	Roadmap
Propositional Logic & SAT Solvers Maria João Frade HASLab - INESC TEC Departamento de Informática, Universidade do Minho 2020/2021	 Classical Propositional Logic syntax; semantics; decision problems SAT and VAL; BDDs; normal forms; equisatisfiability; Tseitin's encoding; SAT solving algorithms: basic concepts; DPLL framework and its optimisations; modeling with PL.
Maria João Frade (HASLab, DI-UM) PL & SAT VF 2020/21 1/65	Maria João Frade (HASLab, DI-UM) PL & SAT VF 2020/21 2/65
 The language of propositional logic is based on propositions, or declarative sentences which one can, in principle, argue as being "true" or "false". "The capital of Portugal is Braga." "D. Afonso Herriques was the first king of Portugal." Propositional symbols are the atomic formulas of the language. More complex sentences are constructed using logical connectives. 	 The alphabet of the propositional language is organised into the following categories. Propositional variables: P, Q, R, ∈ V_{Prop} (a countably infinite set) Logical connectives: ⊥ (false) ,⊤ (true), ¬ (not), ∧ (and), ∨ (or), → (implies), ↔ (equivalent) Auxiliary symbols: "(" and ")". The set Form of formulas of propositional logic is given by the abstract syntax Form ∋ A, B ::= P ⊥ ⊤ (¬A) (A ∧ B) (A ∨ B) (A → B) (A ↔ B)
 In classical propositional logic (PL) each sentence is either true or false. 	We let A, B, C, F, G, H, \ldots range over Form .
• In fact, the content of the propositions is not relevant to PL. PL is not the study of truth, but of the relationship between the truth of one statement and that of another.	Outermost parenthesis are usually dropped. In absence of parentheses, we adopt the following convention about precedence. Ranging from the highest precedence to the lowest, we have respectively: \neg , \land , \lor , \rightarrow and \leftrightarrow . All binary connectives are right-associative.

Maria João Frade (HASLab, DI-UM)

PL & SAT

VF 2020/21 4/65

PL & SAT

VF 2020/21 3/65

Semantics

The semantics of a logic provides its meaning. What exactly is meaning? In propositional logic, meaning is given by the truth values *true* and *false*, where true \neq false. We will represent true by 1 and false by 0.

An *assignment* is a function $\mathcal{A} : \mathcal{V}_{\mathsf{Prop}} \rightarrow \{0, 1\}$, that assigns to every propositional variable a truth value.

An assignment \mathcal{A} naturally extends to all formulas, $\mathcal{A} : \mathbf{Form} \rightarrow \{0, 1\}$. The truth value of a formula is computed using *truth tables*:

F	A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \to B$	$A \leftrightarrow B$	\bot	Т
$\mathcal{A}_1(F)$	0	1	1	0	1	1	0	0	1
$\mathcal{A}_2(F)$	0	0	1	0	0	1	1	0	1
$\mathcal{A}_3(F)$	1	1	0	1	1	1	1	0	1
$\mathcal{A}_4(F)$	1	0	0	0	1	0	0	0	1

PL & SAT

Maria João Frade (HASLab, DI-UM)

VF 2020/21 5 / 65

VF 2020/21

7 / 65

Validity, satisfiability, and contradiction

A formula F is

- *valid* iff it holds under every assignment. We write $\models F$. A valid formula is called a *tautology*.
- satisfiable iff it holds under some assignment.
- *unsatisfiable* iff it holds under no assignment. An unsatisfiable formula is called a *contradiction*.
 - refutable iff it is not valid.

Proposition

F is valid iff $\neg F$ is unsatisfiable

PL & SAT

 $(A \wedge (A \to B)) \to B \text{ is valid.} \qquad A \to B \text{ is satisfiable and refutable.}$ $A \wedge \neg A \text{ is a contradiction.}$

Maria João Frade (HASLab, DI-UM)

Semantics

Let \mathcal{A} be an assignment and let F be a formula. If $\mathcal{A}(F) = 1$, then we say F holds under assignment \mathcal{A} , or \mathcal{A} models F. We write $\mathcal{A} \models F$ iff $\mathcal{A}(F) = 1$, and $\mathcal{A} \not\models F$ iff $\mathcal{A}(F) = 0$.

An alternative (inductive) definition of $\mathcal{A} \models F$ is

$\mathcal{A} \models \top$		
$\mathcal{A} eq \perp$		
$\mathcal{A} \models P$	iff	$\mathcal{A}(P) = 1$
$\mathcal{A} \models \neg A$	iff	$\mathcal{A} \not\models A$
$\mathcal{A} \models A \land B$	iff	$\mathcal{A} \models A$ and $\mathcal{A} \models B$
$\mathcal{A} \models A \lor B$	iff	$\mathcal{A} \models A \text{ or } \mathcal{A} \models B$
$\mathcal{A} \models A \to B$	iff	$\mathcal{A} \not\models A$ or $\mathcal{A} \models B$
$\mathcal{A} \models A \leftrightarrow B$	iff	$\mathcal{A} \models A \text{ iff } \mathcal{A} \models B$

PL & SAT

Maria João Frade (HASLab, DI-UM)

Consequence and equivalence

• $F \models G$ iff for every assignment A, if $A \models F$ then $A \models G$. We say G is a *consequence* of F.

• $F \equiv G$ iff $F \models G$ and $G \models F$. We say F and G are *equivalent*.

