Satisfiability with and without Theories

KR 2010, Toronto

Leonardo de Moura
Microsoft Research
Verification/Analysis tools need some form of Symbolic Reasoning
Logic is “The Calculus of Computer Science” (Z. Manna).

High computational complexity
Applications

- Test case generation
- Verifying Compilers
- Predicate Abstraction
- Invariant Generation
- Type Checking
- Model Based Testing
Some Applications @ Microsoft

- Spec# Programming System
- HAVOC
- Formula
- Hyper-V
- Terminator T-2
- VCC
- Microsoft Virtualization
- SLAM
- Vigilante
- NModel
- SpecExplorer
- SAGE
- F7
unsigned GCD(x, y) {
    requires(y > 0);
    while (true) {
        unsigned m = x % y;
        if (m == 0)
            return y;
        x = y;
        y = m;
    }
}

We want a trace where the loop is executed twice.

(y₀ > 0) and
(m₀ = x₀ % y₀) and
not (m₀ = 0) and
(x₁ = y₀) and
(y₁ = m₀) and
(m₁ = x₁ % y₁) and
(m₁ = 0)

x₀ = 2
y₀ = 4
m₀ = 2
x₁ = 4
y₁ = 2
m₁ = 0
Type checking

Signature:

```plaintext
div : int, \{ x : int | x \neq 0 \} \rightarrow int
```

Call site:

```plaintext
if a \leq 1 and a \leq b then
    return div(a, b)
```

Verification condition

```plaintext
a \leq 1 and a \leq b implies b \neq 0
```
Logic is the art and science of effective reasoning.

How can we draw general and reliable conclusions from a collection of facts?

**Formal logic**: Precise, syntactic characterizations of well-formed expressions and valid deductions.

Formal logic makes it possible to calculate consequences at the symbolic level.

Computers can be used to automate such symbolic calculations.
Logic studies the relationship between language, meaning, and (proof) method.

A logic consists of a language in which (well-formed) sentences are expressed.

A semantic that distinguishes the valid sentences from the refutable ones.

A proof system for constructing arguments justifying valid sentences.

Examples of logics include propositional logic, equational logic, first-order logic, higher-order logic, and modal logics.
A language consists of logical symbols whose interpretations are fixed, and non-logical ones whose interpretations vary.

These symbols are combined together to form well-formed formulas.

In propositional logic PL, the connectives \( \land \), \( \lor \), and \( \neg \) have a fixed interpretation, whereas the constants \( p, q, r \) may be interpreted at will.
Propositional Logic

Formulas: \[ \varphi := p \mid \varphi_1 \lor \varphi_2 \mid \varphi_1 \land \varphi_2 \mid \neg \varphi_1 \mid \varphi_1 \Rightarrow \varphi_2 \]

Examples:
\[ p \lor q \Rightarrow q \lor p \]
\[ p \land \neg q \land (\neg p \lor q) \]

We say \( p \) and \( q \) are propositional variables.

Exercise: Using a programming language, define a representation for formulas and a checker for well-formed formulas.
An interpretation $\mathcal{M}$ assigns truth values $\{\top, \bot\}$ to propositional variables.

Let $A$ and $B$ range over $PL$ formulas.

$\mathcal{M}[\phi]$ is the meaning of $\phi$ in $\mathcal{M}$ and is computed using truth tables:

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$A$</th>
<th>$B$</th>
<th>$\neg A$</th>
<th>$A \lor B$</th>
<th>$A \land \neg A$</th>
<th>$A \Rightarrow B$</th>
<th>$A \Rightarrow (B \lor A)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{M}_1(\phi)$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$\mathcal{M}_2(\phi)$</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$\mathcal{M}_3(\phi)$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$\mathcal{M}_4(\phi)$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$\top$</td>
</tr>
</tbody>
</table>
A formula is **satisfiable** if it has an interpretation that makes it logically true.

In this case, we say the **interpretation** is a **model**.

A formula is **unsatisfiable** if it does not have any model.

A formula is **valid** if it is logically true in any interpretation.

A propositional formula is valid if and only if its negation is unsatisfiable.
Satisfiability & Validity: examples

\[ p \lor q \Rightarrow q \lor p \]

\[ p \lor q \Rightarrow q \]

\[ p \land \neg q \land (\neg p \lor q) \]

<table>
<thead>
<tr>
<th>( \phi )</th>
<th>( A )</th>
<th>( B )</th>
<th>( \neg A )</th>
<th>( A \lor B )</th>
<th>( A \land \neg A )</th>
<th>( A \Rightarrow B )</th>
<th>( A \Rightarrow (B \lor A) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_1(\phi) )</td>
<td>\bot</td>
<td>\bot</td>
<td>\top</td>
<td>\bot</td>
<td>\bot</td>
<td>\top</td>
<td>\top</td>
</tr>
<tr>
<td>( M_2(\phi) )</td>
<td>\bot</td>
<td>\top</td>
<td>\top</td>
<td>\top</td>
<td>\bot</td>
<td>\top</td>
<td>\top</td>
</tr>
<tr>
<td>( M_3(\phi) )</td>
<td>\top</td>
<td>\bot</td>
<td>\bot</td>
<td>\top</td>
<td>\bot</td>
<td>\bot</td>
<td>\top</td>
</tr>
<tr>
<td>( M_4(\phi) )</td>
<td>\top</td>
<td>\top</td>
<td>\bot</td>
<td>\top</td>
<td>\bot</td>
<td>\top</td>
<td>\top</td>
</tr>
</tbody>
</table>
Satisfiability & Validity: examples

$p \lor q \Rightarrow q \lor p$  VALID

$p \lor q \Rightarrow q$  SATISFIABLE

$p \land \neg q \land (\neg p \lor q)$  UNSATISFIABLE

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\phi & A & B & \neg A & A \lor B & A \land \neg A & A \Rightarrow B & A \Rightarrow (B \lor A) \\
\hline
\mathcal{M}_1(\phi) & \perp & \perp & T & \perp & \perp & T & T \\
\mathcal{M}_2(\phi) & \perp & T & T & T & \perp & T & T \\
\mathcal{M}_3(\phi) & T & \perp & \perp & T & \perp & \perp & T \\
\mathcal{M}_4(\phi) & T & T & \perp & T & \perp & T & T \\
\hline
\end{array}
\]
Two formulas $A$ and $B$ are equivalent, $A \iff B$, if their truth values agree in each interpretation.

**Exercise 2**  *Prove that the following are equivalent*

1. $\neg\neg A \iff A$

2. $A \Rightarrow B \iff \neg A \lor B$

3. $\neg (A \land B) \iff \neg A \lor \neg B$

4. $\neg (A \lor B) \iff \neg A \land \neg B$

5. $\neg A \Rightarrow B \iff \neg B \Rightarrow A$
Equisatisfiable

We say formulas $A$ and $B$ are **equisatisfiable** if and only if $A$ is satisfiable if and only if $B$ is.

During this course, we will describe transformations that preserve equivalence and equisatisfiability.
Normal Forms

A formula where negation is applied only to propositional atoms is said to be in negation normal form (NNF).

A literal is either a propositional atom or its negation.

A formula that is a multiary conjunction of multiary disjunctions of literals is in conjunctive normal form (CNF).

A formula that is a multiary disjunction of multiary conjunctions of literals is in disjunctive normal form (DNF).

Exercise 3 Show that every propositional formula is equivalent to one in NNF, CNF, and DNF.

Exercise 4 Show that every $n$-ary Boolean function can be expressed using just $\neg$ and $\lor$. 
NNF?

\((p \lor \neg q) \land (q \lor \neg (r \land \neg p))\)
Normal Forms

NNF? NO

\((p \lor \neg q) \land (q \lor \neg (r \land \neg p))\)
Normal Forms

NNF? NO

\((p \lor \neg q) \land (q \lor \neg (r \land \neg p))\)

1. \(\neg \neg A \iff A\)

2. \(A \Rightarrow B \iff \neg A \lor B\)

3. \(\neg (A \land B) \iff \neg A \lor \neg B\)

4. \(\neg (A \lor B) \iff \neg A \land \neg B\)
Normal Forms

NNF? NO

\[(p \lor \neg q) \land (q \lor \neg(r \land \neg p))\]

\[\iff\]

\[(p \lor \neg q) \land (q \lor (\neg r \lor \neg \neg p))\]

\[1. \quad \neg \neg A \iff A\]

\[2. \quad A \implies B \iff \neg A \lor B\]

\[3. \quad \neg (A \land B) \iff \neg A \lor \neg B\]

\[4. \quad \neg (A \lor B) \iff \neg A \land \neg B\]
Normal Forms

NNF? NO

\((p \lor \neg q) \land (q \lor \neg(r \land \neg p))\)

\[\iff\]

\((p \lor \neg q) \land (q \lor (\neg r \lor \neg \neg p))\)

\[\iff\]

\((p \lor \neg q) \land (q \lor (\neg r \lor p))\)

1. \(\neg \neg A \iff A\)

2. \(A \Rightarrow B \iff \neg A \lor B\)

3. \(\neg(A \land B) \iff \neg A \lor \neg B\)

4. \(\neg(A \lor B) \iff \neg A \land \neg B\)
Normal Forms

CNF?

\((p \land s) \lor (\neg q \land r)) \land (q \lor \neg p \lor s) \land (\neg r \lor s)\)
CNF? NO

\((p \land s) \lor (\lnot q \land r)\) \land (q \lor \lnot p \lor s) \land (\lnot r \lor s)\)
Normal Forms

CNF? NO

$((p \land s) \lor (\neg q \land r)) \land (q \lor \neg p \lor s) \land (\neg r \lor s)$

Distributivity

1. $A \lor (B \land C) \iff (A \lor B) \land (A \lor C)$
2. $A \land (B \lor C) \iff (A \land B) \lor (A \land C)$
Normal Forms

CNF? NO

\((p \land s) \lor (\neg q \land r)\) \land (q \lor \neg p \lor s) \land (\neg r \lor s)

\iff\)

\((p \land s) \lor \neg q\) \land ((p \land s) \lor r) \land (q \lor \neg p \lor s) \land (\neg r \lor s)

Distributivity

1. \(A \lor (B \land C) \iff (A \lor B) \land (A \lor C)\)

2. \(A \land (B \lor C) \iff (A \land B) \lor (A \land C)\)
Normal Forms

CNF? NO

\[(p \land s) \lor (\neg q \land r) \land (q \lor \neg p \lor s) \land (\neg r \lor s)\]
\[\iff\]
\[((p \land s) \lor \neg q) \land ((p \land s) \lor r) \land (q \lor \neg p \lor s) \land (\neg r \lor s)\]
\[\iff\]
\[(p \lor \neg q) \land (s \lor \neg q) \land ((p \land s) \lor r) \land (q \lor \neg p \lor s) \land (\neg r \lor s)\]

Distributivity

1. \( A \lor (B \land C) \iff (A \lor B) \land (A \lor C) \)
2. \( A \land (B \lor C) \iff (A \land B) \lor (A \land C) \)
Normal Forms

CNF? NO

\[
((p \land s) \lor (\neg q \land r)) \land (q \lor \neg p \lor s) \land (\neg r \lor s)
\]

⇔

\[
((p \land s) \lor \neg q) \land ((p \land s) \lor r) \land (q \lor \neg p \lor s) \land (\neg r \lor s)
\]

⇔

\[
(p \lor \neg q) \land (s \lor \neg q) \land ((p \land s) \lor r) \land (q \lor \neg p \lor s) \land (\neg r \lor s)
\]

⇔

\[
(p \lor \neg q) \land (s \lor \neg q) \land (p \lor r) \land (s \lor r) \land (q \lor \neg p \lor s) \land (\neg r \lor s)
\]
Normal Forms

DNF?

\[ p \land (\neg p \lor q) \land (\neg q \lor r) \]
DNF? NO, actually this formula is in CNF

\[ p \land (\neg p \lor q) \land (\neg q \lor r) \]
Normal Forms

DNF? NO, actually this formula is in CNF

\[ p \land (\neg p \lor q) \land (\neg q \lor r) \]

Distributivity

1. \[ A \lor (B \land C) \Leftrightarrow (A \lor B) \land (A \lor C) \]
2. \[ A \land (B \lor C) \Leftrightarrow (A \land B) \lor (A \land C) \]
DNF? NO, actually this formula is in CNF

\[ p \land (\neg p \lor q) \land (\neg q \lor r) \]

\[ \iff \]

\[ ((p \land \neg p) \lor (p \lor q)) \land (\neg q \lor r) \]

**Distributivity**

1. \[ A \lor (B \land C) \iff (A \lor B) \land (A \lor C) \]
2. \[ A \land (B \lor C) \iff (A \land B) \lor (A \land C) \]
DNF? NO, actually this formula is in CNF

\[ p \land (\neg p \lor q) \land (\neg q \lor r) \]

\[ \iff \]

\[ ((p \land \neg p) \lor (p \lor q)) \land (\neg q \lor r) \]

\[ \iff \]

\[ (p \lor q) \land (\neg q \lor r) \]

Distributivity

1. \( A \lor (B \land C) \iff (A \lor B) \land (A \lor C) \)

2. \( A \land (B \lor C) \iff (A \land B) \lor (A \land C) \)

Other Rules

1. \( A \land \neg A \iff \bot \)

2. \( A \lor \bot \iff A \)
Normal Forms

DNF? NO, actually this formula is in CNF

\[ p \land (\neg p \lor q) \land (\neg q \lor r) \]

\[ \iff \]

\[ ((p \land \neg p) \lor (p \lor q)) \land (\neg q \lor r) \]

\[ \iff \]

\[ (p \lor q) \land (\neg q \lor r) \]

\[ \iff \]

\[ ((p \lor q) \land \neg q) \lor ((p \lor q) \land r) \]

Distributivity

1. \( A \lor (B \land C) \iff (A \lor B) \land (A \lor C) \)
2. \( A \land (B \lor C) \iff (A \land B) \lor (A \land C) \)

Other Rules

1. \( A \land \neg A \iff \bot \)
2. \( A \lor \bot \iff A \)
Normal Forms

DNF? NO, actually this formula is in CNF

\[ p \land (\neg p \lor q) \land (\neg q \lor r) \]

\[ \iff \]

\[ ((p \land \neg p) \lor (p \lor q)) \land (\neg q \lor r) \]

\[ \iff \]

\[ (p \lor q) \land (\neg q \lor r) \]

\[ \iff \]

\[ ((p \lor q) \land \neg q) \lor ((p \lor q) \land r) \]

\[ \iff \]

\[ (p \land \neg q) \lor (q \land \neg q) \lor ((p \lor q) \land r) \]

\[ \iff \]

\[ (p \land \neg q) \lor (p \land r) \lor (q \land r) \]
A CNF formula is a conjunction of clauses. A clause is a disjunction of literals.

Ex: Implement a linear-time decision procedure for 2CNF (each clause has at most 2 literals).

A clause is trivial if it contains a complementary pair of literals.

Since the order of the literals in a clause is irrelevant, the clause can be treated as a set.

A set of clauses is trivial if it contains the empty clause (false).
Equivalence rules can be used to translate any formula to CNF.

<table>
<thead>
<tr>
<th>eliminate $\Rightarrow$</th>
<th>$A \Rightarrow B \equiv \neg A \lor B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>reduce the scope of $\neg$</td>
<td>$\neg(A \lor B) \equiv \neg A \land \neg B$, $\neg(A \land B) \equiv \neg A \lor \neg B$</td>
</tr>
<tr>
<td>apply distributivity</td>
<td>$A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)$, $A \land (B \lor C) \equiv (A \land B) \lor (A \land C)$</td>
</tr>
</tbody>
</table>
The CNF translation described in the previous slide is too expensive (distributivity rule).

However, there is a *linear time* translation to CNF that produces an *equisatisfiable* formula. Replace the distributivity rules by the following rules:

\[
\frac{F[l_i \text{ op } l_j]}{F[x], x \Leftrightarrow l_i \text{ op } l_j}^* \\
\frac{x \Leftrightarrow l_i \lor l_j}{x \Leftrightarrow l_i \land l_j} \\
\frac{-x \lor l_i \lor l_j, -l_i \lor x, -l_j \lor x}{-x \lor l_i, -x \lor l_j, -l_i \lor -l_j \lor x}
\]

(*) $x$ must be a fresh variable.

Ex: Show that the rules preserve equisatisfiability.
Translation of \((p \land (q \lor r)) \lor t\):

\[
(p \land (q \lor r)) \lor t
\]

\[
(p \land x_1) \lor t, x_1 \iff q \lor r
\]

\[
x_2 \lor t, x_2 \iff p \land x_1, x_1 \iff q \lor r
\]

\[
x_2 \lor t, \neg x_2 \lor p, \neg x_2 \lor x_1, \neg p \lor \neg x_1 \lor x_2, x_1 \iff q \lor r
\]

\[
x_2 \lor t, \neg x_2 \lor p, \neg x_2 \lor x_1, \neg p \lor \neg x_1 \lor x_2, \neg x_1 \lor q \lor r, \neg q \lor x_1, \neg r \lor x_1
\]

Ex: Implement a CNF translator.
A *semantic tree* represents the set of partial interpretations for a set of clauses. A semantic tree for 
\[ \{p \lor \neg q \lor \neg r, p \lor r, p \lor q, \neg p\} : \]

A node $N$ is a *failure node* if its associated interpretation *falsifies* a clause, but its ancestor doesn’t.

Ex: Show that the semantic tree for an unsatisfiable (non-trivial) set of clauses must contain a non failure node such that its descendants are failure nodes.
Resolution

Formula must be in **CNF**.

*Resolution* procedure uses only one rule:

$$
\frac{C_1 \lor p, C_2 \lor \neg p}{C_1 \lor p, C_2 \lor \neg p, C_1 \lor C_2} \quad \text{res}
$$

The result of the resolution rule is also a clause, it is called the **resolvent**. *Duplicate literals* in a clause and *trivial clauses* are eliminated.

There is no *branching* in the resolution procedure.

Example: The resolvent of \( p \lor q \lor r \), and \( \neg p \lor r \lor t \) is \( q \lor r \lor t \).

*Termination argument*: there is a *finite* number of distinct clauses over \( n \) propositional variables.

Ex: Show that the resolution rule is sound.
A refutation of \( \neg p \lor \neg q \lor r, p \lor r, q \lor r, \neg r \):

Ex: Implement a naïve resolution procedure.
Completeness of Resolution

Let $\text{Res}(S)$ be the closure of $S$ under the resolution rule.

Completeness: $S$ is unsatisfiable iff $\text{Res}(S)$ contains the empty clause.

Proof ($\Rightarrow$):

Assume that $S$ is unsatisfiable, and $\text{Res}(S)$ does not contain the empty clause.

Key points: $\text{Res}(S)$ is unsatisfiable, and $\text{Res}(S)$ is a non trivial set of clauses.

The semantic tree of $\text{Res}(S)$ must contain a non failure node $N$ such that its descendants ($N_p$, $N_{\neg p}$) are failure nodes.
Completeness of Resolution

There is $C_1 \lor \neg p$ which is falsified by $N_p$, but not by $N$.

There is $C_2 \lor p$ which is falsified by $N_{\neg p}$, but not by $N$.

$C_1 \lor C_2$ is the resolvent of $C_1 \lor \neg p$ and $C_2 \lor p$.

$C_1 \lor C_2$ is in $Res(S)$, and it is falsified by $N$ (contradiction).

Proof ($\iff$): $Res(S)$ is unsatisfiable, and equivalent to $S$. So, $S$ is unsatisfiable.
The *resolution* procedure may generate several *irrelevant* and *redundant clauses*.

*Subsumption* is a clause *deletion strategy* for the resolution procedure.

\[
\frac{C_1, C_1 \lor C_2}{\text{sub} \quad C_1}
\]

Example: \( p \lor \neg q \) *subsumes* \( p \lor \neg q \lor r \lor t \).

Deletion strategy: Remove the subsumed clauses.
Unit resolution: one of the clauses is a unit clause.

\[
\frac{C \lor \bar{l}, l}{C, l}\text{  unit}
\]

Unit resolution always decreases the configuration size (\(C \lor \bar{l}\) is subsumed by \(C\)).

Input resolution: one of the clauses is in \(S\).

Ex: Show that the unit and input resolution procedures are not complete.

Ex: Show that a set of clauses \(S\) has an unit refutation iff it has an input refutation (hint: induction on the number of propositions).
Horn Clauses

Each clause has at most one positive literal.

Rule base systems \((-p_1 \lor \ldots \lor -p_n \lor q \equiv p_1 \land \ldots \land p_n \Rightarrow q)\).

Positive unit rule:

\[
\frac{C \lor -p, p}{\text{unit}^+} \quad \frac{C, p}{C, p}
\]

Horn clauses are the basis of programming languages as Prolog.

Ex: Show that the positive unit rule is a complete procedure for Horn clauses.

Ex: Implement a linear time algorithm for Horn clauses.
DPLL = Unit resolution + Split rule.

\[
\frac{\Gamma}{\Gamma, p \mid \Gamma, \neg p} \quad \text{split} \quad p \text{ and } \neg p \text{ are not in } \Gamma.
\]

\[
\frac{C \lor \neg l, l}{C, l} \quad \text{unit}
\]

Used in the most efficient SAT solvers.
A literal is **pure** if only occurs positively or negatively.

**Example:**

\[ \varphi = (\neg x_1 \lor x_2) \land (x_3 \lor \neg x_2) \land (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]

\( \neg x_1 \) and \( x_3 \) are pure literals

**Pure literal rule:**

Clauses containing pure literals can be removed from the formula (i.e. just satisfy those pure literals)

\[ \varphi_{\neg x_1, x_3} = (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]

*Preserve satisfiability, not logical equivalency!*
Pure Literals

A literal is **pure** if only occurs positively or negatively.

