprocessing step: detect these literals and add blocking clauses to the input formula: (e.g., $\neg(x = 0) \vee \neg(x = 1)$)

$\Longrightarrow$ No assignment including one such group of literals is ever generated: as soon as all but one literals are assigned, the remaining one is immediately assigned false by unit-propagation.

Static Learning [2, 4]

- Often possible to quickly detect a priori short and "obviously unsatisfiable" pairs or triplets of literals occurring in φ.
  - mutual exclusion $\{x = 0, x = 1\}$,
  - congruence $\{(x_1 = y_1), (x_2 = y_2), \neg(f(x_1, x_2) = f(y_1, y_2))\}$,
  - transitivity $\{(x - y = 2), (y - z \leq 4), \neg(x - z \leq 7)\}$,
  - substitution $\{(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)\}$
  - ...
- Preprocessing step: detect these literals and add blocking clauses to the input formula: (e.g., $\neg(x = 0) \vee \neg(x = 1)$)

$\Longrightarrow$ No assignment including one such group of literals is ever generated: as soon as all but one literals are assigned, the remaining one is immediately assigned false by unit-propagation.

Other optimization techniques

- $\mathcal{T}$-deduced-literal filtering
- Ghost-literal filtering
- $\mathcal{T}$*-solver* layering
- $\mathcal{T}$*-solver* clustering
- ...

(see [70, 10] for an overview)

Other SAT-solving techniques for SMT?

Frequently-asked question:

Are CDCL SAT solvers the only suitable Boolean Engines for SMT?

Some previous attempts:

- Ordered Binary Decision Diagrams (OBDDs) [81, 60, 1]
- Stochastic Local Search [49]

CDCL based currently much more efficient.

Other SAT-solving techniques for SMT?

Frequently-asked question:

Are CDCL SAT solvers the only suitable Boolean Engines for SMT?

Some previous attempts:

- Ordered Binary Decision Diagrams (OBDDs) [81, 60, 1]
- Stochastic Local Search [49]

CDCL based currently much more efficient.

SMT formulas = “partially-invisible” SAT formulas

An SMT problem φ from the perspective of a SAT solver:

- a “partially-invisible” Boolean CNF formula $\varphi^p \wedge \tau^p$:
  - φ^p: the Boolean abstraction of the input formula φ
  - τ^p: (the Boolean abstraction of) the set τ of all $\mathcal{T}$-lemmas on atoms in φ.

  φ $\mathcal{T}$-satisfiable iff $\varphi^p \wedge \tau^p$ satisfiable.
- the SAT solver:
  - “sees” only φ^p
  - finds μ^ps.t. $\mu^p \models \varphi^p$
  - cannot state if $\mu^p \models \tau^p$
  - invokes $\mathcal{T}$*-solver* on μ^p
- the $\mathcal{T}$*-solver*:
  - “sees” τ^p
  - checks if $\mu^p \models \tau^p$:
    - if yes, returns SAT
    - if no, returns UNSAT and some falsified clauses $c_1^p, ..., c_k^p \in \tau^p$

SMT formulas = “partially-invisible” SAT formulas

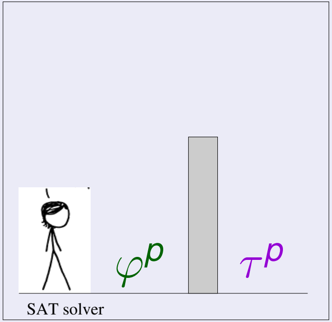
An SMT problem φ from the perspective of a SAT solver:

- a “partially-invisible” Boolean CNF formula $\varphi^p \wedge \tau^p$:
  - φ^p: the Boolean abstraction of the input formula φ
  - τ^p: (the Boolean abstraction of) the set τ of all $\mathcal{T}$-lemmas on atoms in φ.

  φ $\mathcal{T}$-satisfiable iff $\varphi^p \wedge \tau^p$ satisfiable.

- the SAT solver:
  - “sees” only φ^p
  - finds μ^ps.t. $\mu^p \models \varphi^p$
  - cannot state if $\mu^p \models \tau^p$
  - invokes $\mathcal{T}$-*solver* on μ^p
- the $\mathcal{T}$-*solver*:
  - “sees” τ^p
  - checks if $\mu^p \models \tau^p$:
    - if yes, returns SAT
    - if no, returns UNSAT and some falsified clauses $c_1^p, ..., c_k^p \in \tau^p$

© www.xkcd.com



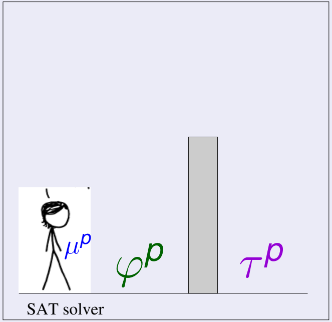
SMT formulas = “partially-invisible” SAT formulas

An SMT problem φ from the perspective of a SAT solver:

- a “partially-invisible” Boolean CNF formula $\varphi^p \wedge \tau^p$:
  - φ^p: the Boolean abstraction of the input formula φ
  - τ^p: (the Boolean abstraction of) the set τ of all $\mathcal{T}$-lemmas on atoms in φ.

  φ $\mathcal{T}$-satisfiable iff $\varphi^p \wedge \tau^p$ satisfiable.
- the SAT solver:
  - “sees” only φ^p
  - finds μ^ps.t. $\mu^p \models \varphi^p$
  - cannot state if $\mu^p \models \tau^p$
  - invokes $\mathcal{T}$-*solver* on μ^p
- the $\mathcal{T}$-*solver*:
  - “sees” τ^p
  - checks if $\mu^p \models \tau^p$:
    - if yes, returns SAT
    - if no, returns UNSAT and some falsified clauses $c_1^p, ..., c_k^p \in \tau^p$

© www.xkcd.com



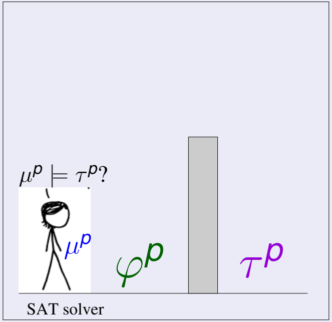
SMT formulas = "partially-invisible" SAT formulas

An SMT problem φ from the perspective of a SAT solver:

- a "partially-invisible" Boolean CNF formula $\varphi^p \wedge \tau^p$:
  - φ^p: the Boolean abstraction of the input formula φ
  - τ^p: (the Boolean abstraction of) the set τ of all $\mathcal{T}$-lemmas on atoms in φ.

  φ $\mathcal{T}$-satisfiable iff $\varphi^p \wedge \tau^p$ satisfiable.
- the SAT solver:
  - "sees" only φ^p
  - finds μ^ps.t. $\mu^p \models \varphi^p$
  - cannot state if $\mu^p \models \tau^p$
  - invokes $\mathcal{T}$-*solver* on μ^p
- the $\mathcal{T}$-*solver*:
  - "sees" τ^p
  - checks if $\mu^p \models \tau^p$:
    - if yes, returns SAT
    - if no, returns UNSAT and some falsified clauses $c_1^p, ..., c_k^p \in \tau^p$

© www.xkcd.com



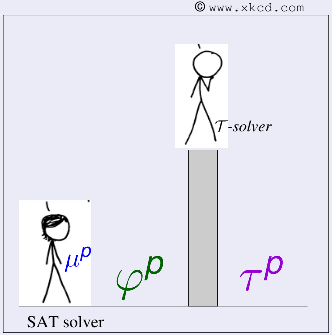
SMT formulas = "partially-invisible" SAT formulas

An SMT problem φ from the perspective of a SAT solver:

- a "partially-invisible" Boolean CNF formula $\varphi^p \wedge \tau^p$:
  - φ^p: the Boolean abstraction of the input formula φ
  - τ^p: (the Boolean abstraction of) the set τ of all $\mathcal{T}$-lemmas on atoms in φ.

φ $\mathcal{T}$-satisfiable iff $\varphi^p \wedge \tau^p$ satisfiable.

- the SAT solver:
  - "sees" only φ^p
  - finds μ^ps.t. $\mu^p \models \varphi^p$
  - cannot state if $\mu^p \models \tau^p$
  - invokes $\mathcal{T}$-*solver* on μ^p
- the $\mathcal{T}$-*solver*:
  - "sees" τ^p
  - checks if $\mu^p \models \tau^p$:
    - if yes, returns SAT
    - if no, returns UNSAT and some falsified clauses $c_1^p, \ldots, c_k^p \in \tau^p$



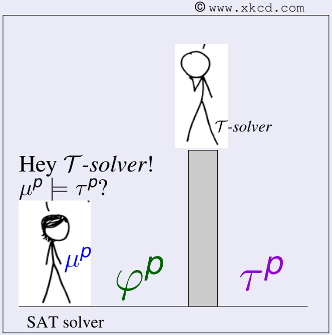
SMT formulas = “partially-invisible” SAT formulas

An SMT problem φ from the perspective of a SAT solver:

- a “partially-invisible” Boolean CNF formula $\varphi^p \wedge \tau^p$:
  - φ^p: the Boolean abstraction of the input formula φ
  - τ^p: (the Boolean abstraction of) the set τ of all $\mathcal{T}$-lemmas on atoms in φ.

φ $\mathcal{T}$-satisfiable iff $\varphi^p \wedge \tau^p$ satisfiable.

- the SAT solver:
  - “sees” only φ^p
  - finds μ^p s.t. $\mu^p \models \varphi^p$
  - cannot state if $\mu^p \models \tau^p$
  - invokes *$\mathcal{T}$-solver* on μ^p
- the *$\mathcal{T}$-solver*:
  - “sees” τ^p
  - checks if $\mu^p \models \tau^p$:
    - if yes, returns SAT
    - if no, returns UNSAT and some falsified clauses $c_1^p, ..., c_k^p \in \tau^p$

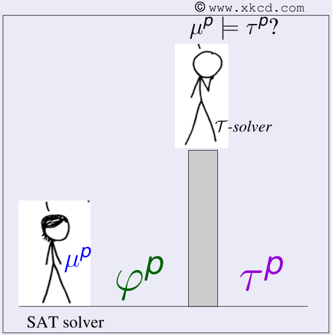

© www.xkcd.com

SMT formulas = "partially-invisible" SAT formulas

An SMT problem φ from the perspective of a SAT solver:

- a "partially-invisible" Boolean CNF formula $\varphi^p \wedge \tau^p$:
  - φ^p: the Boolean abstraction of the input formula φ
  - τ^p: (the Boolean abstraction of) the set τ of all $\mathcal{T}$-lemmas on atoms in φ.

  φ $\mathcal{T}$-satisfiable iff $\varphi^p \wedge \tau^p$ satisfiable.
- the SAT solver:
  - "sees" only φ^p
  - finds μ^ps.t. $\mu^p \models \varphi^p$
  - cannot state if $\mu^p \models \tau^p$
  - invokes $\mathcal{T}$*-solver* on μ^p
- the $\mathcal{T}$*-solver*:
  - "sees" τ^p
  - checks if $\mu^p \models \tau^p$:
    - if yes, returns SAT
    - if no, returns UNSAT and some falsified clauses $c_1^p, ..., c_k^p \in \tau^p$



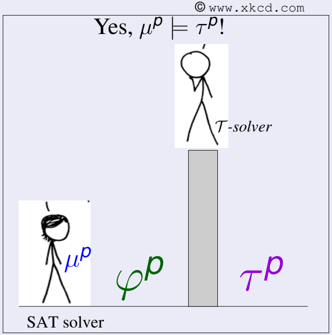
SMT formulas = "partially-invisible" SAT formulas

An SMT problem φ from the perspective of a SAT solver:

- a "partially-invisible" Boolean CNF formula $\varphi^p \wedge \tau^p$:
  - φ^p: the Boolean abstraction of the input formula φ
  - τ^p: (the Boolean abstraction of) the set τ of all $\mathcal{T}$-lemmas on atoms in φ.

φ $\mathcal{T}$-satisfiable iff $\varphi^p \wedge \tau^p$ satisfiable.

- the SAT solver:
  - "sees" only φ^p
  - finds μ^ps.t. $\mu^p \models \varphi^p$
  - cannot state if $\mu^p \models \tau^p$
  - invokes $\mathcal{T}$-*solver* on μ^p
- the $\mathcal{T}$-*solver*:
  - "sees" τ^p
  - checks if $\mu^p \models \tau^p$:
    - if yes, returns SAT
    - if no, returns UNSAT and some falsified clauses $c_1^p, \dots, c_k^p \in \tau^p$



SMT formulas = "partially-invisible" SAT formulas

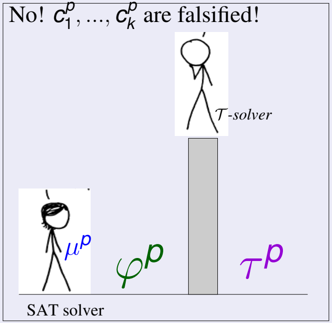
An SMT problem φ from the perspective of a SAT solver:

- a "partially-invisible" Boolean CNF formula $\varphi^p \wedge \tau^p$:
  - φ^p: the Boolean abstraction of the input formula φ
  - τ^p: (the Boolean abstraction of) the set τ of all $\mathcal{T}$-lemmas on atoms in φ.

φ $\mathcal{T}$-satisfiable iff $\varphi^p \wedge \tau^p$ satisfiable.

- the SAT solver:
  - "sees" only φ^p
  - finds μ^ps.t. $\mu^p \models \varphi^p$
  - cannot state if $\mu^p \models \tau^p$
  - invokes $\mathcal{T}$-*solver* on μ^p
- the $\mathcal{T}$-*solver*:
  - "sees" τ^p
  - checks if $\mu^p \models \tau^p$:
    - if yes, returns SAT
    - if no, returns UNSAT and some falsified clauses $c_1^p, ..., c_k^p \in \tau^p$

© www.xkcd.com



Example

φ :
c_1 : $\{A_1\}$
c_2 : $\{\neg A_1 \vee (x - z > 4)\}$
c_3 : $\{\neg A_3 \vee A_1 \vee (y \geq 1)\}$
c_4 : $\{\neg A_2 \vee \neg(x - z > 4) \vee \neg A_1\}$
c_5 : $\{(x - y \leq 3) \vee \neg A_4 \vee A_5\}$
c_6 : $\{\neg(y - z \leq 1) \vee (x + y = 1) \vee \neg A_5\}$
c_7 : $\{A_3 \vee \neg(x + y = 0) \vee A_2\}$
c_8 : $\{\neg A_3 \vee (z + y = 2)\}$
$\mathcal{T}$: (all possible $\mathcal{T}$-lemmas on the $\mathcal{T}$-atoms of φ)
c_9 : $\{\neg(x + y = 0) \vee \neg(x + y = 1)\}$
c_{10} : $\{\neg(x - z > 4) \vee \neg(x - y \leq 3) \vee \neg(y - z \leq 1)\}$
c_{11} : $\{(x - z > 4) \vee (x - y \leq 3) \vee (y - z \leq 1)\}$
c_{12} : $\{\neg(x - z > 4) \vee \neg(x + y = 1) \vee \neg(z + y = 2)\}$
c_{13} : $\{\neg(x - z > 4) \vee \neg(x + y = 0) \vee \neg(z + y = 2)\}$
... ...

φ^p :
c_1 : $\{A_1\}$
c_2 : $\{\neg A_1 \vee B_1\}$
c_3 : $\{\neg A_3 \vee A_1 \vee B_2\}$
c_4 : $\{\neg A_2 \vee \neg B_1 \vee \neg A_1\}$
c_5 : $\{B_3 \vee \neg A_4 \vee A_5\}$
c_6 : $\{\neg B_4 \vee B_5 \vee \neg A_5\}$
c_7 : $\{A_3 \vee \neg B_6 \vee A_2\}$
c_8 : $\{\neg A_3 \vee B_7\}$
τ^p :
c_9 : $\{\neg B_6 \vee \neg B_5\}$
c_{10} : $\{\neg B_1 \vee \neg B_3 \vee \neg B_4\}$
c_{11} : $\{B_1 \vee B_3 \vee B_4\}$
c_{12} : $\{\neg B_1 \vee \neg B_5 \vee \neg B_7\}$
c_{13} : $\{\neg B_1 \vee \neg B_6 \vee \neg B_7\}$
... ...

μ_1^p : $\{A_1, B_1, \neg A_2, A_3, \neg A_4, \neg A_5, \neg B_6, B_5, B_3, B_4, B_7, \neg B_2\}$
μ_1 : $\{\underline{(x - z > 4)}, \neg(x + y = 0), \underline{(x + y = 1)}, \underline{(x - y \leq 3)}, \underline{(y - z \leq 1)},$
$\underline{(z + y = 2)}, \neg(y \geq 1)\}$

satisfies φ^p, but violates both c_{10} and c_{12} in τ^p.

Example

φ :

c_1 : $\{A_1\}$

c_2 : $\{\neg A_1 \vee (x - z > 4)\}$

c_3 : $\{\neg A_3 \vee A_1 \vee (y \geq 1)\}$

c_4 : $\{\neg A_2 \vee \neg(x - z > 4) \vee \neg A_1\}$

c_5 : $\{(x - y \leq 3) \vee \neg A_4 \vee A_5\}$

c_6 : $\{\neg(y - z \leq 1) \vee (x + y = 1) \vee \neg A_5\}$

c_7 : $\{A_3 \vee \neg(x + y = 0) \vee A_2\}$

c_8 : $\{\neg A_3 \vee (z + y = 2)\}$

$\mathcal{T}$: (all possible $\mathcal{T}$-lemmas on the $\mathcal{T}$-atoms of φ)

c_9 : $\{\neg(x + y = 0) \vee \neg(x + y = 1)\}$

c_{10} : $\{\neg(x - z > 4) \vee \neg(x - y \leq 3) \vee \neg(y - z \leq 1)\}$

c_{11} : $\{(x - z > 4) \vee (x - y \leq 3) \vee (y - z \leq 1)\}$

c_{12} : $\{\neg(x - z > 4) \vee \neg(x + y = 1) \vee \neg(z + y = 2)\}$

c_{13} : $\{\neg(x - z > 4) \vee \neg(x + y = 0) \vee \neg(z + y = 2)\}$

... ...

φ^p :

c_1 : $\{A_1\}$

c_2 : $\{\neg A_1 \vee B_1\}$

c_3 : $\{\neg A_3 \vee A_1 \vee B_2\}$

c_4 : $\{\neg A_2 \vee \neg B_1 \vee \neg A_1\}$

c_5 : $\{B_3 \vee \neg A_4 \vee A_5\}$

c_6 : $\{\neg B_4 \vee B_5 \vee \neg A_5\}$

c_7 : $\{A_3 \vee \neg B_6 \vee A_2\}$

c_8 : $\{\neg A_3 \vee B_7\}$

τ^p :

c_9 : $\{\neg B_6 \vee \neg B_5\}$

c_{10} : $\{\neg B_1 \vee \neg B_3 \vee \neg B_4\}$

c_{11} : $\{B_1 \vee B_3 \vee B_4\}$

c_{12} : $\{\neg B_1 \vee \neg B_5 \vee \neg B_7\}$

c_{13} : $\{\neg B_1 \vee \neg B_6 \vee \neg B_7\}$

... ...

μ_1^p : $\{A_1, B_1, \neg A_2, A_3, \neg A_4, \neg A_5, \neg B_6, B_5, B_3, B_4, B_7, \neg B_2\}$

μ_1 : $\{\underline{\underline{(x - z > 4)}}, \neg(x + y = 0), \underline{(x + y = 1)}, \underline{(x - y \leq 3)}, \underline{(y - z \leq 1)},$

$\underline{\underline{(z + y = 2)}}, \neg(y \geq 1)\}$

satisfies φ^p, but violates both c_{10} and c_{12} in τ^p.

Exercise

Consider the following formula in the theory $\mathcal{EUF}$.

$$
\begin{aligned}
\varphi = \quad & \{(f(x) = f(f(y))) \ \vee A_2\} \quad \wedge \\
& \{\neg(h(x, f(y)) = h(g(x), y)) \ \vee \underline{\neg(h(x, g(z) = h(f(x), y)))} \ \vee \neg A_1\} \ \wedge \\
& \{A_1 \vee (h(x, y) = h(y, x))\} \quad \wedge \\
& \{\underline{(x = f(x))} \ \vee A_3 \vee \neg A_1\} \ \wedge \\
& \{\underline{\neg(w(x) = g(f(y)))} \ \vee A_1\} \wedge \\
& \{\underline{\neg A_2} \vee (w(g(x)) = w(f(x)))\} \wedge \\
& \{A_1 \vee \underline{(y = g(z))} \ \vee A_2\}
\end{aligned}
$$

and consider the partial truth assignment μ given by the underlined literals above:

$$\{\neg(w(x) = g(f(y))), \neg A_2, \neg(h(x, g(z) = h(f(x), y))), (x = f(x)), (y = g(z))\}.$$

1. Does (the Boolean abstraction of) μ propositionally satisfy (the Boolean abstraction of) φ?
2. Is μ satisfiable in $\mathcal{EUF}$?
    1. If no, find a minimal conflict set for μ and the corresponding conflict clause C.
    2. If yes, show one unassigned literal which can be deduced from μ, and show the corresponding deduction clause C.

Outline

1. Introduction
   - Basics on First-order Logic
   - What is a Theory?
   - Satisfiability Modulo Theories
   - Motivations and Goals of SMT

2. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for Theories of Interest (hints)
   - SMT for Combinations of Theories

3. Beyond Solving: Advanced SMT Functionalities
   - Proofs and Unsatisfiable Cores
   - All-SMT & Predicate Abstraction (hints)
   - SMT with Optimization (Optimization Modulo Theories)

Summary: desirable properties for $\mathcal{T}$*-solver*

- Correctness & Completeness: be correct & complete
- Time efficiency: be fast
- Incrementality & backtrackability: $\mathcal{T}$*-solver*$(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}$*-solver*(μ_1)
- Diagnosis capabilities: $\mathcal{T}$*-solver* able to produce conflict sets
- Deduction capabilities: $\mathcal{T}$*-solver* able to deduce assignments

$\mathcal{T}$-solvers for Equality and Uninterpreted Functions ($\mathcal{EUF}$)

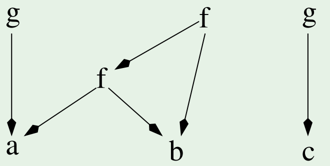
- Typically used as a “core” $\mathcal{T}$*-solver*
- $\mathcal{EUF}$ polynomial: $O(n \cdot log(n))$
- Fully incremental and backtrackable (stack-based)
- Uses a congruence closure data structures (E-Graphs) [39, 64, 34],
  - based on the Union-Find data-structure for equivalence classes
- Supports efficient $\mathcal{T}$-propagation
  - Exhaustive for positive equalities
  - Incomplete for disequalities
- Supports Lazy explanations and conflict generation
  - However, minimality not guaranteed
- Supports efficient extensions
  (e.g., Integer offsets, Bit-vector slicing and concatenation)

$\mathcal{T}$-solvers for $\mathcal{EUF}$: Example

Idea (sketch), based on E-Graphs:
given the set of terms occurring in the formula represented as nodes in a DAG (aka term bank):

- if $(t = s)$, then merge the eq. classes of t and s
  - e.g. use the union-find data structure
- if $\forall i \in 1...k$, t_i and s_i pairwise belong to the same eq. classes, then merge the eq. classes of $f(t_1, ..., t_k)$ and $f(s_1, ..., s_k)$
- if $(t \neq s)$ and t and s belong to the same eq. class, then conflict

E-Graph:



$$f(a, b) = a$$
$$f(f(a, b), b) = c$$
$$g(a) \neq g(c)$$

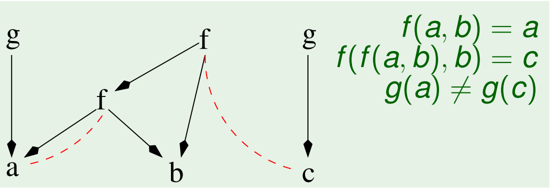
Example borrowed from [39].

$\mathcal{T}$-solvers for $\mathcal{EUF}$: Example

Idea (sketch), based on E-Graphs:
given the set of terms occurring in the formula represented as nodes in a DAG (aka term bank):

- if $(t = s)$, then merge the eq. classes of t and s
  - e.g. use the union-find data structure
- if $\forall i \in 1...k$, t_i and s_i pairwise belong to the same eq. classes, then merge the eq. classes of $f(t_1, ..., t_k)$ and $f(s_1, ..., s_k)$
- if $(t \neq s)$ and t and s belong to the same eq. class, then conflict

E-Graph:



$$f(a, b) = a$$
$$f(f(a, b), b) = c$$
$$g(a) \neq g(c)$$

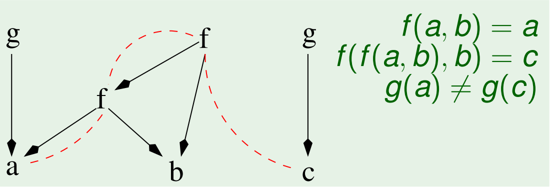
Example borrowed from [39].

$\mathcal{T}$-solvers for $\mathcal{EUF}$: Example

Idea (sketch), based on E-Graphs:
given the set of terms occurring in the formula represented as nodes in a DAG (aka term bank):

- if $(t = s)$, then merge the eq. classes of t and s
  - e.g. use the union-find data structure
- if $\forall i \in 1...k$, t_i and s_i pairwise belong to the same eq. classes, then merge the eq. classes of $f(t_1, ..., t_k)$ and $f(s_1, ..., s_k)$
- if $(t \neq s)$ and t and s belong to the same eq. class, then conflict

E-Graph:



$$f(a,b) = a$$
$$f(f(a,b),b) = c$$
$$g(a) \neq g(c)$$

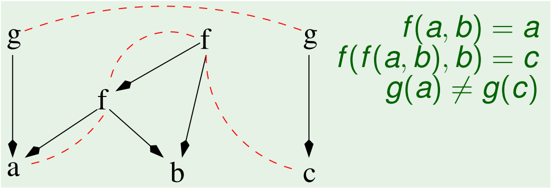
Example borrowed from [39].

$\mathcal{T}$-solvers for $\mathcal{EUF}$: Example

Idea (sketch), based on E-Graphs:
given the set of terms occurring in the formula represented as nodes in a DAG (aka term bank):

- if $(t = s)$, then merge the eq. classes of t and s
  - e.g. use the union-find data structure
- if $\forall i \in 1...k$, t_i and s_i pairwise belong to the same eq. classes, then merge the eq. classes of $f(t_1, ..., t_k)$ and $f(s_1, ..., s_k)$
- if $(t \neq s)$ and t and s belong to the same eq. class, then conflict

E-Graph:



$$f(a,b) = a$$
$$f(f(a,b),b) = c$$
$$g(a) \neq g(c)$$

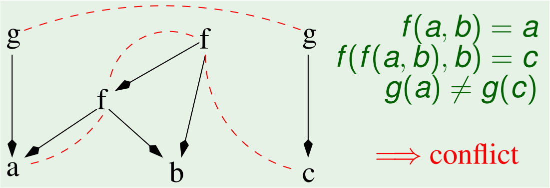
Example borrowed from [39].

$\mathcal{T}$-solvers for $\mathcal{EUF}$: Example

Idea (sketch), based on E-Graphs:
given the set of terms occurring in the formula represented as nodes in a DAG (aka term bank):

- if $(t = s)$, then merge the eq. classes of t and s
  - e.g. use the union-find data structure
- if $\forall i \in 1...k$, t_i and s_i pairwise belong to the same eq. classes, then merge the eq. classes of $f(t_1, ..., t_k)$ and $f(s_1, ..., s_k)$
- if $(t \neq s)$ and t and s belong to the same eq. class, then conflict

E-Graph:



$$f(a,b) = a$$
$$f(f(a,b),b) = c$$
$$g(a) \neq g(c)$$

$\Longrightarrow$ conflict

Example borrowed from [39].

Exercises

For each of the following sets of $\mathcal{EUF}$-literals

1. build the E-graph
2. say if the set ie $\mathcal{EUF}$-satisfiable or not
3. if $\mathcal{EUF}$-unsatisfiable, return a $\mathcal{EUF}$-conflict set

Sets of $\mathcal{EUF}$-literals:

(a) $\{g(a) = b,\ h(b) = a,\ h(a) = c,\ g(h(b)) = c,\ h(h(g(a))) \neq c\}$

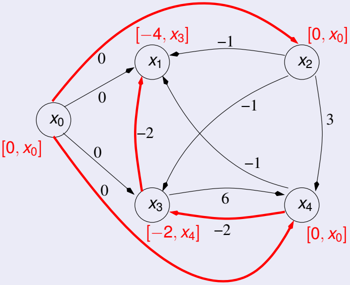
(b) $\{g(a) = b,\ h(b) = a,\ h(a) = c,\ f(h(g(a)), g(h(b))) \neq f(a, b)\}$

(c) [... invent your own sets of $\mathcal{EUF}$-literals]

$\mathcal{T}$-solvers for Difference logic ($\mathcal{DL}$)

- $\mathcal{DL}$ polynomial: $O(\#vars \cdot \#constraints)$
- variants of the Bellman-Ford shortest-path algorithm: a negative cycle reveals a conflict [65, 33]
- Ex:

$$\{(x_1 - x_2 \le -1), (x_1 - x_4 \le -1), (x_1 - x_3 \le -2), (x_2 - x_1 \le 2), (x_3 - x_4 \le -2), (x_3 - x_2 \le -1), (x_4 - x_2 \le 3), (x_4 - x_3 \le 6)\}$$

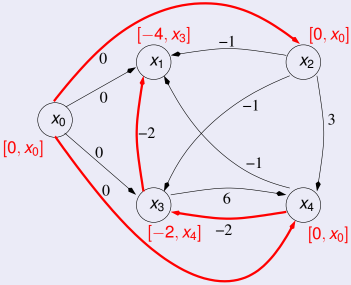


$\Longrightarrow$ Sat

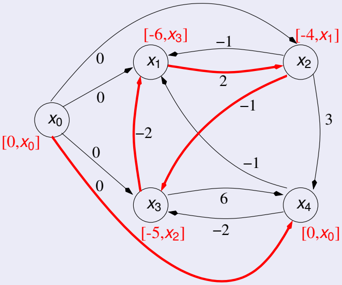
$\mathcal{T}$-solvers for Difference logic ($\mathcal{DL}$)

- $\mathcal{DL}$ polynomial: $O(\#vars \cdot \#constraints)$
- variants of the Bellman-Ford shortest-path algorithm: a negative cycle reveals a conflict [65, 33]
- Ex:

$$\{(x_1 - x_2 \le -1), (x_1 - x_4 \le -1), (x_1 - x_3 \le -2), (x_2 - x_1 \le 2),$$
$$(x_3 - x_4 \le -2), (x_3 - x_2 \le -1), (x_4 - x_2 \le 3), (x_4 - x_3 \le 6)\}$$



$\Longrightarrow$ Sat



$\Longrightarrow$ Unsat

Exercises

For each of the following sets of $\mathcal{DL}$-literals

1. build the minimum-path graph
2. say if the set ie $\mathcal{DL}$-satisfiable or not
3. if $\mathcal{DL}$-unsatisfiable, return a $\mathcal{DL}$-conflict set

Sets of $\mathcal{DL}$-literals:

$$(a)\ \left\{\begin{array}{l}(x_1 - x_3 \leq 5), (x_1 - x_2 \leq -2), (x_1 - x_4 \leq -3), (x_2 - x_4 \leq 0), (x_3 - x_5 \leq -2),\\ (x_4 - x_5 \leq 8), (x_5 - x_1 \leq -4)\end{array}\right\}$$

$$(b)\ \left\{\begin{array}{l}(x_1 - x_2 \leq 3), (x_1 - x_4 \leq -1), (x_1 - x_5 \leq 4), (x_2 - x_3 \leq 0), (x_2 - x_5 \leq -1)\\ (x_3 - x_1 \leq 1), (x_3 - x_2 \leq 2), (x_4 - x_3 \leq 1), (x_5 - x_3 \leq -2), (x_5 - x_4 \leq -5)\end{array}\right\}$$

(c) [... invent your own sets of $\mathcal{DL}$-literals]

$\mathcal{T}$-solvers for Linear arithmetic over the rationals ($\mathcal{LRA}$)

- EX: $\{(s_1 - s_2 \leq 5.2), (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0), \neg(s_1 = s_0)\}$
- $\mathcal{LRA}$ polynomial
- variants of the simplex LP algorithm [41]
- [41] allows for detecting conflict sets & performing $\mathcal{T}$-propagation
- strict inequalities $t < 0$ rewritten as $t + \epsilon \leq 0$, ϵ treated symbolically

$$\begin{array}{c}\mathcal{B}\\ \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_N \end{bmatrix}\end{array} = \begin{bmatrix} \ldots A_{1j} \ldots \\ \vdots \\ A_{i1} \ldots A_{ij} \ldots A_{iM} \\ \vdots \\ \ldots A_{Nj} \ldots \end{bmatrix} \begin{array}{c}\mathcal{N}\\ \begin{bmatrix} x_{N+1} \\ \vdots \\ x_j \\ \vdots \\ x_{N+M} \end{bmatrix}\end{array};$$

Invariant: $\beta(x_j) \in [l_j, u_j]\ \forall x_j \in \mathcal{N}$

Remark: infinite precision arithmetic

In order to avoid incorrect results due to numerical errors and to overflows, all $\mathcal{T}$*-solvers* for $\mathcal{LRA}$, $\mathcal{LIA}$ and their subtheories which are based on numerical algorithms must be implemented on top of infinite-precision-arithmetic software packages.