Let Γ = {F₁, F₂, F₃,...} be a set of formulas. *A* ⊨ Γ iff *A* ⊨ F_i for each formula F_i in Γ. We say *A* models Γ.
Γ ⊨ G iff *A* ⊨ Γ implies *A* ⊨ G for every assignment *A*. We say G is a consequence of Γ.

Proposition• $F \models G$ iff $\models F \rightarrow G$

• $\Gamma \models G$ and Γ finite iff $\models \bigwedge \Gamma \to G$

Maria João Frade (HASLab, DI-UM)

PL & SAT

VF 2020/21 6/65

Some basic equivalences

$\begin{array}{c} A \lor A \\ A \land A \end{array}$	≡	A A	$\begin{array}{c} A \wedge \neg A \\ A \vee \neg A \end{array}$	≡	⊥ ⊤
$\begin{array}{c} A \lor B \\ A \land B \end{array}$	≡	$\begin{array}{c} B \lor A \\ B \land A \end{array}$	$\begin{array}{c} A \wedge \top \\ A \vee \top \end{array}$	≡	$A \\ \top$
$A \wedge (A \vee B)$	≡	A	$A \wedge \bot$	=	\perp
$\begin{array}{l} A \land (B \lor C) \\ A \lor (B \land C) \end{array}$	≡	$\begin{array}{l} (A \wedge B) \lor (A \wedge C) \\ (A \lor B) \land (A \lor C) \end{array}$	$\neg \neg A$	=	A
$\neg (A \lor B) \\ \neg (A \land B)$	=	$\neg A \land \neg B \\ \neg A \lor \neg B$	$A \rightarrow B$	≡	$\neg A \lor B$

Maria João Frade (HASLab, DI-UM)

VF 2020/21 9/65

Theories

A set of formulas \mathcal{T} is *closed* under logical consequence iff for all formulas F, if $\mathcal{T} \models F$ then $F \in \mathcal{T}$.

PL & SAT

 \mathcal{T} is a *theory* iff it is closed under logical consequence. The elements of \mathcal{T} are called *theorems*.

Let Γ be a set of formulas.

 $\mathcal{T}(\Gamma) = \{F \mid \Gamma \models F\}$ is called the *theory of* Γ . The formulas of Γ are called *axioms* and the theory $\mathcal{T}(\Gamma)$ is *axiomatizable*.

Consistency

- Let $\Gamma = \{F_1, F_2, F_3, \dots\}$ be a set of formulas.
 - Γ is *consistent* or *satisfiable* iff there is an assignment that models Γ .
 - We say that Γ is *inconsistent* or *unsatisfiable* iff it is not consistent and denote this by Γ ⊨ ⊥.

Proposition

• $\{F, \neg F\} \models \bot$ • If $\Gamma \models \bot$ and $\Gamma \subseteq \Gamma'$, then $\Gamma' \models \bot$. • $\Gamma \models F$ iff $\Gamma, \neg F \models \bot$

Maria João Frade (HASLab, DI-UM)

VF 2020/21 10 / 65

Substitution

• Formula G is a *subformula* of formula F if it occurs syntactically within F.

PL & SAT

• Formula G is a strict subformula of F if G is a subformula of F and $G \neq F$

Substitution theorem

Suppose $F \equiv G$. Let H be a formula that contains F as a subformula. Let H' be the formula obtained by replacing some occurrence of F in H with G. Then $H \equiv H'$.

Maria João Frade (HASLab, DI-UM)

PL & SAT

VF 2020/21 11/65

Maria João Frade (HASLab, DI-UM)

PL & SAT

Adquate sets of connectives for PL

There is some redundancy among the logical connectives.

Some smaller adquate sets of conectives for PL:

$$\{\wedge, \neg\} \qquad \bot \equiv P \land \neg P, \quad \top \equiv \neg (P \land \neg P), \\ A \lor B \equiv \neg (\neg A \land \neg B), \quad A \to B \equiv \neg (A \land \neg B)$$

- $\begin{cases} \lor, \neg \end{cases} \qquad \top \equiv A \lor \neg A, \quad \bot \equiv \neg (A \lor \neg A), \\ A \land B \equiv \neg (\neg A \lor \neg B), \quad A \to B \equiv A \lor B \end{cases}$
- $\{ \rightarrow, \neg \} \qquad \top \equiv A \rightarrow A, \quad \bot \equiv \neg (A \rightarrow A), \\ A \lor B \equiv \neg A \rightarrow B, \quad A \land B \equiv \neg (A \rightarrow \neg B)$
- $\begin{cases} \rightarrow, \bot \\ & \neg A \equiv A \rightarrow \bot, \quad \top \equiv A \rightarrow A, \\ & A \lor B \equiv (A \rightarrow \bot) \rightarrow B), \quad A \land B \equiv (A \rightarrow B \rightarrow \bot) \rightarrow \bot$

$$A \leftrightarrow B \equiv (A \to B) \land (B \to A)$$

Maria João Frade (HASLab, DI-UM)

VF 2020/21 13/65

VF 2020/21

15 / 65

Decidability

Any algorithm that works for one of these problems also works for all of these problems!

PL & SAT

 $\begin{array}{ll} F \text{ is satisfiable} & \text{iff} & \neg F \text{ is not valid} \\ F \models G & \text{iff} & \neg (F \rightarrow G) \text{ is not satisfiable} \\ F \equiv G & \text{iff} & F \models G \text{ and } G \models F \\ F \text{ is valid} & \text{iff} & F \equiv \top \end{array}$

Truth-table method

For the satisfiability problem, we first compute a truth table for F and then check to see if its truth value is ever one.