**Example:**

\[ \varphi = (\neg x_1 \lor x_2) \land (x_3 \lor \neg x_2) \land (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]

\(\neg x_1\) and \(x_3\) are pure literals

**Pure literal rule:**

Clauses containing pure literals can be removed from the formula (i.e. just satisfy those pure literals)

\[ \varphi_{\neg x_1, x_3} = (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]

Preserve satisfiability, not logical equivalency!
DPLL (as a procedure)

- Standard **backtrack search**
- DPLL(F):
  - Apply unit propagation
  - If conflict identified, return **UNSAT**
  - Apply the pure literal rule
  - If F is satisfied (empty), return **SAT**
  - Select decision variable x
    - If DPLL(F \( \land x \)) = SAT return **SAT**
    - return DPLL(F \( \land \neg x \))
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
DPLL (example)

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
\( \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \)
\( \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \)
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
Some Applications
Let $x$, $y$ and $z$ be 8-bit (unsigned) integers.

Is $x > 0 \land y > 0 \land z = x + y \Rightarrow z > 0$ valid?

Is $x > 0 \land y > 0 \land z = x + y \land \neg(z > 0)$ satisfiable?
We can encode bit-vector satisfiability problems in propositional logic.

Idea 1:
Use $n$ propositional variables to encode $n$-bit integers.
$$x \rightarrow (x_1, \ldots, x_n)$$

Idea 2:
Encode arithmetic operations using hardware circuits.
\[ p \iff q \text{ is equivalent to } (\neg p \lor q) \land (\neg q \lor p) \]

The bit-vector equation \( x = y \) is encoded as:
\[ (x_1 \iff y_1) \land ... \land (x_n \iff y_n) \]
We use \((r_1, \ldots, r_n)\) to store the result of \(x + y\)

\(p \text{xor} q\) is defined as \(\neg(p \leftrightarrow q)\)

XOR is the 1-bit adder

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Encoding 1-bit full adder

1-bit full adder

Three inputs: $x$, $y$, $c_{in}$

Two outputs: $r$, $c_{out}$

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$c_{in}$</th>
<th>$r = x \oplus y \oplus c_{in}$</th>
<th>$c_{out} = (x \land y) \lor (x \land c_{in}) \lor (y \land c_{in})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
We use \((r_1, \ldots, r_n)\) to store the result of \(x + y\), and \((c_1, \ldots, c_n)\)

\[
\begin{align*}
    r_1 & \iff (x_1 \text{xor} \ y_1) \\
    c_1 & \iff (x_1 \text{ and} \ y_1) \\
    r_2 & \iff (x_2 \text{xor} \ y_2 \text{xor} \ c_1) \\
    c_2 & \iff (x_2 \text{ and} \ y_2) \text{ or} (x_2 \text{ and} \ c_1) \text{ or} (y_2 \text{ and} \ c_1) \\
    \vdots \\
    r_n & \iff (x_n \text{xor} \ y_n \text{xor} \ c_{n-1}) \\
    c_n & \iff (x_n \text{ and} \ y_n) \text{ or} (x_n \text{ and} \ c_{n-1}) \text{ or} (y_n \text{ and} \ c_{n-1})
\end{align*}
\]
unsigned GCD(x, y) {
    requires(y > 0);
    while (true) {
        unsigned m = x % y;
        if (m == 0) return y;
        x = y;
        y = m;
    }
}

We want a trace where the loop is executed twice.

(y₀ > 0) and
(m₀ = x₀ % y₀) and
not (m₀ = 0) and
(x₁ = y₀) and
(y₁ = m₀) and
(m₁ = x₁ % y₁) and
(m₁ = 0)

x₀ = 2
y₀ = 4
m₀ = 2
x₁ = 4
y₁ = 2
m₁ = 0

SSA
Solver
Experimental Exercises

- The first step is to pick up a SAT solver.
- Play with simple examples
- Translate your problem into SAT
- Experiment
Available SAT Solvers

Several open source SAT solvers exist:

**Minisat (C++)** [www.minisat.se](http://www.minisat.se) Presumably the most widely used within the SAT community. Used to be the best general purpose SAT solver. A large community around the solver.

**Picosat (C)/Precosat (C++)**


**SAT4J (Java)** [http://www.sat4j.org](http://www.sat4j.org). For Java users. Far less efficient than the two others.

**UBCSAT (C)** [http://www.satlib.org/ubcsat/](http://www.satlib.org/ubcsat/) Very efficient stochastic local search for SAT.

[http://www.satcompetition.org](http://www.satcompetition.org) Both the binaries and the source code of the solvers are made available for research purposes.
Available Examples

- Satisfiability library: [http://www.satlib.org](http://www.satlib.org)
- The SAT competition: [http://www.satcompetition.org](http://www.satcompetition.org)
- Search the WEB: “SAT benchmarks”
All SAT solvers support the very simple DIMACS CNF input format:

\[(a \lor b \lor \neg c) \land (\neg b \lor \neg c)\]

will be translated into

```
p cnf 3 2
1 2 -3 0
-2 -3 0
```

The first line is of the form
```
p cnf <maxVarId> <numberOfClauses>
```
Each variable is represented by an integer, negative literals as negative integers, 0 is the clause separator.
Is formula $F$ satisfiable modulo theory $T$?

SMT solvers have specialized algorithms for $T$. 
b + 2 = c and f(read(write(a,b,3), c-2)) ≠ f(c-b+1)
Satisfiability Modulo Theories (SMT)

\[ b + 2 = c \quad \text{and} \quad f(\text{read}(\text{write}(a, b, 3), c-2)) \neq f(c-b+1) \]
b + 2 = c  and  \( f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \)
\( b + 2 = c \) and \( f(\text{read(\text{write}(a,b,3), c-2))) \neq f(c-b+1) \)
b + 2 = c \text{ and } f(\text{read}(\text{write}(a, b, 3), c-2)) \neq f(c-b+1)

Substituting c by b+2
b + 2 = c and $f(\text{read(write(a,b,3), b+2-2))} \neq f(b+2-b+1)$

Simplifying
b + 2 = c and \( f(\text{read}(\text{write}(a,b,3), b)) \neq f(3) \)
\[ b + 2 = c \text{ and } f(\text{read}(\text{write}(a,b,3), b)) \neq f(3) \]

Applying array theory axiom
forall a,i,v: \text{read}(\text{write}(a,i,v), i) = v
b + 2 = c and \( f(3) \neq f(3) \)

Inconsistent/Unsatisfiable
Repository of Benchmarks

http://www.smtlib.org

Benchmarks are divided in “logics”:

- **QF_UF**: unquantified formulas built over a signature of uninterpreted sort, function and predicate symbols.

- **QF_UFLIA**: unquantified linear integer arithmetic with uninterpreted sort, function, and predicate symbols.

- **AUFLIA**: closed linear formulas over the theory of integer arrays with free sort, function and predicate symbols.
For most SMT solvers: **F is a set of ground formulas**

Many Applications

- Bounded Model Checking
- Test-Case Generation
An SMT Solver is a collection of 
Little Engines of Proof
An SMT Solver is a collection of Little Engines of Proof.

Examples:
- SAT Solver
- Equality solver
\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e, \ a \neq s \]
Deciding Equality

\[ a = b, \quad b = c, \quad d = e, \quad b = s, \quad d = t, \quad a \neq e, \quad a \neq s \]
Deciding Equality

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e, \ a \neq s \]
Deciding Equality

\[ a = b, \quad b = c, \quad d = e, \quad b = s, \quad d = t, \quad a \neq e, \quad a \neq s \]
Deciding Equality

\[ a = b, \quad b = c, \quad d = e, \quad b = s, \quad d = t, \quad a \neq e, \quad a \neq s \]
Deciding Equality

a = b, b = c, d = e, b = s, d = t, a ≠ e, a ≠ s
Deciding Equality

\[a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e, \ a \neq s\]
Deciding Equality

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e, \ a \neq s \]

- **a, b, c**
- **d, e**
- **s**
- **t**
Deciding Equality

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e, \ a \neq s \]
Deciding Equality

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e, \ a \neq s \]
Deciding Equality

\[ a = b, \quad b = c, \quad d = e, \quad b = s, \quad d = t, \quad a \neq e, \quad a \neq s \]
Deciding Equality

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e, \ a \neq s \]
Deciding Equality

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e, \ a \neq s \]

Unsatisfiable

- \{a, b, c, s\}
- \{d, e, t\}
Deciding Equality

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e \]

Model construction
Deciding Equality

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e \]

Model construction

\[ \vert M \vert = \{ \diamond_1, \diamond_2 \} \] (universe, aka domain)
Deciding Equality

\[
a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e
\]

Model construction

\[|M| = \{\spadesuit_1, \spadesuit_2\} \quad \text{(universe, aka domain)}\]

\[M(a) = \spadesuit_1 \quad \text{(assignment)}\]
a = b, b = c, d = e, b = s, d = t, a ≠ e

Model construction

|M| = \{♦₁, ♦₂\} (universe, aka domain)
M(a) = ♦₁ (assignment)

Alternative notation:
\(a^M = ♦_1\)
Deciding Equality

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq e \]

Model construction

\[ |M| = \{ \diamondsuit_1, \diamondsuit_2 \} \] (universe, aka domain)

\[ M(a) = M(b) = M(c) = M(s) = \diamondsuit_1 \]

\[ M(d) = M(e) = M(t) = \diamondsuit_2 \]
Deciding Equality: Termination, Soundness, Completeness

- Termination: easy

- Soundness
  - Invariant: all constants in a “ball” are known to be equal.
  - The “ball” merge operation is justified by:
    - Transitivity and Symmetry rules.

- Completeness
  - We can build a model if an inconsistency was not detected.
  - Proof template (by contradiction):
    - Build a candidate model.
    - Assume a literal was not satisfied.
    - Find contradiction.
Completeness

We can build a model if an inconsistency was not detected.

Instantiating the template for our procedure:

Assume some literal $c = d$ is not satisfied by our model.

That is, $M(c) \neq M(d)$.

This is impossible, $c$ and $d$ must be in the same “ball”.

$$M(c) = M(d) = \diamond_i$$
Completeness

We can build a model if an inconsistency was not detected.

Instantiating the template for our procedure:
- Assume some literal $c \neq d$ is not satisfied by our model.
- That is, $M(c) = M(d)$.
- Key property: we only check the disequalities after we processed all equalities.
- This is impossible, $c$ and $d$ must be in the different “balls”

\[ M(c) = \Diamond_i \]
\[ M(d) = \Diamond_j \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ f(a, g(d)) \neq f(b, g(e)) \]

Congruence Rule:

\[ x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ f(a, g(d)) \neq f(b, g(e)) \]

First Step: “Naming” subterms

Congruence Rule:

\[ x_1 = y_1, \ldots, x_n = y_n \] implies \[ f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ f(a, v_1) \neq f(b, g(e)) \]

\[ v_1 \equiv g(d) \]

First Step: “Naming” subterms

Congruence Rule:

\[ x_1 = y_1, \ldots, \ x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ f(a, v_1) \neq f(b, g(e)) \]

\[ v_1 \equiv g(d) \]

First Step: “Naming” subterms

Congruence Rule:

\[ x_1 = y_1, ..., x_n = y_n \implies f(x_1, ..., x_n) = f(y_1, ..., y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ f(a, v_1) \neq f(b, v_2) \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e) \]

First Step: “Naming” subterms

Congruence Rule:
\[ x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ f(a, v_1) \neq f(b, v_2) \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e) \]

First Step: “Naming” subterms

Congruence Rule:
\[ x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ v_3 \neq f(b, v_2) \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1) \]

First Step: “Naming” subterms

Congruence Rule:
\[ x_1 = y_1, \ ..., \ x_n = y_n \ \text{implies} \ f(x_1, ..., x_n) = f(y_1, ..., y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ v_3 \neq f(b, v_2) \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1) \]

First Step: “Naming” subterms

Congruence Rule:

\[ x_1 = y_1, ..., x_n = y_n \text{ implies } f(x_1, ..., x_n) = f(y_1, ..., y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ v_3 \neq v_4 \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

First Step: “Naming” subterms

Congruence Rule:
\[ x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ v_3 \neq v_4 \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

**Congruence Rule:**

\[ x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ v_3 \neq v_4 \]

\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

Congruence Rule:

\[ x_1 = y_1, ..., \ x_n = y_n \implies f(x_1, ..., x_n) = f(y_1, ..., y_n) \]

\[ d = e \implies g(d) = g(e) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, b = c, d = e, b = s, d = t, v_3 \neq v_4 \]

\[ v_1 \equiv g(d), v_2 \equiv g(e), v_3 \equiv f(a, v_1), v_4 \equiv f(b, v_2) \]

**Congruence Rule:**

\[ x_1 = y_1, ..., x_n = y_n \text{ implies } f(x_1, ..., x_n) = f(y_1, ..., y_n) \]

\[ d = e \text{ implies } v_1 = v_2 \]
We say: $v_1$ and $v_2$ are congruent.

To decide equality of values in uninterpreted functions, consider:

- $a = b$, $b = c$, $d = e$, $b = s$, $d = t$
- $v_1 \equiv g(d)$, $v_2 \equiv g(e)$, $v_3 \equiv f(a, v_1)$, $v_4 \equiv f(b, v_2)$

**Congruence Rule:**

$x_1 = y_1, ..., x_n = y_n$ implies $f(x_1, ..., x_n) = f(y_1, ..., y_n)$

$d = e$ implies $v_1 = v_2$
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ v_3 \neq v_4 \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

Congruence Rule:
\[ x_1 = y_1, \ldots, x_n = y_n \implies f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
\[ a = b, \ v_1 = v_2 \implies f(a, v_1) = f(b, v_2) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ v_3 \neq v_4 \]

\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

Congruence Rule:

\[ x_1 = y_1, \ ... , \ x_n = y_n \text{ implies } f(x_1, ..., x_n) = f(y_1, ..., y_n) \]

\[ a = b, \ v_1 = v_2 \text{ implies } v_3 = v_4 \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ v_3 \neq v_4 \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

Congruence Rule:
\[ x_1 = y_1, \ldots, x_n = y_n \implies f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
\[ a = b, \ v_1 = v_2 \implies v_3 = v_4 \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ v_3 \neq v_4 \]

\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

Congruence Rule:

\[ x_1 = y_1, \ldots, x_n = y_n \implies f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq v_4, \ v_2 \neq v_3 \]

\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

Changing the problem

Congruence Rule:

\[ x_1 = y_1, \ldots, \ x_n = y_n \implies f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq v_4, \ v_2 \neq v_3 \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

Congruence Rule:
\[ x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \]
a = b, b = c, d = e, b = s, d = t, a \neq v_4, v_2 \neq v_3

v_1 \equiv g(d), v_2 \equiv g(e), v_3 \equiv f(a, v_1), v_4 \equiv f(b, v_2)

**Congruence Rule:**

\[ x_1 = y_1, \ldots, x_n = y_n \] implies \( f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \)
Deciding Equality + (uninterpreted) Functions

\[
a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq v_4, \ v_2 \neq v_3 \\
v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2)
\]

Model construction:

\[
|M| = \{\diamondsuit_1, \diamondsuit_2, \diamondsuit_3, \diamondsuit_4\} \\
M(a) = M(b) = M(c) = M(s) = \diamondsuit_1 \\
M(d) = M(e) = M(t) = \diamondsuit_2 \\
M(v_1) = M(v_2) = \diamondsuit_3 \\
M(v_3) = M(v_4) = \diamondsuit_4
\]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq v_4, \ v_2 \neq v_3 \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1) , \ v_4 \equiv f(b, v_2) \]

Model construction:
\[ |M| = \{ \heartsuit_1, \heartsuit_2, \heartsuit_3, \heartsuit_4 \} \]
\[ M(a) = M(b) = M(c) = M(s) = \heartsuit_1 \]
\[ M(d) = M(e) = M(t) = \heartsuit_2 \]
\[ M(v_1) = M(v_2) = \heartsuit_3 \]
\[ M(v_3) = M(v_4) = \heartsuit_4 \]

Missing: Interpretation for \( f \) and \( g \).
Building the interpretation for function symbols

- $M(g)$ is a mapping from $|M|$ to $|M|$
- Defined as:
  $$M(g)(\diamond_i) = \diamond_j \text{ if there is } v \equiv g(a) \text{ s.t.}$$
  $$M(a) = \diamond_i$$
  $$M(v) = \diamond_j$$
  $$= \diamond_k, \text{ otherwise } (\diamond_k \text{ is an arbitrary element})$$

- Is $M(g)$ well-defined?
Building the interpretation for function symbols

- $M(g)$ is a mapping from $|M|$ to $|M|$.
- Defined as:
  
  $M(g)(\diamond_i) = \diamond_j$ if there is $v \equiv g(a)$ s.t.
  
  $M(a) = \diamond_i$
  $M(v) = \diamond_j$
  
  $= \diamond_k$, otherwise ($\diamond_k$ is an arbitrary element).

Is $M(g)$ well-defined?

- Problem: we may have
  
  $v \equiv g(a)$ and $w \equiv g(b)$ s.t.
  
  $M(a) = M(b) = \diamond_1$ and $M(v) = \diamond_2 \neq \diamond_3 = M(w)$.
  
  So, is $M(g)(\diamond_1) = \diamond_2$ or $M(g)(\diamond_1) = \diamond_3$?
Building the interpretation for function symbols

- $M(g)$ is a mapping from $|M|$ to $|M|$.

Defined as:

$$M(g)(\diamond_i) = \diamond_j \text{ if there is } v \equiv g(a) \text{ s.t.}$$

- $M(a) = \diamond_i$
- $M(v) = \diamond_j$
- $= \diamond_k$, otherwise ($\diamond_k$ is an arbitrary element)

Is $M(g)$ well-defined?

Problem: we may have

- $v \equiv g(a)$ and $w \equiv g(b)$ s.t.
- $M(a) = M(b) = \diamond_1$ and $M(v) = \diamond_2 \neq \diamond_3 = M(w)$

So, is $M(g)(\diamond_1) = \diamond_2$ or $M(g)(\diamond_1) = \diamond_3$?

This is impossible because of the congruence rule!

$a$ and $b$ are in the same “ball”, then so are $v$ and $w$.
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq v_4, \ v_2 \neq v_3 \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

Model construction:

\[ |M| = \{ \diamond_1, \diamond_2, \diamond_3, \diamond_4 \} \]
\[ M(a) = M(b) = M(c) = M(s) = \diamond_1 \]
\[ M(d) = M(e) = M(t) = \diamond_2 \]
\[ M(v_1) = M(v_2) = \diamond_3 \]
\[ M(v_3) = M(v_4) = \diamond_4 \]
a = b, b = c, d = e, b = s, d = t, a \neq v_4, v_2 \neq v_3

v_1 \equiv g(d), v_2 \equiv g(e), v_3 \equiv f(a, v_1), v_4 \equiv f(b, v_2)

Model construction:

|M| = \{\bullet_1, \bullet_2, \bullet_3, \bullet_4\}

M(a) = M(b) = M(c) = M(s) = \bullet_1

M(d) = M(e) = M(t) = \bullet_2

M(v_1) = M(v_2) = \bullet_3

M(v_3) = M(v_4) = \bullet_4

M(g)(\bullet_i) = \bullet_j \text{ if there is } v \equiv g(a) \text{ s.t.}

M(a) = \bullet_i

M(v) = \bullet_j

= \bullet_k, \text{ otherwise}
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq v_4, \ v_2 \neq v_3 \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

Model construction:

\[ |M| = \{ \Diamond_1, \Diamond_2, \Diamond_3, \Diamond_4 \} \]
\[ M(a) = M(b) = M(c) = M(s) = \Diamond_1 \]
\[ M(d) = M(e) = M(t) = \Diamond_2 \]
\[ M(v_1) = M(v_2) = \Diamond_3 \]
\[ M(v_3) = M(v_4) = \Diamond_4 \]
\[ M(g) = \{ \Diamond_2 \rightarrow \Diamond_3 \} \]

\[ M(g)(\Diamond_i) = \Diamond_j \] if there is \( v \equiv g(a) \) s.t.
\[ M(a) = \Diamond_i \]
\[ M(v) = \Diamond_j \]
\[ = \Diamond_k, \] otherwise
Deciding Equality + (uninterpreted) Functions

\[ a = b, \quad b = c, \quad d = e, \quad b = s, \quad d = t, \quad a \neq v_4, \quad v_2 \neq v_3 \]

\[ v_1 \equiv g(d), \quad v_2 \equiv g(e), \quad v_3 \equiv f(a, v_1), \quad v_4 \equiv f(b, v_2) \]

Model construction:

\[ |M| = \{ \diamond_1, \diamond_2, \diamond_3, \diamond_4 \} \]

\[ M(a) = M(b) = M(c) = M(s) = \diamond_1 \]

\[ M(d) = M(e) = M(t) = \diamond_2 \]

\[ M(v_1) = M(v_2) = \diamond_3 \]

\[ M(v_3) = M(v_4) = \diamond_4 \]

\[ M(g) = \{ \diamond_2 \rightarrow \diamond_3 \} \]

\[ M(g)(\diamond_i) = \diamond_j \text{ if there is } v \equiv g(a) \text{ s.t.} \]

\[ M(a) = \diamond_i \]

\[ M(v) = \diamond_j \]

\[ = \diamond_k, \text{ otherwise} \]
Deciding Equality + (uninterpreted) Functions

\[ a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq v_4, \ v_2 \neq v_3 \]
\[ v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2) \]

Model construction:
\[ |M| = \{ \blacklozenge_1, \blacklozenge_2, \blacklozenge_3, \blacklozenge_4 \} \]
\[ M(a) = M(b) = M(c) = M(s) = \blacklozenge_1 \]
\[ M(d) = M(e) = M(t) = \blacklozenge_2 \]
\[ M(v_1) = M(v_2) = \blacklozenge_3 \]
\[ M(v_3) = M(v_4) = \blacklozenge_4 \]
\[ M(g) = \{ \blacklozenge_2 \rightarrow \blacklozenge_3, \text{else } \rightarrow \blacklozenge_1 \} \]
\[ M(g)(\blacklozenge_i) = \blacklozenge_j \text{ if there is } v \equiv g(a) \text{ s.t.} \]
\[ M(a) = \blacklozenge_i \]
\[ M(v) = \blacklozenge_j \]
\[ = \blacklozenge_k, \text{ otherwise} \]
Deciding Equality + (uninterpreted) Functions

\[
a = b, \ b = c, \ d = e, \ b = s, \ d = t, \ a \neq v_4, \ v_2 \neq v_3 \\
v_1 \equiv g(d), \ v_2 \equiv g(e), \ v_3 \equiv f(a, v_1), \ v_4 \equiv f(b, v_2)
\]

Model construction:

\[
|M| = \{\bullet_1, \bullet_2, \bullet_3, \bullet_4\}
\]

\[
M(a) = M(b) = M(c) = M(s) = \bullet_1 \\
M(d) = M(e) = M(t) = \bullet_2 \\
M(v_1) = M(v_2) = \bullet_3 \\
M(v_3) = M(v_4) = \bullet_4
\]

\[
M(g)(\bullet_i) = \bullet_j \text{ if there is } v \equiv g(a) \text{ s.t.} \\
M(a) = \bullet_i \text{, } M(v) = \bullet_j \\
= \bullet_k, \text{ otherwise}
\]

\[
M(f) = \{ (\bullet_1, \bullet_3) \rightarrow \bullet_4, \text{ else } \rightarrow \bullet_1 \}
\]
Deciding Equality + (uninterpreted) Functions

What about predicates?

\[ p(a, b), \neg p(c, b) \]
What about predicates?

\[ p(a, b), \neg p(c, b) \]

\[ f_p(a, b) = T, \quad f_p(c, b) \neq T \]
It is possible to eliminate function symbols using a method called **Ackermannization**.

\[
\begin{align*}
a &= b, & b &= c, & d &= e, & b &= s, & d &= t, & a \neq v_4, & v_2 \neq v_3 \\
v_1 &\equiv g(d), & v_2 &\equiv g(e), & v_3 &\equiv f(a, v_1), & v_4 &\equiv f(b, v_2) \\
\end{align*}
\]

\[
\begin{align*}
a &= b, & b &= c, & d &= e, & b &= s, & d &= t, & a \neq v_4, & v_2 \neq v_3 \\
d &\neq e \lor v_1 = v_2, & a \neq v_1 \lor b \neq v_2 \lor v_3 = v_4
\end{align*}
\]
It is possible to eliminate function symbols using a method called **Ackermannization**.

\[
\begin{align*}
a &= b, & b &= c, & d &= e, & b &= s, & d &= t, & a \neq v_4, & v_2 \neq v_3 \\
v_1 &= g(d), & v_2 &= g(e), & v_3 &= f(a, v_1), & v_4 &= f(b, v_2)
\end{align*}
\]

\[
\begin{align*}
a &= b, & b &= c, & d &= e, & b &= s, & d &= t, & a \neq v_4, & v_2 \neq v_3 \\
d &= e & v_1 &= v_2, \\
a &\neq v_1 & b &\neq v_2 & v_3 = v_4
\end{align*}
\]

Main Problem: quadratic blowup
It is possible to implement our procedure in \( O(n \log n) \)
Deciding Equality +
(uninterpreted) Functions

Sets (equivalence classes)

\[ d,e,t \]
\[ d,e \cup t = d,e,t \]

Union

\[ a,b,c,s \]
\[ a \neq s \]

Membership
Deciding Equality + (uninterpreted) Functions

Sets (equivalence classes)

Key observation: The sets are disjoint!

