GRASP: A Search Algorithm for Propositional Satisfiability

Authors: JoaÄo P. Marques-Silva, and Karem A. Sakallah

Presentors: Jing Ji, Qilu Guo

Introduction

- **Boolean Formula**
- **Boolean Satisfiability Problem (SAT)**
- **Conjunctive normal form (CNF)**
- **DPLL (David-Putnam-Longemann-Loveland)**
  - Decision Tree
  - Backtrack
- **Boolean constraint propagation (BCP)**

Introduction

- **Boolean Formula**

Boolean Functions can be represented by formulae defined as well-formed sequence of:

- Literals: $a, \bar{a}, b, \bar{b}$
- Boolean operators: $OR\ (+), AND\ (\cdot), NOT (\neg)$
- Parentheses: ()

Example:

$$f = \bar{a}b + a\bar{b}$$

- Literals: $a, \bar{a}, b, \bar{b}$
- Sum of Products (SoP): can intuitively think of it as disjunction of conjunctions of literals
- Product of Sum (PoS): can intuitively think of it as conjunction of disjunctions of literals

$$f = (a + b) \cdot (\bar{a} + \bar{b})$$

- **Boolean Satisfiability Problem (SAT)**
- **Conjunctive normal form (CNF)**
- **DPLL (David-Putnam-Longemann-Loveland)**
- **Boolean constraint propagation (BCP)**

Introduction

- **Boolean Satisfiability Problem (SAT)**
- Conjunctive normal form (CNF)
- DPLL (David-Putnam-Longemann-Loveland)
- Boolean constraint propagation (BCP)

The problem of determining if there exists an interpretation that satisfies a given Boolean formula

Definition:

- Given a Boolean formula $f(a, b, ..)$, is there an assignment $(a_1, b_1, ...)$ such that $f(a, b, ..) = 1$?
- If the answer is yes, then we say the formula is *satisfiable*
- Otherwise we say the formula is *unsatisfiable*

Examples:

- Is $a \cdot \bar{a}$ satisfiable?
- Is $(a + c) \cdot (b + c) \cdot (\neg a + \neg b + \neg c)$ satisfiable?
- Is $(a + b) \cdot (\neg a + \neg b) \cdot (\neg a + b)$ satisfiable?

Introduction

- **Conjunctive normal form (CNF)**

A product-of-sums (PoS) representation of a Boolean function

- A sum term in a CNF is also called as a *clause*
- Clausal normal form: a conjunction of clauses

Unit Clause Rule:

A clause is a *unit clause* if it has exactly one unassigned literal

Example:

$$\varphi = (a + c)(b + c)(\neg a + \neg b + \neg c)$$

Suppose a and b are assigned to 1. Then

$$\varphi = (1)(1)(\neg c)$$

literal

clause

The third clause is now a unit clause, and it implies that c must be set to 0 to have the formula satisfied

Introduction

- **DPLL (David-Putnam-Longemann-Loveland)**

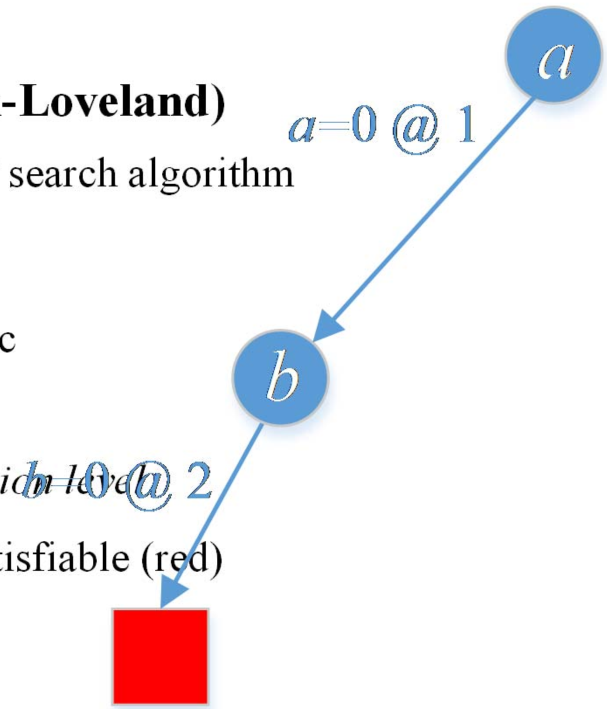
Complete, *backtracking-based depth-first* search algorithm

