Satisfiability Modulo Theories (SMT): ideas and applications

Università Degli Studi Di Milano
Scuola di Dottorato in Informatica, 2010

Leonardo de Moura
Microsoft Research
Software development crisis

Software malfunction is a common problem.

Software complexity is increasing.

We need new methods and tools.
I proved my program to be correct.

What does it mean?
We need **models** and **tools** to reason about them?

Does my model/software have property X?
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
- Hyper-V Virtualization
- Terminator T-2
- VCC
- NModel
- SLAM
- SpecExplorer
- SAGE
- Vigilante
- Yogi
- F7
- Pex
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.

(y_0 > 0) and
(m_0 = x_0 \% y_0) and
not (m_0 = 0) and
(x_1 = y_0) and
(y_1 = m_0) and
(m_1 = x_1 \% y_1) and
(m_1 = 0)

x_0 = 2
y_0 = 4
m_0 = 2
x_1 = 4
y_1 = 2
m_1 = 0
Signature:
\[ \text{div} : \text{int}, \{ x : \text{int} | x \neq 0 \} \rightarrow \text{int} \]

Call site:
if \( a \leq 1 \) and \( a \leq b \) then
\[ \text{return div}(a, b) \]

Verification condition
\( a \leq 1 \) 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.
What is logic?

- 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

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

\[ p \lor q \Rightarrow q \]

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

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\phi & A & B & \neg A & A \lor B & A \land \neg A & A \Rightarrow B & A \Rightarrow (B \lor A) \\
\hline
\mathcal{M}_1(\phi) & \bot & \bot & T & \bot & \bot & T & T \\
\mathcal{M}_2(\phi) & \bot & T & T & T & \bot & T & T \\
\mathcal{M}_3(\phi) & T & \bot & \bot & T & \bot & \bot & T \\
\mathcal{M}_4(\phi) & T & T & \bot & T & \bot & T & T \\
\hline
\end{array}
\]
Satisfiability & Validity: examples

\( p \lor q \Rightarrow q \lor p \)  \hspace{1cm} \text{VALID}

\( p \lor q \Rightarrow q \) \hspace{1cm} \text{SATISFIABLE}

\( p \land \neg q \land (\neg p \lor q) \) \hspace{1cm} \text{UNSATISFIABLE}

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\phi & A & B & \neg A & A \lor B & A \land \neg A & A \Rightarrow B & A \Rightarrow (B \lor A) \\
\hline
M_1(\phi) & \bot & \bot & T & \bot & \bot & T & T \\
M_2(\phi) & \bot & T & T & T & \bot & T & T \\
M_3(\phi) & T & \bot & \bot & T & \bot & \bot & T \\
M_4(\phi) & T & T & \bot & T & \bot & T & T \\
\hline
\end{array}
\]
Equivalence

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. $\neg A \Rightarrow B \iff \neg B \Rightarrow A$
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 $\vee$.***
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. \(\neg \neg A \iff A\)

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

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

\[\iff\]

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

\[\iff\]

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

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

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

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

\(((p \land s) \lor (\neg q \land r)) \land (q \lor \neg p \lor s) \land (\neg r \lor s)\)
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. \[A \land (B \lor C) \iff (A \land B) \lor (A \land C)\]
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. \(A \land (B \lor C) \iff (A \land B) \lor (A \land C)\)
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

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

\iff

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

DNF?

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

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

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

Distributivity

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

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

\[ \equiv \]

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

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. \( A \lor \bot \iff A \)
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 decision procedure determines if a collection of formulas is satisfiable.

A decision procedure is given by a collection of reduction rules on a logical state $\psi$.

State $\psi$ is of the form $\kappa_1|\ldots|\kappa_n$, where each $\kappa_i$ is a configuration.

The logical content of $\kappa$ is either $\bot$ or is given by a finite set of formulas of the form $A_1, \ldots, A_m$.

A state $\psi$ of the form $\kappa_1, \ldots, \kappa_n$ is satisfiable if some configuration $\kappa_i$ is satisfiable.

A configuration $\kappa$ of the form $A_1, \ldots, A_m$ is satisfiable if there is an interpretation $M$ such that $M|=$ $A_i$ for $1 \leq i \leq m$. 
Inference Systems for Decision Procedures

A refutation procedure proves $A$ by refuting $\neg A$ through the application of reduction rules.