Union

Membership

a \neq s
Deciding Equality + (uninterpreted) Functions

Union-Find data-structure

Every set (equivalence class) has a root element (representative).

We say: find[c] is b
Deciding Equality + (uninterpreted) Functions

Union-Find data-structure

\[
\begin{array}{c}
\text{a,b,c} \\
\text{b} \\
\text{a} \\
\text{c} \\
\text{s,r} \\
\text{s} \\
\text{r} \\
\text{a,b,c,s,r} \\
\end{array}
\]

\[
\bigcup
\]

=
Deciding Equality + (uninterpreted) Functions

Tracking the equivalence classes size is important!

\[ a_1 \rightarrow a_2 \cup a_3 = a_1 \rightarrow a_2 \rightarrow a_3 \]

\[ a_1 \rightarrow a_2 \rightarrow a_3 \cup a_4 = a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \]

\[ \ldots \]

\[ a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \ldots \rightarrow a_{n-1} \cup a_n = \]

\[ a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \ldots \rightarrow a_{n-1} \rightarrow a_n \]
Deciding Equality + (uninterpreted) Functions

Tracking the equivalence classes size is important!

\[ a_1 \rightarrow a_2 \cup a_3 = a_1 \rightarrow a_2 \leftarrow a_3 \]

\[ a_1 \rightarrow a_2 \leftarrow a_3 \cup a_4 = a_1 \rightarrow a_2 \leftarrow a_3 \]

... 

\[ a_2 \quad \cup \quad a_n = a_2 \leftarrow a_n \]

\[ a_1 \quad a_3 \quad a_{n-1} \]

\[ a_1 \quad a_3 \quad a_{n-1} \]
Deciding Equality + (uninterpreted) Functions

Tracking the equivalence classes size is important!

\[ a_1 \rightarrow a_2 \cup a_3 = a_1 \rightarrow a_2 \leftarrow a_3 \]

\[ a_1 \rightarrow a_2 \leftarrow a_3 \cup a_4 = a_1 \rightarrow a_2 \leftarrow a_3 \]

\[ \ldots \]

\[ a_2 \leftarrow a_4 \]

\[ a_1 \rightarrow a_2 \leftarrow a_3 \cup a_n = a_1 \rightarrow a_2 \leftarrow a_n \]

Each constant has two fields: find and size.
Implementing the congruence rule.

Occurrences of a constant: we say a occurs in v iff $v \equiv f(\ldots, a, \ldots)$

When we “merge” two equivalence classes we can traverse these occurrences to find new congruences.

\[
\begin{align*}
\text{occurrences}[b] &= \{ v_1 \equiv g(b), v_2 \equiv f(a) \} \\
\text{occurrences}[s] &= \{ v_3 \equiv f(r) \}
\end{align*}
\]
Implementing the congruence rule.

Occurrences of a constant: we say \( a \) occurs in \( v \) iff \( v \equiv f(\ldots,a,\ldots) \)

When we “merge” two equivalence classes we can traverse these occurrences to find new congruences.

Inefficient version:
for each \( v \) in \( \text{occurrences}(b) \)
for each \( w \) in \( \text{occurrences}(s) \)
if \( v \) and \( w \) are congruent
add \( (v,w) \) to \( \text{todo queue} \)

\[
\text{occurrences}(b) = \{ v_1 \equiv g(b), \ v_2 \equiv f(a) \}
\]
\[
\text{occurrences}(s) = \{ v_3 \equiv f(r) \}
\]
Deciding Equality + (uninterpreted) Functions

\[
\begin{align*}
\text{occurrences}[b] &= \{ v_1 \equiv g(b), v_2 \equiv f(a) \} \\
\text{occurrences}[s] &= \{ v_3 \equiv f(r) \}
\end{align*}
\]

We also need to merge \text{occurrences}[b] with \text{occurrences}[s]. This can be done in constant time:
Use circular lists to represent the occurrences. (More later)
Avoiding the nested loop:
for each \( v \) in occurrences[b]
  for each \( w \) in occurrences[s]
    ...

Use a hash table to store the elements \( v_1 \equiv f(a_1, \ldots, a_n) \).
Each constant has an identifier (e.g., natural number).
Compute hash code using the identifier of the (equivalence class) roots of the arguments.

\[
hash(v_1) = hash\text{-}tuple(id(f), id(root(a_1)), \ldots, id(root(a_n)))
\]
Avoiding the nested loop:
for each $v$ in occurrences(b)
    for each $w$ in occurrences(s)
        ...

Use a hash table to store the elements $v_1 \equiv f(a_1, ..., a_n)$.
Each constant has an identifier (e.g., natural number).
Compute hash code using the identifier of the (equivalence class) roots of the arguments.

$\text{hash}(v_1) = \text{hash-tuple}(\text{id}(f), \text{id}(\text{root}(a_1)), ..., \text{id}(\text{root}(a_n)))$

hash-tuple can be the Jenkin’s hash function for strings. Just adding the ids produces a very bad hash-code!
Efficient implementation of the congruence rule.
Merging the equivalences classes with roots: $a_1$ and $a_2$
Assume $a_2$ is smaller than $a_1$

Before merging the equivalence classes: $a_1$ and $a_2$
for each $v$ in occurrences[$a_2$]
  remove $v$ from the hash table  (its hashcode will change)

After merging the equivalence classes: $a_1$ and $a_2$
for each $v$ in occurrences[$a_2$]
  if there is $w$ congruent to $v$ in the hash-table
    add ($v,w$) to todo queue
  else add $v$ to hash-table
Deciding Equality + (uninterpreted) Functions

Efficient implementation of the congruence rule.

Merging the equivalences classes with roots $a_1$ and $a_2$

Assume $a_2$ is smaller than $a_1$

Before merging the equivalence classes: $a_1$ and $a_2$

for each $v$ in $\text{occurrences}[a_2]$

remove $v$ from the hash table (its hashcode will change)

After merging the equivalence classes: $a_1$ and $a_2$

for each $v$ in $\text{occurrences}[a_2]$

if there is $w$ congruent to $v$ in the hash-table

add $(v,w)$ to todo queue

else add $v$ to hash-table

add $v$ to occurrences($a_1$)
Deciding Equality + (uninterpreted) Functions

The efficient version is not optimal (in theory).
Problem: we may have \( v \equiv f(a_1, \ldots, a_n) \) with “huge” \( n \).

Solution: currying

Use only binary functions, and represent \( f(a_1, a_2, a_3, a_4) \) as \( f(a_1, h(a_2, h(a_3, a_4))) \)

This is not necessary in practice, since the \( n \) above is small.
Each constant has now three fields: 
find, size, and occurrences.

We also has use a hash-table for implementing the congruence rule.

We will need many more improvements!
Many verification/analysis problems require:

\[ \text{case-analysis} \quad \]

\[ x \geq 0, \quad y = x + 1, \quad (y > 2 \lor y < 1) \]
Many verification/analysis problems require: case-analysis

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Naïve Solution: Convert to DNF

\[(x \geq 0, \ y = x + 1, \ y > 2) \lor (x \geq 0, \ y = x + 1, \ y < 1)\]
Many verification/analysis problems require:

\[
x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Naïve Solution: Convert to DNF

\[(x \geq 0, \ y = x + 1, \ y > 2) \lor (x \geq 0, \ y = x + 1, \ y < 1)\]

Too Inefficient! (exponential blowup)
SMT : Basic Architecture

SAT + Theory Solvers = SMT

- Equality + UF
- Arithmetic
- Bit-vectors
- ...

Case Analysis
DPLL

Partial model

Set of clauses

M | F
Guessing

\[ p \; | \; p \lor q, \neg q \lor r \]

\[ p, \neg q \; | \; p \lor q, \neg q \lor r \]
Deducing

\[ p \mid p \lor q, \neg p \lor s \]

\[ p, s \mid p \lor q, \neg p \lor s \]
Backtracking

\[ p, \neg s, q \mid p \lor q, s \lor q, \neg p \lor \neg q \]

\[ p, s \mid p \lor q, s \lor q, \neg p \lor \neg q \]
Modern DPLL

- Efficient indexing (two-watch literal)
- Non-chronological backtracking (backjumping)
- Lemma learning
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, (p_3 \lor p_4) \]
\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, (p_3 \lor p_4) \quad p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

SAT Solver
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \]

\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

Assignment

\[ p_1, \ p_2, \ \neg p_3, \ p_4 \]
**Basic Idea**

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

**Abstract (aka “naming” atoms)**

\[ p_1, \ p_2, \ (p_3 \lor p_4) \]

\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

**Assignment**

\[ p_1, \ p_2, \ \neg p_3, \ p_4 \]

\[ x \geq 0, \ y = x + 1, \]
\[ \neg (y > 2), \ y < 1 \]
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \]

\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

SAT Solver

Assignment

\[ p_1, \ p_2, \ \neg p_3, \ p_4 \]

\[ x \geq 0, \ y = x + 1, \]
\[ \neg(y > 2), \ y < 1 \]

Theory Solver

Unsatisfiable

\[ x \geq 0, \ y = x + 1, \ y < 1 \]
Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \]

\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

SAT Solver

Assignment

\[ p_1, \ p_2, \ \neg p_3, \ p_4 \]

\[ x \geq 0, \ y = x + 1, \ \neg(y > 2), \ y < 1 \]

New Lemma

\[ \neg p_1 \lor \neg p_2 \lor \neg p_4 \]

Unsatisfiable

\[ x \geq 0, \ y = x + 1, \ y < 1 \]

Theory Solver
New Lemma
\neg p_1 \lor \neg p_2 \lor \neg p_4

 Unsatisfiable
x \geq 0, y = x + 1, y < 1

AKA Theory conflict

Theory Solver
procedure SmtSolver(F)
  
  \((F_p, M) := \text{Abstract}(F)\)

  \(\text{loop}\)

  \((R, A) := \text{SAT\_solver}(F_p)\)

  \(\text{if } R = \text{UNSAT then return } \text{UNSAT}\)

  \(S := \text{Concretize}(A, M)\)

  \((R, S') := \text{Theory\_solver}(S)\)

  \(\text{if } R = \text{SAT then return } \text{SAT}\)

  \(L := \text{New\_Lemma}(S', M)\)

  \(\text{Add } L \text{ to } F_p\)
SAT + Theory solvers

Basic Idea

\[ \mathbf{F}: x \geq 0, \; y = x + 1, \; (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ \mathbf{F_p}: \neg p_1, \; p_2, \; (p_3 \lor p_4) \]

\[ \mathbf{M}: \neg p_1 \equiv (x \geq 0), \; p_2 \equiv (y = x + 1), \; p_3 \equiv (y > 2), \; p_4 \equiv (y < 1) \]

SAT Solver

\[ \mathbf{A}: \text{Assignment} \]

\[ p_1, \; p_2, \; \neg p_3, \; p_4 \]

\[ \mathbf{S}: x \geq 0, \; y = x + 1, \; \neg (y > 2), \; y < 1 \]

Theory Solver

\[ \mathbf{L}: \text{New Lemma} \]

\[ \neg p_1 \lor \neg p_2 \lor \neg p_4 \]

\[ \mathbf{S'}: \text{Unsatisfiable} \]

\[ x \geq 0, \; y = x + 1, \; y < 1 \]
\( F: x \geq 0, y = x + 1, (y > 2 \lor y < 1) \)

Abstract (aka “naming” atoms)

\( F_p: p_1, p_2, (p_3 \lor p_4) \)

\( M: p_1 \equiv (x \geq 0), p_2 \equiv (y = x + 1), p_3 \equiv (y > 2), p_4 \equiv (y < 1) \)

SAT Solver

A: Assignment
\( p_1, p_2, \neg p_3, p_4 \)

S: \( x \geq 0, y = x + 1, \neg(y > 2), y < 1 \)

Theory Solver

L: New Lemma
\( \neg p_1 \lor \neg p_2 \lor \neg p_4 \)

S': Unsatisfiable
\( x \geq 0, y = x + 1, y < 1 \)

procedure SMT_Solver\( (F) \)
\( (F_p, M) := \text{Abstract}(F) \)
loop
\( (R, A) := \text{SAT}\_\text{solver}(F_p) \)
if \( R = \text{UNSAT} \) then return UNSAT
S = Concretize\( (A, M) \)
(\( R, S' \)) := Theory\_solver\( (S) \)
if \( R = \text{SAT} \) then return SAT
L := New\_Lemma\( (S, M) \)
Add \( L \) to \( F_p \)

“Lazy translation” to DNF
State-of-the-art SMT solvers implement many improvements.
Incrementality
Send the literals to the Theory solver as they are assigned by the SAT solver

\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1), \ p_5 \equiv (x < 2), \]
\[ p_1, \ p_2, \ p_4 \ | \ p_1, \ p_2, (p_3 \lor p_4), (p_5 \lor \neg p_4) \]

Partial assignment is already Theory inconsistent.
Efficient Backtracking

We don’t want to restart from scratch after each backtracking operation.
Efficient Lemma Generation (computing a small $S'$)

Avoid lemmas containing redundant literals.

\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1), \ p_5 \equiv (x < 2), \]
\[ p_1, \ p_2, \ p_3, \ p_4 \ \text{||} \ p_1, \ p_2, (p_3 \lor p_4), (p_5 \lor \neg p_4) \]

\[ \neg p_1 \lor \neg p_2 \lor \neg p_3 \lor \neg p_4 \]

Imprecise Lemma
Theory Propagation

It is the SMT equivalent of unit propagation.

\[ p_1 \equiv (x \geq 0), \quad p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \quad p_4 \equiv (y < 1), \quad p_5 \equiv (x < 2), \]
\[ p_1, \ p_2 \mid p_1, \ p_2, \ (p_3 \lor p_4), \ (p_5 \lor \lnot p_4) \]

\[ p_1, \ p_2 \text{ imply } \lnot p_4 \text{ by theory propagation} \]

\[ p_1, \ p_2, \ \lnot p_4 \mid p_1, \ p_2, \ (p_3 \lor p_4), \ (p_5 \lor \lnot p_4) \]
Theory Propagation
It is the SMT equivalent of unit propagation.

\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1), \ p_5 \equiv (x < 2), \]
\[ p_1, \ p_2 \models p_1, \ p_2, (p_3 \lor p_4), (p_5 \lor \neg p_4) \]

\[ p_1, \ p_2 \implies \neg p_4 \] by theory propagation

\[ p_1, \ p_2 , \neg p_4 \models p_1, \ p_2, (p_3 \lor p_4), (p_5 \lor \neg p_4) \]

Tradeoff between precision \( \times \) performance.
An Architecture: the core

Core

Arithmetic
Bit-Vectors
Scalar Values

Equality
Uninterpreted Functions

SAT Solver
An Architecture: the core

Core

- Arithmetic
- Bit-Vectors
- Scalar Values

Equality
Uninterpreted
Functions

SAT Solver

Case Analysis
An Architecture: the core

Core

- Arithmetic
- Bit-Vectors
- Scalar Values

Equality
Uninterpreted Functions

SAT Solver

Blackboard: equalities, disequalities, predicates
Deciding Equality + (uninterpreted) Functions

**Problem**: our procedure for Equality + UF does not support:

- Incrementality
- Efficient Backtracking
- Theory Propagation
- Lemma Learning
Incrementality (main problem):

We were processing the disequalities after we processed all equalities.

\[ p_1 \equiv a = b, \quad p_2 \equiv b = c, \]
\[ p_3 \equiv d = e, \quad p_4 \equiv a = c \]

\[ p_1, \neg p_4, p_2 \mid p_1, p_3 \lor \neg p_4, p_2 \lor p_4 \]

\[ a = b, \quad a \neq c, \quad b = c, \]
Incrementality (main problem):

We were processing the disequalities after we processed all equalities.

\[ p_1 \equiv a = b, \quad p_2 \equiv b = c, \]
\[ p_3 \equiv d = e, \quad p_4 \equiv a = c \]

\[ p_1, \neg p_4, p_2 \mid p_1, p_3 \lor \neg p_4, p_2 \lor p_4 \]

\[ a = b, \quad a \neq c, \quad b = c, \]
Incrementality

Store the disequalities of a constant.

Very similar to the structure occurrences.

\[
a = b, \ a \neq c
\]

\[
diseqs[b] = \{ a \neq c \}
diseqs[c] = \{ a \neq c \}
\]
Incrementality

Store the disequalities of a constant.

Very similar to the structure occurrences.

\[ a = b, \ a \neq c \]

\[
\begin{align*}
\text{diseqs}[b] &= \{ \ a \neq c \ } \\
\text{diseqs}[c] &= \{ \ a \neq c \ }
\end{align*}
\]

When we merge two equivalence classes, we must merge the sets diseqs. (circular lists again!)
**Incrementality**

Store the disequalities of a constant.

Very similar to the structure occurrences.

\[ a = b, a \neq c \]

\[ \text{diseqs}(b) = \{ a \neq c \} \]

\[ \text{diseqs}(c) = \{ a \neq c \} \]

When we merge two equivalence classes, we must merge the sets diseqs. (circular lists again!)

Before merging two equivalence classes, traverse one (the smallest) set of diseqs. (track the size of diseqs!)
Deciding Equality + (uninterpreted) Functions

Backtracking

Option 1: functional data-structures (too slow).
Option 2: trail stack (aka undo stack, fine grain backtracking)
    Associate an undo operation to each update operation.
    “Log” all update operations in a stack.
    During backtracking execute the associated undo operations.
Backtracking

We can do better: coarse grain backtracking.

Minimize the size of the undo stack.

Do not track each small update, but a big operation (merge).
Deciding Equality + (uninterpreted) Functions

Backtracking

We can do better: coarse grain backtracking.
Minimize the size of the undo stack.
Do not track each small update, but a big operation (merge).

Let us change the union-find data-structure a little bit.

Before:

```
   s
  /|
 b /|
| a c |
 r
```

**Fields:** find, size

After:

```
   s
  /|
 b /|
| a -- c -- r |
 r
```

**Fields:** root, next, size
Deciding Equality + (uninterpreted) Functions

Backtracking

We can do better: coarse grain backtracking.

Minimize the size of the undo stack.

Do not track each small update, but a big operation (merge).

Let us change the union-find data structure a little bit.

New design possibility:

We do not need to merge occurrences and diseqs.

We can access all occurrences and diseqs by traversing the next fields.

Before:

```
  s
 /\  
|  \ 
|   \ 
|    \ 
a    b    r
```

Fields: find, size

After:

```
  s
 /\  
|  \ 
|   \ 
|    \ 
next element
```

```
  a ← b ← c ← r
```

Fields: root, next, size
Deciding Equality + (uninterpreted) Functions

New union-find:
Deciding Equality + (uninterpreted) Functions

New union-find:

What was updated?
root[s], root[r],
next[b], next[s],
size[b]
Deciding Equality + (uninterpreted) Functions

New union-find:

We only need to store s in the undo stack!

What was updated?
root[c], root[r], next[b], next[s], size[b]
Deciding Equality + (uninterpreted) Functions

What about the congruence table?

hash table used to implement the congruence rule.