This algorithm certainly works, but is very inefficient. It's exponential-time! $\mathcal{O}(2^n)$

If F has n atomic formulas, then the truth table for F has 2^n rows.

```
Maria João Frade (HASLab, DI-UM)
```

PL & SAT

Decidability

A *decision problem* is any problem that, given certain input, asks a question to be answered with a "yes" or a "no".

A *solution* to a decision problem is a program that takes problem instances as input and always terminates, producing a correct "yes" or "no" output. A decision problem is *decidable* if it has a solution.

Given formulas F and G as input, we may ask:

Decision problems

Validity problem:	"Is F valid ?"
Satisfiability problem:	"Is F satisfiable ?"
Consequence problem:	"Is G a consequence of F ?"
Equivalence problem:	"Are F and G equivalent ?"

PL & SAT

All these problems are **decidable!**

Maria João Frade (HASLab, DI-UM)

Complexity

An algorithm is *polynomial-time* if there exists a polynomial p(x) such that given input of size n, the algorithm halts in fewer than p(n) steps. The class of all decision problems that can be resolved by some polynomial-time algorithm is denoted by **P** (or PTIME).

It is not known whether the satisfiability problem (and the other three decision problems) is in \mathbf{P} .

We do not know of a polynomial-time algorithm for satisfiability.

If it exists, then $\mathbf{P} = \mathbf{NP}$!

The Satisfiability problem for PL (PSAT) is **NP**-complete (it was the first one to be shown NP-complete).

Maria João Frade (HASLab, DI-UM)

PL & SAT

VF 2020/21 14 / 65

Complexity

- A *deterministic* algorithm is a step-by-step procedure. At any stage of the algorithm, the next step is completely determined.
- In contrast, a *non-deterministic* algorithm may have more than one possible "next step" at a given stage. That is, there may be more than one computation for a given input.

NP (non-deterministic polynomial-time) decision problems

Let PROB be an arbitrary decision problem. Given certain input, PROB produces an output of either "yes" or "no". Let Y be the set of all inputs for which PROB produces the output of "yes" and let N be the analogous set of inputs that produce output "no".

- If there exists a non-deterministic algorithm which, given input x, can produce the output "yes" in polynomial-time if and only if $x \in Y$, then PROB is in NP.
- If there exists a non-deterministic algorithm which, given input x, can produce the output "no" in polynomial-time if and only if $x \in N$, then PROB is in coNP.

PL & SAT

```
Maria João Frade (HASLab, DI-UM)
```

VF 2020/21 17/65

VF 2020/21

19/65

BDDs

- A *Binary Decision Diagram* (BDD) is a data structure that is used to represent Boolean formulas.
- The Boolean formula is represented as a rooted, directed, acyclic graph, which consists of several decision nodes (labeled by Boolean variables) and terminal nodes (0 and 1).
- Each decision node x has two child nodes called *low child* and *high child*. The edge from node x to o a low (resp. high) child represents an assignment of x to 0 (resp. 1).
- A BDD is called *ordered* if different variables appear in the same order on all paths from the root. A BDD is said to be *reduced* if the following rules have been applied to its graph:

PL & SAT

- Merge the terminal nodes into two nodes 0 and 1.
- Merge isomorphic subtrees.
- Remove redundant nodes.

Maria João Frade (HASLab, DI-UM)

Complexity

Essentially, a decision problem is in NP(coNP) if a "yes" ("no") answer can be obtained in polynomial-time by guessing.

Satisfiability problem is **NP**. Given a formula F compute an assignment A for F. If A(F) = 1, then F is satisfiable. Validity problem is **coNP**.

NP-complete

A decision problem Π is **NP**-complete if it is in **NP** and for every problem Π_1 in **NP**, Π_1 is polynomially reducible to Π ($\Pi_1 \propto \Pi$).

PL & SAT

Cook's theorem (1971)

PSAT is **NP**-complete.

Maria João Frade (HASLab, DI-UM)

VF 2020/21

18 / 65



BDDs

- A path from the root node to the 1-terminal (resp. 0-termimal) represents a (possibly partial) variable assignment for which the represented Boolean formula is true (resp. false).
- The process of turning a binary tree into a BDD is not very useful by itself, as one does not want to build the binary decision tree in the first place, owing to its exponential size.
- Instead, one creates the reduced ordered BDD directly: given a formula, its BDD is built recursively from the BDDs of its subexpressions.

PL & SAT

VF 2020/21 21/65

BDDs

However,

- Building the BDD for a given formula can take exponential space and time.
- The size of the BDD is extremely sensitive to variable ordering.
- Computing a optimal variable ordering for a BDD is an NP-complete problem.
- It is know that the BDD representation of certain Boolean formulas is exponential in size regardless of variable order.
- In practice, the Boolean formula that BDDs can represent are often limited to several hundred variables in size.

Nonetheless, BDDs are extensively used in the verification of HW circuits, and optimization and synthesis of logic circuits.

Another usual application of BDDs is symbolic model checking.

Maria João Frade (HASLab, DI-UM)	
----------------------------------	--

PL & SAT

VF 2020/21 23/65

ROBDDs [Bryant, 1986]

Reduced Ordered Binary Decision Diagrams (ROBDDs, or **BDD**s for short) are graph-based data structure for manipulating Boolean formulas.

