SMT Solvers: Theory and Practice

Clark Barrett

barrett@cs.nyu.edu

New York University

Exercise

There is a triangle of 15 pegs with one missing. You have to jump pegs until there is only one left.

```
        X
      X   X
    X   O   X
  X   X   X   X
X   X   X   X   X
```

SMT solvers: Motivation

SAT solvers are automatic and efficient.

As a result, they are frequently used as the “engine” behind verification applications.

However, systems are usually designed and modeled at a higher level than the Boolean level and the translation to Boolean logic can be expensive.

A primary goal of research in *Satisfiability Modulo Theories (SMT)* is to create verification engines that can reason natively at a higher level of abstraction, while still retaining the speed and automation of today’s Boolean engines.

Roadmap

Satisfiability Modulo Theories

- First-Order Logic
- Specific Theories
- Theory Solvers
- Combining Theory Solvers
- Combining with SAT
- Modeling in SMT

SMT Solvers: Language

Whereas the language of SAT solvers is Boolean logic, the language of SMT solvers is *first-order logic* [End00].

The language includes the Boolean operations of Boolean logic, but instead of propositional variables, more complicated expressions involving constant, function, and predicate symbols are used.

Examples of predicates

- $f(x) = f(y)$
- $x + y < z$
- $a \in S$

First-Order Logic: Syntax

As with propositional logic, expressions in first-order logic are made up of sequences of symbols.

Symbols are divided into *logical symbols* and *non-logical symbols* or *parameters*.

First-Order Logic: Syntax

Logical Symbols

- Parentheses: $($, $)$
- Propositional connectives: $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$
- Variables: v_1, v_2, $\ldots$
- Quantifiers: $\forall$, $\exists$

Parameters

- Equality symbol (optional): $=$
- Predicate symbols: e.g. $p(x)$, $x > y$
- Constant symbols: e.g. 0, *John*, π
- Function symbols: e.g. $f(x)$, $x + y$, $x +_{[2]} y$

First-Order Logic: Syntax

Each predicate and function symbol has an associated *arity*: a natural number indicating how many arguments it takes.

Equality is a special predicate symbol of arity 2.

Constant symbols can also be thought of as functions whose arity is 0.

A *first-order language* must first specify its parameters.

First-Order Languages: Examples

Propositional Logic

- Equality: *no*
- Predicate symbols: $A_1, A_2, \ldots$
- Constant symbols: *none*
- Function symbols: *none*

Set Theory

- Equality: *yes*
- Predicate symbols: $\in$
- Constant symbols: $\emptyset$
- Function symbols: *none*

First-Order Languages: Examples

Elementary Number Theory

- Equality: *yes*
- Predicate symbols: $<$
- Constant symbols: 0
- Function symbols: S (successor), $+$, $\times$, exp

First-Order Logic: Terms

The first important concept on the way to defining well-formed formulas is that of *terms*.

For each function symbol f of arity n, we define a term-building operation $\mathcal{F}_f$:

$$\mathcal{F}_f(\alpha_1, \ldots, \alpha_n) = f\alpha_1, \ldots, \alpha_n$$

Note that we are using prefix notation to avoid ambiguity.

The set of *terms* is the set of expressions generated from the constant symbols and variables by the $\mathcal{F}_f$ operations.

Terms are expressions which name objects.

Theorem
The set of terms is freely generated from the set of variables and constant symbols by the $\mathcal{F}_f$ operations.

First-Order Logic: Formulas

Atomic Formulas

An *atomic formula* is an expression of the form: $Pt_1, \ldots, t_n$ where P is a predicate symbol of arity n and t_1,. . . ,t_n are terms.

If the language includes the equality symbol, we consider the equality symbol to be a special predicate of arity 2.

Formulas

We define the following formula-building operations:

- $\mathcal{E}_{\neg}(\alpha) = (\neg\alpha)$
- $\mathcal{E}_{\rightarrow}(\alpha, \beta) = (\alpha \rightarrow \beta)$
- $\mathcal{Q}_i(\alpha) = \forall\, v_i\ \alpha$

First-Order Logic: Formulas

The set of *well-formed formulas* is the set of expressions generated from the atomic formulas by the operations $\mathcal{E}_\neg$, $\mathcal{E}_\rightarrow$, and $\mathcal{Q}_i\ i = 1, 2, \ldots$.

This set is also freely generated.

Formula Examples

In the language of elementary number theory introduced above, which of the following are terms?

1. v_6

$v_2 + v_3$ *yes*

$\forall v_1 .\, 0 \times v_1 = 0$

Formula Examples

In the language of elementary number theory introduced above, which of the following are terms?

1. v_6 *yes*
2. $v_2 + v_3$

$v_2 + v_3$ *yes*

$\forall v_1 .\, 0 \times v_1 = 0$

Formula Examples

In the language of elementary number theory introduced above, which of the following are terms?

1. v_6 *yes*
2. $v_2 + v_3$ *yes*
3. $P_1 \wedge P_2$

$$\forall v_1 .\, 0 \times v_1 = 0$$

Formula Examples

In the language of elementary number theory introduced above, which of the following are terms?

1.	v_6	*yes*
2.	$v_2 + v_3$	*yes*
3.	$P_1 \wedge P_2$	*no*

atomic formulas?

1. $(v_1 + 0)^{v_2} = S(v_3)$

$$\forall v_1 . \, 0 \times v_1 = 0$$

Formula Examples

In the language of elementary number theory introduced above, which of the following are terms?

1. v_6 *yes*
2. $v_2 + v_3$ *yes*
3. $P_1 \wedge P_2$ *no*

atomic formulas?

1. $(v_1 + 0)^{v_2} = S(v_3)$ *yes*
2. $\neg(v_2 = v_3)$

$$\forall\, v_1 .\, 0 \times v_1 = 0$$

Formula Examples

In the language of elementary number theory introduced above, which of the following are terms?

1.	v_6	*yes*
2.	$v_2 + v_3$	*yes*
3.	$P_1 \wedge P_2$	*no*

atomic formulas?

1.	$(v_1 + 0)^{v_2} = S(v_3)$	*yes*
2.	$\neg(v_2 = v_3)$	*no*

well-formed formulas?

1. $\neg(v_2 = v_3)$

$$\forall\, v_1 \,.\, 0 \times v_1 = 0$$

Formula Examples

In the language of elementary number theory introduced above, which of the following are terms?

1.	v_6	*yes*
2.	$v_2 + v_3$	*yes*
3.	$P_1 \wedge P_2$	*no*

atomic formulas?

1.	$(v_1 + 0)^{v_2} = S(v_3)$	*yes*
2.	$\neg(v_2 = v_3)$	*no*

well-formed formulas?

1.	$\neg(v_2 = v_3)$	*no*

$$\forall v_1 .\, 0 \times v_1 = 0$$

Formula Examples

In the language of elementary number theory introduced above, which of the following are terms?

1. v_6 *yes*
2. $v_2 + v_3$ *yes*
3. $P_1 \wedge P_2$ *no*

atomic formulas?

1. $(v_1 + 0)^{v_2} = S(v_3)$ *yes*
2. $\neg(v_2 = v_3)$ *no*

well-formed formulas?

1. $\neg(v_2 = v_3)$ *no*
2. $0 \times v_1$

$$\forall\, v_1 .\, 0 \times v_1 = 0$$

Formula Examples

In the language of elementary number theory introduced above, which of the following are terms?

1. v_6 *yes*
2. $v_2 + v_3$ *yes*
3. $P_1 \wedge P_2$ *no*

atomic formulas?

1. $(v_1 + 0)^{v_2} = S(v_3)$ *yes*
2. $\neg(v_2 = v_3)$ *no*

well-formed formulas?

1. $\neg(v_2 = v_3)$ *no*
2. $0 \times v_1$ *no*
3. $\forall v_1 .\, 0 \times v_1 = 0$

Formula Examples

In the language of elementary number theory introduced above, which of the following are terms?

1.	v_6	*yes*
2.	$v_2 + v_3$	*yes*
3.	$P_1 \wedge P_2$	*no*

atomic formulas?

1.	$(v_1 + 0)^{v_2} = S(v_3)$	*yes*
2.	$\neg(v_2 = v_3)$	*no*

well-formed formulas?

1.	$\neg(v_2 = v_3)$	*no*
2.	$0 \times v_1$	*no*
3.	$\forall v_1 .\, 0 \times v_1 = 0$	*yes*

Free and Bound Variables

We define by recursion what it means for a variable x to *occur free* in a *wff* α:

- If α is an atomic formula, then x occurs free in α iff x occurs in α.
- x occurs free in $(\neg\alpha)$ iff x occurs free in α.
- x occurs free in $(\alpha \rightarrow \beta)$ iff x occurs free in α or in β.
- x occurs free in $\forall\, v_i\, \alpha$ iff x occurs free in α and $x \neq v_i$.

If $\forall\, v_i$ appears in α, then v_i is said to be *bound* in α.

If no variable occurs free in a *wff* α, then α is a *sentence*.

First-Order Logic: Semantics

In propositional logic, the truth of a formula was determined by a *truth assignment* over the propositional symbols.

In first-order logic, we use a *model* (also called a *structure*) to determine the truth of a formula.

First-Order Logic: Semantics

A *signature* is a set of non-logical symbols (predicates, constants, and functions). Given a signature Σ, a model M of Σ consists of the following:

1. A nonempty set called the *domain* of M, written $\mathit{dom}(M)$. Elements of $\mathit{dom}(M)$ are also referred to as elements of M.
2. A mapping from each constant c in Σ to an element c^M of M.
3. A mapping from each n-ary function symbol f in Σ to f^M, an n-ary function from $[\mathit{dom}(M)]^n$ to $\mathit{dom}(M)$.
4. A mapping from each n-ary predicate symbol p in Σ to $p^M \subseteq [\mathit{dom}(M)]^n$, an n-ary relation on the set $\mathit{dom}(M)$.

Example

Consider the signature corresponding to the language of set theory which has a single predicate symbol $\in$ and a single constant symbol $\emptyset$.

A possible model M for this signature has $\mathit{dom}(M) = \mathcal{N}$, the set of natural numbers, $\in^M = <$, and $\emptyset^M = 0$.

Now consider the sentence $\exists x \, \forall y \, \neg y \in x$.

Example

Consider the signature corresponding to the language of set theory which has a single predicate symbol $\in$ and a single constant symbol $\emptyset$.

A possible model M for this signature has $\textit{dom}(M) = \mathcal{N}$, the set of natural numbers, $\in^M = <$, and $\emptyset^M = 0$.

Now consider the sentence $\exists\, x\, \forall\, y\, \neg y \in x$.

What does this sentence mean in this model?

Example

Consider the signature corresponding to the language of set theory which has a single predicate symbol $\in$ and a single constant symbol $\emptyset$.

A possible model M for this signature has $\mathit{dom}(M) = \mathcal{N}$, the set of natural numbers, $\in^M = <$, and $\emptyset^M = 0$.

Now consider the sentence $\exists x \, \forall y \, \neg y \in x$.

What does this sentence mean in this model?

The translation of the sentence in the model M is that there is a natural number x such that no other natural number is smaller than x.

Example

Consider the signature corresponding to the language of set theory which has a single predicate symbol $\in$ and a single constant symbol $\emptyset$.

A possible model M for this signature has $\mathit{dom}(M) = \mathcal{N}$, the set of natural numbers, $\in^M = <$, and $\emptyset^M = 0$.

Now consider the sentence $\exists x \, \forall y \, \neg y \in x$.

What does this sentence mean in this model?

The translation of the sentence in the model M is that there is a natural number x such that no other natural number is smaller than x.

Is this sentence true in the model?

Example

Consider the signature corresponding to the language of set theory which has a single predicate symbol $\in$ and a single constant symbol $\emptyset$.

A possible model M for this signature has *dom*$(M) = \mathcal{N}$, the set of natural numbers, $\in^M = <$, and $\emptyset^M = 0$.

Now consider the sentence $\exists\, x\, \forall\, y\, \neg y \in x$.

What does this sentence mean in this model?

The translation of the sentence in the model M is that there is a natural number x such that no other natural number is smaller than x.

Is this sentence true in the model?

Since 0 has this property, the sentence is true in this model.

First-Order Logic: Semantics

We will often use a shorthand when discussing both signatures and models. The signature shorthand lists each symbol in the signature.

The model shorthand lists the domain and the interpretation of each symbol of the signature.

The signature for set theory can thus be described as ($\in$, $\emptyset$), and the above model as ($\mathcal{N}$, $<$, 0).

First-Order Logic: Semantics

Given a model M, a *variable assignment* s is a function which assigns to each variable an element of M.

Given a wff ϕ, we say that M *satisfies* ϕ with s and write $\models_M \phi[s]$ if ϕ is true in the model M with variable assignment s.

To define this formally, we first define the extension $\overline{s} : T \to \mathit{dom}(M)$, a function from the set T of all terms into the domain of M:

1. For each variable x, $\overline{s}(x) = s(x)$.
2. For each constant symbol c, $\overline{s}(c) = c^M$.
3. If t_1,...,t_n are terms and f is an n-ary function symbol, then $\overline{s}(ft_1, \ldots, t_n) = f^M(\overline{s}(t_1), \ldots, \overline{s}(t_n))$.

First-Order Logic: Semantics

Atomic Formulas

1. $\models_M = t_1 t_2[s]$ iff $\overline{s}(t_1) = \overline{s}(t_2)$.
2. For an n-ary predicate symbol P,
   $\models_M P t_1, \ldots, t_n[s]$ iff $\langle \overline{s}(t_1), \ldots, \overline{s}(t_n) \rangle \in P^M$.

Other *wffs*

1. $\models_M (\neg \phi)[s]$ iff $\not\models_M \phi[s]$.
2. $\models_M (\phi \to \psi)[s]$ iff $\not\models_M \phi[s]$ or $\models_M \psi[s]$.
3. $\models_M \forall x\, \phi[s]$ iff $\models_M \phi[s(x|d)]$ for every $d \in$ *dom*(M).

$s(x|d)$ signifies the function which is the same as s everywhere except at x where its value is d.

Logical Definitions

Suppose Σ is a signature. A Σ-*formula* is a well-formed formula whose non-logical symbols are contained in Σ.

Let Γ be a set of Σ-formulas. We write $\models_M \Gamma[s]$ to signify that $\models_M \phi[s]$ for every $\phi \in \Gamma$.

