

# Decision Procedures for Verification

Part 1. Propositional Logic (3)

5.11.2018

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

- 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  $\mathcal{A} : \Pi \rightarrow \{0, 1\}$ ).

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,  $\mathcal{A} \cup \{P \mapsto 1\}$ );
  elsif (N contains unit clause  $\neg P$ ) return DPLL(N,  $\mathcal{A} \cup \{P \mapsto 0\}$ );
  elsif (N contains pure literal P) return DPLL(N,  $\mathcal{A} \cup \{P \mapsto 1\}$ );
  elsif (N contains pure literal  $\neg P$ ) return DPLL(N,  $\mathcal{A} \cup \{P \mapsto 0\}$ );
  else {
    let P be some undefined variable in N;
    if (DPLL(N,  $\mathcal{A} \cup \{P \mapsto 0\}$ )) return true;
    else return DPLL(N,  $\mathcal{A} \cup \{P \mapsto 1\}$ );
  }
}
```

# 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

$$M \parallel F, C \vee L \Rightarrow M, L \parallel F, C \vee L$$

if  $M \models \neg C$ , and  $L$  undef. in  $M$

## Decide

$$M \parallel F \Rightarrow M, L^d \parallel F$$

if  $L$  or  $\neg L$  occurs in  $F$ ,  $L$  undef. in  $M$

## Fail

$$M \parallel F, C \Rightarrow \text{Fail}$$

if  $M \models \neg C$ ,  $M$  contains no decision literals

## Backjump

$$M, L^d, N \parallel F \Rightarrow M, L' \parallel F$$

if  $\left\{ \begin{array}{l} \text{there is some clause } C \vee L' \text{ s.t.:} \\ F \models C \vee L', M \models \neg C, \\ L' \text{ undefined in } M \\ L' \text{ or } \neg L' \text{ occurs in } F. \end{array} \right.$



# Backjump

---

## Backjump

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

We will see later that the Backjump rule is always applicable, if the list of literals  $ML^dN$  contains at least one decision literal and some clause in  $N$  is false under  $M$ . There are many possible backjump clauses. One candidate:  $\neg L_1 \vee \dots \vee \neg L_n$ , where the  $L_i$  are all the decision literals in  $ML^dN$ . (But usually there are better choices.)

# 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)

# Example

---

Assignment:	Clause set:
$\emptyset$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2 \Rightarrow (\text{Decide})$
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2 \Rightarrow (\text{UnitProp})$

# 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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)

# 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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)

# 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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)

# 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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d P_4 P_5^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)

# 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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d P_4 P_5^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4 P_5^d \neg P_6$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Backtrack)



# 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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d P_4 P_5^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4 P_5^d \neg P_6$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Backtrack)
$P_1^d P_2 P_3^d P_4 \neg P_5$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	...

# 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.

# 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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d P_4 P_5^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4 P_5^d \neg P_6$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Backtrack)

# Backjumping

---

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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d P_4 P_5^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4 P_5^d \neg P_6$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	