- BDDs are *canonical* representation of Boolean formulas (if two formulas are equivalent, then their BDD representations are isomorphic) assuming the same variable ordering.
- Implications of canonicity:
 - ► All tautologies have the same BDD (a single node with "true").
 - ► All contradictions have the same BDD (a single node with "false").
 - To check if a formula is satisfiable check if its BDD is isomorphic to false.
- Checking for satisfiability, validity, or contradiction can be done in constant time for a given BDD.

PL & SAT

However ...

Maria João Frade (HASLab, DI-UM)

VF 2020/21 22 / 65

SAT solving algorithms

- There are several techniques and algorithms for SAT solving.
- The majority of modern SAT solvers can be classified into two main categories:
 - SAT solvers based on a stochastic local search: the solver guesses a full assignment, and then, if the formula is evaluated to false under this assignment, starts to flip values of variables according to some (greedy) heuristic.
 - SAT solvers based on the DPLL framework: optimizations to the Davis-Putnam-Logemann-Loveland algorithm (DPLL) which corresponds to backtrack search through the space of possible variable assignments.
- DPLL-based SAT solvers, however, are considered better in most cases.

Usually SAT solvers receive as input a formula in a specific syntatical format. So, one has first to transform the input formula to this specific format preserving satisfiability.

Maria João Frade (HASLab, DI-UM)

PL & SAT

VF 2020/21 24 / 65

Stochastic local search

- Local search is incomplete; usually it cannot prove unsatisfiability.
- However, it can be very effective in specific contexts.
- The algorithm:
 - Start with a (random) assignment,
 - And repeat a number of times:
 - * If not all clauses satisfied, change the value of a variable.
 - ★ If all clauses satisfied, it is done.
 - Repeat (random) selection of assignment a number of times.
- The algorithm terminates when a satisfying assignment is found or when a time bound is elapsed (inconclusive answer).

PL & SAT

Maria João Frade (HASLab, DI-UM)

VF 2020/21 25/65

Normalization

Transforming a formula F to equivalent formula F' in NNF can be computed by repeatedly replace any subformula that is an instance of the left-hand-side of one of the following equivalences by the corresponding right-hand-side

 $A \to B \equiv \neg A \lor B \qquad \neg \neg A \equiv A$ $\neg (A \land B) \equiv \neg A \lor \neg B \qquad \neg (A \lor B) \equiv \neg A \land \neg B$

This algoritm is linear on the size of the formula.

Normal forms

SAT solvers usually take input in *conjunctive normal form*.

- A *literal* is a propositional variable or its negation.
 A literal is *negative* if it is a negated atom, and *positive* otherwise.
- A formula *A* is in *negation normal form (NNF)*, if the only connectives used in *A* are ¬, ∧ and ∨, and negation only appear in literals.
- A *clause* is a disjunction of literals.
- A formula is in *conjunctive normal form (CNF)* if it is a conjunction of clauses, i.e., it has the form

PL & SAT

$$\bigwedge_i \left(\bigvee_j l_{ij}\right)$$

where l_{ij} is the j-th literal in the i-th clause.

Maria João Frade (HASLab, DI-UM)

Normalization

To transform a formula already in NNF into an equivalent CNF, apply recursively the following equivalences (left-to-right):

$A \lor (B \land C) \equiv (A \lor A)$	$(A \lor C) \land (A \lor C)$	$(A \land B) \lor C \equiv (A \lor C)$	$(C) \land (B \lor C)$
$A \wedge \bot \equiv \bot$	$\bot \wedge A \equiv \bot$	$A \wedge \top \equiv A$	$\top \wedge A \equiv A$
$A \lor \bot \equiv A$	$\bot \lor A \equiv A$	$A \vee \top \equiv \top$	$\top \lor A \equiv \top$

This althoritm converts a NNF formula into an equivalent CNF, but its worst case is exponential on the size of the formula.

VF 2020/21 27 / 65

Maria João Frade (HASLab, DI-UM)

VF 2020/21 26 / 65

Example

Compute the CNF of $((P \rightarrow Q) \rightarrow P) \rightarrow P$

The first step is to compute its NNF by transforming implications into disjunctions and pushing negations to proposition symbols:

$$(P \to Q) \to P) \to P \equiv \neg((P \to Q) \to P) \lor P$$
$$\equiv \neg(\neg(P \to Q) \lor P) \lor P$$
$$\equiv \neg(\neg(\neg P \lor Q) \lor P) \lor P$$
$$\equiv \neg((P \land \neg Q) \lor P) \lor P$$
$$\equiv (\neg(P \land \neg Q) \land \neg P) \lor P$$
$$\equiv ((\neg P \lor Q) \land \neg P) \lor P$$

To reach a CNF, distributivity is then applied to pull the conjunction outside:

 $((\neg P \lor Q) \land \neg P) \lor P \equiv (\neg P \lor Q \lor P) \land (\neg P \lor P)$

PL & SAT

```
Maria João Frade (HASLab, DI-UM)
```

VF 2020/21 29/65

Definitional CNF

Equisatisfiability

Two formulas F and F' are *equisatisfiable* when F is satisfiable iff F' is satisfiable.

Any propositional formula can be transformed into a equisatisfiable CNF formula with only linear increase in the size of the formula.

The price to be paid is n new Boolean variables, where n is the number of logical conectives in the formula.

This transformation can be done via *Tseitin's encoding* [Tseitin, 1968].

This tranformation compute what is called the *definitional CNF* of a formula, because they rely on the introduction of new proposition symbols that act as names for subformulas of the original formula.