An application of an reduction rule transforms a state $\psi$ to a state $\psi'$ (written $\psi \Rightarrow \psi'$).

Rules preserve satisfiability.

If relation $\Rightarrow$ between states is well-founded and any non-bottom irreducible state is satisfiable, we say that the inference system is a decision procedure.

Ex: Prove that a decision procedure as given above is sound and complete.
An inference rule \( \frac{k}{\kappa_1| \ldots |\kappa_n} \) is shorthand for \( \frac{\psi[k]}{\psi[\kappa_1| \ldots |\kappa_n]} \).

The truth table procedure can be viewed as a model elimination procedure.

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

\[
\frac{\Gamma, F}{\perp} \quad \text{elim} \quad F \text{ is falsified by the literals in } \Gamma.
\]

A literal is a proposition or the negation of a proposition.

The literals in \( \Gamma \) can be viewed as a partial interpretation.

Ex: Prove correctness (soundness, termination, and completeness).
A truth table refutation of \( \{ p \lor \neg q \lor \neg r, p \lor r, p \lor q, \neg p \} \):

\[
\begin{array}{cccc}
p \lor \neg q \lor \neg r, & p \lor r, & p \lor q, & \neg p \\

p \lor \neg q \lor \neg r, & p \lor r, & p \lor q, & \neg p, q |
\end{array}
\]

\[
\begin{array}{cccc}
p \lor \neg q \lor \neg r, & p \lor r, & p \lor q, & \neg p, q \\

p \lor \neg q \lor \neg r, & p \lor r, & p \lor q, & \neg p, q, \neg r
\end{array}
\]

Ex: Implement the truth table procedure.
The inference rules for the Semantic Tableaux procedure are:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{A \land B, \Gamma}{A, B, \Gamma} ) (\land^+)</td>
<td>(\neg(A \land B), \Gamma ) (\land^-)</td>
<td></td>
</tr>
<tr>
<td>( \frac{\neg(A \lor B), \Gamma}{\neg A, \neg B, \Gamma} ) (\lor^-)</td>
<td>( \frac{(A \lor B), \Gamma}{A, \Gamma</td>
<td>B, \Gamma} ) (\lor^+)</td>
</tr>
<tr>
<td>( \frac{\neg(A \Rightarrow B), \Gamma}{A, \neg B, \Gamma} ) (\Rightarrow^-)</td>
<td>( \frac{(A \Rightarrow B), \Gamma}{\neg A, \Gamma</td>
<td>B, \Gamma} ) (\Rightarrow^+)</td>
</tr>
</tbody>
</table>
| \( \frac{\neg
\neg A, \Gamma}{A, \Gamma} \) \(\neg\) | \( \frac{A, \neg A, \Gamma}{\bot} \) \(\bot\) |

Semantic Tableaux is a “DNF translator”.

Ex: Prove correctness.
Refutation of \((p \vee q \Rightarrow q \vee p)\):

\[
\begin{array}{|c|c|}
\hline
A \land B, \Gamma & \neg(A \land B), \Gamma \\
\hline
A, B, \Gamma & \neg A, \Gamma, \neg B, \Gamma \\
\hline
\neg(A \lor B), \Gamma & (A \lor B), \Gamma \\
\hline
\neg A, \neg B, \Gamma & (A \Rightarrow B), \Gamma \\
\hline
A, \neg B, \Gamma & (A \Rightarrow B), \Gamma \\
\hline
\neg A, \Gamma & (A \Rightarrow B), \Gamma \\
\hline
A, \neg A, \Gamma & \bot \\
\hline
\end{array}
\]

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

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

\[p, \neg(q \lor p) \mid q, \neg(q \lor p)\]

\[p, \neg q, \neg p \mid q, \neg(q \lor p)\]

\[\bot \mid q, \neg(q \lor p)\]

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

\[q, \neg q, \neg p\]

\[\bot\]

Ex: Use the Semantic Tableaux procedure to refute
\(\neg(p \lor (q \land r) \Rightarrow (p \lor q) \land (p \lor r))\).