If Γ is a set of Σ-formulas and ϕ is a Σ-formula, then Γ *logically implies* ϕ, written $\Gamma \models \phi$, iff for every model M of Σ and every variable assignment s, if $\models_M \Gamma[s]$ then $\models_M \phi[s]$.

We write $\psi \models \phi$ as an abbreviation for $\{\psi\} \models \phi$.

ψ and ϕ are *logically equivalent* (written $\psi \models\!\!\dashv \phi$) iff $\psi \models \phi$ and $\phi \models \psi$.

A Σ-formula ϕ is *valid*, written $\models \phi$ iff $\emptyset \models \phi$ (i.e. $\models_M \phi[s]$ for every M and s).

Examples

Suppose that P is a unary predicate and Q a binary predicate. Which of the following are true?

1. $\forall v_1\, Pv_1 \models Pv_2$

$\forall x\, \exists y\, Qxy \models \exists y\, \forall x\, Qxy$

Examples

Suppose that P is a unary predicate and Q a binary predicate. Which of the following are true?

1. $\forall v_1\, Pv_1 \models Pv_2$ *True*
2. $Pv_1 \models \forall v_1\, Pv_1$

$$\forall x\, \exists y\, Qxy \models \exists y\, \forall x\, Qxy$$

Examples

Suppose that P is a unary predicate and Q a binary predicate. Which of the following are true?

1. $\forall v_1\, Pv_1 \models Pv_2$ *True*
2. $Pv_1 \models \forall v_1\, Pv_1$ *False*
3. $\forall v_1\, Pv_1 \models \exists v_2\, Pv_2$

$$\forall x\, \exists y\, Qxy \models \exists y\, \forall x\, Qxy$$

Examples

Suppose that P is a unary predicate and Q a binary predicate. Which of the following are true?

1. $\forall v_1\, Pv_1 \models Pv_2$ *True*
2. $Pv_1 \models \forall v_1\, Pv_1$ *False*
3. $\forall v_1\, Pv_1 \models \exists v_2\, Pv_2$ *True*
4. $\exists x\, \forall y\, Qxy \models \forall y\, \exists x\, Qxy$

$\forall x\, \exists y\, Qxy \models \exists y\, \forall x\, Qxy$

Examples

Suppose that P is a unary predicate and Q a binary predicate. Which of the following are true?

1. $\forall v_1\, Pv_1 \models Pv_2$ *True*
2. $Pv_1 \models \forall v_1\, Pv_1$ *False*
3. $\forall v_1\, Pv_1 \models \exists v_2\, Pv_2$ *True*
4. $\exists x\, \forall y\, Qxy \models \forall y\, \exists x\, Qxy$ *True*
5. $\forall x\, \exists y\, Qxy \models \exists y\, \forall x\, Qxy$

Examples

Suppose that P is a unary predicate and Q a binary predicate. Which of the following are true?

1. $\forall v_1\, Pv_1 \models Pv_2$ *True*
2. $Pv_1 \models \forall v_1\, Pv_1$ *False*
3. $\forall v_1\, Pv_1 \models \exists v_2\, Pv_2$ *True*
4. $\exists x\, \forall y\, Qxy \models \forall y\, \exists x\, Qxy$ *True*
5. $\forall x\, \exists y\, Qxy \models \exists y\, \forall x\, Qxy$ *False*
6. $\models \exists x\, (Px \to \forall y\, Py)$

Examples

Suppose that P is a unary predicate and Q a binary predicate. Which of the following are true?

1. $\forall v_1\, Pv_1 \models Pv_2$ *True*
2. $Pv_1 \models \forall v_1\, Pv_1$ *False*
3. $\forall v_1\, Pv_1 \models \exists v_2\, Pv_2$ *True*
4. $\exists x \forall y\, Qxy \models \forall y \exists x\, Qxy$ *True*
5. $\forall x \exists y\, Qxy \models \exists y \forall x\, Qxy$ *False*
6. $\models \exists x\, (Px \rightarrow \forall y\, Py)$ *True*

Which models satisfy the following sentences?

1. $\forall x \forall y\, x = y$

Examples

Suppose that P is a unary predicate and Q a binary predicate. Which of the following are true?

1. $\forall v_1\, Pv_1 \models Pv_2$ *True*
2. $Pv_1 \models \forall v_1\, Pv_1$ *False*
3. $\forall v_1\, Pv_1 \models \exists v_2\, Pv_2$ *True*
4. $\exists x\, \forall y\, Qxy \models \forall y\, \exists x\, Qxy$ *True*
5. $\forall x\, \exists y\, Qxy \models \exists y\, \forall x\, Qxy$ *False*
6. $\models \exists x\, (Px \rightarrow \forall y\, Py)$ *True*

Which models satisfy the following sentences?

1. $\forall x\, \forall y\; x = y$ *Models with exactly one element*.
2. $\forall x\, \forall y\, Qxy$

Examples

Suppose that P is a unary predicate and Q a binary predicate. Which of the following are true?

1. $\forall v_1\, Pv_1 \models Pv_2$ *True*
2. $Pv_1 \models \forall v_1\, Pv_1$ *False*
3. $\forall v_1\, Pv_1 \models \exists v_2\, Pv_2$ *True*
4. $\exists x\, \forall y\, Qxy \models \forall y\, \exists x\, Qxy$ *True*
5. $\forall x\, \exists y\, Qxy \models \exists y\, \forall x\, Qxy$ *False*
6. $\models \exists x\, (Px \rightarrow \forall y\, Py)$ *True*

Which models satisfy the following sentences?

1. $\forall x\, \forall y\, x = y$ *Models with exactly one element*.
2. $\forall x\, \forall y\, Qxy$ *Models* (A, R) *where* $R = A \times A$.
3. $\forall x\, \exists y\, Qxy$

Examples

Suppose that P is a unary predicate and Q a binary predicate. Which of the following are true?

1.	$\forall v_1\, Pv_1 \models Pv_2$	*True*
2.	$Pv_1 \models \forall v_1\, Pv_1$	*False*
3.	$\forall v_1\, Pv_1 \models \exists v_2\, Pv_2$	*True*
4.	$\exists x\, \forall y\, Qxy \models \forall y\, \exists x\, Qxy$	*True*
5.	$\forall x\, \exists y\, Qxy \models \exists y\, \forall x\, Qxy$	*False*
6.	$\models \exists x\, (Px \to \forall y\, Py)$	*True*

Which models satisfy the following sentences?

1.	$\forall x\, \forall y\; x = y$	*Models with exactly one element*.
2.	$\forall x\, \forall y\, Qxy$	*Models* (A, R) *where* $R = A \times A$.
3.	$\forall x\, \exists y\, Qxy$	*Models* (A, R) *where* $\mathsf{dom}(R) = A$.

Validity and Satisfiability Modulo Theories

A *theory* is a set of sentences. For a given signature Σ, a Σ*-theory* is a set of sentences, each of which is a Σ-formula.

We will assume for convenience that theories are *closed under logical implication*.

Given a Σ-theory T, a Σ-formula ϕ is

1. T*-valid* if $\models_M \phi[s]$ for all models M of T and all variable assignments s.
2. T*-satisfiable* if there exists some model M of T and variable assignment s such that $\models_M \phi[s]$.
3. T*-unsatisfiable* if $\not\models_M \phi[s]$ for all models M of T and all variable assignments s.

Validity and Satisfiability Modulo Theories

The *validity problem* for T is the problem of deciding, for each Σ-formula ϕ, whether ϕ is T-valid.

The *satisfiability problem* for T is the problem of deciding, for each Σ-formula ϕ, whether ϕ is T-satisfiable.

Similarly, one can define the *quantifier-free validity problem* and the *quantifier-free satisfiability problem* for a Σ-theory T by restricting the formula ϕ to be quantifier-free.

Note that validity problems can always be reduced to satisfiability problems:

ϕ is T-valid iff $\neg\phi$ is T-unsatisfiable.

We will consider a few examples of theories which are of particular interest in verification applications [MZ03].

Roadmap

Satisfiability Modulo Theories

- First-Order Logic
- Specific Theories
- Theory Solvers
- Combining Theory Solvers
- Combining with SAT
- Modeling in SMT

The Theory $T_{\mathcal{E}}$ of Equality

The theory $T_{\mathcal{E}}$ of equality is the empty theory.

The theory does not restrict the possible values of symbols in any way. For this reason, it is sometimes called the theory of *equality with uninterpreted functions (EUF)*.

The satisfiability problem for $T_{\mathcal{E}}$ is just the satisfiability problem for first order logic, which is undecidable.

The satisfiability problem for conjunctions of literals in $T_{\mathcal{E}}$ is decidable in polynomial time using *congruence closure*.

The Theory $T_{\mathcal{Z}}$ of Integers

Let $\Sigma_{\mathcal{Z}}$ be the signature $(0, 1, +, -, \leq)$.

Let $\mathcal{A}_{\mathcal{Z}}$ be the standard model of the integers with domain $\mathcal{Z}$.

Then $T_{\mathcal{Z}}$ is defined to be the set of all $\Sigma_{\mathcal{Z}}$-sentences true in the model $A_{\mathcal{Z}}$.

As showed by Presburger in 1929, the general satisfiability problem for $T_{\mathcal{Z}}$ is decidable, but its complexity is triply-exponential.

The quantifier-free satisfiability problem for $T_{\mathcal{Z}}$ is “only” NP-complete.

The Theory $T_{\mathcal{Z}}$ of Integers

Let $\Sigma_{\mathcal{Z}}^{\times}$ be the same as $\Sigma_{\mathcal{Z}}$ with the addition of the symbol $\times$ for multiplication, and define $\mathcal{A}_{\mathcal{Z}}^{\times}$ and $T_{\mathcal{Z}}^{\times}$ in the obvious way.

The satisfiability problem for $T_{\mathcal{Z}}^{\times}$ is undecidable (a consequence of Gödel's incompleteness theorem).

In fact, even the quantifier-free satisfiability problem for $T_{\mathcal{Z}}^{\times}$ is undecidable.

The Theory $T_{\mathcal{R}}$ of Reals

Let $\Sigma_{\mathcal{R}}$ be the signature $(0, 1, +, -, \leq)$.

Let $\mathcal{A}_{\mathcal{R}}$ be the standard model of the reals with domain $\mathcal{R}$.

Then $T_{\mathcal{R}}$ is defined to be the set of all $\Sigma_{\mathcal{R}}$-sentences true in the model $A_{\mathcal{R}}$.

The satisfiability problem for $T_{\mathcal{R}}$ is decidable, but the complexity is doubly-exponential.

The quantifier-free satisfiability problem for conjunctions of literals (atomic formulas or their negations) in $T_{\mathcal{R}}$ is solvable in polynomial time, though exponential methods (like Simplex or Fourier-Motzkin) tend to perform best in practice.

The Theory $T_{\mathcal{R}}$ of Reals

Let $\Sigma_{\mathcal{R}}^{\times}$ be the same as $\Sigma_{\mathcal{R}}$ with the addition of the symbol $\times$ for multiplication, and define $\mathcal{A}_{\mathcal{R}}^{\times}$ and $T_{\mathcal{R}}^{\times}$ in the obvious way.

In contrast to the theory of integers, the satisfiability problem for $T_{\mathcal{R}}^{\times}$ is decidable though the complexity is inherently doubly-exponential.

The Theory $T_{\mathcal{A}}$ of Arrays

Let $\Sigma_{\mathcal{A}}$ be the signature $(\mathit{read}, \mathit{write})$.

Let $\Lambda_{\mathcal{A}}$ be the following axioms:

$$\forall a \, \forall i \, \forall v \, (\mathit{read}(\mathit{write}(a, i, v), i) = v)$$
$$\forall a \, \forall i \, \forall j \, \forall v \, (i \neq j \rightarrow \mathit{read}(\mathit{write}(a, i, v), j) = \mathit{read}(a, j))$$
$$\forall a \, \forall b \, ((\forall i \, (\mathit{read}(a, i) = \mathit{read}(b, i))) \rightarrow a = b)$$

Then $T_{\mathcal{A}} = \mathit{Cn} \ \Lambda_{\mathcal{A}}$.

The satisfiability problem for $T_{\mathcal{A}}$ is undecidable, but the quantifier-free satisfiability problem for $T_{\mathcal{A}}$ is decidable (the problem is NP-complete).

Theories of Inductive Data Types

An *inductive data type* (IDT) defines one or more *constructors*, and possibly also *selectors* and *testers*.

Example: *list* **of** *int*

- Constructors: $\mathit{cons} : (\mathit{int}, \mathit{list}) \longrightarrow \mathit{list}$, $\mathit{null} : \mathit{list}$
- Selectors: $\mathit{car} : \mathit{list} \longrightarrow \mathit{int}$, $\mathit{cdr} : \mathit{list} \longrightarrow \mathit{list}$
- Testers: *is_cons*, *is_null*

The *first order theory* of a inductive data type associates a function symbol with each constructor and selector and a predicate symbol with each tester.

Example: $\forall\, x : \mathit{list}.\ (x = \mathit{null} \vee \exists\, y : \mathit{int}, z : \mathit{list}.\ x = \mathit{cons}(y, z))$

Theories of Inductive Data Types

An *inductive data type* (IDT) defines one or more *constructors*, and possibly also *selectors* and *testers*.

Example: *list* **of** *int*

- Constructors: $cons : (int, list) \longrightarrow list$, $null : list$
- Selectors: $car : list \longrightarrow int$, $cdr : list \longrightarrow list$
- Testers: *is_cons*, *is_null*

For IDTs with a single constructor, a conjunction of literals is decidable in polynomial time [Opp80].

For more general IDTs, the problem is NP complete, but reasonbly efficient algorithms exist in practice [ZSM04a, ZSM04b, BST07].

Other Interesting Theories

Some other interesting theories include:

- Theories of bit-vectors
  [CMR97, Möl97, BDL98, BP98, EKM98, GBD05]
- Fragments of set theory [CZ00]
- Theories of pointers and reachability:
  [RBH07, YRS$^+$06, LQ08]

Roadmap

Satisfiability Modulo Theories

- First-Order Logic
- Specific Theories
- Theory Solvers
- Combining Theory Solvers
- Combining with SAT
- Modeling in SMT

Theory Solvers

The theory reasoning in an SMT solver is done with a *theory solver*.

Given a Σ-theory T, a theory solver for T takes as input a set of Σ-literals and determines whether the set is satisfiable or unsatisfiable.

We next consider some examples of theory solvers.

