# Decision Procedures for Verification

Part 1. Propositional Logic (3)

14.11.2016

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

# Last time

## 1.1 Syntax

- Language
  - propositional variables
  - logical symbols

    $\Rightarrow$ Boolean combinations
- Propositional Formulae

## 1.2 Semantics

- Valuations
- Truth value of a formula in a valuation
- Models, Validity, and Satisfiability
- Entailment and Equivalence

# Canonical forms

- CNF and DNF
- Computing CNF/DNF by rewriting the formulae
- Structure-Preserving Translation for CNF
- Optimized translation using polarity

# Decision Procedures for Satisfiability

- Simple Decision Procedures

  truth table method — Logik f. Informatiker

  Discrete Algebraic Structures

- The Resolution Procedure — started last time

  to be continued today

- The Davis-Putnam-Logemann-Loveland Algorithm — today

# 1.6 The Propositional Resolution Calculus

Resolution inference rule:

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

Terminology: $C \vee D$: resolvent; $A$: resolved atom

(Positive) factorisation inference rule:

$$\frac{C \vee A \vee A}{C \vee A}$$

# The Resolution Calculus *Res*

These are schematic inference rules; for each substitution of the schematic variables $C$, $D$, and $A$, respectively, by propositional clauses and atoms we obtain an inference rule.

As "$\vee$" is considered associative and commutative, we assume that $A$ and $\neg A$ can occur anywhere in their respective clauses.

# Soundness and Completeness of Resolution

**Theorem 1.10.** Propositional resolution is sound.

**Completeness:**

How to show refutational completeness of propositional resolution:

- We have to show: $N \models \bot \;\Rightarrow\; N \vdash_{Res} \bot$,
  or equivalently: If $N \not\vdash_{Res} \bot$, then $N$ has a model.

- Idea: Suppose that we have computed sufficiently many inferences (and not derived $\bot$).

  Now order the clauses in $N$ according to some appropriate ordering, inspect the clauses in ascending order, and construct a series of valuations.

- The limit valuation can be shown to be a model of $N$.

# Clause Orderings

1. We assume that $\succ$ is any fixed ordering on propositional variables that is *total* and well-founded.

2. Extend $\succ$ to an ordering $\succ_L$ on literals:

$$
\begin{array}{lcll}
[\neg]P & \succ_L & [\neg]Q & \text{, if } P \succ Q \\
\neg P & \succ_L & P &
\end{array}
$$

3. Extend $\succ_L$ to an ordering $\succ_C$ on clauses:
   $\succ_C = (\succ_L)_{\mathrm{mul}}$, the multi-set extension of $\succ_L$.

   *Notation:* $\succ$ also for $\succ_L$ and $\succ_C$.

