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.

Microsoft Research

Program correctness

I proved my program to be correct.

What does it mean?

Microsoft Research

Software models

We need models and tools to reason about them?

Does my model/software has property X?

Microsoft Research

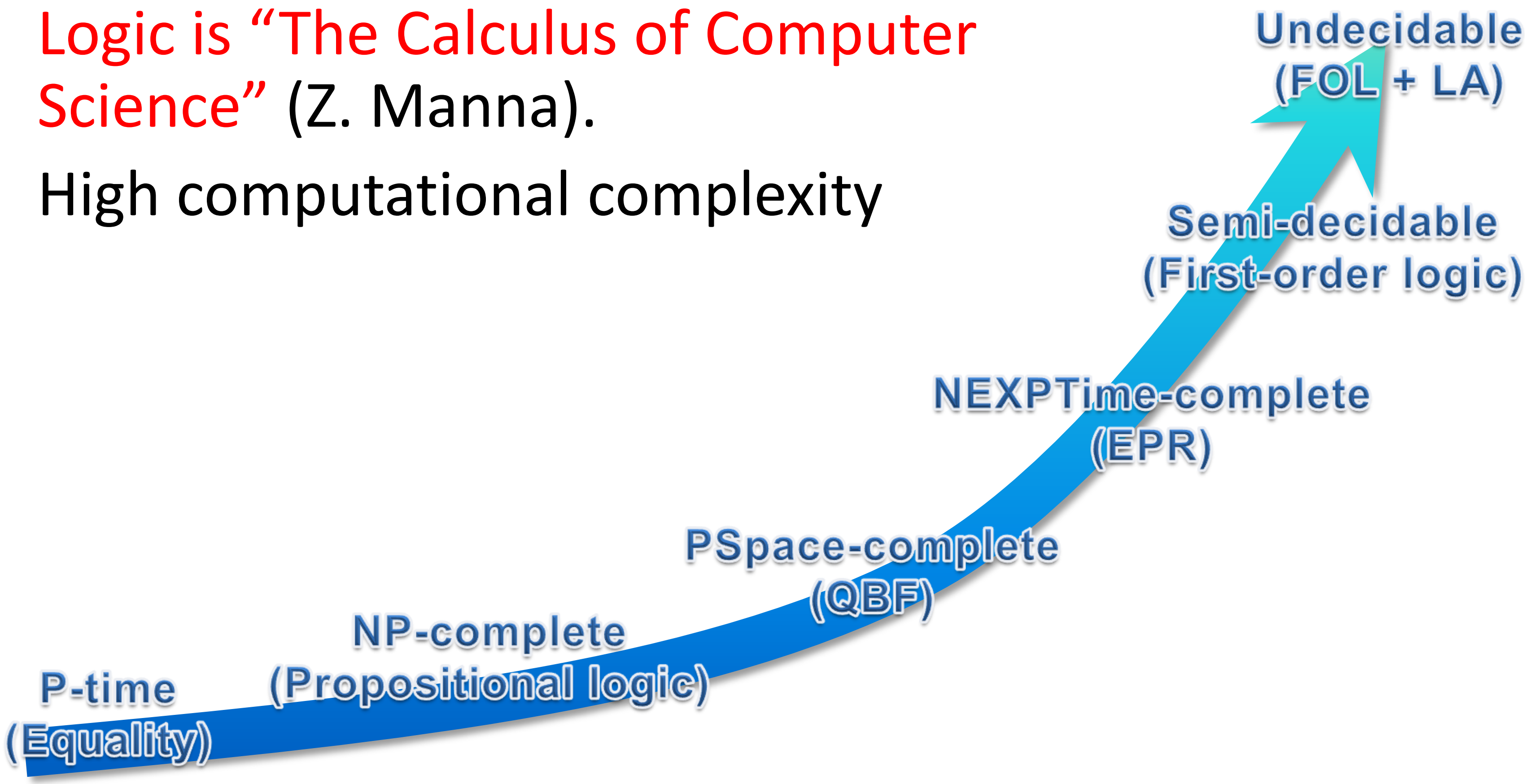
Symbolic Reasoning

Verification/Analysis tools need some form of **Symbolic Reasoning**

Microsoft Research

Symbolic Reasoning

- Logic is "The Calculus of Computer Science" (Z. Manna).
- High computational complexity



Microsoft Research

Applications

- Test case generation
- Verifying Compilers
- Predicate Abstraction
- Invariant Generation
- Type Checking
- Model Based Testing

Some Applications @ Microsoft

The Spec# Programming System

HAVOC

ForμLa

Hyper-V

Microsoft | Virtualization

Terminator T-2

VCC

SLAM

Yogi

NModel

Vigilante

SpecExplorer

Pex

F7

SAGE

Microsoft Research

Test case generation

```
unsigned GCD(x, y) {
  requires(y > 0);
  while (true) {
    unsigned m = x % y;
    if (m == 0) return y;
    x = y;
    y = m;
  }
}
```

$\xrightarrow{\text{SSA}}$

$(y_0 > 0)$ 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

$(m_1 = 0)$

$\xrightarrow{\text{Solver}}$

$x_0 = 2$

$y_0 = 4$

$m_0 = 2$

$x_1 = 4$

$y_1 = 2$

$m_1 = 0$

We want a trace where the loop is executed twice.