Congruence Closure [NO80]

Let $G = (V, E)$ be a directed graph such that for each vertex v in G, the successors of v are ordered.

Let C be any equivalence relation on V.

The *congruence closure* C^* of C is the finest equivalence relation on V that contains C and satisfies the following property for all vertices v and w:

> Let v and w have successors $v_1, \ldots, v_k$ and $w_1, \ldots, w_l$ respectively. If $k = l$ and $(v_i, w_i) \in C^*$ for $1 \leq i \leq k$, then $(v, w) \in C^*$.

Congruence Closure

In other words, if the corresponding successors of v and w are equivalent under C^*, then v and w are themselves equivalent under C^*.
Often, the vertices are labeled by some labeling function λ. In this case, the property becomes:

> If $\lambda(v) = \lambda(w)$ and if $k = l$ and $(v_i, w_i) \in C^*$ for $1 \leq i \leq k$, then $(v, w) \in C^*$.

A Simple Algorithm

Let $C_0 = C$ and $i = 0$.

1. Number the equivalence classes in C_i starting with 1.
2. Let α assign to each vertex v the number $\alpha(v)$ of the equivalence class containing v.
3. For each vertex v construct a *signature* $s(v) = \lambda(v)(\alpha(v_1), \ldots, \alpha(v_k))$, where $v_1, \ldots, v_k$ are the successors of v.
4. Group the vertices into classes of vertices having equal signatures.
5. Let C_{i+1} be the finest equivalence relation on V such that two vertices equivalent under C_i or having the same signature are equivalent under C_{i+1}.
6. If $C_{i+1} = C_i$, let $C^* = C_i$; otherwise increment i and repeat.

Congruence Closure and $T_{\mathcal{E}}$

Recall that $T_{\mathcal{E}}$ is the empty theory with equality over some signature Σ containing only function symbols.

If Γ is a set of ground Σ-equalities and Δ is a set of ground Σ-disequalities, then the satisfiability of $\Gamma \cup \Delta$ can be determined as follows.

- Let G be a graph which corresponds to the abstract syntax trees of terms in $\Gamma \cup \Delta$, and let v_t denote the vertex of G associated with the term t.
- Let C be the equiavlence relation on the vertices of G induced by Γ.
- $\Gamma \cup \Delta$ is satisfiable iff for each $s \neq t \in \Delta$, $(v_s, v_t) \notin C^*$.

An Algorithm for $T_{\mathcal{E}}$

union and *find* are abstract operations for manipulating equivalence classes.

union(x, y) merges the equivalence classes of x and y.

find(x) returns a unique representative of the equivalence class of x.

```
cc(Γ, Δ)
    Construct $G(V, E)$ from terms in $\Gamma$ and $\Delta$.
    while $\Gamma \neq \emptyset$
        Remove some equality $a = b$ from $\Gamma$;
        Merge$(a, b)$;
    if find$(a)$ = find$(b)$ for some $a \neq b \in \Delta$ then
        return False;
    return True;
```

An Algorithm for $T_{\mathcal{E}}$

```
Merge$(a, b)$
    if find$(a)$ = find$(b)$ then return;
    Let $A$ be the set of all predecessors
      of all vertices equivalent to $a$;
    Let $B$ be the set of all predecessors
      of all vertices equivalent to $b$;
    union$(a, b)$;
    foreach $x \in A$ and $y \in B$
      if $s(x) = s(y)$ then Merge$(x, y)$;
```

Shostak's Method

In 1984 [Sho84], Robert Shostak published a paper which detailed a particular strategy for deciding satisfiability of quantifier-free formulas in certain kinds of theories.

The original version promises three main things:

1. For theories T which meet the criteria (we will call these *Shostak theories*), the method gives a decision procedure for quantifier-free T-satisfiability.
2. The method has the theory $T_{\mathcal{E}}$ “built-in”, so for any Shostak theory T, the method gives a decision procedure for quantifier-free $T \cup T_{\mathcal{E}}$-satisfiability.
3. Any two Shostak theories T_1 and T_2 can be combined to form a new Shostak theory $T_1 \cup T_2$.

Shostak's Method

Unfortunately, the original paper contains many errors and a number of papers have since been dedicated to correcting them [CLS96, RS01, Gan02, BDS02b, KC03].

When the dust settled, it finally became clear that the first two claims can be justified, but the last one is false.

Most proponents of the method now agree that any attempt to *combine* theories is best understood in the context of the Nelson-Oppen method.

However, in this context, there is much that can be learned from Shostak's method.

Shostak's Method

The first helpful insight is how to build a decision procedure for a single Shostak theory.

Recall that the Nelson-Oppen method gives a decision procedure for a combination of theories given decision procedures for the component theories.

However, the Nelson-Oppen method provides no help on how to build the decision procedures for the component theories.

Shostak provides one solution for the special case of *Shostak theories*.

Shostak Theories

A consistent theory T with signature Σ is a *Shostak theory* if the following conditions hold.

Shostak Theories

A consistent theory T with signature Σ is a *Shostak theory* if the following conditions hold.

1. Σ does not contain any predicate symbols.

Shostak Theories

A consistent theory T with signature Σ is a *Shostak theory* if the following conditions hold.

1. Σ does not contain any predicate symbols.
2. T is *convex*.

Shostak Theories

A consistent theory T with signature Σ is a *Shostak theory* if the following conditions hold.

1. Σ does not contain any predicate symbols.
2. T is *convex*.
   Recall that a theory T is *convex* if for any conjunction of literals φ and variables $x_1, \ldots x_n, y_1, \ldots, y_n$,

   $T \cup \phi \models x_1 = y_1 \vee \cdots \vee x_n = y_n$ implies
   $T \cup \phi \models x_i = y_i$ for some $1 \leq i \leq n$.

Shostak Theories

A consistent theory T with signature Σ is a *Shostak theory* if the following conditions hold.

1. Σ does not contain any predicate symbols.
2. T is *convex*.
3. T has a *solver* solve.

Shostak Theories

A consistent theory T with signature Σ is a *Shostak theory* if the following conditions hold.

1. Σ does not contain any predicate symbols.
2. T is *convex*.
3. T has a *solver* *solve*.
   The solver *solve* must be a computable function from Σ-equations to sets of Σ-formulas defined as follows:
   (a) If $T \models a \neq b$, then *solve*$(a = b) \equiv \{$*False*$\}$.
   (b) Otherwise, *solve*$(a = b)$ returns a set $\mathcal{E}$ of equations *in solved form* such that $T \models [(a = b) \leftrightarrow \exists\, \overline{w}.\, \mathcal{E}]$, where $\overline{w}$ are fresh variables not appearing in a or b.

Shostak Theories

A consistent theory T with signature Σ is a *Shostak theory* if the following conditions hold.

1. Σ does not contain any predicate symbols.
2. T is *convex*.
3. T has a *solver* *solve*.
   The solver *solve* must be a computable function from Σ-equations to sets of Σ-formulas defined as follows:
   (a) If $T \models a \neq b$, then $\mathit{solve}(a = b) \equiv \{\mathit{False}\}$.
   (b) Otherwise, $\mathit{solve}(a = b)$ returns a set $\mathcal{E}$ of equations *in solved form* such that $T \models [(a = b) \leftrightarrow \exists \overline{w}.\, \mathcal{E}]$, where $\overline{w}$ are fresh variables not appearing in a or b.

   $\mathcal{E}$ is in *solved form* iff the left-hand side of each equation in $\mathcal{E}$ is a variable which appears only once in $\mathcal{E}$.

Shostak Theories

A consistent theory T with signature Σ is a *Shostak theory* if the following conditions hold.

1. Σ does not contain any predicate symbols.
2. T is *convex*.
3. T has a *solver* *solve*.
   The solver *solve* must be a computable function from Σ-equations to sets of Σ-formulas defined as follows:
   (a) If $T \models a \neq b$, then *solve*$(a = b) \equiv \{$*False*$\}$.
   (b) Otherwise, *solve*$(a = b)$ returns a set $\mathcal{E}$ of equations *in solved form* such that $T \models [(a = b) \leftrightarrow \exists \overline{w}.\, \mathcal{E}]$, where $\overline{w}$ are fresh variables not appearing in a or b.

   We denote by $\mathcal{E}(X)$ the result of applying $\mathcal{E}$ as a substitution to X.

Shostak Theories

A consistent theory T with signature Σ is a *Shostak theory* if the following conditions hold.

1. Σ does not contain any predicate symbols.
2. T is *convex*.
3. T has a *solver* solve.
4. T has a *canonizer* canon.

Shostak Theories

A consistent theory T with signature Σ is a *Shostak theory* if the following conditions hold.

1. Σ does not contain any predicate symbols.
2. T is *convex*.
3. T has a *solver* *solve*.
4. T has a *canonizer* *canon*.
   The canonizer *canon* must be a computable function from Σ-terms to Σ-terms with the property that

   $$T \models a = b \text{ iff } \mathit{canon}(a) \equiv \mathit{canon}(b).$$

Algorithm Sh

Algorithm Sh checks the satisfiability in T of a set of equalities, Γ, and an set of disequalities, Δ.

```
Sh(Γ, Δ, canon, solve)
1.  $\mathcal{E}$ := $\emptyset$;
2.  while $\Gamma \neq \emptyset$ do begin
3.    Remove some equality $a = b$ from $\Gamma$;
4.    $a^*$ := $\mathcal{E}(a)$; $b^*$ := $\mathcal{E}(b)$;
5.    $\mathcal{E}^*$ := $solve(a^* = b^*)$;
6.    if $\mathcal{E}^* = \{False\}$ then return $False$;
7.    $\mathcal{E}$ := $\mathcal{E}^*(\mathcal{E}) \cup \mathcal{E}^*$;
8.  end
9.  if $canon(\mathcal{E}(a)) \equiv canon(\mathcal{E}(b))$ for some $a \neq b \in \Delta$
      then return $False$;
10. return $True$;
```

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$-x - 3y + 2z = 1$	$-x - 3y + 2z = 1$
$x - y - 6z = 1$	
$2x + y - 10z = 3$	

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$-x - 3y + 2z = 1$	$-x - 3y + 2z = 1$
$x - y - 6z = 1$	
$2x + y - 10z = 3$	

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$x = -3y + 2z + 1$	$-x - 3y + 2z = 1$
$x - y - 6z = 1$	
$2x + y - 10z = 3$	

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$x - y - 6z = 1$	$x = -3y + 2z + 1$
$2x + y - 10z = 3$	
$2x + y - 10z = 3$	

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$x - y - 6z = 1$	$x = -3y + 2z + 1$
$2x + y - 10z = 3$	
$2x + y - 10z = 3$	

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$-4y - 4z + 1 = 1$	$x = -3y + 2z + 1$
$2x + y - 10z = 3$	
$2x + y - 10z = 3$	

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$y = -z$	$x = -3y + 2z + 1$
$2x + y - 10z = 3$	
$2x + y - 10z = 3$	

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$y = -z$	$x = 3z + 2z + 1$
$2x + y - 10z = 3$	
$2x + y - 10z = 3$	$x = -3y + 2z + 1$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$2x + y - 10z = 3$	$x = 5z + 1$
	$y = -z$
$2x + y - 10z = 3$	$x = -3y + 2z + 1$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$2x + y - 10z = 3$	$x = 5z + 1$
	$y = -z$
$2x + y - 10z = 3$	$x = -3y + 2z + 1$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$z = -1$	$x = 5z + 1$
	$y = -z$
$2x + y - 10z = 3$	$x = -3y + 2z + 1$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
$z = -1$	$x = 5(-1) + 1$ $y = -(-1)$
$2x + y - 10z = 3$	$x = -3y + 2z + 1$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
	$x = -4$
	$y = 1$
$2x + y - 10z = 3$	$z = -1$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.

Γ	$\mathcal{E}$
	$x = -4$
	$y = 1$
$2x + y - 10z = 3$	$z = -1$

Note that for this theory, the main loop of Shostak's algorithm is equivalent to Gaussian elimination with back-substitution.

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.
- Step 2: Use $\mathcal{E}$ and *canon* to check if any disequality is violated:

  For each $a \neq b \in \Delta$, check if $\mathit{canon}(\mathcal{E}(a)) \equiv \mathit{canon}(\mathcal{E}(b))$.

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.
- Step 2: Use $\mathcal{E}$ and *canon* to check if any disequality is violated:

  For each $a \neq b \in \Delta$, check if *canon*$(\mathcal{E}(a)) \equiv$ *canon*$(\mathcal{E}(b))$.

$\mathcal{E}$	Δ
$x = -4$	$x \neq 4y$
$y = 1$	$x + w \neq w + z - 3y$
$z = -1$	$-4 + w \neq w + (-1) - 3(1)$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.
- Step 2: Use $\mathcal{E}$ and *canon* to check if any disequality is violated:

  For each $a \neq b \in \Delta$, check if $\mathit{canon}(\mathcal{E}(a)) \equiv \mathit{canon}(\mathcal{E}(b))$.

$\mathcal{E}$	Δ
$x = -4$	$x \neq 4y$
$y = 1$	$x + w \neq w + z - 3y$
$z = -1$	$-4 + w \neq w + (-1) - 3(1)$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.
- Step 2: Use $\mathcal{E}$ and *canon* to check if any disequality is violated:

  For each $a \neq b \in \Delta$, check if *canon*$(\mathcal{E}(a)) \equiv$ *canon*$(\mathcal{E}(b))$.

$\mathcal{E}$	Δ
$x = -4$	$-4 \neq 4(1)$
$y = 1$	$x + w \neq w + z - 3y$
$z = -1$	$-4 + w \neq w + (-1) - 3(1)$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.
- Step 2: Use $\mathcal{E}$ and *canon* to check if any disequality is violated:

  For each $a \neq b \in \Delta$, check if *canon*$(\mathcal{E}(a)) \equiv$ *canon*$(\mathcal{E}(b))$.

$\mathcal{E}$	Δ
$x = -4$	$-4 \neq 4$
$y = 1$	$x + w \neq w + z - 3y$
$z = -1$	$-4 + w \neq w + (-1) - 3(1)$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.
- Step 2: Use $\mathcal{E}$ and *canon* to check if any disequality is violated:

  For each $a \neq b \in \Delta$, check if *canon*$(\mathcal{E}(a)) \equiv$ *canon*$(\mathcal{E}(b))$.

