PROPOSITIONAL LOGIC: SAT SOLVING

VL Logic, Lecture 4, WS 20/21

Armin Biere, Martina Seidl

Institute for Formal Models and Verification
Johannes Kepler University Linz

JOHANNES KEPLER
UNIVERSITY LINZ

PROPOSITIONAL FORMULAS: THE SAT PROBLEM

VL Logic

Part I: Propositional Logic

JOHANNES KEPLER
UNIVERSITY LINZ

Satisfiability Checking

Definition (Satisfiability Problem of Propositional Logic (SAT))

Given a formula ϕ, is there an assignment ν such that $[\phi]_\nu = \mathbf{1}$?

- oldest **NP**-complete problem
  - checking a solution (assignment satisfies formula) is easy (polynomial effort)
  - finding a solution is difficult (probably exponential in the worst case)
- many practical applications (used in industry)
- efficient SAT solvers (solving tools) are available
- other problems can be translated to SAT:

problem	**formulation in propositional logic**
ϕ is valid	$\neg\phi$ is unsatisfiable
ϕ is refutable	to $\neg\phi$ is satisfiable
$\phi \Leftrightarrow \psi$	to $\neg(\phi \leftrightarrow \psi)$ is unsatisfiable
$\phi_1, \ldots, \phi_n \models \psi$	$\phi_1 \wedge \ldots \wedge \phi_n \wedge \neg\psi$ is unsatisfiable

Reasoning with (Propositional) Calculi

- **goal**: automatically reason about (propositional) formulas
  i.e., mechanically show validity / unsatisfiability
- **basic idea:** use syntactical manipulations to prove/refute a formula
- **elements of a calculus**:
  - **axioms**: trivial truths/trivial contradictions
  - **rules**: inference of new formulas
- **approach**: construct a **proof/refutation**
  - apply the rules of the calculus until only axioms are inferred
  - if this is not possible, then the formula is not valid/unsatisfiable
- **examples of calculi**:
  - sequence calculus: shows validity (actually entailment)
  - resolution calculus: shows unsatisfiability

Logic Entailment

Let $\phi_1, \ldots \phi_n, \psi$ be propositional formulas.
Then $\phi_1, \ldots \phi_n$ **entail** ψ (written as $\phi_1, \ldots, \phi_n \models \psi$) iff
$[\phi_1]_v = \mathbf{1}, \ldots [\phi_n]_v = \mathbf{1}$ implies that $[\psi]_v = \mathbf{1}$.

Informal meaning: True premises derive a true conclusion.

- $\models$ is a **meta-symbol** (it is not part of the language)
- $\phi_1, \ldots \phi_n \models \psi$ iff $(\phi_1 \wedge \ldots \wedge \phi_n) \rightarrow \psi$ is valid,
  i.e., we can express semantics by means of syntactics.
- If $\phi_1, \ldots \phi_n$ do not entail ψ, we write $\phi_1, \ldots \phi_n \not\models \psi$.

Example:

- $a \models a \vee b$
- $\models a \vee \neg a$
- $a, a \rightarrow b \models b$
- $a, b \models a \wedge b$
- $\not\models a \wedge \neg a$
- $\bot \models a \wedge \neg a$

Formula Strength

- formula ϕ is stronger than formula ψ iff $\phi \models \psi$
- formula ψ is weaker than formula ϕ iff $\phi \models \psi$
- formulas ϕ and ψ are equally strong iff $\phi \models \psi$ and $\psi \models \phi$

Examples

- $a \oplus b$ is stronger than $a \vee b$
- $a \wedge b$ is stronger than $a \vee b$
- $\perp$ is the strongest formula
- $\top$ is the weakest formula

PROPOSITIONAL FORMULAS: SEQUENT CALCULUS

VL Logic

Part I: Propositional Logic

JOHANNES KEPLER
UNIVERSITY LINZ

Sequents

Definition

A **sequent** is an expression of the form

$$\phi_1, \ldots, \phi_n \vdash \psi$$

where $\phi_1, \ldots, \phi_n, \psi$ are propositional formulas.