Example:

Decision Tree:

$f = \neg(\neg a + \neg b)$

- Nodes represent variables
- Is actually the CNF form of AND logic
- Edges represent decisions
- Assignments are associated with *decision level*
- Ends either satisfiable (green) or unsatisfiable (red)



Introduction

- **DPLL (David-Putnam-Longemann-Loveland)**

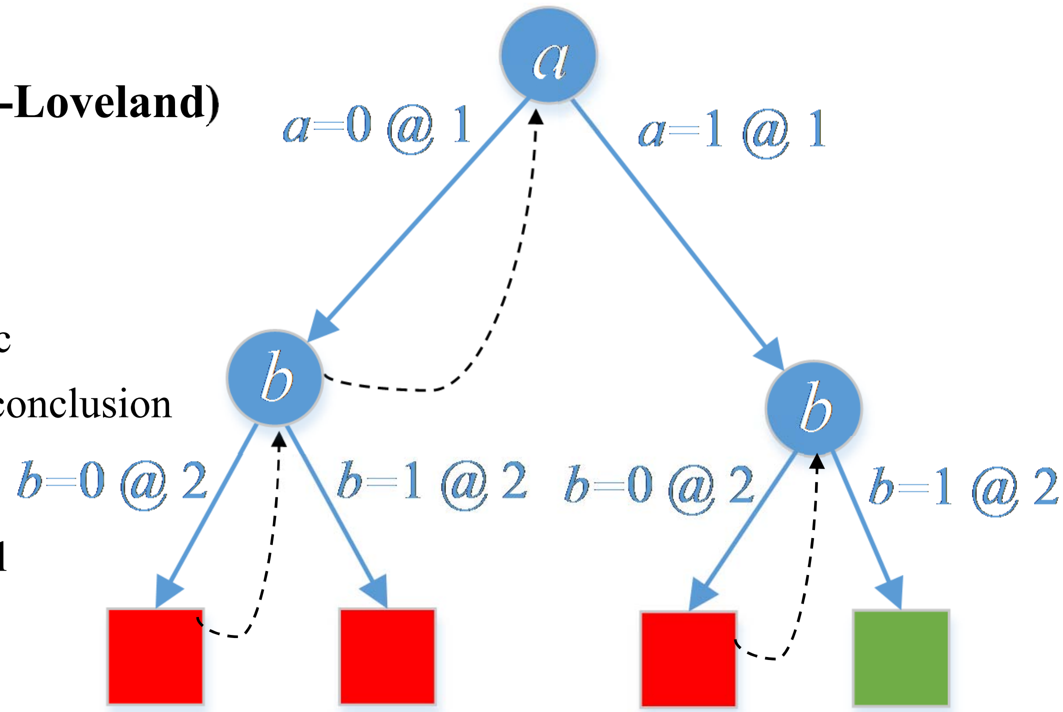
Example:

$$f = \neg(\neg a + \neg b)$$

- Is actually the CNF form of AND logic

Backtracking: If reaches an unsatisfiable conclusion

- Return back one decision level
- Redo the decision at that decision level



Introduction

- **Boolean constraint propagation (BCP)**
  - The basic mechanism for deriving implications from a given clause database
  - Unit propagation: The procedure is based on *unit clause*
  - The sequence of implications generated by BCP is captured by a *directed implication graph*

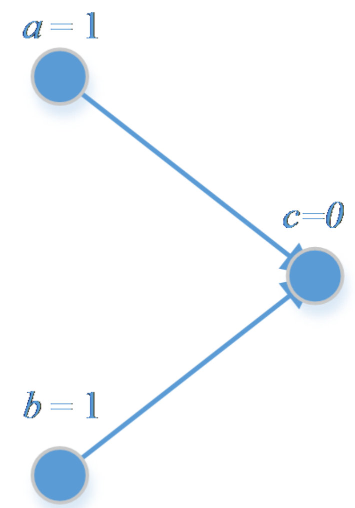
Example:

$$\varphi = (a + c)(b + c)(\neg a + \neg b + \neg c)$$

If *a* and *b* are both assigned to 1,

$$\varphi = (1)(1)(\neg c)$$

Then *c is implied* to be 0.