Conflict clause:  $P_6 \vee \neg P_5 \vee \neg P_2$

# Backjumping

---

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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d P_4 P_5^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4 P_5^d \neg P_6$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	

Conflict clause:  $P_6 \vee \neg P_5 \vee \neg P_2$

Decisions  $P_1^d$ ,  $P_3^d$ , and  $P_5^d$  are the cause; cannot be taken at the same time.

Possible backjump clause:  $\neg P_1 \vee \neg P_3 \vee \neg P_5$ . Can we do better?

# Backjumping

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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d P_4 P_5^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4 P_5^d \neg P_6$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	

Conflict clause:  $P_6 \vee \neg P_5 \vee \neg P_2$

Decisions  $P_1^d$  and  $P_5^d$  are the cause, but not  $P_3^d$

Decisions  $P_1^d$  and  $P_5^d$  cannot be taken at the same time  $\mapsto$  derive new clause  $\neg P_1 \vee \neg P_5$ ; use for backjump.

# Backjumping

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)
$P_1^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 P_3^d P_4 P_5^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (UnitProp)
$P_1^d P_2 P_3^d P_4 P_5^d \neg P_6$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Backjump)
$P_1^d P_2 \neg P_5$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	$\Rightarrow$ (Decide)
$P_1^d P_2 \neg P_5 P_4^d$	$\ \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2$	

Conflict clause:  $P_6 \vee \neg P_5 \vee \neg P_2$

Decisions  $P_1^d$  and  $P_5^d$  are the cause, but not  $P_3^d$

Decisions  $P_1^d$  and  $P_5^d$  cannot be taken at the same time  $\mapsto$  derive new clause  $\neg P_1 \vee \neg P_5$ ; use for backjump.

# Backjump

---

## Backjump

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

$C$  conflict clause;  $C \vee L'$  backjump clause



# Backjump

---

## Backjump

$$M, L^d, N \parallel F, C \Rightarrow M, L' \parallel F, C \quad \text{if} \quad \left\{ \begin{array}{l} ML^d N \models \neg C \\ \text{There is some clause } C' \vee L' \text{ s.t.:} \\ F, C \models C' \vee L', M \models \neg C', \\ L' \text{ undefined in } M \\ L' \text{ or } \neg L' \text{ occurs in } F. \end{array} \right.$$

$C$  conflict clause;  $C' \vee L'$  backjump clause

$$\begin{array}{l} P_1^d P_2 P_3^d P_4 P_5^d \neg P_6 \quad \parallel \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2 \quad \Rightarrow (\text{Backjump}) \\ P_1^d P_2 \neg P_5 \quad \parallel \neg P_1 \vee P_2, \neg P_3 \vee P_4, \neg P_5 \vee \neg P_6, P_6 \vee \neg P_5 \vee \neg P_2 \end{array}$$

Backjump clause:  $C' \vee L'$

with  $C' = \neg P_1$  and  $L' = \neg P_5$  (Lemma, learned clause)

Determined e.g. using a “conflict graph”.

# Backjump

---

Suppose that we have reached a state  $M||F$  such that some clause  $C \in F$  (or following from  $F$ ) is false under  $M$ .

Consequently, every literal of  $C$  is the complement of some literal in  $M$ .

- (1) If every literal in  $C$  is the complement of a decision literal of  $M$ , then  $C$  is a backjump clause.
- (2) Otherwise,  $C = C' \vee \neg L$ , such that  $L$  is a deduced literal.

For every deduced literal  $L$ , there is a (unit or backjump) clause  $D \vee L$ , such that  $F \models D \vee L$  and  $D$  is false under  $M$ .

Then  $F \models D \vee C'$  and  $D \vee C'$  is also false under  $M$ .

( $D \vee C'$  is a resolvent of  $C' \vee \neg L$  and  $D \vee L$ .)

# Backjump

---

By repeating this process, we will eventually obtain a clause that satisfies the requirements of a backjump clause (or the empty clause). Usually, one resolves the literals in the reverse order in which they were added to  $M$  and stops as soon as one obtains a clause in which all but one literal are complements of literals occurring in  $M$  before the last decision literal.