Type checking

Signature:

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

Subtype

Call site:

if a $\leq$ 1 and a $\leq$ b then

return div(a, b)

Verification condition

a $\leq$ 1 and a $\leq$ b implies b $\neq$ 0

What is logic?

- 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.

Microsoft Research

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.

Microsoft Research

What is logical language?

- 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 $\wedge$, $\vee$, and $\neg$ have a fixed interpretation, whereas the constants p, q, r may be interpreted at will.

Microsoft Research

Propositional Logic

Formulas: $\varphi := p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi_1 \mid \varphi_1 \Rightarrow \varphi_2$

Examples:

$p \vee q \Rightarrow q \vee p$

$p \wedge \neg q \wedge (\neg p \vee q)$

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.

Microsoft Research

Interpretation

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 ϕ in $\mathcal{M}$ and is computed using *truth tables*:

ϕ	A	B	$\neg A$	$A \vee B$	$A \wedge \neg A$	$A \Rightarrow B$	$A \Rightarrow (B \vee A)$
$\mathcal{M}_1(\phi)$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\mathcal{M}_2(\phi)$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\top$
$\mathcal{M}_3(\phi)$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\mathcal{M}_4(\phi)$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$

Satisfiability & Validity

- 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

$p \vee q \Rightarrow q \vee p$

$p \vee q \Rightarrow q$

$p \wedge \neg q \;\; \wedge (\neg p \vee q)$

ϕ	A	B	$\neg A$	$A \vee B$	$A \wedge \neg A$	$A \Rightarrow B$	$A \Rightarrow (B \vee A)$
$\mathcal{M}_1(\phi)$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\mathcal{M}_2(\phi)$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\top$
$\mathcal{M}_3(\phi)$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\mathcal{M}_4(\phi)$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$

Satisfiability & Validity: examples

$p \vee q \Rightarrow q \vee p$ VALID

$p \vee q \Rightarrow q$ SATISFIABLE

$p \wedge \neg q \ \wedge (\neg p \vee q)$ UNSATISFIABLE

ϕ	A	B	$\neg A$	$A \vee B$	$A \wedge \neg A$	$A \Rightarrow B$	$A \Rightarrow (B \vee A)$
$\mathcal{M}_1(\phi)$	$\perp$	$\perp$	$\top$	$\perp$	$\perp$	$\top$	$\top$
$\mathcal{M}_2(\phi)$	$\perp$	$\top$	$\top$	$\top$	$\perp$	$\top$	$\top$
$\mathcal{M}_3(\phi)$	$\top$	$\perp$	$\perp$	$\top$	$\perp$	$\perp$	$\top$
$\mathcal{M}_4(\phi)$	$\top$	$\top$	$\perp$	$\top$	$\perp$	$\top$	$\top$

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

Normal Forms

NNF?

$(p \vee \neg q) \wedge (q \vee \neg(r \wedge \neg p))$

Normal Forms

NNF? NO

$(p \vee \neg q) \wedge (q \vee \neg(r \wedge \neg p))$

Normal Forms

NNF? NO

$(p \vee \neg q) \wedge (q \vee \neg(r \wedge \neg p))$

1. $\neg\neg A \iff A$
2. $A \Rightarrow B \iff \neg A \vee B$
3. $\neg(A \wedge B) \iff \neg A \vee \neg B$
4. $\neg(A \vee B) \iff \neg A \wedge \neg B$

Normal Forms

NNF? NO

$(p \vee \neg q) \wedge (q \vee \neg(r \wedge \neg p))$

$\Leftrightarrow$

$(p \vee \neg q) \wedge (q \vee (\neg r \vee \neg\neg p))$

1. $\neg\neg A \iff A$
2. $A \Rightarrow B \iff \neg A \vee B$
3. $\neg(A \wedge B) \iff \neg A \vee \neg B$
4. $\neg(A \vee B) \iff \neg A \wedge \neg B$

Normal Forms

NNF? NO

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

$\Leftrightarrow$

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

$\Leftrightarrow$

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

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?

$((p \wedge s) \vee (\neg q \wedge r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

Normal Forms

CNF? NO

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

Normal Forms

CNF? NO

$((p \wedge s) \vee (\neg q \wedge r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Normal Forms

CNF? NO

$((p \wedge s) \vee (\neg q \wedge r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

$\Leftrightarrow$

$((p \wedge s) \vee \neg q)) \wedge ((p \wedge s) \vee r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Normal Forms

CNF? NO

$((p \wedge s) \vee (\neg q \wedge r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

$\Leftrightarrow$

$((p \wedge s) \vee \neg q)) \wedge ((p \wedge s) \vee r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

$\Leftrightarrow$

$(p \vee \neg q) \wedge (s \vee \neg q) \wedge ((p \wedge s) \vee r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Normal Forms

CNF? NO

$((p \wedge s) \vee (\neg q \wedge r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

$\Leftrightarrow$

$((p \wedge s) \vee \neg q)) \wedge ((p \wedge s) \vee r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

$\Leftrightarrow$

$(p \vee \neg q) \wedge (s \vee \neg q) \wedge ((p \wedge s) \vee r)) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

$\Leftrightarrow$

$(p \vee \neg q) \wedge (s \vee \neg q) \wedge (\mathrm{p} \vee \mathrm{r}) \wedge (\mathrm{s} \vee \mathrm{r}) \wedge (q \vee \neg p \vee s) \wedge (\neg r \vee s)$

Normal Forms

DNF?