$\mathcal{T}$-solvers for Linear arithmetic over the integers ($\mathcal{LIA}$)

- EX: $\{(x := x_l + 2^{16} x_h), (x \geq 0), (x \leq 2^{16} - 1)\}$
- $\mathcal{LIA}$ NP-complete
- combination of many techniques: simplex, branch&bound, cutting planes, ... [41, 47]

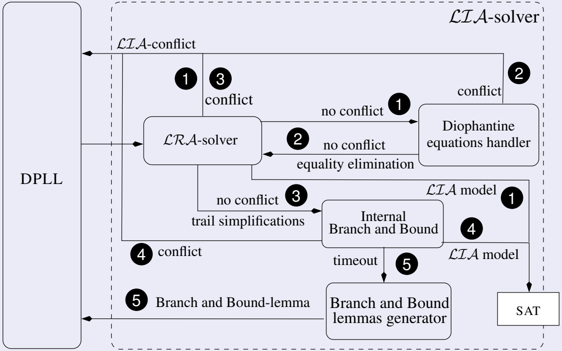


Figure courtesy of A. Griggio [47]

$\mathcal{T}$-solvers for Arrays ($\mathcal{AR}$)

- EX: $(write(A, i, v) = write(B, i, w)) \wedge \neg(v = w)$
- NP-complete
- congruence closure ($\mathcal{EUF}$) plus on-the-fly instantiation of array's axioms:

$$\forall a.\forall i.\forall e.\ (read(write(a, i, e), i) = e), \tag{1}$$

$$\forall a.\forall i.\forall j.\forall e.\ ((i \neq j) \rightarrow read(write(a, i, e), j) = read(a, j)), \tag{2}$$

$$\forall a.\forall b.\ (\forall i.(read(a, i) = read(b, i)) \rightarrow (a = b)). \tag{3}$$

- EX:

$$\begin{array}{ll} Input: & (write(A, i, v) = write(B, i, w)) \wedge \neg(v = w) \\ inst.\ (1): & (read(write(A, i, v), i) = v) \\ & (read(write(B, i, w), i) = w) \\ \models_{\mathcal{EUF}} & (v = w) \\ \models_{Bool} & \bot \end{array}$$

- many strategies discussed in the literature (e.g., [39, 46, 20, 38])

Exercises

For each of the following sets of $\mathcal{AR}$-literals, try to infer $\bot$ by instantiating the $\mathcal{AR}$ axioms and applying $\mathcal{EUF}$-inference and $\mathcal{BOOL}$-inference.
Sets of $\mathcal{AR}$-literals:

(*a*) $\{\ A_1 = Write(A_0, i, x), A_2 = Write(A_0, j, y), (i = j), \neg(k = i), \neg(Read(A_2, k) = Read(A_1, k))\ \}$

(*b*) $\{\ A_1 = Write(A_0, i, x), A_2 = Write(A_0, i, y), \neg(x = y), (A_1 = A_2)\ \}$

(*c*) $\{\ A_1 = Write(A_0, i, x), Read(A_0, i) = Read(A_1, i), \neg(A_1 = A_0)\ \}$

(*d*) [... invent your own sets of $\mathcal{AR}$-literals]

$\mathcal{T}$-solvers for Bit vectors ($\mathcal{BV}$)

Bit vectors ($\mathcal{BV}$)

- EX: $\{(x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[16]}[3:0]), ...\}$
- NP-hard
- involve complex word-level operations: word partition/concat, modulo-2^N arithmetic, shifts, bitwise-operations, multiplexers, ...
- $\mathcal{T}$-solving: combination of rewriting & simplification techniques with either:
  - final encoding into $\mathcal{LIA}$ [19, 22]
  - final encoding into SAT (lazy bit-blasting) [25, 43, 21, 42]

Eager approach

Most solvers use an eager approach for $\mathcal{BV}$ (e.g., [21]):

- Heavy preprocessing, based on rewriting rules
- bit-blasting

$\mathcal{T}$-solvers for Bit vectors ($\mathcal{BV}$)

Bit vectors ($\mathcal{BV}$)

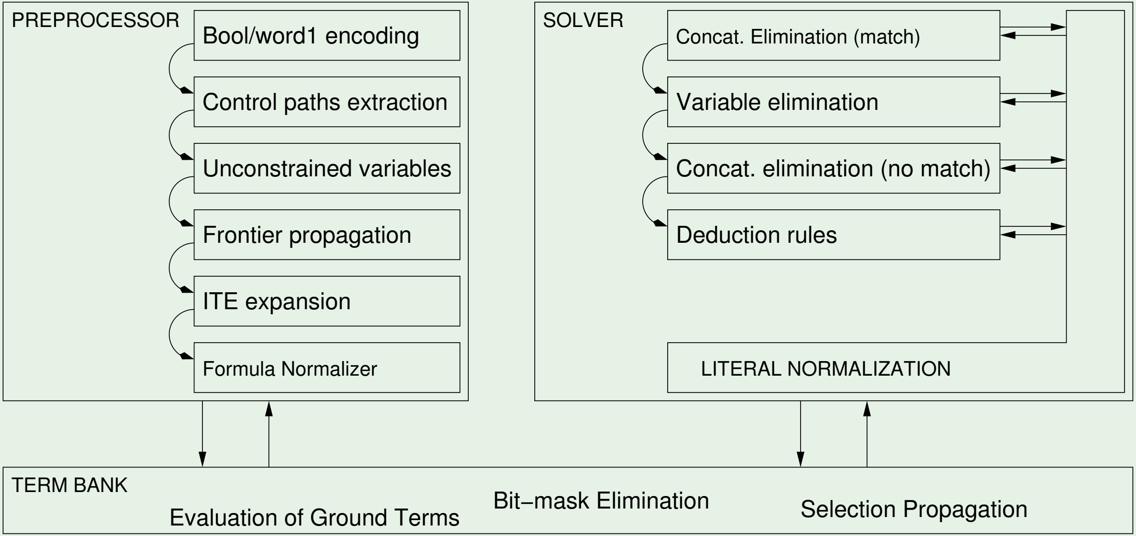
- EX: $\{(x_{[16]}[15:0] = (y_{[16]}[15:8] :: z_{[16]}[7:0]) << w_{[16]}[3:0]), ...\}$
- NP-hard
- involve complex word-level operations: word partition/concat, modulo-2^N arithmetic, shifts, bitwise-operations, multiplexers, ...
- $\mathcal{T}$-solving: combination of rewriting & simplification techniques with either:
  - final encoding into $\mathcal{LIA}$ [19, 22]
  - final encoding into SAT (lazy bit-blasting) [25, 43, 21, 42]

Eager approach

Most solvers use an eager approach for $\mathcal{BV}$ (e.g., [21]):

- Heavy preprocessing, based on rewriting rules
- bit-blasting

$\mathcal{T}$-solvers for Bit vectors ($\mathcal{BV}$) [cont.]



PREPROCESSOR

Bool/word1 encoding

Control paths extraction

Unconstrained variables

Frontier propagation

ITE expansion

Formula Normalizer

SOLVER

Concat. Elimination (match)

Variable elimination

Concat. elimination (no match)

Deduction rules

LITERAL NORMALIZATION

TERM BANK

Evaluation of Ground Terms

Bit–mask Elimination

Selection Propagation

Example borrowed from [22]

$\mathcal{T}$-solvers for Bit vectors ($\mathcal{BV}$) [cont.]

Lazy bit-blasting

- Two nested SAT solvers
- bit-blast each $\mathcal{BV}$ atom ψ_i
  $\Longrightarrow \Phi \stackrel{\text{def}}{=} \bigwedge_i (A_i \leftrightarrow BB(\psi_i))$,
  A_i fresh variables labeling $\mathcal{BV}$-atoms ψ_i in φ
  $\Longrightarrow \varphi$ $\mathcal{BV}$-satisfiable iff $\varphi^p \wedge \Phi$ satisfiable
- Exploit SAT under assumptions
  - let μ^p an assignment for φ^p, s.t. $\mu^p \stackrel{\text{def}}{=} \{[\neg]A_1, ..., [\neg]A_n\}$
  - $\mathcal{T}$-solver for $\mathcal{BV}$: $SAT_{assumption}(\Phi, \mu^p)$
  - If UNSAT, generate the unsat core $\eta^p \subseteq \mu^p$

  $\Longrightarrow$ $\neg\eta^p$ used as blocking clause

Outline

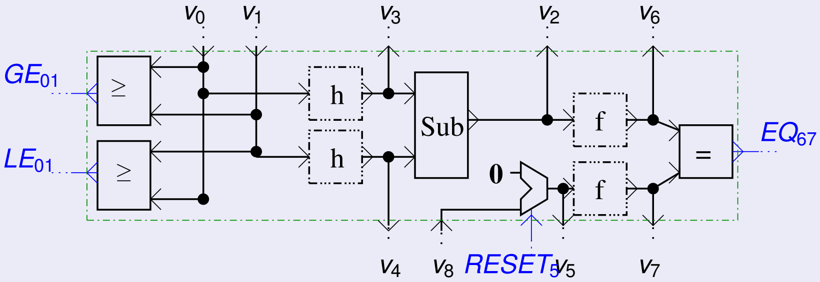
1. Introduction
   - Basics on First-order Logic
   - What is a Theory?
   - Satisfiability Modulo Theories
   - Motivations and Goals of SMT

2. **Efficient SMT solving**
   - Combining SAT with Theory Solvers
   - Theory Solvers for Theories of Interest (hints)
   - **SMT for Combinations of Theories**

3. Beyond Solving: Advanced SMT Functionalities
   - Proofs and Unsatisfiable Cores
   - All-SMT & Predicate Abstraction (hints)
   - SMT with Optimization (Optimization Modulo Theories)

SMT for combined theories: $SMT(\bigcup_i \mathcal{T}_i)$

Problem: Many problems can be expressed as SMT problems only in combination of theories



$\bigcup_i \mathcal{T}_i$ — $SMT(\bigcup_i \mathcal{T}_i)$

$\mathcal{LIA}$: $(GE_{01} \leftrightarrow (v_0 \geq v_1)) \wedge (LE_{01} \leftrightarrow (v_0 \leq v_1))\wedge$
$\mathcal{EUF}$: $(v_3 = h(v_0)) \wedge (v_4 = h(v_1))\wedge$
$\mathcal{LIA}$: $(v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0))\wedge$
$\mathcal{EUF}$ or $\mathcal{LIA}$: $(\neg RESET_5 \rightarrow (v_5 = v_8))\wedge$
$\mathcal{EUF}$: $(v_6 = f(v_2)) \wedge (v_7 = f(v_5))\wedge$
$\mathcal{EUF}$ or $\mathcal{LIA}$: $(EQ_{67} \leftrightarrow (v_6 = v_7)) \wedge ....$

SMT for combined theories: SMT($\mathcal{T}_1 \cup \mathcal{T}_2$)

- Combined theories may be much harder to decide [Pratt'77]
- Solvers have to be combined
- Standard approach for combining $\mathcal{T}_i$-*solver*s:
  (deterministic) Nelson-Oppen/Shostak (N.O.) [61, 63, 75]
  - based on deduction and exchange of equalities on shared variables
  - combined $\mathcal{T}_i$-*solver*s integrated with a SAT tool
- SMT-specific approaches: Delayed Theory Combination [15, 14] and Model-Based Theory Combination [36]
  - based on Boolean search on equalities on shared variables
  - $\mathcal{T}_i$-*solver*s integrated directly with a SAT tool

SMT for combined theories: SMT($\mathcal{T}_1 \cup \mathcal{T}_2$)

- Combined theories may be much harder to decide [Pratt'77]
- Solvers have to be combined
- Standard approach for combining $\mathcal{T}_i$-*solver*'s: (deterministic) Nelson-Oppen/Shostak (N.O.) [61, 63, 75]
  - based on deduction and exchange of equalities on shared variables
  - combined $\mathcal{T}_i$-*solver*'s integrated with a SAT tool
- SMT-specific approaches: Delayed Theory Combination [15, 14] and Model-Based Theory Combination [36]
  - based on Boolean search on equalities on shared variables
  - $\mathcal{T}_i$-*solver*'s integrated directly with a SAT tool

SMT for combined theories: SMT($\mathcal{T}_1 \cup \mathcal{T}_2$)

- Combined theories may be much harder to decide [Pratt'77]
- Solvers have to be combined
- Standard approach for combining $\mathcal{T}_i$-*solver*'s: (deterministic) Nelson-Oppen/Shostak (N.O.) [61, 63, 75]
  - based on deduction and exchange of equalities on shared variables
  - combined $\mathcal{T}_i$-*solver*'s integrated with a SAT tool
- SMT-specific approaches: Delayed Theory Combination [15, 14] and Model-Based Theory Combination [36]
  - based on Boolean search on equalities on shared variables
  - $\mathcal{T}_i$-*solver*'s integrated directly with a SAT tool

Background: Pure Formulas

Consider two theories $\mathcal{T}_1$, $\mathcal{T}_2$ with equality and disjoint signatures Σ_1, Σ_2

- W.l.o.g. we assume all input formulas $\phi \in \mathcal{T}_1 \cup \mathcal{T}_2$ are pure.
  - A formula ϕ is pure iff every atom in ϕ is *i*-pure for some $i \in \{1, 2\}$.
  - An atom/literal ψ in ϕ is *i*-pure if only $=$, variables and symbols from Σ_i can occur in ψ

Purification:

Maps a formula into an equisatisfiable pure formula by labeling terms with fresh variables

$$(f(\underbrace{x + 3y}_{w}) = g(\underbrace{2x - y}_{t})) \qquad [\textit{not pure}]$$

$$\Downarrow$$

$$(w = x + 3y) \wedge (t = 2x - y) \wedge (f(w) = g(t)) \quad [\textit{pure}]$$

Background: Pure Formulas

Consider two theories $\mathcal{T}_1$, $\mathcal{T}_2$ with equality and disjoint signatures Σ_1, Σ_2

- W.l.o.g. we assume all input formulas $\phi \in \mathcal{T}_1 \cup \mathcal{T}_2$ are pure.
  - A formula ϕ is pure iff every atom in ϕ is *i*-pure for some $i \in \{1, 2\}$.
  - An atom/literal ψ in ϕ is *i*-pure if only $=$, variables and symbols from Σ_i can occur in ψ

Purification:

Maps a formula into an equisatisfiable pure formula by labeling terms with fresh variables

$$(f(\underbrace{x + 3y}_{w}) = g(\underbrace{2x - y}_{t})) \qquad [\mathit{not\ pure}]$$

$$\Downarrow$$

$$(w = x + 3y) \wedge (t = 2x - y) \wedge (f(w) = g(t)) \quad [\mathit{pure}]$$

Background: Pure Formulas

Consider two theories $\mathcal{T}_1$, $\mathcal{T}_2$ with equality and disjoint signatures Σ_1, Σ_2

- W.l.o.g. we assume all input formulas $\phi \in \mathcal{T}_1 \cup \mathcal{T}_2$ are pure.
  - A formula ϕ is pure iff every atom in ϕ is *i*-pure for some $i \in \{1, 2\}$.
  - An atom/literal ψ in ϕ is *i*-pure if only $=$, variables and symbols from Σ_i can occur in ψ

Purification:

Maps a formula into an equisatisfiable pure formula by labeling terms with fresh variables

$$(f(\underbrace{x + 3y}_{w}) = g(\underbrace{2x - y}_{t})) \qquad [\textit{not pure}]$$

$$\Downarrow$$

$$(w = x + 3y) \land (t = 2x - y) \land (f(w) = g(t)) \qquad [\textit{pure}]$$

Background: Pure Formulas

Consider two theories $\mathcal{T}_1$, $\mathcal{T}_2$ with equality and disjoint signatures Σ_1, Σ_2

- W.l.o.g. we assume all input formulas $\phi \in \mathcal{T}_1 \cup \mathcal{T}_2$ are pure.
  - A formula ϕ is pure iff every atom in ϕ is *i*-pure for some $i \in \{1, 2\}$.
  - An atom/literal ψ in ϕ is *i*-pure if only $=$, variables and symbols from Σ_i can occur in ψ

Purification:

Maps a formula into an equisatisfiable pure formula by labeling terms with fresh variables

$$(f(\underbrace{x + 3y}_{w}) = g(\underbrace{2x - y}_{t})) \qquad [\textit{not pure}]$$

$$\Downarrow$$

$$(w = x + 3y) \wedge (t = 2x - y) \wedge (f(w) = g(t)) \qquad [\textit{pure}]$$

Exercise

- Purify the following $\mathcal{LIA} \cup \mathcal{EUF} \cup \mathcal{AR}$-formula (see beginning of chapter):

$$\begin{array}{l} \varphi \stackrel{\text{def}}{=} (d \geq 0) \land (d < 1) \land \\ ((f(d) = f(0)) \rightarrow (read(write(V, i, x), i + d) = x + 1)) \end{array}$$

Background: Interface equalities

Interface variables & equalities

- A variable v occurring in a pure formula ϕ is an interface variable iff it occurs in both 1-pure and 2-pure atoms of ϕ.
- An equality $(v_i = v_j)$ is an interface equality for ϕ iff v_i, v_j are interface variables for ϕ.
- We denote the interface equality $v_i = v_j$ by "e_{ij}"

Example:

$$\begin{array}{ll}
\mathcal{LIA}: & (GE_{01} \leftrightarrow (v_0 \geq v_1)) \wedge (LE_{01} \leftrightarrow (v_0 \leq v_1)) \wedge \\
\mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge \\
\mathcal{LIA}: & (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\
\mathcal{EUF} \text{ or } \mathcal{LIA}: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \\
\mathcal{EUF}: & (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\
\mathcal{EUF} \text{ or } \mathcal{LIA}: & (EQ_{67} \leftrightarrow (v_6 = v_7)) \wedge \ldots.
\end{array}$$

$v_0, v_1, v_2, v_3, v_4, v_5$ are interface variables, v_6, v_7, v_8 are not

$\Longrightarrow (v_0 = v_1)$ is an interface equality, $(v_0 = v_6)$ is not.

Background: Interface equalities

Interface variables & equalities

- A variable v occurring in a pure formula ϕ is an interface variable iff it occurs in both 1-pure and 2-pure atoms of ϕ.
- An equality $(v_i = v_j)$ is an interface equality for ϕ iff v_i, v_j are interface variables for ϕ.
- We denote the interface equality $v_i = v_j$ by "e_{ij}"

Example:

$$\begin{array}{ll}
\mathcal{LIA}: & (GE_{01} \leftrightarrow (v_0 \geq v_1)) \wedge (LE_{01} \leftrightarrow (v_0 \leq v_1)) \wedge \\
\mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge \\
\mathcal{LIA}: & (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\
\mathcal{EUF} \text{ or } \mathcal{LIA}: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \\
\mathcal{EUF}: & (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\
\mathcal{EUF} \text{ or } \mathcal{LIA}: & (EQ_{67} \leftrightarrow (v_6 = v_7)) \wedge \ldots.
\end{array}$$

$v_0, v_1, v_2, v_3, v_4, v_5$ are interface variables, v_6, v_7, v_8 are not
$\Longrightarrow (v_0 = v_1)$ is an interface equality, $(v_0 = v_6)$ is not.

Background: Stably-infinite & Convex Theories

Stably-infinite Theories

A Σ-theory $\mathcal{T}$ is stably-infinite iff every quantifier-free $\mathcal{T}$-satisfiable formula is satisfiable in an infinite model of $\mathcal{T}$.

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$, $\mathcal{LIA}$ are stably-infinite
- (fixed-width) bit-vector theories are not stably-infinite

Intuition: a variable can be given an infinite amount of distinct values

Convex Theories

A Σ-theory $\mathcal{T}$ is convex iff, for every collection $l_1, ..., l_k, l', l''$ of literals in $\mathcal{T}$ s.t. l', l'' are in the form $(x = y)$, x, y being variables, we have that:

$\{l_1, ..., l_k\} \models_{\mathcal{T}} (l' \vee l'') \iff \{l_1, ..., l_k\} \models_{\mathcal{T}} l'$ *or* $\{l_1, ..., l_k\} \models_{\mathcal{T}} l''$

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$ are convex
- $\mathcal{LIA}$ is not convex:
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \models ((v = v_0) \vee (v = v_1))$,
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \not\models (v = v_0)$
  $\{(v_0 = 0), (v_1 = 1), (v \geq 0), (v \leq v_1)\} \not\models (v = v_1)$

Intuition: non-convexity produces "case splits"

Background: Stably-infinite & Convex Theories

Stably-infinite Theories

A Σ-theory $\mathcal{T}$ is stably-infinite iff every quantifier-free $\mathcal{T}$-satisfiable formula is satisfiable in an infinite model of $\mathcal{T}$.

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$, $\mathcal{LIA}$ are stably-infinite
- (fixed-width) bit-vector theories are not stably-infinite

Intuition: a variable can be given an infinite amount of distinct values

Convex Theories

A Σ-theory $\mathcal{T}$ is convex iff, for every collection $l_1, ..., l_k, l', l''$ of literals in $\mathcal{T}$ s.t. l', l'' are in the form $(x = y)$, x, y being variables, we have that:

$\{l_1, ..., l_k\} \models_{\mathcal{T}} (l' \vee l'') \iff \{l_1, ..., l_k\} \models_{\mathcal{T}} l'$ *or* $\{l_1, ..., l_k\} \models_{\mathcal{T}} l''$

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$ are convex
- $\mathcal{LIA}$ is not convex:
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \models ((v = v_0) \vee (v = v_1))$,
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \not\models (v = v_0)$
  $\{(v_0 = 0), (v_1 = 1), (v \geq 0), (v \leq v_1)\} \not\models (v = v_1)$

Intuition: non-convexity produces “case splits”

Background: Stably-infinite & Convex Theories

Stably-infinite Theories

A Σ-theory $\mathcal{T}$ is stably-infinite iff every quantifier-free $\mathcal{T}$-satisfiable formula is satisfiable in an infinite model of $\mathcal{T}$.

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$, $\mathcal{LIA}$ are stably-infinite
- (fixed-width) bit-vector theories are not stably-infinite

Intuition: a variable can be given an infinite amount of distinct values

Convex Theories

A Σ-theory $\mathcal{T}$ is convex iff, for every collection $l_1, ..., l_k, l', l''$ of literals in $\mathcal{T}$ s.t. l', l'' are in the form $(x = y)$, x, y being variables, we have that:

$\{l_1, ..., l_k\} \models_{\mathcal{T}} (l' \vee l'') \iff \{l_1, ..., l_k\} \models_{\mathcal{T}} l'$ *or* $\{l_1, ..., l_k\} \models_{\mathcal{T}} l''$

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$ are convex
- $\mathcal{LIA}$ is not convex:
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \models ((v = v_0) \vee (v = v_1))$,
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \not\models (v = v_0)$
  $\{(v_0 = 0), (v_1 = 1), (v \geq 0), (v \leq v_1)\} \not\models (v = v_1)$

Intuition: non-convexity produces "case splits"

Background: Stably-infinite & Convex Theories

Stably-infinite Theories

A Σ-theory $\mathcal{T}$ is stably-infinite iff every quantifier-free $\mathcal{T}$-satisfiable formula is satisfiable in an infinite model of $\mathcal{T}$.

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$, $\mathcal{LIA}$ are stably-infinite
- (fixed-width) bit-vector theories are not stably-infinite

Intuition: a variable can be given an infinite amount of distinct values

Convex Theories

A Σ-theory $\mathcal{T}$ is convex iff, for every collection $l_1, ..., l_k, l', l''$ of literals in $\mathcal{T}$ s.t. l', l'' are in the form $(x = y)$, x, y being variables, we have that:

$\{l_1, ..., l_k\} \models_{\mathcal{T}} (l' \vee l'') \iff \{l_1, ..., l_k\} \models_{\mathcal{T}} l'$ *or* $\{l_1, ..., l_k\} \models_{\mathcal{T}} l''$

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$ are convex
- $\mathcal{LIA}$ is not convex:
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \models ((v = v_0) \vee (v = v_1))$,
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \not\models (v = v_0)$
  $\{(v_0 = 0), (v_1 = 1), (v \geq 0), (v \leq v_1)\} \not\models (v = v_1)$

Intuition: non-convexity produces "case splits"

Background: Stably-infinite & Convex Theories

Stably-infinite Theories

A Σ-theory $\mathcal{T}$ is stably-infinite iff every quantifier-free $\mathcal{T}$-satisfiable formula is satisfiable in an infinite model of $\mathcal{T}$.

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$, $\mathcal{LIA}$ are stably-infinite
- (fixed-width) bit-vector theories are not stably-infinite

Intuition: a variable can be given an infinite amount of distinct values

Convex Theories

A Σ-theory $\mathcal{T}$ is convex iff, for every collection $l_1, ..., l_k, l', l''$ of literals in $\mathcal{T}$ s.t. l', l'' are in the form $(x = y)$, x, y being variables, we have that:

$$\{l_1, ..., l_k\} \models_{\mathcal{T}} (l' \vee l'') \iff \{l_1, ..., l_k\} \models_{\mathcal{T}} l' \;\; or \;\; \{l_1, ..., l_k\} \models_{\mathcal{T}} l''$$

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$ are convex
- $\mathcal{LIA}$ is not convex:
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \models ((v = v_0) \vee (v = v_1))$,
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \not\models (v = v_0)$
  $\{(v_0 = 0), (v_1 = 1), (v \geq 0), (v \leq v_1)\} \not\models (v = v_1)$

Intuition: non-convexity produces “case splits”

Background: Stably-infinite & Convex Theories

Stably-infinite Theories

A Σ-theory $\mathcal{T}$ is stably-infinite iff every quantifier-free $\mathcal{T}$-satisfiable formula is satisfiable in an infinite model of $\mathcal{T}$.

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$, $\mathcal{LIA}$ are stably-infinite
- (fixed-width) bit-vector theories are not stably-infinite

Intuition: a variable can be given an infinite amount of distinct values

Convex Theories

A Σ-theory $\mathcal{T}$ is convex iff, for every collection $l_1, ..., l_k, l', l''$ of literals in $\mathcal{T}$ s.t. l', l'' are in the form $(x = y)$, x, y being variables, we have that:

$$\{l_1, ..., l_k\} \models_{\mathcal{T}} (l' \vee l'') \iff \{l_1, ..., l_k\} \models_{\mathcal{T}} l' \;\; or \;\; \{l_1, ..., l_k\} \models_{\mathcal{T}} l''$$

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$ are convex
- $\mathcal{LIA}$ is not convex:

$$\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \models ((v = v_0) \vee (v = v_1)),$$
$$\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \not\models (v = v_0)$$
$$\{(v_0 = 0), (v_1 = 1), (v \geq 0), (v \leq v_1)\} \not\models (v = v_1)$$

Intuition: non-convexity produces “case splits”

Background: Stably-infinite & Convex Theories

Stably-infinite Theories

A Σ-theory $\mathcal{T}$ is stably-infinite iff every quantifier-free $\mathcal{T}$-satisfiable formula is satisfiable in an infinite model of $\mathcal{T}$.

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$, $\mathcal{LIA}$ are stably-infinite
- (fixed-width) bit-vector theories are not stably-infinite

Intuition: a variable can be given an infinite amount of distinct values

Convex Theories

A Σ-theory $\mathcal{T}$ is convex iff, for every collection $l_1, ..., l_k, l', l''$ of literals in $\mathcal{T}$ s.t. l', l'' are in the form $(x = y)$, x, y being variables, we have that:

$$\{l_1, ..., l_k\} \models_{\mathcal{T}} (l' \vee l'') \iff \{l_1, ..., l_k\} \models_{\mathcal{T}} l' \;\; or \;\; \{l_1, ..., l_k\} \models_{\mathcal{T}} l''$$

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$ are convex
- $\mathcal{LIA}$ is not convex:

  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \models ((v = v_0) \vee (v = v_1))$,

  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \not\models (v = v_0)$

  $\{(v_0 = 0), (v_1 = 1), (v \geq 0), (v \leq v_1)\} \not\models (v = v_1)$

Intuition: non-convexity produces “case splits”

Background: Stably-infinite & Convex Theories

Stably-infinite Theories

A Σ-theory $\mathcal{T}$ is stably-infinite iff every quantifier-free $\mathcal{T}$-satisfiable formula is satisfiable in an infinite model of $\mathcal{T}$.