The formulas $\phi_1, \ldots, \phi_n$ are called **assumptions**, ψ is called **goal**.

remarks:

- **intuitively** $\boxed{\phi_1, \ldots, \phi_n \vdash \psi}$ means goal ψ follows from $\{\phi_1, \ldots, \phi_n\}$
- **special case** $n = 0$:
  - written as $\boxed{\vdash \psi}$
  - meaning: we have to prove that ψ is valid
- **notation**: for sequent $\phi_1, \ldots, \phi_n \vdash \psi$, we write $K \ldots \phi_i \vdash \psi$ if we are only interested in assumption ϕ_i
- the assumptions are **orderless** not ordered

Axioms

- axiom **"goal in assumption"**:
  If the goal is among the assumptions, the goal can be proved.

$$\text{GoalAssum}\ \frac{}{K\ldots,\psi\ \vdash\ \psi}$$

- axiom **"contradiction in assumptions"**:
  If the assumptions are contradicting, anything can be proved.

$$\text{ContrAssum}\ \frac{}{K\ldots,\phi,\neg\phi\ \vdash\ \psi}$$

Negation Rules

- rules **"contradiction"**:

$$\text{A-}\neg\ \frac{K\ldots,\neg\psi\ \vdash\ \phi}{K\ldots,\neg\phi\ \vdash\ \psi} \qquad\qquad \text{P-}\neg\ \frac{K\ldots,\phi\ \vdash\ \bot}{K\ldots\ \vdash\ \neg\phi}$$

A-¬: We know $\neg\phi$ and have to prove ψ. Thus we may assume $\neg\psi$ and prove ϕ.

P-¬: We have to prove $\neg\phi$. Thus we may assume ϕ and derive a contradition.

- rules **"elimination of double negation"**:

$$\text{P-}\neg_d\ \frac{K\ldots\ \vdash\ \psi}{K\ldots\ \vdash\ \neg\neg\psi} \qquad\qquad \text{A-}\neg_d\ \frac{K\ldots,\phi\ \vdash\ \psi}{K\ldots,\neg\neg\phi\ \vdash\ \psi}$$

Binary Connective Rules

- rules **"conjunction"**:

$$\text{A-}\wedge\ \frac{K\ldots,\phi_1,\phi_2\ \vdash\ \psi}{K\ldots,\phi_1\wedge\phi_2\ \vdash\ \psi}$$

$$\text{P-}\wedge\ \frac{K\ldots\ \vdash\ \psi_1 \qquad K\ldots\ \vdash\ \psi_2}{K\ldots\ \vdash\ \psi_1\wedge\psi_2}$$

- rules **"disjunction"**:

$$\text{P-}\vee\ \frac{K\ldots,\neg\psi_1\ \vdash\ \psi_2}{K\ldots\ \vdash\ \psi_1\vee\psi_2}$$

$$\text{P-}\vee\ \frac{K\ldots,\neg\psi_2\ \vdash\ \psi_1}{K\ldots\ \vdash\ \psi_1\vee\psi_2}$$

$$\text{A-}\vee\ \frac{K\ldots,\phi_1\ \vdash\ \psi \qquad K\ldots,\phi_2\ \vdash\ \psi}{K\ldots,\phi_1\vee\phi_2\ \vdash\ \psi}$$

P-$\vee$: indeterministic!!!

Rules for other connectives like implication “$\rightarrow$” and equivalence “$\leftrightarrow$” are constructed accordingly.

Some Remarks on Sequent Calculus

- **premises** of a rule: sequent(s) above the line
- **conclusion** of a rule: sequent below the line
- **axiom**: rule without premises
- **non-deterministic rule**: P-$\vee$
- **further non-determinism**: decision which rule to apply next
- **rules with case split**: P-$\wedge$, A-$\vee$
- **proof of formula ψ**
  1. start with $\vdash \psi$
  2. apply rules from bottom to top as long as possible, i.e., for given conclusion, find suitable premise(s)
  3. if finally all sequents are axioms then ψ is valid
- note: there are many variants of the sequent calculus