↳ 1UIP (first unique implication point) strategy.

## Further Information

---

The ideas described so far have 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.

# Applications of propositional logic

---

- A toy example (sudoku)
- Scheduling
- Verification

# Sudoku

---

1							1	
2	4							
3		2						
4				5		4		7
5			8			3		
6			1		9			
7	3			4			2	
8		5		1				
9				8		6		

Idea:  $p_{i,j}^d = \text{true}$  iff the value of square  $i, j$  is  $d$

For example:  $p_{3,5}^8 = \text{true}$

# Sudoku

---

1							1	
2	4							
3		2						
4				5	4		7	
5			8			3		
6			1	9				
7	3			4		2		
8		5		1				
9				8	6			

Coding SUDOKU by propositional clauses:

- Concrete values result in units:  $p_{i,j}^d$ .
- For every value, column we generate:  $\neg p_{i,j}^d \vee \neg p_{i,k}^d$  (if  $j \neq k$ ).  
Accordingly for all rows and  $3 \times 3$  boxes.
- For every square we generate:  $p_{i,j}^1 \vee \dots \vee p_{i,j}^9$ .  
For every two different values  $d, d'$ , and every square we generate:  $\neg p_{i,j}^d \vee \neg p_{i,j}^{d'}$ .
- For every value  $d$  and every column we generate:  
 $p_{i,1}^d \vee \dots \vee p_{i,9}^d$ .  
Accordingly for all rows and  $3 \times 3$  boxes.

# Sudoku

---

1							1	
2	4							
3		2						
4				5		4		7
5			8			3		
6			1		9			
7	3			4			2	
8		5		1				
9				8		6		

Set of clauses satisfiable  $\Leftrightarrow$  Sudoku has a solution

Let  $\mathcal{A}$  be a satisfying assignment

$\mathcal{A}(p_{i,j}^k) = 1$  iff a  $k$  appears in line  $i$ , column  $j$ .



# Scheduling

---

**Example:** A simple scheduling problem

In a school there are three teachers with the following specialization combinations:

Müller      Mathematics

Schmidt     German

Körner      Mathematics, German

	Group a	Group b
8:00– 8:50	Mathematics	German
9:00– 9:50	German	German
10:00–10:50	Math	Mathematics

Each teacher must teach at least two classes.

# Scheduling

---

Müller      Mathematics  
Schmidt     German  
Körner      Mathematics, German

	Group a	Group b
1) 8:00– 8:50	Mathematics	German
2) 9:00– 9:50	German	German
3) 10:00–10:50	Math	Mathematics

## Modeling:

**Propositional variables:**  $P_{s,k,N,f}$  'Teacher  $N$  teaches subject  $f$  in group  $k$  in time slot  $s$ '

# Scheduling

---

Müller      Mathematics  
 Schmidt     German  
 Körner      Mathematics, German

	Group a	Group b
1) 8:00– 8:50	Mathematics	German
2) 9:00– 9:50	German	German
3) 10:00–10:50	Math	Mathematics

## Modeling:

**Propositional variables:**  $P_{s,k,N,f}$  'Teacher  $N$  teaches subject  $f$  in group  $k$  in time slot  $s$ '

Rules:  $(P_{1,a,M,m} \vee P_{1,a,K,m}) \wedge (P_{1,b,S,d} \vee P_{1,b,K,d})$

$(P_{2,a,S,d} \vee P_{2,a,K,d}) \wedge (P_{2,b,S,d} \vee P_{2,b,K,d})$

$(P_{3,a,M,m} \vee P_{3,a,K,m}) \wedge (P_{3,b,S,d} \vee P_{3,a,K,d})$

$\neg(P_{1,a,K,m} \wedge P_{1,b,K,d}) \wedge \neg(P_{2,a,K,d} \wedge P_{2,b,K,d}) \wedge \neg(P_{2,a,S,d} \wedge P_{2,b,S,d}) \wedge$

$\neg(P_{3,a,K,m} \wedge P_{3,b,K,m}) \wedge (P_{1,a,M,m} \wedge P_{1,b,M,m}) \dots$

# Program Verification

---

- Bounded model checking
- Model checking
- Invariant checking/generation
- Abstraction

# Finite-state systems

---

- $X$  finite set of variables,  $V$  finite set of possible values for the variables  $p_{xv}^i$  (in the  $i$ -th step  $x$  takes value  $v$ )
- Other propositional variables  $q_k, k \in K$
- Transitions (variables change their value)

$$Tr(i, i + 1) := \bigvee \left( \text{Cond}(p_{x_1 v_1}^i, \dots, p_{x_n v_n}^i) \wedge \bigwedge_{j=1}^n p_{x_j v_j}^{i+1} \wedge \bigwedge_k q_k^{i+1} \right)$$

(where  $v_j^{i+1}, q_k^{i+1}$  suitably computed)

$F(p_{x_1, v_1}^k, \dots, p_{x_n, v_n}^k, \dots)$  property of assignments

**Bounded model checking:**

$$\bigwedge_{j=1}^n p_{x_j, v_j}^1 \wedge \bigwedge q_k^1 \wedge Tr(1, 2) \wedge \dots \wedge Tr(k - 1, k) \wedge \neg F(p_{x_1, v_1}^k, \dots, p_{x_n, v_n}^k, \dots)$$

# Example

---

**Question:** Does BUBBLESORT return a sorted array?