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$, $\mathcal{LIA}$ are stably-infinite
- (fixed-width) bit-vector theories are not stably-infinite

Intuition: a variable can be given an infinite amount of distinct values

Convex Theories

A Σ-theory $\mathcal{T}$ is convex iff, for every collection $l_1, ..., l_k, l', l''$ of literals in $\mathcal{T}$ s.t. l', l'' are in the form $(x = y)$, x, y being variables, we have that:

$\{l_1, ..., l_k\} \models_{\mathcal{T}} (l' \vee l'') \iff \{l_1, ..., l_k\} \models_{\mathcal{T}} l'$ *or* $\{l_1, ..., l_k\} \models_{\mathcal{T}} l''$

- $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LRA}$ are convex
- $\mathcal{LIA}$ is not convex:
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \models ((v = v_0) \vee (v = v_1)),$
  $\{(v_0 = 0), (v_1 = 1), (v \geq v_0), (v \leq v_1)\} \not\models (v = v_0)$
  $\{(v_0 = 0), (v_1 = 1), (v \geq 0), (v \leq v_1)\} \not\models (v = v_1)$

Intuition: non-convexity produces “case splits”

$SMT(\bigcup_i \mathcal{T}_i)$ via “classic” Nelson-Oppen

Main Problem

- Given $\mu \overset{\text{def}}{=} \bigcup_i \mu_i$ s.t. each μ_i contains i-pure literals
  - distinct $\mathcal{T}_i$-solver can be invoked separately on each μ_i...
  - ...producing distinct $\mathcal{T}_i$-specific models $\mathcal{M}_i$
- If the theories share no constant, function or predicate symbol $\Longrightarrow$ trivial
- Problem: One predicate (nearly) always shared between distinct theories $\mathcal{T}_i$: equality “=”

$\Longrightarrow$ all models must agree on interface equalities:

$$\mathcal{M}_i \models_{\mathcal{T}_i} (v_k = v_l) \;\; iff \;\; \mathcal{M}_j \models_{\mathcal{T}_j} (v_k = v_l),$$

for every pair of shared variables v_k, v_l

Main idea

Combine two or more $\mathcal{T}_i$*-solvers* into one $(\bigcup_i \mathcal{T}_i)$-solver via Nelson-Oppen/Shostak (N.O.) combination procedure [62, 76]

- based on the deduction and exchange of equalities between shared variables/terms (interface equalities, e_{ij}s)
- important improvements and evolutions [68, 7, 39]

$SMT(\bigcup_i \mathcal{T}_i)$ via “classic” Nelson-Oppen

Main Problem

- Given $\mu \stackrel{\text{def}}{=} \bigcup_i \mu_i$ s.t. each μ_i contains i-pure literals
  - distinct $\mathcal{T}_i$-solver can be invoked separately on each μ_i...
  - ...producing distinct $\mathcal{T}_i$-specific models $\mathcal{M}_i$
- If the theories share no constant, function or predicate symbol $\Longrightarrow$ trivial
- Problem: One predicate (nearly) always shared between distinct theories $\mathcal{T}_i$: equality “=”

$\Longrightarrow$ all models must agree on interface equalities:

$$\mathcal{M}_i \models_{\mathcal{T}_i} (v_k = v_l) \;\; iff \;\; \mathcal{M}_j \models_{\mathcal{T}_j} (v_k = v_l),$$

for every pair of shared variables v_k, v_l

Main idea

Combine two or more $\mathcal{T}_i$-*solvers* into one $(\bigcup_i \mathcal{T}_i)$-solver via Nelson-Oppen/Shostak (N.O.) combination procedure [62, 76]

- based on the deduction and exchange of equalities between shared variables/terms (interface equalities, e_{ij}s)
- important improvements and evolutions [68, 7, 39]

$SMT(\bigcup_i \mathcal{T}_i)$ via "classic" Nelson-Oppen

Main Problem

- Given $\mu \overset{\text{def}}{=} \bigcup_i \mu_i$ s.t. each μ_i contains i-pure literals
  - distinct $\mathcal{T}_i$-solver can be invoked separately on each μ_i...
  - ...producing distinct $\mathcal{T}_i$-specific models $\mathcal{M}_i$
- If the theories share no constant, function or predicate symbol $\Longrightarrow$ trivial
- Problem: One predicate (nearly) always shared between distinct theories $\mathcal{T}_i$: equality "="

$\Longrightarrow$ all models must agree on interface equalities:

$$\mathcal{M}_i \models_{\mathcal{T}_i} (v_k = v_l) \ \ iff \ \ \mathcal{M}_j \models_{\mathcal{T}_j} (v_k = v_l),$$

for every pair of shared variables v_k, v_l

Main idea

Combine two or more $\mathcal{T}_i$-*solvers* into one $(\bigcup_i \mathcal{T}_i)$-solver via Nelson-Oppen/Shostak (N.O.) combination procedure [62, 76]

- based on the deduction and exchange of equalities between shared variables/terms (interface equalities, e_{ij}s)
- important improvements and evolutions [68, 7, 39]

$SMT(\bigcup_i \mathcal{T}_i)$ via "classic" Nelson-Oppen

Main Problem

- Given $\mu \stackrel{\text{def}}{=} \bigcup_i \mu_i$ s.t. each μ_i contains i-pure literals
  - distinct $\mathcal{T}_i$-solver can be invoked separately on each μ_i...
  - ...producing distinct $\mathcal{T}_i$-specific models $\mathcal{M}_i$
- If the theories share no constant, function or predicate symbol $\Longrightarrow$ trivial
- Problem: One predicate (nearly) always shared between distinct theories $\mathcal{T}_i$: equality "="

$\Longrightarrow$ all models must agree on interface equalities:

$$\mathcal{M}_i \models_{\mathcal{T}_i} (v_k = v_l) \;\; iff \;\; \mathcal{M}_j \models_{\mathcal{T}_j} (v_k = v_l),$$

for every pair of shared variables v_k, v_l

Main idea

Combine two or more $\mathcal{T}_i$-*solvers* into one $(\bigcup_i \mathcal{T}_i)$-solver via Nelson-Oppen/Shostak (N.O.) combination procedure [62, 76]

- based on the deduction and exchange of equalities between shared variables/terms (interface equalities, e_{ij}s)
- important improvements and evolutions [68, 7, 39]

$SMT(\bigcup_i \mathcal{T}_i)$ via "classic" Nelson-Oppen

Main Problem

- Given $\mu \stackrel{\text{def}}{=} \bigcup_i \mu_i$ s.t. each μ_i contains i-pure literals
  - distinct $\mathcal{T}_i$-solver can be invoked separately on each μ_i...
  - ...producing distinct $\mathcal{T}_i$-specific models $\mathcal{M}_i$
- If the theories share no constant, function or predicate symbol $\Longrightarrow$ trivial
- Problem: One predicate (nearly) always shared between distinct theories $\mathcal{T}_i$: equality "="

$\Longrightarrow$ all models must agree on interface equalities:

$$\mathcal{M}_i \models_{\mathcal{T}_i} (v_k = v_l) \;\; iff \;\; \mathcal{M}_j \models_{\mathcal{T}_j} (v_k = v_l),$$

for every pair of shared variables v_k, v_l

Main idea

Combine two or more $\mathcal{T}_i$-*solvers* into one $(\bigcup_i \mathcal{T}_i)$-solver via Nelson-Oppen/Shostak (N.O.) combination procedure [62, 76]

- based on the deduction and exchange of equalities between shared variables/terms (interface equalities, e_{ij}s)
- important improvements and evolutions [68, 7, 39]

$SMT(\bigcup_i \mathcal{T}_i)$ via “classic” Nelson-Oppen

Main Problem

- Given $\mu \overset{\text{def}}{=} \bigcup_i \mu_i$ s.t. each μ_i contains i-pure literals
  - distinct $\mathcal{T}_i$-solver can be invoked separately on each μ_i...
  - ...producing distinct $\mathcal{T}_i$-specific models $\mathcal{M}_i$
- If the theories share no constant, function or predicate symbol $\Longrightarrow$ trivial
- Problem: One predicate (nearly) always shared between distinct theories $\mathcal{T}_i$: equality “=”

$\Longrightarrow$ all models must agree on interface equalities:

$$\mathcal{M}_i \models_{\mathcal{T}_i} (v_k = v_l) \;\; iff \;\; \mathcal{M}_j \models_{\mathcal{T}_j} (v_k = v_l),$$

for every pair of shared variables v_k, v_l

Main idea

Combine two or more $\mathcal{T}_i$*-solvers* into one $(\bigcup_i \mathcal{T}_i)$-solver via Nelson-Oppen/Shostak (N.O.) combination procedure [62, 76]

- based on the deduction and exchange of equalities between shared variables/terms (interface equalities, e_{ij}s)
- important improvements and evolutions [68, 7, 39]

$SMT(\bigcup_i \mathcal{T}_i)$ via “classic” Nelson-Oppen

Main Problem

- Given $\mu \overset{\text{def}}{=} \bigcup_i \mu_i$ s.t. each μ_i contains i-pure literals
  - distinct $\mathcal{T}_i$-solver can be invoked separately on each μ_i...
  - ...producing distinct $\mathcal{T}_i$-specific models $\mathcal{M}_i$
- If the theories share no constant, function or predicate symbol $\Longrightarrow$ trivial
- Problem: One predicate (nearly) always shared between distinct theories $\mathcal{T}_i$: equality “=”

$\Longrightarrow$ all models must agree on interface equalities:

$$\mathcal{M}_i \models_{\mathcal{T}_i} (v_k = v_l) \ \ iff \ \ \mathcal{M}_j \models_{\mathcal{T}_j} (v_k = v_l),$$

for every pair of shared variables v_k, v_l

Main idea

Combine two or more $\mathcal{T}_i$-*solvers* into one $(\bigcup_i \mathcal{T}_i)$-solver via Nelson-Oppen/Shostak (N.O.) combination procedure [62, 76]

- based on the deduction and exchange of equalities between shared variables/terms (interface equalities, e_{ij}s)
- important improvements and evolutions [68, 7, 39]

$SMT(\bigcup_i \mathcal{T}_i)$ via “classic” Nelson-Oppen

Main Problem

- Given $\mu \stackrel{\text{def}}{=} \bigcup_i \mu_i$ s.t. each μ_i contains i-pure literals
  - distinct $\mathcal{T}_i$-solver can be invoked separately on each μ_i...
  - ...producing distinct $\mathcal{T}_i$-specific models $\mathcal{M}_i$
- If the theories share no constant, function or predicate symbol $\Longrightarrow$ trivial
- Problem: One predicate (nearly) always shared between distinct theories $\mathcal{T}_i$: equality “=”

$\Longrightarrow$ all models must agree on interface equalities:

$$\mathcal{M}_i \models_{\mathcal{T}_i} (v_k = v_l) \;\; \textit{iff} \;\; \mathcal{M}_j \models_{\mathcal{T}_j} (v_k = v_l),$$

for every pair of shared variables v_k, v_l

Main idea

Combine two or more $\mathcal{T}_i$*-solvers* into one $(\bigcup_i \mathcal{T}_i)$-solver via Nelson-Oppen/Shostak (N.O.) combination procedure [62, 76]

- based on the deduction and exchange of equalities between shared variables/terms (interface equalities, e_{ij}s)
- important improvements and evolutions [68, 7, 39]

Schema of N.O. combination of $\mathcal{T}$-solvers: no($\mathcal{T}_1, \mathcal{T}_2$)

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be a stably infinite theory admitting a satisfiability $\mathcal{T}_i$-solver, and μ_i a set of i-pure literals.
We want to to decide the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\mu_1 \cup \mu_2$

- each $\mathcal{T}_i$-solver, in turn
  - checks the $\mathcal{T}_i$-satisfiability of μ_i,
  - deduces all the (disjunctions of) interface equalities which derive from μ_i
  - passes them to $\mathcal{T}_j$-solve, $j \neq i$, which adds them to μ_j

  until either:
  - one $\mathcal{T}_i$-solver detects unsatisfiability ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsat)
  - no more deductions are possible ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-sat)
- disjunctions of literals (due to non-convexity) force case-splitting

Schema of N.O. combination of $\mathcal{T}$-solvers: no($\mathcal{T}_1, \mathcal{T}_2$)

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be a stably infinite theory admitting a satisfiability $\mathcal{T}_i$-solver, and μ_i a set of i-pure literals.
We want to to decide the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\mu_1 \cup \mu_2$

- each $\mathcal{T}_i$-solver, in turn
  - checks the $\mathcal{T}_i$-satisfiability of μ_i,
  - deduces all the (disjunctions of) interface equalities which derive from μ_i
  - passes them to $\mathcal{T}_j$-solve, $j \neq i$, which adds them to μ_j

  until either:
  - one $\mathcal{T}_i$-solver detects unsatisfiability ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsat)
  - no more deductions are possible ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-sat)
- disjunctions of literals (due to non-convexity) force case-splitting

Schema of N.O. combination of $\mathcal{T}$-solvers: no($\mathcal{T}_1, \mathcal{T}_2$)

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be a stably infinite theory admitting a satisfiability $\mathcal{T}_i$-solver, and μ_i a set of i-pure literals.

We want to to decide the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\mu_1 \cup \mu_2$

- each $\mathcal{T}_i$-solver, in turn
  - checks the $\mathcal{T}_i$-satisfiability of μ_i,
  - deduces all the (disjunctions of) interface equalities which derive from μ_i
  - passes them to $\mathcal{T}_j$-solve, $j \neq i$, which adds them to μ_j

  until either:
  - one $\mathcal{T}_i$-solver detects unsatisfiability ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsat)
  - no more deductions are possible ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-sat)
- disjunctions of literals (due to non-convexity) force case-splitting

Schema of N.O. combination of $\mathcal{T}$-solvers: no($\mathcal{T}_1, \mathcal{T}_2$)

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be a stably infinite theory admitting a satisfiability $\mathcal{T}_i$-solver, and μ_i a set of i-pure literals.

We want to to decide the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\mu_1 \cup \mu_2$

- each $\mathcal{T}_i$-solver, in turn
  - checks the $\mathcal{T}_i$-satisfiability of μ_i,
  - deduces all the (disjunctions of) interface equalities which derive from μ_i
  - passes them to $\mathcal{T}_j$-solve, $j \neq i$, which adds them to μ_j

  until either:
  - one $\mathcal{T}_i$-solver detects unsatisfiability ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsat)
  - no more deductions are possible ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-sat)
- disjunctions of literals (due to non-convexity) force case-splitting

Schema of N.O. combination of $\mathcal{T}$-solvers: no($\mathcal{T}_1, \mathcal{T}_2$)

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be a stably infinite theory admitting a satisfiability $\mathcal{T}_i$-solver, and μ_i a set of i-pure literals.

We want to to decide the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\mu_1 \cup \mu_2$

- each $\mathcal{T}_i$-solver, in turn
  - checks the $\mathcal{T}_i$-satisfiability of μ_i,
  - deduces all the (disjunctions of) interface equalities which derive from μ_i
  - passes them to T_j-solve, $j \neq i$, which adds them to μ_j

  until either:
  - one $\mathcal{T}_i$-solver detects unsatisfiability ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsat)
  - no more deductions are possible ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-sat)
- disjunctions of literals (due to non-convexity) force case-splitting

Schema of N.O. combination of $\mathcal{T}$-solvers: no($\mathcal{T}_1, \mathcal{T}_2$)

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be a stably infinite theory admitting a satisfiability $\mathcal{T}_i$-solver, and μ_i a set of i-pure literals.
We want to to decide the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\mu_1 \cup \mu_2$

- each $\mathcal{T}_i$-solver, in turn
  - checks the $\mathcal{T}_i$-satisfiability of μ_i,
  - deduces all the (disjunctions of) interface equalities which derive from μ_i
  - passes them to T_j-solve, $j \neq i$, which adds them to μ_j

  until either:
  - one $\mathcal{T}_i$-solver detects unsatisfiability ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsat)
  - no more deductions are possible ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-sat)
- disjunctions of literals (due to non-convexity) force case-splitting

Schema of N.O. combination of $\mathcal{T}$-solvers: no($\mathcal{T}_1, \mathcal{T}_2$)

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be a stably infinite theory admitting a satisfiability $\mathcal{T}_i$-solver, and μ_i a set of *i*-pure literals.

We want to to decide the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\mu_1 \cup \mu_2$

- each $\mathcal{T}_i$-solver, in turn
    - checks the $\mathcal{T}_i$-satisfiability of μ_i,
    - deduces all the (disjunctions of) interface equalities which derive from μ_i
    - passes them to T_j-solve, $j \neq i$, which adds them to μ_j

  until either:
    - one $\mathcal{T}_i$-solver detects unsatisfiability ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsat)
    - no more deductions are possible ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-sat)
- disjunctions of literals (due to non-convexity) force case-splitting

Schema of N.O. combination of $\mathcal{T}$-solvers: no($\mathcal{T}_1, \mathcal{T}_2$)

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be a stably infinite theory admitting a satisfiability $\mathcal{T}_i$-solver, and μ_i a set of i-pure literals.
We want to to decide the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\mu_1 \cup \mu_2$

- each $\mathcal{T}_i$-solver, in turn
  - checks the $\mathcal{T}_i$-satisfiability of μ_i,
  - deduces all the (disjunctions of) interface equalities which derive from μ_i
  - passes them to T_j-solve, $j \neq i$, which adds them to μ_j

  until either:
  - one $\mathcal{T}_i$-solver detects unsatisfiability ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsat)
  - no more deductions are possible ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-sat)
- disjunctions of literals (due to non-convexity) force case-splitting

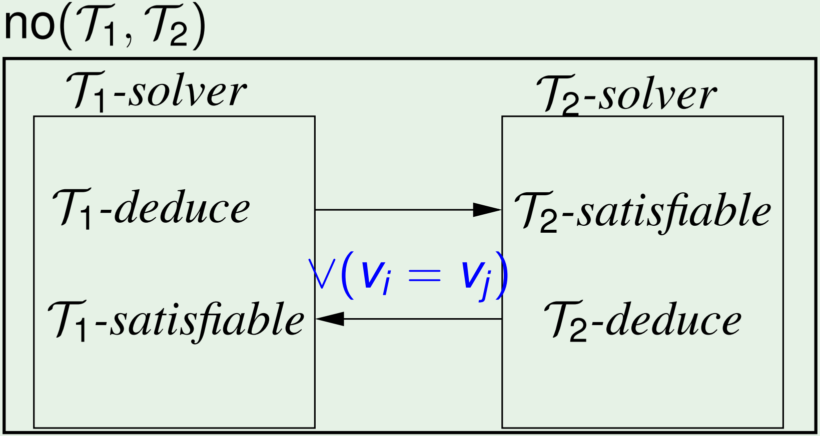
Schema of N.O. combination of $\mathcal{T}$-solvers: no($\mathcal{T}_1, \mathcal{T}_2$)

For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be a stably infinite theory admitting a satisfiability $\mathcal{T}_i$-solver, and μ_i a set of i-pure literals.
We want to to decide the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\mu_1 \cup \mu_2$

- each $\mathcal{T}_i$-solver, in turn
  - checks the $\mathcal{T}_i$-satisfiability of μ_i,
  - deduces all the (disjunctions of) interface equalities which derive from μ_i
  - passes them to T_j-solve, $j \neq i$, which adds them to μ_j

  until either:
  - one $\mathcal{T}_i$-solver detects unsatisfiability ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsat)
  - no more deductions are possible ($\mu_1 \cup \mu_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$-sat)
- disjunctions of literals (due to non-convexity) force case-splitting

Schema of N.O. combination of T-solvers: $no(\mathcal{T}_1, \mathcal{T}_2)$

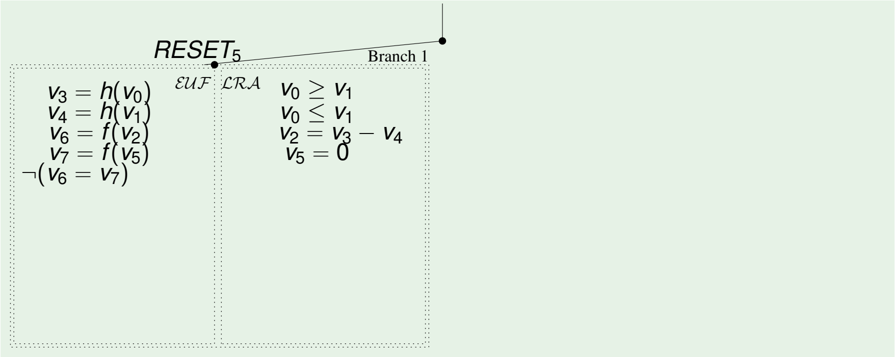


N.O. Example (Convex Theory)

$$\begin{array}{ll}
\mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\
\mathcal{LRA}: & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\
Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).
\end{array}$$

N.O. Example (Convex Theory)

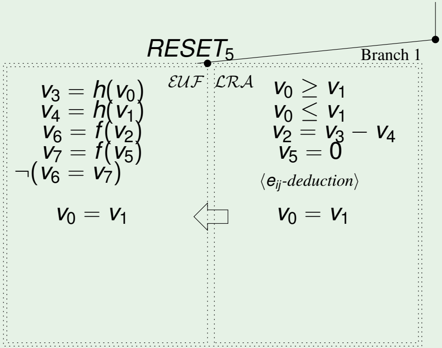
$$\begin{array}{ll}
\mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\
\mathcal{LRA}: & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\
Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).
\end{array}$$



$$\begin{array}{r}
v_3 = h(v_0) \\
v_4 = h(v_1) \\
v_6 = f(v_2) \\
v_7 = f(v_5) \\
\neg(v_6 = v_7)
\end{array}
\qquad
\begin{array}{l}
v_0 \geq v_1 \\
v_0 \leq v_1 \\
v_2 = v_3 - v_4 \\
v_5 = 0
\end{array}$$

N.O. Example (Convex Theory)

$$\begin{array}{ll} \mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\ \mathcal{LRA}: & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\ Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7). \end{array}$$



$RESET_5$

Branch 1

$\mathcal{EUF}$ $\mathcal{LRA}$

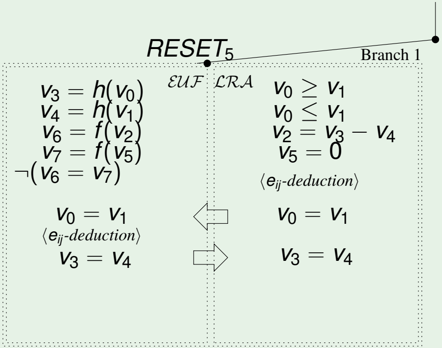
$$\begin{array}{c} v_3 = h(v_0) \\ v_4 = h(v_1) \\ v_6 = f(v_2) \\ v_7 = f(v_5) \\ \neg(v_6 = v_7) \end{array} \qquad \begin{array}{c} v_0 \geq v_1 \\ v_0 \leq v_1 \\ v_2 = v_3 - v_4 \\ v_5 = 0 \end{array}$$

$\langle e_{ij}$-deduction$\rangle$

$$v_0 = v_1 \quad \Leftarrow \quad v_0 = v_1$$

N.O. Example (Convex Theory)

$$\begin{array}{ll} \mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\ \mathcal{LRA}: & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\ Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7). \end{array}$$



$RESET_5$

Branch 1

$\mathcal{EUF}$ | $\mathcal{LRA}$

$$\begin{array}{r} v_3 = h(v_0) \\ v_4 = h(v_1) \\ v_6 = f(v_2) \\ v_7 = f(v_5) \\ \neg(v_6 = v_7) \end{array}$$

$$\begin{array}{l} v_0 \geq v_1 \\ v_0 \leq v_1 \\ v_2 = v_3 - v_4 \\ v_5 = 0 \end{array}$$

$\langle e_{ij}$-deduction$\rangle$

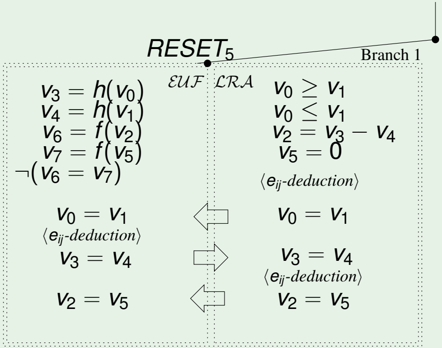
$v_0 = v_1 \Leftarrow v_0 = v_1$

$\langle e_{ij}$-deduction$\rangle$

$v_3 = v_4 \Rightarrow v_3 = v_4$

N.O. Example (Convex Theory)

$$\begin{array}{ll}
\mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\
\mathcal{LRA}: & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\
Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).
\end{array}$$



$RESET_5$

Branch 1

$\mathcal{EUF}$ | $\mathcal{LRA}$

$$\begin{array}{l}
v_3 = h(v_0)\\
v_4 = h(v_1)\\
v_6 = f(v_2)\\
v_7 = f(v_5)\\
\neg(v_6 = v_7)
\end{array}
\qquad
\begin{array}{l}
v_0 \geq v_1\\
v_0 \leq v_1\\
v_2 = v_3 - v_4\\
v_5 = 0
\end{array}$$

$\langle e_{ij}\text{-deduction}\rangle$

$v_0 = v_1 \Leftarrow v_0 = v_1$

$\langle e_{ij}\text{-deduction}\rangle$

$v_3 = v_4 \Rightarrow v_3 = v_4$

$\langle e_{ij}\text{-deduction}\rangle$

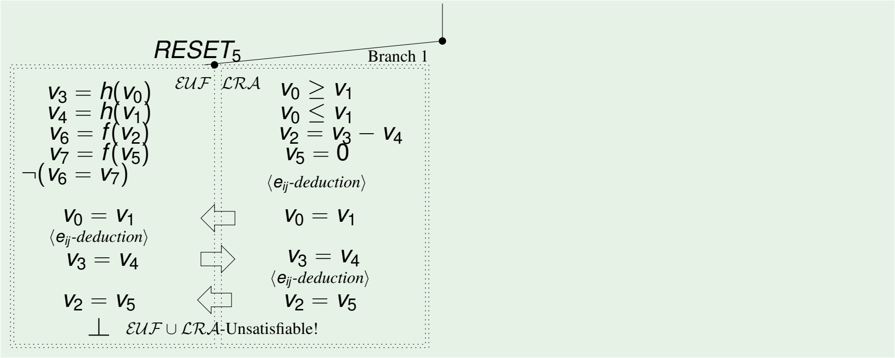
$v_2 = v_5 \Leftarrow v_2 = v_5$

N.O. Example (Convex Theory)

$\mathcal{EUF}$: $(v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge$

$\mathcal{LRA}$: $(v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge$

Both : $(\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7)$.



$RESET_5$

Branch 1

$\mathcal{EUF}$ | $\mathcal{LRA}$

$v_3 = h(v_0)$
$v_4 = h(v_1)$
$v_6 = f(v_2)$
$v_7 = f(v_5)$
$\neg(v_6 = v_7)$

$v_0 \geq v_1$
$v_0 \leq v_1$
$v_2 = v_3 - v_4$
$v_5 = 0$

$\langle e_{ij}$-*deduction*$\rangle$

$v_0 = v_1$ ⇦ $v_0 = v_1$

$\langle e_{ij}$-*deduction*$\rangle$

$v_3 = v_4$ ⇨ $v_3 = v_4$

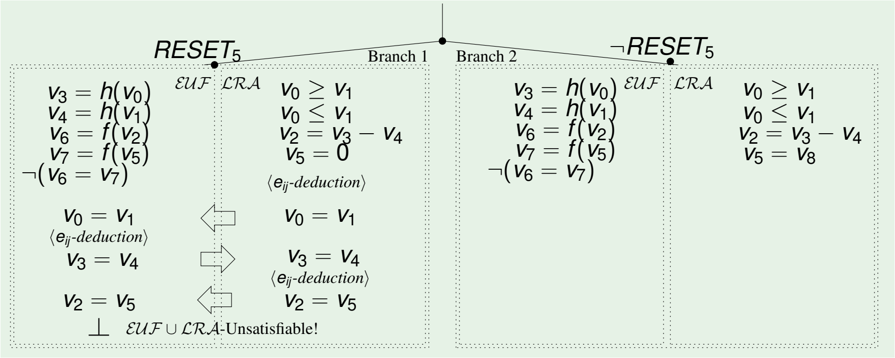
$\langle e_{ij}$-*deduction*$\rangle$

$v_2 = v_5$ ⇦ $v_2 = v_5$

$\bot$ $\mathcal{EUF} \cup \mathcal{LRA}$-Unsatisfiable!

N.O. Example (Convex Theory)

$$\begin{aligned}
\mathcal{EUF}: &\quad (v_3 = h(v_0)) \land (v_4 = h(v_1)) \land (v_6 = f(v_2)) \land (v_7 = f(v_5)) \land \\
\mathcal{LRA}: &\quad (v_0 \geq v_1) \land (v_0 \leq v_1) \land (v_2 = v_3 - v_4) \land (RESET_5 \rightarrow (v_5 = 0)) \land \\
Both: &\quad (\neg RESET_5 \rightarrow (v_5 = v_8)) \land \neg(v_6 = v_7).
\end{aligned}$$



$RESET_5$

Branch 1

Branch 2

$\neg RESET_5$

$\mathcal{EUF}$ $\mathcal{LRA}$

$$\begin{aligned} v_3 &= h(v_0) \\ v_4 &= h(v_1) \\ v_6 &= f(v_2) \\ v_7 &= f(v_5) \\ \neg(v_6 &= v_7) \end{aligned} \qquad \begin{aligned} v_0 &\geq v_1 \\ v_0 &\leq v_1 \\ v_2 &= v_3 - v_4 \\ v_5 &= 0 \end{aligned}$$

$\langle e_{ij}$-deduction$\rangle$

$v_0 = v_1$ ⇐ $v_0 = v_1$

$\langle e_{ij}$-deduction$\rangle$

$v_3 = v_4$ ⇒ $v_3 = v_4$

$\langle e_{ij}$-deduction$\rangle$

$v_2 = v_5$ ⇐ $v_2 = v_5$

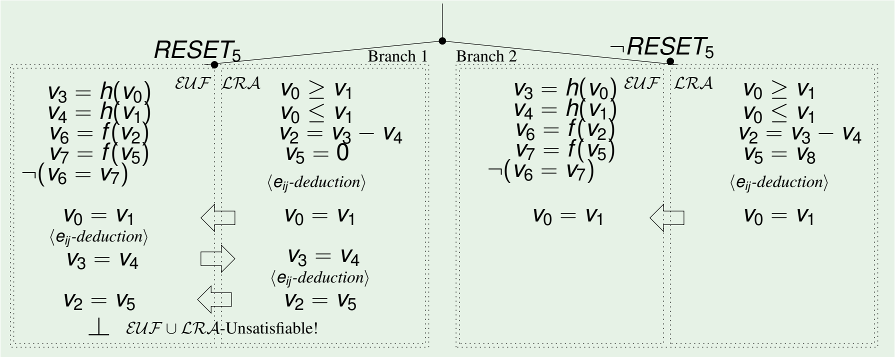
$\bot$ $\mathcal{EUF} \cup \mathcal{LRA}$-Unsatisfiable!

$\mathcal{EUF}$ $\mathcal{LRA}$

$$\begin{aligned} v_3 &= h(v_0) \\ v_4 &= h(v_1) \\ v_6 &= f(v_2) \\ v_7 &= f(v_5) \\ \neg(v_6 &= v_7) \end{aligned} \qquad \begin{aligned} v_0 &\geq v_1 \\ v_0 &\leq v_1 \\ v_2 &= v_3 - v_4 \\ v_5 &= v_8 \end{aligned}$$

N.O. Example (Convex Theory)

$$\begin{array}{ll}\mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\ \mathcal{LRA}: & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\ Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).\end{array}$$



$RESET_5$ Branch 1

Branch 2 $\neg RESET_5$

$\mathcal{EUF}$ $\mathcal{LRA}$

$$\begin{array}{c}v_3 = h(v_0)\\ v_4 = h(v_1)\\ v_6 = f(v_2)\\ v_7 = f(v_5)\\ \neg(v_6 = v_7)\end{array} \qquad \begin{array}{c}v_0 \geq v_1\\ v_0 \leq v_1\\ v_2 = v_3 - v_4\\ v_5 = 0\end{array}$$

$\langle e_{ij}$-deduction$\rangle$

$$v_0 = v_1 \Leftarrow v_0 = v_1$$

$\langle e_{ij}$-deduction$\rangle$

$$v_3 = v_4 \Rightarrow v_3 = v_4$$

$\langle e_{ij}$-deduction$\rangle$

$$v_2 = v_5 \Leftarrow v_2 = v_5$$

$\bot$ $\mathcal{EUF} \cup \mathcal{LRA}$-Unsatisfiable!