Computing with Sequent Calculus

Algorithm: entails

Data: set of assumptions $\mathcal{A}$, formula ψ
Result: **1** iff $\mathcal{A}$ entails ψ, i.e., $\mathcal{A} \models \psi$

if ($\psi \in \mathcal{A}$) **or** ($\phi, \neg\phi \in \mathcal{A}$) **then return 1**;
if $\mathcal{A} \cup \{\psi\}$ **contains only literals then return 0**;

if $\psi = \neg\neg\psi'$ **then return entails** $(\mathcal{A}, \psi')$;
if $\neg\neg\phi \in \mathcal{A}$ **then return entails** $(\mathcal{A}\setminus\{\neg\neg\phi\} \cup \{\phi\}, \psi)$;
if $\neg\phi \in \mathcal{A}$ **then return entails** $(\mathcal{A}\setminus\{\neg\phi\} \cup \{\neg\psi\}, \phi)$;
if $\phi_1 \wedge \phi_2 \in \mathcal{A}$ **then return entails** $(\mathcal{A}\setminus\{\phi_1 \wedge \phi_2\} \cup \{\phi_1, \phi_2\}, \psi)$;
if $\phi_1 \vee \phi_2 \in \mathcal{A}$ **then return entails** $(\mathcal{A} \cup \{\phi_1\}, \psi)$ **&& entails** $(\mathcal{A} \cup \{\phi_2\}, \psi)$;
switch ψ **do**
  case $\neg\psi'$ **do return entails** $(\mathcal{A} \cup \{\psi'\}, \bot)$;
  case $\psi_1 \vee \psi_2$ **do return entails** $(\mathcal{A} \cup \{\neg\psi_1\}, \psi_2)$ || **entails** $(\mathcal{A} \cup \{\neg\psi_2\}, \psi_1)$;
  case $\psi_1 \wedge \psi_2$ **do return entails** $(\mathcal{A}, \psi_1)$ **&& entails** $(\mathcal{A}, \psi_2)$;

Proving XOR stronger than OR

$$
\begin{prooftree}
\AxiomC{}
\LeftLabel{GoalAssum}
\UnaryInfC{$b, (\neg a \lor \neg b), \neg a \ \vdash \ b$}
\AxiomC{}
\LeftLabel{ContrAssum}
\UnaryInfC{$a, (\neg a \lor \neg b), \neg a \ \vdash \ b$}
\LeftLabel{A-$\lor$}
\BinaryInfC{$(a \lor b), (\neg a \lor \neg b), \neg a \ \vdash \ b$}
\LeftLabel{P-$\lor$}
\UnaryInfC{$(a \lor b), (\neg a \lor \neg b) \ \vdash \ a \lor b$}
\LeftLabel{A-$\land$}
\UnaryInfC{$(a \lor b) \land (\neg a \lor \neg b) \ \vdash \ a \lor b$}
\LeftLabel{A-$\neg_d$}
\UnaryInfC{$\neg\neg((a \lor b) \land (\neg a \lor \neg b)) \ \vdash \ a \lor b$}
\LeftLabel{P-$\lor$}
\UnaryInfC{$\vdash \ \neg((a \lor b) \land (\neg a \lor \neg b)) \lor (a \lor b)$}
\end{prooftree}
$$

proof direction

Refuting XOR stronger than AND

\begin{prooftree}
\AxiomC{}
\LeftLabel{GAss}
\UnaryInfC{$a, (\neg a \vee \neg b) \;\vdash\; a$}
\AxiomC{}
\LeftLabel{CAss}
\UnaryInfC{$b, \neg b \;\vdash\; a$}
\AxiomC{$\color{red}{b, \neg a \;\vdash\; a}$}
\LeftLabel{A-$\vee$}
\BinaryInfC{$b, (\neg a \vee \neg b) \;\vdash\; a$}
\LeftLabel{A-$\vee$}
\BinaryInfC{$(a \vee b), (\neg a \vee \neg b) \;\vdash\; a$}
\AxiomC{$\vdots$}
\AxiomC{$\vdots$}
\LeftLabel{A-$\vee$}
\BinaryInfC{$(a \vee b), (\neg a \vee \neg b) \;\vdash\; b$}
\LeftLabel{P-$\wedge$}
\BinaryInfC{$(a \vee b), (\neg a \vee \neg b) \;\vdash\; a \wedge b$}
\LeftLabel{A-$\wedge$}
\UnaryInfC{$(a \vee b) \wedge (\neg a \vee \neg b) \;\vdash\; a \wedge b$}
\LeftLabel{A-$\neg_d$}
\UnaryInfC{$\neg\neg((a \vee b) \wedge (\neg a \vee \neg b)) \;\vdash\; a \wedge b$}
\LeftLabel{P-$\vee$}
\UnaryInfC{$\vdash\; \neg((a \vee b) \wedge (\neg a \vee \neg b)) \vee (a \wedge b)$}
\end{prooftree}

counter example to validity: $a = \bot, b = \top$

Soundness and Completeness

For any calculus important properties are,
soundness, i.e. the question **“Can only valid formulas be shown as valid?”** and
completeness, i.e. the question **”Is there a proof for every valid formula?”**.