Worst-case example

Compute the CNF of $(P_1 \land Q_1) \lor (P_2 \land Q_2) \lor \ldots \lor (P_n \land Q_n)$

 $\begin{array}{l} (P_1 \wedge Q_1) \vee (P_2 \wedge Q_2) \vee \ldots \vee (P_n \wedge Q_n) \\ \equiv & (P_1 \vee (P_2 \wedge Q_2) \vee \ldots \vee (P_n \wedge Q_n)) \wedge (Q_1 \vee (P_2 \wedge Q_2) \vee \ldots \vee (P_n \wedge Q_n)) \\ \equiv & \ldots \\ \equiv & (P_1 \vee \ldots \vee P_n) \wedge \\ & (P_1 \vee \ldots \vee P_{n-1} \vee Q_n) \wedge \\ & (P_1 \vee \ldots \vee P_{n-2} \vee Q_{n-1} \vee P_n) \wedge \\ & (P_1 \vee \ldots \vee P_{n-2} \vee Q_{n-1} \vee Q_n) \wedge \\ & \ldots \wedge \\ & (Q_1 \vee \ldots \vee Q_n) \end{array}$

The original formula has 2n literals, while the equivalent CNF has 2^n clauses, each with n literals.

PL & SAT

The size of the formula increases exponentially.

Maria João Frade (HASLab, DI-UM)

VF 2020/21

30 / 65

Tseitin's encoding

Tseitin transformation

- **1** Introduce a new fresh variable for each compound subformula.
- Assign new variable to each subformula.
- Second elecal constraints as CNF.
- **9** Make conjunction of local constraints and the root variable.
- This transformation produces a formula that is equisatisfiable: the result is satisfiable iff and only the original formula is satisfiable.
- One can get a satisfying assignment for original formula by projecting the satisfying assignment onto the original variables.

There are various optimizations that can be performed in order to reduce the size of the resulting formula and the number of additional variables.

PL & SAT

VF 2020/21 31/65

Maria João Frade (HASLab, DI-UM)



- The strict shape of CNFs make them particularly suited for checking validity problems.
 - A CNF is a tautology iff all of its clauses are tautologies.
 - \blacktriangleright A clause C is a tautology precisely when there exists a proposition symbol P such that both P and $\neg P$ are in C (such clauses are said to be *closed*).
 - So, a CNF is a tautology iff all of its clauses are closed.
- However, the applicability of this simple criterion for validity is compromised by the potential exponential growth in the CNF transformation.
- This limitation is overcomed considering instead SAT, with satisfiability preserving CNFs (definitional CNF). Recall that

 $\neg F$ is **unsatisfiable** F is valid iff

PL & SAT

VF 2020/21

35 / 65

Tseitin's encoding: a circuit to CNF



CNFs satisfiability

- A CNF is satisfied by an assignment if all its clauses are satisfied. And a *clause is satisfied* if at least one of its literals is satisfied.
- The ideia is to incrementally construct an assignment compatible with a CNF.
 - An assignment of a formula F is a function mapping F's variables to 1 or 0. We say it is
 - \star full if all of F's variables are assigned,
 - * and *partial* otherwise.
- Most current state-of-the-art SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) framework: in this framework the tool can be thought of as traversing and backtracking on a binary tree, in which
 - internal nodes represent partial assignments
 - and leaves represent full assignments

Maria João Frade (HASLab, DI-UM)

PL & SAT

Basic concepts

Pure literals

• A literal is *pure* if only occurs as a positive literal or as a negative literal in a CNF formula.

• Pure literal rule

Clauses containing pure literals can be removed from the formula (i.e. just assign pure literals to the values that satisfy the clauses).

• This technique was extensively used until the mid 90s, but nowadays seldom used.

Example

 ${\rm Let} \ F \ {\rm be} \ \ (Q \lor P) \land (R \lor Q \lor \neg P) \land (\neg R \lor P) \land (P \lor \neg X).$

- Q and $\neg X$ are pure literals in F.
- The resulting (equisatisfiable) formula is $(\neg R \lor P)$.

Maria João Frade (HASLab, DI-UM)

VF 2020/21 37 / 65

Basic concepts

Resolution

• Resolution rule If a formula F contains clauses $(A \lor P)$ and $(\neg P \lor B)$, then one can infer $(A \lor B)$. The formula $(A \lor B)$ is called the *resolvent*.

PL & SAT

- The resolvent can be added as a conjunction to *F* to produce an equivalent formula still in CNF.
- If even \perp is deduced via resolution, F must be unsatisfiable.
- A CNF formula that does not contain \perp and to which no more resolutions can be applied represents all possible satisfying interpretations.

The resolvent of $(\neg Q \lor X)$ with $(Q \lor A)$ is $(X \lor A)$. Resolving this with $\neg X$ we get A. Resolving this with $\neg A$ we get \bot . So, $(\neg Q \lor X) \land (Q \lor A) \land \neg X \land \neg A$ is UNSAT.

Basic concepts

Unit propagation (also called Boolean Constraint Propagation (BCP))

- A clause is a *unit* if all literals but one are assigned value 0, and the remaining literal is unassigned.
- Unit clause rule

Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied.

- *Unit propagation* is the iterated application of the unit clause rule.
- This technique is extensively used.
- Consider the partial assignment $\mathcal{A}(P) = 0, \mathcal{A}(Q) = 1$. Under this assignment $(P \lor \neg R \lor \neg Q)$ is a unit clause. R must be assigned the value 0.
- Consider the partial assignment $\mathcal{A}(R) = 1, \mathcal{A}(Q) = 1.$
 - By unit propagation we can conclude that $(P \lor \neg R \lor \neg Q) \land (\neg P \lor \neg Q \lor X) \land (\neg P \lor \neg R \lor X)$ is satisfiable.
 - What about $(P \lor \neg R \lor \neg Q) \land (\neg P \lor \neg Q \lor X) \land (\neg P \lor \neg R \lor \neg X)$?