$\mathcal{EUF}$ $\mathcal{LRA}$

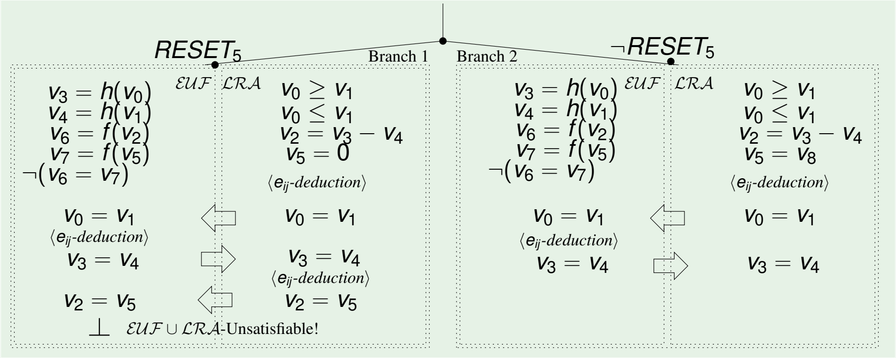
$$\begin{array}{c}v_3 = h(v_0)\\ v_4 = h(v_1)\\ v_6 = f(v_2)\\ v_7 = f(v_5)\\ \neg(v_6 = v_7)\end{array} \qquad \begin{array}{c}v_0 \geq v_1\\ v_0 \leq v_1\\ v_2 = v_3 - v_4\\ v_5 = v_8\end{array}$$

$\langle e_{ij}$-deduction$\rangle$

$$v_0 = v_1 \Leftarrow v_0 = v_1$$

N.O. Example (Convex Theory)

$$\begin{array}{ll}\mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\ \mathcal{LRA}: & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\ Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).\end{array}$$



$RESET_5$ Branch 1

$\mathcal{EUF}$ | $\mathcal{LRA}$

$v_3 = h(v_0)$
$v_4 = h(v_1)$
$v_6 = f(v_2)$
$v_7 = f(v_5)$
$\neg(v_6 = v_7)$

$v_0 \geq v_1$
$v_0 \leq v_1$
$v_2 = v_3 - v_4$
$v_5 = 0$
$\langle e_{ij}\text{-deduction}\rangle$

$v_0 = v_1$ ⇐ $v_0 = v_1$
$\langle e_{ij}\text{-deduction}\rangle$
$v_3 = v_4$ ⇒ $v_3 = v_4$
$\langle e_{ij}\text{-deduction}\rangle$
$v_2 = v_5$ ⇐ $v_2 = v_5$

$\bot$ $\mathcal{EUF} \cup \mathcal{LRA}$-Unsatisfiable!

Branch 2 $\neg RESET_5$

$\mathcal{EUF}$ | $\mathcal{LRA}$

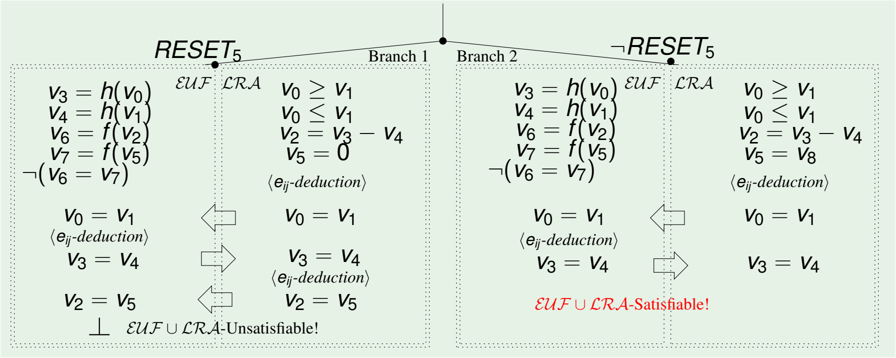
$v_3 = h(v_0)$
$v_4 = h(v_1)$
$v_6 = f(v_2)$
$v_7 = f(v_5)$
$\neg(v_6 = v_7)$

$v_0 \geq v_1$
$v_0 \leq v_1$
$v_2 = v_3 - v_4$
$v_5 = v_8$
$\langle e_{ij}\text{-deduction}\rangle$

$v_0 = v_1$ ⇐ $v_0 = v_1$
$\langle e_{ij}\text{-deduction}\rangle$
$v_3 = v_4$ ⇒ $v_3 = v_4$

N.O. Example (Convex Theory)

$$\begin{array}{ll} \mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\ \mathcal{LRA}: & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\ Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7). \end{array}$$



$RESET_5$ Branch 1

$\mathcal{EUF}$ | $\mathcal{LRA}$

$$\begin{array}{ll} v_3 = h(v_0) & v_0 \geq v_1 \\ v_4 = h(v_1) & v_0 \leq v_1 \\ v_6 = f(v_2) & v_2 = v_3 - v_4 \\ v_7 = f(v_5) & v_5 = 0 \\ \neg(v_6 = v_7) & \langle e_{ij}\text{-deduction}\rangle \\ v_0 = v_1 \quad \Leftarrow & v_0 = v_1 \\ \langle e_{ij}\text{-deduction}\rangle & \\ v_3 = v_4 \quad \Rightarrow & v_3 = v_4 \\ & \langle e_{ij}\text{-deduction}\rangle \\ v_2 = v_5 \quad \Leftarrow & v_2 = v_5 \end{array}$$

$\bot$ $\mathcal{EUF} \cup \mathcal{LRA}$-Unsatisfiable!

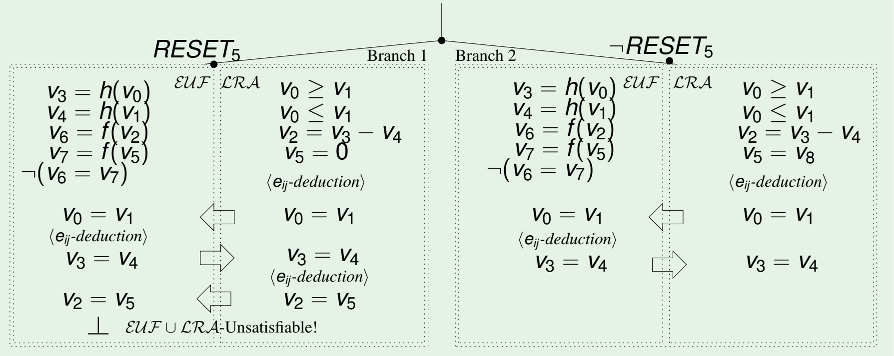
Branch 2 $\neg RESET_5$

$\mathcal{EUF}$ | $\mathcal{LRA}$

$$\begin{array}{ll} v_3 = h(v_0) & v_0 \geq v_1 \\ v_4 = h(v_1) & v_0 \leq v_1 \\ v_6 = f(v_2) & v_2 = v_3 - v_4 \\ v_7 = f(v_5) & v_5 = v_8 \\ \neg(v_6 = v_7) & \langle e_{ij}\text{-deduction}\rangle \\ v_0 = v_1 \quad \Leftarrow & v_0 = v_1 \\ \langle e_{ij}\text{-deduction}\rangle & \\ v_3 = v_4 \quad \Rightarrow & v_3 = v_4 \end{array}$$

$\mathcal{EUF} \cup \mathcal{LRA}$-Satisfiable!

N.O.: example (convex theory) [cont.]



$RESET_5$ Branch 1

$\mathcal{EUF}$ $\mathcal{LRA}$

$v_3 = h(v_0)$
$v_4 = h(v_1)$
$v_6 = f(v_2)$
$v_7 = f(v_5)$
$\neg(v_6 = v_7)$

$v_0 \geq v_1$
$v_0 \leq v_1$
$v_2 = v_3 - v_4$
$v_5 = 0$

$\langle e_{ij}\text{-deduction}\rangle$

$v_0 = v_1$ ⇦ $v_0 = v_1$

$\langle e_{ij}\text{-deduction}\rangle$

$v_3 = v_4$ ⇨ $v_3 = v_4$

$\langle e_{ij}\text{-deduction}\rangle$

$v_2 = v_5$ ⇦ $v_2 = v_5$

$\bot$ $\mathcal{EUF} \cup \mathcal{LRA}$-Unsatisfiable!

Branch 2 $\neg RESET_5$

$\mathcal{EUF}$ $\mathcal{LRA}$

$v_3 = h(v_0)$
$v_4 = h(v_1)$
$v_6 = f(v_2)$
$v_7 = f(v_5)$
$\neg(v_6 = v_7)$

$v_0 \geq v_1$
$v_0 \leq v_1$
$v_2 = v_3 - v_4$
$v_5 = v_8$

$\langle e_{ij}\text{-deduction}\rangle$

$v_0 = v_1$ ⇦ $v_0 = v_1$

$\langle e_{ij}\text{-deduction}\rangle$

$v_3 = v_4$ ⇨ $v_3 = v_4$

$$
\begin{array}{ll}
\mathcal{EUF}\text{-conflict :} & ((v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \neg(v_6 = v_7) \wedge (v_2 = v_5)) \rightarrow \bot \\
\mathcal{LRA}\text{-deduction :} & ((v_2 = v_3 - v_4) \wedge (v_5 = 0) \wedge (v_3 = v_4)) \rightarrow (v_2 = v_5) \\
\mathcal{EUF}\text{-deduction :} & ((v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_0 = v_1)) \rightarrow (v_3 = v_4) \\
\mathcal{LRA}\text{-deduction :} & ((v_0 \geq v_1) \wedge (v_0 \leq v_1)) \rightarrow (v_0 = v_1) \\
\Longrightarrow & \\
\mathcal{EUF} \cup \mathcal{LRA}\text{-conflict :} & ((v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \neg(v_6 = v_7) \wedge (v_2 = v_3 - v_4) \wedge \\
 & (v_5 = 0) \wedge (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_0 \geq v_1)) \rightarrow \bot.
\end{array}
$$

Exercises

For the previous N.O. example:

- write the (minimal) clauses corresponding to each e_{ij}-deduction
- find the final conflict clauses by resolving the e_{ij}-deduction clauses

Exercises

For the previous N.O. example:

- write the (minimal) clauses corresponding to each e_{ij}-deduction
- find the final conflict clauses by resolving the e_{ij}-deduction clauses

N.O.: example (non-convex theory)

$\mu_{\mathcal{LIA}}$

$$v_1 \geq 0$$
$$v_1 \leq 1$$
$$v_2 \geq v_6$$
$$v_2 \leq v_6 + 1$$

$$v_5 = v_4 - 1$$
$$v_3 = 0$$
$$v_4 = 1$$

$\mu_{\mathcal{EUF}}$

$$\neg(f(v_1) = f(v_2))$$
$$\neg(f(v_2) = f(v_4))$$
$$f(v_3) = v_5$$
$$f(v_1) = v_6$$

N.O.: example (non-convex theory)

$\mu_{\mathcal{LIA}}$

$$\begin{array}{ll} v_1 \geq 0 & v_5 = v_4 - 1 \\ v_1 \leq 1 & v_3 = 0 \\ v_2 \geq v_6 & v_4 = 1 \\ v_2 \leq v_6 + 1 & \end{array}$$

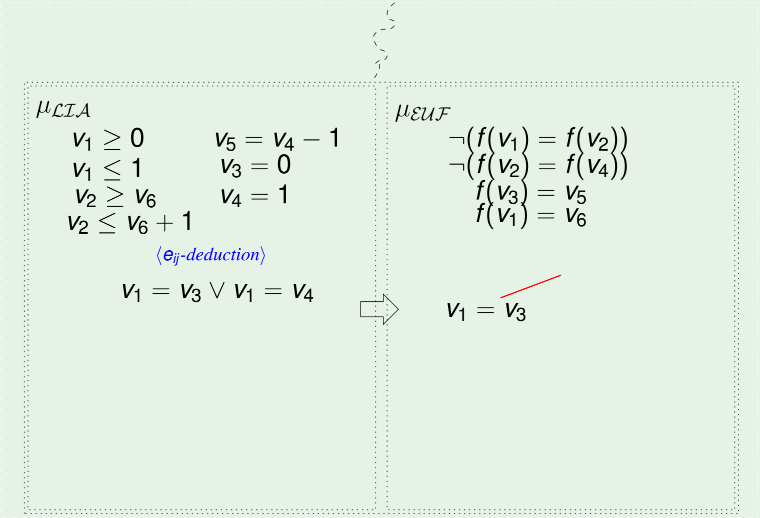
$\langle e_{ij}$*-deduction*$\rangle$

$$v_1 = v_3 \vee v_1 = v_4$$

$\mu_{\mathcal{EUF}}$

$$\begin{array}{l} \neg(f(v_1) = f(v_2)) \\ \neg(f(v_2) = f(v_4)) \\ f(v_3) = v_5 \\ f(v_1) = v_6 \end{array}$$

N.O.: example (non-convex theory)



$\mu_{\mathcal{LIA}}$

$$\begin{array}{rcl} v_1 \geq 0 & & v_5 = v_4 - 1 \\ v_1 \leq 1 & & v_3 = 0 \\ v_2 \geq v_6 & & v_4 = 1 \\ v_2 \leq v_6 + 1 & & \end{array}$$

$\langle e_{ij}\textit{-deduction}\rangle$

$$v_1 = v_3 \vee v_1 = v_4$$

$\mu_{\mathcal{EUF}}$

$$\begin{array}{c} \neg(f(v_1) = f(v_2)) \\ \neg(f(v_2) = f(v_4)) \\ f(v_3) = v_5 \\ f(v_1) = v_6 \end{array}$$

$$v_1 = v_3$$

N.O.: example (non-convex theory)

$\mu_{\mathcal{LIA}}$

$$\begin{array}{rcl@{\qquad}rcl} v_1 & \geq & 0 & v_5 & = & v_4 - 1 \\ v_1 & \leq & 1 & v_3 & = & 0 \\ v_2 & \geq & v_6 & v_4 & = & 1 \\ v_2 & \leq & v_6 + 1 & & & \end{array}$$

$\langle e_{ij}\textit{-deduction}\rangle$

$$v_1 = v_3 \vee v_1 = v_4$$

$\Rightarrow$

$\mu_{\mathcal{EUF}}$

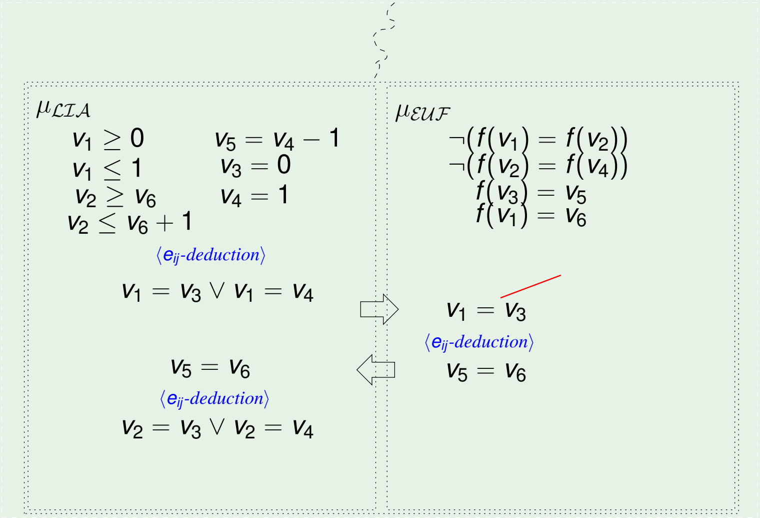
$$\begin{array}{l} \neg(f(v_1) = f(v_2)) \\ \neg(f(v_2) = f(v_4)) \\ f(v_3) = v_5 \\ f(v_1) = v_6 \end{array}$$

$$v_1 = v_3$$

$\langle e_{ij}\textit{-deduction}\rangle$

$$v_5 = v_6$$

N.O.: example (non-convex theory)



$\mu_{\mathcal{LIA}}$

$$v_1 \geq 0 \qquad v_5 = v_4 - 1$$
$$v_1 \leq 1 \qquad v_3 = 0$$
$$v_2 \geq v_6 \qquad v_4 = 1$$
$$v_2 \leq v_6 + 1$$

$\langle e_{ij}$*-deduction*$\rangle$

$$v_1 = v_3 \vee v_1 = v_4$$

$\mu_{\mathcal{EUF}}$

$$\neg(f(v_1) = f(v_2))$$
$$\neg(f(v_2) = f(v_4))$$
$$f(v_3) = v_5$$
$$f(v_1) = v_6$$

$$v_1 = v_3$$

$\langle e_{ij}$*-deduction*$\rangle$

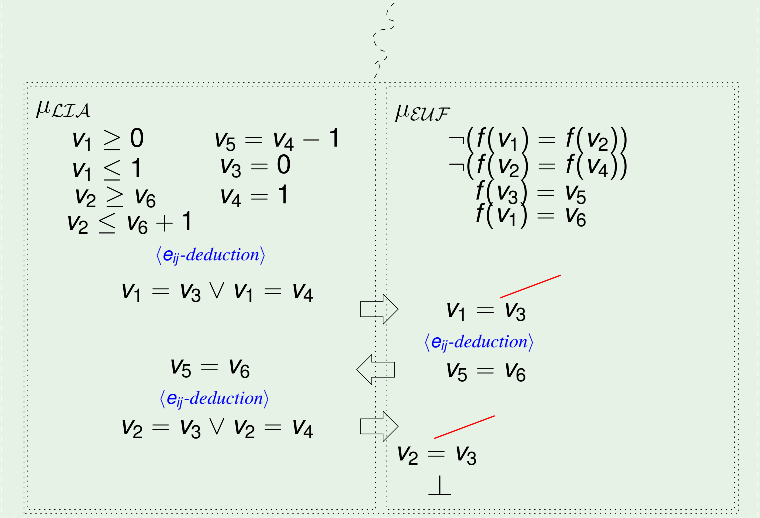
$$v_5 = v_6$$

$$v_5 = v_6$$

$\langle e_{ij}$*-deduction*$\rangle$

$$v_2 = v_3 \vee v_2 = v_4$$

N.O.: example (non-convex theory)



$\mu_{\mathcal{LIA}}$

$v_1 \geq 0$ $\quad v_5 = v_4 - 1$

$v_1 \leq 1$ $\quad v_3 = 0$

$v_2 \geq v_6$ $\quad v_4 = 1$

$v_2 \leq v_6 + 1$

$\mu_{\mathcal{EUF}}$

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\langle e_{ij}\textit{-deduction}\rangle$

$v_1 = v_3 \vee v_1 = v_4$ $\Rightarrow$ $v_1 = v_3$

$\langle e_{ij}\textit{-deduction}\rangle$

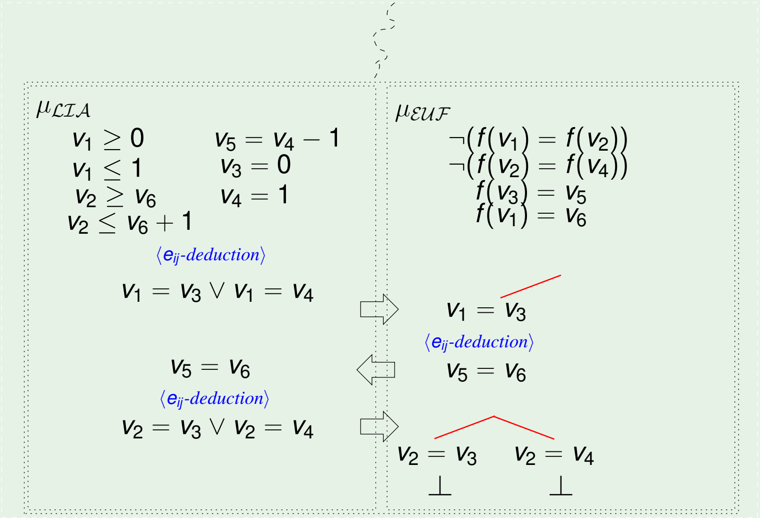
$v_5 = v_6$ $\Leftarrow$ $v_5 = v_6$

$\langle e_{ij}\textit{-deduction}\rangle$

$v_2 = v_3 \vee v_2 = v_4$ $\Rightarrow$ $v_2 = v_3$

$\bot$

N.O.: example (non-convex theory)



$\mu_{\mathcal{LIA}}$

$$v_1 \geq 0 \qquad v_5 = v_4 - 1$$
$$v_1 \leq 1 \qquad v_3 = 0$$
$$v_2 \geq v_6 \qquad v_4 = 1$$
$$v_2 \leq v_6 + 1$$

$\langle e_{ij}$*-deduction*$\rangle$

$$v_1 = v_3 \vee v_1 = v_4$$

$$v_5 = v_6$$

$\langle e_{ij}$*-deduction*$\rangle$

$$v_2 = v_3 \vee v_2 = v_4$$

$\mu_{\mathcal{EUF}}$

$$\neg(f(v_1) = f(v_2))$$
$$\neg(f(v_2) = f(v_4))$$
$$f(v_3) = v_5$$
$$f(v_1) = v_6$$

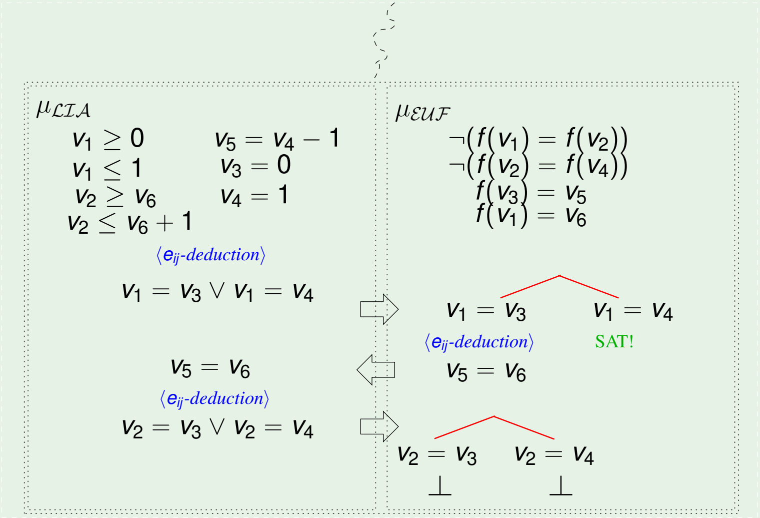
$$v_1 = v_3$$

$\langle e_{ij}$*-deduction*$\rangle$

$$v_5 = v_6$$

$$v_2 = v_3 \qquad v_2 = v_4$$
$$\bot \qquad \bot$$

N.O.: example (non-convex theory)



$\mu_{\mathcal{LIA}}$

$$
\begin{array}{ll}
v_1 \geq 0 & v_5 = v_4 - 1 \\
v_1 \leq 1 & v_3 = 0 \\
v_2 \geq v_6 & v_4 = 1 \\
v_2 \leq v_6 + 1 &
\end{array}
$$

$\langle e_{ij}$*-deduction*$\rangle$

$$v_1 = v_3 \vee v_1 = v_4$$

$$v_5 = v_6$$

$\langle e_{ij}$*-deduction*$\rangle$

$$v_2 = v_3 \vee v_2 = v_4$$

$\mu_{\mathcal{EUF}}$

$$
\begin{array}{c}
\neg(f(v_1) = f(v_2)) \\
\neg(f(v_2) = f(v_4)) \\
f(v_3) = v_5 \\
f(v_1) = v_6
\end{array}
$$

$v_1 = v_3$ $\qquad$ $v_1 = v_4$

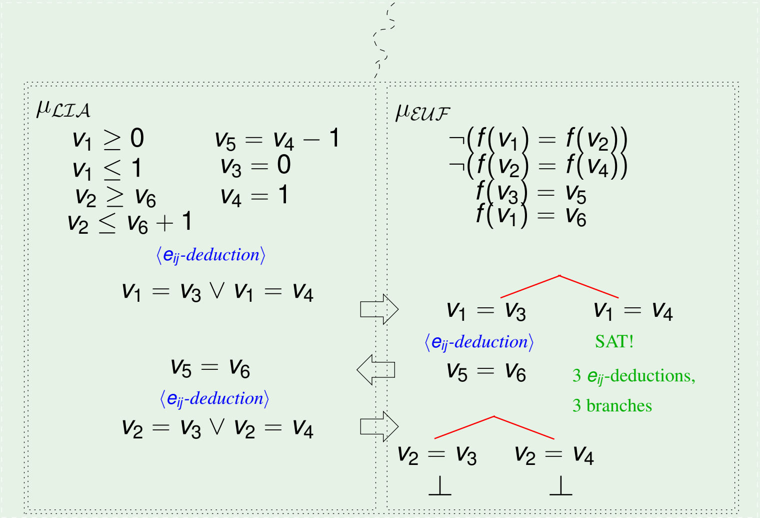
$\langle e_{ij}$*-deduction*$\rangle$ $\qquad$ SAT!

$v_5 = v_6$

$v_2 = v_3$ $\qquad$ $v_2 = v_4$

$\bot$ $\qquad$ $\bot$

N.O.: example (non-convex theory)



$\mu_{\mathcal{LIA}}$

$v_1 \geq 0$

$v_1 \leq 1$

$v_2 \geq v_6$

$v_2 \leq v_6 + 1$

$v_5 = v_4 - 1$

$v_3 = 0$

$v_4 = 1$

$\mu_{\mathcal{EUF}}$

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\langle e_{ij}\text{-deduction}\rangle$

$v_1 = v_3 \vee v_1 = v_4$

$v_1 = v_3$ $\quad$ $v_1 = v_4$

$\langle e_{ij}\text{-deduction}\rangle$ $\quad$ SAT!

$v_5 = v_6$

3 e_{ij}-deductions,

3 branches

$v_5 = v_6$

$\langle e_{ij}\text{-deduction}\rangle$

$v_2 = v_3 \vee v_2 = v_4$

$v_2 = v_3$ $\quad$ $v_2 = v_4$

$\bot$ $\quad$ $\bot$

$SMT(\bigcup_i \mathcal{T}_i)$ via "classic" Nelson-Oppen

Main idea

Combine two or more $\mathcal{T}_i$-*solvers* into one $(\bigcup_i \mathcal{T}_i)$-solver via Nelson-Oppen/Shostak (N.O.) combination procedure [62, 76]

- based on the deduction and exchange of equalities between shared variables/terms (interface equalities, e_{ij}s)
- important improvements and evolutions [68, 7, 39]
- drawbacks [23, 24]:
  - require (possibly expensive) deduction capabilities from $\mathcal{T}_i$-*solvers*
  - [with non-convex theories] case-splits forced by the deduction of disjunctions of e_{ij}'s
  - generate (typically long) $(\bigcup_i \mathcal{T}_i)$-lemmas, without interface equalities
    $\Longrightarrow$ no backjumping & learning from e_{ij}-reasoning

$SMT(\bigcup_i \mathcal{T}_i)$ via “classic” Nelson-Oppen

Main idea

Combine two or more $\mathcal{T}_i$-*solvers* into one $(\bigcup_i \mathcal{T}_i)$-solver via Nelson-Oppen/Shostak (N.O.) combination procedure [62, 76]

- based on the deduction and exchange of equalities between shared variables/terms (interface equalities, e_{ij}s)
- important improvements and evolutions [68, 7, 39]
- drawbacks [23, 24]:
  - require (possibly expensive) deduction capabilities from $\mathcal{T}_i$-*solvers*
  - [with non-convex theories] case-splits forced by the deduction of disjunctions of e_{ij}'s
  - generate (typically long) $(\bigcup_i \mathcal{T}_i)$-lemmas, without interface equalities $\Longrightarrow$ no backjumping & learning from e_{ij}-reasoning

Exercise (cont.)

- Purify the following $\mathcal{LIA} \cup \mathcal{EUF} \cup \mathcal{AR}$-formula (see beginning of chapter):

$$\varphi \stackrel{\text{def}}{=} (d \geq 0) \wedge (d < 1) \wedge \\ ((f(d) = f(0)) \rightarrow (read(write(V, i, x), i + d) = x + 1))$$

- Solve it via Nelson-Oppen procedure

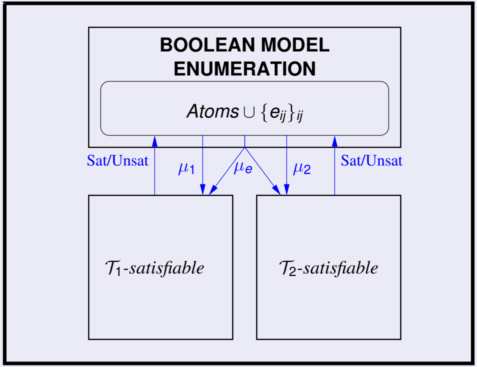
$SMT(\bigcup_i \mathcal{T}_i)$ via Delayed Theory Combination (DTC)

Main idea

Delegate to the CDCL SAT solver part/most of the (possibly very expensive) reasoning effort on interface equalities previously due to the $\mathcal{T}_i$-*solvers* (e_{ij}-deduction, case-split). [15, 16, 24]

- based on Boolean reasoning on interface equalities via CDCL (plus $\mathcal{T}$-propagation)
- important improvements and evolutions [36, 9]
- feature wrt N.O. [23, 24]
  - do not require (possibly expensive) deduction capabilities from $\mathcal{T}_i$-*solvers*
  - with non-convex theories, case-splits on e_{ij}'s handled by SAT solver
  - generate $\mathcal{T}_i$-lemmas with interface equalities
    $\Longrightarrow$ backjumping & learning from e_{ij}-reasoning

DTC: Basic schema



BOOLEAN MODEL ENUMERATION

$Atoms \cup \{e_{ij}\}_{ij}$

Sat/Unsat

μ_1

μ_e

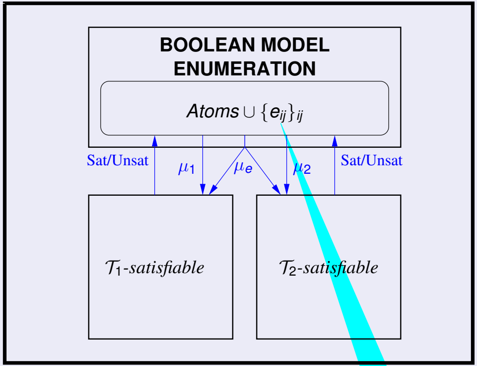
μ_2

Sat/Unsat

$\mathcal{T}_1$-*satisfiable*

$\mathcal{T}_2$-*satisfiable*

DTC: Basic schema



BOOLEAN MODEL ENUMERATION

$Atoms \cup \{e_{ij}\}_{ij}$

Sat/Unsat

μ_1

μ_e

μ_2

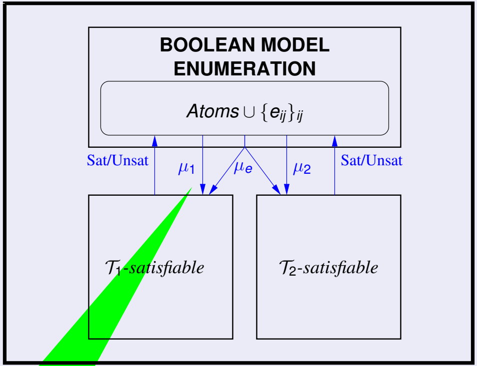
Sat/Unsat

$\mathcal{T}_1$*-satisfiable*

$\mathcal{T}_2$*-satisfiable*

The boolean solver assigns values not only to atoms in $Atoms(\phi)$, but also to interface equalities $\{(v_i = v_j)\}_{ij}$:
$\mu = \mu_1 \cup \mu_2 \cup \mu_e, \quad \mu_e := \{[\neg](v_i = v_j) | v_i, v_j \in \mu_1 \cup \mu_2\}$

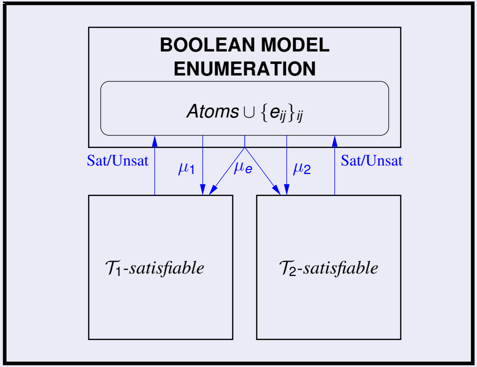
DTC: Basic schema



Each $\mathcal{T}_i$-*solver* interacts only with the boolean solver

- receives $\mu_i' := \mu_i \cup \mu_e$ from Bool
- checks the $\mathcal{T}_i$-satisfiability of μ_i'

DTC: Basic schema



...until either:

- some μ propositionally satisfies ϕ and both $\mu_i' := \mu_i \cup \mu_e$ are T_i-consistent

$\Longrightarrow$ (ϕ is $\mathcal{T}_1 \cup \mathcal{T}_2$-sat)

- no more assignment μ are available

$\Longrightarrow$ (ϕ is $\mathcal{T}_1 \cup \mathcal{T}_2$-unsat)

DTC: enhanced schema

- CDCL-based assignment enumeration on $Atoms(\phi) \cup \{e_{ij}\}_{ij}$,
  $\Longrightarrow$ benefits of state-of-the-art SAT techniques
- Early pruning: invoke the $\mathcal{T}_i$*-solver*'s before every Boolean decision
  $\Longrightarrow$ total assignments generated only when strictly necessary
- Branching: branching on e_{ij}'s postponed
  $\Longrightarrow$ Boolean search on e_{ij}'s performed only when strictly necessary
- Theory-Backjumping & Learning: e_{ij}'s are involved in conflicts
  $\Longrightarrow$ e_{ij}'s can be assigned by unit propagation
- Theory-deduction & learning: if $\mathcal{T}_i$*-solver* deduces unassigned literals l on $Atoms(\phi) \cup \{e_{ij}\}_{ij}$
  - l is passed back to the Boolean solver, which unit-propagates it
  - the deduction $\mu' \models l$ is learned as a clause $\mu' \rightarrow l$ (deduction clause)
- ...

DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)

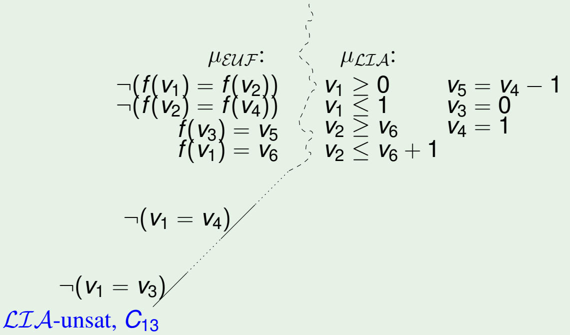
$\mu_{\mathcal{EUF}}$:

$$\begin{array}{r}
\neg(f(v_1) = f(v_2)) \\
\neg(f(v_2) = f(v_4)) \\
f(v_3) = v_5 \\
f(v_1) = v_6
\end{array}$$

$\mu_{\mathcal{LIA}}$:

$$\begin{array}{ll}
v_1 \geq 0 & v_5 = v_4 - 1 \\
v_1 \leq 1 & v_3 = 0 \\
v_2 \geq v_6 & v_4 = 1 \\
v_2 \leq v_6 + 1 &
\end{array}$$

DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$ $\quad v_5 = v_4 - 1$

$v_1 \leq 1$ $\quad v_3 = 0$

$v_2 \geq v_6$ $\quad v_4 = 1$

$v_2 \leq v_6 + 1$

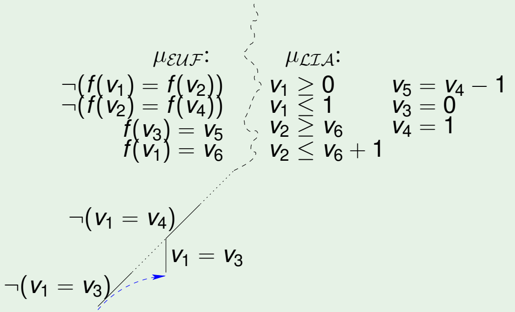
$\neg(v_1 = v_4)$

$\neg(v_1 = v_3)$

$\mathcal{LIA}$-unsat, C_{13}

$$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$$

DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$ $\quad v_5 = v_4 - 1$

$v_1 \leq 1$ $\quad v_3 = 0$

$v_2 \geq v_6$ $\quad v_4 = 1$

$v_2 \leq v_6 + 1$

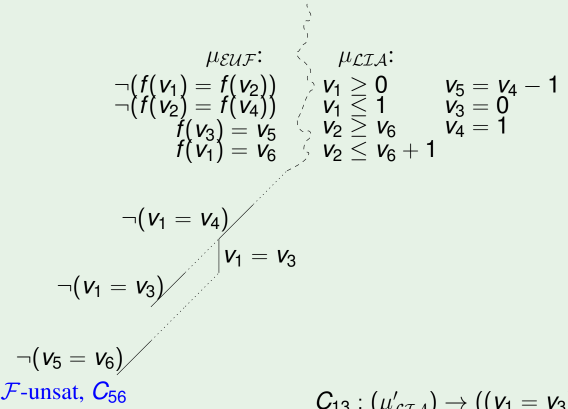
$\neg(v_1 = v_4)$

$v_1 = v_3$

$\neg(v_1 = v_3)$

$$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$$

DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$

$v_1 \leq 1$

$v_2 \geq v_6$

$v_2 \leq v_6 + 1$

$v_5 = v_4 - 1$

$v_3 = 0$

$v_4 = 1$

$\neg(v_1 = v_4)$

$v_1 = v_3$

$\neg(v_1 = v_3)$

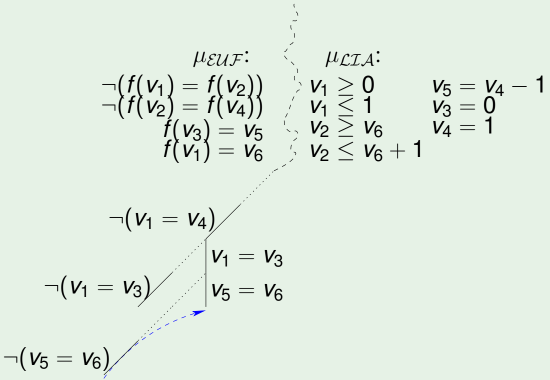
$\neg(v_5 = v_6)$

$\mathcal{EUF}$-unsat, C_{56}

$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$

$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$

DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$
$\neg(f(v_2) = f(v_4))$
$f(v_3) = v_5$
$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$	$v_5 = v_4 - 1$
$v_1 \leq 1$	$v_3 = 0$
$v_2 \geq v_6$	$v_4 = 1$
$v_2 \leq v_6 + 1$	

$\neg(v_1 = v_4)$

$v_1 = v_3$

$\neg(v_1 = v_3)$

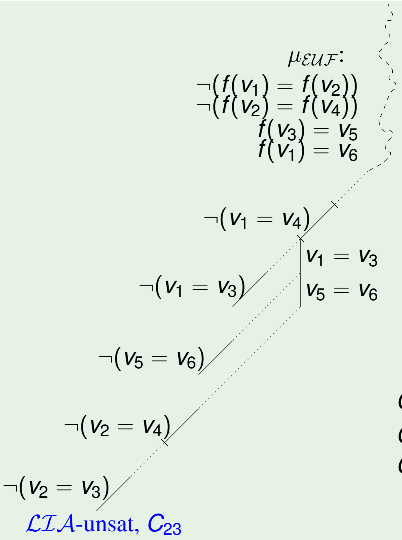
$v_5 = v_6$

$\neg(v_5 = v_6)$

$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$

$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$

DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$
$\neg(f(v_2) = f(v_4))$
$f(v_3) = v_5$
$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$	$v_5 = v_4 - 1$
$v_1 \leq 1$	$v_3 = 0$
$v_2 \geq v_6$	$v_4 = 1$
$v_2 \leq v_6 + 1$	

$\neg(v_1 = v_4)$

$v_1 = v_3$

$\neg(v_1 = v_3)$

$v_5 = v_6$

$\neg(v_5 = v_6)$

$\neg(v_2 = v_4)$

$\neg(v_2 = v_3)$

$\mathcal{LIA}$-unsat, C_{23}

$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$
$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$
$C_{23} : (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$

DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)

$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$

$v_1 \leq 1$

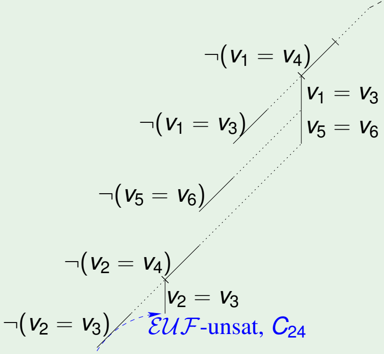
$v_2 \geq v_6$

$v_2 \leq v_6 + 1$

$v_5 = v_4 - 1$

$v_3 = 0$

$v_4 = 1$



$\neg(v_1 = v_4)$

$v_1 = v_3$

$v_5 = v_6$

$\neg(v_1 = v_3)$

$\neg(v_5 = v_6)$

$\neg(v_2 = v_4)$

$v_2 = v_3$

$\neg(v_2 = v_3)$

$\mathcal{EUF}$-unsat, C_{24}

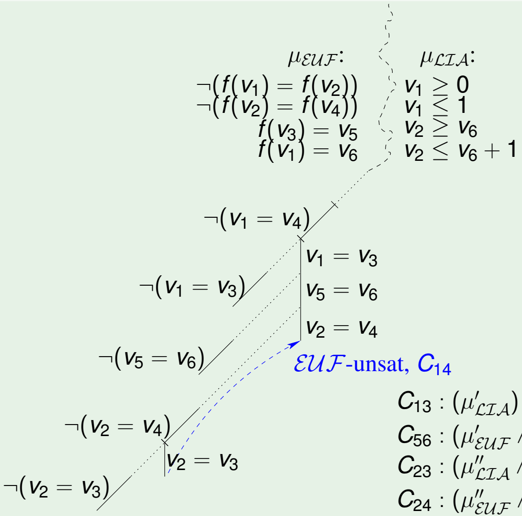
$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$

$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$

$C_{23} : (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$

$C_{24} : (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot$

DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$ $\quad v_5 = v_4 - 1$

$v_1 \leq 1$ $\quad v_3 = 0$

$v_2 \geq v_6$ $\quad v_4 = 1$

$v_2 \leq v_6 + 1$

$\neg(v_1 = v_4)$

$v_1 = v_3$

$\neg(v_1 = v_3)$

$v_5 = v_6$

$\neg(v_5 = v_6)$

$v_2 = v_4$

$\mathcal{EUF}$-unsat, C_{14}

$\neg(v_2 = v_4)$

$v_2 = v_3$

$\neg(v_2 = v_3)$

$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$

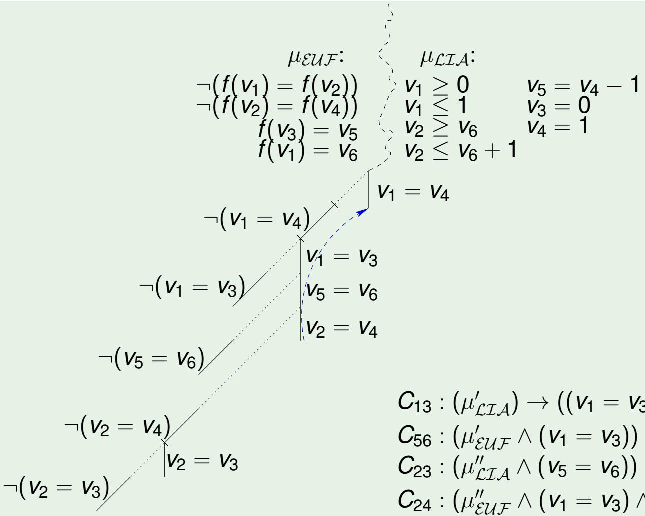
$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$

$C_{23} : (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$

$C_{24} : (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot$

$C_{14} : (\mu'''_{\mathcal{EUF}} \wedge (v_2 = v_4)) \rightarrow \bot$

DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

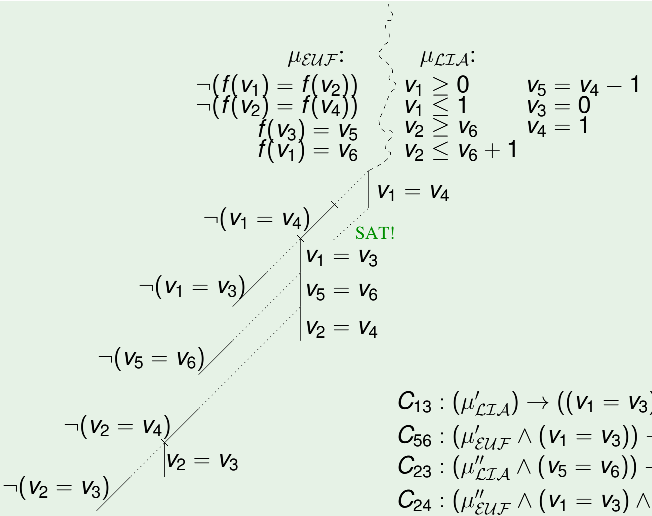
$\neg(f(v_1) = f(v_2))$
$\neg(f(v_2) = f(v_4))$
$f(v_3) = v_5$
$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$
$v_1 \leq 1$
$v_2 \geq v_6$
$v_2 \leq v_6 + 1$
$v_5 = v_4 - 1$
$v_3 = 0$
$v_4 = 1$

$$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$$
$$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$$
$$C_{23} : (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$$
$$C_{24} : (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot$$
$$C_{14} : (\mu'''_{\mathcal{EUF}} \wedge (v_2 = v_4)) \rightarrow \bot$$

DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$ $\quad v_5 = v_4 - 1$

$v_1 \leq 1$ $\quad v_3 = 0$

$v_2 \geq v_6$ $\quad v_4 = 1$

$v_2 \leq v_6 + 1$

$v_1 = v_4$

$\neg(v_1 = v_4)$

SAT!

$v_1 = v_3$

$\neg(v_1 = v_3)$

$v_5 = v_6$

$\neg(v_5 = v_6)$

$v_2 = v_4$

$\neg(v_2 = v_4)$

$v_2 = v_3$

$\neg(v_2 = v_3)$

$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$

$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$

$C_{23} : (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$

$C_{24} : (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot$

$C_{14} : (\mu'''_{\mathcal{EUF}} \wedge (v_2 = v_4)) \rightarrow \bot$

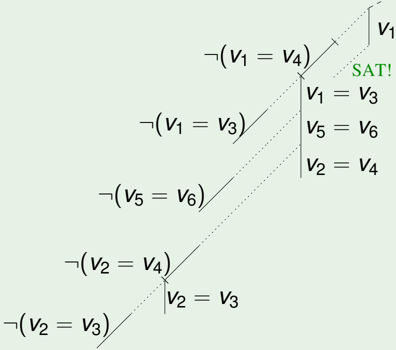
DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)

$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$
$\neg(f(v_2) = f(v_4))$
$f(v_3) = v_5$
$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$
$v_1 \leq 1$
$v_2 \geq v_6$
$v_2 \leq v_6 + 1$
$v_5 = v_4 - 1$
$v_3 = 0$
$v_4 = 1$



$v_1 = v_4$
$\neg(v_1 = v_4)$
SAT!
$v_1 = v_3$
$\neg(v_1 = v_3)$
$v_5 = v_6$
$\neg(v_5 = v_6)$
$v_2 = v_4$
$\neg(v_2 = v_4)$
$v_2 = v_3$
$\neg(v_2 = v_3)$

6 branches

$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$
$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$
$C_{23} : (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$
$C_{24} : (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot$
$C_{14} : (\mu'''_{\mathcal{EUF}} \wedge (v_2 = v_4)) \rightarrow \bot$

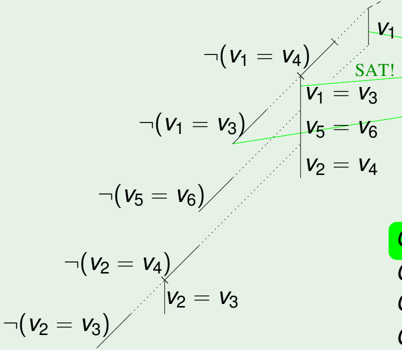
DTC: example w.out $\mathcal{T}$-prop. (non-convex theory)

$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$
$\neg(f(v_2) = f(v_4))$
$f(v_3) = v_5$
$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$ $\quad v_5 = v_4 - 1$
$v_1 \leq 1$ $\quad v_3 = 0$
$v_2 \geq v_6$ $\quad v_4 = 1$
$v_2 \leq v_6 + 1$



$v_1 = v_4$

$\neg(v_1 = v_4)$

SAT!

$v_1 = v_3$

$\neg(v_1 = v_3)$

$v_5 = v_6$

$v_2 = v_4$

$\neg(v_5 = v_6)$

$\neg(v_2 = v_4)$

$v_2 = v_3$

$\neg(v_2 = v_3)$

Mimics the e_{ij}-deduction

$\mu'_{\mathcal{LIA}} \models_{\mathcal{LIA}} ((v_1 = v_3) \vee (v_1 = v_4))$

and the two branches $(v_1 = v_3)$, $(v_1 = v_4)$

$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$
$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$
$C_{23} : (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$
$C_{24} : (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot$
$C_{14} : (\mu'''_{\mathcal{EUF}} \wedge (v_2 = v_4)) \rightarrow \bot$

DTC: example with $\mathcal{T}$-prop. (non-convex theory)

$\mu_{\mathcal{EUF}}$:

$$\begin{array}{r}
\neg(f(v_1) = f(v_2)) \\
\neg(f(v_2) = f(v_4)) \\
f(v_3) = v_5 \\
f(v_1) = v_6
\end{array}$$

$\mu_{\mathcal{LIA}}$:

$$\begin{array}{ll}
v_1 \geq 0 & v_5 = v_4 - 1 \\
v_1 \leq 1 & v_3 = 0 \\
v_2 \geq v_6 & v_4 = 1 \\
v_2 \leq v_6 + 1 &
\end{array}$$

DTC: example with $\mathcal{T}$-prop. (non-convex theory)

$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$ $\quad v_5 = v_4 - 1$

$v_1 \leq 1$ $\quad v_3 = 0$

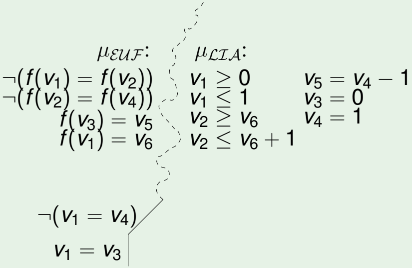
$v_2 \geq v_6$ $\quad v_4 = 1$

$v_2 \leq v_6 + 1$

$\mathcal{LIA}$-deduce $(v_1 = v_4) \vee (v_1 = v_3)$, C_{13}

$$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$$

DTC: example with $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$ $\quad$ $v_5 = v_4 - 1$

$v_1 \leq 1$ $\quad$ $v_3 = 0$

$v_2 \geq v_6$ $\quad$ $v_4 = 1$

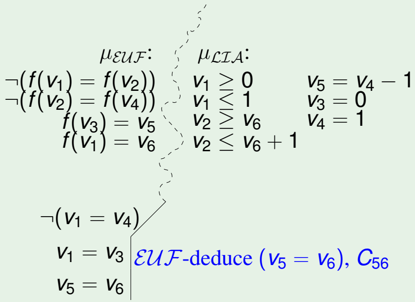
$v_2 \leq v_6 + 1$

$\neg(v_1 = v_4)$

$v_1 = v_3$

$$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$$

DTC: example with $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$
$\neg(f(v_2) = f(v_4))$
$f(v_3) = v_5$
$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$
$v_1 \leq 1$
$v_2 \geq v_6$
$v_2 \leq v_6 + 1$

$v_5 = v_4 - 1$
$v_3 = 0$
$v_4 = 1$

$\neg(v_1 = v_4)$

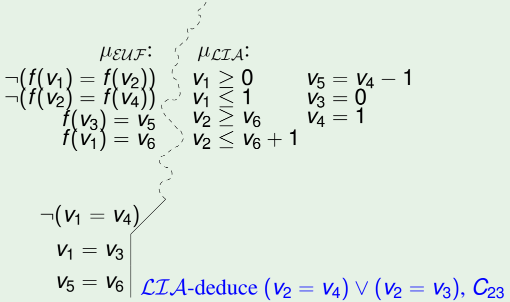
$v_1 = v_3$

$v_5 = v_6$

$\mathcal{EUF}$-deduce $(v_5 = v_6)$, C_{56}

$$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$$
$$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$$

DTC: example with $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$

$v_1 \leq 1$

$v_2 \geq v_6$

$v_2 \leq v_6 + 1$

$v_5 = v_4 - 1$

$v_3 = 0$

$v_4 = 1$

$\neg(v_1 = v_4)$

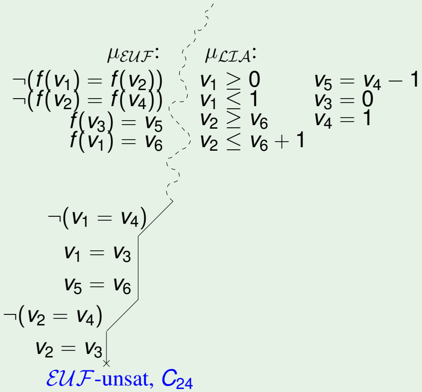
$v_1 = v_3$

$v_5 = v_6$

$\mathcal{LIA}$-deduce $(v_2 = v_4) \vee (v_2 = v_3)$, C_{23}

$$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$$
$$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$$
$$C_{23} : (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$$

DTC: example with $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$

$\neg(f(v_2) = f(v_4))$

$f(v_3) = v_5$

$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$ $\quad v_5 = v_4 - 1$

$v_1 \leq 1$ $\quad v_3 = 0$

$v_2 \geq v_6$ $\quad v_4 = 1$

$v_2 \leq v_6 + 1$

$\neg(v_1 = v_4)$

$v_1 = v_3$

$v_5 = v_6$

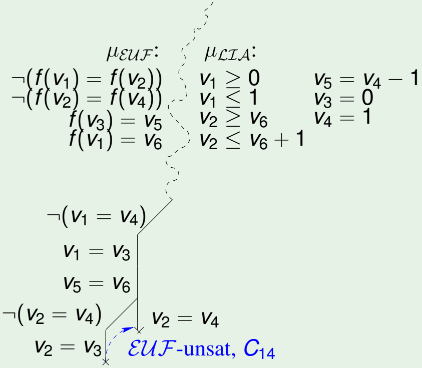
$\neg(v_2 = v_4)$

$v_2 = v_3$

$\mathcal{EUF}$-unsat, C_{24}

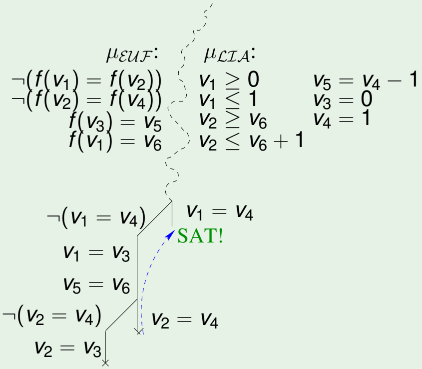
$$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$$
$$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$$
$$C_{23} : (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$$
$$C_{24} : (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot$$

DTC: example with $\mathcal{T}$-prop. (non-convex theory)



$$
\begin{aligned}
C_{13} &: (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4)) \\
C_{56} &: (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6) \\
C_{23} &: (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4)) \\
C_{24} &: (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot \\
C_{14} &: (\mu'''_{\mathcal{EUF}} \wedge (v_2 = v_4)) \rightarrow \bot
\end{aligned}
$$

DTC: example with $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$
$\neg(f(v_2) = f(v_4))$
$f(v_3) = v_5$
$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

$v_1 \geq 0$ $\quad v_5 = v_4 - 1$
$v_1 \leq 1$ $\quad v_3 = 0$
$v_2 \geq v_6$ $\quad v_4 = 1$
$v_2 \leq v_6 + 1$

$\neg(v_1 = v_4)$ $\quad v_1 = v_4$

SAT!

$v_1 = v_3$

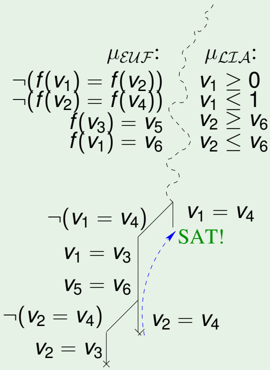
$v_5 = v_6$

$\neg(v_2 = v_4)$ $\quad v_2 = v_4$

$v_2 = v_3$

$$
\begin{aligned}
C_{13} &: (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4)) \\
C_{56} &: (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6) \\
C_{23} &: (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4)) \\
C_{24} &: (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot \\
C_{14} &: (\mu'''_{\mathcal{EUF}} \wedge (v_2 = v_4)) \rightarrow \bot
\end{aligned}
$$

DTC: example with $\mathcal{T}$-prop. (non-convex theory)



$\mu_{\mathcal{EUF}}$:

$\neg(f(v_1) = f(v_2))$
$\neg(f(v_2) = f(v_4))$
$f(v_3) = v_5$
$f(v_1) = v_6$

$\mu_{\mathcal{LIA}}$:

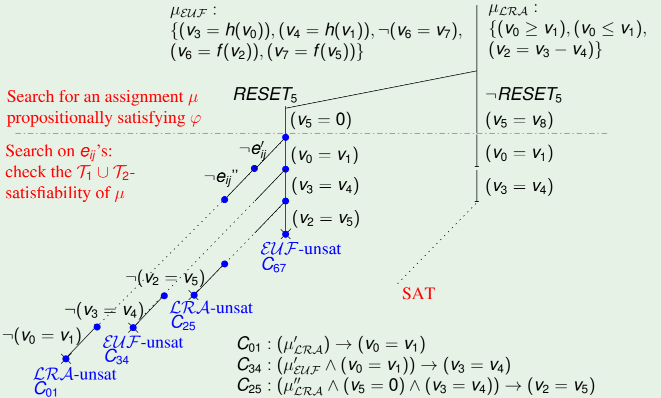
$v_1 \geq 0$
$v_1 \leq 1$
$v_2 \geq v_6$
$v_2 \leq v_6 + 1$
$v_5 = v_4 - 1$
$v_3 = 0$
$v_4 = 1$

3 e_{ij}-deductions
3 branches

$$C_{13} : (\mu'_{\mathcal{LIA}}) \rightarrow ((v_1 = v_3) \vee (v_1 = v_4))$$
$$C_{56} : (\mu'_{\mathcal{EUF}} \wedge (v_1 = v_3)) \rightarrow (v_5 = v_6)$$
$$C_{23} : (\mu''_{\mathcal{LIA}} \wedge (v_5 = v_6)) \rightarrow ((v_2 = v_3) \vee (v_2 = v_4))$$
$$C_{24} : (\mu''_{\mathcal{EUF}} \wedge (v_1 = v_3) \wedge (v_2 = v_3)) \rightarrow \bot$$
$$C_{14} : (\mu'''_{\mathcal{EUF}} \wedge (v_2 = v_4)) \rightarrow \bot$$

DTC: example without $\mathcal{T}$-propagation (convex theory)

$$
\begin{array}{ll}
\mathcal{EUF}: & (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \\
\mathcal{LRA}: & (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge \\
Both: & (\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).
\end{array}
$$



$\mu_{\mathcal{EUF}}$:
$\{(v_3 = h(v_0)), (v_4 = h(v_1)), \neg(v_6 = v_7),$
$(v_6 = f(v_2)), (v_7 = f(v_5))\}$

$\mu_{\mathcal{LRA}}$:
$\{(v_0 \geq v_1), (v_0 \leq v_1),$
$(v_2 = v_3 - v_4)\}$

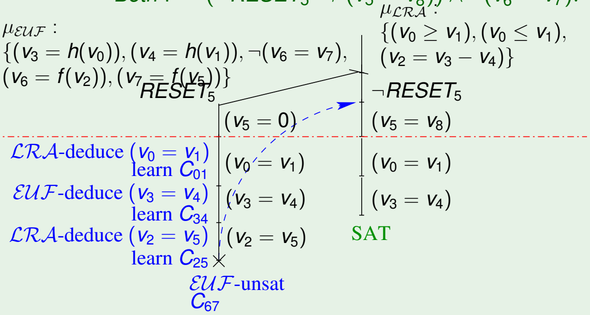
Search for an assignment μ propositionally satisfying φ

Search on e_{ij}'s: check the $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of μ

$C_{01} : (\mu'_{\mathcal{LRA}}) \rightarrow (v_0 = v_1)$
$C_{34} : (\mu'_{\mathcal{EUF}} \wedge (v_0 = v_1)) \rightarrow (v_3 = v_4)$
$C_{25} : (\mu''_{\mathcal{LRA}} \wedge (v_5 = 0) \wedge (v_3 = v_4)) \rightarrow (v_2 = v_5)$
$C_{67} : (\mu''_{\mathcal{EUF}} \wedge (v_2 = v_5)) \rightarrow (v_6 = v_7)$

DTC: example with $\mathcal{T}$-propagation (convex theory)

$\mathcal{EUF}$: $(v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge$
$\mathcal{LRA}$: $(v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge$
Both : $(\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7)$.