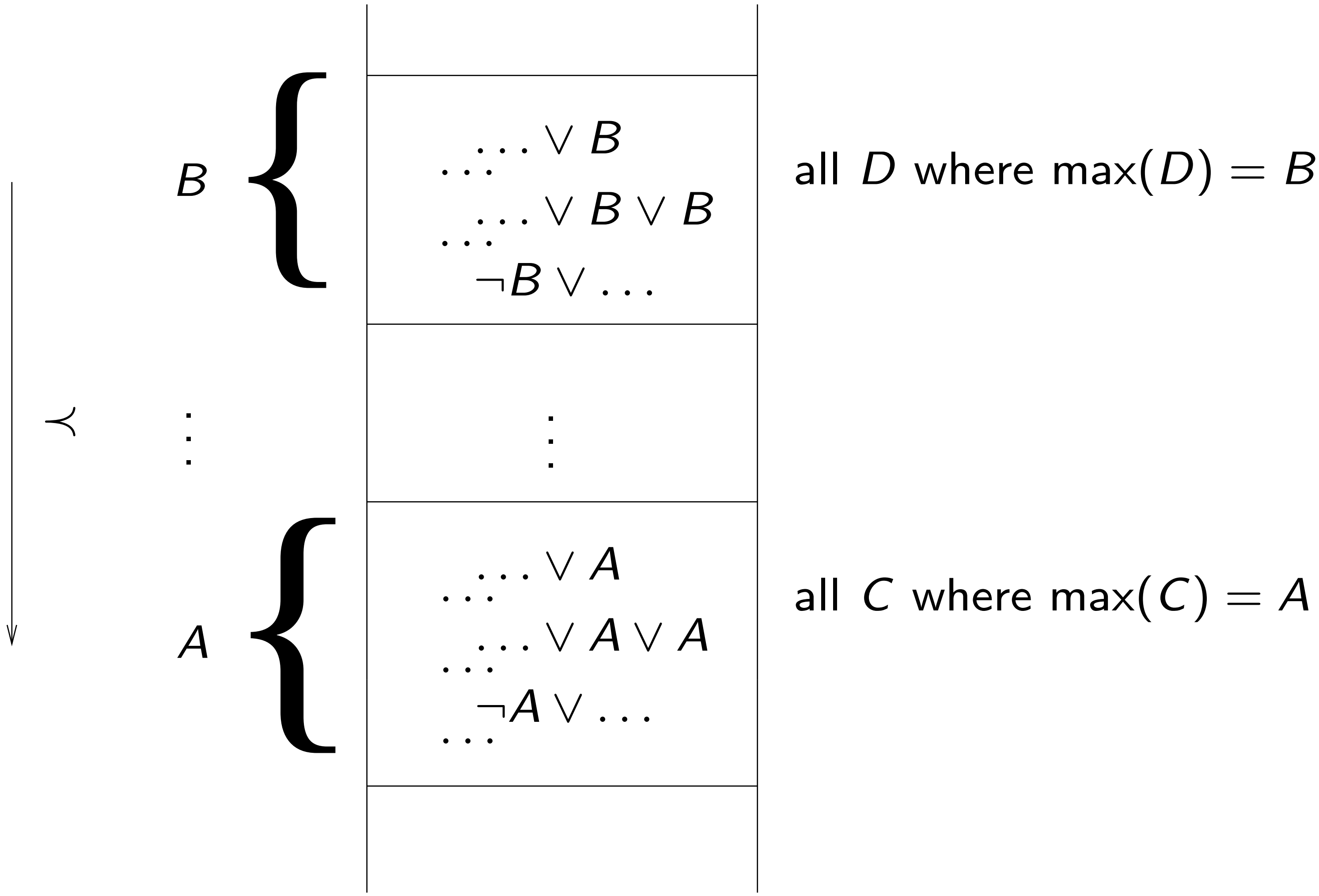
# Multi-Set Orderings

Let $(M, \succ)$ be a partial ordering. The multi-set extension of $\succ$ to multi-sets over $M$ is defined by

$$
\begin{aligned}
S_1 \succ_{\mathsf{mul}} S_2 :\Leftrightarrow\ & S_1 \neq S_2 \\
& \text{and } \forall m \in M : [S_2(m) > S_1(m) \\
& \quad \Rightarrow \quad \exists m' \in M : (m' \succ m \text{ and } S_1(m') > S_2(m'))]
\end{aligned}
$$

# Stratified Structure of Clause Sets

Let $A \succ B$. Clause sets are then stratified in this form:



# Construction of Interpretations

Given: set $N$ of clauses, atom ordering $\succ$.

Wanted: Valuation $\mathcal{A}$ such that

- "many" clauses from $N$ are valid in $\mathcal{A}$;
- $\mathcal{A} \models N$, *if* $N$ *is saturated and* $\bot \notin N$.

Construction according to $\succ$, starting with the minimal clause.

# Main Ideas of the Construction

- Clauses are considered in the order given by $\prec$. We construct a model for $N$ incrementally.

- When considering $C$, one already has a partial interpretation $I_C$ (initially $I_C = \emptyset$) available.

  In what follows, instead of referring to partial valuations $\mathcal{A}_C$ we will refer to partial interpretations $I_C$ (the set of atoms which are true in the valuation $\mathcal{A}_C$).

- If $C$ is true in the partial interpretation $I_C$, nothing is done. ($\Delta_C = \emptyset$).

- If $C$ is false, one would like to change $I_C$ such that $C$ becomes true.