Maria João Frade (HASLab, DI-UM)

PL & SAT

VF 2020/21 38 / 65

Historical perspective

• The DP algorithm [Davis&Putnam, 1960]

- Based on the resolution rule.
- Eliminate one variable at each step, using resolution.
- Applied the pure literal rule and unit propagation.
- The DPLL algorithm [Davis&Putnam&Logemann&Loveland, 1962]
 - Based on backtrack search.
 - Progresses by making a decision about a variable and its value.
 - Propagates implications of this decision that are easy to detect.
 - Backtracks in case a conflict is detected in the form of a falsified clause.
- In the last two decades, several enhancements have been introduced to improve the efficiency of DPLL-based SAT solving.

PL & SAT

VF 2020/21 39/65

Maria João Frade (HASLab, DI-UM)

DP algorithm

- Iteratively apply the following steps:
 - ► Select variable X.
 - Apply resolution rule between every pair of clauses of the form $(X \lor A)$ and $(\neg X \lor B)$.
 - Remove all clauses containing either X or $\neg X$.
 - Apply the pure literal rule and unit propagation.
- Terminate when either the empty clause (⊥) or the empty formula (⊤) is derived.

PL & SAT

• The algorithm is complete but inefficient.

Maria João Frade (HASLab, DI-UM)

VF 2020/21 41/65

CNFs (as sets of sets of literals)

• Recall that CNFs are formulas with the following shape (each l_{ij} denotes a literal):

 $(l_{11} \vee l_{12} \vee \ldots \vee l_{1k}) \wedge \ldots \wedge (l_{n1} \vee l_{n2} \vee \ldots \vee l_{nj})$

• Associativity, commutativity and idempotence of both disjunction and conjunction allow us to treat each CNF as a set of sets of literals S

 $S = \{\{l_{11}, l_{12}, \dots, l_{1k}\}, \dots, \{l_{n1}, l_{n2}, \dots, l_{nj}\}\}\$

• An empty inner set will be identified with $\bot,$ and an empty outer set with $\top.$ Therefore,

PL & SAT

- if $\{\} \in S$, then S is equivalent to \bot ;
- if $S = \{\}$, then S is \top .

DPLL algorithm

- Traditionally the DPLL algorithm is presented as a recursive procedure.
- The procedure DPLL is called with the CNF and a partial assignment.
- We will represent a CNF by a set of sets of literals.
- We will represent the partial assignment by a set of literals (P denote that P is set to 1, and $\neg P$ that P is set to 0).
- The algorithm:
 - Progresses by making a decision about a variable and its value.
 - Propagates implications of this decision that are easy to detect, simplifying the clauses.
 - Backtracks in case a conflict is detected in the form of a falsified clause.

Maria João Frade (HASLab, DI-UM)

VF 2020/21 42/65

Simplification of a clause under an assignment

If we fix the assignment of a particular proposition symbol, we are able to simplify the corresponding CNF accordingly.

PL & SAT

The *opposite* of a literal l, written -l, is defined by

 $-l = \begin{cases} \neg P & \text{, if } l = P \\ P & \text{, if } l = \neg P \end{cases}$

When we set a literal l to be true,

- any clause that has the literal *l* is now guaranteed to be satisfied, so we throw it away for the next part of the search.
- any clause that had the literal -l, on the other hand, must rely on one of the other literals in the clause, hence we throw out the literal -l before going forward.

Simplification of S assuming l holds

 $S|_{l} = \{c \setminus \{-l\} \mid c \in S \text{ and } l \notin c\}$

PL & SAT

VF 2020/21

43 / 65

Maria João Frade (HASLab, DI-UM)

VF 2020/21 44 / 65

Simplification of a clause under an assignment

If a CNF S contains a clause that consists of a single literal (called *unit clause*), we know for certain that the literal must be set to true and S can be simplified.

One should apply this rule while it is possible and worthwhile.

UNIT_PROPAGAT while $\{\} \notin S$ $S \leftarrow S _l$; $\mathcal{A} \leftarrow \mathcal{A} \cup \{d\}$ }	TE (S, \mathcal{A}) { and S has a unit clause l do { l }	
Maria João Frade (HASLab, I	DI-UM) PL & SAT V	F 2020/21 45/65
DPLL algorith Is $(\neg P \lor Q) \land (\neg P$	m $ee R) \land (Q \lor R) \land (\neg Q \lor \neg R) \land (P \lor \neg R \lor Q)$ s	atisfiable?
	S	
DPLL	$\{\{\neg P, Q\}, \{\neg P, R\}, \{Q, R\}, \{\neg Q, \neg R\}, \{P, \neg R, Q\}\}$	} Ø
UNIT_PROPAGATE	$\{\{\neg P, Q\}, \{\neg P, R\}, \{Q, R\}, \{\neg Q, \neg R\}, \{P, \neg R, Q\}\}$	Ø
choose $l = P$ DPLL $S _l$	$\{\{Q\}, \{B\}, \{Q, R\}, \{\neg Q, \neg R\}\}$	{ <i>P</i> }
UNIT_PROPAGATE		
-	{{}}	$\{P,Q,R\}$
$\frac{-i}{DPLL} \frac{S _{-l}}{S _{-l}}$	$\{\{Q,R\},\{\neg Q,\neg R\},\{\neg R,Q\}\}$	$\{\neg P\}$
UNIT_PROPAGATE	$\{\{Q, R\}, \{\neg Q, \neg R\}, \{\neg R, Q\}\}$	$\{\neg P\}$
choose $l = Q$ DPLL Sh	$\{\{\neg R\}\}$	$\{\neg P O\}$
UNIT_PROPAGATE		
	{} SAT	$\{\neg P, Q, \neg R\}$
Maria João Frada (HASLah J		F 2020/21 47 / 65
mana soao made (mastab, t		. 1010/21