Let us use an additional field \( cg \).

It is only relevant for subterms: \( v_3 \equiv f(a, v_1) \)

Invariant: a constant (e.g., \( v_3 \)) is in the table iff \( cg[v_3] = v_3 \)

Otherwise, \( cg[v_3] \) contains the subterm congruent to \( v_3 \)

Example:
\( v_3 \equiv f(a, v_1), v_4 \equiv f(b, v_2) \)

Assume \( v_3 \) and \( v_4 \) are congruent (i.e., \( a = b \) and \( v_1 = v_2 \))

Moreover, \( v_3 \) is in the congruence table.

Then: \( cg[v_4] = v_3 \) and \( cg[v_3] = v_3 \)
Deciding Equality + (uninterpreted) Functions

procedure Merge(a, b)
    ar := root[a]; br := root[b]
    if ar = br then return
    if not CheckDiseqs(ar, br) then return
    if size[a] < size[b] then swap a, b; swap ar, br
    AddToTrailStack(MERGE, br)
    RemoveParentsFromHashTable(br)
    c := br
    do
        root[c] := ar
        c := next[c]
    while c ≠ br
    ReinsertParentsToHashTable(br)
    swap next[ar], next[br]
    size[ar] := size[ar] + size[br]
Deciding Equality + (uninterpreted) Functions

procedure UndoMerge(b_r)
    a_r := root[b_r]
    size[a_r] := size[a_r] - size[b_r]
    swap next[a_r], next[b_r]
RemoveParentsFromHashTable(b_r)
c := b_r
do
    root[c] := b_r
    c := next[c]
while c ≠ b_r
for each parent p of b_r
    if p = cg[p] or not congruent(p, cg[p])
        add p to hash table
        cg[p] := p
Deciding Equality + (uninterpreted) Functions

procedure UndoMerge(b_r)
    a_r := root[b_r]
    size[a_r] := size[a_r] – size[b_r]
    swap next[a_r], next[b_r]
    RemoveParentsFromHashTable(b_r)
    c := b_r
do
    root[c] := b_r
c := next[c]
while c ≠ b_r
    for each parent p of b_r
        if p = cg[p] or not congruent(p, cg[p])
            add p to hash table
        cg[p] := p
p was in the hash table before and after the merge.
Propagating equalities (and disequalities)

Store the atom occurrences of a constant.

\[ p_1 \equiv a = b, \ p_2 \equiv b = c, \]
\[ p_3 \equiv d = e, \ p_4 \equiv a = c \]

\[
\begin{align*}
\text{atom}	extunderscore\text{occs}[a] &= \{ p_1, p_4 \} \\
\text{atom}	extunderscore\text{occs}[b] &= \{ p_1, p_2 \} \\
\text{atom}	extunderscore\text{occs}[c] &= \{ p_2, p_4 \} \\
\text{atom}	extunderscore\text{occs}[d] &= \{ p_3 \} \\
\text{atom}	extunderscore\text{occs}[e] &= \{ p_4 \}
\end{align*}
\]

When merging or adding new disequalities traverse these sets.
Deciding Equality + (uninterpreted) Functions

Propagating disequalities (hard case)

\[ v_1 \equiv f(a, b), \quad v_2 \equiv f(c, d) \]

Assume we know that

\[ v_1 \neq v_2 \]
\[ a = c \]

Then, \( b \neq d \)

More about that later.
Efficient Lemma Generation (computing a small $S'$)

In EUF (equality + UF) a minimal unsatisfiable set is composed on:

- $n$ equalities
- 1 disequality

It is easy to find the disequality $a \neq b$.

So, our problem consists in finding the minimal set of equalities that implies $a = b$. 
Deciding Equality + (uninterpreted) Functions

Efficient Lemma Generation (computing a small $S'$)

First idea:

If $a = b$ is implied by a set of equalities, then $a$ and $b$ are in the same equivalence class.

Store all equalities used to “create” the equivalence class.

\[
p_1 \equiv (a = c), \ p_2 \equiv (b = c), \ p_3 \equiv (s = r), \ p_4 \equiv (c = r)
\]

$p_1, p_2, p_3, p_4, \ldots | \ldots$

Too imprecise for justifying $a = b$. We need only $p_1, p_2$.

The equivalence class was “created” using $p_1, p_2, p_3, p_4$. 
Efficient Lemma Generation (computing a small $S'$)

Second idea: Store a “proof tree”.

Each constant $c$ has a non-redundant “proof” for $c = \text{root}[c]$.

The proof is a path from $c$ to $\text{root}[c]$

$$p_1 \equiv (a = c), \quad p_2 \equiv (b = c),$$
$$p_3 \equiv (s = r), \quad p_4 \equiv (c = r)$$
procedure Merge(a, b, p_i)
    a_r := root[a]; b_r := root[b]
    if a_r = b_r then return
    if not CheckDiseqs(a_r, b_r) then return
    if size[a] < size[b] then swap a, b; swap a_r, b_r
    InvertPathFrom(b, b_r); AddProofEdge(b, a, p_i)
    AddToTrailStack(MERGE, b_r, b)
...

Deciding Equality + (uninterpreted) Functions
Deciding Equality + (uninterpreted) Functions

Non redundant proof for \( a = b \)

\( p_1, \ldots, p_n, q_1, \ldots, q_m \)

Common ancestor in the proof tree.
Extract a non redundant proof for $a = r$, $a = b$ and $a = s$. 
What about congruence?

New form of justification for an edge in the “proof tree”.

\[ v_1 \equiv f(b), \ v_2 \equiv f(c) \]
What about congruence?

New form of justification for an edge in the “proof tree”.

\[ v_1 \equiv f(b), \ v_2 \equiv f(c) \]

When computing the “proof” for \( a = v_2 \)

Recursive call for computing the proof for \( v_1 = v_2 \)

Result: \( \{p_1, p_2\} \)
The new algorithm may compute redundant proofs for EUF.

Using notation $a = b$ for $p \equiv a = b$, and $p$ assigned by SAT solver

\[
\begin{align*}
  f_1(a_1) &= a_1 = a_2 = f_1(a_5) \\
  f_2(a_1) &= a_2 = a_3 = f_2(a_5) \\
  f_3(a_1) &= a_3 = a_4 = f_3(a_5) \\
  f_4(a_1) &= a_4 = a_5 = f_4(a_5)
\end{align*}
\]
The new algorithm may compute redundant proofs for EUF.

Using notation $a = b$ for $p \equiv a = b$, and $p$ assigned by SAT solver

\[
\begin{align*}
  f_1(a_1) &= a_1 = a_2 = f_1(a_5) \\
  f_2(a_1) &= a_2 = a_3 = f_2(a_5) \\
  f_3(a_1) &= a_3 = a_4 = f_3(a_5) \\
  f_4(a_1) &= a_4 = a_5 = f_4(a_5)
\end{align*}
\]

Two non redundant proofs $f_2(a_1) = f_2(a_5)$:
- $\{p_2, q_2, s_2\}$ using transitivity
- $\{q_1, q_2, q_3, q_4\}$ using congruence $a_1 = a_5$

Similar for $f_1, f_3, f_4$. 
Deciding Equality + (uninterpreted) Functions

The new algorithm may compute redundant proofs for EUF.

Using notation \( a = b \) for \( p \equiv a = b \), and \( p \) assigned by SAT solver

\[
\begin{align*}
    f_1(a_1) &= a_1 = a_2 = f_1(a_5) \\
    f_2(a_1) &= a_2 = a_3 = f_2(a_5) \\
    f_3(a_1) &= a_3 = a_4 = f_3(a_5) \\
    f_4(a_1) &= a_4 = a_5 = f_4(a_5)
\end{align*}
\]

Two non redundant proofs \( f_2(a_1) = f_2(a_5) \):

\[
\begin{align*}
    \{ p_2, q_2, s_2 \} & \text{ using transitivity} \\
    \{ q_1, q_2, q_3, q_4 \} & \text{ using congruence } a_1 = a_5
\end{align*}
\]

Similar for \( f_1, f_3, f_4 \).

So there are 16 proofs for

\[
g(f_1(a_1), f_2(a_1), f_3(a_1), f_4(a_1)) = g(f_1(a_5), f_2(a_5), f_3(a_5), f_4(a_5))
\]

The only non redundant is \( \{ q_1, q_2, q_3, q_4 \} \)
Some benchmarks are very hard for our procedure.

\[
p_1 \lor a_1 = c_0, \neg p_1 \lor a_1 = c_1, \quad p_1 \lor b_1 = c_0, \neg p_1 \lor b_1 = c_1, \\
p_2 \lor a_2 = c_0, \neg p_2 \lor a_2 = c_1, \quad p_2 \lor b_2 = c_0, \neg p_2 \lor b_2 = c_1, \\
\ldots, \\
p_n \lor a_n = c_0, \neg p_n \lor a_n = c_1, \quad p_n \lor b_n = c_0, \neg p_n \lor b_n = c_1, \\
f(a_n, \ldots, f(a_2, a_1)\ldots) \neq f(b_n, \ldots, f(b_2, b_1)\ldots) 
\]
Some benchmarks are very hard for our procedure.

\[ p_1 \lor a_1 = c_0, \neg p_1 \lor a_1 = c_1, \quad p_1 \lor b_1 = c_0, \neg p_1 \lor b_1 = c_1, \]
\[ p_2 \lor a_2 = c_0, \neg p_2 \lor a_2 = c_1, \quad p_2 \lor b_2 = c_0, \neg p_2 \lor b_2 = c_1, \]
\[ \ldots, \]
\[ p_n \lor a_n = c_0, \neg p_n \lor a_n = c_1, \quad p_n \lor b_n = c_0, \neg p_n \lor b_n = c_1, \]
\[ f(a_n, \ldots, f(a_2, a_1)\ldots) \neq f(b_n, \ldots, f(b_2, b_1)\ldots) \]

Lemmas learned during the search are not useful.
They only use atoms that are already in the problem!
Deciding Equality + (uninterpreted) Functions

Some benchmarks are very hard for our procedure.

\[ p_1 \lor a_1 = c_0, \neg p_1 \lor a_1 = c_1, \quad p_1 \lor b_1 = c_0, \neg p_1 \lor b_1 = c_1, \]
\[ p_2 \lor a_2 = c_0, \neg p_2 \lor a_2 = c_1, \quad p_2 \lor b_2 = c_0, \neg p_2 \lor b_2 = c_1, \]
\[ \ldots, \]
\[ p_n \lor a_n = c_0, \neg p_n \lor a_n = c_1, \quad p_n \lor b_n = c_0, \neg p_n \lor b_n = c_1, \]
\[ f(a_n, \ldots, f(a_2, a_1)\ldots) \neq f(b_n, \ldots, f(b_2, b_1)\ldots) \]

Lemmas learned during the search are not useful. They only use atoms that are already in the problem! Solution: congruence rule suggests which new atoms must be created.
Deciding Equality + (uninterpreted) Functions

Some benchmarks are very hard for our procedure.

\[
\begin{align*}
    p_1 \lor a_1 &= c_0, & \neg p_1 \lor a_1 &= c_1, & p_1 \lor b_1 &= c_0, & \neg p_1 \lor b_1 &= c_1, \\
    p_2 \lor a_2 &= c_0, & \neg p_2 \lor a_2 &= c_1, & p_2 \lor b_2 &= c_0, & \neg p_2 \lor b_2 &= c_1, \\
    \ldots, \\
    p_n \lor a_n &= c_0, & \neg p_n \lor a_n &= c_1, & p_n \lor b_n &= c_0, & \neg p_n \lor b_n &= c_1, \\
    f(a_n, \ldots, f(a_2, a_1)\ldots) &\neq f(b_n, \ldots, f(b_2, b_1)\ldots)
\end{align*}
\]

Solution: congruence rule suggests which new atoms must be created.

Whenever, the congruence rules

\[
a_i = b_i, a_j = b_j \text{ implies } f(a_i, a_j) = f(b_i, b_j)
\]

is used to (immediately) deduce a conflict. Add the clause:

\[
a_i \neq b_i \lor a_j \neq b_j \lor f(a_i, a_j) = f(b_i, b_j)
\]
Solution: congruence rule suggests which new atoms must be created.

Whenever, the congruence rules
\[ a_i = b_i, \ a_j = b_j \] implies \( f(a_i, a_j) = f(b_i, b_j) \)
is used to (immediately) deduce a conflict. Add the clause:
\[ a_i \neq b_i \lor a_j \neq b_j \lor f(a_i, a_j) = f(b_i, b_j) \]

“Dynamic Ackermannization”
It allows the solver to perform the missing disequality propagation.
We can solve the QF_UF SMT-Lib benchmarks!
Linear Arithmetic

- Many approaches
  - Graph-based for difference logic: \( a - b \leq 3 \)
  - Fourier-Motzkin elimination:
    \[
    t_1 \leq ax, \ bx \leq t_2 \implies bt_1 \leq at_2
    \]
  - Standard Simplex
  - General Form Simplex
Very useful in practice!

Most arithmetical constraints in software verification/analysis are in this fragment.

\[ x := x + 1 \]
\[ x_1 = x_0 + 1 \]
\[ x_1 - x_0 \leq 1, x_0 - x_1 \leq -1 \]
### Job shop scheduling

<table>
<thead>
<tr>
<th>$d_{i,j}$</th>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

$\max = 8$

**Solution**

$t_{1,1} = 5$, $t_{1,2} = 7$, $t_{2,1} = 2$,
$t_{2,2} = 6$, $t_{3,1} = 0$, $t_{3,2} = 3$

**Encoding**

\[(t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land (t_{2,1} \geq 0) \land (t_{2,2} \geq t_{2,1} + 3) \land (t_{2,2} + 1 \leq 8) \land (t_{3,1} \geq 0) \land (t_{3,2} \geq t_{3,1} + 2) \land (t_{3,2} + 3 \leq 8) \land ((t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)) \land ((t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)) \land ((t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)) \land ((t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)) \land ((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)) \land ((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1))\]
Chasing negative cycles!

Algorithms based on Bellman-Ford ($O(mn)$).
Many solvers (e.g., ICS, Simplify) are based on the Standard Simplex.

\[
\begin{align*}
a - d + 2e &= 3 \\
b - d &= 1 \\
c + d - e &= -1 \\
a, b, c, d, e &\geq 0
\end{align*}
\]
Many solvers (e.g., ICS, Simplify) are based on the Standard Simplex.

\[
\begin{align*}
  a - d + 2e &= 3 \\
  b - d &= 1 \\
  c + d - e &= -1 \\
  a, b, c, d, e &\geq 0
\end{align*}
\]

\[
\begin{pmatrix}
  1 & 0 & 0 & -1 & 2 \\
  0 & 1 & 0 & -1 & 0 \\
  0 & 0 & 1 & 1 & -1
\end{pmatrix}
\begin{pmatrix}
  a \\
  b \\
  c \\
  d \\
  e
\end{pmatrix}
= 
\begin{pmatrix}
  3 \\
  1 \\
  -1
\end{pmatrix}
\]

\[Ax = b \text{ and } x \geq 0.\]
Many solvers (e.g., ICS, Simplify) are based on the Standard Simplex.

We say \( a, b, c \) are the basic (or dependent) variables.

\[
\begin{align*}
a - d + 2e &= 3 \\
b - d &= 1 \\
c + d - e &= -1 \\
a, b, c, d, e &\geq 0
\end{align*}
\]

\[
\begin{bmatrix}
1 & 0 & 0 & -1 & 2 \\
0 & 1 & 0 & -1 & 0 \\
0 & 0 & 1 & 1 & -1
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c \\
d \\
e
\end{bmatrix}
=
\begin{bmatrix}
3 \\
1 \\
-1
\end{bmatrix}
\]

\[Ax = b \text{ and } x \geq 0.\]
Many solvers (e.g., ICS, Simplify) are based on the Standard Simplex.

\[
\begin{align*}
a - d + 2e &= 3 \\
b - d &= 1 \\
c + d - e &= -1 \\
\end{align*}
\]
\[a, b, c, d, e \geq 0\]

We say \(a, b, c\) are the basic (or dependent) variables.

\[
\begin{pmatrix}
1 & 0 & 0 & -1 & 2 \\
0 & 1 & 0 & -1 & 0 \\
0 & 0 & 1 & 1 & -1 \\
\end{pmatrix}
\begin{pmatrix}
a \\
b \\
c \\
d \\
e \\
\end{pmatrix}
= \begin{pmatrix}
3 \\
1 \\
-1 \\
\end{pmatrix}
\]

We say \(d, e\) are the non-basic (or non-dependent) variables.

\[Ax = b \text{ and } x \geq 0.\]
Standard Simplex

- Incrementality: add/remove equations
- Slow backtracking
- No theory propagation
Fast Linear Arithmetic

- Simplex General Form
- Algorithm based on the dual simplex
- Non redundant proofs
- Efficient backtracking
- Efficient theory propagation
- Support for string inequalities: $t > 0$
- Preprocessing step
- Integer problems:
  - Gomory cuts, Branch & Bound, GCD test
General Form: \( Ax = 0 \) and \( l_j \leq x_j \leq u_j \)

Example:

\[
\begin{align*}
x & \geq 0, (x + y \leq 2 \lor x + 2y \geq 6), (x + y = 2 \lor x + 2y > 4) \\
\downarrow \\
s_1 & \equiv x + y, s_2 \equiv x + 2y, \\
x & \geq 0, (s_1 \leq 2 \lor s_2 \geq 6), (s_1 = 2 \lor s_2 > 4)
\end{align*}
\]

Only bounds (e.g., \( s_1 \leq 2 \)) are asserted during the search.

Unconstrained variables can be eliminated before the beginning of the search.
From Definitions to a Tableau

\[ s_1 \equiv x + y, \quad s_2 \equiv x + 2y \]
From Definitions to a Tableau

\[ s_1 \equiv x + y, \quad s_2 \equiv x + 2y \]

\[ s_1 = x + y, \quad s_2 = x + 2y \]
From Definitions to a Tableau

\[ s_1 \equiv x + y, \quad s_2 \equiv x + 2y \]

\[ s_1 = x + y, \quad s_2 = x + 2y \]

\[ s_1 - x - y = 0 \]
\[ s_2 - x - 2y = 0 \]
From Definitions to a Tableau

\[ s_1 \equiv x + y, \quad s_2 \equiv x + 2y \]

\[ s_1 = x + y, \quad s_2 = x + 2y \]

\[ s_1 - x - y = 0 \quad s_1, s_2 \text{ are basic (dependent)} \]

\[ s_2 - x - 2y = 0 \quad x, y \text{ are non-basic} \]
Pivoting

A way to swap a basic with a non-basic variable!
It is just equational reasoning.
Key invariant: a basic variable occurs in only one equation.
Example: swap \( s_1 \) and \( y \)

\[
\begin{align*}
s_1 - x - y &= 0 \\
s_2 - x - 2y &= 0
\end{align*}
\]
A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap $s_1$ and $y$

\[
\begin{align*}
  s_1 - x - y & = 0 \\
  s_2 - x - 2y & = 0 \\
  -s_1 + x + y & = 0 \\
  s_2 - x - 2y & = 0 
\end{align*}
\]
Pivoting

A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap \( s_1 \) and \( y \)

\[
\begin{align*}
  s_1 - x - y & = 0 \\
  s_2 - x - 2y & = 0 \\
  -s_1 + x + y & = 0 \\
  s_2 - x - 2y & = 0 \\
  -s_1 + x + y & = 0 \\
  s_2 - 2s_1 + x & = 0
\end{align*}
\]
A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap $s_1$ and $y$

\[
\begin{align*}
  s_1 - x - y &= 0 \\
  s_2 - x - 2y &= 0 \\
  -s_1 + x + y &= 0 \\
  s_2 - x - 2y &= 0 \\
  -s_1 + x + y &= 0 \\
  s_2 - 2s_1 + x &= 0
\end{align*}
\]

It is just substituting equals by equals.
A way to swap a basic with a non-basic variable!
It is just equational reasoning.
Key invariant: a basic variable occurs in only one equation.
Example: swap $s_1$ and $y$

\[
\begin{align*}
    s_1 - x - y &= 0 \\
    s_2 - x - 2y &= 0 \\
    -s_1 + x + y &= 0 \\
    s_2 - x - 2y &= 0 \\
    -s_1 + x + y &= 0 \\
    s_2 - 2s_1 + x &= 0
\end{align*}
\]

It is just substituting equals by equals.

Key Property: If an assignment satisfies the equations before a pivoting step, then it will also satisfy them after!
A way to swap a basic with a non-basic variable!

It is just equational reasoning.

Key invariant: a basic variable occurs in only one equation.

Example: swap $s_2$ and $y$

$$s_1 - x - y = 0$$
$$s_2 - x - 2y = 0$$
$$-s_1 + x + y = 0$$

It is just substituting equals by equals.

Example:

$M(x) = 1$
$M(y) = 1$
$M(s_1) = 2$
$M(s_2) = 3$

Key Property:
If an assignment satisfies the equations before a pivoting step, then it will also satisfy them after!
An **assignment** (model) is a mapping from variables to values.

We maintain an **assignment** that satisfies all **equations** and **bounds**.

The assignment of non dependent variables implies the assignment of dependent variables.

**Equations** + **Bounds** can be used to derive **new bounds**.

Example: $x = y - z$, $y \leq 2$, $z \geq 3 \Rightarrow x \leq -1$.

The **new bound** may be inconsistent with the already known bounds.

