Modern SAT Solvers

Albert Schimpf

6. Januar 2015

Propositional Satisfiability Problem (SAT)

SAT

SAT

- First NP-complete problem

SAT

- First NP-complete problem
- Recent development enabled industry use

SAT

- First NP-complete problem
- Recent development enabled industry use
- Random problems <-> Structured problems

SAT

- First NP-complete problem
- Recent development enabled industry use
- Random problems <-> Structured problems

SAT

Random SAT instance

SAT

Random SAT instance

- ”Harder”

SAT

Random SAT instance

- ”Harder”
- Small clause and variable size

SAT

Random SAT instance

- ”Harder”
- Small clause and variable size
- Solved by stochastic algorithms

SAT

Random SAT instance

- ”Harder”
- Small clause and variable size
- Solved by stochastic algorithms

Industry SAT instance

- ”Structured”

SAT

Random SAT instance

- "Harder"
- Small clause and variable size
- Solved by stochastic algorithms

Industry SAT instance

- "Structured"
- Very large clause and variable size (around 10^6)

SAT

Random SAT instance

- "Harder"
- Small clause and variable size
- Solved by stochastic algorithms

Industry SAT instance

- "Structured"
- Very large clause and variable size (around 10^6)
- Solved by conflict-driven algorithms

SAT

Random SAT instance

- ”Harder”
- Small clause and variable size
- Solved by stochastic algorithms

Industry SAT instance

- ”Structured”
- Very large clause and variable size (around 10^6)
- Solved by conflict-driven algorithms

Focus: conflict-driven SAT solvers

Specifically: depth-first search and backtrack based

Davis-Putnam-Logemann-Loveland Procedure

DPLL(formula, assignment){

Davis-Putnam-Logemann-Loveland Procedure

```
DPLL(formula, assignment){
   necessary = deduction(formula, assignment);
```

Davis-Putnam-Logemann-Loveland Procedure

```
DPLL(formula, assignment){
  necessary = deduction(formula, assignment);
  newAsgnmnt = union(necessary, assignment);
```

Davis-Putnam-Logemann-Loveland Procedure

```
DPLL(formula, assignment){
    necessary = deduction(formula, assignment);
    newAsgnmnt = union(necessary, assignment);
    if(isSatisfied(formula, newAsgnmnt)){
        return SAT;
    }
```

Davis-Putnam-Logemann-Loveland Procedure

```
DPLL(formula, assignment){
    necessary = deduction(formula, assignment);
    newAsgnmnt = union(necessary, assignment);
    if(isSatisfied(formula, newAsgnmnt)){
        return SAT;
    }
    if(isConflicting(formula, newAsgnmnt)){
        return CONFLICT;
    }
```

Davis-Putnam-Logemann-Loveland Procedure

```
DPLL(formula, assignment){
    necessary = deduction(formula, assignment);
    newAsgnmnt = union(necessary, assignment);
    if(isSatisfied(formula, newAsgnmnt)){
        return SAT;
    }
    if(isConflicting(formula, newAsgnmnt)){
        return CONFLICT;
    }
    var = chooseFreeVariable(formula, newAsgnmnt);
```

Davis-Putnam-Logemann-Loveland Procedure

```
DPLL(formula, assignment){
    necessary = deduction(formula, assignment);
    newAsgnmnt = union(necessary, assignment);
    if(isSatisfied(formula, newAsgnmnt)){
        return SAT;
    }
    if(isConflicting(formula, newAsgnmnt)){
        return CONFLICT;
    }
    var = chooseFreeVariable(formula, newAsgnmnt);
    asgn1 = union(newAsgnmnt, assign(var, 1));
```

Davis-Putnam-Logemann-Loveland Procedure

```
DPLL(formula, assignment){
    necessary = deduction(formula, assignment);
    newAsgnmnt = union(necessary, assignment);
    if(isSatisfied(formula, newAsgnmnt)){
        return SAT;
    }
    if(isConflicting(formula, newAsgnmnt)){
        return CONFLICT;
    }
    var = chooseFreeVariable(formula, newAsgnmnt);
    asgn1 = union(newAsgnmnt, assign(var, 1));
    if( DPLL(formula, newAsgnmnt) == SAT ){
        return SAT;
    }
```

Davis-Putnam-Logemann-Loveland Procedure

```
DPLL(formula, assignment){
    necessary = deduction(formula, assignment);
    newAsgnmnt = union(necessary, assignment);
    if(isSatisfied(formula, newAsgnmnt)){
        return SAT;
    }
    if(isConflicting(formula, newAsgnmnt)){
        return CONFLICT;
    }
    var = chooseFreeVariable(formula, newAsgnmnt);
    asgn1 = union(newAsgnmnt, assign(var, 1));
    if( DPLL(formula, newAsgnmnt) == SAT ){
        return SAT;
    } else {
        asgn2 = union(newAsgnmnt, assign(var, 0);
```

Davis-Putnam-Logemann-Loveland Procedure

```
DPLL(formula, assignment){
   necessary = deduction(formula, assignment);
   newAsgnmnt = union(necessary, assignment);
   if(isSatisfied(formula, newAsgnmnt)){
      return SAT;
   }
   if(isConflicting(formula, newAsgnmnt)){
      return CONFLICT;
   }
   var = chooseFreeVariable(formula, newAsgnmnt);
   asgn1 = union(newAsgnmnt, assign(var, 1));
   if( DPLL(formula, newAsgnmnt) == SAT ){
      return SAT;
   } else {
      asgn2 = union(newAsgnmnt, assign(var, 0);
      return = DPLL(formula, asgn2);
} }
```

Davis-Putnam-Logemann-Loveland Procedure

```
DPLL(formula, assignment){
    necessary = deduction(formula, assignment);
    newAsgnmnt = union(necessary, assignment);
    if(isSatisfied(formula, newAsgnmnt)){
        return SAT;
    }
    if(isConflicting(formula, newAsgnmnt)){
        return CONFLICT;
    }
    var = chooseFreeVariable(formula, newAsgnmnt);
    asgn1 = union(newAsgnmnt, assign(var, 1));
    if( DPLL(formula, newAsgnmnt) == SAT ){
        return SAT;
    } else {
        asgn2 = union(newAsgnmnt, assign(var, 0);
        return = DPLL(formula, asgn2);
} }
```

DPLL

```
DPLL(formula, assignment){
  necessary = deduction(formula, assignment);
  newAsgnmnt = union(necessary assignment);
  if(isSatisfied(formula, newAsgnmnt)){
    return SAT;
  }
  if(isConflicting(formula, newAsgnmnt)){
    return CONFLICT;
  }
  var = chooseFreeVariable(formula, newAsgnmnt);
  asgn1 = union(newAsgnmnt, assign(var, 1) );
  if( DPLL(formula, newAsgnmnt) == SAT ){
    return SAT;
  }else {
    asgn2 = union(newAsgnmnt, assign(var, 0) ;
    return = DPLL(formula, asgn2);
} }
```

Improvements:

DPLL

```
DPLL(formula, assignment){
  necessary = deduction(formula, assignment);
  newAsgnmnt = union(necessary assignment);
  if(isSatisfied(formula, newAsgnmnt)){
    return SAT;
  }
  if(isConflicting(formula, newAsgnmnt)){
    return CONFLICT;
  }
  var = chooseFreeVariable(formula, newAsgnmnt);
  asgn1 = union(newAsgnmnt, assign(var, 1) );
  if( DPLL(formula, newAsgnmnt) == SAT ){
    return SAT;
  }else {
    asgn2 = union(newAsgnmnt, assign(var, 0) ;
    return = DPLL(formula, asgn2);
} }
```

Improvements:

Describe high level structure

DPLL

```
DPLL(formula, assignment){
  necessary = deduction(formula, assignment);
  newAsgnmnt = union(necessary assignment);
  if(isSatisfied(formula, newAsgnmnt)){
    return SAT;
  }
  if(isConflicting(formula, newAsgnmnt)){
    return CONFLICT;
  }
  var = chooseFreeVariable(formula, newAsgnmnt);
  asgn1 = union(newAsgnmnt, assign(var, 1) );
  if( DPLL(formula, newAsgnmnt) == SAT ){
    return SAT;
  }else {
    asgn2 = union(newAsgnmnt, assign(var, 0) ;
    return = DPLL(formula, asgn2);
} }
```

Improvements:

Describe high level structure

Deduction phase

DPLL

```
DPLL(formula, assignment){
  necessary = deduction(formula, assignment);
  newAsgnmnt = union(necessary assignment);
  if(isSatisfied(formula, newAsgnmnt)){
    return SAT;
  }
  if(isConflicting(formula, newAsgnmnt)){
    return CONFLICT;
  }
  var = chooseFreeVariable(formula, newAsgnmnt);
  asgn1 = union(newAsgnmnt, assign(var, 1) );
  if( DPLL(formula, newAsgnmnt) == SAT ){
    return SAT;
  }else {
    asgn2 = union(newAsgnmnt, assign(var, 0) ;
    return = DPLL(formula, asgn2);
} }
```

Improvements:

Describe high level structure

Deduction phase

Variable selection

DPLL

```
DPLL(formula, assignment){
  necessary = deduction(formula, assignment);
  newAsgnmnt = union(necessary, assignment);
  if(isSatisfied(formula, newAsgnmnt)){
    return SAT;
  }
  if(isConflicting(formula, newAsgnmnt)){
    return CONFLICT;
  }
  var = chooseFreeVariable(formula, newAsgnmnt);
  asgn1 = union(newAsgnmnt, assign(var, 1) );
  if( DPLL(formula, newAsgnmnt) == SAT ){
    return SAT;
  }else {
    asgn2 = union(newAsgnmnt, assign(var, 0) );
    return = DPLL(formula, asgn2);
} }
```

Improvements:

Describe high level structure

Deduction phase

Variable selection

Data structure

DPLL

```
DPLL(formula, assignment){
   necessary = deduction(formula, assignment);
   newAsgnmnt = union(necessary, assignment);
   if(isSatisfied(formula, newAsgnmnt)){
      return SAT;
   }
   if(isConflicting(formula, newAsgnmnt)){
      return CONFLICT;
   }
   var = chooseFreeVariable(formula, newAsgnmnt);
   asgn1 = union(newAsgnmnt, assign(var, 1) );
   if( DPLL(formula, newAsgnmnt) == SAT ){
      return SAT;
   }else {
      asgn2 = union(newAsgnmnt, assign(var, 0) ;
      return = DPLL(formula, asgn2);
} }
```

Improvements:

Describe high level structure

Deduction phase

Variable selection

Data structure

Backtracking

1 Basic State Transition System

2 Non-Chronological Backtracking

3 Two-Watched-Literals Scheme

4 VSIDS Heuristic

5 Summary: The Modern Solver

Basic Definitions

Main idea: describe solving process precisely.

Basic Definitions

Main idea: describe solving process precisely.
We need to describe:

- Formula F

Basic Definitions

Main idea: describe solving process precisely.
We need to describe:

- Formula F
- Recursive stacks created by DPLL calls

Basic Definitions

Main idea: describe solving process precisely.
We need to describe:

- Formula F
- Recursive stacks created by DPLL calls
- Truth assignment trail M

Basic Definitions

Main idea: describe solving process precisely.
We need to describe:

- Formula F
- Recursive stacks created by DPLL calls
- Truth assignment trail M
- Conflict management set C

Basic Definitions

Main idea: describe solving process precisely.
We need to describe:

- Formula F
- Recursive stacks created by DPLL calls
- Truth assignment trail M
- Conflict management set C
- Transition Rules (e.g. unit, decision etc.)

Basic Definitions - Trail M

- Recursive calls: Introduce checkpoint symbol $\diamondsuit$

red: variable assignment already visited

Basic Definitions - Trail M

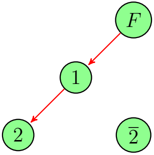
- Recursive calls: Introduce checkpoint symbol $\diamondsuit$

red: variable assignment already visited

Basic Definitions - Trail M

- Recursive calls: Introduce checkpoint symbol $\diamondsuit$

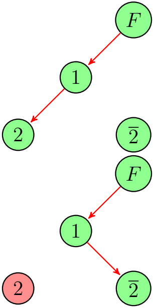
red: variable assignment already visited



Basic Definitions - Trail M

- Recursive calls: Introduce checkpoint symbol $\diamondsuit$

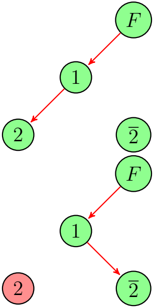
red: variable assignment already visited



Basic Definitions - Trail M

- Recursive calls: Introduce checkpoint symbol $\diamondsuit$

red: variable assignment already visited

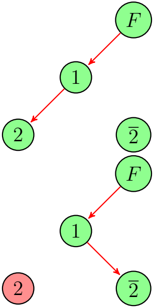


As a trail with checkpoints:

Basic Definitions - Trail M

- Recursive calls: Introduce checkpoint symbol $\diamondsuit$

red: variable assignment already visited



As a trail with checkpoints:

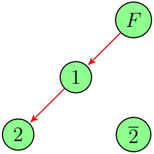
$$M = \{\diamondsuit 1 \diamondsuit 2\}$$

Basic Definitions - Trail M

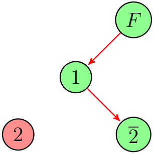
- Recursive calls: Introduce checkpoint symbol $\diamondsuit$

red: variable assignment already visited

As a trail with checkpoints:



$$M = \{\diamondsuit 1 \diamondsuit 2\}$$



$$M = \{\diamondsuit 1 \overline{2}\}$$

Basic Definitions - State

A DPLL $state$ is a triple $\langle F, M, C \rangle$.

Basic Definitions - State

A DPLL $state$ is a triple $\langle F, M, C \rangle$.

- F is the formula

Basic Definitions - State

A DPLL $state$ is a triple $\langle F, M, C \rangle$.

- F is the formula
- M is a check-pointed truth assignment trail

Basic Definitions - State

A DPLL $state$ is a triple $\langle F, M, C \rangle$.

- F is the formula
- M is a check-pointed truth assignment trail
- C is a set of conflict literals or the symbol *no_cflct*

Basic Definitions - State

A DPLL $state$ is a triple $\langle F, M, C \rangle$.