DPLL algorithm

DPLL is called with a CNF S and a partial assignment $\mathcal A$ (initially \emptyset).

```
\begin{array}{l} \label{eq:DPLL(S, \mathcal{A}) \\ \text{UNIT_PROPAGATE}(S, \mathcal{A}); \\ \text{if } S = \{\} \text{ then return SAT}; \\ \text{else if } \{\} \in S \text{ then return UNSAT}; \\ \text{else } \{ l \leftarrow \texttt{a} \text{ literal of } S ; \\ & \text{if } \text{DPLL } (S|_l, \mathcal{A} \cup \{l\}) = \text{SAT then return SAT}; \\ & \text{else return } \text{DPLL } (S|_{-l}, \mathcal{A} \cup \{-l\}) \\ & \\ \} \end{array}
```

- DPLL complete algorithm for SAT.
- Unsatisfiability of the complete formula can only be detected after exhaustive search.

PL & SAT

Maria João Frade (HASLab, DI-UM)

VF 2020/21 46 / 65

DPLL recursive algorithm



DPLL-based iterative algorithm [Marques-Silva&Sakallah,1996]

At each step:

- **Decide** on the assignment of a variable (which is called the *decision variable*, and it will have a *decision level* associated with it).
- **Deduce** the consequences of the decision made. (Variables assigned will have the same decision level as the decision variable.)
 - ▶ If all the clauses are satisfied, then the instance is satisfiable.
 - If there exists a conflicting clause, then **analyze** the conflit and determine the decision level to backtrack. (The solver may perform some analysis and record some information from the current conflict in order to prune the search space for the future.)
 - Decision level ≤ 0 indicates that the formula is unsatisfiable.
 - Otherwise, proceed with another decision.

Different DPLL-based modern solvers differ mainly in the detailed implementation of each of these functions.

PL & SAT

```
Maria João Frade (HASLab, DI-UM)
```

VF 2020/21 49/65

DPLL framework: heuristics & optimizations

Many different techniques are applied to achieve efficiency in DPLL-based SAT solvers.

- Decision heuristic: a very important feature in SAT solving is the strategy by which the literals are chosen.
- Look-ahead: exploit information about the remaining search space.
 - unit propagation
 - ► pure literal rule
 - decision (spliting) heuristics
- Look-back: exploit information about search which has already taken place.
 - non-chronological backtracking (a.k.a. backjumping)
 - clause learning
- Other techniques:
 - preprocessing (detection of subsumed clauses, simplification, ...)
 - (random) restart (restarting the solver when it seams to be is a hopeless branch of the search tree)

PL & SAT

```
Maria João Frade (HASLab, DI-UM)
```

VF 2020/21 51/65

DPLL-based iterative algorithm

```
while(1) {
          decide_next_branch(); //branching heuristics
          while (true) {
               status = deduce(); //deduction mechanism
               if (status == CONF) {
                    bl = analyze_conflict(); //conflict analysis
                   if (bl == 0) return UNSAT:
                    else backtrack(bl);
               }
               else if (status == SATISFIABLE) return SAT;
               else break:
          }
     }
Maria João Frade (HASLab, DI-UM)
                                   PL & SAT
                                                              VF 2020/21 50 / 65
Conflict analysis and learning
   • Non-chronological backtracking: does not necessarily flip the last assignment
     and can backtrack to an earlier decision level.
   • The process of adding conflict clauses is generally referred to as learning.
   • The conflict clauses record the reasons deduced from the conflict to avoid
     making the same mistake in the future search. For that implication graphs
     are used.
   • Conflict-driven backtracking uses the conflict clauses learned to determine
     the actual reasons for the conflict and the decision level to backtrack in
```

PL & SAT

order to prevent the repetition of the same conflict.

Conflict-Driven Clause Learning (CDCL) solvers

- DPLL framework.
- New clauses are learnt from conflicts.
- Structure (implication graphs) of conflicts exploited.
- Backtracking can be non-chronological.
- Efficient data structures (compact and reduced maintenance overhead).
- Backtrack search is periodically restarted.
- Can deal with hundreds of thousand variables and tens of million clauses!

PL & SAT

Maria João Frade (HASLab, DI-UM)

VF 2020/21 53/65

DIMACS CNF format

- DIMACS CNF format is a standard format for CNF used by most SAT solvers.
- Plain text file with following structure:
 - c <comments>

```
p cnf <num.of variables> <num.of clauses>
<clause> 0
<clause> 0
```

```
. . .
```

. . .

- Every number 1, 2, . . . corresponds to a variable (variable names have to be mapped to numbers).
- A negative number denote the negation of the corresponding variable.
- Every clause is a list of numbers, separated by spaces. (One or more lines per clause).