$\mathcal{E}$	Δ
$x = -4$	$-4 \neq 4$
$y = 1$	$x + w \neq w + z - 3y$
$z = -1$	$-4 + w \neq w + (-1) - 3(1)$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.
- Step 2: Use $\mathcal{E}$ and *canon* to check if any disequality is violated:

  For each $a \neq b \in \Delta$, check if $\mathit{canon}(\mathcal{E}(a)) \equiv \mathit{canon}(\mathcal{E}(b))$.

$\mathcal{E}$	Δ
$x = -4$	$-4 \neq 4$
$y = 1$	$-4 + w \neq w + (-1) - 3(1)$
$z = -1$	$-4 + w \neq w + (-1) - 3(1)$

Example

The most obvious example of a Shostak theory is $T_{\mathcal{R}}$ (without $\leq$).

- Step 1: Use the solver to convert Γ into an equisatisfiable set $\mathcal{E}$ of equations in solved form.
- Step 2: Use $\mathcal{E}$ and *canon* to check if any disequality is violated:

  For each $a \neq b \in \Delta$, check if *canon*$(\mathcal{E}(a)) \equiv$ *canon*$(\mathcal{E}(b))$.

$\mathcal{E}$	Δ
$x = -4$	$-4 \neq 4$
$y = 1$	$w + (-4) \neq w + (-4)$
$z = -1$	$-4 + w \neq w + (-1) - 3(1)$

Other Shostak Theories

A few other theories can be handled using this algorithm:

- $T_{\mathcal{Z}}$ (without $\leq$) is also a Shostak theory.
- A simple theory of lists (without the NULL list).

However, the idea of using solvers and canonizers can be applied to help decide other theories as well:

- One component in decision procedures for $T_{\mathcal{R}}$ and $T_{\mathcal{Z}}$ (with $\leq$).
- Partial canonizing and solving is useful in $T_{\mathcal{A}}$.
- Partial canonizing and solving is useful for the theory of bit-vectors.

Shostak and Theory Combination

As mentioned, Shostak's second claim is that a combination with $T_{\mathcal{E}}$ can be easily achieved.

The details are a bit technical, and the easiest way to understand it is as a special case of a Nelson-Oppen combination.

The *special* part is that the abstract Nelson-Oppen is refined in several ways that make implementation easier.

We will look at this next.

Shostak's Method: Summary

Shostak's method provides

- a simple decision procedure for Shostak theories
- insight into the usefulness of solvers and canonizers
- insight into practical ways to refine Nelson-Oppen (next)

Shostak's method does *not* provide

- a general method for combining theories

Roadmap

Satisfiability Modulo Theories

- First-Order Logic
- Specific Theories
- Theory Solvers
- Combining Theory Solvers
- Combining with SAT
- Modeling in SMT

The Nelson-Oppen Method

A very general method for combining decision procedures is the *Nelson-Oppen* method [NO79, TH96].

This method is applicable when

1. The signatures Σ_i are disjoint.
2. The theories T_i are stably-infinite.

   A Σ-theory T is *stably-infinite* if every T-satisfiable quantifier-free Σ-formula is satisfiable in an infinite model.
3. The formulas to be tested for satisfiability are conjunctions of quantifier-free literals.

In practice, only the requirement that formulas be quantifier-free is a significant restriction.

The Nelson-Oppen Method

Definitions

1. A member of Σ_i is an i-*symbol*.
2. A term t is an i-*term* if it starts with an i-symbol.
3. An *atomic* i-*formula* is an application of an i-predicate , an equation whose lhs is an i-term, or an equation whose lhs is a variable and whose rhs is an i-term.
4. An i-*literal* is an atomic i-formula or the negation of one.
5. An occurrence of a term t in either an i-term or an i-literal is i-*alien* if t is a j-term with $i \neq j$ and all of its super-terms (if any) are i-terms.
6. An expression is *pure* if it contains only variables and i-symbols for some i.

The Nelson-Oppen Method

Now we can explain step one of the Nelson-Oppen method:

1. Conversion to Separate Form

Given a conjunction of literals, ϕ, we desire to convert it into a *separate form*: a T-equisatisfiable conjunction of literals $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$, where each ϕ_i is a Σ_i-formula.

The following algorithm accomplishes this:

1. Let ψ be some literal in ϕ.
2. If ψ is a pure i-literal, for some i, remove ψ from ϕ and add ψ to ϕ_i; if ϕ is empty then stop; otherwise goto step 1.
3. Otherwise, ψ is an i-literal for some i. Let t be a term occuring i-alien in ψ. Replace t in ϕ with a new variable z, and add $z = t$ to ϕ. Goto step 1.

The Nelson-Oppen Method

It is easy to see that ϕ is T-satisfiable iff $\phi_1 \wedge \cdots \wedge \phi_n$ is T-satisfiable.

Furthermore, because each ϕ_i is a Σ_i-formula, we can run *Sat*$_i$ on each ϕ_i.

Clearly, if *Sat*$_i$ reports that any ϕ_i is unsatisfiable, then ϕ is unsatisfiable.

But the converse is not true in general.

We need a way for the decision procedures to communicate with each other about shared variables.

First a definition: If S is a set of terms and $\sim$ is an equivalence relation on S, then the *arrangement of S induced by* $\sim$ is $Ar_\sim = \{x = y \mid x \sim y\} \cup \{x \neq y \mid x \not\sim y\}$.

The Nelson-Oppen Method

Suppose that T_1 and T_2 are theories with disjoint signatures Σ_1 and Σ_2 respectively. Let $T = \mathit{Cn} \bigcup T_i$ and $\Sigma = \bigcup \Sigma_i$. Given a Σ-formula ϕ and decision procedures Sat_1 and Sat_2 for T_1 and T_2 respectively, we wish to determine if ϕ is T-satisfiable. The non-deterministic Nelson-Oppen algorithm for this is as follows:

1. Convert ϕ to its separate form $\phi_1 \wedge \phi_2$.
2. Let S be the set of variables shared between ϕ_1 and ϕ_2. Guess an equivalence relation $\sim$ on S.
3. Run Sat_1 on $\phi_1 \cup Ar_\sim$.
4. Run Sat_2 on $\phi_2 \cup Ar_\sim$.

The Nelson-Oppen Method

If there exists an equivalence relation $\sim$ such that both Sat_1 and Sat_2 succeed, then ϕ is T-satisfiable.

If no such equivalence relation exists, then ϕ is T-unsatisfiable.

The generalization to more than two theories is straightforward.

Example

Consider the following $\Sigma_{\mathcal{E}} \cup \Sigma_{\mathcal{Z}}$ formula:

$$\phi = 1 \leq x \land x \leq 2 \land f(x) \neq f(1) \land f(x) \neq f(2).$$

Example

Consider the following $\Sigma_{\mathcal{E}} \cup \Sigma_{\mathcal{Z}}$ formula:

$\phi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$.

We first convert ϕ to a separate form:

$$\begin{aligned} \phi_{\mathcal{E}} &= f(x) \neq f(y) \wedge f(x) \neq f(z) \\ \phi_{\mathcal{Z}} &= 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2 \end{aligned}$$

The shared variables are $\{x, y, z\}$. There are 5 possible arrangements based on equivalence classes of x, y, and z.

Example

$$\phi_{\mathcal{E}} = f(x) \neq f(y) \wedge f(x) \neq f(z)$$
$$\phi_{\mathcal{Z}} = 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$$

1. $\{x = y, x = z, y = z\}$
2. $\{x = y, x \neq z, y \neq z\}$
3. $\{x \neq y, x = z, y \neq z\}$
4. $\{x \neq y, x \neq z, y = z\}$
5. $\{x \neq y, x \neq z, y \neq z\}$

Example

$$\phi_{\mathcal{E}} = f(x) \neq f(y) \wedge f(x) \neq f(z)$$
$$\phi_{\mathcal{Z}} = 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$$

1. $\{x = y, x = z, y = z\}$: inconsistent with $\phi_{\mathcal{E}}$.
2. $\{x = y, x \neq z, y \neq z\}$
3. $\{x \neq y, x = z, y \neq z\}$
4. $\{x \neq y, x \neq z, y = z\}$
5. $\{x \neq y, x \neq z, y \neq z\}$

Example

$$\phi_{\mathcal{E}} = f(x) \neq f(y) \wedge f(x) \neq f(z)$$
$$\phi_{\mathcal{Z}} = 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$$

1. $\{x = y, x = z, y = z\}$: inconsistent with $\phi_{\mathcal{E}}$.
2. $\{x = y, x \neq z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
3. $\{x \neq y, x = z, y \neq z\}$
4. $\{x \neq y, x \neq z, y = z\}$
5. $\{x \neq y, x \neq z, y \neq z\}$

Example

$$\phi_{\mathcal{E}} = f(x) \neq f(y) \wedge f(x) \neq f(z)$$
$$\phi_{\mathcal{Z}} = 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$$

1. $\{x = y, x = z, y = z\}$: inconsistent with $\phi_{\mathcal{E}}$.
2. $\{x = y, x \neq z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
3. $\{x \neq y, x = z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
4. $\{x \neq y, x \neq z, y = z\}$
5. $\{x \neq y, x \neq z, y \neq z\}$

Example

$$\phi_{\mathcal{E}} = f(x) \neq f(y) \land f(x) \neq f(z)$$
$$\phi_{\mathcal{Z}} = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2$$

1. $\{x = y, x = z, y = z\}$: inconsistent with $\phi_{\mathcal{E}}$.
2. $\{x = y, x \neq z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
3. $\{x \neq y, x = z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
4. $\{x \neq y, x \neq z, y = z\}$: inconsistent with $\phi_{\mathcal{Z}}$.
5. $\{x \neq y, x \neq z, y \neq z\}$

Example

$$\phi_{\mathcal{E}} = f(x) \neq f(y) \wedge f(x) \neq f(z)$$
$$\phi_{\mathcal{Z}} = 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$$

1. $\{x = y, x = z, y = z\}$: inconsistent with $\phi_{\mathcal{E}}$.
2. $\{x = y, x \neq z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
3. $\{x \neq y, x = z, y \neq z\}$: inconsistent with $\phi_{\mathcal{E}}$.
4. $\{x \neq y, x \neq z, y = z\}$: inconsistent with $\phi_{\mathcal{Z}}$.
5. $\{x \neq y, x \neq z, y \neq z\}$: inconsistent with $\phi_{\mathcal{Z}}$.

Correctness of Nelson-Oppen

We define an *interpretation* of a signature Σ to be a model of Σ together with a variable assignment. If A is an interpretation, we write $A \models \phi$ to mean that ϕ is satisfied by the model and variable assignment contained in A.

Two interpretations A and B are *isomorphic* if there exists an isomorphism h of the model of A into the model of B and $h(x^A) = x^B$ for each variable x (where x^A signifies the value assigned to x by the variable assignment of A).

We furthermore define $A^{\Sigma,V}$ to be the restriction of A to the symbols in Σ and the variables in V.

Correctness of Nelson-Oppen

Theorem

Let Σ_1 and Σ_2 be signatures, and for $i = 1, 2$, let ϕ_i be a set of Σ_i-formulas, and V_i the set of variables appearing in ϕ_i. Then $\phi_1 \cup \phi_2$ is satisfiable iff there exists a Σ_1-interpretation A satisfying ϕ_1 and a Σ_2-interpretation B satisfying ϕ_2 such that:

$A^{\Sigma_1 \cap \Sigma_2, V_1 \cap V_2}$ is isomorphic to $B^{\Sigma_1 \cap \Sigma_2, V_1 \cap V_2}$.

Correctness of Nelson-Oppen

Proof

Let $\Sigma = \Sigma_1 \cap \Sigma_2$ and $V = V_1 \cap V_2$.
Suppose $\phi_1 \cup \phi_2$ is satisfiable. Let M be an interpretation satisfying $\phi_1 \cup \phi_2$. If we let $A = M^{\Sigma_1, V_1}$ and $B = M^{\Sigma_2, V_2}$, then clearly:

- $A \models \phi_1$
- $B \models \phi_2$
- $A^{\Sigma, V}$ is isomorphic to $B^{\Sigma, V}$

On the other hand, suppose that we have A and B satisfying the three conditions listed above. Let h be an isomorphism from $A^{\Sigma, V}$ to $B^{\Sigma, V}$.

Correctness of Nelson-Oppen

We define an interpretation M as follows:

- $\mathit{dom}(M) = \mathit{dom}(A)$
- For each variable or constant u, $u^M =$
$$\begin{cases} u^A & \text{if } u \in (\Sigma_1^C \cup V_1) \\ h^{-1}(u^B) & \text{otherwise} \end{cases}$$
- For function symbols of arity n,
$$f^M(a_1, \ldots, a_n) = \begin{cases} f^A(a_1, \ldots, a_n) & \text{if } f \in \Sigma_1^F \\ h^{-1}(f^B(h(a_1), \ldots, h(a_n))) & \text{otherwise} \end{cases}$$
- For predicate symbols of arity n,

  $(a_1, \ldots, a_n) \in P^M$ iff $(a_1, \ldots, a_n) \in P^A$ if $P \in \Sigma_1^P$
  $(a_1, \ldots, a_n) \in P^M$ iff $(h(a_1), \ldots, h(a_n)) \in P^B$
  otherwise

Correctness of Nelson-Oppen

By construction, M^{Σ_1,V_1} is isomorphic to A. In addition, it is easy to verify that h is an isomorphism of M^{Σ_2,V_2} to B.

It follows by the homomorphism theorem (a standard theorem of first-order logic) that M satisfies $\phi_1 \cup \phi_2$.

$\square$

Correctness of Nelson-Oppen

Theorem

Let Σ_1 and Σ_2 be signatures, with $\Sigma_1 \cap \Sigma_2 = \emptyset$, and for $i = 1, 2$, let ϕ_i be a set of Σ_i-formulas, and V_i the set of variables appearing in ϕ_i. As before, let $V = V_1 \cap V_2$. Then $\phi_1 \cup \phi_2$ is satisfiable iff there exists an interpretation A satisfying ϕ_1 and an interpretation B satisfying ϕ_2 such that:

1. $|A| = |B|$, and
2. $x^A = y^A$ iff $x^B = y^B$ for every pair of variables $x, y \in V$.

Correctness of Nelson-Oppen

Proof

Clearly, if $\phi_1 \cup \phi_2$ is satisfiable in some interpretation M, then the only if direction holds by letting $A = M$ and $B = M$.

Consider the converse. Let $h : V^A \rightarrow V^B$ be defined as $h(x^A) = x^B$. This definition is well-formed by property 2 above.