- F is the formula
- M is a check-pointed truth assignment trail
- C is a set of conflict literals or the symbol *no_cflct*

Checkpoints allow for iterative implementation.

Transition Rules: Decide

var = chooseFreeVariable(formula, newAsgnmnt);

Transition Rules: Decide

```
var = chooseFreeVariable(formula, newAsgnmnt);
  asgn1 = union(newAsgnmnt, assign(var, 1));
```

Transition Rules: Decide

var = chooseFreeVariable(formula, newAsgnmnt);
asgn1 = union(newAsgnmnt, assign(var, 1));

$$(Decide)\ \frac{l \in P}{}$$

Transition Rules: Decide

var = chooseFreeVariable(formula, newAsgnmnt);
asgn1 = union(newAsgnmnt, assign(var, 1));

$$(Decide)\ \frac{l \in P \text{ and } l, \bar{l} \notin M}{}$$

Transition Rules: Decide

```
var = chooseFreeVariable(formula, newAsgnmnt);
  asgn1 = union(newAsgnmnt, assign(var, 1));
```

$$(Decide)\ \frac{l \in P \text{ and } l, \bar{l} \notin M}{M :=}$$

Transition Rules: Decide

var = chooseFreeVariable(formula, newAsgnmnt);
asgn1 = union(newAsgnmnt, assign(var, 1));

$$(Decide)\ \frac{l \in P \text{ and } l, \bar{l} \notin M}{M := M + \diamondsuit + l}$$

Transition Rules: Unit

necessary = deduction(formula, assignment);

Transition Rules: Unit

necessary = deduction(formula, assignment);

$$(UnitPropag)$$

$$l \vee l_1 \vee ... \vee l_k \in F$$

Transition Rules: Unit

necessary = deduction(formula, assignment);

$$\frac{}{l \vee l_1 \vee ... \vee l_k \in F \text{ and } \overline{l_1}, ..., \overline{l_k} \in M} \; (UnitPropag)$$

Transition Rules: Unit

necessary = deduction(formula, assignment);

$$(UnitPropag)$$

$$l \vee l_1 \vee ... \vee l_k \in F \text{ and } \overline{l_1}, ..., \overline{l_k} \in M \text{ and } l, \bar{l} \notin M$$

Transition Rules: Unit

necessary = deduction(formula, assignment);

$$(UnitPropag)$$
$$\frac{l \vee l_1 \vee ... \vee l_k \in F \text{ and } \overline{l_1}, ..., \overline{l_k} \in M \text{ and } l, \bar{l} \notin M}{M := M + l}$$

Transition Rules: Conflict and Backtrack

$$(Conflict)$$

$$C = no\_cflct$$

Transition Rules: Conflict and Backtrack

$$\frac{}{C = no\_cflct \text{ and } \overline{l_1} \vee ... \vee \overline{l_k} \in F} (Conflict)$$

Transition Rules: Conflict and Backtrack

$$\frac{}{C = no\_cflct \text{ and } \overline{l_1} \vee ... \vee \overline{l_k} \in F \text{ and } l_1, ..., l_k \in M} (Conflict)$$

Transition Rules: Conflict and Backtrack

$$\frac{C = no\_cflct \text{ and } \overline{l_1} \vee ... \vee \overline{l_k} \in F \text{ and } l_1, ..., l_k \in M}{C := \{l_1, ..., l_k\}} \ (Conflict)$$

Transition Rules: Conflict and Backtrack

$$(Conflict)\quad \frac{C = no\_cflct \text{ and } \overline{l_1} \vee ... \vee \overline{l_k} \in F \text{ and } l_1, ..., l_k \in M}{C := \{l_1, ..., l_k\}}$$

$$(Backtrack)$$

$C = \{l, l_1, ..., l_k\}$

Transition Rules: Conflict and Backtrack

$$\frac{C = no\_cflct \text{ and } \overline{l_1} \vee ... \vee \overline{l_k} \in F \text{ and } l_1, ..., l_k \in M}{C := \{l_1, ..., l_k\}} \; (Conflict)$$

$(Backtrack)$

$C = \{l, l_1, ..., l_k\}$ & $\bar{l} \vee \overline{l_1} \vee ... \vee \overline{l_k} \in F$

Transition Rules: Conflict and Backtrack

$$\frac{C = no\_cflct \text{ and } \overline{l_1} \vee ... \vee \overline{l_k} \in F \text{ and } l_1, ..., l_k \in M}{C := \{l_1, ..., l_k\}} \;(Conflict)$$

$$\frac{C = \{l, l_1, ..., l_k\} \;\&\; \bar{l} \vee \overline{l_1} \vee ... \vee \overline{l_k} \in F \;\&\; \text{lvl } l \geq \text{lvl } l_i > 0 \text{ for (i = 1,..,k)}}{} \;(Backtrack)$$

Transition Rules: Conflict and Backtrack

$$\frac{C = no\_cflct \text{ and } \overline{l_1} \vee ... \vee \overline{l_k} \in F \text{ and } l_1, ..., l_k \in M}{C := \{l_1, ..., l_k\}} \; (Conflict)$$

$$\frac{C = \{l, l_1, ..., l_k\} \text{ \& } \overline{l} \vee \overline{l_1} \vee ... \vee \overline{l_k} \in F \text{ \& lvl } l \geq \text{lvl } l_i > 0 \text{ for (i = 1,..,k)}}{C := no\_cflct} \; (Backtrack)$$

Transition Rules: Conflict and Backtrack

$$\frac{C = no\_cflct \text{ and } \overline{l_1} \vee ... \vee \overline{l_k} \in F \text{ and } l_1, ..., l_k \in M}{C := \{l_1, ..., l_k\}} \; (Conflict)$$

$$\frac{C = \{l, l_1, ..., l_k\} \text{ \& } \overline{l} \vee \overline{l_1} \vee ... \vee \overline{l_k} \in F \text{ \& lvl } l \geq \text{lvl } l_i > 0 \text{ for (i = 1,..,k)}}{C := no\_cflct \text{ and } M := M^{[level\ l-1]} + \overline{l}^{level\ l}} \; (Backtrack)$$

Strategy

- System is *non-deterministic*

Strategy

- System is *non-deterministic*
- Restrict rule usage to speed up process

Strategy

- System is *non-deterministic*
- Restrict rule usage to speed up process
- Classic DPLL:

Strategy

- System is *non-deterministic*
- Restict rule usage to speed up process
- Classic DPLL:

(((Conflict; Backtrack) || UnitPropag)*

Strategy

- System is *non-deterministic*
- Restrict rule usage to speed up process
- Classic DPLL:

(((Conflict; Backtrack) || UnitPropag)* ; [Decide])*

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$< F, [], no\_cflct >$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$< F, [], no\_cflct > \quad \xrightarrow{Decide}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$< F, [], no\_cflct >$	$\xrightarrow{Decide}$	$< F, [\diamondsuit 1], no\_cflct >$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$< F, [], no\_cflct > \quad \xrightarrow{Decide} \quad < F, [\Diamond 1], no\_cflct > \quad \xrightarrow{UnitPropag}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$< F, [], no\_cflct > \xrightarrow{Decide} < F, [\Diamond 1], no\_cflct > \xrightarrow{UnitPropag} < F, [\Diamond 12], no\_cflct >$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$< F, [], no\_cflct > \xrightarrow{Decide} < F, [\diamondsuit 1], no\_cflct > \xrightarrow{UnitPropag}$$
$$< F, [\diamondsuit 12], no\_cflct > \xrightarrow{Decide}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$< F, [], no\_cflct > \quad \xrightarrow{Decide} \quad < F, [\Diamond 1], no\_cflct > \quad \xrightarrow{UnitPropag}$$
$$< F, [\Diamond 12], no\_cflct > \quad \xrightarrow{Decide} \quad < F, [\Diamond 12 \Diamond 3], no\_cflct >$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$\begin{array}{cccc} < F, [], no\_cflct > & \xrightarrow{\underline{Decide}} & < F, [\diamondsuit 1], no\_cflct > & \xrightarrow{\underline{UnitPropag}} \\ < F, [\diamondsuit 12], no\_cflct > & \xrightarrow{\underline{Decide}} & < F, [\diamondsuit 12 \diamondsuit 3], no\_cflct > & \xrightarrow{\underline{UnitPropag}} \end{array}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$< F, [], no\_cflct > \xrightarrow{Decide} < F, [\Diamond 1], no\_cflct > \xrightarrow{UnitPropag}$$
$$< F, [\Diamond 12], no\_cflct > \xrightarrow{Decide} < F, [\Diamond 12 \Diamond 3], no\_cflct > \xrightarrow{UnitPropag}$$
$$< F, [\Diamond 12 \Diamond 34], no\_cflct >$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$< F, [], no\_cflct > \xrightarrow{Decide} < F, [\diamondsuit 1], no\_cflct > \xrightarrow{UnitPropag}$$
$$< F, [\diamondsuit 12], no\_cflct > \xrightarrow{Decide} < F, [\diamondsuit 12 \diamondsuit 3], no\_cflct > \xrightarrow{UnitPropag}$$
$$< F, [\diamondsuit 12 \diamondsuit 34], no\_cflct > \xrightarrow{Decide}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$< F, [], no\_cflct > \xrightarrow{Decide} < F, [\Diamond 1], no\_cflct > \xrightarrow{UnitPropag}$$
$$< F, [\Diamond 12], no\_cflct > \xrightarrow{Decide} < F, [\Diamond 12 \Diamond 3], no\_cflct > \xrightarrow{UnitPropag}$$
$$< F, [\Diamond 12 \Diamond 34], no\_cflct > \xrightarrow{Decide} < F, [\Diamond 12 \Diamond 34 \Diamond 5], no\_cflct >$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$\begin{array}{cccc}
< F, [], no_cflct > & \xrightarrow{Decide} & < F, [\Diamond 1], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\Diamond 12], no_cflct > & \xrightarrow{Decide} & < F, [\Diamond 12 \Diamond 3], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\Diamond 12 \Diamond 34], no_cflct > & \xrightarrow{Decide} & < F, [\Diamond 12 \Diamond 34 \Diamond 5], no_cflct > & \xrightarrow{UnitPropag}
\end{array}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$\begin{array}{cccc}
< F, [], no_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 1], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\diamondsuit 12], no_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 12 \diamondsuit 3], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\diamondsuit 12 \diamondsuit 34], no_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 12 \diamondsuit 34 \diamondsuit 5], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\diamondsuit 12 \diamondsuit 34 \diamondsuit 56], no_cflct > & & &
\end{array}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$\begin{array}{|cccc|} \hline < F, [], no\_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 1], no\_cflct > & \xrightarrow{UnitPropag} \\ < F, [\diamondsuit 12], no\_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 12 \diamondsuit 3], no\_cflct > & \xrightarrow{UnitPropag} \\ < F, [\diamondsuit 12 \diamondsuit 34], no\_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 12 \diamondsuit 34 \diamondsuit 5], no\_cflct > & \xrightarrow{UnitPropag} \\ < F, [\diamondsuit 12 \diamondsuit 34 \diamondsuit 56], no\_cflct > & \xrightarrow{Conflict} & & \end{array}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$\begin{array}{cccc}
< F, [], no_cflct > & \xrightarrow{Decide} & < F, [\Diamond 1], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\Diamond 12], no_cflct > & \xrightarrow{Decide} & < F, [\Diamond 12 \Diamond 3], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\Diamond 12 \Diamond 34], no_cflct > & \xrightarrow{Decide} & < F, [\Diamond 12 \Diamond 34 \Diamond 5], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\Diamond 12 \Diamond 34 \Diamond 56], no_cflct > & \xrightarrow{Conflict} & < F, [\Diamond 12 \Diamond 34 \Diamond 56], \{2, 5, 6\} > &
\end{array}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$< F, [], no\_cflct >$	$\xrightarrow{Decide}$	$< F, [\Diamond 1], no\_cflct >$	$\xrightarrow{UnitPropag}$
$< F, [\Diamond 12], no\_cflct >$	$\xrightarrow{Decide}$	$< F, [\Diamond 12 \Diamond 3], no\_cflct >$	$\xrightarrow{UnitPropag}$
$< F, [\Diamond 12 \Diamond 34], no\_cflct >$	$\xrightarrow{Decide}$	$< F, [\Diamond 12 \Diamond 34 \Diamond 5], no\_cflct >$	$\xrightarrow{UnitPropag}$
$< F, [\Diamond 12 \Diamond 34 \Diamond 56], no\_cflct >$	$\xrightarrow{Conflict}$	$< F, [\Diamond 12 \Diamond 34 \Diamond 56], \{2, 5, 6\} >$	$\xrightarrow{Backtrack}$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$\begin{array}{|cccc}
< F, [], no_cflct > & \xrightarrow{\underline{Decide}} & < F, [\diamondsuit 1], no_cflct > & \xrightarrow{\underline{UnitPropag}} \\
< F, [\diamondsuit 12], no_cflct > & \xrightarrow{\underline{Decide}} & < F, [\diamondsuit 12 \diamondsuit 3], no_cflct > & \xrightarrow{\underline{UnitPropag}} \\
< F, [\diamondsuit 12 \diamondsuit 34], no_cflct > & \xrightarrow{\underline{Decide}} & < F, [\diamondsuit 12 \diamondsuit 34 \diamondsuit 5], no_cflct > & \xrightarrow{\underline{UnitPropag}} \\
< F, [\diamondsuit 12 \diamondsuit 34 \diamondsuit 56], no_cflct > & \xrightarrow{\underline{Conflict}} & < F, [\diamondsuit 12 \diamondsuit 34 \diamondsuit 56], \{2, 5, 6\} > & \xrightarrow{\underline{Backtrack}} \\
< F, [\diamondsuit 12 \diamondsuit 34\overline{5}], no_cflct > & & &
\end{array}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$\begin{array}{|cccc}
< F, [], no_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 1], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\diamondsuit 12], no_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 12 \diamondsuit 3], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\diamondsuit 12 \diamondsuit 34], no_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 12 \diamondsuit 34 \diamondsuit 5], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\diamondsuit 12 \diamondsuit 34 \diamondsuit 56], no_cflct > & \xrightarrow{Conflict} & < F, [\diamondsuit 12 \diamondsuit 34 \diamondsuit 56], \{2, 5, 6\} > & \xrightarrow{Backtrack} \\
< F, [\diamondsuit 12 \diamondsuit 34 \overline{5}], no_cflct > & \xrightarrow{Decide} & &
\end{array}$$