Example: $x \leq -1$, $x \geq 0$. 
If the assignment of a non-basic variable does not satisfy a bound, then fix it and propagate the change to all dependent variables.

\[
\begin{align*}
a &= c - d \\
b &= c + d \\
M(a) &= 0 \\
M(b) &= 0 \\
M(c) &= 0 \\
M(d) &= 0 \\
1 &\leq c
\end{align*}
\]

\[
\begin{align*}
a &= c - d \\
b &= c + d \\
M(a) &= 1 \\
M(b) &= 1 \\
M(c) &= 1 \\
M(d) &= 0 \\
1 &\leq c
\end{align*}
\]
If the assignment of a non-basic variable does not satisfy a bound, then fix it and propagate the change to all dependent variables. **Of course, we may introduce new “problems”**.

\[ a = c - d \]
\[ b = c + d \]
\[ M(a) = 0 \]
\[ M(b) = 0 \]
\[ M(c) = 0 \]
\[ M(d) = 0 \]
\[ 1 \leq c \]
\[ a \leq 0 \]
If the assignment of a basic variable does not satisfy a bound, then pivot it, fix it, and propagate the change to its new dependent variables.

\[
\begin{align*}
& a = c - d \\
& b = c + d \\
& M(a) = 0 \\
& M(b) = 0 \\
& M(c) = 0 \\
& M(d) = 0 \\
& 1 \leq a
\end{align*}
\]

\[
\begin{align*}
& c = a + d \\
& b = a + 2d \\
& M(a) = 0 \\
& M(b) = 0 \\
& M(c) = 0 \\
& M(d) = 0 \\
& 1 \leq a
\end{align*}
\]

\[
\begin{align*}
& c = a + d \\
& b = a + 2d \\
& M(a) = 1 \\
& M(b) = 1 \\
& M(c) = 1 \\
& M(d) = 0 \\
& 1 \leq a
\end{align*}
\]
Sometimes, a model cannot be repaired. It is pointless to pivot.

\[ a = b - c \]
\[ a \leq 0, \ 1 \leq b, \ c \leq 0 \]
\[ M(a) = 1 \]
\[ M(b) = 1 \]
\[ M(c) = 0 \]

The value of \( M(a) \) is too big. We can reduce it by:
- reducing \( M(b) \)
  not possible \( b \) is at lower bound
- increasing \( M(c) \)
  not possible \( c \) is at upper bound
Extracting proof from failed repair attempts is easy.

\[ s_1 \equiv a + d, \quad s_2 \equiv c + d \]
\[ a = s_1 - s_2 + c \]
\[ a \leq 0, \quad 1 \leq s_1, \quad s_2 \leq 0, \quad 0 \leq c \]
\[ M(a) = 1 \]
\[ M(s_1) = 1 \]
\[ M(s_2) = 0 \]
\[ M(c) = 0 \]
Extracting proof from failed repair attempts is easy.

\[ s_1 \equiv a + d, \ s_2 \equiv c + d \]
\[ a = s_1 - s_2 + c \]
\[ a \leq 0, \ 1 \leq s_1, \ s_2 \leq 0, \ 0 \leq c \]
\[ M(a) = 1 \]
\[ M(s_1) = 1 \]
\[ M(s_2) = 0 \]
\[ M(c) = 0 \]

\{ a \leq 0, \ 1 \leq s_1, \ s_2 \leq 0, \ 0 \leq c \} \text{ is inconsistent}
Extracting proof from failed repair attempts is easy.

\[ s_1 \equiv a + d, \quad s_2 \equiv c + d \]
\[ a = s_1 - s_2 + c \]
\[ a \leq 0, \; 1 \leq s_1, \; s_2 \leq 0, \; 0 \leq c \]

\[ M(a) = 1 \]
\[ M(s_1) = 1 \]
\[ M(s_2) = 0 \]
\[ M(c) = 0 \]

\{ a \leq 0, \; 1 \leq s_1, \; s_2 \leq 0, \; 0 \leq c \} \text{ is inconsistent}

\{ a \leq 0, \; 1 \leq a + d, \; c + d \leq 0, \; 0 \leq c \} \text{ is inconsistent}
The method described only handles non-strict inequalities (e.g., $x \leq 2$).

For integer problems, strict inequalities can be converted into non-strict inequalities. $x < 1 \leadsto x \leq 0$.

For rational/real problems, strict inequalities can be converted into non-strict inequalities using a small $\delta$. $x < 1 \leadsto x \leq 1 - \delta$.

We do not compute a $\delta$, we treat it symbolically.

$\delta$ is an infinitesimal parameter: $(c, k) = c + k\delta$
Example

- Initial state

\[ s \geq 1, x \geq 0 \]
\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = 0 )</td>
<td>( s = x + y )</td>
<td></td>
</tr>
<tr>
<td>( M(y) = 0 )</td>
<td>( u = x + 2y )</td>
<td></td>
</tr>
<tr>
<td>( M(s) = 0 )</td>
<td>( v = x - y )</td>
<td></td>
</tr>
<tr>
<td>( M(u) = 0 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M(v) = 0 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example

- **Asserting** $s \geq 1$

\[
s \geq 1, \; x \geq 0 \\
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 0$</td>
<td>$s = x + y$</td>
<td></td>
</tr>
<tr>
<td>$M(y) = 0$</td>
<td>$u = x + 2y$</td>
<td></td>
</tr>
<tr>
<td>$M(s) = 0$</td>
<td>$v = x - y$</td>
<td></td>
</tr>
<tr>
<td>$M(u) = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v) = 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Example**

- Asserting $s \geq 1$ assignment does not satisfy new bound.

\[ s \geq 1, x \geq 0 \]
\[ (y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 0$</td>
<td>$s = x + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 0$</td>
<td>$u = x + 2y$</td>
<td></td>
</tr>
<tr>
<td>$M(s) = 0$</td>
<td>$v = x - y$</td>
<td></td>
</tr>
<tr>
<td>$M(u) = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v) = 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Example**

- Asserting $s \geq 1$ pivot $s$ and $x$ ($s$ is a dependent variable).

  $s \geq 1, x \geq 0$

  $(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$</td>
<td>$s = x + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$</td>
<td>$u = x + 2y$</td>
<td></td>
</tr>
<tr>
<td>$M(s)$</td>
<td>$v = x - y$</td>
<td></td>
</tr>
<tr>
<td>$M(u)$</td>
<td>$M(v) = 0$</td>
<td></td>
</tr>
</tbody>
</table>
Example

- Asserting $s \geq 1$ pivot $s$ and $x$ ($s$ is a dependent variable).

\[ s \geq 1, x \geq 0 \]
\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 0$</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 0$</td>
<td>$u = x + 2y$</td>
<td></td>
</tr>
<tr>
<td>$M(s) = 0$</td>
<td>$v = x - y$</td>
<td></td>
</tr>
<tr>
<td>$M(u) = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v) = 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Asserting $s \geq 1$ pivot $s$ and $x$ ($s$ is a dependent variable).

\[
\begin{align*}
\text{Model} & \quad \text{Equations} & \quad \text{Bounds} \\
M(x) = 0 & \quad x = s - y & \quad s \geq 1 \\
M(y) = 0 & \quad u = s + y \\
M(s) = 0 & \quad v = s - 2y \\
M(u) = 0 & \\
M(v) = 0 & 
\end{align*}
\]
Example

- Asserting $s \geq 1$ update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$ = 0</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$ = 0</td>
<td>$u = s + y$</td>
<td></td>
</tr>
<tr>
<td>$M(s)$ = 1</td>
<td>$v = s - 2y$</td>
<td></td>
</tr>
<tr>
<td>$M(u)$ = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v)$ = 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Asserting $s \geq 1$, update dependent variables assignment.

\[ s \geq 1, x \geq 0 \]
\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 1$</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 0$</td>
<td>$u = s + y$</td>
<td></td>
</tr>
<tr>
<td>$M(s) = 1$</td>
<td>$v = s - 2y$</td>
<td></td>
</tr>
<tr>
<td>$M(u) = 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v) = 1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

Asserting $x \geq 0$

$s \geq 1, x \geq 0$

$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 1$</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 0$</td>
<td>$u = s + y$</td>
<td></td>
</tr>
<tr>
<td>$M(s) = 1$</td>
<td>$v = s - 2y$</td>
<td></td>
</tr>
<tr>
<td>$M(u) = 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v) = 1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Asserting $x \geq 0$ assignment satisfies new bound.

\[
s \geq 1, \quad x \geq 0
\]

\[
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$ = 1</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$ = 0</td>
<td>$u = s + y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s)$ = 1</td>
<td>$v = s - 2y$</td>
<td></td>
</tr>
<tr>
<td>$M(u)$ = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v)$ = 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Example**

- **Case split** \( \neg y \leq 1 \)

\[
s \geq 1, \quad x \geq 0
\]

\[
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = 1 )</td>
<td>( x = s - y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 0 )</td>
<td>( u = s + y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 1 )</td>
<td>( v = s - 2y )</td>
<td></td>
</tr>
<tr>
<td>( M(u) = 1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M(v) = 1 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

Case split $\neg y \leq 1$ assignment does not satisfy new bound.

$s \geq 1$, $x \geq 0$

$(y \leq 1 \lor u \geq 2)$, $(v \leq -2 \lor v \geq 0)$, $(v \leq -2 \lor u \leq -1)$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 1$</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 0$</td>
<td>$u = s + y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s) = 1$</td>
<td>$v = s - 2y$</td>
<td>$y &gt; 1$</td>
</tr>
<tr>
<td>$M(u) = 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v) = 1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case split $\neg y \leq 1$ update assignment.

$s \geq 1, x \geq 0$

$(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$ = 1</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 1 + \delta$</td>
<td>$u = s + y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s) = 1$</td>
<td>$v = s - 2y$</td>
<td>$y &gt; 1$</td>
</tr>
<tr>
<td>$M(u) = 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v) = 1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case split \( \neg y \leq 1 \) update dependent variables assignment.

\[ s \geq 1, \quad x \geq 0 \]

\[(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) ) = (-\delta)</td>
<td>( x = s - y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) ) = (1 + \delta)</td>
<td>( u = s + y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) ) = (1)</td>
<td>( v = s - 2y )</td>
<td>( y &gt; 1 )</td>
</tr>
<tr>
<td>( M(u) ) = (2 + \delta)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M(v) ) = (-1 - 2\delta)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- **Bound violation**

\[ s \geq 1, \ x \geq 0 \]

\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) ) = (-\delta)</td>
<td>( x = s - y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) ) = (1 + \delta)</td>
<td>( u = s + y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) ) = (1)</td>
<td>( v = s - 2y )</td>
<td>( y &gt; 1 )</td>
</tr>
<tr>
<td>( M(u) ) = (2 + \delta)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M(v) ) = (-1 - 2\delta)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- **Bound violation** pivot $x$ and $s$ ($x$ is a dependent variables).

\[ s \geq 1, \ x \geq 0 \]
\[ (y \leq 1 \lor v \geq 2), \ (v \leq -2 \lor v \geq 0), \ (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$</td>
<td>$u = s + y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s)$</td>
<td>$v = s - 2y$</td>
<td>$y &gt; 1$</td>
</tr>
<tr>
<td>$M(u)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ M(x) = -\delta \]
\[ M(y) = 1 + \delta \]
\[ M(s) = 1 \]
\[ M(u) = 2 + \delta \]
\[ M(v) = -1 - 2\delta \]
Example

- Bound violation on pivot $x$ and $s$ ($x$ is a dependent variable).

\[
s \geq 1, \ x \geq 0
\]

\[
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$ = $-\delta$</td>
<td>$s = x + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$ = $1 + \delta$</td>
<td>$u = s + y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s)$ = $1$</td>
<td>$v = s - 2y$</td>
<td>$y &gt; 1$</td>
</tr>
<tr>
<td>$M(u)$ = $2 + \delta$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v)$ = $-1 - 2\delta$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bound violation pivot $x$ and $s$ ($x$ is a dependent variables).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)$$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$ = $-\delta$</td>
<td>$s = x + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$ = $1 + \delta$</td>
<td>$u = x + 2y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s)$ = $1$</td>
<td>$v = x - y$</td>
<td>$y &gt; 1$</td>
</tr>
<tr>
<td>$M(u)$ = $2 + \delta$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v)$ = $-1 - 2\delta$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Bound violation update assignment.

\[ s \geq 1, x \geq 0 \]

\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = 0 )</td>
<td>( s = x + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 1 + \delta )</td>
<td>( u = x + 2y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 1 )</td>
<td>( v = x - y )</td>
<td>( y &gt; 1 )</td>
</tr>
<tr>
<td>( M(u) = 2 + \delta )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M(v) = -1 - 2\delta )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Bound violation: update dependent variables assignment.

\[ s \geq 1, x \geq 0 \]

\[(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M(x)) = 0</td>
<td>(s = x + y)</td>
<td>(s \geq 1)</td>
</tr>
<tr>
<td>(M(y) = 1 + \delta)</td>
<td>(u = x + 2y)</td>
<td>(x \geq 0)</td>
</tr>
<tr>
<td>(M(s) = 1 + \delta)</td>
<td>(v = x - y)</td>
<td>(y &gt; 1)</td>
</tr>
<tr>
<td>(M(u) = 2 + 2\delta)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(M(v) = -1 - \delta)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Theory propagation: \( x \geq 0, y > 1 \implies u > 2 \)

\[
s \geq 1, x \geq 0 \\
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) ) 0</td>
<td>( s = x + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) ) ( 1 + \delta )</td>
<td>( u = x + 2y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) ) ( 1 + \delta )</td>
<td>( v = x - y )</td>
<td>( y &gt; 1 )</td>
</tr>
<tr>
<td>( M(u) ) ( 2 + 2\delta )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M(v) ) ( -1 - \delta )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

Theory propagation: \( u > 2 \implies -u \leq -1 \)

\[
\begin{align*}
    s & \geq 1, x \geq 0 \\
    (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\end{align*}
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) ) = 0</td>
<td>( s = x + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 1 + \delta )</td>
<td>( u = x + 2y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 1 + \delta )</td>
<td>( v = x - y )</td>
<td>( y &gt; 1 )</td>
</tr>
<tr>
<td>( M(u) = 2 + 2\delta )</td>
<td></td>
<td>( u &gt; 2 )</td>
</tr>
<tr>
<td>( M(v) = -1 - \delta )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Boolean propagation \( \neg y \leq 1 \leadsto v \geq 2 \)

\[
\begin{align*}
 s & \geq 1, \ x \geq 0 \\
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\end{align*}
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = 0 )</td>
<td>( s = x + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 1 + \delta )</td>
<td>( u = x + 2y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 1 + \delta )</td>
<td>( v = x - y )</td>
<td>( y &gt; 1 )</td>
</tr>
<tr>
<td>( M(u) = 2 + 2\delta )</td>
<td></td>
<td>( u &gt; 2 )</td>
</tr>
<tr>
<td>( M(v) = -1 - \delta )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Theory propagation $v \geq 2 \implies \neg v \leq -2$

  
  $s \geq 1, x \geq 0$

  $(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 0$</td>
<td>$s = x + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 1 + \delta$</td>
<td>$u = x + 2y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s) = 1 + \delta$</td>
<td>$v = x - y$</td>
<td>$y &gt; 1$</td>
</tr>
<tr>
<td>$M(u) = 2 + 2\delta$</td>
<td></td>
<td>$u &gt; 2$</td>
</tr>
<tr>
<td>$M(v) = -1 - \delta$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

Conflict empty clause

\[ s \geq 1, x \geq 0 \]
\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = 0 )</td>
<td>( s = x + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 1 + \delta )</td>
<td>( u = x + 2y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 1 + \delta )</td>
<td>( v = x - y )</td>
<td>( y &gt; 1 )</td>
</tr>
<tr>
<td>( M(u) = 2 + 2\delta )</td>
<td></td>
<td>( u &gt; 2 )</td>
</tr>
<tr>
<td>( M(v) = -1 - \delta )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

Backtracking

$s \geq 1, x \geq 0$

$(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$</td>
<td>$s = x + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$</td>
<td>$u = x + 2y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s)$</td>
<td>$v = x - y$</td>
<td></td>
</tr>
<tr>
<td>$M(u)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Asserting $y \leq 1$

$s \geq 1, x \geq 0$

$(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$</td>
<td>$s = x + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$</td>
<td>$u = x + 2y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s)$</td>
<td>$v = x - y$</td>
<td></td>
</tr>
<tr>
<td>$M(u)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

 Asserting \( y \leq 1 \) assignment does not satisfy new bound.

\[
s \geq 1, \ x \geq 0 \\
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) ) = 0</td>
<td>( s = x + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 1 + \delta )</td>
<td>( u = x + 2y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 1 + \delta )</td>
<td>( v = x - y )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>( M(u) = 2 + 2\delta )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M(v) = -1 - \delta )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Asserting $y \leq 1$ update assignment.

\[
s \geq 1, \quad x \geq 0 \]
\[
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor u \geq 0), (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 0$</td>
<td>$s = x + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 1$</td>
<td>$u = x + 2y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s) = 1 + \delta$</td>
<td>$v = x - y$</td>
<td>$y \leq 1$</td>
</tr>
<tr>
<td>$M(u) = 2 + 2\delta$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v) = -1 - \delta$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Asserting $y \leq 1$ update dependent variables assignment.

\[ s \geq 1, \ x \geq 0 \]

\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 0$</td>
<td>$s = x + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 1$</td>
<td>$u = x + 2y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s) = 1$</td>
<td>$v = x - y$</td>
<td>$y \leq 1$</td>
</tr>
<tr>
<td>$M(u) = 2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v) = -1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Theory propagation: $s \geq 1, y \leq 1 \Rightarrow v \geq -1$

- $s \geq 1, x \geq 0$

- $(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$</td>
<td>$u = s + y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s)$</td>
<td>$v = s - 2y$</td>
<td>$y \leq 1$</td>
</tr>
<tr>
<td>$M(u)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$M(v)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Theory propagation: $v \geq -1 \iff -v \leq -2$

$s \geq 1, x \geq 0$

$(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 0$</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 1$</td>
<td>$u = s + y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s) = 1$</td>
<td>$v = s - 2y$</td>
<td>$y \leq 1$</td>
</tr>
<tr>
<td>$M(u) = 2$</td>
<td></td>
<td>$v \geq -1$</td>
</tr>
<tr>
<td>$M(v) = -1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- **Boolean propagation**  
  \(-v \leq -2 \iff v \geq 0\)
  
  \(s \geq 1, x \geq 0\)

  \((y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)\)

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M(x) = 0)</td>
<td>(x = s - y)</td>
<td>(s \geq 1)</td>
</tr>
<tr>
<td>(M(y) = 1)</td>
<td>(u = s + y)</td>
<td>(x \geq 0)</td>
</tr>
<tr>
<td>(M(s) = 1)</td>
<td>(v = s - 2y)</td>
<td>(y \leq 1)</td>
</tr>
<tr>
<td>(M(u) = 2)</td>
<td>(v \geq -1)</td>
<td></td>
</tr>
</tbody>
</table>
**Example**

- **Bound violation** assignment does not satisfy new bound.

\[ s \geq 1, \ x \geq 0 \]
\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M(x) = 0)</td>
<td>(x = s - y)</td>
<td>(s \geq 1)</td>
</tr>
<tr>
<td>(M(y) = 1)</td>
<td>(u = s + y)</td>
<td>(x \geq 0)</td>
</tr>
<tr>
<td>(M(s) = 1)</td>
<td>(v = s - 2y)</td>
<td>(y \leq 1)</td>
</tr>
<tr>
<td>(M(u) = 2)</td>
<td></td>
<td>(v \geq 0)</td>
</tr>
<tr>
<td>(M(v) = -1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- **Bound violation** pivot $u$ and $s$ ($u$ is a dependent variable).

\[
s \geq 1, \quad x \geq 0
\]

\[
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 0$</td>
<td>$x = s - y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 1$</td>
<td>$u = s + y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s) = 1$</td>
<td>$v = s - 2y$</td>
<td>$y \leq 1$</td>
</tr>
<tr>
<td>$M(u) = 2$</td>
<td></td>
<td>$v \geq 0$</td>
</tr>
<tr>
<td>$M(v) = -1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Bound violation pivot \( u \) and \( s \) (\( u \) is a dependent variable).

\[
s \geq 1, \ x \geq 0
\]
\[
(y \leq 1 \lor v \geq 2), \ (v \leq -2 \lor v \geq 0), \ (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = 0 )</td>
<td>( x = s - y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 1 )</td>
<td>( u = s + y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 1 )</td>
<td>( s = v + 2y )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>( M(u) = 2 )</td>
<td>( v \geq 0 )</td>
<td></td>
</tr>
<tr>
<td>( M(v) = -1 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Bound violation pivot \( u \) and \( s \) (\( u \) is a dependent variable).

\[
s \geq 1, \quad x \geq 0
\]

\[
(y \leq 1 \lor v \geq 2), \quad (v \leq -2 \lor v \geq 0), \quad (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model ( M(x) )</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x = v + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>1</td>
<td>( u = v + 3y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>1</td>
<td>( s = v + 2y )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>( v \geq 0 )</td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Bound violation update assignment.

\[ s \geq 1, x \geq 0 \]

\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = 0 )</td>
<td>( x = v + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 1 )</td>
<td>( u = v + 3y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 1 )</td>
<td>( s = v + 2y )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>( M(u) = 2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M(v) = 0 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- Bound violation: update dependent variables assignment.

\[ s \geq 1, x \geq 0 \]
\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = 1 )</td>
<td>( x = v + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 1 )</td>
<td>( u = v + 3y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 2 )</td>
<td>( s = v + 2y )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>( M(u) = 3 )</td>
<td></td>
<td>( v \geq 0 )</td>
</tr>
<tr>
<td>( M(v) = 0 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- **Boolean propagation**
  \[ \neg v \leq -2 \iff u \leq -1 \]

  \[ s \geq 1, \ x \geq 0 \]

  \[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = 1 )</td>
<td>( x = v + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 1 )</td>
<td>( u = v + 3y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 2 )</td>
<td>( s = v + 2y )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>( M(u) = 3 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( M(v) = 0 )</td>
<td></td>
<td>( v \geq 0 )</td>
</tr>
</tbody>
</table>
**Example**

- **Bound violation** assignment does not satisfy new bound.

\[ s \geq 1, x \geq 0 \]

\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = 1 )</td>
<td>( x = v + y )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = 1 )</td>
<td>( u = v + 3y )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 2 )</td>
<td>( s = v + 2y )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>( M(u) = 3 )</td>
<td></td>
<td>( v \geq 0 )</td>
</tr>
<tr>
<td>( M(v) = 0 )</td>
<td></td>
<td>( u \leq -1 )</td>
</tr>
</tbody>
</table>
**Example**