In fact, h is bijective. To show that h is injective, let $h(a_1) = h(a_2)$. Then there exist variables $x, y \in V$ such that $a_1 = x^A$, $a_2 = y^A$, and $x^B = y^B$. By property 2, $x^A = y^A$, and therefore $a_1 = a_2$.

Correctness of Nelson-Oppen

To show that h is surjective, let $b \in V^B$. Then there exists a variable $x \in V^B$ such that $x^B = b$. But then $h(x^A) = b$.

Since h is bijective, it follows that $|V^A| = |V^B|$, and since $|A| = |B|$, we also have that $|A - V^A| = |B - V^B|$. We can therefore extend h to a bijective function h' from A to B.

By construction, h' is an isomorphism of A^V to B^V. Thus, by the previous theorem, we can obtain an interpretation satisfying $\phi_1 \cup \phi_2$.

$\square$

Correctness of Nelson-Oppen

We can now prove the correctness of the non-deterministic Nelson-Oppen method:

Theorem

Let T_i be a stably-infinite Σ_i-theory, for $i = 1, 2$, and suppose that $\Sigma_1 \cap \Sigma_2 = \emptyset$. Also, let ϕ_i be a set of Σ_i literals, $i = 1, 2$, and let S be the set of variables appearing in both ϕ_1 and ϕ_2. Then $\phi_1 \cup \phi_2$ is $T_1 \cup T_2$-satisfiable iff there exists an equivalence relation $\sim$ on S such that $\phi_i \cup Ar_\sim$ is T_i-satisfiable, $i = 1, 2$.

Correctness of Nelson-Oppen

Proof

($\Rightarrow$) Suppose M is an interpretation satisfying $\phi_1 \cup \phi_2$. Define $x \sim y$ iff $x, y \in S$ and $x^M = y^M$. By construction, M is a T_i-interpretation satisfying $\phi_i \cup Ar_\sim$, $i = 1, 2$.

($\Leftarrow$) Suppose there exists $\sim$ such that $\phi_i \cup Ar_\sim$ is T_i-satisfiable, $i = 1, 2$. Since each T_i is stably-infinite, there is are infinite interpretations A and B such that A satisfies $\phi_1 \cup Ar_\sim$ and B satisfies $\phi_2 \cup Ar_\sim$.

By another standard theorem (LST), we can take the least upper bound of $|A|$ and $|B|$ and obtain interpretations of that cardinality.

Then we have $|A| = |B|$ and $x^A = y^A$ iff $x^B = y^B$ for every variable $x, y \in S$. By the previous theorem, there exists of a $(\Sigma_1 \cup \Sigma_2)$-interpretation satisfying $\phi_1 \cup \phi_2$. $\square$

Nelson-Oppen Example

Consider the following example in a combination of $T_{\mathcal{E}}$, $T_{\mathcal{Z}}$, and $T_{\mathcal{A}}$:

$$\neg p(y) \wedge s = \textit{write}\,(t, i, 0) \wedge x - y - z = 0 \wedge$$
$$z + \textit{read}\,(s, i) = f(x - y) \wedge p(x - f(f(z))).$$

After purification, we have the following:

$\varphi_{\mathcal{E}}$	$\varphi_{\mathcal{Z}}$	$\varphi_{\mathcal{A}}$
$\neg p(y)$	$l - z = j$	$s = \textit{write}\,(t, i, j)$
$m = f(l)$	$j = 0$	$k = \textit{read}\,(s, i)$
$p(v)$	$l = x - y$	
$n = f(f(z))$	$m = z + k$	
	$v = x - n$	

Example

$\varphi_{\mathcal{E}}$	$\varphi_{\mathcal{Z}}$	$\varphi_{\mathcal{A}}$
$\neg p(y)$	$l - z = j$	$s = \textit{write}\,(t, i, j)$
$m = f(l)$	$j = 0$	$k = \textit{read}\,(s, i)$
$p(v)$	$l = x - y$	
$n = f(f(z))$	$m = z + k$	
	$v = x - n$	

There are 12 constants in this example:

- Shared: l, z, j, y, m, k, v, n
- Unshared: x, s, t, i

There are 21147 arrangements of $\{l, z, j, y, m, k, v, n\}$. Clearly, a practical implementation cannot consider all of these separately.

Implementing Nelson-Oppen

In order to obtain a more practical implementation of Nelson-Oppen, we will consider the following refinements:

- Eliminating the purification step
- Incremental processing with theory-specific rewrites
- Strategies for searching through arrangements

Implementing Nelson-Oppen

As most implementers of SMT systems will tell you, the purification step is not really necessary in practice.

In fact, a simple variation of Nelson-Oppen can be obtained that does not require purification [BDS02b].

Given a set of mixed (impure) literals Γ, define a *shared term* to be any term in Γ which occurs i-alien for some i in some literal or sub-term in Γ.

Note that these are exactly the terms that would have been replaced with new constants in the purification step.

Assume also that each Sat_i is modified so that it treats alien terms as constants.

Implementing Nelson-Oppen

The following is a variation of Nelson-Oppen which does not use purification.

1. *Partition* Γ into sets φ_i, where each literal in φ_i is an i-literal (literals containing only equalities between variables can go anywhere).
2. Let S be the set of *shared terms* in Γ.
3. For each arrangement Δ of S,

   Check $\mathit{Sat}_i(\varphi_i \wedge \Delta)$ for each i.

Example

Consider again the example from before:

$$\neg p(y) \land s = \textit{write}\,(t, i, 0) \land x - y - z = 0 \land$$
$$z + \textit{read}\,(s, i) = f(x - y) \land p(x - f(f(z))).$$

After partitioning, we have the following:

$\varphi_{\mathcal{E}}$	$\varphi_{\mathcal{Z}}$	$\varphi_{\mathcal{A}}$
$\neg p(y)$	$x - y - z = 0$	$s = \textit{write}\,(t, i, 0)$
$p(x - f(f(z)))$	$z + \textit{read}\,(s, i) = f(x - y)$	

The shared terms are:

$$\textit{read}\,(s, i), x - y, f(x - y), 0, y, z, f(f(z)), x - f(f(z)).$$

Unfortunately, there are still too many arrangements.

Implementing Nelson-Oppen

The next refinement is to process Γ incrementally, allowing theory-specific rewrites that can potentially reduce the number of shared terms.

Examples of theory-specific rewrites include canonization or partial canonization and simplification based on previously seen literals.

1. For each $\varphi \in \Gamma$
   (a) (Optionally) apply theory-specific rewrites to φ to get φ'
   (b) Identify the shared terms in φ' and add these to S
   (c) Where φ' is an i-literal, add φ' to φ_i
2. For each arrangement Δ of S,
   Check $\mathit{Sat}_i(\varphi_i \wedge \Delta)$ for each i.

Example

Let's see what happens if we process our example incrementally:

$$\neg p(y) \wedge s = \textit{write}\,(t, i, 0) \wedge x - y - z = 0 \wedge$$
$$z + \textit{read}\,(s, i) = f(x - y) \wedge p(x - f(f(z))).$$

$\varphi_{\mathcal{E}}$	$\varphi_{\mathcal{Z}}$	$\varphi_{\mathcal{A}}$
$\neg p(y)$	$x = y + z$	$s = \textit{write}\,(t, i, 0)$
$p(y)$	$z = f(z)$	

The shared terms are: $0, y, z, f(z)$. There are only 52 arrangements now.

More importantly, $\varphi_{\mathcal{E}}$ is now inconsistent in the theory $T_{\mathcal{E}}$, making it unnecessary to examine any arrangements.

Implementing Nelson-Oppen

We have seen two ways to avoid searching through too many arrangements:

1. Reduce the number of shared terms
2. Detect an inconsistency early

As a further example of (2), we can build arrangements incrementally, backtracking if any theory detects an inconsistency.

For convex theories, this strategy is very efficient.

For non-convex theories, we may have to explore the entire search space of arrangements.

Implementing Nelson-Oppen

The strategies we have looked at so far do not assume any help from the theory decision procedures (beyond the ability to determine inconsistency).

If the theory decision procedures are able to give additional information, it may significantly help to prune the arrangement search.

The next refinement of our algorithm captures this.

Implementing Nelson-Oppen

1. For each $\varphi \in \Gamma$
   (a) (Optional) Apply theory-specific rewrites to φ to get φ'
   (b) Identify the shared terms in φ' and add these to S
   (c) Where φ' is an i-literal, add φ' to φ_i
   (d) (Optional) If $\varphi_i \models s_1 = s_2$ or $\varphi_i \models s_1 \neq s_2$, $s_1, s_2 \in S$, add this fact to Γ.
2. Incrementally search through arrangements Δ of S that are consistent with $\bigwedge \varphi_i$. For each arrangement, check $\mathit{Sat}_i(\varphi_i \wedge \Delta)$ for each i.

Implementing Nelson-Oppen

Finally, for maximum efficiency and flexibility, we can push the entire burden of arrangement finding onto the theory decision procedures.

Suppose $\Phi = \bigwedge \varphi_i$ is the partition of literals from Γ that have been processed so far and that S is the set of shared terms.

The *equivalence relation* R *on* S *induced by* Φ is defined as follows: for $x, y \in S$, xRy iff $x = y \in \varphi_i$ for some i.

The *arrangement* $\Delta(\Phi)$ *of* S *induced by* Φ is the arrangement induced by R.

Implementing Nelson-Oppen

For each $\varphi \in \Gamma$:

1. (Optional) Apply theory-specific rewrites to φ to get φ'
2. Identify the shared terms in φ' and add these to S
3. Where φ' is an i-literal, add φ' to φ_i
4. If $\varphi_i \wedge \Delta(\Phi)$ is not satisfiable, compute some formula ψ such that $\varphi_i \models \psi$ and $\psi \wedge \Delta(\Phi)$ is inconsistent. Add ψ to Γ.

Implementing Nelson-Oppen

For each $\varphi \in \Gamma$:

1. (Optional) Apply theory-specific rewrites to φ to get φ'
2. Identify the shared terms in φ' and add these to S
3. Where φ' is an i-literal, add φ' to φ_i
4. If $\varphi_i \wedge \Delta(\Phi)$ is not satisfiable, compute some formula ψ such that $\varphi_i \models \psi$ and $\psi \wedge \Delta(\Phi)$ is inconsistent. Add ψ to Γ.

Some notes:

- In general, ψ does not have to be a literal. In this case, ψ must be processed by the SAT solver (more on this next).
- Theories can be lazy until Γ is empty.
- Termination becomes the responsibility of the theory decision procedures.

Implementing Nelson-Oppen

For each $\varphi \in \Gamma$:

1. (Optional) Apply theory-specific rewrites to φ to get φ'
2. Identify the shared terms in φ' and add these to S
3. Where φ' is an i-literal, add φ' to φ_i
4. If $\varphi_i \wedge \Delta(\Phi)$ is not satisfiable, compute some formula ψ such that $\varphi_i \models \psi$ and $\psi \wedge \Delta(\Phi)$ is inconsistent. Add ψ to Γ.

More notes:

- It is not hard to fit a Shostak-style decision procedure into this framework.
- This is essentially the algorithm used in the CVC tools [Bar03].

Roadmap

Satisfiability Modulo Theories

- First-Order Logic
- Specific Theories
- Theory Solvers
- Combining Theory Solvers
- Combining with SAT
- Modeling in SMT

Combining with SAT

Theory solvers check the satisfiability of conjunctions of literals.

What about more general Boolean combinations of literals?

What is needed is a combination of SAT reasoning and theory reasoning.

The so-called *eager* approach to SMT tries to find ways of encoding everything into SAT. There are a variety of techniques and for some theories, this works quite well.

Here, I will focus on the *lazy* combination of SAT and Theory reasoning. The lazy approach is the basis for most modern SMT solvers [**?** BDS02a].

Abstract DPLL Modulo Theories

The *Abstract DPLL Modulo Theories* framework extends the Abstract DPLL framework, providing an abstract and formal setting for reasoning about the combination of SAT and theory reasoning [NOT06].

Assume we have a theory T with signature Σ and a solver Sat_T that can check T-satisfiability of conjunctions of Σ-literals.

Suppose we want to check the satisfiability of an *arbitray* (quantifier-free) Σ-formula ϕ.

We start by converting ϕ to CNF.

Now, let's consider using the *Abstract DPLL* rules (allowing *any first-order* literal where before we had propositional literals).

Abstract DPLL Rules

UnitProp :

$$M \parallel F, C \vee l \quad \Longrightarrow \quad M\ l \parallel F, C \vee l \quad \textbf{if} \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$$

PureLiteral :

$$M \parallel F \quad \Longrightarrow \quad M\ l \parallel F \quad \textbf{if} \begin{cases} l \text{ occurs in some clause of } F \\ -l \text{ occurs in no clause of } F \\ l \text{ is undefined in } M \end{cases}$$

Decide :

$$M \parallel F \quad \Longrightarrow \quad M\ l^{\mathsf{d}} \parallel F \quad \textbf{if} \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases}$$

Fail :

$$M \parallel F, C \quad \Longrightarrow \quad \mathit{fail} \quad \textbf{if} \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases}$$

Abstract DPLL Rules

Backjump :

$$M\ l^{\mathsf{d}}\ N \parallel F, C \quad\Longrightarrow\quad M\ l' \parallel F, C \quad \textbf{if} \begin{cases} M\ l^{\mathsf{d}}\ N \models \neg C, \text{ and there is} \\ \text{some clause } C' \vee l' \text{ such that:} \\ \quad F, C \models C' \vee l' \text{ and } M \models \neg C', \\ \quad l' \text{ is undefined in } M, \text{ and} \\ \quad l' \text{ or } \neg l' \text{ occurs in } F \text{ or in } M\ l^{\mathsf{d}}\ N \end{cases}$$

Learn :

$$M \parallel F \quad\Longrightarrow\quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } F \\ F \models C \end{cases}$$

Forget :

$$M \parallel F, C \quad\Longrightarrow\quad M \parallel F \quad \textbf{if} \left\{ F \models C \right.$$

Restart :

$$M \parallel F \quad\Longrightarrow\quad \emptyset \parallel F$$

Abstract DPLL Modulo Theories

The first change is to the definition of a *final state*. A final state is now:

- the special fail state: $fail$, or
- $M \parallel F$, where $M \models F$, *and* $\mathsf{Sat}_T(M)$ reports satisfiable.