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$((p \wedge \neg p) \vee (p \vee q)) \wedge (\neg q \vee r)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$((p \wedge \neg p) \vee (p \vee q)) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$(p \vee q) \wedge (\neg q \vee r)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Other Rules

1. $A \wedge \neg A \Leftrightarrow \perp$
2. $A \vee \perp \Leftrightarrow A$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$((p \wedge \neg p) \vee (p \vee q)) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$(p \vee q) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$((p \vee q) \wedge \neg q) \vee ((p \vee q) \wedge r)$

Distributivity

1. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
2. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

Other Rules

1. $A \wedge \neg A \Leftrightarrow \perp$
2. $A \vee \perp \Leftrightarrow A$

Normal Forms

DNF? NO, actually this formula is in CNF

$p \wedge (\neg p \vee q) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$((p \wedge \neg p) \vee (p \vee q)) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$(p \vee q) \wedge (\neg q \vee r)$

$\Leftrightarrow$

$((p \vee q) \wedge \neg q) \vee ((p \vee q) \wedge r)$

$\Leftrightarrow$

$(p \wedge \neg q) \vee (q \wedge \neg q) \vee ((p \vee q) \wedge r)$

$\Leftrightarrow$

$(p \wedge \neg q) \vee (p \wedge r) \vee (q \wedge \mathrm{r})$

Refutation Decision Procedures

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* ψ.

State ψ is of the form $\kappa_1|\ldots|\kappa_n$, where each κ_i is a *configuration*.

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

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

A configuration κ of the form $A_1, \ldots, A_m$ is satisfiable if there is an interpretation M such that
$M \models A_i$ 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 ψ to a state ψ' (written $\psi \models\!\!\!\Rightarrow \psi'$).

Rules preserve satisfiability.

If relation $\models\!\!\!\Rightarrow$ 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.

Truth Table

An inference rule $\dfrac{\kappa}{\kappa_1 | \ldots | \kappa_n}$ is shorthand for $\dfrac{\psi[\kappa]}{\psi[\kappa_1 | \ldots | \kappa_n]}$.

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

$$\frac{\Gamma}{\Gamma, p \mid \Gamma, \neg p}\mathit{split} \qquad p \text{ and } \neg p \text{ are not in } \Gamma.$$

$$\frac{\Gamma, F}{\bot}\mathit{elim} \qquad F \text{ is falsified by the literals in } \Gamma.$$

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

The literals in Γ can be viewed as a *partial interpretation*.

Ex: Prove correctness (soundness, termination, and completeness).

Truth Table (example)

A truth table refutation of $\{p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p\}$:

$$
\dfrac{p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p}{\dfrac{p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p, q \mid p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p, \neg q}{\dfrac{p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p, q \mid \bot}{\dfrac{p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p, q}{\dfrac{p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p, q, r \mid p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p, q, \neg r}{\dfrac{\bot \mid p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p, q, \neg r}{\dfrac{p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p, q, \neg r}{\bot}}}}}}}
$$

Ex: Implement the truth table procedure.

Semantic Tableaux

The inference rules for the Semantic Tableaux procedure are:

$\dfrac{A \wedge B, \Gamma}{A, B, \Gamma} \wedge +$	$\dfrac{\neg(A \wedge B), \Gamma}{\neg A, \Gamma \mid \neg B, \Gamma} \wedge -$
$\dfrac{\neg(A \vee B), \Gamma}{\neg A, \neg B, \Gamma} \vee -$	$\dfrac{(A \vee B), \Gamma}{A, \Gamma \mid B, \Gamma} \vee +$
$\dfrac{\neg(A \Rightarrow B), \Gamma}{A, \neg B, \Gamma} \Rightarrow -$	$\dfrac{(A \Rightarrow B), \Gamma}{\neg A, \Gamma \mid B, \Gamma} \Rightarrow +$
$\dfrac{\neg\neg A, \Gamma}{A, \Gamma} \neg$	$\dfrac{A, \neg A, \Gamma}{\bot} \bot$

Semantic Tableaux is a *"DNF translator"*.

Ex: Prove correctness.

Semantic Tableaux (example)

Refutation of $\neg(p \vee q \Rightarrow q \vee p)$:

$\dfrac{A \wedge B, \Gamma}{A, B, \Gamma}\wedge+$	$\dfrac{\neg(A \wedge B), \Gamma}{\neg A, \Gamma \mid \neg B, \Gamma}\wedge-$
$\dfrac{\neg(A \vee B), \Gamma}{\neg A, \neg B, \Gamma}\vee-$	$\dfrac{(A \vee B), \Gamma}{A, \Gamma \mid B, \Gamma}\vee+$
$\dfrac{\neg(A \Rightarrow B), \Gamma}{A, \neg B, \Gamma}\Rightarrow -$	$\dfrac{(A \Rightarrow B), \Gamma}{\neg A, \Gamma \mid B, \Gamma}\Rightarrow +$
$\dfrac{\neg\neg A, \Gamma}{A, \Gamma}\neg$	$\dfrac{A, \neg A, \Gamma}{\bot}\bot$

$$\begin{array}{c}
\neg(p \vee q \Rightarrow q \vee p) \\ \hline
p \vee q, \neg(q \vee p) \\ \hline
p, \neg(q \vee p) \mid q, \neg(q \vee p) \\ \hline
p, \neg q, \neg p \mid q, \neg(q \vee p) \\ \hline
\bot \mid q, \neg(q \vee p) \\ \hline
q, \neg(q \vee p) \\ \hline
q, \neg q, \neg p \\ \hline
\bot
\end{array}$$

Ex: Use the Semantic Tableaux procedure to refute $\neg(p \vee (q \wedge r) \Rightarrow (p \vee q) \wedge (p \vee r))$.