Classic DPLL Run

$$F_{init} = \{\overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee 6, \overline{2} \vee \overline{5} \vee \overline{6}\}$$

$$\boxed{\begin{array}{cccc}
< F, [], no_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 1], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\diamondsuit 12], no_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 12\diamondsuit 3], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\diamondsuit 12\diamondsuit 34], no_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 12\diamondsuit 34\diamondsuit 5], no_cflct > & \xrightarrow{UnitPropag} \\
< F, [\diamondsuit 12\diamondsuit 34\diamondsuit 56], no_cflct > & \xrightarrow{Conflict} & < F, [\diamondsuit 12\diamondsuit 34\diamondsuit 56], \{2, 5, 6\} > & \xrightarrow{Backtrack} \\
< F, [\diamondsuit 12\diamondsuit 34\overline{5}], no_cflct > & \xrightarrow{Decide} & < F, [\diamondsuit 12\diamondsuit 34\overline{5}6], no_cflct > &
\end{array}}$$

1. Basic State Transition System
2. Non-Chronological Backtracking
3. Two-Watched-Literals Scheme
4. VSIDS Heuristic
5. Summary: The Modern Solver

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

Decision	Unit propagation
x1 = 1	x2 = 1
x3 = 1	x4 = 1
x5 = 1	x6 = 0

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$(\neg x_1$

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$(\neg x_1 \vee x_2)$$

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$(\neg x_1 \vee x_2) \wedge (\neg x_3$$

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4)$$

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5$$

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6)$$

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

Decision	Unit propagation
x1 = 1	x2 = 1
x3 = 1	x4 = 1
x5 = 1	x6 = 0

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$M = \{\diamondsuit x_1 x_2 \diamondsuit x_3 x_4 \diamondsuit x_5 \overline{x}_6\}$$

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$M = \{\Diamond x_1 x_2 \Diamond x_3 x_4 \Diamond x_5 \overline{x}_6\}$$

Transform trail into an *implication graph*:

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$M = \{\diamondsuit x_1 x_2 \diamondsuit x_3 x_4 \diamondsuit x_5 \overline{x}_6\}$$

Transform trail into an *implication graph*:

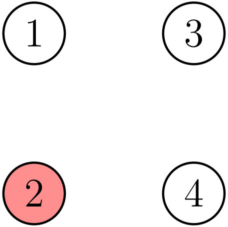
Reason for the conflict: Decision 1 and 5, or $(\overline{1} \vee \overline{5})$

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$M = \{\Diamond x_1 x_2 \Diamond x_3 x_4 \Diamond x_5 \overline{x}_6\}$$

Transform trail into an *implication graph*:



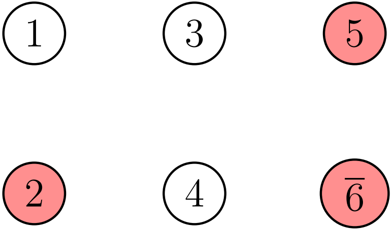
Reason for the conflict: Decision 1 and 5, or $(\overline{1} \vee \overline{5})$

Motivation

$$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_5 \vee \neg x_2)$$

$$M = \{\diamondsuit x_1 x_2 \diamondsuit x_3 x_4 \diamondsuit x_5 \overline{x}_6\}$$

Transform trail into an *implication graph*:



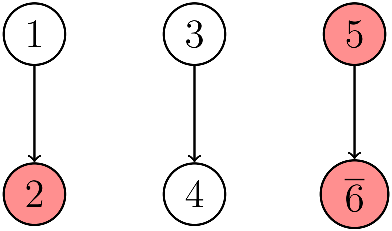
Reason for the conflict: Decision 1 and 5, or $(\overline{1} \vee \overline{5})$

Motivation

$$(\neg x_1 \lor x_2) \land (\neg x_3 \lor x_4) \land (\neg x_5 \lor \neg x_6) \land (x_6 \lor \neg x_5 \lor \neg x_2)$$

$$M = \{\Diamond x_1 x_2 \Diamond x_3 x_4 \Diamond x_5 \overline{x}_6\}$$

Transform trail into an *implication graph*:



Reason for the conflict: Decision 1 and 5, or $(\overline{1} \lor \overline{5})$

Goals

- Determine reason of current conflict

Goals

- Determine reason of current conflict
- Jump over decision levels that are irrelevant

Goals

- Determine reason of current conflict
- Jump over decision levels that are irrelevant
- Save reason to prevent future conflicts

Constructing Partial Implication Graphs

Consider a state $< F, M, \{\overline{1}, 2, \overline{3}\} >$ where F = $(\overline{9} \vee \overline{6} \vee 7 \vee \overline{8}) \wedge (8 \vee 7 \vee \overline{5}) \wedge (\overline{6} \vee 8 \vee 4) \wedge (\overline{4} \vee \overline{1}) \wedge (\overline{4} \vee 5 \vee 2) \wedge (5 \vee 7 \vee \overline{3}) \wedge (1 \vee \overline{2} \vee 3\} \cup F_{other}$ and

$M = \{...6...\overline{7}...\diamondsuit 9\overline{8}\overline{5}4\overline{1}2\overline{3}\}$

Constructing Partial Implication Graphs

Consider a state $< F, M, \{\overline{1}, 2, \overline{3}\} >$ where F = $(\overline{9} \vee \overline{6} \vee 7 \vee \overline{8}) \wedge (8 \vee 7 \vee \overline{5}) \wedge (\overline{6} \vee 8 \vee 4) \wedge (\overline{4} \vee \overline{1}) \wedge (\overline{4} \vee 5 \vee 2) \wedge (5 \vee 7 \vee \overline{3}) \wedge (1 \vee \overline{2} \vee 3\} \cup F_{other}$ and

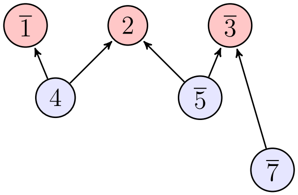
$M = \{...6...\overline{7}...\diamondsuit 9\overline{8}\overline{5}4\overline{1}2\overline{3}\}$

Constructing Partial Implication Graphs

Consider a state $< F, M, \{\bar{1}, 2, \bar{3}\} >$ where F = $(\bar{9} \vee \bar{6} \vee 7 \vee \bar{8}) \wedge (8 \vee 7 \vee \bar{5}) \wedge (\bar{6} \vee 8 \vee 4) \wedge (\bar{4} \vee \bar{1}) \wedge (\bar{4} \vee 5 \vee 2) \wedge (5 \vee 7 \vee \bar{3}) \wedge (1 \vee \bar{2} \vee 3\} \cup F_{other}$ and

$M = \{...6...\bar{7}...\diamondsuit 9\bar{8}\bar{5}4\ \bar{1}2\bar{3}\ \}$

Work backwards starting with conflict literals

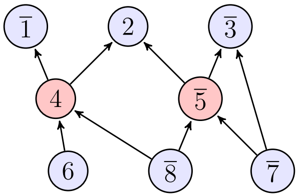


Constructing Partial Implication Graphs

Consider a state $< F, M, \{\overline{1}, 2, \overline{3}\} >$ where F = $(\overline{9} \lor \overline{6} \lor 7 \lor \overline{8}) \land (8 \lor 7 \lor \overline{5}) \land (\overline{6} \lor 8 \lor 4) \land (\overline{4} \lor \overline{1}) \land (\overline{4} \lor 5 \lor 2) \land (5 \lor 7 \lor \overline{3}) \land (1 \lor \overline{2} \lor 3\} \cup F_{other}$ and

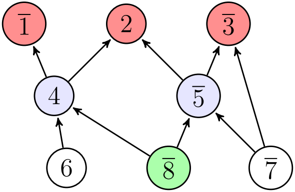
$M = \{...6...\overline{7}...\diamond 9\overline{8}\,\overline{5}4\,\overline{1}2\overline{3}\}$.

