SAT and SMT Solving

Sarah Winkler

SS 2018

Department of Computer Science
University of Innsbruck

Outline

- Summary of Last Week
- Conflict Analysis
- Conflict Driven Clause Learning
- Bit Vectors

Summary of Last Week

Approach

- most state-of-the-art SAT solvers use variation of Davis - Putnam - Logemann - Loveland (DPLL) procedure (1962)
- DPLL is sound and complete backtracking-based search algorithm
- can be described abstractly by transition system (Nieuwenhuis, Oliveras, Tinelli 2006)

Definition (Abstract DPLL)

- decision literal is annotated literal l^d
- state is pair $M \parallel F$ for
  - list M of (decision) literals
  - formula F in CNF
- transition rules

$$M \parallel F \qquad \Longrightarrow \qquad M' \parallel F' \quad \text{or} \quad \text{FailState}$$

Definition (DPLL Transition Rules)

- unit propagation $\qquad M \parallel F,\ C \lor l \quad \Longrightarrow \quad M\ l \parallel F,\ C \lor l$
  if $M \vDash \neg C$ and l is undefined in M
- pure literal $\qquad M \parallel F \quad \Longrightarrow \quad M\ l \parallel F$
  if l occurs in F but l^c does not occur in F, and l is undefined in M
- decide $\qquad M \parallel F \quad \Longrightarrow \quad M\ l^d \parallel F$
  if l or l^c occurs in F, and l is undefined in M
- backtrack $\qquad M\ l^d\ N \parallel F, C \quad \Longrightarrow \quad M\ l^c \parallel F, C$
  if $M\ l^d\ N \vDash \neg C$ and N contains no decision literals
- fail $\qquad M \parallel F, C \quad \Longrightarrow \quad$ FailState
  if $M \vDash \neg C$ and M contains no decision literals
- backjump $\qquad M\ l^d\ N \parallel F, C \quad \Longrightarrow \quad M\ l' \parallel F, C$
  if $M\ l^d\ N \vDash \neg C$ and $\exists$ clause $C' \lor l'$ such that
  - $F, C \vDash C' \lor l'$ $\qquad$ backjump clause
  - $M \vDash \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\ l^d\ N$

Definition

basic DPLL $\mathcal{B}$ consists of unit propagation, decide, fail, and backjump

Theorem (Termination)

there are no infinite derivations $\quad \| \, F \quad \Longrightarrow_{\mathcal{B}} \quad S_1 \quad \Longrightarrow_{\mathcal{B}} \quad S_2 \quad \Longrightarrow_{\mathcal{B}} \quad \dots$

Theorem (Correctness)

for derivation with final state S_n:

$$\| \, F \quad \Longrightarrow_{\mathcal{B}} \quad S_1 \quad \Longrightarrow_{\mathcal{B}} \quad S_2 \quad \Longrightarrow_{\mathcal{B}} \quad \dots \quad \Longrightarrow_{\mathcal{B}} \quad S_n$$

- *if* $S_n = \text{FailState}$ *then* F *is unsatisfiable*
- *if* $S_n = M \| F'$ *then* F *is satisfiable and* $M \vDash F$

Conflict Analysis

Outline

- Summary of Last Week
- Conflict Analysis
- Conflict Driven Clause Learning
- Bit Vectors

Backjump: Idea

- backjump clause $C' \vee l'$ is entailed by formula (magically detected)
- prefix M of current literal list entails $\neg C'$, so l' must be true

Backjump: Idea

- backjump clause $C' \vee l'$ is entailed by formula (magically detected)
- prefix M of current literal list entails $\neg C'$, so l' must be true

Backjump to Definition

- backjump $\quad M\, l^d\, N \parallel F, C \quad \Longrightarrow \quad M\, l' \parallel F, C$

  if $M\, l^d\, N \models \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \models C' \vee l'$ backjump clause
  - $M \models \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\, l^d\, N$

Backjump: Idea

- backjump clause $C' \vee l'$ is entailed by formula (magically detected)
- prefix M of current literal list entails $\neg C'$, so l' must be true

Backjump to Definition

- backjump $\quad M\, l^d\, N \parallel F, C \quad \Longrightarrow \quad M\, l' \parallel F, C$

  if $M\, l^d\, N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \vDash C' \vee l'$ — backjump clause
  - $M \vDash \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\, l^d\, N$

Example

$1^d\ 2\ 3^d\ 4^d\ \overline{5} \parallel \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5,\ \overline{1} \vee \overline{5} \vee 6,\ \overline{2} \vee \overline{5} \vee \overline{6}$