# Example

Let $P_5 \succ P_4 \succ P_3 \succ P_2 \succ P_1 \succ P_0$ (max. literals in red)

Construction of $I$:

| | clauses $C$ | $I_C$ | $\Delta_C$ | Remarks |
|---|---|---|---|---|
| 1 | $\neg P_0$ | $\emptyset$ | $\emptyset$ | true in $\mathcal{A}_C$ |
| 2 | $P_0 \lor P_1$ | $\emptyset$ | $\{P_1\}$ | |
| 3 | $P_1 \lor P_2$ | $\{P_1\}$ | $\emptyset$ | true in $\mathcal{A}_C$ |
| 4 | $\neg P_1 \lor P_2$ | $\{P_1\}$ | $\{P_2\}$ | |
| 5 | $\neg P_1 \lor \neg P_1 \lor P_3 \lor P_0$ | $\{P_1, P_2\}$ | $\{P_3\}$ | |
| 6 | $\neg P_1 \lor \neg P_1 \lor P_3 \lor P_3 \lor P_0$ | $\{P_1, P_2, P_3\}$ | $\emptyset$ | true in $\mathcal{A}_C$ |
| 7 | $\neg P_1 \lor P_4 \lor P_3 \lor P_0$ | $\{P_1, P_2, P_3\}$ | $\emptyset$ | true in $\mathcal{A}_C$ |
| 8 | $\neg P_1 \lor \neg P_4 \lor P_3$ | $\{P_1, P_2, P_3\}$ | $\emptyset$ | true in $\mathcal{A}_C$ |
| 9 | $\neg P_3 \lor P_5$ | $\{P_1, P_2, P_3\}$ | $\{P_5\}$ | |

The resulting $I = \{P_1, P_2, P_3, P_5\}$ is a model of the clause set.

# Construction of Candidate Models Formally

Let $N, \succ$ be given. We define sets $I_C$ and $\Delta_C$ for all ground clauses $C$ over the given signature inductively over $\succ$:

$$I_C := \bigcup_{C \succ D} \Delta_D$$

$$\Delta_C := \begin{cases} \{A\}, & \text{if } C \in N,\ C = C' \vee A,\ A \succ C',\ I_C \not\models C \\ \emptyset, & \text{otherwise} \end{cases}$$

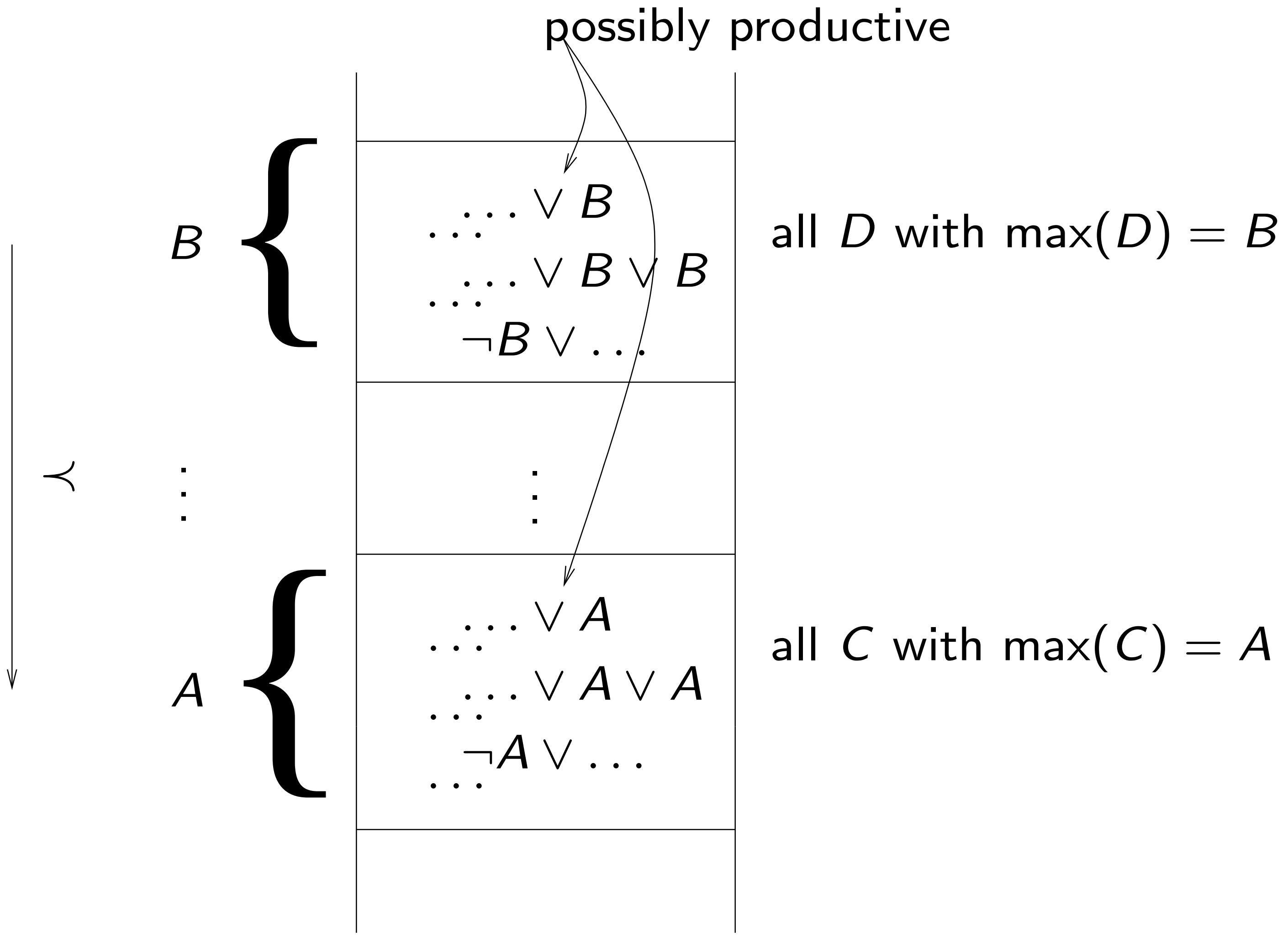
We say that $C$ produces $A$, if $\Delta_C = \{A\}$.