Stop at first unique implication point (UIP)



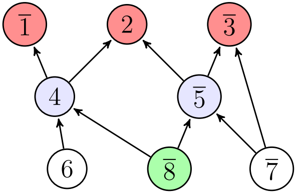
Unique Implication Point and Backjump Clause

$$M = \{\ldots 6 \ldots \overline{7} \ldots \diamondsuit 9 \overline{8} \overline{5} 4 \overline{1} 2 \overline{3}\}$$



Unique Implication Point and Backjump Clause

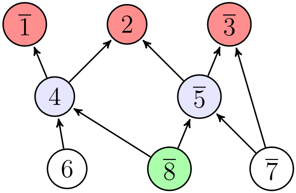
$$M = \{...6...\overline{7}...\diamondsuit 9\overline{8}\overline{5}4\overline{1}2\overline{3}\}$$



- Stop at *first* UIP ($\overline{8}$)

Unique Implication Point and Backjump Clause

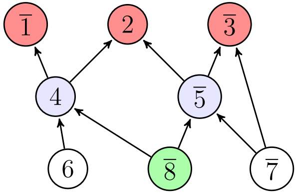
$$M = \{...6...\overline{7}...\diamondsuit 9\overline{8}\overline{5}4\overline{1}2\overline{3}\}$$



- Stop at *first* UIP ($\overline{8}$)
- Reason of the conflict are all literals without incoming edges

Unique Implication Point and Backjump Clause

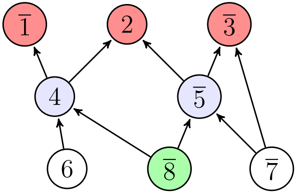
$$M = \{\ldots 6 \ldots \overline{7} \ldots \Diamond 9 \overline{8} \overline{5} 4 \overline{1} 2 \overline{3}\}$$



- Stop at *first* UIP ($\overline{8}$)
- Reason of the conflict are all literals without incoming edges
- Backjump clause is the negated disjunction of reason literals ($\overline{6} \vee 8 \vee 7$)

Unique Implication Point and Backjump Clause

$$M = \{\ldots 6 \ldots \overline{7} \ldots \diamondsuit 9 \overline{8} \overline{5} 4 \overline{1} 2 \overline{3}\}$$



- Stop at *first* UIP ($\overline{8}$)
- Reason of the conflict are all literals without incoming edges
- Backjump clause is the negated disjunction of reason literals ($\overline{6} \vee 8 \vee 7$)
- Jump to *second most recent decision level* of backjump clause

Resolution

Apply resolution until first UIP is found.
In this case: stop resolution if only one literal is in current decision level. Previous example:

Resolution

Apply resolution until first UIP is found.
In this case: stop resolution if only one literal is in current decision level. Previous example:

$$M = \{...6...\overline{7}...\diamondsuit 9\overline{8}\overline{5}4\overline{1}2\overline{3}\}$$

$$\frac{5 \vee 7 \vee \mathbf{\overline{3}} \qquad\qquad 1 \vee \overline{2} \vee \mathbf{3}}{}$$

Resolution step can now be defined as a rule and integrated in the state transition system.

Resolution

Apply resolution until first UIP is found.
In this case: stop resolution if only one literal is in current decision level. Previous example:

$$M = \{ \ldots 6 \ldots \bar{7} \ldots \diamond 9 \bar{8} \bar{5} 4 \bar{1} 2 \bar{3} \}$$

$$\dfrac{\bar{4} \vee 5 \vee \mathbf{2} \qquad \dfrac{5 \vee 7 \vee \bar{\mathbf{3}} \qquad 1 \vee \bar{2} \vee \mathbf{3}}{5 \vee 7 \vee 1 \vee \bar{\mathbf{2}}}}{}$$

Resolution step can now be defined as a rule and integrated in the state transition system.

Resolution

Apply resolution until first UIP is found.
In this case: stop resolution if only one literal is in current decision level. Previous example:

$$M = \{ ...6...\bar{7}...\Diamond 9\bar{8}\bar{5}4\bar{1}2\bar{3} \}$$

$$\cfrac{\bar{4} \lor \bar{\mathbf{1}} \qquad \cfrac{\bar{4} \lor 5 \lor \mathbf{2} \qquad \cfrac{5 \lor 7 \lor \bar{\mathbf{3}} \qquad 1 \lor \bar{2} \lor \mathbf{3}}{5 \lor 7 \lor 1 \lor \bar{\mathbf{2}}}}{\bar{4} \lor 5 \lor 7 \lor \mathbf{1}}}{}$$

Resolution step can now be defined as a rule and integrated in the state transition system.

Resolution

Apply resolution until first UIP is found.
In this case: stop resolution if only one literal is in current decision level. Previous example:

$$M = \{ ...6...\bar{7}...\diamondsuit 9\bar{8}\bar{5}4\bar{1}2\bar{3} \}$$

$$\dfrac{\bar{6} \vee 8 \vee \mathbf{4} \qquad \dfrac{\bar{4} \vee \bar{\mathbf{1}} \qquad \dfrac{\bar{4} \vee 5 \vee \mathbf{2} \qquad \dfrac{5 \vee 7 \vee \bar{\mathbf{3}} \qquad 1 \vee \bar{2} \vee \mathbf{3}}{5 \vee 7 \vee 1 \vee \bar{\mathbf{2}}}}{\bar{4} \vee 5 \vee 7 \vee \mathbf{1}}}{5 \vee 7 \vee \bar{\mathbf{4}}}}{}$$

Resolution step can now be defined as a rule and integrated in the state transition system.

Resolution

Apply resolution until first UIP is found.
In this case: stop resolution if only one literal is in current decision level. Previous example:

$$M = \{...6...\overline{7}...\diamondsuit 9\overline{8}54\overline{1}2\overline{3}\}$$

$$\dfrac{8 \vee 7 \vee \mathbf{\overline{5}} \qquad \dfrac{\overline{6} \vee 8 \vee \mathbf{4} \qquad \dfrac{\overline{4} \vee \mathbf{\overline{1}} \qquad \dfrac{\overline{4} \vee 5 \vee \mathbf{2} \qquad \dfrac{5 \vee 7 \vee \mathbf{\overline{3}} \qquad 1 \vee \overline{2} \vee \mathbf{3}}{5 \vee 7 \vee 1 \vee \mathbf{\overline{2}}}}{\overline{4} \vee 5 \vee 7 \vee \mathbf{1}}}{5 \vee 7 \vee \mathbf{\overline{4}}}}{\overline{6} \vee 8 \vee 7 \vee \mathbf{5}}}{}$$

Resolution step can now be defined as a rule and integrated in the state transition system.