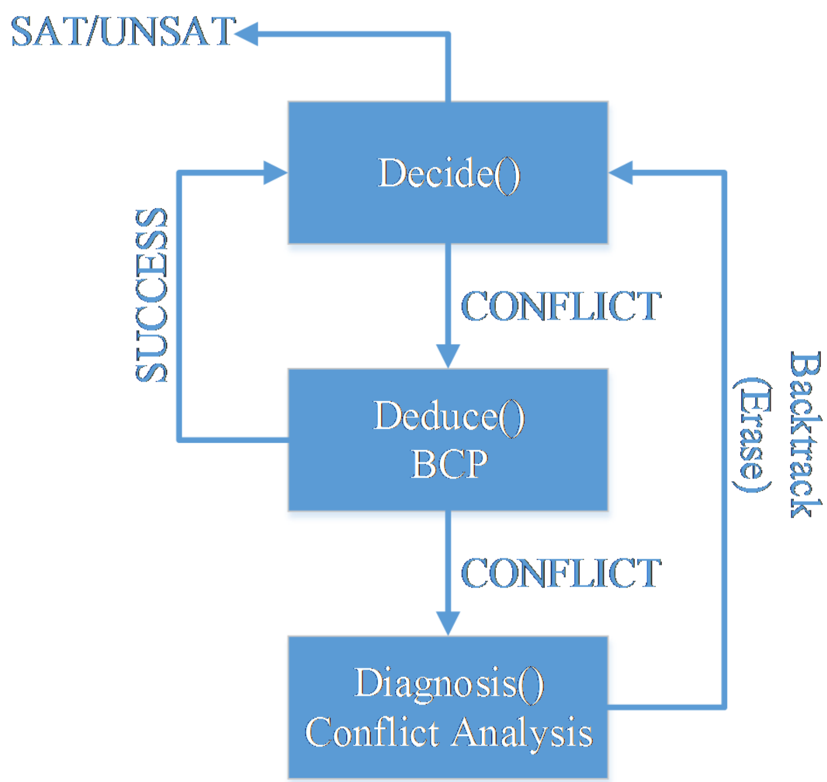


Outline

- **Search Algorithm Template**
- **Conflict Analysis Procedure**
- **Experimental Results**
- **Conclusion**