Abstract DPLL Modulo Theories

The first change is to the definition of a *final state*. A final state is now:

- the special fail state: $fail$, or
- $M \parallel F$, where $M \models F$, *and* $\mathsf{Sat}_T(M)$ reports satisfiable.

What happens if we reach a state in which: $M \parallel F$, $M \models F$, and $\mathsf{Sat}_T(M)$ reports *unsatisfiable*? (call this a *pseudo-final* state)

Abstract DPLL Modulo Theories

The first change is to the definition of a *final state*. A final state is now:

- the special fail state: fail, or
- $M \parallel F$, where $M \models F$, *and* $\text{Sat}_T(M)$ reports satisfiable.

What happens if we reach a state in which: $M \parallel F$, $M \models F$, and $\text{Sat}_T(M)$ reports *unsatisfiable*? (call this a *pseudo-final* state)

We need to backtrack. The SAT solver will take care of this automatically if we can add a clause C such that $M \models \neg C$.

Abstract DPLL Modulo Theories

The first change is to the definition of a *final state*. A final state is now:

- the special fail state: $fail$, or
- $M \parallel F$, where $M \models F$, *and* $\textsf{Sat}_T(M)$ reports satisfiable.

What happens if we reach a state in which: $M \parallel F$, $M \models F$, and $\textsf{Sat}_T(M)$ reports *unsatisfiable*? (call this a *pseudo-final* state)

We need to backtrack. The SAT solver will take care of this automatically if we can add a clause C such that $M \models \neg C$.

What clause should we add?

Abstract DPLL Modulo Theories

The first change is to the definition of a *final state*. A final state is now:

- the special fail state: $fail$, or
- $M \parallel F$, where $M \models F$, *and* Sat$_T(M)$ reports satisfiable.

What happens if we reach a state in which: $M \parallel F$, $M \models F$, and Sat$_T(M)$ reports *unsatisfiable*? (call this a *pseudo-final* state)

We need to backtrack. The SAT solver will take care of this automatically if we can add a clause C such that $M \models \neg C$.

What clause should we add? How about $\neg M$?

Abstract DPLL Modulo Theories

The justification for adding $\neg M$ is that $T \models \neg M$.

We can generalize this to any clause C such that $T \models C$. The following modified Learn rule allows this (we also modify the Forget rule in an analagous way):

Theory Learn :

$$M \parallel F \quad\Longrightarrow\quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } F \\ F \models_T C \end{cases}$$

Theory Forget :

$$M \parallel F, C \quad\Longrightarrow\quad M \parallel F \quad \textbf{if} \left\{ F \models_T C \right.$$

Abstract DPLL Modulo Theories

The resulting set of rules is almost enough to correctly implement an SMT solver. We need one more change.

Abstract DPLL Modulo Theories

The resulting set of rules is almost enough to correctly implement an SMT solver. We need one more change.

A somewhat surprising observation is that the *pure literal* rule has to be abandoned. *Why?*

Abstract DPLL Modulo Theories

The resulting set of rules is almost enough to correctly implement an SMT solver. We need one more change.

A somewhat surprising observation is that the *pure literal* rule has to be abandoned. *Why?*

Propositional literals are independent of each other, but first order literals may not be.

Abstract DPLL Modulo Theories

The resulting set of rules is almost enough to correctly implement an SMT solver. We need one more change.

A somewhat surprising observation is that the *pure literal* rule has to be abandoned. *Why?*

Propositional literals are independent of each other, but first order literals may not be.

The remaining rules form a sound and complete procedure for SMT.

From SAT to SMT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\emptyset \;\; \| \quad 1,\; \overline{2} \vee 3,\; \overline{4} \vee \overline{3}$$

From SAT to SMT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$\emptyset \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} \quad \Longrightarrow \quad$ (UnitProp)

$1 \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3}$

From SAT to SMT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll} \emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \text{(UnitProp)} \\ 1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \text{(Decide)} \\ 1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \end{array}$$

From SAT to SMT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} &
\end{array}$$

From SAT to SMT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll} \emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\textsf{UnitProp}) \\ 1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\textsf{Decide}) \\ 1\ \overline{2}^{\textsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\textsf{Decide}) \\ 1\ \overline{2}^{\textsf{d}}\ \overline{4}^{\textsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\textsf{Theory Learn}) \\ 1\ \overline{2}^{\textsf{d}}\ \overline{4}^{\textsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 \vee 4 & \end{array}$$

From SAT to SMT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \text{(UnitProp)} \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \text{(Decide)} \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \text{(Decide)} \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \text{(Theory Learn)} \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 \vee 4 & \Longrightarrow \text{(Backjump)} \\
1\ \overline{2}^{\mathsf{d}}\ 4 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 \vee 4 &
\end{array}$$

From SAT to SMT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$
\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 \vee 4 & \Longrightarrow \quad (\text{Backjump}) \\
1\ \overline{2}^{\mathsf{d}}\ 4 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 \vee 4 & \Longrightarrow \quad (\text{UnitProp}) \\
1\ \overline{2}^{\mathsf{d}}\ 4\ \overline{3} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 \vee 4 &
\end{array}
$$

From SAT to SMT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Decide}) \\
1\ \overline{2}^{\text{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Decide}) \\
1\ \overline{2}^{\text{d}}\ \overline{4}^{\text{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Theory Learn}) \\
1\ \overline{2}^{\text{d}}\ \overline{4}^{\text{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 \vee 4 & \Longrightarrow \quad (\text{Backjump}) \\
1\ \overline{2}^{\text{d}}\ 4 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 \vee 4 & \Longrightarrow \quad (\text{UnitProp}) \\
1\ \overline{2}^{\text{d}}\ 4\ \overline{3} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 \vee 4 & \Longrightarrow \quad (\text{Theory Learn}) \\
1\ \overline{2}^{\text{d}}\ 4\ \overline{3} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 \vee 4,\ \overline{1} \vee 2 \vee \overline{4} \vee 3 &
\end{array}$$

From SAT to SMT

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad \underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3} \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}} \vee \underbrace{g(a)\neq d}_{\overline{3}}$$

$$\begin{array}{rcl}
\emptyset & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} \qquad\qquad \Longrightarrow \quad (\textsf{UnitProp}) \\
1 & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} \qquad\qquad \Longrightarrow \quad (\textsf{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} \qquad\qquad \Longrightarrow \quad (\textsf{Decide}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} \qquad\qquad \Longrightarrow \quad (\textsf{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4 \qquad\qquad \Longrightarrow \quad (\textsf{Backjump}) \\
1\ \overline{2}^{\mathsf{d}}\ 4 & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4 \qquad\qquad \Longrightarrow \quad (\textsf{UnitProp}) \\
1\ \overline{2}^{\mathsf{d}}\ 4\ \overline{3} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4 \qquad\qquad \Longrightarrow \quad (\textsf{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}}\ 4\ \overline{3} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4,\ \overline{1}\vee 2\vee\overline{4}\vee 3 \qquad\qquad \Longrightarrow \quad (\textsf{Backjump}) \\
1\ 2 & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4,\ \overline{1}\vee 2\vee\overline{4}\vee 3
\end{array}$$

From SAT to SMT

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad \underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3} \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}} \vee \underbrace{g(a)\neq d}_{\overline{3}}$$

$\emptyset$	$\Vert$	$1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3}$	$\Longrightarrow$	(UnitProp)
1	$\Vert$	$1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3}$	$\Longrightarrow$	(Decide)
$1\ \overline{2}^{\mathsf{d}}$	$\Vert$	$1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3}$	$\Longrightarrow$	(Decide)
$1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}}$	$\Vert$	$1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3}$	$\Longrightarrow$	(Theory Learn)
$1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}}$	$\Vert$	$1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4$	$\Longrightarrow$	(Backjump)
$1\ \overline{2}^{\mathsf{d}}\ 4$	$\Vert$	$1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4$	$\Longrightarrow$	(UnitProp)
$1\ \overline{2}^{\mathsf{d}}\ 4\ \overline{3}$	$\Vert$	$1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4$	$\Longrightarrow$	(Theory Learn)
$1\ \overline{2}^{\mathsf{d}}\ 4\ \overline{3}$	$\Vert$	$1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4,\ \overline{1}\vee 2\vee\overline{4}\vee 3$	$\Longrightarrow$	(Backjump)
$1\ 2$	$\Vert$	$1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4,\ \overline{1}\vee 2\vee\overline{4}\vee 3$	$\Longrightarrow$	(UnitProp)
$1\ 2\ 3\ \overline{4}$	$\Vert$	$1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4,\ \overline{1}\vee 2\vee\overline{4}\vee 3$		

From SAT to SMT

$$\underbrace{g(a)=c}_{1} \quad \land \quad \underbrace{f(g(a))\neq f(c)}_{\overline{2}} \lor \underbrace{g(a)=d}_{3} \quad \land \quad \underbrace{c\neq d}_{\overline{4}} \lor \underbrace{g(a)\neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3} & \Longrightarrow\ (\text{UnitProp}) \\
1 & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3} & \Longrightarrow\ (\text{Decide}) \\
1\ \overline{2}^{\text{d}} & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3} & \Longrightarrow\ (\text{Decide}) \\
1\ \overline{2}^{\text{d}}\ \overline{4}^{\text{d}} & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3} & \Longrightarrow\ (\text{Theory Learn}) \\
1\ \overline{2}^{\text{d}}\ \overline{4}^{\text{d}} & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3},\ \overline{1}\lor 2\lor 4 & \Longrightarrow\ (\text{Backjump}) \\
1\ \overline{2}^{\text{d}}\ 4 & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3},\ \overline{1}\lor 2\lor 4 & \Longrightarrow\ (\text{UnitProp}) \\
1\ \overline{2}^{\text{d}}\ 4\ \overline{3} & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3},\ \overline{1}\lor 2\lor 4 & \Longrightarrow\ (\text{Theory Learn}) \\
1\ \overline{2}^{\text{d}}\ 4\ \overline{3} & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3},\ \overline{1}\lor 2\lor 4,\ \overline{1}\lor 2\lor\overline{4}\lor 3 & \Longrightarrow\ (\text{Backjump}) \\
1\ 2 & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3},\ \overline{1}\lor 2\lor 4,\ \overline{1}\lor 2\lor\overline{4}\lor 3 & \Longrightarrow\ (\text{UnitProp}) \\
1\ 2\ 3\ \overline{4} & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3},\ \overline{1}\lor 2\lor 4,\ \overline{1}\lor 2\lor\overline{4}\lor 3 & \Longrightarrow\ (\text{Theory Learn}) \\
1\ 2\ 3\ \overline{4} & \parallel & 1,\ \overline{2}\lor 3,\ \overline{4}\lor\overline{3},\ \overline{1}\lor 2\lor 4,\ \overline{1}\lor 2\lor\overline{4}\lor 3,\ \overline{1}\lor\overline{2}\lor\overline{3}\lor 4 &
\end{array}$$

From SAT to SMT

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad \underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3} \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}} \vee \underbrace{g(a)\neq d}_{\overline{3}}$$

$$\begin{array}{rcllc l}
\emptyset & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} & & \Longrightarrow & (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} & & \Longrightarrow & (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} & & \Longrightarrow & (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} & & \Longrightarrow & (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4 & & \Longrightarrow & (\text{Backjump}) \\
1\ \overline{2}^{\mathsf{d}}\ 4 & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4 & & \Longrightarrow & (\text{UnitProp}) \\
1\ \overline{2}^{\mathsf{d}}\ 4\ \overline{3} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4 & & \Longrightarrow & (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}}\ 4\ \overline{3} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4,\ \overline{1}\vee 2\vee\overline{4}\vee 3 & & \Longrightarrow & (\text{Backjump}) \\
1\ 2 & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4,\ \overline{1}\vee 2\vee\overline{4}\vee 3 & & \Longrightarrow & (\text{UnitProp}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4,\ \overline{1}\vee 2\vee\overline{4}\vee 3 & & \Longrightarrow & (\text{Theory Learn}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2\vee 4,\ \overline{1}\vee 2\vee\overline{4}\vee 3,\ \overline{1}\vee\overline{2}\vee\overline{3}\vee 4 & & \Longrightarrow & (\text{Fail}) \\
 & fail & & & &
\end{array}$$

Improving Abstract DPLL Modulo Theories

We will mention three ways to improve the algorithm.

- Minimizing learned clauses
- Eager conflict detection
- Theory propagation

Minimizing Learned Clauses

The main difficulty with the approach as it stands is that learned clauses can be highly redundant.

Suppose that F contains $n+2$ propositional variables.

When a pseudo-final state is reached, M will determine a value for all $n+2$ variables.

But what if only 2 of these assignments are already T-unsatisfiable?

If we always learn $\neg M$ in a pseudo-final state, in the worst case, 2^n clauses will be need to be learned when a single clause containing the two offending literals would have sufficed.

Minimizing Learned Clauses

To avoid this kind of redundancy, we can be smarter about the clauses that are learned with Theory Learn.

In particular, when $\mathsf{Sat}_T(M)$ is called, we should make an effort to find the *smallest* possible subset of M which is inconsistent.

We can then learn a clause derived from *only these* literals.

One way to implement this is to start removing literals one at a time from M and repeatedly call Sat_T until a minimal inconsistent set is found.

However, this is typically too slow to be practical.

Minimizing Learned Clauses

A better, but more difficult way to implement this is to instrument Sat_T to keep track of which facts are used to derive an inconsistency.

We can use a data structure similar to the implication graph discussed earlier.

Alternatively, if Sat_T happens to produce proofs, the proof of unsatisfiability of M can be traversed to obtain this information.

This is the approach used in the CVC tools.