$\mu_{\mathcal{EUF}}$:
$\{(v_3 = h(v_0)), (v_4 = h(v_1)), \neg(v_6 = v_7),$
$(v_6 = f(v_2)), (v_7 = f(v_5))\}$

$\mu_{\mathcal{LRA}}$:
$\{(v_0 \geq v_1), (v_0 \leq v_1),$
$(v_2 = v_3 - v_4)\}$

$\mathcal{LRA}$-deduce $(v_0 = v_1)$
learn C_{01}
$\mathcal{EUF}$-deduce $(v_3 = v_4)$
learn C_{34}
$\mathcal{LRA}$-deduce $(v_2 = v_5)$
learn C_{25}
$\mathcal{EUF}$-unsat
C_{67}

$C_{01} : (\mu'_{\mathcal{LRA}}) \rightarrow (v_0 = v_1)$
$C_{34} : (\mu'_{\mathcal{EUF}} \wedge (v_0 = v_1)) \rightarrow (v_3 = v_4)$
$C_{25} : (\mu''_{\mathcal{LRA}} \wedge (v_5 = 0) \wedge (v_3 = v_4)) \rightarrow (v_2 = v_5)$
$C_{67} : (\mu''_{\mathcal{EUF}} \wedge (v_2 = v_5)) \rightarrow (v_6 = v_7)$

DTC + Model-based heuristic (aka Model-Based Theory Combination) [36]

- Initially, no interface equalities generated
- When a model is found, check against all the possible interface equalities
  - If $\mathcal{T}_1$ and $\mathcal{T}_2$ agree on the implied equalities, then return SAT
  - Otherwise, branch on equalities implied by $\mathcal{T}_1$-model but not by $\mathcal{T}_2$-model
- "Optimistic" approach, similar to axiom instantiation

Exercises

For each of the previous DTC examples:

- write the (minimal) clauses corresponding to each e_{ij}-deduction (as clauses rather than as implications)
- compute the conflict-analysis steps leading to the backjumping steps in the figures.

For each of the previous DTC examples, draw the case in which the $\mathcal{EUF}$-solver has deduction capabilities and the $\mathcal{LRA}$-solver (resp. the $\mathcal{LIA}$-solver) does not.

Exercises

For each of the previous DTC examples:

- write the (minimal) clauses corresponding to each e_{ij}-deduction (as clauses rather than as implications)
- compute the conflict-analysis steps leading to the backjumping steps in the figures.

For each of the previous DTC examples, draw the case in which the $\mathcal{EUF}$-solver has deduction capabilities and the $\mathcal{LRA}$-solver (resp. the $\mathcal{LIA}$-solver) does not.

Exercises

For each of the previous DTC examples:

- write the (minimal) clauses corresponding to each e_{ij}-deduction (as clauses rather than as implications)
- compute the conflict-analysis steps leading to the backjumping steps in the figures.

For each of the previous DTC examples, draw the case in which the $\mathcal{EUF}$-solver has deduction capabilities and the $\mathcal{LRA}$-solver (resp. the $\mathcal{LIA}$-solver) does not.

Exercises

For each of the previous DTC examples:

- write the (minimal) clauses corresponding to each e_{ij}-deduction (as clauses rather than as implications)
- compute the conflict-analysis steps leading to the backjumping steps in the figures.

For each of the previous DTC examples, draw the case in which the $\mathcal{EUF}$-solver has deduction capabilities and the $\mathcal{LRA}$-solver (resp. the $\mathcal{LIA}$-solver) does not.

Exercise (cont.)

- Purify the following $\mathcal{LIA} \cup \mathcal{EUF} \cup \mathcal{AR}$-formula (see beginning of chapter):

$$\begin{array}{l}\varphi \stackrel{\text{def}}{=} (d \geq 0) \wedge (d < 1) \wedge \\ ((f(d) = f(0)) \rightarrow (read(write(V, i, x), i + d) = x + 1))\end{array}$$

- Solve it via the DTC procedure
  - assuming no $\mathcal{T}$-deduction capabilities of the $\mathcal{T}$-solvers
  - assuming $\mathcal{T}$-deduction capabilities of the $\mathcal{T}$-solvers

Outline

1. Introduction
   - Basics on First-order Logic
   - What is a Theory?
   - Satisfiability Modulo Theories
   - Motivations and Goals of SMT

2. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for Theories of Interest (hints)
   - SMT for Combinations of Theories

3. Beyond Solving: Advanced SMT Functionalities
   - Proofs and Unsatisfiable Cores
   - All-SMT & Predicate Abstraction (hints)
   - SMT with Optimization (Optimization Modulo Theories)

Beyond Solving: advanced SAT & SMT functionalities

Advanced SMT functionalities (very important in FV):

- Building proofs of $\mathcal{T}$-unsatisfiability
- Extracting $\mathcal{T}$-unsatisfiable Cores
- [Computing Craig interpolants]
- Performing All-SMT and Predicate Abstraction
- Deciding/optimizing SMT problems with costs

Beyond Solving: advanced SAT & SMT functionalities

Advanced SMT functionalities (very important in FV):

- Building proofs of $\mathcal{T}$-unsatisfiability
- Extracting $\mathcal{T}$-unsatisfiable Cores
- [Computing Craig interpolants]
- Performing All-SMT and Predicate Abstraction
- Deciding/optimizing SMT problems with costs

Beyond Solving: advanced SAT & SMT functionalities

Advanced SMT functionalities (very important in FV):

- Building proofs of $\mathcal{T}$-unsatisfiability
- Extracting $\mathcal{T}$-unsatisfiable Cores
- [Computing Craig interpolants]
- Performing All-SMT and Predicate Abstraction
- Deciding/optimizing SMT problems with costs

Beyond Solving: advanced SAT & SMT functionalities

Advanced SMT functionalities (very important in FV):

- Building proofs of $\mathcal{T}$-unsatisfiability
- Extracting $\mathcal{T}$-unsatisfiable Cores
- [Computing Craig interpolants]
- Performing All-SMT and Predicate Abstraction
- Deciding/optimizing SMT problems with costs

Beyond Solving: advanced SAT & SMT functionalities

Advanced SMT functionalities (very important in FV):

- Building proofs of $\mathcal{T}$-unsatisfiability
- Extracting $\mathcal{T}$-unsatisfiable Cores
- [Computing Craig interpolants]
- Performing All-SMT and Predicate Abstraction
- Deciding/optimizing SMT problems with costs

Outline

1. Introduction
   - Basics on First-order Logic
   - What is a Theory?
   - Satisfiability Modulo Theories
   - Motivations and Goals of SMT

2. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for Theories of Interest (hints)
   - SMT for Combinations of Theories

3. Beyond Solving: Advanced SMT Functionalities
   - Proofs and Unsatisfiable Cores
   - All-SMT & Predicate Abstraction (hints)
   - SMT with Optimization (Optimization Modulo Theories)

Building (Resolution) Proofs of $\mathcal{T}$-Unsatisfiability

Resolution proof of $\mathcal{T}$-unsatisfiability

Very similar to building proofs with plain SAT:

- resolution proofs whose leaves are original clauses and $\mathcal{T}$-lemmas returned by the $\mathcal{T}$-*solver* (i.e., $\mathcal{T}$-conflict and $\mathcal{T}$-deduction clauses)
- built by backward traversal of implication graphs, as in CDCL SAT
- Sub-proofs of $\mathcal{T}$-lemmas can be built in some $\mathcal{T}$-specific deduction framework if requested

Important for:

- certifying $\mathcal{T}$-unsatisfiability results
- computing unsatisfiable cores
- computing interpolants

Building (Resolution) Proofs of $\mathcal{T}$-Unsatisfiability

Resolution proof of $\mathcal{T}$-unsatisfiability

Very similar to building proofs with plain SAT:

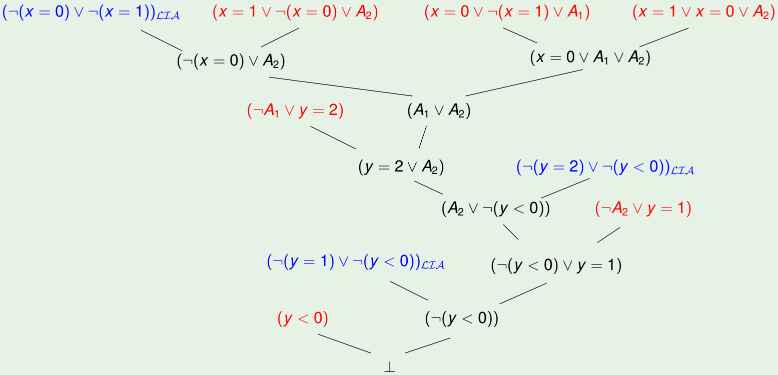
- resolution proofs whose leaves are original clauses and $\mathcal{T}$-lemmas returned by the $\mathcal{T}$-*solver* (i.e., $\mathcal{T}$-conflict and $\mathcal{T}$-deduction clauses)
- built by backward traversal of implication graphs, as in CDCL SAT
- Sub-proofs of $\mathcal{T}$-lemmas can be built in some $\mathcal{T}$-specific deduction framework if requested

Important for:

- certifying $\mathcal{T}$-unsatisfiability results
- computing unsatisfiable cores
- computing interpolants

Building Proofs of $\mathcal{T}$-Unsatisfiability: example

$$(x = 0 \vee \neg(x = 1) \vee A_1) \wedge (x = 0 \vee x = 1 \vee A_2) \wedge (\neg(x = 0) \vee x = 1 \vee A_2) \wedge$$
$$(\neg A_2 \vee y = 1) \wedge (\neg A_1 \vee x + y > 3) \wedge (y < 0) \wedge (A_2 \vee x - y = 4) \wedge (y = 2 \vee \neg A_1) \wedge (x \geq 0),$$



$(\neg(x = 0) \vee \neg(x = 1))_{\mathcal{LIA}}$ $(x = 1 \vee \neg(x = 0) \vee A_2)$ $(x = 0 \vee \neg(x = 1) \vee A_1)$ $(x = 1 \vee x = 0 \vee A_2)$

$(\neg(x = 0) \vee A_2)$ $(x = 0 \vee A_1 \vee A_2)$

$(\neg A_1 \vee y = 2)$ $(A_1 \vee A_2)$

$(y = 2 \vee A_2)$ $(\neg(y = 2) \vee \neg(y < 0))_{\mathcal{LIA}}$

$(A_2 \vee \neg(y < 0))$ $(\neg A_2 \vee y = 1)$

$(\neg(y = 1) \vee \neg(y < 0))_{\mathcal{LIA}}$ $(\neg(y < 0) \vee y = 1)$

$(y < 0)$ $(\neg(y < 0))$

$\perp$

relevant original clauses, irrelevant original clauses, $\mathcal{T}$-lemmas

Example: proof on non-strict $\mathcal{LRA}$ inequalities

- A proof of unsatisfiability for a set of non-strict $\mathcal{LRA}$ inequalities can be obtained by building a linear combination (with positive coefficients) of such inequalities, each time eliminating one or more variables, until you get a contradictory inequality on constant values.
- Example:

$$\varphi \stackrel{\text{def}}{=} (0 \le x_1 - 3x_2 + 1), (0 \le x_1 + x_2), (0 \le x_3 - 2x_1 - 3), (0 \le 1 - 2x_3).$$

A proof of unsatisfiability P for φ is the following:

$$\dfrac{\dfrac{(0 \le x_1 - 3x_2 + 1) \quad (0 \le x_1 + x_2)}{\textsc{Comb}\ \ (0 \le 4x_1 + 1)\ \textit{with coefficients 1 and 3}} \qquad \dfrac{(0 \le x_3 - 2x_1 - 3) \quad (0 \le 1 - 2x_3)}{\textsc{Comb}\ \ (0 \le -4x_1 - 5)\ \textit{with coefficients 2 and 1}}}{\textsc{Comb}\ \ (0 \le -4)\ \textit{with coefficients 1 and 1}}$$

- It is possible to produce such proof from an unsatisfiable tableau in Simplex procedure for $\mathcal{LRA}$ [29, 31]
- It is straightforward to produce such proof from a negative cycle in the graph-based procedure for $\mathcal{DL}$ [29, 31]

Example: proof on non-strict $\mathcal{LRA}$ inequalities

- A proof of unsatisfiability for a set of non-strict $\mathcal{LRA}$ inequalities can be obtained by building a linear combination (with positive coefficients) of such inequalities, each time eliminating one or more variables, until you get a contradictory inequality on constant values.
- Example:

$$\varphi \quad \overset{\text{def}}{=} (0 \leq x_1 - 3x_2 + 1), (0 \leq x_1 + x_2), (0 \leq x_3 - 2x_1 - 3), (0 \leq 1 - 2x_3).$$

A proof of unsatisfiability P for φ is the following:

$$\dfrac{\dfrac{(0 \leq x_1 - 3x_2 + 1) \quad (0 \leq x_1 + x_2)}{\textsc{Comb} \;\; (0 \leq 4x_1 + 1) \; \textit{with coefficients 1 and 3}} \quad \dfrac{(0 \leq x_3 - 2x_1 - 3) \quad (0 \leq 1 - 2x_3)}{\textsc{Comb} \;\; (0 \leq -4x_1 - 5) \; \textit{with coefficients 2 and 1}}}{\textsc{Comb} \;\; (0 \leq -4) \; \textit{with coefficients 1 and 1}}$$

- It is possible to produce such proof from an unsatisfiable tableau in Simplex procedure for $\mathcal{LRA}$ [29, 31]
- It is straightforward to produce such proof from a negative cycle in the graph-based procedure for $\mathcal{DL}$ [29, 31]

Example: proof on non-strict $\mathcal{LRA}$ inequalities

- A proof of unsatisfiability for a set of non-strict $\mathcal{LRA}$ inequalities can be obtained by building a linear combination (with positive coefficients) of such inequalities, each time eliminating one or more variables, until you get a contradictory inequality on constant values.
- Example:

$$\varphi \overset{\text{def}}{=} (0 \le x_1 - 3x_2 + 1), (0 \le x_1 + x_2), (0 \le x_3 - 2x_1 - 3), (0 \le 1 - 2x_3).$$

A proof of unsatisfiability P for φ is the following:

$$\dfrac{\dfrac{(0 \le x_1 - 3x_2 + 1) \quad (0 \le x_1 + x_2)}{\textsc{Comb}\ \ (0 \le 4x_1 + 1)\ \textit{with coefficients 1 and 3}} \quad \dfrac{(0 \le x_3 - 2x_1 - 3) \quad (0 \le 1 - 2x_3)}{\textsc{Comb}\ \ (0 \le -4x_1 - 5)\ \textit{with coefficients 2 and 1}}}{\textsc{Comb}\ \ (0 \le -4)\ \textit{with coefficients 1 and 1}}$$

- It is possible to produce such proof from an unsatisfiable tableau in Simplex procedure for $\mathcal{LRA}$ [29, 31]
- It is straightforward to produce such proof from a negative cycle in the graph-based procedure for $\mathcal{DL}$ [29, 31]

Example: proof on non-strict $\mathcal{LRA}$ inequalities

- A proof of unsatisfiability for a set of non-strict $\mathcal{LRA}$ inequalities can be obtained by building a linear combination (with positive coefficients) of such inequalities, each time eliminating one or more variables, until you get a contradictory inequality on constant values.
- Example:

$$\varphi \stackrel{\text{def}}{=} (0 \le x_1 - 3x_2 + 1), (0 \le x_1 + x_2), (0 \le x_3 - 2x_1 - 3), (0 \le 1 - 2x_3).$$

A proof of unsatisfiability P for φ is the following:

$$\dfrac{\dfrac{(0 \le x_1 - 3x_2 + 1) \quad (0 \le x_1 + x_2)}{\textsc{Comb}\ \ (0 \le 4x_1 + 1)\ \textit{with coefficients 1 and 3}} \qquad \dfrac{(0 \le x_3 - 2x_1 - 3) \quad (0 \le 1 - 2x_3)}{\textsc{Comb}\ \ (0 \le -4x_1 - 5)\ \textit{with coefficients 2 and 1}}}{\textsc{Comb}\ \ (0 \le -4)\ \textit{with coefficients 1 and 1}}$$

- It is possible to produce such proof from an unsatisfiable tableau in Simplex procedure for $\mathcal{LRA}$ [29, 31]
- It is straightforward to produce such proof from a negative cycle in the graph-based procedure for $\mathcal{DL}$ [29, 31]

Exercises

- Given the following $\mathcal{LRA}$-unsatisfiable sets of $\mathcal{LRA}$-inequalities, produce a proof of unsatisfiablity by means of the technique described in the previous slides.
  - $$\left\{\begin{array}{lrrrrr} (0 \leq & +2x_1 & -2x_2 & & +6x_4 & +1) \\ (0 \leq & -x_1 & +2x_2 & & -x_4 & -2) \\ (0 \leq & & -2x_2 & +2x_3 & & +3) \\ (0 \leq & & & -x_3 & -x_4 & -2) \\ (0 \leq & & +2x_2 & & +x_4 & +2) \end{array}\right\}$$
  - [create your own $\mathcal{LRA}$-unsatisfiable set of $\mathcal{LRA}$-inequalities]
- Given the following $\mathcal{DL}$-unsatisfiable sets of $\mathcal{DL}$-inequalities, produce a proof of unsatisfiablity by means of the technique described in the previous slides.
  - the $\mathcal{DL}$ example in the $\mathcal{T}$-solver section and its negative cycle
  - [create your own $\mathcal{DL}$-unsatisfiable set of $\mathcal{LRA}$-inequalities]

Exercises

- Given the following $\mathcal{LRA}$-unsatisfiable sets of $\mathcal{LRA}$-inequalities, produce a proof of unsatisfiablity by means of the technique described in the previous slides.
  - $$\left\{ \begin{array}{rrrrrr} (0 \leq & +2x_1 & -2x_2 & & +6x_4 & +1) \\ (0 \leq & -x_1 & +2x_2 & & -x_4 & -2) \\ (0 \leq & & -2x_2 & +2x_3 & & +3) \\ (0 \leq & & & -x_3 & -x_4 & -2) \\ (0 \leq & & +2x_2 & & +x_4 & +2) \end{array} \right\}$$
  - [create your own $\mathcal{LRA}$-unsatisfiable set of $\mathcal{LRA}$-inequalities]
- Given the following $\mathcal{DL}$-unsatisfiable sets of $\mathcal{DL}$-inequalities, produce a proof of unsatisfiablity by means of the technique described in the previous slides.
  - the $\mathcal{DL}$ example in the $\mathcal{T}$-solver section and its negative cycle
  - [create your own $\mathcal{DL}$-unsatisfiable set of $\mathcal{LRA}$-inequalities]

Extraction of $\mathcal{T}$-unsatisfiable cores

The problem

Given a $\mathcal{T}$-unsatisfiable set of clauses, extract from it a (possibly small/minimal/minimum) $\mathcal{T}$-unsatisfiable subset ($\mathcal{T}$-unsatisfiable core)

- Wide literature in SAT
- Some implementations, very few literature for SMT [28, 56]
- We recognize three approaches:
  - Proof-based approach (CVC4, MathSAT): byproduct of finding a resolution proof
  - Assumption-based approach (Yices): use extra variables labeling clauses, as in the plain Boolean case
  - Lemma-Lifting approach [28] : use an external (possibly-optimized) Boolean unsat-core extractor

The proof-based approach to $\mathcal{T}$-unsat cores

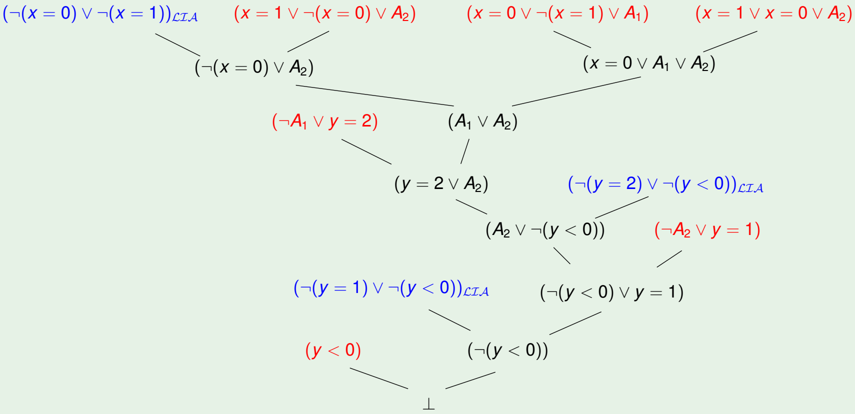
Idea (adapted from [82])

Unsatisfiable core of φ:

- in SAT: the set of leaf clauses of a resolution proof of unsatisfiability of φ
- in SMT($\mathcal{T}$): the set of leaf clauses of a resolution proof of $\mathcal{T}$-unsatisfiability of φ, minus the $\mathcal{T}$-lemmas

The proof-based approach to $\mathcal{T}$-unsat cores: example

$$(x = 0 \vee \neg(x = 1) \vee A_1) \wedge (x = 0 \vee x = 1 \vee A_2) \wedge (\neg(x = 0) \vee x = 1 \vee A_2) \wedge$$
$$(\neg A_2 \vee y = 1) \wedge (\neg A_1 \vee x + y > 3) \wedge (y < 0) \wedge (A_2 \vee x - y = 4) \wedge (y = 2 \vee \neg A_1) \wedge (x \geq 0),$$



$(\neg(x = 0) \vee \neg(x = 1))_{\mathcal{LIA}}$ $(x = 1 \vee \neg(x = 0) \vee A_2)$ $(x = 0 \vee \neg(x = 1) \vee A_1)$ $(x = 1 \vee x = 0 \vee A_2)$

$(\neg(x = 0) \vee A_2)$ $(x = 0 \vee A_1 \vee A_2)$

$(\neg A_1 \vee y = 2)$ $(A_1 \vee A_2)$

$(y = 2 \vee A_2)$ $(\neg(y = 2) \vee \neg(y < 0))_{\mathcal{LIA}}$

$(A_2 \vee \neg(y < 0))$ $(\neg A_2 \vee y = 1)$

$(\neg(y = 1) \vee \neg(y < 0))_{\mathcal{LIA}}$ $(\neg(y < 0) \vee y = 1)$

$(y < 0)$ $(\neg(y < 0))$

$\perp$

The Assumption-based approach to $\mathcal{T}$-unsat cores

Idea (adapted from [57])

Let φ be $\bigwedge_{i=1}^{n} C_i$ s.t. φ unsatisfiable.

1. each clause C_i in φ is substituted by $\neg S_i \vee C_i$, s.t. S_i fresh “selector” variable
2. the resulting formula is checked for satisfiability under the assumption of all S_i’s
3. final conflict clause at dec. level 0: $\bigvee_j \neg S_j$
   $\Longrightarrow \{C_j\}_j$ is the unsat core

Extends straightforwardly to SMT($\mathcal{T}$).

The Assumption-based approach to $\mathcal{T}$-unsat cores

Idea (adapted from [57])

Let φ be $\bigwedge_{i=1}^{n} C_i$ s.t. φ unsatisfiable.

1. each clause C_i in φ is substituted by $\neg S_i \vee C_i$, s.t. S_i fresh "selector" variable
2. the resulting formula is checked for satisfiability under the assumption of all S_i's
3. final conflict clause at dec. level 0: $\bigvee_j \neg S_j$
   $\Longrightarrow \{C_j\}_j$ is the unsat core

Extends straightforwardly to SMT($\mathcal{T}$).

The assumption-based approach to $\mathcal{T}$-unsat cores: Example

$$\begin{aligned}(S_1 \rightarrow (x = 0 \vee \neg(x = 1) \vee A_1)) \wedge (S_2 \rightarrow (x = 0 \vee x = 1 \vee A_2)) \wedge \\ (S_3 \rightarrow (\neg(x = 0) \vee x = 1 \vee A_2)) \wedge (S_4 \rightarrow (\neg A_2 \vee y = 1)) \wedge \\ (S_5 \rightarrow (\neg A_1 \vee x + y > 3)) \wedge (S_6 \rightarrow y < 0) \wedge \\ (S_7 \rightarrow (A_2 \vee x - y = 4)) \wedge (S_8 \rightarrow (y = 2 \vee \neg A_1)) \wedge (S_9 \rightarrow x \geq 0)\end{aligned}$$

Conflict analysis (Yices 1.0.6) returns:

$$\neg S_1 \vee \neg S_2 \vee \neg S_3 \vee \neg S_4 \vee \neg S_6 \vee \neg S_7 \vee \neg S_8,$$

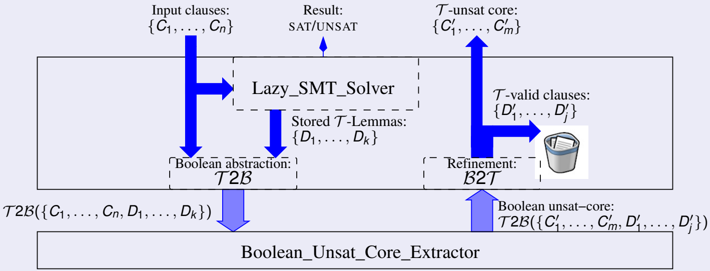
corresponding to the unsat core in red.

The lemma-lifting approach to $\mathcal{T}$-unsat cores

Idea [28, 32]

(i) The $\mathcal{T}$-lemmas D_i are valid in $\mathcal{T}$

(ii) The conjunction of φ with all the $\mathcal{T}$-lemmas $D_1, \ldots, D_k$ is propositionally unsatisfiable: $\mathcal{T}2\mathcal{B}(\varphi \wedge \bigwedge_{i=1}^{n} D_i) \models \bot$.



Input clauses: $\{C_1, \ldots, C_n\}$

Result: SAT/UNSAT

$\mathcal{T}$-unsat core: $\{C'_1, \ldots, C'_m\}$

Lazy_SMT_Solver

$\mathcal{T}$-valid clauses: $\{D'_1, \ldots, D'_j\}$

Stored $\mathcal{T}$-Lemmas: $\{D_1, \ldots, D_k\}$

Boolean abstraction: $\mathcal{T}2\mathcal{B}$

Refinement: $\mathcal{B}2\mathcal{T}$

$\mathcal{T}2\mathcal{B}(\{C_1, \ldots, C_n, D_1, \ldots, D_k\})$

Boolean unsat–core: $\mathcal{T}2\mathcal{B}(\{C'_1, \ldots, C'_m, D'_1, \ldots, D'_j\})$

Boolean_Unsat_Core_Extractor

- interfaces with an external Boolean Unsat-core Extractor

$\Longrightarrow$ benefits for free of all state-of-the-art size-reduction techniques

The lemma-lifting approach to $\mathcal{T}$-unsat cores (cont.)

```
⟨SatValue,Clause_set⟩ $\mathcal{T}$-Unsat_Core(Clause_set $\varphi$) {
   // $\varphi$ is $\{C_1, \ldots, C_n\}$
   if (Lazy_SMT_Solver($\varphi$) == SAT)
      then return ⟨SAT,$\emptyset$⟩;
   // $D_1, \ldots, D_k$ are the $\mathcal{T}$-lemmas stored by Lazy_SMT_Solver
   $\psi^p$=Boolean_Core_Extractor($\mathcal{T}2\mathcal{B}(\{C_1, \ldots, C_n, D_1, \ldots, D_k\})$);
   // $\psi^p$ is $\mathcal{T}2\mathcal{B}(\{C'_1, \ldots, C'_m, D'_1, \ldots, D'_j\})$);
   return ⟨UNSAT,$\{C'_1, \ldots, C'_m\}$⟩;
}
```

The lemma-lifting approach to $\mathcal{T}$-unsat cores: example

$$(x = 0 \vee \neg(x = 1) \vee A_1) \wedge (x = 0 \vee x = 1 \vee A_2) \wedge (\neg(x = 0) \vee x = 1 \vee A_2) \wedge$$
$$(\neg A_2 \vee y = 1) \wedge (\neg A_1 \vee x + y > 3) \wedge (y < 0) \wedge (A_2 \vee x - y = 4) \wedge (y = 2 \vee \neg A_1) \wedge (x \geq 0),$$

1. The SMT solver generates the following set of $\mathcal{LIA}$-lemmas:

$$\{(\neg(x = 1) \vee \neg(x = 0)), \quad (\neg(y = 2) \vee \neg(y < 0)), \quad (\neg(y = 1) \vee \neg(y < 0))\}.$$

2. The following formula is passed to the external Boolean core extractor

$$(B_0 \vee \neg B_1 \vee A_1) \wedge (B_0 \vee B_1 \vee A_2) \wedge (\neg B_0 \vee B_1 \vee A_2) \wedge$$
$$(\neg A_2 \vee B_2) \wedge (\neg A_1 \vee B_3) \wedge B_4 \wedge (A_2 \vee B_5) \wedge (B_6 \vee \neg A_1) \wedge B_7 \wedge$$
$$(\neg B_1 \vee \neg B_0) \wedge (\neg B_6 \vee \neg B_4) \wedge (\neg B_2 \vee \neg B_4)$$

   which returns the unsat core in red.

3. The unsat-core is mapped back, the three $\mathcal{T}$-lemmas are removed $\Longrightarrow$ the final $\mathcal{T}$-unsat core (in red above).

The lemma-lifting approach to $\mathcal{T}$-unsat cores: example

$$(x = 0 \lor \neg(x = 1) \lor A_1) \land (x = 0 \lor x = 1 \lor A_2) \land (\neg(x = 0) \lor x = 1 \lor A_2) \land$$
$$(\neg A_2 \lor y = 1) \land (\neg A_1 \lor x + y > 3) \land (y < 0) \land (A_2 \lor x - y = 4) \land (y = 2 \lor \neg A_1) \land (x \geq 0),$$

1 The SMT solver generates the following set of $\mathcal{LIA}$-lemmas:

$$\{(\neg(x = 1) \lor \neg(x = 0)), \quad (\neg(y = 2) \lor \neg(y < 0)), \quad (\neg(y = 1) \lor \neg(y < 0))\}.$$

2 The following formula is passed to the external Boolean core extractor

$$(B_0 \lor \neg B_1 \lor A_1) \land (B_0 \lor B_1 \lor A_2) \land (\neg B_0 \lor B_1 \lor A_2) \land$$
$$(\neg A_2 \lor B_2) \land (\neg A_1 \lor B_3) \land B_4 \land (A_2 \lor B_5) \land (B_6 \lor \neg A_1) \land B_7 \land$$
$$(\neg B_1 \lor \neg B_0) \land (\neg B_6 \lor \neg B_4) \land (\neg B_2 \lor \neg B_4)$$

which returns the unsat core in red.

3 The unsat-core is mapped back, the three $\mathcal{T}$-lemmas are removed
$\Longrightarrow$ the final $\mathcal{T}$-unsat core (in red above).

The lemma-lifting approach to $\mathcal{T}$-unsat cores: example

$$(x = 0 \lor \neg(x = 1) \lor A_1) \land (x = 0 \lor x = 1 \lor A_2) \land (\neg(x = 0) \lor x = 1 \lor A_2) \land$$
$$(\neg A_2 \lor y = 1) \land (\neg A_1 \lor x + y > 3) \land (y < 0) \land (A_2 \lor x - y = 4) \land (y = 2 \lor \neg A_1) \land (x \geq 0),$$

1 The SMT solver generates the following set of $\mathcal{LIA}$-lemmas:

$$\{(\neg(x = 1) \lor \neg(x = 0)), \quad (\neg(y = 2) \lor \neg(y < 0)), \quad (\neg(y = 1) \lor \neg(y < 0))\}.$$

2 The following formula is passed to the external Boolean core extractor

$$(B_0 \lor \neg B_1 \lor A_1) \land (B_0 \lor B_1 \lor A_2) \land (\neg B_0 \lor B_1 \lor A_2) \land$$
$$(\neg A_2 \lor B_2) \land (\neg A_1 \lor B_3) \land B_4 \land (A_2 \lor B_5) \land (B_6 \lor \neg A_1) \land B_7 \land$$
$$(\neg B_1 \lor \neg B_0) \land (\neg B_6 \lor \neg B_4) \land (\neg B_2 \lor \neg B_4)$$

which returns the unsat core in red.