Ex: Implement the Semantic Tableaux.
The complexity of *Semantic Tableaux* proofs depends on the *length* of the *formula* to be decided.

The complexity of the *truth-table* procedure depends only on the number of *distinct propositional variables* which occur in it.

The *Semantic Tableaux* procedure does not *p-simulate* the *truth-table* procedure. Consider *fat* formulas such as:

\[
(p_1 \lor p_2 \lor p_3) \land (\neg p_1 \lor p_2 \lor p_3) \land \\
(p_1 \lor \neg p_2 \lor p_3) \land (\neg p_1 \lor \neg p_2 \lor p_3) \land \\
(p_1 \lor p_2 \lor \neg p_3) \land (\neg p_1 \lor p_2 \lor \neg p_3) \land \\
(p_1 \lor \neg p_2 \lor \neg p_3) \land (\neg p_1 \lor \neg p_2 \lor \neg p_3)
\]

**Ex:** Use Semantic Tableaux to refute the formula above.
The classical notion of truth is governed by two basic principles:

**Non-contradiction** no proposition can be true and false at the same time.

**Bivalence** every proposition is either true of false.

There is *no rule* in the *Semantic Tableaux* procedure which corresponds to the *principle of bivalence*.

The elimination of the *principle of bivalence* seem to be inadequate from the point of view of efficiency.
The principle of bivalence can be recovered if we replace the Semantic Tableaux branching rules by:

\[
\begin{array}{c}
\frac{\neg(A \land B), \Gamma}{\neg A, \Gamma | A, \neg B, \Gamma} & \frac{\neg(A \land B), \Gamma}{\neg B, \Gamma | B, \neg A, \Gamma} \\
\frac{(A \lor B), \Gamma}{A, \Gamma | \neg A, B, \Gamma} & \frac{(A \lor B), \Gamma}{B, \Gamma | \neg B, A, \Gamma} \\
\frac{(A \Rightarrow B), \Gamma}{\neg A, \Gamma | A, B, \Gamma} & \frac{(A \Rightarrow B), \Gamma}{B, \Gamma | \neg B, \neg A, \Gamma}
\end{array}
\]

The new rules are asymmetric.

Ex: Show that the new rules are sound.
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>Rule</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>eliminate $\Rightarrow$</td>
<td>$A \Rightarrow B \equiv \neg A \lor B$</td>
</tr>
<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>
CNF (again)

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:

\[
\begin{align*}
F[l_i \text{ op } l_j] & \\
F[x], x \Leftrightarrow l_i \text{ op } l_j & \\
x \Leftrightarrow l_i \lor l_j & \\
-x \lor l_i \lor l_j, -l_i \lor x, -l_j \lor x & \\
x \Leftrightarrow l_i \land l_j & \\
-x \lor l_i, -x \lor l_j, -l_i \lor -l_j \lor x & \text{ (*))}
\end{align*}
\]

(*) $x$ 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:

\[ \frac{C_1 \lor p, C_2 \lor \neg p}{C_1 \lor p, C_2 \lor \neg p, C_1 \lor C_2} \]

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 $\neg p \lor \neg q \lor r$, $p \lor r$, $q \lor r$, $r$:

Ex: Implement a naïve resolution procedure.
Let $Res(S)$ be the closure of $S$ under the resolution rule.

Completeness: $S$ is unsatisfiable iff $Res(S)$ contains the empty clause.

Proof ($\Rightarrow$):

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

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

The semantic tree of $Res(S)$ 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 ($\Leftarrow$): $Res(S)$ 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.

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

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.

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

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

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 \((\neg p_1 \lor \ldots \lor \neg p_n \lor q \equiv p_1 \land \ldots \land p_n \Rightarrow q)\).

Positive unit rule:

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

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.
Remark: An interpretation $I$ can be used to divide an unsatisfiable set of clauses $S$.