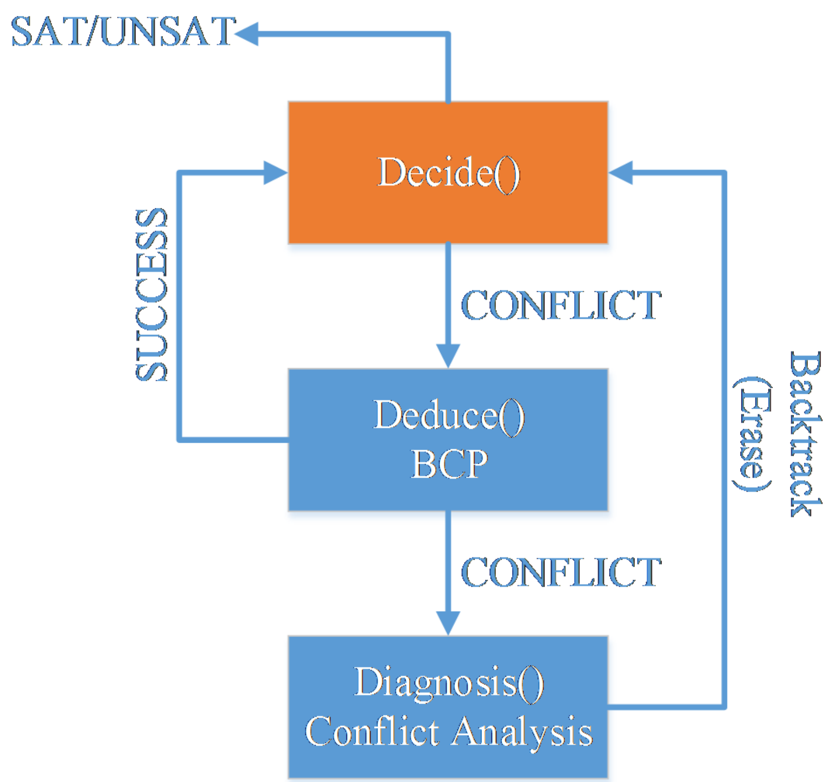
GRASP — Search Algorithm Template

- **Search Algorithm Template**
- Conflict Analysis Procedure
- Experimental Results
- Conclusion



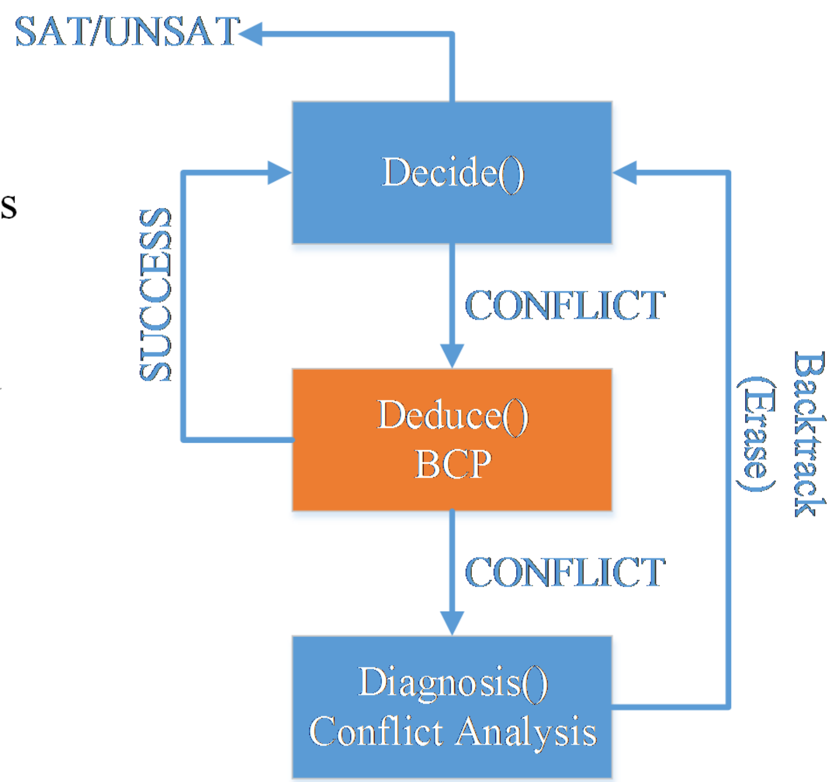
GRASP — Search Algorithm Template

- **Decision Engine**
  - Choose a decision assignment for one literal at each stage
  - Maximize the number of clauses that are directly satisfied by this assignment



GRASP – Search Algorithm Template

- **Deduction Engine (BCP)**
  - Implements BCP and (implicitly) maintains the resulting implication graph
  - Repeatedly applies the unit clause rule and check for unsatisfiable clauses