From SAT to SMT — Minimized Clauses

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\emptyset \;\; \| \quad 1,\; \overline{2} \vee 3,\; \overline{4} \vee \overline{3}$$

From SAT to SMT — Minimized Clauses

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll} \emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{UnitProp}) \\ 1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \end{array}$$

From SAT to SMT — Minimized Clauses

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Decide}) \\
1\ \overline{2}^{\text{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} &
\end{array}$$

From SAT to SMT — Minimized Clauses

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \text{(UnitProp)} \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \text{(Decide)} \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \text{(Decide)} \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} &
\end{array}$$

From SAT to SMT — Minimized Clauses

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcllll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & &
\end{array}$$

From SAT to SMT — Minimized Clauses

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad \underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3} \quad \wedge \quad \underbrace{c\neq d}_{\overline{4}} \vee \underbrace{g(a)\neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} & \Longrightarrow \ (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} & \Longrightarrow \ (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} & \Longrightarrow \ (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3} & \Longrightarrow \ (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2 & \Longrightarrow \ (\text{Backjump}) \\
1\ 2 & \| & 1,\ \overline{2}\vee 3,\ \overline{4}\vee\overline{3},\ \overline{1}\vee 2 &
\end{array}$$

From SAT to SMT — Minimized Clauses

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \ (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \ (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \ (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \ (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & \Longrightarrow \ (\text{Backjump}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & \Longrightarrow \ (\text{UnitProp}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 &
\end{array}$$

From SAT to SMT — Minimized Clauses

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcllcl}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & & \Longrightarrow & (\text{Backjump}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & & \Longrightarrow & (\text{UnitProp}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & & \Longrightarrow & (\text{Theory Learn}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 & & &
\end{array}$$

From SAT to SMT — Minimized Clauses

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$
\begin{array}{rcllcl}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}}\ \overline{4}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & & \Longrightarrow & (\text{Backjump}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & & \Longrightarrow & (\text{UnitProp}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & & \Longrightarrow & (\text{Theory Learn}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 & & \Longrightarrow & (\text{Fail}) \\
 & fail & & & &
\end{array}
$$

Eager Conflict Detection

Currently, we have indicated that we will check M for T-satisfiability only when a pseudo-final state is reached.

In contrast, a more eager policy would be to check M for T-satisfiability every time M changes.

Experimental results show that this approach is significantly better.

It requires *Sat*$_T$ be *online*: able quickly to determine the consistency of incrementally more literals or to backtrack to a previous state.

It also requires that the SAT solver be instrumented to call *Sat*$_T$ every time a variable is assigned a value.

From SAT to SMT — Eager Conflict Detection

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\emptyset \;\|\; \quad 1,\; \overline{2} \vee 3,\; \overline{4} \vee \overline{3}$$

From SAT to SMT — Eager Conflict Detection

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{lll} \emptyset \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{UnitProp}) \\ 1 \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \end{array}$$

From SAT to SMT — Eager Conflict Detection

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll} \emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\textsf{UnitProp}) \\ 1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\textsf{Decide}) \\ 1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \end{array}$$

From SAT to SMT — Eager Conflict Detection

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcllcl}
\emptyset & \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{UnitProp}) \\
1 & \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \Longrightarrow & (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}} & \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & & &
\end{array}$$

From SAT to SMT — Eager Conflict Detection

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \ (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \ (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \ (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & \Longrightarrow \ (\text{Backjump}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 &
\end{array}$$

From SAT to SMT — Eager Conflict Detection

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$\emptyset$	$\parallel$	$1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3}$	$\Longrightarrow$	(UnitProp)
1	$\parallel$	$1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3}$	$\Longrightarrow$	(Decide)
$1\ \overline{2}^{\mathsf{d}}$	$\parallel$	$1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3}$	$\Longrightarrow$	(Theory Learn)
$1\ \overline{2}^{\mathsf{d}}$	$\parallel$	$1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2$	$\Longrightarrow$	(Backjump)
$1\ 2$	$\parallel$	$1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2$	$\Longrightarrow$	(UnitProp)
$1\ 2\ 3\ \overline{4}$	$\parallel$	$1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2$		

From SAT to SMT — Eager Conflict Detection

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$
\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Decide}) \\
1\ \overline{2}^{\text{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Theory Learn}) \\
1\ \overline{2}^{\text{d}} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & \Longrightarrow \quad (\text{Backjump}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & \Longrightarrow \quad (\text{UnitProp}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2 & \Longrightarrow \quad (\text{Theory Learn}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 &
\end{array}
$$

From SAT to SMT — Eager Conflict Detection

$$\underbrace{g(a) = c}_{1} \quad \land \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \lor \underbrace{g(a) = d}_{3} \quad \land \quad \underbrace{c \neq d}_{\overline{4}} \lor \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \lor 3,\ \overline{4} \lor \overline{3} & \Longrightarrow \quad (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \lor 3,\ \overline{4} \lor \overline{3} & \Longrightarrow \quad (\text{Decide}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \lor 3,\ \overline{4} \lor \overline{3} & \Longrightarrow \quad (\text{Theory Learn}) \\
1\ \overline{2}^{\mathsf{d}} & \| & 1,\ \overline{2} \lor 3,\ \overline{4} \lor \overline{3},\ \overline{1} \lor 2 & \Longrightarrow \quad (\text{Backjump}) \\
1\ 2 & \| & 1,\ \overline{2} \lor 3,\ \overline{4} \lor \overline{3},\ \overline{1} \lor 2 & \Longrightarrow \quad (\text{UnitProp}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2} \lor 3,\ \overline{4} \lor \overline{3},\ \overline{1} \lor 2 & \Longrightarrow \quad (\text{Theory Learn}) \\
1\ 2\ 3\ \overline{4} & \| & 1,\ \overline{2} \lor 3,\ \overline{4} \lor \overline{3},\ \overline{1} \lor 2,\ \overline{1} \lor \overline{3} \lor 4 & \Longrightarrow \quad (\text{Fail}) \\
 & & \textit{fail} &
\end{array}$$

Theory Propagation

A final improvement is to add the following rule:

Theory Propagate :

$$M \parallel F \quad \Longrightarrow \quad M\ l \parallel F \quad \textbf{if} \begin{cases} M \models_T l \\ l \text{ or } \neg l \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases}$$

This rule allows a theory solver to inform the SAT solver if it happens to know that an unassigned literal is entailed by M.

Experimental results show that this can also be very helpful in practice.

From SAT to SMT — Theory Propagation

$$\underbrace{g(a) = c}_{1} \quad\wedge\quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad\wedge\quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\emptyset \;\|\; 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3}$$

From SAT to SMT — Theory Propagation

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{lclcl} \emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{UnitProp}) \\ 1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & & \end{array}$$

From SAT to SMT — Theory Propagation

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcllcl}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{Theory Propagate}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & &
\end{array}$$

From SAT to SMT — Theory Propagation

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \ (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \ (\text{Theory Propagate}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \ (\text{UnitProp}) \\
1\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} &
\end{array}$$

From SAT to SMT — Theory Propagation

$$\underbrace{g(a)=c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{UnitProp}) \\
1 \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{Theory Propagate}) \\
1\ 2 \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{UnitProp}) \\
1\ 2\ 3 \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow & (\text{Theory Propagate}) \\
1\ 2\ 3\ 4 \parallel & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & &
\end{array}$$

From SAT to SMT — Theory Propagation

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}} \vee \underbrace{g(a) \neq d}_{\overline{3}}$$

$$\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{UnitProp}) \\
1 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Theory Propagate}) \\
1\ 2 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{UnitProp}) \\
1\ 2\ 3 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Theory Propagate}) \\
1\ 2\ 3\ 4 & \| & 1,\ \overline{2} \vee 3,\ \overline{4} \vee \overline{3} & \Longrightarrow \quad (\text{Fail}) \\
 & \mathit{fail} & &
\end{array}$$

Extensions

We briefly mention two extensions.

The first is to allow the theory solver to use the SAT solver for internal case splitting [BNOT06]

We do this by allowing the learning rule to introduce new variables and terms

Extended T-Learn :

$$M \parallel F \quad \Longrightarrow \quad M \parallel F, C \quad \textbf{if} \begin{cases} \text{each atom of } C \text{ occurs in } F \text{ or in } \mathcal{L}(M) \\ F \models_T \exists^*(C) \end{cases}$$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F$

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathrm{d}}, x \neq z \parallel F$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \qquad \Longrightarrow \qquad$ UnitProp

$x = \{y\}, x = y \cup z \parallel F$

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F$ $\Longrightarrow$ UnitProp

$x = \{y\}, x = y \cup z \parallel F$ $\Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}} \parallel F$

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$$\begin{array}{lll}
\emptyset \parallel F & \Longrightarrow & \textsf{UnitProp} \\
x = \{y\}, x = y \cup z \parallel F & \Longrightarrow & \textsf{Decide} \\
x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}} \parallel F & \Longrightarrow & \textsf{UnitProp} \\
x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F & &
\end{array}$$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F$	$\Longrightarrow$	UnitProp
$x = \{y\}, x = y \cup z \parallel F$	$\Longrightarrow$	Decide
$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}} \parallel F$	$\Longrightarrow$	UnitProp
$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F$	$\Longrightarrow$	Extended T-Learn
$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$		

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$$\begin{array}{lll}
\emptyset \parallel F & \Longrightarrow & \text{UnitProp} \\
x = \{y\}, x = y \cup z \parallel F & \Longrightarrow & \text{Decide} \\
x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}} \parallel F & \Longrightarrow & \text{UnitProp} \\
x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F & \Longrightarrow & \text{Extended } T\text{-Learn} \\
\multicolumn{3}{l}{x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)} \\
 & \Longrightarrow & \text{Decide}
\end{array}$$

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \\ w \in x^{\mathsf{d}} \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \implies$ UnitProp

$x = \{y\}, x = y \cup z \parallel F \implies$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}} \parallel F \implies$ UnitProp

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F \implies$ Extended T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\implies$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z$
$w \in x^{\mathsf{d}}$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\implies$ UnitProp

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z$
$w \in x^{\mathsf{d}}, w \notin z$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$$
\begin{array}{lll}
\emptyset \parallel F & \Longrightarrow & \text{UnitProp} \\
x = \{y\}, x = y \cup z \parallel F & \Longrightarrow & \text{Decide} \\
x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}} \parallel F & \Longrightarrow & \text{UnitProp} \\
x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F & \Longrightarrow & \text{Extended } T\text{-Learn}
\end{array}
$$

$$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

$$\Longrightarrow \quad \text{Decide}$$

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \\ w \in x^{\mathsf{d}} \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

$$\Longrightarrow \quad \text{UnitProp}$$

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \\ w \in x^{\mathsf{d}}, w \notin z \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

Theory: $w \in y \cup z$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \quad \Longrightarrow$ UnitProp

$x = \{y\}, x = y \cup z \parallel F \quad \Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}} \parallel F \quad \Longrightarrow$ UnitProp

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F \quad \Longrightarrow$ Extended T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z$
$w \in x^{\mathsf{d}}$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ UnitProp

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z$
$w \in x^{\mathsf{d}}, w \notin z$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

Theory: $w \in y \cup z \quad \ldots \quad w \in y$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \quad \Longrightarrow$ UnitProp

$x = \{y\}, x = y \cup z \parallel F \quad \Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}} \parallel F \quad \Longrightarrow$ UnitProp

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F \quad \Longrightarrow$ Extended T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z$
$w \in x^{\mathsf{d}}$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ UnitProp

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z$
$w \in x^{\mathsf{d}}, w \notin z$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

Theory: $w \in y \cup z \quad \ldots \quad w \in y \quad \ldots \quad w \in \emptyset$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$$\begin{array}{lll}
\emptyset \parallel F & \Longrightarrow & \text{UnitProp} \\
x = \{y\}, x = y \cup z \parallel F & \Longrightarrow & \text{Decide} \\
x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}} \parallel F & \Longrightarrow & \text{UnitProp} \\
x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F & \Longrightarrow & \text{Extended } T\text{-Learn}
\end{array}$$

$$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

$$\Longrightarrow \quad \text{Decide}$$

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \\ w \in x^{\mathsf{d}} \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

$$\Longrightarrow \quad \text{UnitProp}$$

$$\begin{array}{l} x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \\ w \in x^{\mathsf{d}}, w \notin z \end{array} \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$$

Theory: $w \in y \cup z \quad \ldots \quad w \in y \quad \ldots \quad w \in \emptyset \quad \ldots \quad \perp$

Example: Theory of Sets

Let $F = (x = \{y\}), (x = y \cup z), (y \neq \emptyset \vee x \neq z)$:

$\emptyset \parallel F \quad \Longrightarrow$ UnitProp

$x = \{y\}, x = y \cup z \parallel F \quad \Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}} \parallel F \quad \Longrightarrow$ UnitProp

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F \quad \Longrightarrow$ Extended T-Learn

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z \parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ Decide

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z$
$w \in x^{\mathsf{d}}$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

$\Longrightarrow$ UnitProp

$x = \{y\}, x = y \cup z, y = \emptyset^{\mathsf{d}}, x \neq z$
$w \in x^{\mathsf{d}}, w \notin z$ $\parallel F, (x = z \vee w \in x \vee w \in z), (x = z \vee w \notin x \vee w \notin z)$

Theory: $w \in y \cup z \quad \ldots \quad w \in y \quad \ldots \quad w \in \emptyset \quad \ldots \quad \perp$

$\Longrightarrow$ Backjump

$\ldots$

Quantifiers

The Abstract DPLL Modulo Theories framework can also be extended to include rules for quantifier instantiation [GBT07].

- First, we extend the notion of literal to that of an *abstract* literal which may include quantified formulas in place of atomic formulas.
- Add two additional rules:

Inst_$\exists$:

$$M \parallel F \quad \Longrightarrow \quad M \parallel F, (\neg\exists\, x.\, P \vee P[x/sk]) \quad \textbf{if} \begin{cases} \exists\, x\; P \text{ is an abstract literal in } M \\ sk \text{ is a fresh constant.} \end{cases}$$

Inst_$\forall$:

$$M \parallel F \quad \Longrightarrow \quad M \parallel F, (\neg\forall\, x.\, P \vee P[x/t]) \quad \textbf{if} \begin{cases} \forall\, x\; P \text{ is an abstract literal in } M \\ t \text{ is a ground term.} \end{cases}$$

An Example

Suppose a and b are constant symbols and f is an uninterpreted function symbol. We show how to prove the validity of the following formula:

$$(0 \leq b \wedge (\forall x.\, 0 \leq x \rightarrow f(x) = a)) \rightarrow f(b) = a$$

We first negate the formula and put it into abstract CNF. The result is three unit clauses:

$$(0 \leq b) \wedge (\forall x.\, (\neg 0 \leq x \vee f(x) = a)) \wedge (\neg f(b) = a)$$

An Example

Let l_1, l_2, l_3 denote the three abstract literals in the above clauses. Then the following is a derivation in the extended framework:

$$\begin{array}{rcll}
\emptyset & \| & (l_1)(l_2)(l_3) & \Longrightarrow (\textsf{UnitProp}) \\
l_1, l_2, l_3 & \| & (l_1)(l_2)(l_3) & \Longrightarrow (\textsf{Inst\_}\forall) \\
l_1, l_2, l_3 & \| & (l_1)(l_2)(l_3)(\neg(0 \leq b) \vee f(b) = a) & \Longrightarrow (\textsf{Fail}) \\
\mathit{fail} & & &
\end{array}$$

The last transition is possible because M falsifies the last clause in F and contains no decisions (case-splits). As a result, we may conclude that the original set of clauses is unsatisfiable, which implies that the original formula is valid.