Backjump: Idea

- backjump clause $C' \vee l'$ is entailed by formula (magically detected)
- prefix M of current literal list entails $\neg C'$, so l' must be true

Backjump to Definition

- backjump $M\ l^d\ N \parallel F, C \quad \Longrightarrow \quad M\ l' \parallel F, C$
  if $M\ l^d\ N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \vDash C' \vee l'$ backjump clause
  - $M \vDash \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\ l^d\ N$

Example

$1^d\ 2\ 3^d\ 4^d\ \overline{5} \parallel \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5,\ \overline{1} \vee \overline{5} \vee 6,\ \overline{2} \vee \overline{5} \vee \overline{6}$

$M = 1^d\ 2 \qquad l = 3 \qquad N = 4^d\ \overline{5} \qquad C = \overline{4} \vee 5 \qquad C' = \overline{1} \qquad l' = \overline{5}$

Backjump: Idea

- backjump clause $C' \vee l'$ is entailed by formula (magically detected)
- prefix M of current literal list entails $\neg C'$, so l' must be true

Backjump to Definition

- backjump $\quad M\, l^d\, N \parallel F, C \quad \Longrightarrow \quad M\, l' \parallel F, C$

  if $M\, l^d\, N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \vDash C' \vee l'$ backjump clause
  - $M \vDash \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\, l^d\, N$

Example

$1^d\, 2\, 3^d\, 4^d\, \overline{5} \parallel \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5,\ \overline{1} \vee \overline{5} \vee 6,\ \overline{2} \vee \overline{5} \vee \overline{6}$

$M = 1^d\, 2 \qquad l = 3 \qquad N = 4^d\, \overline{5} \qquad C = \overline{4} \vee 5 \qquad C' = \overline{1} \qquad l' = \overline{5}$

- $1^d\, 2\, 3^d\, 4^d\, \overline{5} \vDash \neg(\overline{4} \vee 5)$

Backjump: Idea

- backjump clause $C' \vee l'$ is entailed by formula (magically detected)
- prefix M of current literal list entails $\neg C'$, so l' must be true

Backjump to Definition

- backjump $\quad M\ l^d\ N \parallel F, C \implies M\ l' \parallel F, C$
  if $M\ l^d\ N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \vDash C' \vee l'$ backjump clause
  - $M \vDash \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\ l^d\ N$

Example

$1^d\ 2\ 3^d\ 4^d\ \overline{5} \parallel \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5,\ \overline{1} \vee \overline{5} \vee 6,\ \overline{2} \vee \overline{5} \vee \overline{6}$

$M = 1^d\ 2 \qquad l = 3 \qquad N = 4^d\ \overline{5} \qquad C = \overline{4} \vee 5 \qquad C' = \overline{1} \qquad l' = \overline{5}$

- $1^d\ 2\ 3^d\ 4^d\ \overline{5} \ \vDash\ \neg(\overline{4} \vee 5)$
- backjump clause $C' \vee l' = \overline{1} \vee \overline{5}$ satisfies $\varphi \vDash C' \vee l'$

Backjump: Idea

- backjump clause $C' \vee l'$ is entailed by formula (magically detected)
- prefix M of current literal list entails $\neg C'$, so l' must be true

Backjump to Definition

- backjump $\quad M\, l^d\, N \parallel F, C \quad \Longrightarrow \quad M\, l' \parallel F, C$

  if $M\, l^d\, N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \vDash C' \vee l'$ — backjump clause
  - $M \vDash \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\, l^d\, N$

Example

$1^d\; 2\; 3^d\; 4^d\; \overline{5} \parallel \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5,\ \overline{1} \vee \overline{5} \vee 6,\ \overline{2} \vee \overline{5} \vee \overline{6}$

$M = 1^d\, 2 \qquad l = 3 \qquad N = 4^d\, \overline{5} \qquad C = \overline{4} \vee 5 \qquad C' = \overline{1} \qquad l' = \overline{5}$

- $1^d\; 2\; 3^d\; 4^d\; \overline{5} \vDash \neg(\overline{4} \vee 5)$
- backjump clause $C' \vee l' = \overline{1} \vee \overline{5}$ satisfies $\varphi \vDash C' \vee l'$
- $1^d\; 2 \vDash 1$

Backjump: Idea

- backjump clause $C' \vee l'$ is entailed by formula (magically detected)
- prefix M of current literal list entails $\neg C'$, so l' must be true