GRASP – Search Algorithm Template

- **Deduction Engine (BCP)**

$\omega_1 = (\neg x_1 + x_2) \quad \rightarrow \omega_1 = (x_2)$

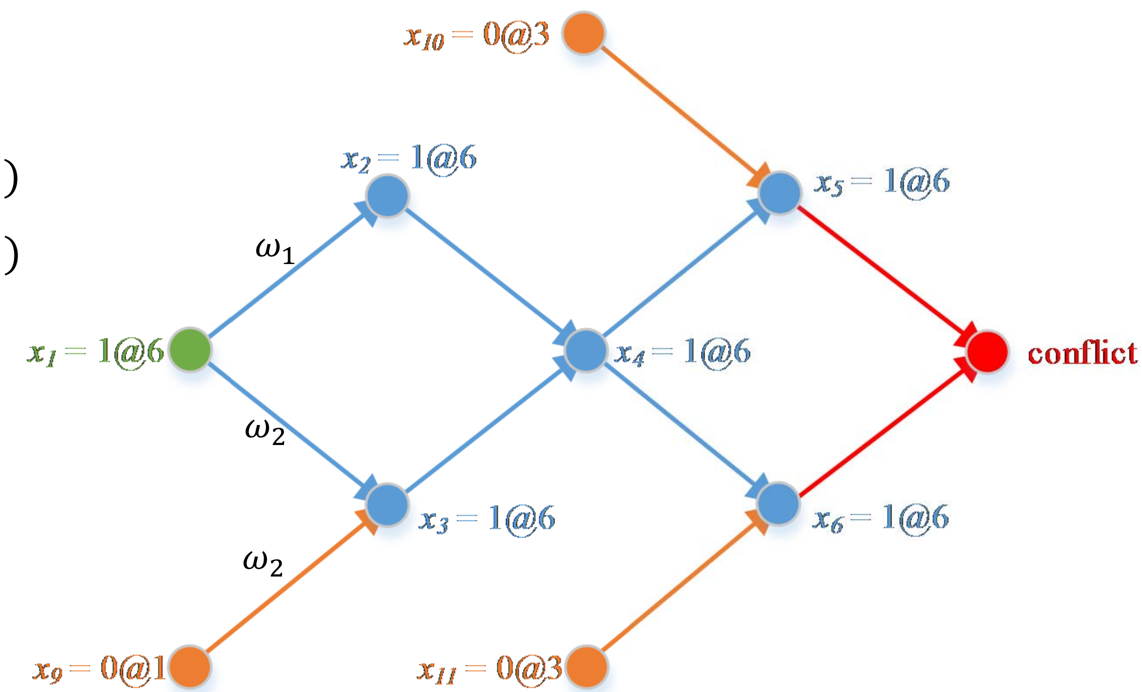
$\omega_2 = (\neg x_1 + x_3 + x_9) \quad \rightarrow \omega_2 = (x_3)$