Quantifiers

The simple technique of quantifier instantiation is remarkably effective on verification benchmarks.

The main difficulty is coming up with the right terms to instantiate.

Matching techniques pioneered by Simplify [DNS03] have recently been adopted and improved by several modern SMT solvers [BdM07, GBT07].

SMT Solvers: State of the Art

Building on the fast SAT technology, SMT solvers have improving dramatically.

The winners of this year's SMT competition are orders of magnitude faster than those of just a couple of years ago.

Current leading solvers include:

- Barcelogic (U Barcelona, Spain)
- CVC3 (NYU, U Iowa)
- MathSAT (U Trento, Italy)
- Yices (SRI)
- Z3 (Microsoft)

SMT solvers are becoming the engine of choice for an ever-increasing set of verification applications.

Roadmap

Satisfiability Modulo Theories

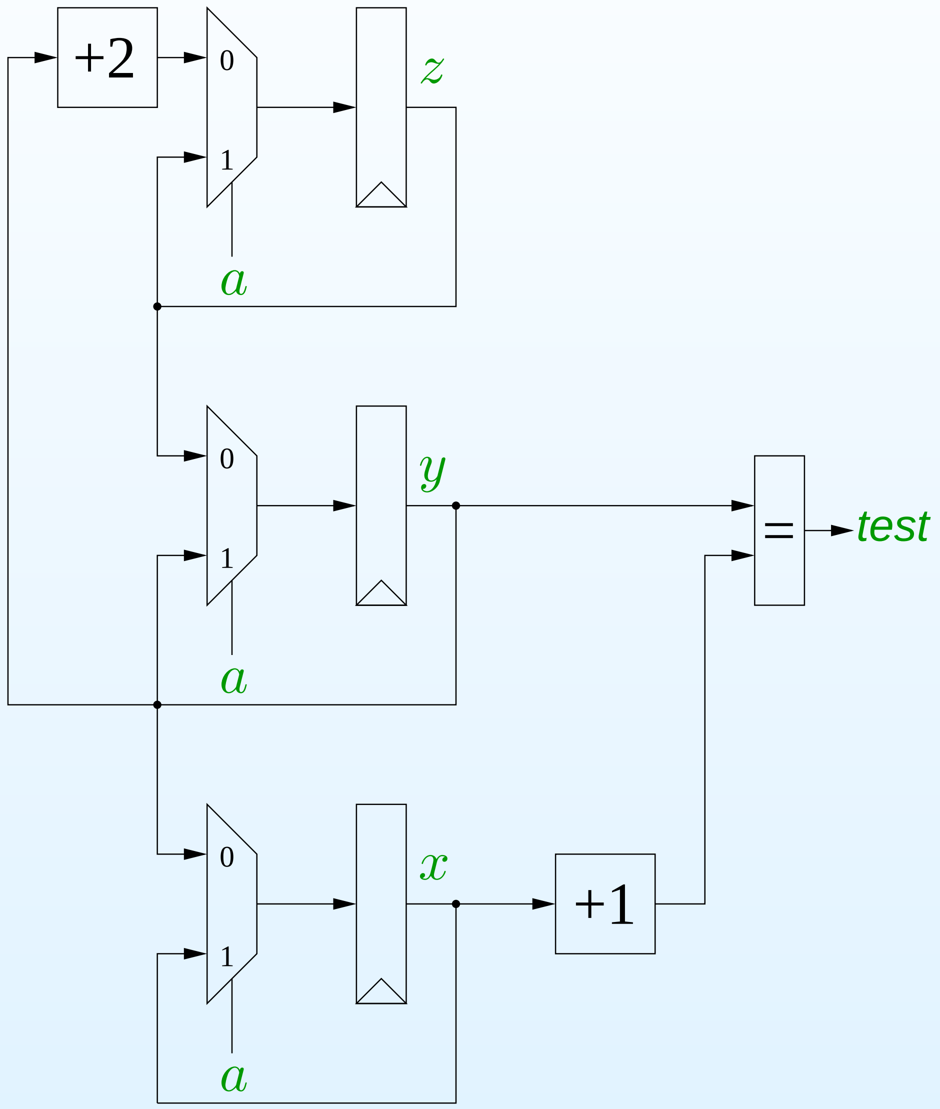
- First-Order Logic
- Specific Theories
- Theory Solvers
- Combining Theory Solvers
- Combining with SAT
- Modeling in SMT

Modeling

The language of SMT allows us to model at a higher level of abstraction.

Consider again the circuit example.

Running Example



Modeling

Recall that the invariant of the circuit is captured by the following formula:

```
(y = x + 1 AND z = x + 2 AND
 x' = IF a THEN x ELSE y AND
 y' = IF a THEN y ELSE z AND
 z' = IF a THEN z ELSE y + 2) IMPLIES
 y' = x' + 1 AND z' = x' + 2
```

When using a SAT solver, this formula had to be encoded into propositional logic

Using an SMT solver, the formula can be solved as it is

Modeling

Notice that at this level of abstraction, we can prove the formula is true for arbitrary integers, eliminating the need to consider the size of the registers

Alternatively, if we are concerned about overflow, we can use the theory of bitvectors, which still has the advantage that the size of the formula does not increase with increasing bit-width.

Modeling Software Using SMT

Consider the following lines of code:

$$\begin{aligned} l_0 &: \quad a[i] := a[i] + 1; \\ l_1 &: \quad a[i+1] := a[i-1] - 1; \\ l_2 &: \end{aligned}$$

This can be modeled in SMT as follows:

```
i0, i1, i2 : INT;
a0, a1, a2 : ARRAY INT OF INT;

ASSERT (a1 = a0 WITH [i0] := a0[i0]+1) AND (i1 = i0);
ASSERT (a2 = a1 WITH [i1+1] := a1[i1-1]-1) AND (i2 = i1);
```

Modeling Software Using SMT

A proof rule used in compiler verification needs to check whether the result is equivalent when the two statements are swapped. This can be modeled as follows:

```
i0, i1, i2, i3, i4 : INT;
a0, a1, a2, a3, a4 : ARRAY INT OF INT;

ASSERT (a1 = a0 WITH [i0] := a0[i0]+1) AND (i1 = i0);
ASSERT (a2 = a1 WITH [i1+1] := a1[i1-1]-1) AND (i2 = i1);

ASSERT (a3 = a0 WITH [i0+1] := a0[i0-1]-1) AND (i3 = i0);
ASSERT (a4 = a3 WITH [i3] := a3[i3]+1) AND (i4 = i3);

QUERY (i2 = i4 AND a2 = a4);
```

Modeling Reactive Systems Using CVC3

A more efficient encoding ignores variables that do not change and uses the *LET* construct to introduce temporary expressions.

```
i : INT;
a : ARRAY INT OF INT;

QUERY
 (LET a1 = a WITH [i] := a[i]+1 IN
      a1 WITH [i+1] := a1[i-1]-1) =
 (LET a1 = a WITH [i+1] := a[i-1]-1 IN
      a1 WITH [i] := a1[i]+1);
```

Some Challenges

- Better integration of SAT and SMT
- More complete techniques for quantifiers
- Parallel SMT
- Improving the SMT-LIB standard
- Producing and Checking Proofs
- Nonlinear arithmetic

References

[Bar03] Clark Barrett. *Checking Validity of Quantifier-Free Formulas in Combinations of First-Order Theories*. PhD thesis, Stanford University, 2003

[BdM07] Nikolaj Bjørner and Leonardo de Moura. Efficient E-matching for SMT solvers. In Frank Pfenning, editor, *Proceedings of the 21^{st} International Conference on Automated Deduction (CADE '07)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 183–198. Springer-Verlag, July 2007

[BDL98] Clark W. Barrett, David L. Dill, and Jeremy R. Levitt. A decision procedure for bit-vector arithmetic. In *Proceedings of the 35^{th} Design Automation Conference (DAC '98)*, pages 522–527. Association for Computing Machinery, June 1998. San Francisco, California. *Best paper award*

[BDS02a] Clark W. Barrett, David L. Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the 14^{th} International Conference on Computer Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer-Verlag, July 2002. Copenhagen, Denmark

References

[BDS02b] Clark W. Barrett, David L. Dill, and Aaron Stump. A generalization of Shostak's method for combining decision procedures. In Alessandro Armando, editor, *Proceedings of the 4^{th} International Workshop on Frontiers of Combining Systems (FroCoS '02)*, volume 2309 of *Lecture Notes in Artificial Intelligence*, pages 132–146. Springer-Verlag, April 2002. Santa Margherita Ligure, Italy

[BNOT06] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in SAT modulo theories. In Miki Hermann and Andrei Voronkov, editors, *Proceedings of the 13^{th} International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '06)*, volume 4246 of *Lecture Notes in Computer Science*, pages 512–526. Springer-Verlag, November 2006. Phnom Penh, Cambodia

[BP98] Nikolaj Bjørner and Mark C. Pichora. Deciding fixed and non-fixed size bit-vectors. In *TACAS '98: Proceedings of the 4^{th} International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 376–392. Springer-Verlag, 1998

[BST07] Clark Barrett, Igor Shikanian, and Cesare Tinelli. An abstract decision procedure for a theory of inductive data types. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:21–46, 2007

References

[CLS96] D. Cyrluk, P. Lincoln, and N. Shankar. On shostak's decision procedure for combinations of theories. In M. McRobbie and J. Slaney, editors, *13th International Conference on Computer Aided Deduction*, volume 1104 of *Lecture Notes in Computer Science*, pages 463–477. Springer-Verlag, 1996

[CMR97] David Cyrluk, M. Oliver Möller, and Harald Ruess. An efficient decision procedure for the theory of fixed-size bit-vectors. In *Proceedings of the* 9^{th} *International Conference on Computer Aided Verification (CAV '97)*, pages 60–71. Springer-Verlag, 1997

[CZ00] Domenico. Cantone and Calogero G. Zarba. A new fast tableau-based decision procedure for an unquantified fragment of set theory. In Ricardo Caferra and Gernot Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *Lecture Notes in Artificial Intelligence*, pages 127–137. Springer, 2000

[dMRS02] L. de Moura, H. Rueß, and M. Sorea. Lazy Theorem Proving for Bounded Model Checking over Infinite Domains. In *Proc. of the 18th International Conference on Automated Deduction*, volume 2392 of *LNCS*, pages 438–455. Springer, July 2002

[DNS03] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Laboratories Palo Alto, 2003

References

[End00] Herbert B. Enderton. *A Mathematical Introduction to Logic*. Undergraduate Texts in Mathematics. Academic Press, second edition edition, 2000

[EKM98] Jacob Elgaard, Nils Klarlund, and Anders Möller. Mona 1.x: New techniques for WS1S and WS2S. In *Proceedings of the 10^{th} International Conference on Computer Aided Verification (CAV '98)*, volume 1427 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998

[Gan02] Harald Ganzinger. Shostak light. In Andrei Voronkov, editor, *Automated Deduction – CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 332–346. Springer, 2002

[GBD05] Vijay Ganesh, Sergey Berezin, and David L. Dill. A decision procedure for fixed-width bit-vectors, January 2005. Unpublished Manuscript

[GBT07] Yeting Ge, Clark Barrett, and Cesare Tinelli. Solving quantified verification condisions using satisfiability modulo theories. In *Proceedings of the 21^{st} International Conference on Automated Deduction (CADE '07)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, July 2007. Bremen, Germany

[KC03] Sava Krstic and Sylvain Conchon. Canonization for disjoint union of theories. In *Proceedings of the 19^{th} International Conference on Computer Aided Deduction (CADE '03)*, 2003

References

[LQ08] S. K. Lahiri and S. Qadeer. Back to the future: Revisiting precise program verification using smt solvers. In *Proceedings of the 35th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, 2008

[Möl97] M. Oliver Möller. *Solving Bit-Vector Equations – a Decision Procedure for Hardware Verification*. PhD thesis, University of Ulm, 1997

[MZ03] Zohar Manna and Calogero Zarba. Combining decision procedures. In *Formal Methods at the Crossroads: from Panacea to Foundational Support*, volume 2787 of *Lecture Notes in Computer Science*, pages 381–422. Springer-Verlag, November 2003

[NO79] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, October 1979

[NO80] Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006

References

[Opp80] Derek C. Oppen. Reasoning about recursively defined data structures. *J. ACM*, 27(3):403–411, 1980

[RBH07] Zvonimir Rakamarić, Jesse Bingham, and Alan J. Hu. An inference-rule-based decision procedure for verification of heap-manipulating programs with mutable data and cyclic data structures. In *Verification, Model Checking, and Abstract Interpretation: 8th International Conference*, pages 106–121. Springer, 2007. Lecture Notes in Computer Science Vol. 4349

[RS01] H. Ruess and N. Shankar. Deconstructing shostak. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 19–28, June 2001

[Sho84] R. Shostak. Deciding combinations of theories. *Journal of the Association for Computing Machinery*, 31(1):1–12, 1984

[TH96] C. Tinelli and M. Harandi. A new correctness proof of the nelson-oppen combination procedure. In F. Baader and K. Schulz, editors, *1st International Workshop on Frontiers of Combining Systems (FroCoS'96)*, volume 3 of *Applied Logic Series*. Kluwer Academic Publishers, 1996

References

[YRS$^+$06] Greta Yorsh, Alexander Rabinovich, Mooly Sagiv, Antoine Meyer, and Ahmed Bouajjani. A logic of reachable patterns in linked data-structures. In *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS '06)*, 2006

[ZSM04a] T. Zhang, H. B. Sipma, and Z. Manna. Decision procedures for term algebras with integer constraints. In *Proceedings of the 2^{nd} International Joint Conference on Automated Reasoning (IJCAR '04) LNCS 3097*, pages 152–167, 2004

[ZSM04b] Ting Zhang, Henny B. Sipma, and Zohar Manna. Term algebras with length function and bounded quantifier alternation. In *Proceedings of the 17^{th} International Conference on Theorem Proving in Higher Order Logics (TPHOLs '04)*, volume 3223 of *Lecture Notes in Computer Science*, pages 321–336, 2004