Backjump to Definition

- backjump $M\ l^d\ N \parallel F, C \quad \Longrightarrow \quad M\ l' \parallel F, C$
  if $M\ l^d\ N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \vDash C' \vee l'$ backjump clause
  - $M \vDash \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\ l^d\ N$

Example

$1^d\ 2\ 3^d\ 4^d\ \overline{5} \parallel \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5,\ \overline{1} \vee \overline{5} \vee 6,\ \overline{2} \vee \overline{5} \vee \overline{6}$

$M = 1^d\ 2 \qquad l = 3 \qquad N = 4^d\ \overline{5} \qquad C = \overline{4} \vee 5 \qquad C' = \overline{1} \qquad l' = \overline{5}$

- $1^d\ 2\ 3^d\ 4^d\ \overline{5} \vDash \neg(\overline{4} \vee 5)$
- backjump clause $C' \vee l' = \overline{1} \vee \overline{5}$ satisfies $\varphi \vDash C' \vee l'$
- $1^d\ 2 \vDash 1$ and 5 is undefined in $1^d\ 2$ but occurs in formula

Backjump: Idea

- backjump clause $C' \vee l'$ is entailed by formula (magically detected)
- prefix M of current literal list entails $\neg C'$, so l' must be true

Backjump to Definition

- backjump $\qquad M\ l^d\ N \parallel F, C \quad \Longrightarrow \quad M\ l' \parallel F, C$
  if $M\ l^d\ N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \vDash C' \vee l'$ backjump clause
  - $M \vDash \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\ l^d\ N$

Example

$$1^d\ 2\ 3^d\ 4^d\ \overline{5} \parallel \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5,\ \overline{1} \vee \overline{5} \vee 6,\ \overline{2} \vee \overline{5} \vee \overline{6}$$

$$\Longrightarrow\ 1^d\ 2\ \overline{5} \parallel \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4 \vee 5,\ \overline{2} \vee \overline{4} \vee \overline{5},\ 4 \vee \overline{5},\ \overline{4} \vee 5,\ \overline{1} \vee \overline{5} \vee 6,\ \overline{2} \vee \overline{5} \vee \overline{6}$$

$M = 1^d\ 2 \qquad l = 3 \qquad N = 4^d\ \overline{5} \qquad C = \overline{4} \vee 5 \qquad C' = \overline{1} \qquad l' = \overline{5}$

- $1^d\ 2\ 3^d\ 4^d\ \overline{5}\ \vDash\ \neg(\overline{4} \vee 5)$
- backjump clause $C' \vee l' = \overline{1} \vee \overline{5}$ satisfies $\varphi \vDash C' \vee l'$
- $1^d\ 2 \vDash 1$ and 5 is undefined in $1^d\ 2$ but occurs in formula

Outline

- Summary of Last Week
- Conflict Analysis
- Conflict Driven Clause Learning
- Bit Vectors

Desirable Properties of Backjump Clauses

- small
- should trigger progress

How to Determine Backjump Clauses?

- implication graph
- resolution

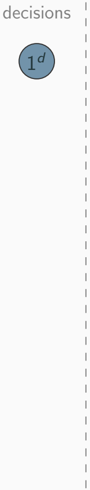
Example: Implication Graph

$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$

decisions

Example: Implication Graph

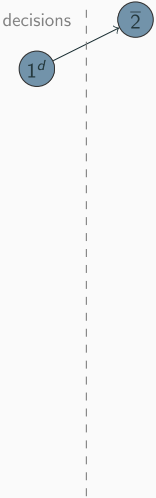
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision

Example: Implication Graph

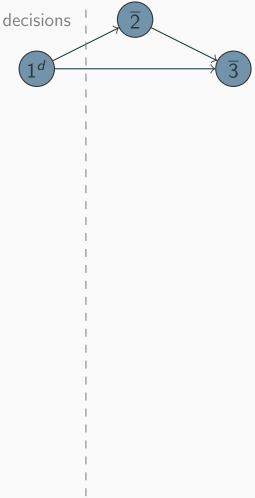
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$

Example: Implication Graph

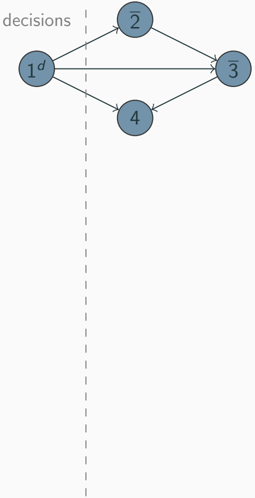
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$