$\omega_3 = (\neg x_2 + \neg x_3 + x_4)$

$\omega_4 = (\neg x_4 + x_5 + x_{10})$

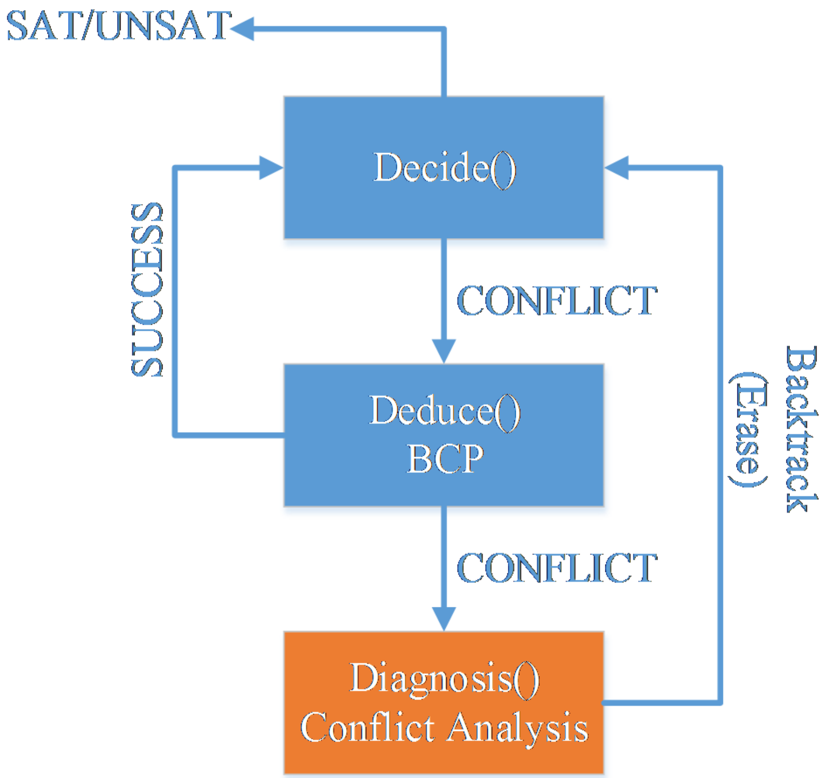
$\omega_5 = (\neg x_4 + x_6 + x_{11})$

$\omega_6 = (\neg x_5 + \neg x_6)$



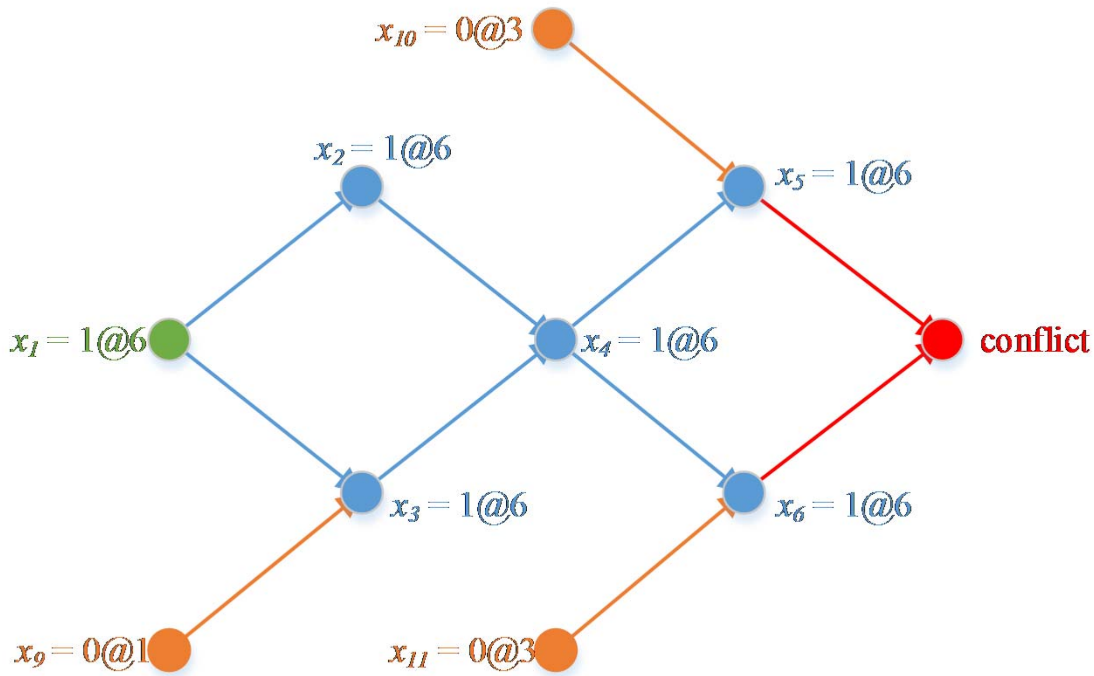
GRASP — Search Algorithm Template

- **Diagnosis Engine**
  - Identify the cause of conflict
    - Conflict learning
  - Determine the backtrack level
    - Nonchronological backtracking