The candidate model for $N$ (wrt. $\succ$) is given as $I_N^{\succ} := \bigcup_C \Delta_C$.

We also simply write $I_N$, or $I$, for $I_N^{\succ}$ if $\succ$ is either irrelevant or known from the context.

# Structure of $N, \succ$

Let $A \succ B$; producing a new atom does not affect smaller clauses.



# Model Existence Theorem

**Theorem 1.14** (Bachmair & Ganzinger):
Let $\succ$ be a clause ordering, let $N$ be saturated wrt. $Res$, and suppose that $\bot \not\in N$. Then $I_N^{\succ} \models N$.

**Corollary 1.15:**
Let $N$ be saturated wrt. $Res$. Then $N \models \bot \Leftrightarrow \bot \in N$.

# Model Existence Theorem

Proof:

Suppose $\bot \not\in N$, but $I_N^{\succ} \not\models N$. Let $C \in N$ minimal (in $\succ$) such that $I_N^{\succ} \not\models C$. Since $C$ is false in $I_N$, $C$ is not productive. As $C \neq \bot$ there exists a maximal atom $A$ in $C$.

Case 1: $C = \neg A \vee C'$ (i.e., the maximal atom occurs negatively)
$\Rightarrow I_N \models A$ and $I_N \not\models C'$
$\Rightarrow$ some $D = D' \vee A \in N$ produces A. As $\frac{D' \vee A \qquad \neg A \vee C'}{D' \vee C'}$, we infer that $D' \vee C' \in N$, and $C \succ D' \vee C'$ and $I_N \not\models D' \vee C'$
$\Rightarrow$ contradicts minimality of $C$.

Case 2: $C = C' \vee A \vee A$. Then $\frac{C' \vee A \vee A}{C' \vee A}$ yields a smaller counterexample $C' \vee A \in N$. $\Rightarrow$ contradicts minimality of $C$.

# Ordered Resolution with Selection

Ideas for improvement:

1. In the completeness proof (Model Existence Theorem) one only needs to resolve and factor maximal atoms
   $\Rightarrow$ if the calculus is restricted to inferences involving maximal atoms, the proof remains correct
   $\Rightarrow$ *order restrictions*

2. In the proof, it does not really matter with which negative literal an inference is performed
   $\Rightarrow$ choose a negative literal don't-care-nondeterministically
   $\Rightarrow$ *selection*

# Selection Functions

A selection function is a mapping

$$S : C \quad \mapsto \quad \text{set of occurrences of } \textit{negative} \text{ literals in } C$$

Example of selection with selected literals indicated as $\boxed{X}$:

$$\boxed{\neg A} \vee \neg A \vee B$$

$$\boxed{\neg B_0} \vee \boxed{\neg B_1} \vee A$$

# Ordered resolution

In the completeness proof, we talk about (strictly) maximal literals of clauses.