3 The unsat-core is mapped back, the three $\mathcal{T}$-lemmas are removed $\Longrightarrow$ the final $\mathcal{T}$-unsat core (in red above).

Exercise

Consider the following set of clauses φ in $\mathcal{EUF}$.

$$\left\{ \begin{array}{l} (\neg(x = y) \vee \quad (f(x) = f(y))), \\ (\neg(x = y) \vee \neg(f(x) = f(y))), \\ (\quad (x = y) \vee \quad (f(x) = f(y))), \\ (\quad (x = y) \vee \neg(f(x) = f(y))) \end{array} \right\}$$

Find a minimal $\mathcal{EUF}$-unsatisfiable core.

Outline

1. Introduction
   - Basics on First-order Logic
   - What is a Theory?
   - Satisfiability Modulo Theories
   - Motivations and Goals of SMT
2. Efficient SMT solving
   - Combining SAT with Theory Solvers
   - Theory Solvers for Theories of Interest (hints)
   - SMT for Combinations of Theories
3. Beyond Solving: Advanced SMT Functionalities
   - Proofs and Unsatisfiable Cores
   - All-SMT & Predicate Abstraction (hints)
   - SMT with Optimization (Optimization Modulo Theories)

All-SAT/All-SMT (hints)

- All-SAT: enumerate all truth assignments satisfying φ
- All-SMT: enumerate all $\mathcal{T}$-satisfiable truth assignments propositionally satisfying φ
- All-SMT over an “important” subset of atoms $\Gamma \stackrel{\text{def}}{=} \{\gamma_i\}_i$: enumerate all assignments over Γ which can be extended to $\mathcal{T}$-satisfiable truth assignments propositionally satisfying φ
  $\Longrightarrow$ can compute predicate abstraction
- Algorithms:
  - BCLT [53]
    each time a $\mathcal{T}$-satisfiable assignment $\{l_1, ..., l_n\}$ is found, perform conflict-driven backjumping as if the restricted clause $(\bigvee_i \neg l_i) \downarrow \Gamma$ belonged to the clause set
  - MathSAT/NuSMV [26]
    As above, plus the Boolean search of the SMT solver is driven by an OBDD.

All-SAT/All-SMT (hints)

- **All-SAT:** enumerate all truth assignments satisfying φ
- **All-SMT:** enumerate all $\mathcal{T}$-satisfiable truth assignments propositionally satisfying φ
- All-SMT over an "important" subset of atoms $\Gamma \overset{\text{def}}{=} \{\gamma_i\}_i$: enumerate all assignments over Γ which can be extended to $\mathcal{T}$-satisfiable truth assignments propositionally satisfying φ
  $\Longrightarrow$ can compute predicate abstraction
- Algorithms:
  - BCLT [53]
    each time a $\mathcal{T}$-satisfiable assignment $\{l_1, ..., l_n\}$ is found, perform conflict-driven backjumping as if the restricted clause $(\bigvee_i \neg l_i) \downarrow \Gamma$ belonged to the clause set
  - MathSAT/NuSMV [26]
    As above, plus the Boolean search of the SMT solver is driven by an OBDD.

All-SAT/All-SMT (hints)

- All-SAT: enumerate all truth assignments satisfying φ
- All-SMT: enumerate all $\mathcal{T}$-satisfiable truth assignments propositionally satisfying φ
- All-SMT over an "important" subset of atoms $\Gamma \overset{\text{def}}{=} \{\gamma_i\}_i$: enumerate all assignments over Γ which can be extended to $\mathcal{T}$-satisfiable truth assignments propositionally satisfying φ
  $\Longrightarrow$ can compute predicate abstraction
- Algorithms:
  - BCLT [53]
    each time a $\mathcal{T}$-satisfiable assignment $\{l_1, ..., l_n\}$ is found, perform conflict-driven backjumping as if the restricted clause $(\bigvee_i \neg l_i) \downarrow \Gamma$ belonged to the clause set
  - MathSAT/NuSMV [26]
    As above, plus the Boolean search of the SMT solver is driven by an OBDD.

All-SAT/All-SMT (hints)

- All-SAT: enumerate all truth assignments satisfying φ
- All-SMT: enumerate all $\mathcal{T}$-satisfiable truth assignments propositionally satisfying φ
- All-SMT over an "important" subset of atoms $\Gamma \overset{\text{def}}{=} \{\gamma_i\}_i$: enumerate all assignments over Γ which can be extended to $\mathcal{T}$-satisfiable truth assignments propositionally satisfying φ
  $\Longrightarrow$ can compute predicate abstraction
- Algorithms:
  - BCLT [53]
    each time a $\mathcal{T}$-satisfiable assignment $\{l_1, ..., l_n\}$ is found, perform conflict-driven backjumping as if the restricted clause $(\bigvee_i \neg l_i) \downarrow \Gamma$ belonged to the clause set
  - MathSAT/NuSMV [26]
    As above, plus the Boolean search of the SMT solver is driven by an OBDD.

Predicate Abstraction

Predicate abstraction

if $\varphi(\mathbf{v})$ is a SMT formula over the domain variables $\mathbf{v} \stackrel{\text{def}}{=} \{v_j\}_j$, $\{\gamma_i\}_i$ is a set of “relevant” predicates over $\mathbf{v}$, and $\mathbf{P} \stackrel{\text{def}}{=} \{P_i\}_i$ a set of fresh Boolean labels, then:

$$\begin{array}{rl} & \mathit{PredAbs}_{\mathbf{P}}(\varphi) \\ \stackrel{\text{def}}{=} & \exists \mathbf{v}.(\ \varphi(\mathbf{v}) \wedge \bigwedge_i P_i \leftrightarrow \gamma_i(\mathbf{v})\ ) \\ = & \bigvee \left\{ \mu \mid \begin{array}{c} \mu \text{ truth assignment on } \mathbf{P} \\ \text{s.t. } \mu \wedge \varphi \wedge \bigwedge_i (P_i \leftrightarrow \gamma_i) \text{ is } \mathcal{T}\text{-satisfiable} \end{array} \right\} \end{array}$$

- projection of φ over (the Boolean abstraction of) the set $\{\gamma_i\}_i$.
- important step in FV: extracts finite-state abstractions from a infinite state space

Predicate Abstraction

Predicate abstraction

if $\varphi(\mathbf{v})$ is a SMT formula over the domain variables $\mathbf{v} \overset{\text{def}}{=} \{v_j\}_j$, $\{\gamma_i\}_i$ is a set of “relevant” predicates over $\mathbf{v}$, and $\mathbf{P} \overset{\text{def}}{=} \{P_i\}_i$ a set of fresh Boolean labels, then:

$$\begin{aligned} & \mathit{PredAbs}_{\mathbf{P}}(\varphi) \\ \overset{\text{def}}{=} \quad & \exists \mathbf{v}.(\ \varphi(\mathbf{v}) \wedge \bigwedge_i P_i \leftrightarrow \gamma_i(\mathbf{v})\ ) \\ = \quad & \bigvee \left\{ \mu \ \middle| \begin{array}{c} \mu \text{ truth assignment on } \mathbf{P} \\ \text{s.t. } \mu \wedge \varphi \wedge \bigwedge_i (P_i \leftrightarrow \gamma_i) \text{ is } \mathcal{T}\text{-satisfiable} \end{array} \right\} \end{aligned}$$

- projection of φ over (the Boolean abstraction of) the set $\{\gamma_i\}_i$.
- important step in FV: extracts finite-state abstractions from a infinite state space

Predicate Abstraction: example

$$\begin{aligned}
\varphi &\stackrel{\text{def}}{=} (v_1 + v_2 > 12) \\
\gamma_1 &\stackrel{\text{def}}{=} (v_1 + v_2 = 2) \\
\gamma_2 &\stackrel{\text{def}}{=} (v_1 - v_2 < 10)
\end{aligned}$$

$$\Downarrow$$

$$\begin{aligned}
PreAbs(\varphi)_{\{P_1,P_2\}} &\stackrel{\text{def}}{=} \exists\, v_1\ v_2\ .\left(\begin{array}{ll} (v_1 + v_2 > 12) & \wedge \\ (P_1 \leftrightarrow (v_1 + v_2 = 2)) & \wedge \\ (P_2 \leftrightarrow (v_1 - v_2 < 10)) & \end{array}\right) \\
&= (\neg P_1 \wedge \neg P_2) \vee (\neg P_1 \wedge P_2) \\
&= \neg P_1 .
\end{aligned}$$

Predicate Abstraction: example

$$
\begin{array}{rcl}
\varphi & \stackrel{\text{def}}{=} & (v_1 + v_2 > 12) \\
\gamma_1 & \stackrel{\text{def}}{=} & (v_1 + v_2 = 2) \\
\gamma_2 & \stackrel{\text{def}}{=} & (v_1 - v_2 < 10)
\end{array}
$$

$$\Downarrow$$

$$
\begin{array}{rcl}
PreAbs(\varphi)_{\{P_1,P_2\}} & \stackrel{\text{def}}{=} & \exists\, v_1\ v_2\,.\left(\begin{array}{ll}(v_1 + v_2 > 12) & \wedge \\ (P_1 \leftrightarrow (v_1 + v_2 = 2)) & \wedge \\ (P_2 \leftrightarrow (v_1 - v_2 < 10)) & \end{array}\right) \\
& = & (\neg P_1 \wedge \neg P_2) \vee (\neg P_1 \wedge P_2) \\
& = & \neg P_1 .
\end{array}
$$

Outline

1. Introduction
    - Basics on First-order Logic
    - What is a Theory?
    - Satisfiability Modulo Theories
    - Motivations and Goals of SMT

2. Efficient SMT solving
    - Combining SAT with Theory Solvers
    - Theory Solvers for Theories of Interest (hints)
    - SMT for Combinations of Theories

3. Beyond Solving: Advanced SMT Functionalities
    - Proofs and Unsatisfiable Cores
    - All-SMT & Predicate Abstraction (hints)
    - SMT with Optimization (Optimization Modulo Theories)

Optimization Modulo Theories: General Case

Ingredients: $\langle \varphi, cost \rangle$

- a SMT formula φ in some background theory $\mathcal{T} = \mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$
  - $\bigcup_i \mathcal{T}_i$ may be empty
  - $\mathcal{T}_{\preceq}$ has a predicate $\preceq$ representing a total order
- a $\mathcal{T}_{\preceq}$-variable/term "*cost*" occurring in φ

Optimization Modulo $\mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$ (OMT($\mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$))

The problem of finding a model $\mathcal{M}$ for φ whose value of *cost* is minimum according to $\preceq$.

- maximization is dual

Note

The cost term can be rewritten as a variable

$$\langle \varphi, term \rangle \implies \langle \varphi \wedge (cost = term), cost \rangle, \quad cost\ fresh$$

Optimization Modulo Theories: General Case

Ingredients: $\langle\varphi, cost\rangle$

- a SMT formula φ in some background theory $\mathcal{T} = \mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$
  - $\bigcup_i \mathcal{T}_i$ may be empty
  - $\mathcal{T}_{\preceq}$ has a predicate $\preceq$ representing a total order
- a $\mathcal{T}_{\preceq}$-variable/term "*cost*" occurring in φ

Optimization Modulo $\mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$ (OMT($\mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$))

The problem of finding a model $\mathcal{M}$ for φ whose value of *cost* is minimum according to $\preceq$.

- maximization is dual

Note

The cost term can be rewritten as a variable

$$\langle\varphi, term\rangle \implies \langle\varphi \wedge (cost = term), cost\rangle, \quad cost\ fresh$$

Optimization Modulo Theories: General Case

Ingredients: $\langle \varphi, cost \rangle$

- a SMT formula φ in some background theory $\mathcal{T} = \mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$
  - $\bigcup_i \mathcal{T}_i$ may be empty
  - $\mathcal{T}_{\preceq}$ has a predicate $\preceq$ representing a total order
- a $\mathcal{T}_{\preceq}$-variable/term "*cost*" occurring in φ

Optimization Modulo $\mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$ (OMT($\mathcal{T}_{\preceq} \cup \bigcup_i \mathcal{T}_i$))

The problem of finding a model $\mathcal{M}$ for φ whose value of *cost* is minimum according to $\preceq$.

- maximization is dual

Note

The cost term can be rewritten as a variable

$$\langle \varphi, term \rangle \implies \langle \varphi \wedge (cost = term), cost \rangle, \quad cost\ fresh$$

Optimization Modulo Theories with $\mathcal{LA}$ costs

Ingredients

- an SMT formula φ on $\mathcal{LA} \cup \mathcal{T}$
  - $\mathcal{LA}$ can be $\mathcal{LRA}$, $\mathcal{LIA}$ or a combination of both
  - $\mathcal{T} \stackrel{\text{def}}{=} \bigcup_i \mathcal{T}_i$, possibly empty
  - $\mathcal{LA}$ and $\mathcal{T}_i$ Nelson-Oppen theories
    (i.e. signature-disjoint infinite-domain theories)
- a $\mathcal{LA}$ variable [term] "*cost*" occurring in φ
- (optionally) two constant numbers lb (lower bound) and ub (upper bound) s.t.
  lb $\leq$ *cost* $<$ ub (lb, ub may be $\mp\infty$)

Optimization Modulo Theories with $\mathcal{LA}$ costs (OMT($\mathcal{LA} \cup \mathcal{T}$))

Find a model for φ whose value of *cost* is minimum.

- maximization dual

We first restrict to the case $\mathcal{LA} = \mathcal{LRA}$ and $\bigcup_i \mathcal{T}_i = \{\}$ (OMT($\mathcal{LRA}$)).

Optimization Modulo Theories with $\mathcal{LRA}$ costs

Ingredients

- an SMT formula φ on $\mathcal{LRA} \cup \mathcal{T}$
  - $\mathcal{LA}$ can be $\mathcal{LRA}$, $\mathcal{LIA}$ or a combination of both
  - $\mathcal{T} \stackrel{\text{def}}{=} \bigcup_i \mathcal{T}_i$, possibly empty
  - $\mathcal{LRA}$ and $\mathcal{T}_i$ Nelson-Oppen theories
    (i.e. signature-disjoint infinite-domain theories)
- a $\mathcal{LRA}$ variable [term] "*cost*" occurring in φ
- (optionally) two constant numbers lb (lower bound) and ub (upper bound) s.t.
  lb $\leq$ *cost* $<$ ub (lb, ub may be $\mp\infty$)

Optimization Modulo Theories with $\mathcal{LRA}$ costs (OMT($\mathcal{LRA} \cup \mathcal{T}$))

Find a model for φ whose value of *cost* is minimum.

- maximization dual

We first restrict to the case $\mathcal{LA} = \mathcal{LRA}$ and $\bigcup_i \mathcal{T}_i = \{\}$ (OMT($\mathcal{LRA}$)).

Solving OMT($\mathcal{LRA}$) [71, 72]

General idea

Combine standard SMT and LP minimization techniques.

Offline Schema

- Minimizer: based on the Simplex $\mathcal{LRA}$-solver by [40]
  - Handles strict inequalities
- Search Strategies:
  - Linear-Search strategy
  - Mixed Linear/Binary strategy

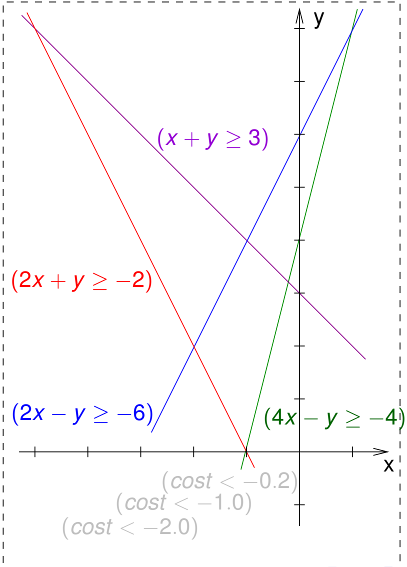
A toy example (linear search)

[w. pure-literal filt. $\Longrightarrow$ partial assignments]

- OMT($\mathcal{LRA}$) problem:

$$
\begin{array}{rl}
\varphi \stackrel{\text{def}}{=} & (\neg A_1 \vee (2x + y \geq -2)) \\
\wedge & (\;\; A_1 \vee (x + y \geq 3)) \\
\wedge & (\neg A_2 \vee (4x - y \geq -4)) \\
\wedge & (\;\; A_2 \vee (2x - y \geq -6)) \\
\wedge & (cost < -0.2) \\
\wedge & (cost < -1.0) \\
\wedge & (cost < -2.0) \\
cost \stackrel{\text{def}}{=} & x
\end{array}
$$

- $\mu = \left\{ \begin{array}{l} A_1, \neg A_1, \quad A_2, \neg A_2, \\ (4x - y \geq -4), \\ (x + y \geq 3), \\ (2x + y \geq -2), \\ (2x - y \geq -6) \\ (cost < -0.2) \\ (cost < -1.0) \\ (cost < -2.0) \end{array} \right\}$



A toy example (linear search)

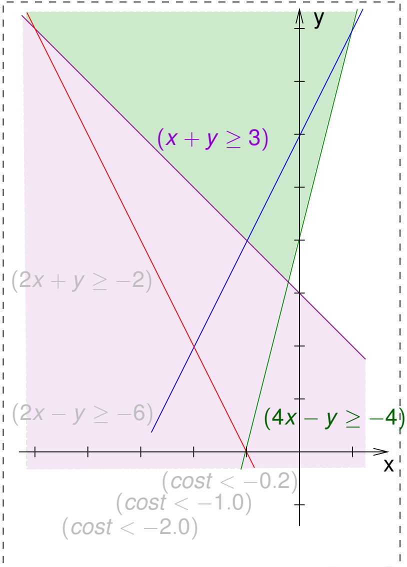
[w. pure-literal filt. $\Longrightarrow$ partial assignments]

- OMT($\mathcal{LRA}$) problem:

$$\begin{array}{rll} \varphi & \overset{\text{def}}{=} & (\neg A_1 \vee (2x + y \geq -2)) \\ & \wedge & (\ A_1 \vee (x + y \geq 3)) \\ & \wedge & (\neg A_2 \vee (4x - y \geq -4)) \\ & \wedge & (\ A_2 \vee (2x - y \geq -6)) \\ & \wedge & (cost < -0.2) \\ & \wedge & (cost < -1.0) \\ & \wedge & (cost < -2.0) \\ cost & \overset{\text{def}}{=} & x \end{array}$$

- $\mu = \left\{ \begin{array}{l} A_1, \neg A_1,\ \ A_2, \neg A_2, \\ (4x - y \geq -4), \\ (x + y \geq 3), \\ (2x + y \geq -2), \\ (2x - y \geq -6) \\ (cost < -0.2) \\ (cost < -1.0) \\ (cost < -2.0) \end{array} \right\}$

$\Longrightarrow$ SAT, $min = -0.2$



A toy example (linear search)

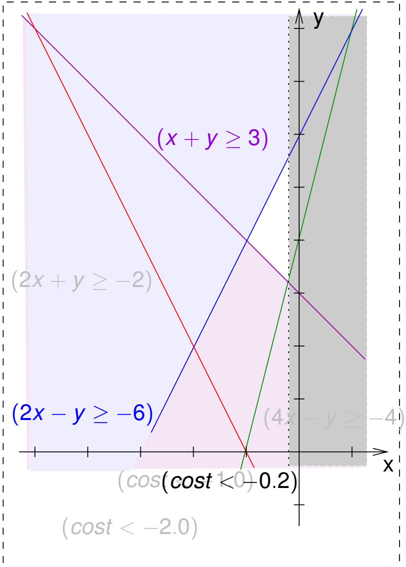
[w. pure-literal filt. $\Longrightarrow$ partial assignments]

- OMT($\mathcal{LRA}$) problem:

$$\begin{array}{rll} \varphi \overset{\text{def}}{=} & & (\neg A_1 \vee (2x+y \geq -2)) \\ & \wedge & (A_1 \vee (x+y \geq 3)) \\ & \wedge & (\neg A_2 \vee (4x-y \geq -4)) \\ & \wedge & (A_2 \vee (2x-y \geq -6)) \\ & \wedge & (cost < -0.2) \\ & \wedge & (cost < -1.0) \\ & \wedge & (cost < -2.0) \\ cost \overset{\text{def}}{=} & & x \end{array}$$

- $$\mu = \left\{ \begin{array}{l} A_1, \neg A_1, \quad A_2, \neg A_2, \\ (4x-y \geq -4), \\ (x+y \geq 3), \\ (2x+y \geq -2), \\ (2x-y \geq -6) \\ (cost < -0.2) \\ (cost < -1.0) \\ (cost < -2.0) \end{array} \right\}$$

$\Longrightarrow$ SAT, $min = -1.0$



A toy example (linear search)

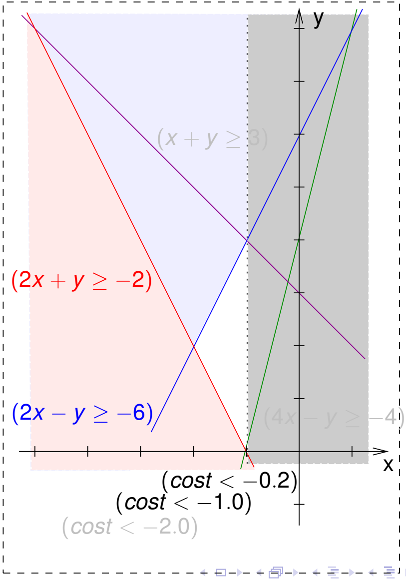
[w. pure-literal filt. $\Longrightarrow$ partial assignments]

- OMT($\mathcal{LRA}$) problem:

$$\begin{array}{rcl} \varphi & \overset{\text{def}}{=} & (\neg A_1 \vee (2x + y \geq -2)) \\ & \wedge & (\ \ A_1 \vee (x + y \geq 3)) \\ & \wedge & (\neg A_2 \vee (4x - y \geq -4)) \\ & \wedge & (\ \ A_2 \vee (2x - y \geq -6)) \\ & \wedge & (cost < -0.2) \\ & \wedge & (cost < -1.0) \\ & \wedge & (cost < -2.0) \\ cost & \overset{\text{def}}{=} & x \end{array}$$

- $\mu = \left\{ \begin{array}{l} \ \ A_1, \neg A_1, \ \ A_2, \neg A_2, \\ (4x - y \geq -4), \\ (x + y \geq 3), \\ (2x + y \geq -2), \\ (2x - y \geq -6) \\ (cost < -0.2) \\ (cost < -1.0) \\ (cost < -2.0) \end{array} \right\}$

$\Longrightarrow$ SAT, $min = -2.0$



A toy example (linear search)

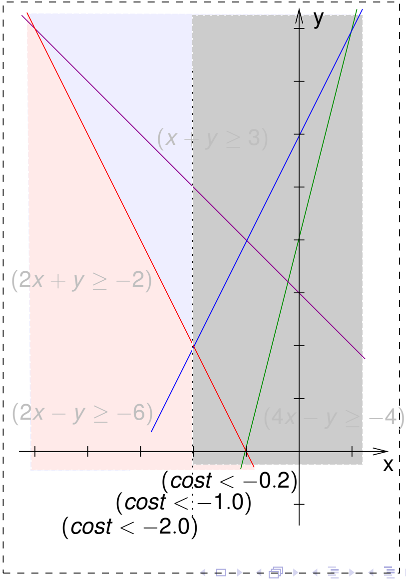
[w. pure-literal filt. $\Longrightarrow$ partial assignments]

- OMT($\mathcal{LRA}$) problem:

$$
\begin{array}{rl}
\varphi \stackrel{\text{def}}{=} & (\neg A_1 \vee (2x + y \geq -2)) \\
\wedge & (\ \ A_1 \vee (x + y \geq 3)) \\
\wedge & (\neg A_2 \vee (4x - y \geq -4)) \\
\wedge & (\ \ A_2 \vee (2x - y \geq -6)) \\
\wedge & (cost < -0.2) \\
\wedge & (cost < -1.0) \\
\wedge & (cost < -2.0) \\
cost \stackrel{\text{def}}{=} & x
\end{array}
$$

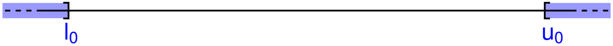
- $\mu = \left\{ \begin{array}{l} A_1, \neg A_1, \quad A_2, \neg A_2, \\ (4x - y \geq -4), \\ (x + y \geq 3), \\ (2x + y \geq -2), \\ (2x - y \geq -6) \\ (cost < -0.2) \\ (cost < -1.0) \\ (cost < -2.0) \end{array} \right\}$

$\Longrightarrow$ UNSAT, $min = -2.0$



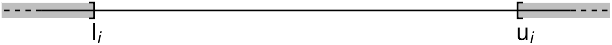
Offline Schema: Mixed Linear/Binary-Search Strategy

```
**Input:** $\langle \varphi, cost, \mathrm{lb}, \mathrm{ub} \rangle$ // lb can be $-\infty$, ub can be $+\infty$
$\mathrm{l} \leftarrow \mathrm{lb}; \mathrm{u} \leftarrow \mathrm{ub}; \mathcal{M} \leftarrow \emptyset; \varphi \leftarrow \varphi \cup \{\neg(cost < \mathrm{lb}), (cost < \mathrm{ub})\};$
**while** $(\mathrm{l} < \mathrm{u})$ **do**
```



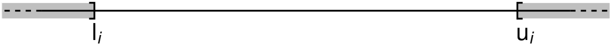
Offline Schema: Mixed Linear/Binary-Search Strategy

```
Input: $\langle \varphi, cost, \mathrm{lb}, \mathrm{ub} \rangle$ // lb can be $-\infty$, ub can be $+\infty$
$\mathrm{l} \leftarrow \mathrm{lb}$; $\mathrm{u} \leftarrow \mathrm{ub}$; $\mathcal{M} \leftarrow \emptyset$; $\varphi \leftarrow \varphi \cup \{\neg(cost < \mathrm{lb}), (cost < \mathrm{ub})\}$;
while ($\mathrm{l} < \mathrm{u}$) do
    if (BinSearchMode()) then  // Binary-search Mode

    else  // Linear-search Mode

```



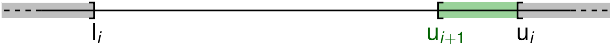
Offline Schema: Mixed Linear/Binary-Search Strategy

```
Input: $\langle \varphi, cost, lb, ub \rangle$ // lb can be $-\infty$, ub can be $+\infty$
$l \leftarrow lb$; $u \leftarrow ub$; $\mathcal{M} \leftarrow \emptyset$; $\varphi \leftarrow \varphi \cup \{\neg(cost < lb), (cost < ub)\}$;
while ($l < u$) do
    if (BinSearchMode()) then  // Binary-search Mode

    else  // Linear-search Mode
        $\langle res, \mu \rangle \leftarrow$ SMT.IncrementalSolve($\varphi$);
```



Offline Schema: Mixed Linear/Binary-Search Strategy

```
Input: $\langle \varphi, cost, \mathsf{lb}, \mathsf{ub} \rangle$ // lb can be $-\infty$, ub can be $+\infty$
$\mathsf{l} \leftarrow \mathsf{lb}$; $\mathsf{u} \leftarrow \mathsf{ub}$; $\mathcal{M} \leftarrow \emptyset$; $\varphi \leftarrow \varphi \cup \{\neg(cost < \mathsf{lb}), (cost < \mathsf{ub})\}$;
while ($\mathsf{l} < \mathsf{u}$) do
    if (BinSearchMode()) then  // Binary-search Mode

    else  // Linear-search Mode
        $\langle \mathsf{res}, \mu \rangle \leftarrow$ SMT.IncrementalSolve($\varphi$);
    if ($\mathsf{res} = \mathsf{SAT}$) then
        $\langle \mathcal{M}, \mathsf{u} \rangle \leftarrow \mathcal{LRA}$-Solver.Minimize($cost, \mu$);
        $\varphi \leftarrow \varphi \cup \{(cost < \mathsf{u})\}$;
    else {$\mathsf{res} = \mathsf{UNSAT}$}
```

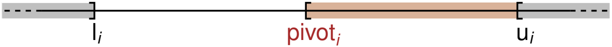


Offline Schema: Mixed Linear/Binary-Search Strategy

```
Input: $\langle \varphi, cost, lb, ub \rangle$ // lb can be $-\infty$, ub can be $+\infty$
$l \leftarrow lb$; $u \leftarrow ub$; $\mathcal{M} \leftarrow \emptyset$; $\varphi \leftarrow \varphi \cup \{\neg(cost < lb), (cost < ub)\}$;
while ($l < u$) do
    if (BinSearchMode()) then  // Binary-search Mode

    else  // Linear-search Mode
        $\langle res, \mu \rangle \leftarrow$ SMT.IncrementalSolve($\varphi$);
    if ($res = SAT$) then

    else {$res = UNSAT$}
            $l \leftarrow u$;
return $\langle \mathcal{M}, u \rangle$
```

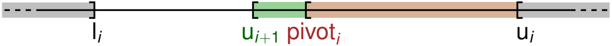
Offline Schema: Mixed Linear/Binary-Search Strategy

```
Input: $\langle \varphi, cost, \mathrm{lb}, \mathrm{ub} \rangle$ // lb can be $-\infty$, ub can be $+\infty$
$\mathrm{l} \leftarrow \mathrm{lb}$; $\mathrm{u} \leftarrow \mathrm{ub}$; $\mathcal{M} \leftarrow \emptyset$; $\varphi \leftarrow \varphi \cup \{\neg(cost < \mathrm{lb}), (cost < \mathrm{ub})\}$;
while ($\mathrm{l} < \mathrm{u}$) do
    if (BinSearchMode()) then  // Binary-search Mode
        pivot $\leftarrow$ ComputePivot(l, u);
        $\varphi \leftarrow \varphi \cup \{(cost < \mathrm{pivot})\}$;
        $\langle \mathrm{res}, \mu \rangle \leftarrow$ SMT.IncrementalSolve($\varphi$);
    else  // Linear-search Mode
```



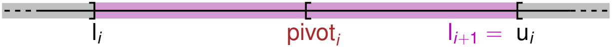
Offline Schema: Mixed Linear/Binary-Search Strategy

```
Input: $\langle \varphi, cost, lb, ub \rangle$ // lb can be $-\infty$, ub can be $+\infty$
$l \leftarrow lb$; $u \leftarrow ub$; $\mathcal{M} \leftarrow \emptyset$; $\varphi \leftarrow \varphi \cup \{\neg(cost < lb), (cost < ub)\}$;
while ($l < u$) do
    if (BinSearchMode()) then  // Binary-search Mode
        $pivot \leftarrow ComputePivot(l, u)$;
        $\varphi \leftarrow \varphi \cup \{(cost < pivot)\}$;
        $\langle res, \mu \rangle \leftarrow$ SMT.IncrementalSolve($\varphi$);
    else  // Linear-search Mode

    if ($res = $ SAT) then
        $\langle \mathcal{M}, u \rangle \leftarrow \mathcal{LRA}$-Solver.Minimize($cost, \mu$);
        $\varphi \leftarrow \varphi \cup \{(cost < u)\}$;
    else {res = UNSAT}
```



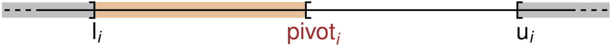
Offline Schema: Mixed Linear/Binary-Search Strategy

```
Input: $\langle\varphi, cost, \mathrm{lb}, \mathrm{ub}\rangle$ // lb can be $-\infty$, ub can be $+\infty$
$\mathrm{l} \leftarrow \mathrm{lb}$; $\mathrm{u} \leftarrow \mathrm{ub}$; $\mathcal{M} \leftarrow \emptyset$; $\varphi \leftarrow \varphi \cup \{\neg(cost < \mathrm{lb}), (cost < \mathrm{ub})\}$;
while ($\mathrm{l} < \mathrm{u}$) do
    if (BinSearchMode()) then // Binary-search Mode
        pivot $\leftarrow$ ComputePivot(l, u);
        $\varphi \leftarrow \varphi \cup \{(cost < \mathrm{pivot})\}$;
        $\langle \mathrm{res}, \mu\rangle \leftarrow$ SMT.IncrementalSolve($\varphi$);
    else // Linear-search Mode

    if (res = SAT) then


    else {res = UNSAT}
        if (($cost < \mathrm{pivot}$) $\notin$ SMT.ExtractUnsatCore($\varphi$)) then
            $\mathrm{l} \leftarrow \mathrm{u}$;
        else


return $\langle\mathcal{M}, \mathrm{u}\rangle$
```



Offline Schema: Mixed Linear/Binary-Search Strategy

```
Input: $\langle \varphi, cost, \text{lb}, \text{ub} \rangle$ // lb can be $-\infty$, ub can be $+\infty$
$\text{l} \leftarrow \text{lb}; \text{u} \leftarrow \text{ub}; \mathcal{M} \leftarrow \emptyset; \varphi \leftarrow \varphi \cup \{\neg(cost < \text{lb}), (cost < \text{ub})\};$
while $(\text{l} < \text{u})$ do
    if (BinSearchMode()) then  // Binary-search Mode
        pivot $\leftarrow$ ComputePivot(l, u);
        $\varphi \leftarrow \varphi \cup \{(cost < \text{pivot})\};$
        $\langle \text{res}, \mu \rangle \leftarrow$ SMT.IncrementalSolve($\varphi$);
    else  // Linear-search Mode

    if (res = SAT) then


    else  {res = UNSAT}
        if ((cost < pivot) $\notin$ SMT.ExtractUnsatCore($\varphi$)) then


        else
            l $\leftarrow$ pivot;
            $\varphi \leftarrow (\varphi \setminus \{(cost < \text{pivot})\} \cup \{\neg(cost < \text{pivot})\};$
```



OMT with Independent Objectives (aka Boxed OMT) [55, 74]

The problem: $\langle\varphi, \{cost_1, ..., cost_k\}\rangle$ [55]

Given $\langle\varphi, \mathcal{C}\rangle$ s.t.:

- φ is the input formula
- $\mathcal{C} \overset{\text{def}}{=} \{cost_1, ..., cost_k\}$ is a set of $\mathcal{LA}$-terms on variables in φ,

$\langle\varphi, \mathcal{C}\rangle$ is the problem of finding a set of independent $\mathcal{LA}$-models $\mathcal{M}_1, ..., \mathcal{M}_k$ s.t. s.t. each $\mathcal{M}_i$ makes $cost_i$ minimum.