- Bound violation pivot $u$ and $y$ ($u$ is a dependent variable).

\[
\begin{align*}
  s & \geq 1, \quad x \geq 0 \\
  (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\end{align*}
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 1$</td>
<td>$x = v + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 1$</td>
<td>$u = v + 3y$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s) = 2$</td>
<td>$s = v + 2y$</td>
<td>$y \leq 1$</td>
</tr>
<tr>
<td>$M(u) = 3$</td>
<td>$u \leq -1$</td>
<td></td>
</tr>
<tr>
<td>$M(v) = 0$</td>
<td></td>
<td>$v \geq 0$</td>
</tr>
</tbody>
</table>
Example

- Bound violation pivot $u$ and $y$ ($u$ is a dependent variable).

\[ s \geq 1, x \geq 0 \]

\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$ = 1</td>
<td>$x = v + y$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$ = 1</td>
<td>$y = \frac{1}{3}u - \frac{1}{3}v$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s)$ = 2</td>
<td>$s = v + 2y$</td>
<td></td>
</tr>
<tr>
<td>$M(u)$ = 3</td>
<td></td>
<td>$y \leq 1$</td>
</tr>
<tr>
<td>$M(v)$ = 0</td>
<td></td>
<td>$v \geq 0$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$u \leq -1$</td>
</tr>
</tbody>
</table>
**Example**

- **Bound violation** pivot $u$ and $y$ ($u$ is a dependent variable).

\[
s \geq 1, \quad x \geq 0
\]
\[
(y \leq 1 \lor v \geq 2), \quad (v \leq -2 \lor v \geq 0), \quad (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x) = 1$</td>
<td>$x = \frac{1}{3}u + \frac{2}{3}v$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y) = 1$</td>
<td>$y = \frac{1}{3}u - \frac{1}{3}v$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s) = 2$</td>
<td>$s = \frac{2}{3}u + \frac{1}{3}v$</td>
<td>$y \leq 1$</td>
</tr>
<tr>
<td>$M(u) = 3$</td>
<td></td>
<td>$v \geq 0$</td>
</tr>
<tr>
<td>$M(v) = 0$</td>
<td></td>
<td>$u \leq -1$</td>
</tr>
</tbody>
</table>
**Example**

- **Bound violation** update assignment.

\[
\begin{align*}
    s & \geq 1, 
    x & \geq 0 \\
    (y & \leq 1 \lor v \geq 2), 
    (v \leq -2 \lor v \geq 0), 
    (v \leq -2 \lor u \leq -1)
\end{align*}
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M(x) = 1)</td>
<td>(x = \frac{1}{3}u + \frac{2}{3}v)</td>
<td>(s \geq 1)</td>
</tr>
<tr>
<td>(M(y) = 1)</td>
<td>(y = \frac{1}{3}u - \frac{1}{3}v)</td>
<td>(x \geq 0)</td>
</tr>
<tr>
<td>(M(s) = 2)</td>
<td>(s = \frac{2}{3}u + \frac{1}{3}v)</td>
<td>(y \leq 1)</td>
</tr>
<tr>
<td>(M(u) = -1)</td>
<td></td>
<td>(v \geq 0)</td>
</tr>
<tr>
<td>(M(v) = 0)</td>
<td></td>
<td>(u \leq -1)</td>
</tr>
</tbody>
</table>
Example

- Bound violation

\[ s \geq 1, x \geq 0 \]
\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = -\frac{1}{3} )</td>
<td>( x = \frac{1}{3}u + \frac{2}{3}v )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = -\frac{1}{3} )</td>
<td>( y = \frac{1}{3}u - \frac{1}{3}v )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = -\frac{2}{3} )</td>
<td>( s = \frac{2}{3}u + \frac{1}{3}v )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>( M(u) = -1 )</td>
<td></td>
<td>( v \geq 0 )</td>
</tr>
<tr>
<td>( M(v) = 0 )</td>
<td></td>
<td>( u \leq -1 )</td>
</tr>
</tbody>
</table>
Example

- Bound violations

\[ s \geq 1, \ x \geq 0 \]

\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) ) = (-\frac{1}{3})</td>
<td>( x = \frac{1}{3}u + \frac{2}{3}v )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) ) = (-\frac{1}{3})</td>
<td>( y = \frac{1}{3}u - \frac{1}{3}v )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) ) = (-\frac{2}{3})</td>
<td>( s = \frac{2}{3}u + \frac{1}{3}v )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>( M(u) ) = (-1)</td>
<td></td>
<td>( v \geq 0 )</td>
</tr>
<tr>
<td>( M(v) ) = (0)</td>
<td></td>
<td>( u \leq -1 )</td>
</tr>
</tbody>
</table>
Example

- Bound violations pivot \( s \) and \( v \) (\( s \) is a dependent variable).

\[
s \geq 1, \ x \geq 0
\]

\[
(y \leq 1 \lor v \geq 2), \ (v \leq -2 \lor v \geq 0), \ (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) ) = (-\frac{1}{3})</td>
<td>( x = \frac{1}{3} u + \frac{2}{3} v )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) ) = (-\frac{1}{3})</td>
<td>( y = \frac{1}{3} u - \frac{1}{3} v )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) ) = (-\frac{2}{3})</td>
<td>( s = \frac{2}{3} u + \frac{1}{3} v )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>( M(u) ) = (-1)</td>
<td></td>
<td>( v \geq 0 )</td>
</tr>
<tr>
<td>( M(v) ) = (0)</td>
<td></td>
<td>( u \leq -1 )</td>
</tr>
</tbody>
</table>
Example

- Bound violations pivot $s$ and $v$ ($s$ is a dependent variable).

\[
s \geq 1, x \geq 0 \\
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$</td>
<td>$x = \frac{1}{3}u + \frac{2}{3}v$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$</td>
<td>$y = \frac{1}{3}u - \frac{1}{3}v$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s)$</td>
<td>$v = 3s - 2u$</td>
<td>$y \leq 1$</td>
</tr>
<tr>
<td>$M(u)$</td>
<td>$u \leq -1$</td>
<td>$v \geq 0$</td>
</tr>
<tr>
<td>$M(v)$</td>
<td>$v \geq 0$</td>
<td>$u \leq -1$</td>
</tr>
</tbody>
</table>
Example

- Bound violations pivot $s$ and $v$ ($s$ is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)$$

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M(x)$ = $-\frac{1}{3}$</td>
<td>$x = 2s - u$</td>
<td>$s \geq 1$</td>
</tr>
<tr>
<td>$M(y)$ = $-\frac{1}{3}$</td>
<td>$y = -s + u$</td>
<td>$x \geq 0$</td>
</tr>
<tr>
<td>$M(s)$ = $-\frac{2}{3}$</td>
<td>$v = 3s - 2u$</td>
<td>$y \leq 1$</td>
</tr>
<tr>
<td>$M(u)$ = $-1$</td>
<td></td>
<td>$v \geq 0$</td>
</tr>
<tr>
<td>$M(v)$ = $0$</td>
<td></td>
<td>$u \leq -1$</td>
</tr>
</tbody>
</table>
Example

- Bound violations update assignment.

\[ s \geq 1, x \geq 0 \]

\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M(x) = -\frac{1}{3} )</td>
<td>( x = 2s - u )</td>
<td>( s \geq 1 )</td>
</tr>
<tr>
<td>( M(y) = -\frac{1}{3} )</td>
<td>( y = -s + u )</td>
<td>( x \geq 0 )</td>
</tr>
<tr>
<td>( M(s) = 1 )</td>
<td>( v = 3s - 2u )</td>
<td>( y \leq 1 )</td>
</tr>
<tr>
<td>( M(u) = -1 )</td>
<td></td>
<td>( v \geq 0 )</td>
</tr>
<tr>
<td>( M(v) = 0 )</td>
<td></td>
<td>( u \leq -1 )</td>
</tr>
</tbody>
</table>
Example

Bound violations: update dependent variables assignment.

\[ s \geq 1, x \geq 0 \]
\[ (y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1) \]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M(x)) = 3</td>
<td>(x = 2s - u)</td>
<td>(s \geq 1)</td>
</tr>
<tr>
<td>(M(y)) = -2</td>
<td>(y = -s + u)</td>
<td>(x \geq 0)</td>
</tr>
<tr>
<td>(M(s)) = 1</td>
<td>(v = 3s - 2u)</td>
<td>(y \leq 1)</td>
</tr>
<tr>
<td>(M(u)) = -1</td>
<td></td>
<td>(v \geq 0)</td>
</tr>
<tr>
<td>(M(v)) = 5</td>
<td></td>
<td>(u \leq -1)</td>
</tr>
</tbody>
</table>
**Example**

- Found satisfying assignment

\[
s \geq 1, \ x \geq 0
\]

\[
(y \leq 1 \lor v \geq 2), (v \leq -2 \lor v \geq 0), (v \leq -2 \lor u \leq -1)
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Equations</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M(x)) = 3</td>
<td>(x = 2s - u)</td>
<td>(s \geq 1)</td>
</tr>
<tr>
<td>(M(y)) = -2</td>
<td>(y = -s + u)</td>
<td>(x \geq 0)</td>
</tr>
<tr>
<td>(M(s)) = 1</td>
<td>(v = 3s - 2u)</td>
<td>(y \leq 1)</td>
</tr>
<tr>
<td>(M(u)) = -1</td>
<td></td>
<td>(v \geq 0)</td>
</tr>
<tr>
<td>(M(v)) = 5</td>
<td></td>
<td>(u \leq -1)</td>
</tr>
</tbody>
</table>
Completeness: trivial
Soundness: also trivial
Termination: non trivial.

We cannot choose arbitrary variable to pivot.
Assume the variables are ordered.
Bland’s rule: select the smallest basic variable $c$ that does not satisfy its bounds, then select the smallest non-basic in the row of $c$ that can be used for pivoting.

Too technical.

Uses the fact that a tableau has a finite number of configurations. Then, any infinite trace will have cycles.
In practice, we need a combination of theories.

\[ b + 2 = c \quad \text{and} \quad f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \]

A theory is a set (potentially infinite) of first-order sentences.

**Main questions:**
Is the union of two theories \( T_1 \cup T_2 \) consistent?
Given a solvers for \( T_1 \) and \( T_2 \), how can we build a solver for \( T_1 \cup T_2 \)?
Two theories are disjoint if they do not share function/constant and predicate symbols.

= is the only exception.

Example:
The theories of arithmetic and arrays are disjoint.

Arithmetic symbols: \{0, -1, 1, -2, 2, \ldots, +, -, *, >, <, \geq, \leq\}
Array symbols: \{ read, write \}
It is a different name for our “naming” subterms procedure.

\[ b + 2 = c, \quad f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \]

\[ b + 2 = c, \quad v_6 \neq v_7 \]
\[ v_1 \equiv 3, \quad v_2 \equiv \text{write}(a, b, v_1), \quad v_3 \equiv c-2, \quad v_4 \equiv \text{read}(v_2, v_3), \quad v_5 \equiv c-b+1, \quad v_6 \equiv f(v_4), \quad v_7 \equiv f(v_5) \]
It is a different name for our “naming” subterms procedure.

\[ b + 2 = c, \quad f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \]

\[ b + 2 = c, \quad v_6 \neq v_7 \]

\[ v_1 \equiv 3, \quad v_2 \equiv \text{write}(a, b, v_1), \quad v_3 \equiv c-2, \quad v_4 \equiv \text{read}(v_2, v_3), \]
\[ v_5 \equiv c-b+1, \quad v_6 \equiv f(v_4), \quad v_7 \equiv f(v_5) \]

\[ b + 2 = c, \quad v_1 \equiv 3, \quad v_3 \equiv c-2, \quad v_5 \equiv c-b+1, \]
\[ v_2 \equiv \text{write}(a, b, v_1), \quad v_4 \equiv \text{read}(v_2, v_3), \]
\[ v_6 \equiv f(v_4), \quad v_7 \equiv f(v_5), \quad v_6 \neq v_7 \]
A theory is stably infinite if every satisfiable QFF is satisfiable in an infinite model.

EUF and arithmetic are stably infinite.

Bit-vectors are not.
The union of two consistent, disjoint, stably infinite theories is consistent.
A theory $T$ is **convex** iff

for all finite sets $S$ of literals and

for all $a_1 = b_1 \lor \ldots \lor a_n = b_n$

$S$ implies $a_1 = b_1 \lor \ldots \lor a_n = b_n$

iff

$S$ implies $a_i = b_i$ for some $1 \leq i \leq n$
Every convex theory with non trivial models is stably infinite.

All Horn equational theories are convex.

formulas of the form $s_1 \neq r_1 \lor \ldots \lor s_n \neq r_n \lor t = t'$

Linear rational arithmetic is convex.
Convexity: Negative Results

Linear integer arithmetic is not convex
\[ 1 \leq a \leq 2, \; b = 1, \; c = 2 \; \text{implies} \; a = b \lor a = c \]

Nonlinear arithmetic
\[ a^2 = 1, \; b = 1, \; c = -1 \; \text{implies} \; a = b \lor a = c \]

Theory of bit-vectors

Theory of arrays
\[ c_1 = \text{read}(\text{write}(a, i, c_2), j), \; c_3 = \text{read}(a, j) \]
implies \[ c_1 = c_2 \lor c_1 = c_3 \]
EUF is convex (O(n log n))
IDL is non-convex (O(nm))

EUF \cup IDL is NP-Complete

Reduce 3CNF to EUF \cup IDL
For each boolean variable \( p_i \) add \( 0 \leq a_i \leq 1 \)
For each clause \( p_1 \lor \neg p_2 \lor p_3 \) add
\[ f(a_1, a_2, a_3) \neq f(0, 1, 0) \]
EUF is convex (O(n log n))
IDL is non-convex (O(nm))

EUF \cup IDL is NP-Complete

Reduce 3CNF to EUF \cup IDL

For each boolean variable \( p_i \) add \( 0 \leq a_i \leq 1 \)

For each clause \( p_1 \lor \neg p_2 \lor p_3 \) add

\[
 f(a_1, a_2, a_3) \neq f(0, 1, 0)
\]

implies

\[
 a_1 \neq 0 \lor a_2 \neq 1 \lor a_3 \neq 0
\]
Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be consistent, stably infinite theories over disjoint (countable) signatures. Assume satisfiability of conjunction of literals can decided in $O(T_1(n))$ and $O(T_2(n))$ time respectively. Then,

1. The combined theory $\mathcal{U}$ is consistent and stably infinite.
2. Satisfiability of quantifier free conjunction of literals in $\mathcal{T}$ can be decided in $O(2^{n^2} \times (T_1(n) + T_2(n)))$.
3. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are convex, then so is $\mathcal{T}$ and satisfiability in $\mathcal{T}$ is in $O(n^3 \times (T_1(n) + T_2(n)))$. 
Nelson-Oppen Combination

The combination procedure:

**Initial State:** $\phi$ is a conjunction of literals over $\Sigma_1 \cup \Sigma_2$.

**Purification:** Preserving satisfiability transform $\phi$ into $\phi_1 \land \phi_2$, such that, $\phi_i \in \Sigma_i$.

**Interaction:** Guess a partition of $\forall(\phi_1) \cap \forall(\phi_2)$ into disjoint subsets. Express it as conjunction of literals $\psi$.

Example. The partition $\{x_1\}, \{x_2, x_3\}, \{x_4\}$ is represented as $x_1 \neq x_2, x_1 \neq x_4, x_2 \neq x_4, x_2 = x_3$.

**Component Procedures** : Use individual procedures to decide whether $\phi_i \land \psi$ is satisfiable.

**Return:** If both return yes, return yes. No, otherwise.
Each step is satisfiability preserving.

Say $\phi$ is satisfiable (in the combination).

- Purification: $\phi_1 \land \phi_2$ is satisfiable.
- Iteration: for some partition $\psi$, $\phi_1 \land \phi_2 \land \psi$ is satisfiable.
- Component procedures: $\phi_1 \land \psi$ and $\phi_2 \land \psi$ are both satisfiable in component theories.
- Therefore, if the procedure return unsatisfiable, then $\phi$ is unsatisfiable.
Suppose the procedure returns satisfiable.

- Let $\psi$ be the partition and $A$ and $B$ be models of $\mathcal{T}_1 \land \phi_1 \land \psi$
  and $\mathcal{T}_2 \land \phi_2 \land \psi$.

- The component theories are stably infinite. So, assume the models are infinite (of same cardinality).

- Let $h$ be a bijection between $|A|$ and $|B|$ such that
  \[ h(A(x)) = B(x) \]
  for each shared variable.

- Extend $B$ to $\bar{B}$ by interpretations of symbols in $\Sigma_1$:
  \[ \bar{B}(f)(b_1, \ldots, b_n) = h(A(f)(h^{-1}(b_1), \ldots, h^{-1}(b_n))) \]

- $\bar{B}$ is a model of:
  \[ \mathcal{T}_1 \land \phi_1 \land \mathcal{T}_2 \land \phi_2 \land \psi \]
Instead of guessing, we can deduce the equalities to be shared.

**Purification:** no changes.

**Interaction:** Deduce an equality $x = y$:

$$\mathcal{T}_1 \vdash (\phi_1 \Rightarrow x = y)$$

Update $\phi_2 := \phi_2 \land x = y$. And vice-versa. Repeat until no further changes.

**Component Procedures:** Use individual procedures to decide whether $\phi_i$ is satisfiable.

Remark: $\mathcal{T}_i \vdash (\phi_i \Rightarrow x = y)$ iff $\phi_i \land x \neq y$ is not satisfiable in $\mathcal{T}_i$. 

(sans titre image)
NO deterministic procedure
Completeness

Assume the theories are convex.

- Suppose $\phi_i$ is satisfiable.
- Let $E$ be the set of equalities $x_j = x_k$ ($j \neq k$) such that,
  $$T_i \vdash \phi_i \Rightarrow x_j = x_k.$$
- By convexity, $T_i \vdash \phi_i \Rightarrow \bigvee_E x_j = x_k$.
- $\phi_i \land \bigwedge_E x_j \neq x_k$ is satisfiable.
- The proof now is identical to the nondeterministic case.
- Sharing equalities is sufficient, because a theory $T_1$ can assume that $x^B \neq y^B$ whenever $x = y$ is not implied by $T_2$ and vice versa.
b + 2 = c, \( f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \)

**Arithmetic**
- \( b + 2 = c, \)
- \( v_1 \equiv 3, \)
- \( v_3 \equiv c-2, \)
- \( v_5 \equiv c-b+1 \)

**Arrays**
- \( v_2 \equiv \text{write}(a, b, v_1), \)
- \( v_4 \equiv \text{read}(v_2, v_3) \)

**EUF**
- \( v_6 \equiv f(v_4), \)
- \( v_7 \equiv f(v_5), \)
- \( v_6 \neq v_7 \)
b + 2 = c, f(read(write(a, b, 3), c-2)) ≠ f(c-b+1)

Arithmetic
b + 2 = c,
v₁ ≡ 3,
v₃ ≡ c-2,
v₅ ≡ c-b+1

Arrays
v₂ ≡ write(a, b, v₁),
v₄ ≡ read(v₂, v₃)

EUF
v₆ ≡ f(v₄),
v₇ ≡ f(v₅),
v₆ ≠ v₇

Substituting c
NO procedure: Example

\[ b + 2 = c, \quad f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \]

**Arithmetic**
- \( b + 2 = c, \)
- \( v_1 \equiv 3, \)
- \( v_3 \equiv b, \)
- \( v_5 \equiv 3 \)

**Arrays**
- \( v_2 \equiv \text{write}(a, b, v_1), \)
- \( v_4 \equiv \text{read}(v_2, v_3), \)

**EUF**
- \( v_6 \equiv f(v_4), \)
- \( v_7 \equiv f(v_5), \)
- \( v_6 \neq v_7 \)

**Propagating**
- \( v_3 = b \)
b + 2 = c, f(read(write(a, b, 3), c-2)) ≠ f(c-b+1)

Arithmetic
b + 2 = c,
v₁ ≡ 3,
v₃ ≡ b,
v₅ ≡ 3

Arrays
v₂ ≡ write(a, b, v₁),
v₄ ≡ read(v₂, v₃),
v₃ = b

EUF
v₆ ≡ f(v₄),
v₇ ≡ f(v₅),
v₆ ≠ v₇,
v₃ = b

Deducing v₄ = v₁
b + 2 = c, f(read(write(a,b,3), c-2)) ≠ f(c-b+1)

Arithmetic
b + 2 = c,
v_1 ≜ 3,
v_3 ≜ b,
v_5 ≜ 3

Arrays
v_2 ≜ write(a, b, v_1),
v_4 ≜ read(v_2, v_3),
v_3 = b,
v_4 = v_1

EUF
v_6 ≜ f(v_4),
v_7 ≜ f(v_5),
v_6 ≠ v_7,
v_3 = b

Propagating v_4 = v_1
NO procedure: Example

\[ b + 2 = c, \ f(\text{read}(\text{write}(a, b, 3), \ c-2)) \neq f(c-b+1) \]

**Arithmetic**
- \( b + 2 = c, \)
- \( v_1 \equiv 3, \)
- \( v_3 \equiv b, \)
- \( v_5 \equiv 3, \)
- \( v_4 = v_1 \)

**Arrays**
- \( v_2 \equiv \text{write}(a, b, v_1), \)
- \( v_4 \equiv \text{read}(v_2, v_3), \)
- \( v_3 = b, \)
- \( v_4 = v_1 \)

**EUF**
- \( v_6 \equiv f(v_4), \)
- \( v_7 \equiv f(v_5), \)
- \( v_6 \neq v_7, \)
- \( v_3 = b, \)
- \( v_4 = v_1 \)

Propagating \( v_5 = v_1 \)
b + 2 = c, \( f(\text{read(write}(a,b,3), c-2)) \neq f(c-b+1) \)

**Arithmetic**
- \( b + 2 = c \)
- \( \nu_1 \equiv 3 \)
- \( \nu_3 \equiv b \)
- \( \nu_5 \equiv 3 \)
- \( \nu_4 = \nu_1 \)

**Arrays**
- \( \nu_2 \equiv \text{write}(a, b, \nu_1) \)
- \( \nu_4 \equiv \text{read}(\nu_2, \nu_3) \)
- \( \nu_3 = b \)
- \( \nu_4 = \nu_1 \)

**EUF**
- \( \nu_6 \equiv f(\nu_4) \)
- \( \nu_7 \equiv f(\nu_5) \)
- \( \nu_6 \neq \nu_7 \)
- \( \nu_3 = b \)
- \( \nu_4 = \nu_1 \)
- \( \nu_5 = \nu_1 \)

**Congruence:** \( \nu_6 = \nu_7 \)
NO procedure: Example

\[ b + 2 = c, \quad \text{f(read(write(a,b,3), c-2))} \neq \text{f(c-b+1)} \]

Arithmetic
\[ b + 2 = c, \]
\[ v_1 \equiv 3, \]
\[ v_3 \equiv b, \]
\[ v_5 \equiv 3, \]
\[ v_4 = v_1 \]

Arrays
\[ v_2 \equiv \text{write(a, b, v}_1), \]
\[ v_4 \equiv \text{read(v}_2, v_3), \]
\[ v_3 = b, \]
\[ v_4 = v_1 \]

EUF
\[ v_6 \equiv \text{f(v}_4), \]
\[ v_7 \equiv \text{f(v}_5), \]
\[ v_6 \neq v_7, \]
\[ v_3 = b, \]
\[ v_4 = v_1, \]
\[ v_5 = v_1, \]
\[ v_6 = v_7 \]

Unsatisfiable
Deterministic procedure may fail for non-convex theories.

\[ 0 \leq a \leq 1, \ 0 \leq b \leq 1, \ 0 \leq c \leq 1, \]

\[ f(a) \neq f(b), \]

\[ f(a) \neq f(c), \]

\[ f(b) \neq f(c) \]
Combining Procedures in Practice

Propagate all implied equalities.
- Deterministic Nelson-Oppen.
- Complete only for convex theories.
- It may be expensive for some theories.