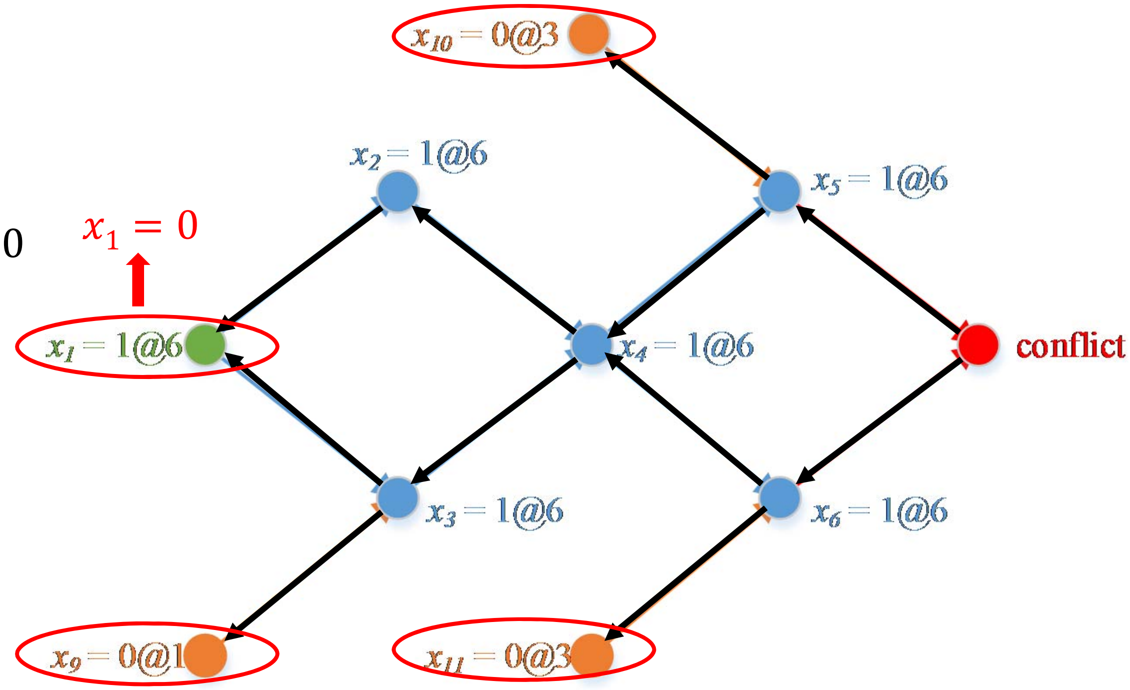
Outline

- **Conflict Analysis Procedure**
- Experimental Results
- Conclusion



Conflict Analysis

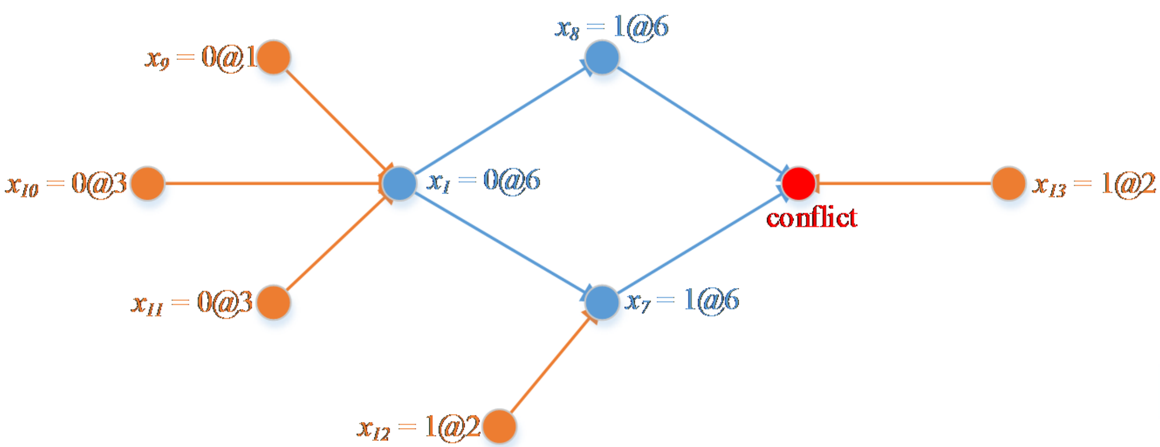
- **Conflict Analysis Procedure**
- Identify the causes of conflict
  - $x_1 = 1,\ \ x_9 = 0,\ \ x_{10} = 0,\ x_{11} \ = 0$
- Create conflict-induced clause
  - $\omega_C(\kappa) = (\neg x_1 + x_9 + x_{10} + x_{11})$
- Add $\omega_C(\kappa)$ to the clause database
- Determine a backtrack level