Resolution

Apply resolution until first UIP is found.
In this case: stop resolution if only one literal is in current decision level. Previous example:

$$M = \{...6...\overline{7}...\diamondsuit 9\overline{8}\overline{5}4\overline{1}2\overline{3}\}$$

$$\frac{8 \lor 7 \lor \overline{\mathbf{5}} \qquad \dfrac{\overline{6} \lor 8 \lor \mathbf{4} \qquad \dfrac{\overline{4} \lor \overline{\mathbf{1}} \qquad \dfrac{\overline{4} \lor 5 \lor \mathbf{2} \qquad \dfrac{5 \lor 7 \lor \overline{\mathbf{3}} \qquad 1 \lor \overline{2} \lor \mathbf{3}}{5 \lor 7 \lor 1 \lor \overline{\mathbf{2}}}}{\overline{4} \lor 5 \lor 7 \lor \mathbf{1}}}{5 \lor 7 \lor \overline{\mathbf{4}}}}{\overline{6} \lor 8 \lor 7 \lor \mathbf{5}}}{8 \lor 7 \lor \overline{6}}$$

Resolution step can now be defined as a rule and integrated in the state transition system.

Theorem - Correctness

Theorem (Correctness): All runs of DPLL are finite. If, initialized with the set of clauses F_{init}, DPLL terminates in the state $\langle F, M, C \rangle$, then: (a) C = no_cflct or C = $\emptyset$; (b) If C = $\emptyset$ then F_{init} is unsatisfiable; (c) If C = no_cflct, then M is a model for F_{init}.

Experiments

Instance	CB	NCB
bf0432-079	>3000	3,73
ssa2670-141	>3000	101,69
ii16e1	>3000	0,53
par16-1-c	165,75	1362,53
flat200-39	656,55	1472,53
4blocksb	>3000	639,87
logistics.c	>3000	38,96
barrel5	189,88	635,12
queueinvar16	>3000	22,9
dlx2_aa	>3000	32,35
2dlx...	>3000	>3000
cnf-r3-b4-k1.2	>3000	18,69

1 Basic State Transition System

2 Non-Chronological Backtracking

3 Two-Watched-Literals Scheme

4 VSIDS Heuristic

5 Summary: The Modern Solver

Basic Definitions

- Formula F is stored as:

Basic Definitions

- Formula F is stored as:
- Set of Clauses

Basic Definitions

- Formula F is stored as:
- Set of Clauses
- Variables

Basic Definitions

- Formula F is stored as:
- Set of Clauses
- Variables

- Variables handle assignments

Basic Definitions

- Formula F is stored as:
- Set of Clauses
- Variables

- Variables handle assignments
- Clauses need to know their current status

Motivation for Lazy Structures

- Specialize in deduction techniques which require little information

Motivation for Lazy Structures

- Specialize in deduction techniques which require little information
- Lose ability to implement techniques which require whole clause status

Motivation for Lazy Structures

- Specialize in deduction techniques which require little information
- Lose ability to implement techniques which require whole clause status
- Most used deduction technique in modern solvers: unit propagation

Two-Watched-Literals Scheme

Main idea: optimize unit clause detection

- Non-lazy: Variables store pointers to all clauses with literals of that variable

Two-Watched-Literals Scheme

Main idea: optimize unit clause detection

- Non-lazy: Variables store pointers to all clauses with literals of that variable
- Updating assignments expensive

Two-Watched-Literals Scheme

Main idea: optimize unit clause detection

- Non-lazy: Variables store pointers to all clauses with literals of that variable
- Updating assignments expensive

- Lazy: Reduce pointer count to a minimum to determine unit status

Two-Watched-Literals Scheme

Main idea: optimize unit clause detection

- Non-lazy: Variables store pointers to all clauses with literals of that variable
- Updating assignments expensive

- Lazy: Reduce pointer count to a minimum to determine unit status
- Status of only *2 literals per clause* is needed to determine unit status

Two-Watched-Literals Example

	$\overline{1}$	4	$\overline{7}$	12	15	Two watched literals are freely chosen

Two-Watched-Literals Example

	$\overline{1}$	4	$\overline{7}$	12	15	Two watched literals are freely chosen
Decide 1	$\overline{1}$	4	$\overline{7}$	12	15	1 not watched, clause not visited

Two-Watched-Literals Example

	$\overline{1}$	4	$\overline{7}$	12	15	Two watched literals are freely chosen
Decide 1	$\overline{1}$	4	$\overline{7}$	12	15	1 not watched, clause not visited
Decide 7 Unit $\overline{15}$	$\overline{1}$	4	$\overline{7}$	12	15	Search for new unassigned watch

Two-Watched-Literals Example

	$\overline{1}$	4	$\overline{7}$	12	15	Two watched literals are freely chosen
Decide 1	$\overline{1}$	4	$\overline{7}$	12	15	1 not watched, clause not visited
Decide 7 Unit $\overline{15}$	$\overline{1}$	4	$\overline{7}$	12	15	Search for new unassigned watch
Decide $\overline{4}$	$\overline{1}$	4	$\overline{7}$	12	15	Search; only free literal is other watch. Clause is unit

Two-Watched-Literals Example

	$\overline{1}$	4	$\overline{7}$	12	15	Two watched literals are freely chosen
Decide 1	$\overline{1}$	4	$\overline{7}$	12	15	1 not watched, clause not visited
Decide 7 Unit $\overline{15}$	$\overline{1}$	4	$\overline{7}$	12	15	Search for new unassigned watch
Decide $\overline{4}$	$\overline{1}$	4	$\overline{7}$	12	15	Search; only free literal is other watch. Clause is unit
Conflict Backjump lvl 1	$\overline{1}$	4	$\overline{7}$	12	15	No work has to be done when backtracking

Two-Watched-Literals Example

	$\overline{1}$	4	$\overline{7}$	12	15	Two watched literals are freely chosen
Decide 1	$\overline{1}$	4	$\overline{7}$	12	15	1 not watched, clause not visited
Decide 7 Unit $\overline{15}$	$\overline{1}$	4	$\overline{7}$	12	15	Search for new unassigned watch
Decide $\overline{4}$	$\overline{1}$	4	$\overline{7}$	12	15	Search; only free literal is other watch. Clause is unit
Conflict Backjump lvl 1	$\overline{1}$	4	$\overline{7}$	12	15	No work has to be done when backtracking
Decide 12 Unit $\overline{7}$	$\overline{1}$	4	$\overline{7}$	12	15	When watch is assigned true, clause not visited