Ex: Implement the Semantic Tableaux.

Semantic Tableaux (cont.)

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:

$$
\begin{aligned}
&(p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee p_3) \wedge \\
&(p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge \\
&(p_1 \vee p_2 \vee \neg p_3) \wedge (\neg p_1 \vee p_2 \vee \neg p_3) \wedge \\
&(p_1 \vee \neg p_2 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2 \vee \neg p_3)
\end{aligned}
$$

Ex: Use Semantic Tableaux to refute the formula above.

Semantic Tableaux (cont.)

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 correspondes to the *principle of bivalence*.

The elimination of the *principle of bivalence* seem to be inadequate from the point of view of efficiency.

Semantic Tableaux + Bivalence

The *principle of bivalence* can be recovered if we replace the Semantic Tableaux *branching* rules by:

$\dfrac{\neg(A \wedge B), \Gamma}{\neg A, \Gamma \mid A, \neg B, \Gamma} \wedge_{left} -$	$\dfrac{\neg(A \wedge B), \Gamma}{\neg B, \Gamma \mid B, \neg A, \Gamma} \wedge_{right} -$
$\dfrac{(A \vee B), \Gamma}{A, \Gamma \mid \neg A, B, \Gamma} \vee_{left} +$	$\dfrac{(A \vee B), \Gamma}{B, \Gamma \mid \neg B, A, \Gamma} \vee_{right} +$
$\dfrac{(A \Rightarrow B), \Gamma}{\neg A, \Gamma \mid A, B, \Gamma} \Rightarrow_{left} +$	$\dfrac{(A \Rightarrow B), \Gamma}{B, \Gamma \mid \neg B, \neg A, \Gamma} \Rightarrow_{right} +$

The new rules are *asymmetric*.

Ex: Show that the new rules are sound.

CNF (again)

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).

CNF (again)

Equivalence rules can be used to translate any formula to CNF.

eliminate $\Rightarrow$	$A \Rightarrow B \equiv \neg A \vee B$
reduce the scope of $\neg$	$\neg(A \vee B) \equiv \neg A \wedge \neg B$, $\neg(A \wedge B) \equiv \neg A \vee \neg B$
apply distributivity	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$, $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$

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:

$$\frac{F[l_i \; op \; l_j]}{F[x], x \Leftrightarrow l_i \; op \; l_j}{}^{*}$$

$$\frac{x \Leftrightarrow l_i \lor l_j}{\neg x \lor l_i \lor l_j, \neg l_i \lor x, \neg l_j \lor x}$$

$$\frac{x \Leftrightarrow l_i \land l_j}{\neg x \lor l_i, \neg x \lor l_j, \neg l_i \lor \neg l_j \lor x}$$

(*) x must be a fresh variable.

Ex: Show that the rules preserve equisatisfiability.

CNF translation (example)

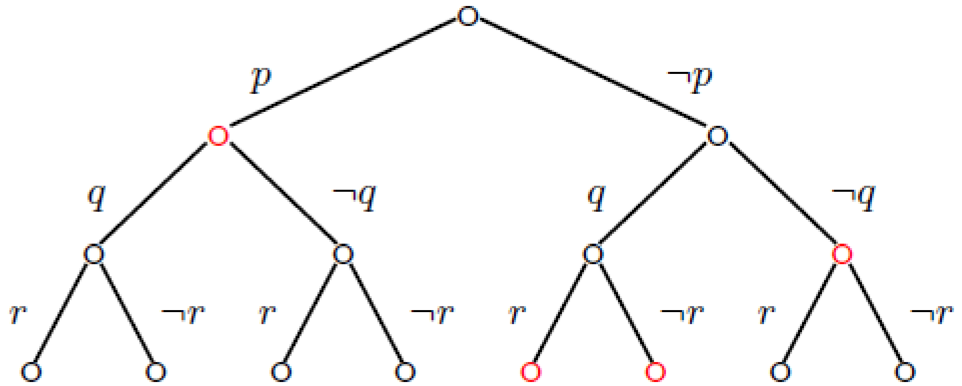
Translation of $(p \wedge (q \vee r)) \vee t$:

$$\frac{\dfrac{\dfrac{\dfrac{(p \wedge (q \vee r)) \vee t}{(p \wedge x_1) \vee t, x_1 \Leftrightarrow q \vee r}}{x_2 \vee t, x_2 \Leftrightarrow p \wedge x_1, x_1 \Leftrightarrow q \vee r}}{x_2 \vee t, \neg x_2 \vee p, \neg x_2 \vee x_1, \neg p \vee \neg x_1 \vee x_2, x_1 \Leftrightarrow q \vee r}}{x_2 \vee t, \neg x_2 \vee p, \neg x_2 \vee x_1, \neg p \vee \neg x_1 \vee x_2, \neg x_1 \vee q \vee r, \neg q \vee x_1, \neg r \vee x_1}$$

Ex: Implement a CNF translator.