Let $I$ be an interpretation, and $P$ an ordering on the propositional variables. A finite set of clauses \( \{E_1, \ldots, E_q, N\} \) is called a clash with respect to $P$ and $I$, if and only if:

- $E_1, \ldots, E_q$ are false in $I$.
- $R_1 = N$, for each $i = 1, \ldots, q$, there is a resolvent $R_{i+1}$ of $R_i$ and $E_i$.
- The literal in $E_i$, which is resolved upon, contains the largest propositional variable.
- $R_{q+1}$ is false in $I$. $R_{q+1}$ is the PI-resolvent of the clash.
Semantic Resolution (example)

Let $I = \{p, \neg q\}$, $S = \{p \lor q, \neg p \lor q, p \lor \neg q, \neg p \lor \neg q\}$, and $P = [p < q]$.

Ex: Show that PI-resolution is complete (hint: induction on the number of propositions).
Positive Hyperresolution: $I$ contains only negative literals.

Negative Hyperresolution: $I$ contains only positive literals.

A subset $T$ of a set of clauses $S$ is called a set-of-support of $S$ if $S - T$ is satisfiable.

A set-of-support resolution is a resolution of two clauses that are not both from $S - T$.

Ex: Show that set-of-support resolution is complete (hint: use PI-resolution completeness).
DPLL = Unit resolution + Split rule.

\[
\begin{align*}
\Gamma & \\
\Gamma, p \mid \Gamma, \neg p & \text{split} \quad p \text{ and } \neg p \text{ are not in } \Gamma. \\
C \lor \overline{l}, l & \\
C, l & \text{unit}
\end{align*}
\]

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!
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 ∧ x) = SAT return SAT
    - return DPLL(F ∧ ¬x)
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land
(\neg b \lor \neg d \lor \neg e) \land
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
DPLL (example)

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land 
(\neg b \lor \neg d \lor \neg e) \land 
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land 
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
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.
$p \iff q$ 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 \(\overline{(p \leftrightarrow q)}\)

\text{xor is the 1-bit adder}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(p \text{ xor } q)</th>
<th>(p \wedge q)</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>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 \text{xor} y \text{xor} 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>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 \((c_1, \ldots, c_n)\)

\[
\begin{align*}
    r_1 & \iff (x_1 \text{ xor } y_1) \\
    c_1 & \iff (x_1 \land y_1) \\
    r_2 & \iff (x_2 \text{ xor } y_2 \text{ xor } c_1) \\
    c_2 & \iff (x_2 \land y_2) \lor (x_2 \land c_1) \lor (y_2 \land c_1) \\
    & \quad \ldots \\
    r_n & \iff (x_n \text{ xor } y_n \text{ xor } c_{n-1}) \\
    c_n & \iff (x_n \land y_n) \lor (x_n \land c_{n-1}) \lor (y_n \land c_{n-1})
\end{align*}
\]
1) Encode $x \ast y$
2) Encode $x > y$ (signed and unsigned versions)
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)

Solver

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

SSA
Model checkers are used to verify/refute properties of transition systems.

Transition systems are used to model hardware and software.

Bounded Model Checking is a special kind of model checker.
Transition system \( M = (S, I, T) \)

\( S \): set of states.

\( I \subseteq S \): set of initial states. Example:

\[
I(s) = s.x = 0 \land s.pc = l_1
\]

\( T \subseteq S \times S \): transition relation. Example:

\[
T(s, s') = (s.pc = l_1 \land s'.x = s.x + 2 \land s'.pc = l_2) \lor
            (s.pc = l_2 \land s.x > 0 \land s'.x = s.x - 2 \land s'.pc = l_2) \lor
            (s.pc = l_2 \land s'.x = s.x \land s'.pc = l_1)
\]
Transition Systems (cont.)

\[ T(s, s') = (s'.pc = l_1 \land s'.x = s.x + 2 \land s'.pc = l_2) \lor \\
(s'.pc = l_2 \land s.x > 0 \land s'.x = s.x - 2 \land s'.pc = l_2) \lor \\
(s'.pc = l_2 \land s'.x = s.x \land s'.pc = l_1) \]

\[ x > 0, \ x' := x - 2 \]

\[ \pi(s_0, \ldots, s_n) \text{ is a path iff } I(s_0) \text{ and } T(s_i, s_{i+1}) \text{ for } 0 \leq i < n. \]

Example:

\[ (l_1, 0) \rightarrow (l_2, 2) \rightarrow (l_1, 2) \rightarrow (l_2, 4) \rightarrow (l_2, 2) \rightarrow (l_2, 0) \rightarrow (l_1, 0) \]
A state $s_k$ is reachable iff there is a path $\pi(s_0, \ldots, s_k)$.