Two-Watched-Literals Example

	$\overline{1}$	4	$\overline{7}$	12	15	Two watched literals are freely chosen
Decide 1	$\overline{1}$	4	$\overline{7}$	12	15	1 not watched, clause not visited
Decide 7 Unit $\overline{15}$	$\overline{1}$	4	$\overline{7}$	12	15	Search for new unassigned watch
Decide $\overline{4}$	$\overline{1}$	4	$\overline{7}$	12	15	Search; only free literal is other watch. Clause is unit
Conflict Backjump lvl 1	$\overline{1}$	4	$\overline{7}$	12	15	No work has to be done when backtracking
Decide 12 Unit $\overline{7}$	$\overline{1}$	4	$\overline{7}$	12	15	When watch is assigned true, clause not visited
Decide 4	$\overline{1}$	4	$\overline{7}$	12	15	Watches keep moving

Two-Watched-Literals Example

	$\overline{1}$	4	$\overline{7}$	12	15	Two watched literals are freely chosen
Decide 1	$\overline{1}$	4	$\overline{7}$	12	15	1 not watched, clause not visited
Decide 7 Unit $\overline{15}$	$\overline{1}$	4	$\overline{7}$	12	15	Search for new unassigned watch
Decide $\overline{4}$	$\overline{1}$	4	$\overline{7}$	12	15	Search; only free literal is other watch. Clause is unit
Conflict Backjump lvl 1	$\overline{1}$	4	$\overline{7}$	12	15	No work has to be done when backtracking
Decide 12 Unit $\overline{7}$	$\overline{1}$	4	$\overline{7}$	12	15	When watch is assigned true, clause not visited
Decide 4	$\overline{1}$	4	$\overline{7}$	12	15	Watches keep moving

Improvements

- Assigning a variable has low computational cost

Improvements

- Assigning a variable has low computational cost
- Backtracking has **no** computational cost

Improvements

- Assigning a variable has low computational cost
- Backtracking has **no** computational cost
- Allows for solving larger instances than before

Experiments

Instance	NCBcb	NCBwl
bf0432-079	3,73	3,11
ssa2670-141	101,69	22,91
ii16e1	0,53	0,48
par16-1-c	1362,53	233,53
flat200-39	1472,53	396,95
sw100-49	17,15	12,19
4blocksb	639,87	248,04
logistics.c	38,96	27,39
facts7hh.13.simple	8,31	9,09
barrel5	635,12	146,74
queueinvar16	22,9	13,06
dlx2_aa	32,35	11,74
dlx2_cc_a_bug17	4,2	3,92
2dlx...	>3000	>3000
cnf-r3-b4-k1.2	18,69	16,48

- Overall big improvement

Experiments

Instance	NCBcb	NCBwl
bf0432-079	3,73	3,11
ssa2670-141	101,69	22,91
ii16e1	0,53	0,48
par16-1-c	1362,53	233,53
flat200-39	1472,53	396,95
sw100-49	17,15	12,19
4blocksb	639,87	248,04
logistics.c	38,96	27,39
facts7hh.13.simple	8,31	9,09
barrel5	635,12	146,74
queueinvar16	22,9	13,06
dlx2_aa	32,35	11,74
dlx2_cc_a_bug17	4,2	3,92
2dlx...	>3000	>3000
cnf-r3-b4-k1.2	18,69	16,48

- Overall big improvement
- Not enough for hardest instances

1. Basic State Transition System
2. Non-Chronological Backtracking
3. Two-Watched-Literals Scheme
4. VSIDS Heuristic
5. Summary: The Modern Solver

Motivation

- Decide rule leaves literal selection open

Motivation

- Decide rule leaves literal selection open
- Random choice often not the best choice

Motivation

- Decide rule leaves literal selection open
- Random choice often not the best choice
- Define decision scheme

Basic Heuristics

- Early heuristics were mostly state-dependent:

Basic Heuristics

- Early heuristics were mostly state-dependent:
- Literal which generates largest number of implications (Jeroslow-Wang)

Basic Heuristics

- Early heuristics were mostly state-dependent:
- Literal which generates largest number of implications (Jeroslow-Wang)
- Maximum Occurrences on Minimum sized clauses (MOM)

Basic Heuristics

- Early heuristics were mostly state-dependent:
- Literal which generates largest number of implications (Jeroslow-Wang)
- Maximum Occurrences on Minimum sized clauses (MOM)
- Mostly useful for random instances

Basic Heuristics

- Early heuristics were mostly state-dependent:

- Literal which generates largest number of implications (Jeroslow-Wang)
- Maximum Occurrences on Minimum sized clauses (MOM)

- Mostly useful for random instances
- Do not capture relevant information about structured problems

VSIDS

- Based on literal counting

VSIDS

- Based on literal counting
- Idea: favor literals in recent conflicts

VSIDS

- Based on literal counting
- Idea: favor literals in recent conflicts

- Simple and cheap heuristic

VSIDS

- Based on literal counting
- Idea: favor literals in recent conflicts

- Simple and cheap heuristic
- Dynamic adaption

VSIDS

- Based on literal counting
- Idea: favor literals in recent conflicts

- Simple and cheap heuristic
- Dynamic adaption
- State-independent

Experiments

Instance	NCBwIRSTslis	NCBwIRSTvsids
bf0432-079	3,17	2,24
ssa2670-141	23,22	1,06
ii16e1	0,5	0,42
par16-1-c	198,51	178,36
flat200-39	125,95	12,96
sw100-49	10,3	2,35
4blocksb	232,74	85,94
logistics.c	30,99	26,8
facts7hh.13.simple	9,91	7,28
barrel5	175,74	41,07
queueinvar16	13,36	16,39
dlx2_aa	12,31	10,12
dlx2_cc_a_bug17	3,58	2,86
2dlx_cc_mc_ex_bp_f2_bug005	197,55	41,66
cnf-r3-b4-k1.2	16,78	15,39

- Big improvement over static heuristic

Experiments

Instance	NCBwIRSTslis	NCBwIRSTvsids
bf0432-079	3,17	2,24
ssa2670-141	23,22	1,06
ii16e1	0,5	0,42
par16-1-c	198,51	178,36
flat200-39	125,95	12,96
sw100-49	10,3	2,35
4blocksb	232,74	85,94
logistics.c	30,99	26,8
facts7hh.13.simple	9,91	7,28
barrel5	175,74	41,07
queueinvar16	13,36	16,39
dlx2_aa	12,31	10,12
dlx2_cc_a_bug17	3,58	2,86
2dlx_cc_mc_ex_bp_f2_bug005	197,55	41,66
cnf-r3-b4-k1.2	16,78	15,39

- Big improvement over static heuristic
- Problem are now solved in acceptable time

1 Basic State Transition System

2 Non-Chronological Backtracking

3 Two-Watched-Literals Scheme

4 VSIDS Heuristic

5 Summary: The Modern Solver

Summary

We are now able to describe the modern Solver *Chaff*:

- Non-chronological backtracking

Summary

We are now able to describe the modern Solver *Chaff*:

- Non-chronological backtracking
- Two-watched-literals scheme

Summary

We are now able to describe the modern Solver *Chaff*:

- Non-chronological backtracking
- Two-watched-literals scheme
- VSIDS heuristic

Summary

We are now able to describe the modern Solver *Chaff*:

- Non-chronological backtracking
- Two-watched-literals scheme
- VSIDS heuristic

The state transition introduced helps to implement the solver and guarantees correctness.