Semantic Trees

A *semantic tree* represents the set of partial interpretations for a set of clauses. A semantic tree for $\{p \vee \neg q \vee \neg r, p \vee r, p \vee q, \neg p\}$:



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*:

$$\frac{C_1 \vee p, C_2 \vee \neg p}{C_1 \vee p, C_2 \vee \neg p, C_1 \vee C_2} res$$

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.

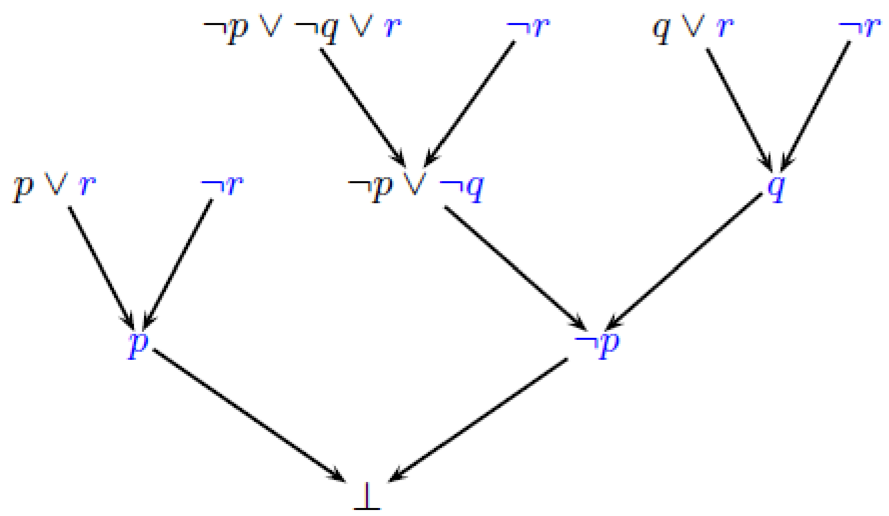
Example: The resolvent of $p \vee q \vee r$, and $\neg p \vee r \vee t$ is $q \vee r \vee t$.

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

Ex: Show that the resolution rule is sound.

Resolution (example)

A refutation of $\neg p \vee \neg q \vee r$, $p \vee r$, $q \vee r$, $\neg r$:



Ex: Implement a naïve resolution procedure.

Completeness of Resolution

Let $Res(S)$ be the closure of S under the resolution rule.

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

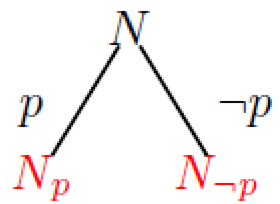
Proof ($\Rightarrow$):

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

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

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

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

Proof ($\Leftarrow$): $Res(S)$ is unsatisfiable, and equivalent to S. So, S is unsatisifiable.

Subsumption

The *resolution* procedure may generate several *irrelevant* and *redundant clauses*.

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

$$\frac{C_1, C_1 \vee C_2}{C_1} sub$$

Example: $p \vee \neg q$ subsumes $p \vee \neg q \vee r \vee t$.

Deletion strategy: Remove the subsumed clauses.

Unit & Input Resolution

Unit resolution: one of the clauses is a unit clause.

$$\frac{C \vee \bar{l}, l}{C, l}\, unit$$

Unit resolution always *decreases* the configuration *size* ($C \vee \bar{l}$ 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 on positive literal.

Rule base systems ($\neg p_1 \vee \ldots \vee \neg p_n \vee q \;\equiv\; p_1 \wedge \ldots \wedge p_n \Rightarrow q$).

Positive unit rule:

$$\frac{C \vee \neg p, p}{C, p}\, unit^+$$

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.

Semantic Resolution

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, ..., 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 $P = [p < q]$.

$$
\begin{tikzcd}
\neg p \lor q \arrow[dr] & & \neg p \lor \neg q \arrow[dl] & \\
& \neg p \arrow[dr] & & p \lor q \arrow[dl] \\
& \neg p \arrow[dr] & q \arrow[d] & p \lor \neg q \arrow[dl] \\
& & \bot &
\end{tikzcd}
$$

Ex: Show that *PI-resolution* is complete (hint: induction on the number of propositions).

Semantic Resolution (special cases)

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

DPLL $=$ Unit resolution $+$ Split rule.

$$\frac{\Gamma}{\Gamma, p \mid \Gamma, \neg p}\,split \qquad p \text{ and } \neg p \text{ are not in } \Gamma.$$

$$\frac{C \vee \bar{l}, l}{C, l}\,unit$$

Used in the most efficient SAT solvers.

Pure Literals

A literal is pure if only occurs positively or negatively.

Example :

$\varphi = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

$\neg x_1$ 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)

$$\varphi_{\neg x_1, x_3} = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$$

Preserve satisfiability, not logical equivalency !

Pure Literals

A literal is pure if only occurs positively or negatively.

Example :

$\varphi = (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

$\neg x_1$ 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)

$$\varphi_{\neg x_1, x_3} = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$$

Preserve satisfiability, not logical equivalency !