Notes

- derives from SW verification problems [55]
- equivalent to k independent problems $\langle\varphi, cost_1\rangle, ..., \langle\varphi, cost_k\rangle$
- intuition: share search effort for the different objectives
- generalizes to OMT($\mathcal{LA} \cup \mathcal{T}$) straightforwardly

OMT with Multiple Objectives [55, 13, 74]

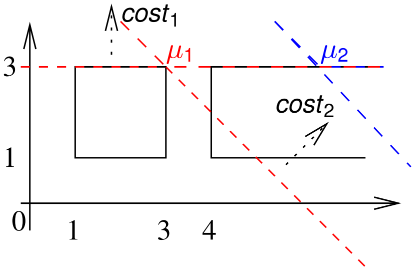
Solution

- Intuition: when a $\mathcal{T}$-satisfiable satisfying assignment μ is found,

```
foreach cost_i
    $\mathtt{min_i} := \mathtt{min}\{\mathtt{min_i}, \mathcal{T}\mathtt{solver.minimize}(\mu, \mathtt{cost_i})\}$;
learn $\bigvee_{\mathtt{i}}(\mathtt{cost_i} < \mathtt{min_i})$;    // $(\mathtt{cost_i} < -\infty) \equiv \bot$
```

  proceed until UNSAT;
- Notice:
  - for each μ, guaranteed improvement of at least one $\mathtt{min}_i$
  - in practice, for each μ, multiple $cost_i$ minima are improved
- Implemented improvements:
  - (a) drop previous clauses $\bigvee_i(cost_i < min_i)$
  - (b) $(cost_i < min_i)$ pushed in μ first: if $\mathcal{T}$-unsatisfiable, skip minimization
  - (c) learn $\neg(cost_i < min_i) \vee (cost_i < min_i^{old})$, s.t. min_i^{old} previous min_i
    $\Longrightarrow$ reuse previously-learned clauses like $\neg(cost_i < min_i^{old}) \vee C$

Boxed OMT: Example [55, 74]



$$\begin{array}{lcl} \varphi & = & (1 \le y) \wedge (y \le 3) \wedge (((1 \le x) \wedge (x \le 3)) \vee (x \ge 4)) \\ & \wedge & (cost_1 = -y) \wedge (cost_2 = -x - y) \end{array}$$

$\mu_1 \quad = \quad \{(1 \le y), (y \le 3), (1 \le x), (x \le 3)\} \Longrightarrow$ SAT $\Longrightarrow [-3, -6]$

$\Longrightarrow$ *learn* $\{(cost_1 < -3) \vee (cost_2 < -6)\}$

$\mu_2 \quad = \quad \{(1 \le y), (y \le 3), (x \ge 4)\} \Longrightarrow$ SAT $\Longrightarrow [-3, -\infty]$

$\Longrightarrow$ *learn* $\{(cost_1 < -3)\}$

$\Longrightarrow$ UNSAT

OMT with Lexicographic Combination of Objectives [13]

The problem

Find one optimal model $\mathcal{M}$ minimizing $\underline{c} \stackrel{\text{def}}{=} cost_1, cost_2, ..., cost_k$ lexicographically.

Solution

- Intuition:

  $\{minimize\ cost_1\}$
  $when$ UNSAT
  $\quad \{substitute\ unit\ clause\ (cost_1 < min_1)\ with\ (cost_1 = min_1)\}$
  $\{minimize\ cost_2\}$
  ...
- improvement:
  - each time UNSAT is found, add $\bigwedge_i (cost_i \leq \mathcal{M}_i(cost_i))$ to φ

Optimization problems encoded into OMT($\mathcal{LA} \cup \mathcal{T}$) I

SMT with Pseudo-Boolean Constraints & Weighted MaxSMT

$$
\begin{array}{rl}
OMT + PB: & \sum_j w_j \cdot A_j,\ \ w_i > 0 \ \ //(\sum_j ite(A_j, w_j, 0)) \\
 & \qquad\quad \Downarrow \\
 & \sum_j x_j,\ \ x_j\ fresh \\
\text{s.t.} & \ldots \wedge \bigwedge_j (A_j \rightarrow (x_j = w_j)) \wedge (\neg A_j \rightarrow (x_j = 0)) \\
 & \qquad\quad \wedge (x_j \geq 0) \wedge (x_j \leq w_j)
\end{array}
$$

$$
\begin{array}{rl}
MaxSMT: & \langle \varphi_h, \bigwedge_j \psi_j \rangle \quad s.t.\ \psi_j\ soft,\ \ w_j = weight(\psi_j),\ \ w_i > 0 \\
 & \qquad\quad \Downarrow \\
 & minimize\ \sum_j x_j,\ \ x_j, A_j\ fresh \\
 & \varphi_h \wedge \bigwedge_j (A_j \vee \psi_j) \wedge \bigwedge_j (\neg A_j \vee (x_j = w_j)) \wedge (A_j \vee (x_j = 0)) \\
 & \qquad\quad \wedge (x_j \geq 0) \wedge (x_j \leq w_j)
\end{array}
$$

Remark: range constraints “$(x_j \geq 0) \wedge (x_j \leq w_j)$”

$$
\begin{array}{lll}
OMT + PB: & \sum_j w_j \cdot A_j,\ w_i > 0\ \ //(\sum_j ite(A_j, w_j, 0)) \\
& \Downarrow \\
& \sum_j x_j,\ x_j\ fresh \\
\text{s.t.} & \ldots \wedge \bigwedge_j (A_j \rightarrow (x_j = w_j)) \wedge (\neg A_j \rightarrow (x_j = 0)) \\
& \wedge (x_j \geq 0) \wedge (x_j \leq w_j)
\end{array}
$$

Range constraints “$(x_j \geq 0) \wedge (x_j \leq w_j)$” logically redundant, but essential for efficiency:

- Without range constraints, the SMT solver can detect the violation of a bound only after all A_i’s are assigned :
  Ex: $w_1 = 4$, $w_2 = 7$, $\sum_{i=1} x_i < 10$, $A_1 = A_2 = \top$, $A_i = * \ \forall i > 2$.
- With range constraints, the SMT solver detects the violation as soon as the assigned A_i’s violate a bound
  $\Longrightarrow$ drastic pruning of the search
- same for weighted MaxSMT

Remark: range constraints "$(x_j \geq 0) \wedge (x_j \leq w_j)$"

$$
\begin{array}{rl}
OMT + PB: & \sum_j w_j \cdot A_j, \;\; w_i > 0 \;\; //(\sum_j ite(A_j, w_j, 0)) \\
& \Downarrow \\
& \sum_j x_j, \;\; x_j \; fresh \\
\text{s.t.} & \ldots \wedge \bigwedge_j (A_j \rightarrow (x_j = w_j)) \wedge (\neg A_j \rightarrow (x_j = 0)) \\
& \wedge (x_j \geq 0) \wedge (x_j \leq w_j)
\end{array}
$$

Range constraints "$(x_j \geq 0) \wedge (x_j \leq w_j)$" logically redundant, but essential for efficiency:

- Without range constraints, the SMT solver can detect the violation of a bound only after all A_i's are assigned :
  Ex: $w_1 = 4$, $w_2 = 7$, $\sum_{i=1} x_i < 10$, $A_1 = A_2 = \top$, $A_i = * \; \forall i > 2$.
- With range constraints, the SMT solver detects the violation as soon as the assigned A_i's violate a bound
  $\Longrightarrow$ drastic pruning of the search
- same for weighted MaxSMT

Remark: range constraints “$(x_j \geq 0) \wedge (x_j \leq w_j)$”

$$OMT + PB: \quad \sum_j w_j \cdot A_j, \;\; w_i > 0 \;\; //(\sum_j ite(A_j, w_j, 0))$$

$$\Downarrow$$

$$\sum_j x_j, \;\; x_j \; fresh$$

$$\text{s.t.} \quad \ldots \wedge \bigwedge_j (A_j \rightarrow (x_j = w_j)) \wedge (\neg A_j \rightarrow (x_j = 0))$$

$$\wedge (x_j \geq 0) \wedge (x_j \leq w_j)$$

Range constraints “$(x_j \geq 0) \wedge (x_j \leq w_j)$” logically redundant, but essential for efficiency:

- Without range constraints, the SMT solver can detect the violation of a bound only after all A_i’s are assigned :
  Ex: $w_1 = 4$, $w_2 = 7$, $\sum_{i=1} x_i < 10$, $A_1 = A_2 = \top$, $A_i = * \; \forall i > 2$.
- With range constraints, the SMT solver detects the violation as soon as the assigned A_i’s violate a bound
  $\Longrightarrow$ drastic pruning of the search
- same for weighted MaxSMT

Remark: range constraints “$(x_j \geq 0) \wedge (x_j \leq w_j)$”

$$
\begin{array}{rl}
OMT+PB: & \sum_j w_j \cdot A_j,\ \ w_i > 0 \ \ //(\sum_j ite(A_j, w_j, 0)) \\
 & \Downarrow \\
 & \sum_j x_j,\ \ x_j\ fresh \\
\text{s.t.} & \ldots \wedge \bigwedge_j (A_j \rightarrow (x_j = w_j)) \wedge (\neg A_j \rightarrow (x_j = 0)) \\
 & \wedge (x_j \geq 0) \wedge (x_j \leq w_j)
\end{array}
$$

Range constraints “$(x_j \geq 0) \wedge (x_j \leq w_j)$” logically redundant, but essential for efficiency:

- Without range constraints, the SMT solver can detect the violation of a bound only after all A_i’s are assigned :
  Ex: $w_1 = 4$, $w_2 = 7$, $\sum_{i=1} x_i < 10$, $A_1 = A_2 = \top$, $A_i = * \ \forall i > 2$.
- With range constraints, the SMT solver detects the violation as soon as the assigned A_i’s violate a bound
  $\Longrightarrow$ drastic pruning of the search
- same for weighted MaxSMT

Optimization problems encoded into OMT($\mathcal{LA} \cup \mathcal{T}$) II

OMT with Min-Max [Max-Min] optimization

Given $\langle \varphi, \{cost_1, ..., cost_k\} \rangle$, find a solution which minimizes the maximum value among $\{cost_1, ..., cost_k\}$. (Max-Min dual.)

- Frequent in some applications (e.g. [72, 79])

$\Longrightarrow$ encode into OMT($\mathcal{LA} \cup \mathcal{T}$) problem $\{\varphi \wedge \bigwedge_i (cost_i \leq cost), cost\}$ s.t. $cost$ fresh.

OMT with linear combinations of costs

Given $\langle \varphi, \{cost_1, ..., cost_k\} \rangle$ and a set of weights $\{w_1, ..., w_k\}$, find a solution which minimizes $\sum_i w_i \cdot cost_i$.

$\Longrightarrow$ encode into OMT($\mathcal{LA} \cup \mathcal{T}$) problem $\{\varphi \wedge (cost = \sum_i w_i \cdot cost_i), cost\}$ s.t. $cost$ fresh.

These objectives can be composed with other OMT($\mathcal{LA}$) objectives.

Other OMT Functionalities [hints]

Incremental interface [13, 74]

Allows for pushing/popping sub-formulas into a stack, and then run OMT incrementally over them, reusing previous search.

- useful in some applications (e.g., BMC with parametric systems)
- straightforward variant of incremental SAT and SMT solvers

Pareto Fronts [13, 12]

- Given $cost_1$, $cost_2$, compute $\mathcal{M}_1, ..., \mathcal{M}_i, ..., \mathcal{M}_j, ...$ s.t.:
  - either $\mathcal{M}_i(cost_1) > \mathcal{M}_j(cost_1)$ or $\mathcal{M}_i(cost_2) > \mathcal{M}_j(cost_2)$ and $\mathcal{M}_i(cost_1) < \mathcal{M}_j(cost_1)$ or $\mathcal{M}_i(cost_2) < \mathcal{M}_j(cost_2)$
  - for each $\mathcal{M}_i$, no $\mathcal{M}'$ dominates $\mathcal{M}_i$
- no objective can be improved without degrading some other one

Some OMT tools

- BCLT [66, 54]
  `https://www.cs.upc.edu/~oliveras/bclt-main.html`
- OPTIMATHSAT [71, 72, 74, 73], on top of MATHSAT [27]
  `https://optimathsat.disi.unitn.it`
- SYMBA [55], on top of Z3 [37]
  `https://bitbucket.org/arieg/symba/src`
- νZ [13, 12], on top of Z3 [37]
  `https://z3.codeplex.com`

Links I

- survey papers:
  - Roberto Sebastiani: "Lazy Satisfiability Modulo Theories". Journal on Satisfiability, Boolean Modeling and Computation, JSAT. Vol. 3, 2007. Pag 141–224, ©IOS Press.
  - Clark Barrett, Roberto Sebastiani, Sanjit Seshia, Cesare Tinelli "Satisfiability Modulo Theories". Part II, Chapter 33, The Handbook of Satisfiability, II ed. 2021. ©IOS press.
  - Leonardo de Moura and Nikolaj Bjørner. “Satisfiability modulo theories: introduction and applications”. Communications of the ACM, 54 (9), 2011. ©ACM press.
- web links:
  - The SMT library SMT-LIB: `https://goedel.cs.uiowa.edu/smtlib/`
  - The SMT Competition SMT-COMP: `https://www.smtcomp.org/`
  - The SAT/SMT Schools `https://satassociation.org/sat-smt-school.html`

References I

[1] A. Armando.
Simplifying OBDDs in Decidable Theories.
In *Proc. PDPAR'03.*, 2003.

[2] A. Armando, C. Castellini, and E. Giunchiglia.
SAT-based procedures for temporal reasoning.
In *Proc. European Conference on Planning, CP-99*, 1999.

[3] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani.
A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions.
In *Proc. CADE'2002.*, volume 2392 of *LNAI*. Springer, July 2002.

[4] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani.
Integrating Boolean and Mathematical Solving: Foundations, Basic Algorithms and Requirements.
In *Proc. AIARSC'2002*, volume 2385 of *LNAI*. Springer, 2002.

[5] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani.
Verifying Industrial Hybrid Systems with MathSAT.
In *Proc. PDPAR'03*, 2003.

[6] G. Audemard, A. Cimatti, A. Korniłowicz, and R. Sebastiani.
SAT-Based Bounded Model Checking for Timed Systems.
In *Proc. FORTE'02.*, volume 2529 of *LNCS*. Springer, November 2002.

[7] C. Barret, D. Dill, and A. Stump.
A Generalization of Shostak's Method for Combining Decision Procedures.
In *Proc. FROCOS'02*, 2002.

[8] C. Barrett, D. Dill, and A. Stump.
Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT.
In *14th International Conference on Computer-Aided Verification*, 2002.

[9] C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Splitting on Demand in SAT Modulo Theories.
In *Proc. LPAR'06*, volume 4246 of *LNAI*. Springer, 2006.

References II

[10] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli.
Satisfiability Modulo Theories.
In *Handbook of Satisfiability*, chapter 26, pages 825–885. IOS Press, 2009.

[11] P. Baumgartner.
FDPLL - A First Order Davis-Putnam-Longeman-Loveland Procedure.
In *Proceedings of CADE-17*, pages 200–219. Springer-Verlag, 2000.

[12] N. Bjørner, A. Phan, and L. Fleckenstein.
νz - an optimizing SMT solver.
In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 194–199, 2015.

[13] N. Bjorner and A.-D. Phan.
νZ - Maximal Satisfaction with Z3.
In *Proc International Symposium on Symbolic Computation in Software Science*, Gammart, Tunisia, December 2014. EasyChair Proceedings in Computing (EPiC).
`http://www.easychair.org/publications/?page=862275542.`

[14] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani.
Efficient Satisfiability Modulo Theories via Boolean Search.
Information and Computation, 2005.

[15] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani.
Efficient Satisfiability Modulo Theories via Delayed Theory Combination.
In *Proc. CAV 2005*, volume 3576 of *LNCS*. Springer, 2005.

[16] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani.
Efficient Theory Combination via Boolean Search.
Information and Computation, 204(10):1493–1525, 2006.

[17] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani.
Mathsat: Tight integration of sat and mathematical decision procedures.
Journal of Automated Reasoning, 35(1-3):265–293, 2005.

References III

[18] A. Brillout, D. Kroening, P. Rümmer, and T. Wahl.
An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic.
In *Proc. IJCAR*, volume 6173 of *LNCS*. Springer, 2010.

[19] R. Brinkmann and R. Drechsler.
RTL-datapath verification using integer linear programming.
In *Proc. ASP-DAC 2002*, pages 741–746. IEEE, 2002.

[20] R. Brummaryer and A. Biere.
Lemmas on Demand for the Extensional Theory of Arrays.
Journal on Satisfiability, Boolean Modeling and Computation (JSAT), 6, 2009.

[21] R. Brummayer and A. Biere.
Boolector: An efficient smt solver for bit-vectors and arrays.
In *TACAS*, volume 5505 of *LNCS*, pages 174–177. Springer, 2009.

[22] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, Z. Hanna, A. Nadel, A. Palti, and R. Sebastiani.
A Lazy and Layered SMT($\mathcal{BV}$) Solver for Hard Industrial Verification Problems.
In *CAV*, volume 4590 of *LNCS*, pages 547–560. Springer, 2007.

[23] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani.
Delayed Theory Combination vs. Nelson-Oppen for Satisfiability Modulo Theories: A Comparative Analysis.
In *Proc. LPAR*, volume 4246 of *LNCS*. Springer, 2006.

[24] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani.
Delayed Theory Combination vs. Nelson-Oppen for Satisfiability Modulo Theories: A Comparative Analysis. .
Annals of Mathematics and Artificial Intelligence., 55(1-2), 2009.

[25] J. R. Burch and D. L. Dill.
Automatic Verification of Pipelined Microprocessor Control.
In *Proc. CAV '94*, volume 818 of *LNCS*. Springer, 1994.

[26] R. Cavada, A. Cimatti, A. Franzén, K. Kalyanasundaram, M. Roveri, and R. K. Shyamasundar.
Computing Predicate Abstractions by Integrating BDDs and SMT Solvers.
In *FMCAD*, pages 69–76. IEEE Computer Society, 2007.

References IV

[27] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani.
The MathSAT 5 SMT Solver.
In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS'13.*, volume 7795 of *LNCS*, pages 95–109. Springer, 2013.

[28] A. Cimatti, A. Griggio, and R. Sebastiani.
A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories.
In *SAT*, volume 4501 of *LNCS*, pages 334–339. Springer, 2007.

[29] A. Cimatti, A. Griggio, and R. Sebastiani.
Efficient Interpolant Generation in Satisfiability Modulo Theories.
In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08.*, volume 4963 of *LNCS*. Springer, 2008.

[30] A. Cimatti, A. Griggio, and R. Sebastiani.
Interpolant Generation for UTVPI.
In *CADE*, volume 5663 of *LNCS*, pages 167–182, 2009.

[31] A. Cimatti, A. Griggio, and R. Sebastiani.
Efficient Generation of Craig Interpolants in Satisfiability Modulo Theories.
ACM Transaction on Computational Logics – TOCL, 12(1), October 2010.

[32] A. Cimatti, A. Griggio, and R. Sebastiani.
Computing Small Unsatisfiable Cores in SAT Modulo Theories.
Journal of Artificial Intelligence Research, JAIR, 40:701–728, April 2011.

[33] S. Cotton and O. Maler.
Fast and Flexible Difference Logic Propagation for DPLL(T).
In *Proc. SAT'06*, volume 4121 of *LNCS*. Springer, 2006.

[34] L. de Moura and N. Bjørner.
Efficient E-matching for SMT solvers.
In *Proc. CADE-21, 21st International Conference on Automated Deduction*, volume 4603 of *LNCS*. Springer, 2007.

[35] L. de Moura, H. Ruess, and M. Sorea.
Lazy Theorem Proving for Bounded Model Checking over Infinite Domains.
In *Proc. CADE'2002.*, volume 2392 of *LNAI*. Springer, July 2002.

References V

[36] L. M. de Moura and N. Bjørner.
Model-based theory combination.
Electr. Notes Theor. Comput. Sci., 198(2):37–49, 2008.

[37] L. M. de Moura and N. Bjørner.
Z3: an efficient SMT solver.
In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008.

[38] L. M. de Moura and N. Bjørner.
Generalized, efficient array decision procedures.
In *FMCAD*, pages 45–52. IEEE, 2009.

[39] D. Detlefs, G. Nelson, and J. Saxe.
Simplify: a theorem prover for program checking.
Journal of the ACM, 52(3):365–473, 2005.

[40] B. Dutertre and L. de Moura.
A Fast Linear-Arithmetic Solver for DPLL(T).
In *CAV*, volume 4144 of *LNCS*, 2006.

[41] B. Dutertre and L. de Moura.
System Description: Yices 1.0.
In *Proc. on 2nd SMT competition, SMT-COMP'06*, 2006.
Available at `yices.csl.sri.com/yices-smtcomp06.pdf`.

[42] A. Franzen, A. Cimatti, A. Nadel, R. Sebastiani, and J. Shalev.
Applying SMT in Symbolic Execution of Microcode.
In *Proc. Int. Conference on Formal Methods in Computer Aided Design (FMCAD'10)*. IEEE, 2010.

[43] V. Ganesh and D. L. Dill.
A Decision Procedure for Bit-Vectors and Arrays.
In *CAV*, 2007.

References VI

[44] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
DPLL(T): Fast decision procedures.
In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification, CAV'04 (Boston, Massachusetts)*, LNCS. Springer, 2004.

[45] F. Giunchiglia and R. Sebastiani.
Building decision procedures for modal logics from propositional decision procedures - the case study of modal K.
In *Proc. CADE'13*, LNAI, New Brunswick, NJ, USA, August 1996. Springer.

[46] A. Goel, S. Krstić, and A. Fuchs.
Deciding array formulas with frugal axiom instantiation.
In *Proceedings of SMT'08/BPR'08*, pages 12–17, New York, NY, USA, 2008. ACM.

[47] A. Griggio.
A Practical Approach to SMT(LA(Z)).
In *Proc. SMT 2010*, 2010.

[48] A. Griggio, T. T. H. Le, and R. Sebastiani.
Efficient Interpolant Generation in Satisfiability Modulo Linear Integer Arithmetic.
In *Proc. Tools and Algorithms for the Construction and Analysis of Systems, TACAS'11*, LNCS. Springer, 2011.

[49] A. Griggio, Q. S. Phan, R. Sebastiani, and S. Tomasi.
Stochastic Local Search for SMT: Combining Theory Solvers with WalkSAT.
In *Frontiers of Combining Systems, FroCoS'11*, volume 6989 of *LNAI*. Springer, 2011.

[50] I. Horrocks and P. F. Patel-Schneider.
FaCT and DLP.
In *Proc. Tableaux'98*, pages 27–30, 1998.

[51] H. Jain, E. M. Clarke, and O. Grumberg.
Efficient Craig Interpolation for Linear Diophantine (Dis)Equations and Linear Modular Equations.
In A. Gupta and S. Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 254–267. Springer, 2008.

[52] D. Kroening and G. Weissenbacher.
Lifting Propositional Interpolants to the Word-Level.
In *FMCAD*, pages 85–89, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

References VII

[53] S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras.
SMT techniques for fast predicate abstraction.
In *Proc. CAV*, LNCS 4144. Springer, 2006.

[54] D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio.
Minimal-Model-Guided Approaches to Solving Polynomial Constraints and Extensions.
In C. Sinz and U. Egly, editors, *SAT*, volume 8561 of *Lecture Notes in Computer Science*, pages 333–350. Springer, 2014.

[55] Y. Li, A. Albarghouthi, Z. Kincaid, A. Gurfinkel, and M. Chechik.
Symbolic optimization with smt solvers.
In *POPL*, pages 607–618, 2014.

[56] M. Liffiton and K. Sakallah.
Algortithms for Computing Minimal Unsatisfiable Subsets of Constraints.
Journal of Automated Reasoning, 40(1), 2008.

[57] I. Lynce and J. P. Marques-Silva.
On computing minimum unsatisfiable cores.
In *SAT*, 2004.

[58] M. Mahfoudh, P. Niebert, E. Asarin, and O. Maler.
A Satisfibaility Checker for Difference Logic.
In *Proceedings of SAT-02*, pages 222–230, 2002.

[59] K. L. McMillan.
An interpolating theorem prover.
Theor. Comput. Sci., 345(1):101–121, 2005.

[60] J. Moeller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard.
Fully symbolic model checking of timed systems using difference decision diagrams.
In *Proc. Workshop on Symbolic Model Checking (SMC), FLoC'99*, Trento, Italy, July 1999.

[61] C. G. Nelson and D. C. Oppen.
Simplification by cooperating decision procedures.
TOPLAS, 1(2):245–257, 1979.

References VIII

[62] G. Nelson and D. Oppen.
Simplification by Cooperating Decision Procedures.
ACM Trans. on Programming Languages and Systems, 1(2):245–257, 1979.

[63] G. Nelson and D. Oppen.
Fast Decision Procedures Based on Congruence Closure.
Journal of the ACM, 27(2):356–364, 1980.

[64] R. Nieuwenhuis and A. Oliveras.
Congruence closure with integer offsets.
In *In 10th Int. Conf. Logic for Programming, Artif. Intell. and Reasoning (LPAR)*, volume 2850 of *LNAI*, pages 78–90. Springer, 2003.

[65] R. Nieuwenhuis and A. Oliveras.
DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic.
In *Proc. CAV'05*, volume 3576 of *LNCS*. Springer, 2005.

[66] R. Nieuwenhuis and A. Oliveras.
On SAT Modulo Theories and Optimization Problems.
In *Proc. Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *LNCS*. Springer, 2006.

[67] P. Pudlák.
Lower bounds for resolution and cutting planes proofs and monotone computations.
J. of Symb. Logic, 62(3), 1997.

[68] H. Rueßand N. Shankar.
Deconstructing Shostak.
In *Proc. LICS '01*. IEEE Computer Society, 2001.

[69] A. Rybalchenko and V. Sofronie-Stokkermans.
Constraint Solving for Interpolation.
In *Proc. VMCAI*, volume 4349 of *LNCS*. Springer, 2007.

[70] R. Sebastiani.
Lazy Satisfiability Modulo Theories.
Journal on Satisfiability, Boolean Modeling and Computation, JSAT, 3(3-4):141–224, 2007.

References IX

[71] R. Sebastiani and S. Tomasi.
Optimization in SMT with LA(Q) Cost Functions.
In *IJCAR*, volume 7364 of *LNAI*, pages 484–498. Springer, July 2012.

[72] R. Sebastiani and S. Tomasi.
Optimization Modulo Theories with Linear Rational Costs.
ACM Transactions on Computational Logics, 16(2), March 2015.

[73] R. Sebastiani and P. Trentin.
OptiMathSAT: A Tool for Optimization Modulo Theories.
In *Proc. International Conference on Computer-Aided Verification, CAV 2015*, volume 9206 of *LNCS*. Springer, 2015.

[74] R. Sebastiani and P. Trentin.
Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions.
In *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'15*, volume 9035 of *LNCS*. Springer, 2015.

[75] R. Shostak.
A Pratical Decision Procedure for Arithmetic with Function Symbols.
Journal of the ACM, 26(2):351–360, 1979.

[76] R. Shostak.
Deciding Combinations of Theories.
Journal of the ACM, 31:1–12, 1984.

[77] K. Stergiou and M. Koubarakis.
Backtracking algorithms for disjunctions of temporal constraints.
In *Proc. AAAI*, pages 248–253, 1998.

[78] A. Stump, C. W. Barrett, and D. L. Dill.
CVC: A Cooperating Validity Checker.
In *Proc. CAV'02*, number 2404 in LNCS. Springer Verlag, 2002.

[79] S. Teso, R. Sebastiani, and A. Passerini.
Structured learning modulo theories.
Artificial Intelligence, 244:166–187, 2017.

References X

[80] S. Wolfman and D. Weld.
The LPSAT Engine & its Application to Resource Planning.
In *Proc. IJCAI*, 1999.

[81] T. Yavuz-Kahveci, M. Tuncer, and T. Bultan.
A Library for Composite Symbolic Representation.
In *Proc. TACAS2001*, volume 2031 of *LNCS*. Springer Verlag, 2000.

[82] L. Zhang and S. Malik.
Extracting small unsatisfiable cores from unsatisfiable boolean formula.
In *Proc. of SAT*, 2003.

Disclaimer

The list of references above is by no means intended to be all-inclusive. I apologize both with the authors and with the readers for all the relevant works which are not cited here.