```
int [] BUBBLESORT(int[] a) {
    int i, j, t;
    for (i := |a| - 1; i > 0; i := i - 1) {
        for (j := 0; j < i; j := j + 1) {
            if (a[j] > a[j + 1]){t := a[j];
                a[j] := a[j + 1];
                a[j + 1] := t};
        }
    } return a}
```

# Example

---

**Question:** Does BUBBLESORT return a sorted array?

```
int [] BUBBLESORT(int[] a) {
  int i, j, t;
  for (i := |a| - 1; i > 0; i := i - 1) {
    for (j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]){t := a[j];
                               a[j] := a[j + 1];
                               a[j + 1] := t};
    }
  } return a}
```

**Simpler question:**

$|a| = 3$ ;  $a[0]=7$ ,  $a[1]=9$ ,  $a[2]=4$   
does BubbleSort applied to this array  
return a sorted array?

Encoding in propositional logic:

- $p_{ij}^k$  (at step  $k$ ,  $a[i] = j$ )

**Examples:**  $p_{07}^1, p_{19}^1, p_{24}^1$

- $gt_{ij}^k$  (at step  $k$ ,  $a[i] > a[j]$ )

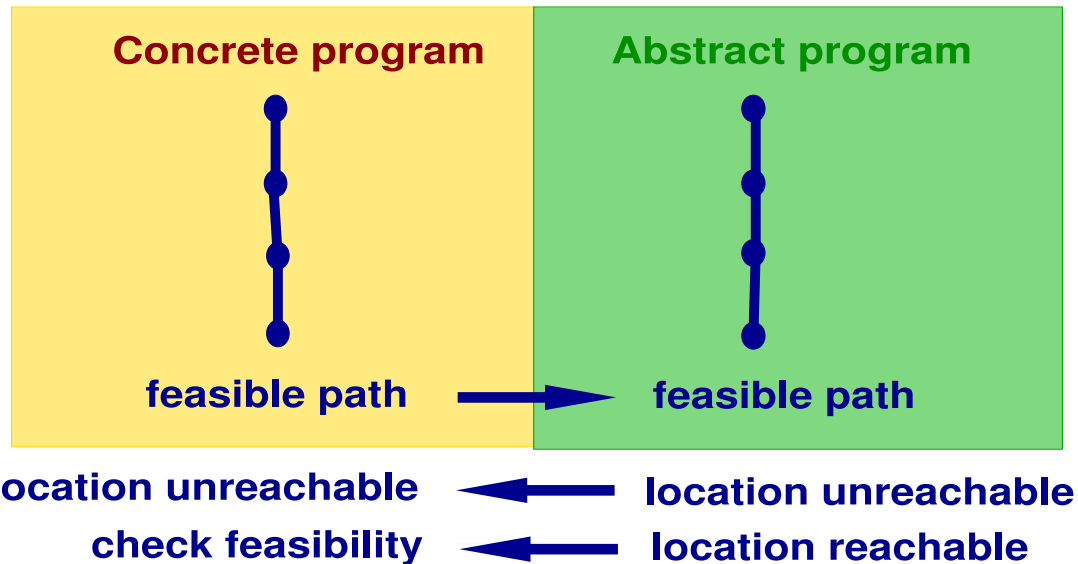
**Examples:**  $gt_{10}^1, \neg gt_{01}^1, gt_{02}^1, \neg gt_{20}^1, \dots$

Model updates with new propositional variables

(complicated; not very expressive)

# Abstraction-Based Verification

---



location unreachable ← location unreachable  
check feasibility ← location reachable



**conjunction of constraints:**  $\phi(1) \wedge Tr(1, 2) \wedge \dots \wedge Tr(n - 1, n) \wedge \neg \text{safe}(n)$

- satisfiable: feasible path
- unsatisfiable: refine abstract program s.t. the path is not feasible



# Tools for SAT checking

---

`http://www.satcompetition.org/`

## **Examples of SAT solvers:**

MiniSat: `http://minisat.se/`

MathSAT: `http://mathsat.fbk.eu/publications.html` (much more)

zChaff: `http://www.princeton.edu/~chaff/zchaff.html`

## **Example of use**

# Tools for SAT checking

---

## Resolution-based theorem provers:

E: <http://www4.informatik.tu-muenchen.de/schulz/E/E.html>

SPASS: <http://www.spass-prover.org/>

Vampire: <http://www.vprover.org/>

... full power for first-order logic (with equality)