# Resolution Calculus $Res_S^{\succ}$

**Ordered Resolution with Selection:**

$$\frac{C \lor A \qquad D \lor \neg A}{C \lor D}$$

if (i) $A \succ C$;

(ii) nothing is selected in $C$ by S;

(iii) $\neg A$ is selected in $D \lor \neg A$,

or else nothing is selected in $D \lor \neg A$ and $\neg A \succeq \max(D)$.

**Ordered Factoring:**

$$\frac{C \lor A \lor A}{(C \lor A)}$$

if $A$ is maximal in $C$ and nothing is selected in $C$.

Note: For positive literals, $A \succ C$ is the same as $A \succ \max(C)$.

# Search Spaces Become Smaller

| | | |
|---|---|---|
| 1 | $A \vee B$ | |
| 2 | $A \vee \boxed{\neg B}$ | |
| 3 | $\neg A \vee B$ | |
| 4 | $\neg A \vee \boxed{\neg B}$ | |
| 5 | $B \vee B$ | Res 1, 3 |
| 6 | $B$ | Fact 5 |
| 7 | $\neg A$ | Res 6, 4 |
| 8 | $A$ | Res 6, 2 |
| 9 | $\bot$ | Res 8, 7 |

we assume $A \succ B$ and $S$ as indicated by $\boxed{X}$. The maximal literal in a clause is depicted in red.

With this ordering and selection function the refutation proceeds strictly deterministically in this example. Generally, proof search will still be non-deterministic but the search space will be much smaller than with unrestricted resolution.

# $\mathrm{Res}_S^{\succ}$: Construction of Candidate Models

Let $N, \succ$ be given. We define sets $I_C$ and $\Delta_C$ for all ground clauses $C$ over the given signature inductively over $\succ$:

$$I_C \quad := \quad \bigcup_{C \succ D} \Delta_D$$

$$\Delta_C \quad := \quad \begin{cases} \{A\}, & \text{if } C \in N,\ C = C' \vee A,\ A \succ C',\ I_C \not\models C \\ & \text{and nothing is selected in } C \\ \emptyset, & \text{otherwise} \end{cases}$$

We say that $C$ produces $A$, if $\Delta_C = \{A\}$.

The candidate model for $N$ (wrt. $\succ$) is given as $I_N^{\succ} := \bigcup_C \Delta_C$.

We also simply write $I_N$, or $I$, for $I_N^{\succ}$ if $\succ$ is either irrelevant or known from the context.

# Model Existence Theorem

**Theorem 1.14$^s$** (Bachmair & Ganzinger):
Let $\succ$ be a clause ordering, let $N$ be saturated wrt. $Res_S^{\succ}$, and suppose that $\bot \notin N$. Then $I_N^{\succ} \models N$.

**Corollary 1.15$^s$:**
Let $N$ be saturated wrt. $Res_S^{\succ}$. Then $N \models \bot \Leftrightarrow \bot \in N$.

# Model Existence Theorem

Proof:

Suppose $\bot \notin N$, but $I_N^{\succ} \not\models N$. Let $C \in N$ minimal (in $\succ$) such that $I_N^{\succ} \not\models C$. Since $C$ is false in $I_N$, $C$ is not productive. As $C \neq \bot$ there exists a maximal atom $A$ in $C$.

Case 1: $C = \neg A \vee C'$

(i.e., the maximal atom occurs negatively or $\neg A$ is selected in $C$)

$\Rightarrow I_N \models A$ and $I_N \not\models C'$

$\Rightarrow$ some $D = D' \vee A \in N$ produces A. As $\frac{D' \vee A \qquad \neg A \vee C'}{D' \vee C'}$, we infer that $D' \vee C' \in N$, and $C \succ D' \vee C'$ and $I_N \not\models D' \vee C'$

$\Rightarrow$ contradicts minimality of $C$.

Case 2: $C = C' \vee A \vee A$. Then $\frac{C' \vee A \vee A}{C' \vee A}$ yields a smaller counterexample $C' \vee A \in N$. $\Rightarrow$ contradicts minimality of $C$.

# Decision Procedures for Satisfiability

- Simple Decision Procedures

  truth table method — Logik f. Informatiker

  Discrete Algebraic Structures

- The Resolution Procedure

- The Davis-Putnam-Logemann-Loveland Algorithm — now

## 1.7 The DPLL Procedure

Goal:

Given a propositional formula in CNF (or alternatively, a finite set $N$ of clauses), check whether it is satisfiable (and optionally: output *one* solution, if it is satisfiable).

