LOGIC
PROPOSITIONAL REASONING

WS 2018/2019 (342.208)

Armin Biere   biere@jku.at
Martina Seidl   martina.seidl@jku.at

Institute for Formal Models and Verification
Johannes Kepler Universität Linz

Version 2019.1

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
$[\phi_1]_\nu = \mathbf{1}, \ldots [\phi_n]_\nu = \mathbf{1}$ implies that $[\psi]_\nu = \mathbf{1}$.

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 $(\phi_1 \wedge \ldots \wedge \phi_n) \to \psi$ 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:

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

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 $a \vee b$
- $\bot$ is the strongest formula
- $\top$ is the weakest formula

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

Axiom and Structural Rules

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

$$\text{GoalAssum} \quad \overline{K \ldots, \psi \ \vdash \ \psi}$$

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

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

- rule **"add valid assumption"**:

$$\text{ValidAssum} \quad \frac{K \ldots, \phi \ \vdash \ \psi}{K \ldots \ \vdash \ \psi} \ \text{if } \phi \text{ is valid}$$

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"**:

$$\text{A-}\wedge \ \frac{K\ldots, \phi_1, \phi_2 \ \vdash \ \psi}{K\ldots, \phi_1 \wedge \phi_2 \ \vdash \ \psi} \qquad\qquad \text{P-}\wedge \ \frac{K\ldots \ \vdash \ \psi_1 \qquad K\ldots \ \vdash \ \psi_2}{K\ldots \ \vdash \ \psi_1 \wedge \psi_2}$$

■ rules **"disjunction"**:

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

$$\text{P-}\vee \ \frac{K\ldots, \neg\psi_2 \ \vdash \ \psi_1}{K\ldots \ \vdash \ \psi_1 \vee \psi_2} \qquad\qquad \text{A-}\vee \ \frac{K\ldots, \phi_1 \ \vdash \ \psi \qquad K\ldots, \phi_2 \ \vdash \ \psi}{K\ldots, \phi_1 \vee \phi_2 \ \vdash \ \psi}$$

P-∨: indeterministic!!!

Rules for other connectives like implication "→" and equivalence "↔"
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

1 Algorithm: entails

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

2 **if** $\psi = \neg\neg\psi'$ **then return** *entails* $(\mathcal{A}, \psi')$;

3 **if** $\neg\neg\phi \in \mathcal{A}$ **then return** *entails* $(\mathcal{A} \setminus \{\neg\neg\phi\} \cup \{\phi\}, \psi)$;

4 **if** $\phi_1 \wedge \phi_2 \in \mathcal{A}$ **then return** *entails* $(\mathcal{A} \setminus \{\phi_1 \wedge \phi_2\} \cup \{\phi_1, \phi_2\}, \psi)$;

5 **if** *($\psi \in \mathcal{A}$) or ($\phi, \neg\phi \in \mathcal{A}$)* **then return** *1*;

6 **if** $\mathcal{A} \cup \{\psi\}$ *contains only literals* **then return** *0*;

7 **switch** ψ **do**

8   **case** $\bot$ **do**

9   **if** $\neg\phi \in \mathcal{A}$ **then return** *entails* $(\mathcal{A} \setminus \{\neg\phi\}, \phi)$;

10   **if** $\phi_1 \vee \phi_2 \in \mathcal{A}$ **then**

11   **if** *! entails* $(\mathcal{A} \setminus \{\neg\phi_1 \vee \phi_2\} \cup \{\phi_1\}, \bot)$ **then return** *0*;

12   **else return** *entails* $(\mathcal{A} \setminus \{\neg\phi_1 \vee \phi_2\} \cup \{\phi_2\}, \bot)$;

13   **case** x *where x is a variable* **do return** *entails* $(\mathcal{A} \cup \{\neg x\}, \bot)$;

14   **case** $\neg\psi'$ **do return** *entails* $(\mathcal{A} \cup \{\psi'\}, \bot)$;

15   **case** $\psi_1 \vee \psi_2$ **do return** *entails* $(\mathcal{A} \cup \{\neg\psi_1\}, \psi_2)$;

16   **case** $\psi_1 \wedge \psi_2$ **do return** *entails* $(\mathcal{A}, \psi_1)$ *&& entails* $(\mathcal{A}, \psi_2)$;