**Invariants** characterize properties that are true of all reachable states in a system.

Any superset of the set of reachable states is an invariant.

Example: $s.x \geq 0$.

A **counterexample** for an invariant $\varphi$ is a path $\pi(s_0, \ldots, s_k)$ such that $\neg \varphi(s_k)$.

Model Checkers can **verify/refute** invariants.

There are different kinds of model checkers:

- **Explicit State**
- **Symbolic** (based on BDDs)
- **Bounded** (based on DP)
Given.

- Transition system $M = (S, I, T)$
- Invariant $\varphi$
- Natural number $k$

Problem.

Is there a counterexample of length $k$ for the invariant $\varphi$?

There is a **counterexample** for the invariant $\varphi$ if the following formula is satisfiable:

$$I(s_1) \land T(s_1, s_2) \land \ldots \land T(s_{k-1}, s_k) \land (\neg \varphi(s_1) \lor \ldots \lor \neg \varphi(s_k))$$
Bounded Model Checking: Invariants

Given.

- Transition system $M = (S, I, T)$
- Invariant $\varphi$
- Natural number $k$

Problem.

Is there a counterexample of length $k$ for the invariant $\varphi$?

There is a counterexample for the invariant $\varphi$ if the following formula is satisfiable:

$$I(s_1) \land T(s_1, s_2) \land \ldots \land T(s_{k-1}, s_k) \land (\neg \varphi(s_1) \lor \ldots \lor \neg \varphi(s_k))$$

$$\pi(s_0, \ldots, s_k)$$
BMC is mainly used for refutation.

Users want counterexamples. The decision procedure (DP) must be able to generate models for satisfiable formulas.

BMC is a complete method for finite systems when the diameter (longest shortest path) of the system is known.

The diameter is usually too expensive to be computed.

The recurrence diameter (longest loop-free path) is usually used as a completeness threshold.

The recurrence diameter can be much longer than the diameter.
A system $M$ contains a loop-free path of length $n$ iff

$$\pi(s_0, \ldots, s_n) \land \bigwedge_{0 \leq i < j \leq n} s_i \neq s_j$$

The recurrence diameter is the smallest $n$ such that the formula above is unsatisfiable.

The diameter of infinite systems (i.e., infinite state space) may be infinite.
Verifying Invariants

An invariant is inductive if:

- \( I(s) \rightarrow \varphi(s) \) (base step)

- \( \varphi(s) \land T(s, s') \rightarrow \varphi(s') \) (inductive step)

Invariants are not usually inductive.

The inductive step is violated.

Example: \((l_2, 1) \rightarrow (l_2, -1)\)
An invariant $\varphi$ is $k$-inductive if:

- $I(s_1) \land T(s_1, s_2) \land \ldots \land T(s_{k-1}, s_k) \rightarrow \varphi(s_1) \land \ldots \land \varphi(s_k)$

- $\varphi(s_1) \land \ldots \land \varphi(s_k) \land T(s_1, s_2) \land \ldots \land T(s_k, s_{k+1}) \rightarrow \varphi(s_{k+1})$

It is harder to violate the inductive step.

The base case is BMC.

If $\varphi$ is $k_1$-inductive then it is also $k_2$-inductive for $k_2 \geq k_1$. 
Verifying Invariants: $k$-induction (cont.)

Can be used to verify finite and infinite systems.

**Not complete** even for finite systems: Self-loops in unreachable states.

**Example:**

![Diagram](image)

**Bad state** $s_4$

**Counterexamples** $s_3 \leadsto s_3 \leadsto \ldots \leadsto s_3 \leadsto s_4$
Completeness for finite systems: consider only loop-free paths.

Not complete for infinite systems. Example:

- \((l_2, 1) \rightarrow (l_2, -1)\)
- \((l_2, 3) \rightarrow (l_2, 1) \rightarrow (l_2, -1)\)
- \((l_2, 5) \rightarrow (l_2, 3) \rightarrow (l_2, 1) \rightarrow (l_2, -1)\)
- ...
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 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. For Java users. Far less efficient than the two others.

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

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
- The SAT competition: 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.