Delayed Theory Combination.
- Nondeterministic Nelson-Oppen.
- Create set of interface equalities \((x = y)\) between shared variables.
- Use SAT solver to guess the partition.
- Disadvantage: the number of additional equality literals is quadratic in the number of shared variables.
Combining Procedures in Practice

Common to these methods is that they are **pessimistic** about which equalities are propagated.

**Model-based Theory Combination**

- **Optimistic approach.**
- Use a candidate model \( M_i \) for one of the theories \( \mathcal{T}_i \) and propagate all equalities implied by the candidate model, hedging that other theories will agree.

\[
\text{if } M_i \models \mathcal{T}_i \cup \Gamma_i \cup \{u = v\} \text{ then propagate } u = v .
\]

- If not, use backtracking to fix the model.
- It is cheaper to enumerate equalities that are implied in a particular model than of all models.
Example

\[ x = f(y - 1), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1 \]

Purifying
Example

\[ x = f(z), \quad f(x) \neq f(y), \quad 0 \leq x \leq 1, \quad 0 \leq y \leq 1, \quad z = y - 1 \]
<table>
<thead>
<tr>
<th>( T_E )</th>
<th>( T_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literal</strong></td>
<td><strong>Eq. Classes</strong></td>
</tr>
<tr>
<td>( x = f(z) )</td>
<td>( {x, f(z)} )</td>
</tr>
<tr>
<td>( f(x) \neq f(y) )</td>
<td>( {y} )</td>
</tr>
<tr>
<td></td>
<td>( {z} )</td>
</tr>
<tr>
<td></td>
<td>( {f(x)} )</td>
</tr>
</tbody>
</table>

Assume \( x = y \)
<table>
<thead>
<tr>
<th>( \mathcal{T}_E )</th>
<th>( \mathcal{T}_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literal</strong>s</td>
<td><strong>Eq. Classes</strong></td>
</tr>
<tr>
<td>( x = f(z) )</td>
<td>( {x, y, f(z)} )</td>
</tr>
<tr>
<td>( f(x) \neq f(y) )</td>
<td>( {z} )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( {f(x), f(y)} )</td>
</tr>
<tr>
<td>( E(f) = {*_1 \leftrightarrow *_3, )</td>
<td>( )</td>
</tr>
<tr>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>( )</td>
<td>( )</td>
</tr>
</tbody>
</table>

**Unsatisfiable**
### Example

<table>
<thead>
<tr>
<th>$T_E$</th>
<th>$T_A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>** Literals</td>
<td>Eq. Classes</td>
</tr>
<tr>
<td>$x = f(z)$</td>
<td>${x, f(z)}$</td>
</tr>
<tr>
<td>$f(x) \neq f(y)$</td>
<td>${y}$</td>
</tr>
<tr>
<td>$x \neq y$</td>
<td>${z}$</td>
</tr>
<tr>
<td></td>
<td>${f(x)}$</td>
</tr>
<tr>
<td></td>
<td>${f(y)}$</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Backtrack, and assert $x \neq y$.

$T_A$ model need to be fixed.
<table>
<thead>
<tr>
<th>$\mathcal{T}_E$</th>
<th>$\mathcal{T}_A$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Literals</strong></td>
<td><strong>Literals</strong></td>
</tr>
<tr>
<td>$x = f(z)$</td>
<td>$0 \leq x \leq 1$</td>
</tr>
<tr>
<td>$f(x) \neq f(y)$</td>
<td>$0 \leq y \leq 1$</td>
</tr>
<tr>
<td>$x \neq y$</td>
<td>$z = y - 1$</td>
</tr>
<tr>
<td><strong>Eq. Classes</strong></td>
<td><strong>Model</strong></td>
</tr>
<tr>
<td>${x, f(z)}$</td>
<td>$E(x) = *_1$,</td>
</tr>
<tr>
<td>${y}$</td>
<td>$E(y) = *_2$,</td>
</tr>
<tr>
<td>${z}$</td>
<td>$E(z) = *_3$,</td>
</tr>
<tr>
<td>${f(x)}$</td>
<td>$E(f) = {*_1 \mapsto *_4,$</td>
</tr>
<tr>
<td>${f(y)}$</td>
<td>$*_2 \mapsto *_5,$</td>
</tr>
<tr>
<td></td>
<td>$*_3 \mapsto *_1,$</td>
</tr>
<tr>
<td></td>
<td>$\textit{else} \mapsto *_6}$</td>
</tr>
</tbody>
</table>

Assume $x = z$
<table>
<thead>
<tr>
<th>Literals</th>
<th>Eq. Classes</th>
<th>Model</th>
<th>Literals</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x = f(z))</td>
<td>({x, z, f(x), f(z)})</td>
<td>(E(x) = *_1)</td>
<td>(0 \leq x \leq 1)</td>
<td>(A(x) = 0)</td>
</tr>
<tr>
<td>(f(x) \neq f(y))</td>
<td>({y})</td>
<td>(E(y) = *_2)</td>
<td>(0 \leq y \leq 1)</td>
<td>(A(y) = 1)</td>
</tr>
<tr>
<td>(x \neq y)</td>
<td>({f(y)})</td>
<td>(E(z) = *_1)</td>
<td>(z = y - 1)</td>
<td>(A(z) = 0)</td>
</tr>
<tr>
<td>(x = z)</td>
<td></td>
<td>(E(f) = {*_1 \mapsto *_1,)</td>
<td>(x \neq y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(*_2 \mapsto *_3,)</td>
<td>(x = z)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(else \mapsto *_4})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Satisfiable**
Let $h$ be the bijection between $|E|$ and $|A|$.

$$h = \{ *_1 \mapsto 0, *_2 \mapsto 1, *_3 \mapsto -1, *_4 \mapsto 2, \ldots \}$$
\begin{center}
\begin{tabular}{ | c | c | c | c | c |}
\hline
\multicolumn{2}{ | c | }{\mathcal{T}_E} & \multicolumn{2}{ | c | }{\mathcal{T}_A} \\
\hline
\textbf{Literals} & \textbf{Model} & \textbf{Literals} & \textbf{Model} \\
\hline
\(x = f(z)\) & \(E(x) = *_1\) & \(0 \leq x \leq 1\) & \(A(x) = 0\) \\
\(f(x) \neq f(y)\) & \(E(y) = *_2\) & \(0 \leq y \leq 1\) & \(A(y) = 1\) \\
x \neq y & \(E(z) = *_1\) & \(z = y - 1\) & \(A(z) = 0\) \\
x = z & \(E(f) = \{*_1 \mapsto *_1,\) & \(x \neq y\) & \(A(f) = \{0 \mapsto 0\) \\
& \(*_2 \mapsto *_3,\) & \(x = z\) & \(1 \mapsto -1\) \\
& \text{else} \mapsto *_4\} & \text{else} \mapsto 2\} \\
\hline
\end{tabular}
\end{center}

Extending \(A\) using \(h\).

\[h = \{*_1 \mapsto 0, *_2 \mapsto 1, *_3 \mapsto -1, *_4 \mapsto 2, \ldots\}\]
Bit-vector theory is not stably-infinite.

How can we support it?

**Solution:** add a predicate $is-bv(x)$ to the bit-vector theory (intuition: $is-bv(x)$ is true iff $x$ is a bitvector).

The result of the bit-vector operation $op(x, y)$ is not specified if $\neg is-bv(x)$ or $\neg is-bv(y)$.

The new bit-vector theory is stably-infinite.
A **reduction function** reduces the satisfiability problem for a complex theory into the satisfiability problem of a simpler theory.

Ackermannization is a reduction function.
Theory of commutative functions.

- $\forall x, y. f(x, y) = f(y, x)$
- Reduction to EUF
- For every $f(a, b)$ in $\phi$, do $\phi := \phi \land f(a, b) = f(b, a)$. 
Applications

- Test case generation
- Verifying Compilers
- Predicate Abstraction
- Invariant Generation
- Type Checking
- Model Based Testing
A formula $F$ is valid

Iff

$\neg F$ is unsatisfiable
A formula $F$ is valid
iff
$\neg F$ is unsatisfiable.
Verification/Analysis Tool: “Template”

Problem

Verification/Analysis Tool

Logical Formula

Theorem Prover/Satisfiability Checker

Satisfiable (Counter-example)

Unsatisfiable
Z3 is a new solver developed at Microsoft Research.
Development/Research driven by internal customers.
Free for academic research.
Interfaces:

http://research.microsoft.com/projects/z3
Test case generation
Test case generation

Test (correctness + usability) is 95% of the deal:
- Dev/Test is 1-1 in products.
- Developers are responsible for unit tests.

Tools:
- Annotations and static analysis (SAL + ESP)
- File Fuzzing
- Unit test case generation
Security is critical

- Security bugs can be very expensive:
  - Cost of each MS Security Bulletin: $600k to $Millions.
  - Cost due to worms: $Billions.
  - The real victim is the customer.
- Most security exploits are initiated via files or packets.
  - Ex: Internet Explorer parses dozens of file formats.
- Security testing: hunting for million dollar bugs
  - Write A/V
  - Read A/V
  - Null pointer dereference
  - Division by zero
Two main techniques used by “black hats”:
- Code inspection (of binaries).
- Black box fuzz testing.

Black box fuzz testing:
- A form of black box random testing.
- Randomly *fuzz* (=modify) a well formed input.
- Grammar-based fuzzing: rules to encode how to fuzz.

Heavily used in security testing
- At MS: several internal tools.
- Conceptually simple yet effective in practice
Directed Automated Random Testing (DART)

Run Test and Monitor → Execution Path → Constraint System → Known Paths → Path Condition

seed → Test Inputs → New input → Solve → Z3
DARTish projects at Microsoft

PEX  Implements DART for .NET.

SAGE  Implements DART for x86 binaries.

YOGI  Implements DART to check the feasibility of program paths generated statically.

Vigilante  Partially implements DART to dynamically generate worm filters.
Test input generator

- Pex starts from parameterized unit tests
- Generated tests are emitted as traditional unit tests
ArrayList: The Spec

.NET Framework Class Library

ArrayList.Add Method

Adds an object to the end of the ArrayList.

Namespace: System.Collections
Assembly: mscorlib (in mscorlib.dll)

Remarks
ArrayList accepts a null reference (Nothing in Visual Basic) as a valid value and allows duplicate elements.

If Count already equals Capacity, the capacity of the ArrayList is increased by automatically reallocating the internal array, and the existing elements are copied to the new array before the new element is added.

If Count is less than Capacity, this method is an O(1) operation. If the capacity needs to be increased to accommodate the new element, this method becomes an O(n) operation, where n is Count.
```csharp
class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item;
    }
    ...
}

class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.Add(item);
        Assert(list[0] == item);
    }
}
```
class ArrayListTest {
[PexMethod]
void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.Add(item);
    Assert(list[0] == item);
}
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item;
    }
...
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.Add(item);
        Assert(list[0] == item);
    }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length) 
            ResizeArray();

        items[this.count++] = item;
    }
...
ArrayList: Run 1, (0,null)

class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.AddItem(item);
        Assert(list[0] == item);
    }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item;
    }
...
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.AddItem(item);
        Assert(list[0] == item);
    }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item;
    }
    ...

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Observed Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,null)</td>
<td>!(c&lt;0) &amp;&amp; 0==c</td>
</tr>
</tbody>
</table>
class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item; }

    ...}

class ArrayListTest {
[PexMethod]
void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.Add(item);
    Assert(list[0] == item); }

...}

Inputs | Observed Constraints
--- | ---
(0, null) | !(c<0) && 0==c

item == item → true

This is a tautology, i.e. a constraint that is always true, regardless of the chosen values. We can ignore such constraints.
class ArrayListTest {
[PexMethod]
void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.Add(item);
    Assert(list[0] == item); }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item;
    }
    ...

Constraints to solve

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Observed Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,null)</td>
<td>!(c&lt;0) &amp;&amp; 0==c</td>
</tr>
<tr>
<td>!(c&lt;0) &amp;&amp; 0!=c</td>
<td></td>
</tr>
</tbody>
</table>

ArrayList: Picking the next branch to cover
class ArrayListTest {
[PexMethod]
void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.AddItem(item);
    Assert(list[0] == item);
}
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item; } ...

<table>
<thead>
<tr>
<th>Constraints to solve</th>
<th>Inputs</th>
<th>Observed Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,null)</td>
<td>!(c&lt;0) &amp;&amp; 0==c</td>
<td></td>
</tr>
<tr>
<td>!(c&lt;0) &amp;&amp; 0!=c</td>
<td>(1,null)</td>
<td></td>
</tr>
</tbody>
</table>
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        listaddItem(item);
        Assert(list[0] == item);}
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item; }

        ...
class ArrayListTest {
[PexMethod]
void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.Add(item);
    Assert(list[0] == item);
}
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();

        items[this.count++] = item;
    }
    ...

class ArrayListTest {
[PexMethod]
void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.AddItem(item);
    Assert(list[0] == item); }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item; }
...
class ArrayListTest {
[PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.AddItem(item);
        Assert(list[0] == item); }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item; }
    ...
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.Add(item);
        Assert(list[0] == item); }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item; }
...

<table>
<thead>
<tr>
<th>Constraints to solve</th>
<th>Inputs</th>
<th>Observed Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(c&lt;0) &amp;&amp; 0==c</td>
<td>(0,null)</td>
<td>!(c&lt;0) &amp;&amp; 0==c</td>
</tr>
<tr>
<td>!(c&lt;0) &amp;&amp; 0!=c</td>
<td>(1,null)</td>
<td>!(c&lt;0) &amp;&amp; 0!=c</td>
</tr>
<tr>
<td>c&lt;0</td>
<td>(-1,null)</td>
<td>c&lt;0</td>
</tr>
</tbody>
</table>

Once again, Pex blows my mind. It's utterly amazing the bugs that it can find :).
White box testing in practice

How to test this code?
(Real code from .NET base class libraries.)
White box testing in practice

```csharp
private void ReadResources()
{
    BCLDebug.Assert(_store != null, "ResourceReader is closed!");
    BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File |
#if !FEATUREPAL
    _typeLimitingBinder = new TypeLimitingDeserializationBinder();
    bf.Binder = _typeLimitingBinder;
#endif
    _objFormatter = bf;
    try {
        // Read ResourceManager header
        // Check for magic number
        int magicNum = _store.ReadInt32();
        if (magicNum != ResourceManager.MagicNumber)
            throw new ArgumentException(ResourceManager.GetResourceString("Resources_StreamNotValid"));
        // Assuming this is ResourceManager header 1 or greater, hopefully
        // after the version number there is a number of bytes to skip
        // to bypass the rest of the ResMgr header.
        int resMgrHeaderVersion = _store.ReadInt32();
        if (resMgrHeaderVersion > 1) {
            int numBytesToSkip = _store.ReadInt32();
            BCLDebug.Log(_store, "RESMGRFILEFORMAT", LogLevel.Status, "ReadResources: Unexpected ResMgr header version: {}", resMgrHeaderVersion);
            BCLDebug.Assert(numBytesToSkip >= 0, "numBytesToSkip in ResMgr header should be positive!");
            _store.BaseStream.Seek(numBytesToSkip, SeekOrigin.Current);
        } else {
            BCLDebug.Log(_store, "RESMGRFILEFORMAT", "ReadResources: Parsing ResMgr header v1.");
            SkipInt32(); // We don't care about numBytesToSkip.
            // Read in type name for a suitable ResourceReader
            // Not ResourceWriter & InternalData use different schemas
        }
    }
```
// Reads in the header information for a .resources file. Verifies some
// of the assumptions about this resource set, and builds the class table
// for the default resource file format.
private void ReadResources()
{
    BCLDebug.Assert(_store != null, "ResourceReader is closed!");
    BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File |
    #if !FEATURE_PAL
        _typeLimitingBinder = new TypeLimitingDeserializationBinder();
        bf.Binder = _typeLimitingBinder;
    #endif

    _objFormatter = bf;
    try {
        // Read ResourceManager header
        // Check for magic number
        int magicNum = _store.ReadInt32();
        if (public virtual int ReadInt32() {
            if (m_isMemoryStream) {
                // read directly from MemoryStream buffer
                // MemoryStream mStream = m_stream as MemoryStream;
                BCLDebug.Assert(mStream != null, "m_stream as MemoryStream != null");
                int readCount = mStream.Read(m_buffer, 0, m_buffer.Length);
                if (readCount == 0)
                {
                    FillBuffer(4);
                }
            }
        }
    }
    // Read in type name for a suitable ResourceReader
    // Note: ResourceWriter & ResourceDesc use different formats.
Pex — Test Input Generation

```csharp
public class ResourceReaderTests
{
    [PexTest]
    public unsafe void ParameterizedTest(byte[] a)
    {
        PexAssume.IsNotNull(a);
        fixed (byte* p = a)
        using (stream = new UnmanagedMemoryStream(p, a.Length))
        {
            var reader = new ResourceReader(stream);
            readEntries(reader);
        }
    }
}
```

Test input, generated by Pex
```
byte[] a = new byte[14];
a[0] = 206;
a[1] = 202;
a[2] = 239;
a[3] = 190;
a[7] = 64;
```

ParameterizedTest(a);
Test Input Generation by Dynamic Symbolic Execution

- Test Inputs
- Constraint System
- Known Paths
- Execution Path

Initially, choose an arbitrary Test Input.

Result: small test suite, high code coverage

Finds only real bugs, no false warnings
Rich Combination

Linear arithmetic

Bitvector

Arrays

Free Functions

Models

Model used as test inputs

∀-Quantifier

Used to model custom theories (e.g., .NET type system)

API

Huge number of small problems. Textual interface is too inefficient.
Undecidable (in general)
Undecidable (in general)

Solution:
Return “Candidate” Model
Check if trace is valid by executing it
Undecidable (in general)

Refined solution:
Support for decidable fragments.
Apply DART to large applications (not units).
Start with well-formed input (not random).
Combine with generational search (not DFS).
  Negate 1-by-1 each constraint in a path constraint.
  Generate many children for each parent run.
Apply DART to large applications (not units).
Start with well-formed input (not random).
Combine with generational search (not DFS).
  Negate 1-by-1 each constraint in a path constraint.
  Generate many children for each parent run.
Starting with 100 zero bytes ...