Soundness

If a formula is shown to be valid in the Gentzen Calculus, then it is valid.

Completeness

Every valid formula can be proven to be valid in the Gentzen Calculus.

PROPOSITIONAL FORMULAS: SOLVING WITH DPLL

VL Logic

Part I: Propositional Logic

JOHANNES KEPLER
UNIVERSITY LINZ

Proving Formulas in Normal Form

- In practice, formulas of arbitrary structure are quite challenging to handle
  - tree structure
  - simplifications affect only subtrees
- We have seen that CNF and DNF are able to represent every formula
  - so why not use them as input for SAT?
- **Conjunctive Normal Form**
  - refutability is easy to show
  - CNF can be efficiently calculated (polynomial)
- **Disjunctive Normal Form**
  - satisfiability is easy to show
  - complexity is in getting the DNF
- CNF and DNF can be obtained from the **truth tables**
  - exponential many assignments have to be considered
- alternative approach
  - **structural rewritings** are (satisfiability) equivalence preserving

DPLL Overview

The DPLL algorithm is ...

- **old** (invented 1962)
- **easy** (basic pseudo-code is less than 10 lines)
- **popular** (well investigated; also theoretical properties)
- usually realized for **formulas in CNF**
- using **binary constraint propagation (BCP)**
- in its modern form as **conflict drive clause learning (CDCL)** basis for state-of-the-art SAT solvers

Binary Constraint Propagation

Definition (Binary Constraint Propagation (BCP))

Let ϕ be a formula in CNF containing a unit clause C, i.e., ϕ has a clause $C = (l)$ which consists only of literal l. Then $BCP(\phi, l)$ is obtained from ϕ by

- removing all clauses with l
- removing all occurrences of $\bar{l}$

- BCP on variable x can trigger application of BCP on variable y
- if BCP produces the empty clause, then the formula is unsatisfiable
- if BCP produces the empty CNF, then the formula is satisfiable

Example

$\phi = \{(\neg a \vee b \vee \neg c), (a \vee b), (\neg a \vee \neg b), (a)\}$

1. $\phi' = BCP(\phi, a) = \{(b \vee \neg c), (\neg b)\}$
2. $\phi'' = BCP(\phi', \neg b) = \{(\neg c)\}$
3. $\phi'' = BCP(\phi', c) = \{\} = \top$

DPLL Algorithm

Algorithm: evaluate

Data: formula ϕ in CNF
Result: **1** iff ϕ satisfiable

while 1 do
    ϕ = BCP(ϕ)
    if ϕ**==** $\top$ **then return 1** ;
    if $\phi == \perp$ **then**
        if stack.isEmpty() then return 0 ;
        (l, ϕ) = stack.pop ()
        $\phi = \phi \wedge l$
    else
        select literal l occurring in ϕ
        stack.push($\bar{l}$, ϕ)
        $\phi = \phi \wedge l$

Some Remarks on DPLL

- DPLL is the basis for most state-of-the-art SAT solvers
  - Lingeling `http://fmv.jku.at/lingeling`
  - CaDiCaL `http://fmv.jku.at/cadical`
  - some more established solvers: MiniSAT, PicoSAT, Glucose, ...
- DPLL alone is not enough - powerful optimizations required for efficiency:
  - learning and non-chronological back-tracking (CDCL)
  - reset strategies and phase-saving
  - compact lazy data-structures
  - variable selection heuristics
  - usually combined with preprocessing before and inprocessing during search
- variants of DPLL are also used for other logics:
  - quantified propositional logic (QBF)
  - satisfiability modulo theories (SMT)
- challenge to parallelize
  - some successful attempts: ManySAT, Plingeling, Penelope, Treengeling, ...