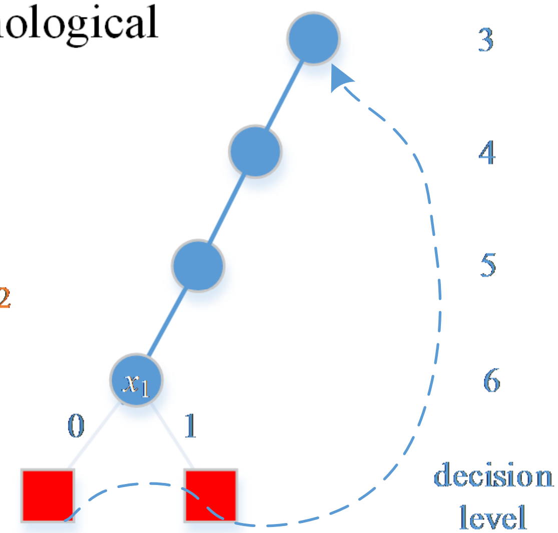


Conflict Analysis

- **Backtracking**
  - Backtrack to the highest decision level

Nonchronological





Drawbacks of Conflict Diagnosis Engine

- Overhead due to conflict analysis:
  - Outweighed by the performance gain

- Exponentially growth in the size of clause database:
  - Selectively add the conflict-induced clause to the clause database
    - $\omega_{C1(\kappa)} = (\neg x_1 + x_9 + x_4)$ ✓
    - $\omega_{C2(\kappa)} = (\neg x_1 + x_9 + x_{10} + x_{11})$ ✗

  - Reduce the size of the implicates
    - $\omega_{C(\kappa)} = (\neg x_1 + x_9 + x_{10} + x_{11})$ ✗
    - $\omega_{C1}(\kappa) = (\neg x_1 + x_9 + x_4)$ & $\omega_{C2}(\kappa) = (\neg x_4 + x_{10} + x_{11})$ ✓

Outline

- **Experimental Results**
- **Conclusion**

Experimental Results

- **CPU Time (s)**
  - Performs better at some cases
  - Performs similar to those cases POSIT performs better
  - Other solvers only perform better on certain cases

Benchmark Class	#M	GRASP	POSIT	SATO	TEGUS	DPL	GSAT
AIM-100	24	1.8	1290	60390	107.9	58510	n/a
AIM-200	24	10.8	117991	150095	14059	156196	n/a
BF	4	7.2	20037	35695	26654	40000	n/a
DUBOIS	13	34.4	77189	71528	90333	96977	n/a
II32	17	7	650.1	10004	1231	21520	83814
PRET	8	18.2	40691	40430	42579	41429	n/a
SSA	8	6.5	85.3	30092	20230	80000	n/a
AIM-50	24	0.4	0.4	12.7	2.2	10.7	n/a
II8	14	23.4	2.3	0.4	11.8	84189	27647
JNH	50	21.3	0.8	11	6055	40	n/a
PAR8	10	0.4	0.1	0.2	1.5	0.8	50005
PAR16	10	9844	72.1	10447	9983	11741	100000
II16	10	10311	10120	85522	269.6	83933	11670
HANOI	2	14480	10117	20000	11.641	20000	20000
HOLE	5	12704	937.9	362.2	21301	11404	n/a
G	4	40000	40000	40000	40000	40000	20079

#M: number of class members

Experimental Results

- **Statistics of Running GRASP**
  - Nonchronological backtracks are common
  - The growth of the clause database is acceptable

#B: number of backtracks
#NCB: number of nonchronological backtracks
%Growth: the growth in size of the clause database

Benchmark	#B	#NCB	%Growth
aim-200-2_0-yes1-2	109	50	152.63
aim-200-2_0-no-2	39	20	43.6
bf0432-007	335	124	47.99
bf1355-075	40	20	6.5
dubois50	485	175	631.92
dubois100	1438	639	1033.54
pret60_40	147	98	407.08
pret150_75	388	257	446.75
ssa0432-003	37	6	30.8
ssa2670-141	377	97	65.71
ii16b1	88325	2588	131.94

Conclusion

- GRASP
  - A faster search algorithm for solving SAT
  - Conflict learning to identify equivalent conflicting conditions
  - Nonchronological backtracking

- Future research work
  - Heuristic control of the rate of growth of the clause database
  - Improve the deduction engine

Q & A

Debate

- Will it be beneficial to split one large clause into several smaller ones?

- When doing nonchronological backtracking, is it better to return to the closest decision level, or to the level as far as possible?