SAGE generates a crashing test for Media1 parser

<table>
<thead>
<tr>
<th>Hex</th>
<th>Value</th>
<th>Hex</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000100h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>00000020h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000030h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>00000040h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td>00000050h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>00000060h</td>
<td>00 00 00 00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generation 0 – seed file
Zero to Crash in 10 Generations

- Starting with 100 zero bytes ...
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...** ...
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 B2 75 76 3A 28 00 00 00 ; ....strf²uv:(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; ................
00000060h: 00 00 00 00
```

Generation 10 – CRASH
SAGE (cont.)

- SAGE is very effective at finding bugs.
- Works on large applications.
- Fully automated
- Easy to deploy (x86 analysis – any language)
- Used in various groups inside Microsoft
- Powered by Z3.
Formulas are usually big conjunctions.

SAGE uses only the bitvector and array theories.

Pre-processing step has a huge performance impact.
  - Eliminate variables.
  - Simplify formulas.

Early unsat detection.
Static Driver Verifier

SLAM

```cpp
if (node->x(); i++) {
    proc() end();
    node();
```
Z3 is part of SDV 2.0 (Windows 7)

It is used for:

- Predicate abstraction (c2bp)
- Counter-example refinement (newton)
Overview

- http://research.microsoft.com/slam/
- *SLAM/SDV* is a software model checker.
- Application domain: *device drivers*.
- Architecture:
  - **c2bp** C program → boolean program (*predicate abstraction*).
  - **bebop** Model checker for boolean programs.
  - **newton** Model refinement (check for path feasibility)
- SMT solvers are used to perform predicate abstraction and to check path feasibility.
- c2bp makes several calls to the SMT solver. The formulas are relatively small.
do {
    KeAcquireSpinLock();

    nPacketsOld = nPackets;

    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
do {
    KeAcquireSpinLock();

    if(*){
        KeReleaseSpinLock();
    }
} while (*);

KeReleaseSpinLock();
do {
    KeAcquireSpinLock();
    nPacketsOld = nPackets;
    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
} while (nPackets != nPacketsOld);
KeReleaseSpinLock();
do {
    KeAcquireSpinLock();

    nPacketsOld = nPackets;
    b = true;

    if (request) {
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
    b = b ? false : *;
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
do {
    KeAcquireSpinLock();

    b = true;

    if(*){
        KeReleaseSpinLock();
        b = b ? false : *;
    }
} while (!b);

KeReleaseSpinLock();
do {
    KeAcquireSpinLock();
    b = true;
    if(*){
        KeReleaseSpinLock();
        b = b ? false : *;
    }
} while (!b);

KeReleaseSpinLock();
Example

do {
    KeAcquireSpinLock();
    b = true;
    if(*){
        KeReleaseSpinLock();
        b = b ? false : *;
    }
} while (!b);

KeReleaseSpinLock();
Automatic discovery of invariants
- driven by property and a finite set of (false) execution paths
- predicates are *not* invariants, but *observations*
- abstraction + model checking computes inductive invariants (Boolean combinations of observations)

A hybrid dynamic/static analysis
- newton executes path through C code symbolically
- c2bp+bebop explore all paths through abstraction

A new form of program slicing
- program code and data not relevant to property are dropped
- non-determinism allows slices to have more behaviors
Predicate Abstraction: $c2bp$

- **Given** a C program $P$ and $F = \{p_1, \ldots, p_n\}$.
- **Produce** a Boolean program $B(P, F)$
  - Same control flow structure as $P$.
  - Boolean variables $\{b_1, \ldots, b_n\}$ to match $\{p_1, \ldots, p_n\}$.
  - Properties true in $B(P, F)$ are true in $P$.
- Each $p_i$ is a pure Boolean expression.
- Each $p_i$ represents set of states for which $p_i$ is true.
- Performs modular abstraction.
Abstracting Expressions via $F$

- $\text{Implies}_F (e)$
  - Best Boolean function over $F$ that implies $e$.
- $\text{ImpliedBy}_F (e)$
  - Best Boolean function over $F$ that is implied by $e$.
  - $\text{ImpliedBy}_F (e) = \text{not Implies}_F (\text{not } e)$
$\text{Implies}_F(e)$ and $\text{ImpliedBy}_F(e)$
minterm \( m = l_1 \text{ and } ... \text{ and } l_n \), where \( l_i = p_i \), or \( l_i = \text{not } p_i \).

\( \text{Implies}_F(e) \): disjunction of all minterms that imply \( e \).

Naive approach

- Generate all \( 2^n \) possible minterms.
- For each minterm \( m \), use SMT solver to check validity of \( m \text{ implies } e \).

Many possible optimizations
Computing $\text{Implies}_F(e)$

- $F = \{ x < y, x = 2 \}$
- $e : y > 1$

Minterms over $F$
- $\neg x < y, \neg x = 2$ implies $y > 1$
- $x < y, \neg x = 2$ implies $y > 1$
- $\neg x < y, x = 2$ implies $y > 1$
- $x < y, x = 2$ implies $y > 1$
Computing $\text{Implies}_F(e)$

- $F = \{ x < y, x = 2 \}$
- $e : y > 1$

Minterms over $F$

- $\neg x < y, \neg x = 2 \implies y > 1$
- $x < y, \neg x = 2 \implies y > 1$
- $\neg x < y, x = 2 \implies y > 1$
- $x < y, x = 2 \implies y > 1$
Computing $\text{Implies}_F(e)$

- $F = \{ x < y, x = 2 \}$
- $e : y > 1$

Minterms over $F$

- $\neg x < y, \neg x = 2$ implies $y > 1$  
- $x < y, \neg x = 2$ implies $y > 1$  
- $\neg x < y, x = 2$ implies $y > 1$  
- $x < y, x = 2$ implies $y > 1$

$\text{Implies}_F(y > 1) = x < y \land x = 2$
Computing $\text{Implies}_F(e)$

- $F = \{ x < y, x = 2 \}$
- $e : y > 1$

Minterms over $F$

- $\neg x < y, \neg x = 2 \implies y > 1$
- $x < y, \neg x = 2 \implies y > 1$
- $\neg x < y, x = 2 \implies y > 1$
- $x < y, x = 2 \implies y > 1$

$\text{Implies}_F(y > 1) = b_1 \land b_2$
Given an error path \( p \) in the Boolean program \( B \).

Is \( p \) a feasible path of the corresponding C program?

- Yes: found a bug.
- No: find predicates that explain the infeasibility.

Execute path symbolically.

Check conditions for inconsistency using SMT solver.
All-SAT
- Better (more precise) Predicate Abstraction
- Unsatisfiable cores
- Why the abstract path is not feasible?
- Fast Predicate Abstraction
Bit-precise Scalable Static Analysis

PREfix [Moy, Bjorner, Sielaff 2009]
What is wrong here?

```c
int binary_search(int[] arr, int low, int high, int key)
while (low <= high)
{
    // Find middle value
    int mid = (low + high) / 2;
    int val = arr[mid];
    if (val == key) return mid;
    if (val < key) low = mid+1;
    else high = mid-1;
}
return -1;
```

```c
void itoa(int n, char* s) {
    if (n < 0) {
        *s++ = '-';
        n = -n;
    }
    // Add digits to s
    ....
}
```

Package: java.util.Arrays
Function: binary_search

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)
```c
int binary_search(int arr[], int low, int high, int key)
{
    while (low <= high)
    {
        // Find middle value
        int mid = (low + high) / 2;
        int val = arr[mid];
        if (val == key) return mid;
        if (val < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

void itoa(int n, char* s) {
    if (n < 0) {
        *s++ = '-';
        n = -n;
    }
    // Add digits to s
    ....
}
```

**Package:** java.util.Arrays  
**Function:** binary_search  
**Book:** Kernighan and Ritchie  
**Function:** itoa (integer to ascii)
int binary_search(int[] arr, int low, int high, int key) {
    while (low <= high) {
        int mid = (low + high) / 2;
        int val = arr[mid];
        if (val == key) return mid;
        if (val < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

void itoa(int n, char* s) {
    if (n < 0) {
        *s++ = '-';
        n = -n;
    }
    // Add digits to s
    ....
}

Package: java.util.Arrays
Function: binary_search

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)
int init_name(char **outname, uint n)
{
    if (n == 0) return 0;
    else if (n > UINT16_MAX) exit(1);
    else if ((*(outname = malloc(n))) == NULL) {
        return 0xC0000095; // NT_STATUS_NO_MEM;
    }
    return 0;
}

int get_name(char* dst, uint size)
{
    char* name;
    int status = 0;
    status = init_name(&name, size);
    if (status != 0) {
        goto error;
    }
    strcpy(dst, name);
error:
    return status;
}
```c
int init_name(char **outname, uint n) {
    if (n == 0) return 0;
    else if (n > UINT16_MAX) exit(1);
    else if ((*(outname = malloc(n))) == NULL) {
        return 0xC0000095; // NT_STATUS_NO_MEM;
    }
    return 0;
}

int get_name(char* dst, uint size) {
    char* name;
    int status = 0;
    status = init_name(&name, size);
    if (status != 0) {
        goto error;
    }
    strcpy(dst, name);
    return status;
}
```

**models**

**outcome init_name_0:**
- **guards:** $n = 0$
- **results:** result == 0

**outcome init_name_1:**
- **guards:** $n > 0; n <= 65535$
- **results:** result == 0xC0000095

**outcome init_name_2:**
- **guards:** $n > 0; n <= 65535$
- **constraints:** valid(outname)
- **results:** result == 0; init(*outname)
int init_name(char **outname, uint n)
{
    if (n == 0) return 0;
    else if (n > UINT16_MAX) exit(1);
    else if (((*outname = malloc(n)) == NULL) {
        return 0xC0000095; // NT_STATUS_NO_MEM;
    }
    return 0;
}

int get_name(char* dst, uint size)
{
    char* name;
    int status = 0;
    status = init_name(&name, size);
    if (status != 0) {
        goto error;
    }
    strcpy(dst, name);
    error:
    return status;
}
Overflow on unsigned addition

```cpp
iElement = m_nSize;
if( iElement >= m_nMaxSize )
{
    bool bSuccess = GrowBuffer( iElement+1 );
    ...
}
::new( m_pData+iElement ) E( element );
m_nSize++;
```

- `m_nSize == m_nMaxSize == UINT_MAX`
- `iElement + 1 == 0`

**Write in unallocated memory**

**Code was written for address space < 4GB**
ULONG AllocationSize;
while (CurrentBuffer != NULL) {
    if (NumberOfBuffers > MAX_ULONG / sizeof(MYBUFFER)) {
        return NULL;
    }
    NumberOfBuffers++;
    CurrentBuffer = CurrentBuffer->NextBuffer;
}
AllocationSize = sizeof(MYBUFFER)*NumberOfBuffers;
UserBuffersHead = malloc(AllocationSize);
Verifying Compilers

Annotated Program $\rightarrow$ Verification Condition $F$

pre/post conditions
invariants
and other annotations
class C {
    private int a, z;
    invariant z > 0

    public void M()
    {
        requires a != 0
        {
            z = 100/a;
        }
    }
}
Source Language
- C# + goodies = Spec#

Specifications
- method contracts,
- invariants,
- field and type annotations.

Program Logic:
- Dijkstra’s weakest preconditions.

Automatic Verification
- type checking,
- verification condition generation (VCG),
- SMT
**Command language**

- $x := E$
  - $x := x + 1$
  - $x := 10$
- $\text{havoc } x$
- $S ; T$
- $\text{assert } P$
- $\text{assume } P$
- $S \ Return \ T$
Hoare triple \( \{ P \} S \{ Q \} \) says that every terminating execution trace of S that starts in a state satisfying P does not go wrong, and terminates in a state satisfying Q.
Hoare triple \( \{ P \} S \{ Q \} \) says that every terminating execution trace of S that starts in a state satisfying P does not go wrong, and terminates in a state satisfying Q.

Given S and Q, what is the weakest \( P' \) satisfying \( \{ P' \} S \{ Q \} \)?

\( P' \) is called the *weakest precondition* of S with respect to Q, written \( \text{wp}(S, Q) \)

to check \( \{ P \} S \{ Q \} \), check \( P \Rightarrow P' \)
Weakest preconditions

\[
\begin{align*}
wp( x := E, Q ) &= Q[ E / x ] \\
wp( \text{havoc } x, Q ) &= (\forall x \cdot Q) \\
wp( \text{assert } P, Q ) &= P \land Q \\
wp( \text{assume } P, Q ) &= P \Rightarrow Q \\
wp( S ; T, Q ) &= wp( S, wp( T, Q ) ) \\
wp( S \mathrel{\boxplus} T, Q ) &= wp( S, Q ) \land wp( T, Q )
\end{align*}
\]
Structured if statement

if $E$ then $S$ else $T$ end =

assume $E$; $S$

assume $\neg E$; $T$
While loop with loop invariant

while E
  invariant J
  do
    S
  end

= assert J;
  havoc x; assume J;
  ( assume E; S; assert J; assume false
    □ assume ¬E
  )

where x denotes the assignment targets of S

check that the loop invariant holds initially

“fast forward” to an arbitrary iteration of the loop

check that the loop invariant is maintained by the loop body
procedure Chunker.NextChunk(this: ref where $IsNotNull(this, Chunker)) returns ($result: ref where $IsNotNull($result, System.String));

// in-parameter: target object
free requires $Heap[this, $allocated];
requires ($Heap[this, $ownerFrame] == $PeerGroupPlaceholder || !($Heap[$Heap[this, $ownerRef], $inv] <: $Heap[this, $ownerFrame]) ||

// out-parameter: return value
free ensures $Heap[$result, $allocated];
ensures ($Heap[$result, $ownerFrame] == $PeerGroupPlaceholder || !($Heap[$Heap[$result, $ownerRef], $inv] <: $Heap[$result, $ownerFrame]) ||

// user-declared postconditions
ensures $StringLength($result) <= $Heap[this, Chunker.ChunkSize];

// frame condition
modifies $Heap;
free ensures (forall $o: ref, $f: name :: { $Heap[$o, $f] } $f != $inv && $f != $localinv && $f != $FirstConsistentOwner && (!$IsStaticField($f)) ||
 !$IsDirectlyModifiableField($f)) && $o != null && old($Heap[$o, $allocated]) && (old($Heap][$o, $ownerFrame] == $PeerGroupPlaceholder ||
 !(old($Heap)[old($Heap][$o, $ownerRef], $inv] <: old($Heap[$o, $ownerFrame]) || old($Heap)[old($Heap][$o, $ownerRef], $localinv] ==
 $BaseClass(old($Heap[$o, $ownerFrame])) && old($o != this || !$Heap[$o] != null) && $f != $exposeVersion) => old($Heap[$o, $f] == $Heap[$o, $f]);

// boilerplate
free requires $BeingConstructed == null;
free ensures (forall $o: ref :: { $Heap[$o$0, $localinv] } { $Heap[$o$0, $inv] } $o != null && old($Heap[$o$0, $allocated]) && $Heap[$o$0, $allocated] =>
 $Heap[$o$0, $inv] == $typeof($o) && $Heap[$o$0, $localinv] == $typeof($o));
free ensures (forall $o: ref :: { $Heap[$o$0, $FirstConsistentOwner] } old($Heap)[old($Heap][$o$0, $FirstConsistentOwner], $exposeVersion] ==
 $Heap[old($Heap][$o$0, $FirstConsistentOwner], $exposeVersion] => old($Heap[$o$0, $FirstConsistentOwner] == $Heap[$o$0, $FirstConsistentOwner]);
free ensures (forall $o: ref :: { $Heap[$o$0, $localinv] } { $Heap[$o$0, $inv] } old($Heap[$o$0, $allocated]) => old($Heap[$o$0, $inv] == $Heap[$o$0, $inv] &&
 old($Heap[$o$0, $localinv] == $Heap[$o$0, $localinv]);
free ensures (forall $o: ref :: { $Heap[$o$0, $allocated] } old($Heap[$o$0, $allocated]) && (forall $ot: ref :: { $Heap[$ot, $ownerFrame] } { $Heap[$ot, $ownerRef] } old($Heap[$ot, $allocated]) && old($Heap[$ot, $ownerFrame] != $PeerGroupPlaceholder =>
 old($Heap[$ot, $ownerRef] == $Heap[$ot, $ownerRef] && old($Heap[$ot, $ownerFrame] == $Heap[$ot, $ownerFrame]) &&
 old($Heap[$BeingConstructed, $NonNullFieldsAreInitialized]) == $Heap[$BeingConstructed, $NonNullFieldsAreInitialized];
Verification conditions: Structure

∀ Axioms (non-ground) + BIG and-or tree (ground) Control & Data Flow
Meta OS: small layer of software between hardware and OS

Mini: 100K lines of non-trivial concurrent systems C code

Critical: must provide functional resource abstraction

Trusted: a verification grand challenge
A partition cannot distinguish (with some exceptions) whether a machine instruction is executed

a) through the HV OR b) directly on a processor
Hypervisor Implementation

- real code, as shipped with Windows Server 2008
- ca. 100 000 lines of C, 5 000 lines of x64 assembly
- concurrency
  - spin locks, r/w locks, rundowns, turnstiles
  - lock-free accesses to volatile data and hardware covered by implicit protocols
- scheduler, memory allocator, etc.
- access to hardware registers (memory management, virtualization support)

Partners:
- European Microsoft Innovation Center
- Microsoft Research
- Microsoft’s Windows Div.
- Universität des Saarlandes

co-funded by the German Ministry of Education and Research

http://www.verisoftxt.de
Challenges for Verification of Concurrent C

1. **Memory model** that is adequate and efficient to reason about
2. **Modular reasoning** about concurrent code
3. **Invariants** for (large and complex) C data structures
4. Huge verification conditions to be proven **automatically**
5. “Live” specifications that **evolve with the code**
The Microsoft Verifying C Compiler (VCC)

- **Source Language**
  - ANSI C +
  - Design-by-Contract Annotations +
  - Ghost state +
  - Theories +
  - Metadata Annotations

- **Program Logic**
  - Dijkstra’s weakest preconditions

- **Automatic Verification**
  - verification condition generation (VCG)
  - automatic theorem proving (SMT)
VCC Architecture

```c
#include <vcc2.h>

typedef struct _BITMAP {
    UINT32 Size;     // Number of bits ...
    PUINT32 Buffer;  // Memory to store
 ...
// private invariants
invvariant(Size > 0 && Size % 32 == 0)
 ...
```

Annotated C

```c
$ref_cnt(olds, #p) == $ref_cnt(olds, #p)
&& $ite.bool($set_in(#p, $owns(olds, #p)),
            $ite.bool($set_in(#p, owns),
            $st_eq(olds, $s, #p),
            $wrapped($s, #p, $typ(#p)) &&
            $timestamp_is_now($s, #p)),
            $ite.bool($set_in(#p, owns),
            $owner($s, #p) == owner && $closed($s,
                ...
```

Generated Boogie

```boogie
:assumption
(forall (?x Int) (?y Int)
    (iff
        (= (IntEqual ?x ?y) boolTrue
        (= ?x ?y)))
:formula
    (...
```

Boogie

```boogie
owner)),
    $ite.bool($set_in(#p, owns),
    $st_eq(olds, $s, #p),
    $wrapped($s, #p, $typ(#p)) &&
    $timestamp_is_now($s, #p)),
    $ite.bool($set_in(#p, owns),
    $owner($s, #p) == owner &&
    $closed($s,
```

SMT

```z3
```

Available at [http://vcc.codeplex.com/](http://vcc.codeplex.com/)
int foo(int x)
    requires(x > 5) // precond
    ensures(result > x) // postcond
{
    ... 
}

void bar(int y; int *z)
    writes(z) // framing
    requires(y > 7)
    maintains(*z > 7) // invariant
{
    *z = foo(y);
    assert(*z > 7);
}

• function contracts: pre-/postconditions, framing
• modularity: bar only knows contract (but not code) of foo

advantages:
• modular verification: one function at a time
• no unfolding of code: scales to large applications
VCs have several Mb
Thousand of non ground clauses
Developers are willing to wait at most 5 min per VC
VCs have several Mb
Thousands of non ground clauses
Developers are willing to wait at most 5 min per VC

Are you willing to wait more than 5 min for your compiler?
Verification Attempt Time vs. Satisfaction and Productivity

By Michal Moskal (VCC Designer and Software Verification Expert)
1. My annotations are not strong enough!
   weak loop invariants and/or contracts

2. My theorem prover is not strong (or fast) enough.
   Send “angry” email to Nikolaj and Leo.
Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime

\[ \forall h, o, f: \]
\[ \text{IsHeap}(h) \land o \neq \text{null} \land \text{read}(h, o, \text{alloc}) = t \]
\[ \Rightarrow \]
\[ \text{read}(h, o, f) = \text{null} \lor \text{read}(h, \text{read}(h, o, f), \text{alloc}) = t \]
Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms

\[ \forall o, f: \]
\[ o \neq \text{null} \land \text{read}(h_0, o, \text{alloc}) = t \Rightarrow \]
\[ \text{read}(h_1, o, f) = \text{read}(h_0, o, f) \lor (o, f) \in M \]
Quantifiers, quantifiers, quantifiers, ...

Modeling the runtime

Frame axioms

User provided assertions

∀ i,j: i ≤ j ⇒ read(a,i) ≤ read(b,j)
Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms
- User provided assertions
- Theories
  - $\forall x: p(x,x)$
  - $\forall x,y,z: p(x,y), p(y,z) \implies p(x,z)$
  - $\forall x,y: p(x,y), p(y,x) \implies x = y$
Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms
- User provided assertions
- Theories
- Solver must be fast in satisfiable instances.

We want to find bugs!
Bad news

There is no sound and refutationally complete procedure for linear integer arithmetic + free function symbols
Many Approaches

- Heuristic quantifier instantiation
- Combining SMT with Saturation provers
- Complete quantifier instantiation
- Decidable fragments
- Model based quantifier instantiation
Is the axiomatization of the runtime consistent? False implies everything. Partial solution: SMT + Saturation Provers. Found many bugs using this approach.
Challenge: Robustness

- Standard complain
  "I made a small modification in my Spec, and Z3 is timingout"

- This also happens with SAT solvers (NP-complete)

- In our case, the problems are undecidable

- Partial solution: parallelization
Joint work with Y. Hamadi (MSRC) and C. Wintersteiger

Multi-core & Multi-node (HPC)

Different strategies in parallel

Collaborate exchanging lemmas
Z3 may be buggy.

Solution: proof/certificate generation.

Engineering problem: these certificates are too big.
Z3 may be buggy.

Solution: proof/certificate generation.

Engineering problem: these certificates are too big.

The Axiomatization of the runtime may be buggy or inconsistent.

Yes, this is true. We are working on new techniques for proving satisfiability (building a model for these axioms)
Hey, I don’t trust these proofs

Z3 may be buggy.
   Solution: proof/certificate generation.
   Engineering problem: these certificates are too big.

The Axiomatization of the runtime may be buggy or inconsistent.
   Yes, this is true. We are working on new techniques for proving satisfiability (building a model for these axioms)

The VCG generator is buggy (i.e., it makes the wrong assumptions)
   Do you trust your compiler?
These are bug-finding tools!

When they return “Proved”, it just means they cannot find more bugs.

I add Loop invariants to speed up the process.

I don’t want to waste time analyzing paths with 1, 2, ..., k, ...

They are successful if they expose bugs not exposed by regular testing.
Conclusion

Powerful, mature, and versatile tools like SMT solvers can now be exploited in very useful ways.

The construction and application of satisfiability procedures is an active research area with exciting challenges.

SMT is hot at Microsoft.

Z3 is a new SMT solver.

Main applications:

- Test-case generation.
- Verifying compiler.
- Model Checking & Predicate Abstraction.
Books

- Bradley & Manna: The Calculus of Computation
- Kroening & Strichman: Decision Procedures, An Algorithmic Point of View
- Chapter in the Handbook of Satisfiability
Z3:
http://research.microsoft.com/projects/z3
http://research.microsoft.com/~leonardo
  ▶ Slides & Papers
http://www.smtlib.org
http://www.smtcomp.org
References


References