PL & SAT

Modern SAT solvers

- In the last two decades, satisfiability procedures have undergone dramatic improvements in efficiency and expressiveness. Breakthrough systems like GRASP (1996), SATO (1997), Chaff (2001) and MiniSAT (2003) have introduced several enhancements to the efficiency of DPLL-based SAT solving.
- Modern SAT solvers can check formulas with hundreds of thousands variables and millions of clauses in a reasonable amount of time.
- New SAT solvers are introduced every year.
 - The satisfiability library SAT Live!¹ is an online resource that proposes, as a standard, a unified notation and a collection of benchmarks for performance evaluation and comparison of tools.
 - Such a uniform test-bed has been serving as a framework for regular tool competitions organised in the context of the regular SAT conferences.²

¹http://www.satlive.org ²http://www.satcompetition.org Maria João Frade (HASLab, DFUM) PL & SAT

VF 2020/21 54 / 65

DIMACS CNF format

Example

```
A_1 \wedge (A_1 \vee P) \wedge (\neg A_1 \vee \neg P \vee A_2) \wedge (A_1 \vee \neg A_2)
```

- We have 3 variables and 4 clauses.
- CNF file:

р	cr	ıf	З	4
1	0			
1	3	0		
-1	1 -	-3	2	C
1	-2	2 ()	

Maria João Frade (HASLab, DI-UM)

PL & SAT

Applications of SAT

- A large number of problems can be described in terms of satisfiability, including graph problems, planning, games, scheduling, software and hardware verification, extended static checking, optimization, test-case generation, among others.
- These problems can be encoded by propositional formulas and solved using SAT solvers.

problem $\mathcal{P} \longrightarrow$ formula $F \longrightarrow$ CNF converter \longrightarrow SAT solver

 $\begin{array}{ll} \mbox{SAT solver output:} & \mbox{If } F \mbox{ is satisfiable: } \mbox{sat } + \mbox{ model} \\ \mbox{If } F \mbox{ is unsatisfiable: } \mbox{unsat} + \mbox{proof} \\ \end{array}$

The satisfying assignments (models) of F are the solutions of \mathcal{P} .

• SAT solvers are core engines for other solvers (like SMT solvers).

PL & SAT

• SAT solvers may be integrated into theorem provers.

Maria João Frade (HASLab, DI-UM)

VF 2020/21 57/65

Modeling with PL

When can the meeting take place?

- Maria cannot meet on Wednesday.
- Peter can only meet either on Monday, Wednesday or Thursday.
- Anne cannot meet on Friday.
- Mike cannot meet neither on Tuesday nor on Thursday.

Encode into the following proposition:

```
\negWed \land (Mon \lor Wed \lor Thu) \land \negFri \land (\negTue \land \negThu)
```

PL & SAT



Graph coloring

Can one assign one of K colors to each of the vertices of graph $G=(V\!,E)$ such that adjacent vertices are assigned different colors?

- Create $|V| \times K$ variables:
 - ▶ $x_{ij} = 1$ iff vertex *i* is assigned color *j*;
 - $x_{ij} = 0$ otherwise.
- For each edge (u, v), require different assigned colors to u and v:

for each $1 \le j \le K$, $(x_{uj} \to \neg x_{vj})$

Maria João Frade (HASLab, DI-UM)

VF 2020/21

59 / 65

Maria João Frade (HASLab, DI-UM)

PL & SAT

VF 2020/21 60/65

Modeling with PL



• No more than one person per chair.

$$\bigwedge_{i=1}^{3} \bigwedge_{a=1}^{2} \bigwedge_{b=a+1}^{3} (\neg x_{ai} \lor \neg x_{bi})$$

PL & SAT

Maria João Frade (HASLab, DI-UM)

VF 2020/21

63 / 65

Modeling with PL

Placement of guests

We have three chairs in a row and we need to place Anne, Susan and Peter.

- Anne does not want to sit near Peter.
- Anne does not want to sit in the left chair.
- Susan does not want to sit to the right of Peter.

Can we satisfy these constrains?

- Denote: Anne = 1, Susan = 2, Peter = 3 left chair = 1, middle chair = 2, right chair = 3
- Introduce a propositional variable for each pair (person, place)

PL & SAT

• $x_{ij} = 1$ iff person *i* is sited in place *j*; $x_{ij} = 0$ otherwise

Maria João Frade (HASLab, DI-UM)

VF 2020/21 62/65

Modeling with PL

Original C code

Equivalence checking of if-then-else chains

Optimized C code

if(!a && !b) h();
else if(!a) g();
else f();

if(a) f(); else if(b) g(); else h();

Are these two programs equivalent?

- Model the variables a and b and the procedures that are called using the Boolean variables a, b, f, g, and h.
- **(a)** Compile if-then-else chains into Boolean formulae compile(if x then y else z) $\equiv (x \land y) \lor (\neg x \land z)$
- So Check the validity of the following formula compile(*original*) ↔ compile(*optimized*)

Reformulate it as a SAT problem: Is the Boolean formula \neg (compile(*original*) \leftrightarrow compile(*optimized*))

satisfiable?

Maria João Frade (HASLab, DI-UM)

PL & SAT

VF 2020/21 64 / 65

Proof system

- So far we have taken the *"semantic"* approach to logic, with the aim of characterising the semantic concept of model, from which validity, satisfiability and semantic entailment were derived.
- However, this is not the only possible point of view.
- Instead of adopting the view based on the notion of truth, we can think of logic as a codification of reasoning. This alternative approach to logic, called *"deductive"*, focuses directly on the deduction relation that is induced on formulas, i.e., on what formulas are logical consequences of other formulas.

PL & SAT

• We will explore this perspective later in this course.

VF 2020/21 65/65