DPLL (as a procedure)

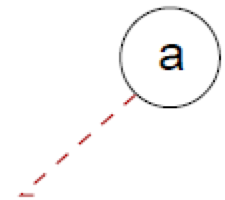
- Standard backtrack search
- DPLL(F) :
  - Apply unit propagation
  - If conflict identified, return UNSAT
  - Apply the pure literal rule
  - If F is satisfied (empty), return SAT
  - Select decision variable x
    - If DPLL($F \wedge x$)=SAT return SAT
    - return DPLL($F \wedge \neg x$)

DPLL (example)

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$

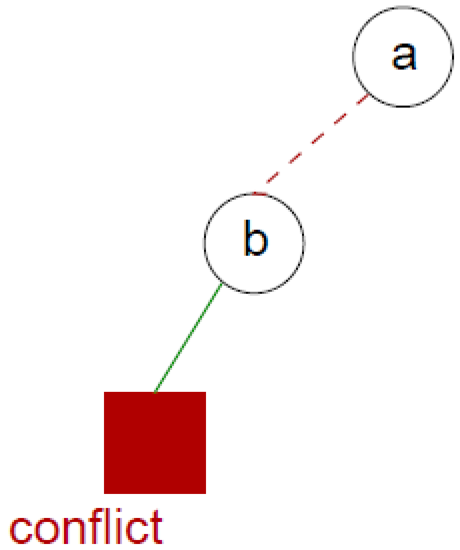
DPLL (example)

$$
\begin{aligned}
\varphi \quad = \quad & (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
& (\neg b \lor \neg d \lor \neg e) \land \\
& (a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
& (a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e)
\end{aligned}
$$



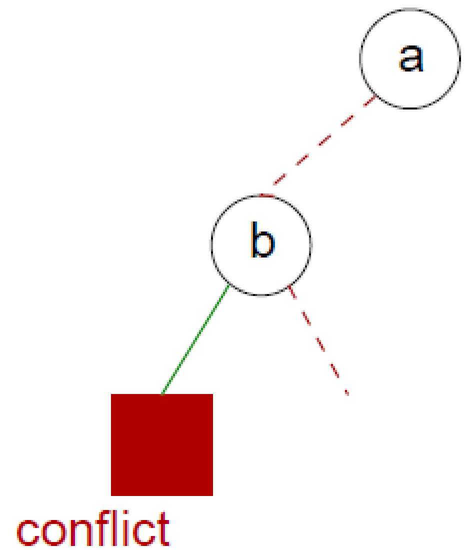
DPLL (example)

$$\begin{aligned}\varphi \;=\; & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



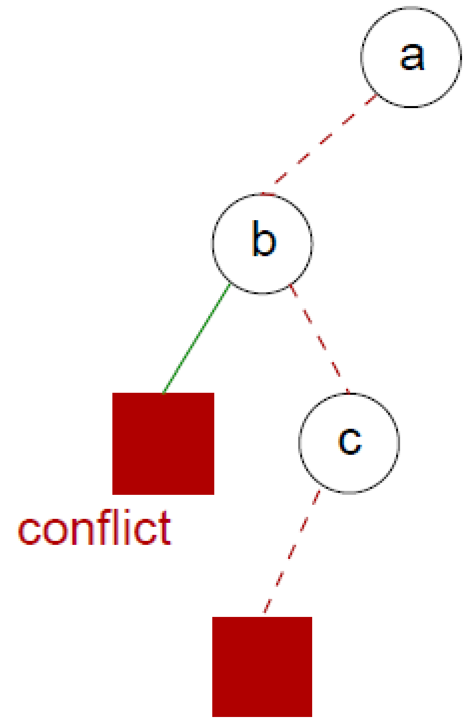
DPLL (example)

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$



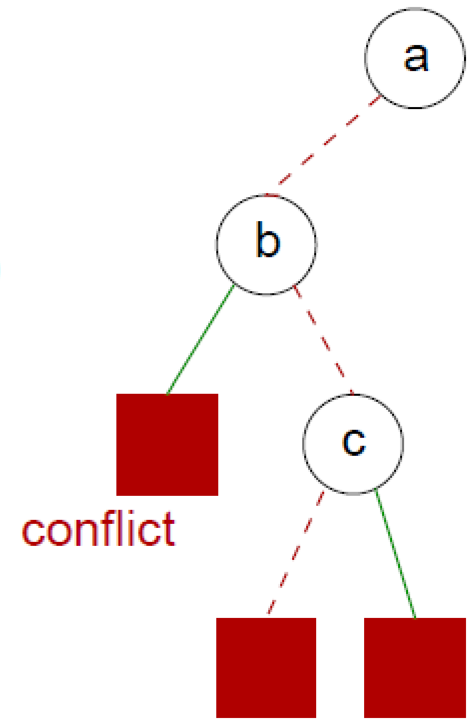
DPLL (example)

$$\begin{aligned}\varphi \;=\; & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