# Satisfiability of Clause Sets

$\mathcal{A} \models N$ if and only if $\mathcal{A} \models C$ for all clauses $C$ in $N$.

$\mathcal{A} \models C$ if and only if $\mathcal{A} \models L$ for some literal $L \in C$.

# Partial Valuations

Since we will construct satisfying valuations incrementally,
we consider partial valuations
(that is, partial mappings $\mathcal{A} : \Pi \to \{0, 1\}$).

We start with an empty valuation and try to extend it
step by step to all variables occurring in $N$.

If $\mathcal{A}$ is a partial valuation, then literals and clauses can be
true, false, or undefined under $\mathcal{A}$.

A clause is true under $\mathcal{A}$ if one of its literals is true;
it is false (or "conflicting") if all its literals are false;
otherwise it is undefined (or "unresolved").

# Unit Clauses

Observation:

Let $\mathcal{A}$ be a partial valuation. If the set $N$ contains a clause $C$, such that all literals but one in $C$ are false under $\mathcal{A}$, then the following properties are equivalent:

- there is a valuation that is a model of $N$ and extends $\mathcal{A}$.
- there is a valuation that is a model of $N$ and extends $\mathcal{A}$ and makes the remaining literal $L$ of $C$ true.

$C$ is called a unit clause; $L$ is called a unit literal.

# Pure Literals

One more observation:

Let $\mathcal{A}$ be a partial valuation and $P$ a variable that is undefined under $\mathcal{A}$. If $P$ occurs only positively (or only negatively) in the unresolved clauses in $N$, then the following properties are equivalent:

- there is a valuation that is a model of $N$ and extends $\mathcal{A}$.
- there is a valuation that is a model of $N$ and extends $\mathcal{A}$ and assigns true (false) to $P$.

$P$ is called a pure literal.

# The Davis-Putnam-Logemann-Loveland Proc.

```
boolean DPLL(clause set N, partial valuation 𝒜) {
    if (all clauses in N are true under 𝒜) return true;
    elsif (some clause in N is false under 𝒜) return false;
    elsif (N contains unit clause P) return DPLL(N, 𝒜 ∪ {P ↦ 1});
    elsif (N contains unit clause ¬P) return DPLL(N, 𝒜 ∪ {P ↦ 0});
    elsif (N contains pure literal P) return DPLL(N, 𝒜 ∪ {P ↦ 1});
    elsif (N contains pure literal ¬P) return DPLL(N, 𝒜 ∪ {P ↦ 0});
    else {
        let P be some undefined variable in N;
        if (DPLL(N, 𝒜 ∪ {P ↦ 0})) return true;
        else return DPLL(N, 𝒜 ∪ {P ↦ 1});
    }
}
```

# The Davis-Putnam-Logemann-Loveland Proc.

Initially, DPLL is called with the clause set $N$ and with an empty partial valuation $\mathcal{A}$.

# The Davis-Putnam-Logemann-Loveland Proc.

In practice, there are several changes to the procedure:

The pure literal check is often omitted (it is too expensive).

The branching variable is not chosen randomly.

The algorithm is implemented iteratively;
the backtrack stack is managed explicitly
(it may be possible and useful to backtrack more than one level).

# DPLL Iteratively

An iterative (and generalized) version:

```
status = preprocess();
if (status != UNKNOWN) return status;
while(1) {
    decide_next_branch();
    while(1) {
        status = deduce();
        if (status == CONFLICT) {
            blevel = analyze_conflict();
            if (blevel == 0) return UNSATISFIABLE;
            else backtrack(blevel); }
        else if (status == SATISFIABLE) return SATISFIABLE;
        else break;
    }
}
```

# DPLL Iteratively

`preprocess()`

preprocess the input (as far as it is possible without branching);
return CONFLICT or SATISFIABLE or UNKNOWN.

`decide_next_branch()`

choose the right undefined variable to branch;
decide whether to set it to 0 or 1;
increase the backtrack level.

# DPLL Iteratively

`deduce()`

make further assignments to variables (e.g., using the unit clause rule) until a satisfying assignment is found, or until a conflict is found, or until branching becomes necessary; return CONFLICT or SATISFIABLE or UNKNOWN.