Proving XOR stronger than OR

$$\cfrac{\text{GoalAssum } \cfrac{}{b, (\neg a \vee \neg b), \neg a \;\vdash\; b} \qquad \text{ContrAssum } \cfrac{}{a, (\neg a \vee \neg b), \neg a \;\vdash\; b}}{\cfrac{(a \vee b), (\neg a \vee \neg b), \neg a \;\vdash\; b}{\cfrac{(a \vee b), (\neg a \vee \neg b) \;\vdash\; a \vee b}{\cfrac{(a \vee b) \wedge (\neg a \vee \neg b) \;\vdash\; a \vee b}{\cfrac{\neg\neg((a \vee b) \wedge (\neg a \vee \neg b)) \;\vdash\; a \vee b}{\vdash\; \neg((a \vee b) \wedge (\neg a \vee \neg b)) \vee (a \vee b)} \text{P-}\vee} \text{A-}\neg_d} \text{A-}\wedge} \text{P-}\vee} \text{A-}\vee$$

proof direction

Refuting XOR stronger than AND

$$
\cfrac{
  \text{GAss } \cfrac{}{a, (\neg a \vee \neg b) \;\vdash\; a}
}{
  \text{A-}\vee \cfrac{(a \vee b), (\neg a \vee \neg b) \;\vdash\; a}{}
}
$$

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.

Proof sketch:
Consider each rule individually and show that from valid premises only valid
conclusions can be drawn.

Completeness

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

Proof sketch:
Show algorithm terminates and that there is at least one case where it returns false if
the formula is not valid.

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

Resolution

- the **resolution calculus** consists of the single resolution rule

$$\frac{x \vee C \qquad \neg x \vee D}{C \vee D}$$

  - $\square$ C and D are (possibly empty) clauses
  - $\square$ the clause $C \vee D$ is called **resolvent**
  - $\square$ variable x is called **pivot**
  - $\square$ usually antecedent clauses $x \vee C$ and $\neg x \vee D$ are assumed not to be tautological, i.e., $x \notin C$ and $x \notin D$.

- in other words:
  $(\neg x \to C), (x \to D) \models C \vee D$

- resolution is **sound** and **complete**.

Example

one application of resolution

$$\frac{x \vee y \vee \neg z \qquad \neg x \vee y' \vee \neg z}{y \vee \neg z \vee y'}$$

derivation of empty clause:

$$\frac{y \qquad \neg y}{\bot}$$
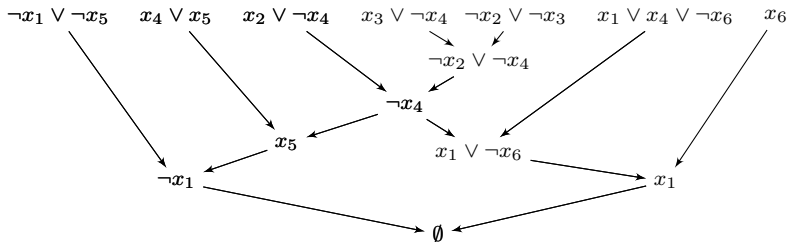
derivation of tautology:

$$\frac{x \vee a \qquad \neg x \vee \neg a}{a \vee \neg a}$$

- the resolution calculus works only on formulas in CNF

- if the empty clause can be derived then the formula is **unsatisfiable**

- if no new clause can be generated by application of the resolution rule then the formula is **satisfiable**

Resolution Example

We **prove unsatisfiability** of

$$\{(\neg x_1 \vee \neg x_5), (x_4 \vee x_5), (x_2 \vee \neg x_4), (x_3 \vee \neg x_4), (\neg x_2 \vee \neg x_3), (x_1 \vee x_4 \vee \neg x_6), (x_6)\}$$

as follows:

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

1 Algorithm: evaluate

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

2 while *1* **do**
3 $\quad$ ϕ = BCP(ϕ)
4 $\quad$ **if** $\phi == \top$ **then return** *1* ;
5 $\quad$ **if** $\phi == \bot$ **then**
6 $\quad\quad$ **if** *stack.isEmpty()* **then return** *0* ;
7 $\quad\quad$ (l, ϕ) = stack.pop ()
8 $\quad\quad$ $\phi = \phi \wedge l$
9 $\quad$ **else**
10 $\quad\quad$ select literal l occurring in ϕ
11 $\quad\quad$ stack.push($\bar{l}, \phi$)
12 $\quad\quad$ $\phi = \phi \wedge l$

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, . . .