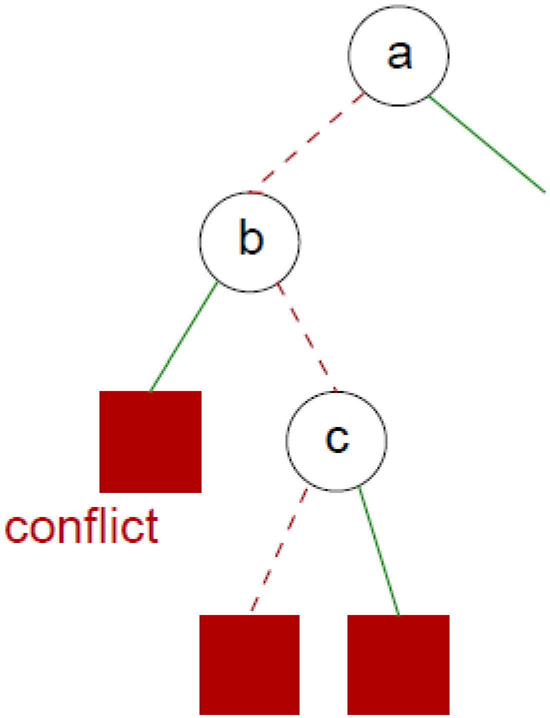
DPLL (example)

$$\begin{aligned}\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\ & (\neg b \vee \neg d \vee \neg e) \wedge \\ & (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\ & (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)\end{aligned}$$



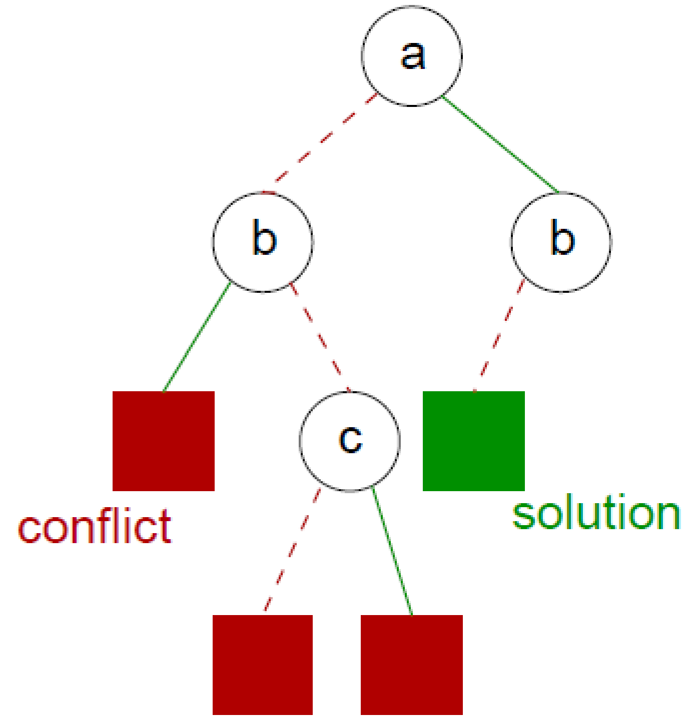
DPLL (example)

$$
\begin{aligned}
\varphi \quad = \quad & (a \vee \neg b \vee d) \wedge (a \vee \neg b \vee e) \wedge \\
& (\neg b \vee \neg d \vee \neg e) \wedge \\
& (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \neg d) \wedge \\
& (a \vee b \vee \neg c \vee e) \wedge (a \vee b \vee \neg c \vee \neg e)
\end{aligned}
$$



DPLL (example)

$$
\begin{aligned}
\varphi \quad = \quad & (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
& (\neg b \lor \neg d \lor \neg e) \land \\
& (a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
& (a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e)
\end{aligned}
$$



Some Applications

Bit-vector / Machine arithmetic

Let x, y and z be 8-bit (unsigned) integers.

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

Is $x > 0 \wedge y > 0 \wedge z = x + y \wedge \neg(z > 0)$ satisfiable?

Bit-vector / Machine arithmetic

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.

Encoding equality

$p \Leftrightarrow q$ is equivalent to $(\neg p \vee q) \wedge (\neg q \vee p)$

The bit-vector equation x = y is encoded as:

$(x_1 \Leftrightarrow y_1) \wedge \ldots \wedge (x_n \Leftrightarrow y_n)$

Encoding addition

We use $(r_1, \ldots, r_n)$ to store the result of $x + y$

p xor q is defined as $\neg(p \Leftrightarrow q)$

xor is the 1-bit adder

p	q	p xor q	$p \wedge q$
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

carry

Encoding 1-bit full adder

1-bit full adder

Three inputs: x, y, c_{in}

Two outputs: r, c_{out}

x	y	c_{in}	$r = x$ xor y xor c_{in}	$c_{out} = (x \wedge y) \vee (x \wedge c_{in}) \vee (y \wedge c_{in})$
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Encoding n-bit adder

We use $(r_1, \ldots, r_n)$ to store the result of $x + y$,
and $(c_1, \ldots, c_n)$

$r_1 \Leftrightarrow (x_1 \text{ xor } y_1)$

$c_1 \Leftrightarrow (x_1 \wedge y_1)$

$r_2 \Leftrightarrow (x_2 \text{ xor } y_2 \text{ xor } c_1)$

$c_2 \Leftrightarrow (x_2 \wedge y_2) \vee (x_2 \wedge c_1) \vee (y_2 \wedge c_1)$

…

$r_n \Leftrightarrow (x_n \text{ xor } y_n \text{ xor } c_{n-1})$

$c_n \Leftrightarrow (x_n \wedge y_n) \vee (x_n \wedge c_{n-1}) \vee (y_n \wedge c_{n-1})$

Exercises

1) Encode x * y
2) Encode x > y (signed and unsigned versions)

Test case generation (again)

```
unsigned GCD(x, y) {
  requires(y > 0);
  while (true) {
    unsigned m = x % y;
     if (m == 0) return y;
     x = y;
     y = m;
  }
}
```

SSA

$(y_0 > 0)$ 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

$(m_1 = 0)$

Solver

$x_0 = 2$

$y_0 = 4$

$m_0 = 2$

$x_1 = 4$

$y_1 = 2$

$m_1 = 0$

We want a trace where the loop is executed twice.