# DPLL Iteratively

`analyze_conflict()`

check where to backtrack.

`backtrack(blevel)`

backtrack to `blevel`;

flip the branching variable on that level;

undo the variable assignments in between.

# Branching Heuristics

Choosing the right undefined variable to branch is important for efficiency, but the branching heuristics may be expensive itself.

State of the art: use branching heuristics that need not be recomputed too frequently.

In general: choose variables that occur frequently.

# The Deduction Algorithm

For applying the unit rule, we need to know the number of literals in a clause that are not false.

Maintaining this number is expensive, however.

# The Deduction Algorithm

Better approach: "Two watched literals":

In each clause, select two (currently undefined) "watched" literals.

For each variable $P$, keep a list of all clauses in which $P$ is watched and a list of all clauses in which $\neg P$ is watched.

If an undefined variable is set to 0 (or to 1), check all clauses in which $P$ (or $\neg P$) is watched and watch another literal (that is true or undefined) in this clause if possible.

Watched literal information need not be restored upon backtracking.

# Conflict Analysis and Learning

Goal: Reuse information that is obtained in one branch in further branches.

Method: Learning:

> If a conflicting clause is found, use the resolution rule to derive a new clause and add it to the current set of clauses.

Problem: This may produce a large number of new clauses; therefore it may become necessary to delete some of them afterwards to save space.

# Backjumping

Related technique:

non-chronological backtracking ("backjumping"):

If a conflict is independent of some earlier branch, try to skip that over that backtrack level.

## Restart

Runtimes of DPLL-style procedures depend extremely on the choice of branching variables.

If no solution is found within a certain time limit, it can be useful to restart from scratch with another choice of branchings (but learned clauses may be kept).

# A succinct formulation

State: $M||F$,

where:

- $M$ partial assignment (sequence of literals),

  some literals are annotated ($L^d$: decision literal)

- $F$ clause set.

# A succinct formulation

UnitPropagation

$M || F, C \vee L \Rightarrow M, L || F, C \vee L$ if $M \models \neg C$, and $L$ undef. in $M$

Decide

$M || F \Rightarrow M, L^d || F$ if $L$ or $\neg L$ occurs in $F$, $L$ undef. in $M$

Fail

$M || F, C \Rightarrow \text{Fail}$ if $M \models \neg C$, $M$ contains no decision literals

Backjump

$$M, L^d, N || F \Rightarrow M, L' || F \quad \text{if} \begin{cases} \text{there is some clause } C \vee L' \text{ s.t.:} \\ F \models C \vee L', M \models \neg C, \\ L' \text{ undefined in } M \\ L' \text{ or } \neg L' \text{ occurs in } F. \end{cases}$$

# Example

| Assignment: | Clause set: | |
|---|---|---|
| $\emptyset$ | $\| \|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$ | $\Rightarrow$ (Decide) |
| $P_1$ | $\| \|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$ | $\Rightarrow$ (UnitProp) |
| $P_1 P_2$ | $\| \|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$ | $\Rightarrow$ (Decide) |
| $P_1 P_2 P_3$ | $\| \|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$ | $\Rightarrow$ (UnitProp) |
| $P_1 P_2 P_3 P_4$ | $\| \|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$ | $\Rightarrow$ (Decide) |
| $P_1 P_2 P_3 P_4 P_5$ | $\| \|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$ | $\Rightarrow$ (UnitProp) |
| $P_1 P_2 P_3 P_4 P_5 \neg P_6$ | $\| \|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$ | $\Rightarrow$ (Backtrack) |
| $P_1 P_2 P_3 P_4 \neg P_5$ | $\| \|\neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$ | ... |

# DPLL with learning

The DPLL system with learning consists of the four transition rules of the Basic DPLL system, plus the following two additional rules:

Learn

$M||F \Rightarrow M||F, C$ if all atoms of $C$ occur in $F$ and $F \models C$

Forget

$M||F, C \Rightarrow M||F$ if $F \models C$

In these two rules, the clause C is said to be learned and forgotten, respectively.

# Further Information

The ideas described so far heve been implemented in the SAT checker Chaff.

Further information:
Lintao Zhang and Sharad Malik:
The Quest for Efficient Boolean Satisfiability Solvers,
Proc. CADE-18, LNAI 2392, pp. 295–312, Springer, 2002.