Bounded Model Checkers

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 Systems

Transition system $M = (S, I, T)$

S: set of states.

$I \subseteq S$: set of initial states. Example:

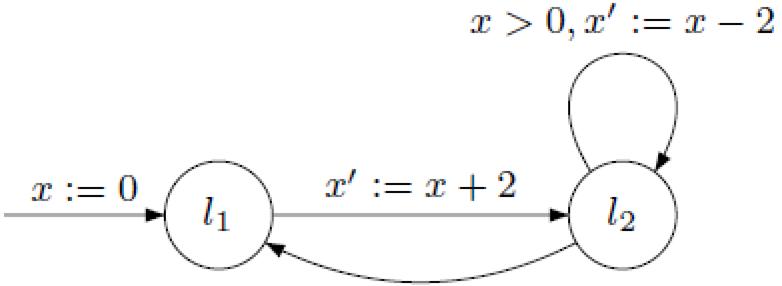
$$I(s) \quad = \quad s.x = 0 \wedge s.pc = l_1$$

$T \subseteq S \times S$: transition relation. Example:

$$\begin{aligned} T(s, s') \quad = \quad & (s.pc = l_1 \wedge s'.x = s.x + 2 \wedge s'.pc = l_2) \vee \\ & (s.pc = l_2 \wedge s.x > 0 \wedge s'.x = s.x - 2 \wedge s'.pc = l_2) \vee \\ & (s.pc = l_2 \wedge s'.x = s.x \wedge s'.pc = l_1) \end{aligned}$$

Transition Systems (cont.)

$$\begin{aligned} T(s, s') \quad = \quad & (s.pc = l_1 \land s'.x = s.x + 2 \land s'.pc = l_2) \lor \\ & (s.pc = l_2 \land s.x > 0 \land s'.x = s.x - 2 \land s'.pc = l_2) \lor \\ & (s.pc = l_2 \land s'.x = s.x \land s'.pc = l_1) \end{aligned}$$



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

$$(l_1, 0) \rightarrow (l_2, 2) \rightarrow (l_1, 2) \rightarrow (l_2, 4) \rightarrow (l_2, 2) \rightarrow (l_2, 0) \rightarrow (l_1, 0)$$

Invariants

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 φ 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)

Bounded Model Checking: Invariants

Given.

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

Problem.

Is there a counterexample of length k for the invariant φ?

There is a counterexample for the invariant φ if the following formula is satisfiable:

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

Bounded Model Checking: Invariants

Given.

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

Problem.

Is there a counterexample of length k for the invariant φ?

There is a counterexample for the invariant φ if the following formula is satisfiable:

$$\underbrace{I(s_1) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_{k-1}, s_k)}_{\pi(s_0, \ldots, s_k)} \wedge (\neg\varphi(s_1) \vee \ldots \vee \neg\varphi(s_k))$$

Bounded Model Checking (cont.)

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 to 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.

Recurrence diameter

A system M contains a loop-free path of length n iff

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

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

Invariants are not usually inductive.

The inductive step is violated.

Example: $(l_2, 1) \rightarrow (l_2, -1)$

Verifying Invariants: *k*-induction

An invariant φ is k-inductive if:

- $I(s_1) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_{k-1}, s_k) \to \varphi(s_1) \wedge \ldots \wedge \varphi(s_k)$
- $\varphi(s_1) \wedge \ldots \wedge \varphi(s_k) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_k, s_{k+1}) \to \varphi(s_{k+1})$

It is harder to violate the inductive step.

The base case is BMC.

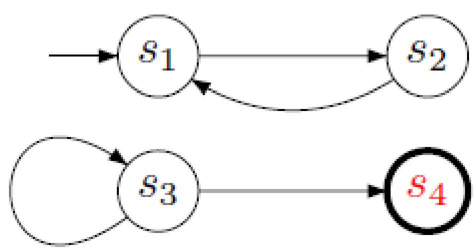
If φ is k_1-inductive then it is also k_2-inductive for $k_2 \geq k_1$.

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:



Bad state s_4

Counterexamples $\underbrace{s_3 \leadsto s_3 \leadsto \ldots \leadsto s_3}_{k} \leadsto s_4$

Verifying Invariants: *k*-induction (cont.)

Completeness for finite systems: consider only loop-free paths.

Not complete for infinite systems. Example:

- $(l_2, 1) \rightarrow (l_2, -1)$
- $(l_2, 3) \rightarrow (l_2, 1) \rightarrow (l_2, -1)$
- $(l_2, 5) \rightarrow (l_2, 3) \rightarrow (l_2, 1) \rightarrow (l_2, -1)$
- $\ldots$

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++) `http://fmv.jku.at/software/index.html` Award winner in 2007 and 2009 of the SAT competition, industrial category.

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 competion: http://www.satcompetition.org
- Search the WEB: “SAT benchmarks”

Using SAT solvers

All SAT solvers support the very simple DIMACS CNF input format :

$$(a \vee b \vee \neg c) \wedge (\neg b \vee \neg c)$$

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.