Example: Implication Graph

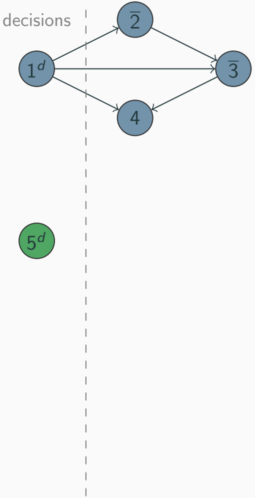
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$

Example: Implication Graph

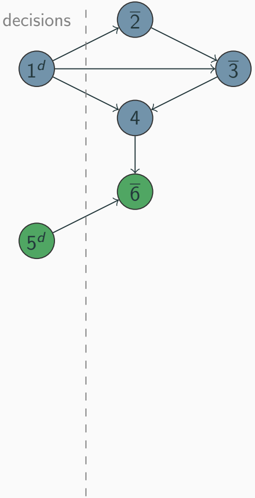
$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$
$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision

Example: Implication Graph

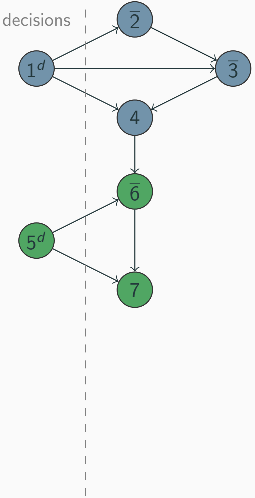
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$

Example: Implication Graph

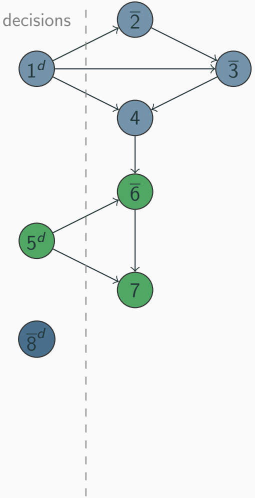
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$

Example: Implication Graph

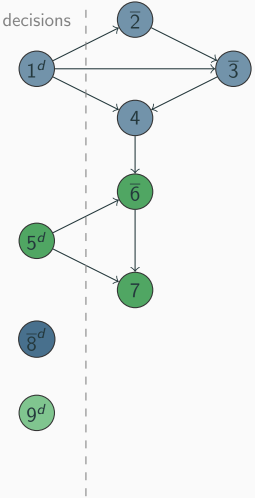
$$\varphi = (\overline{1} \lor \overline{2}) \land (\overline{1} \lor 2 \lor \overline{3}) \land (\overline{1} \lor 3 \lor 4) \land (\overline{4} \lor \overline{5} \lor \overline{6}) \land (\overline{5} \lor 6 \lor 7) \land$$
$$(\overline{7} \lor 8 \lor \overline{9} \lor 10) \land (\overline{10} \lor \overline{11}) \land (\overline{10} \lor 12) \land (\overline{12} \lor \overline{13}) \land (6 \lor 11 \lor 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \lor \overline{2}$
	$\overline{3}$	$\overline{1} \lor 2 \lor \overline{3}$
	4	$\overline{1} \lor 3 \lor 4$
2	5	decision
	$\overline{6}$	$\overline{4} \lor \overline{5} \lor \overline{6}$
	7	$\overline{5} \lor 6 \lor 7$
3	$\overline{8}$	decision

Example: Implication Graph

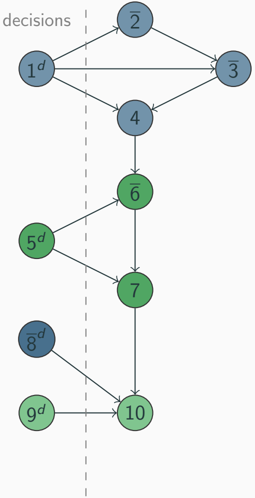
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$
3	$\overline{8}$	decision
4	9	decision

Example: Implication Graph

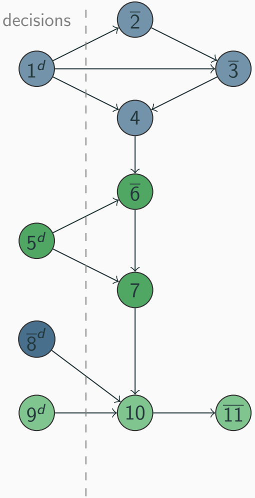
$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$
$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$
3	$\overline{8}$	decision
4	9	decision
	10	$\overline{7} \vee 8 \vee \overline{9} \vee 10$

Example: Implication Graph

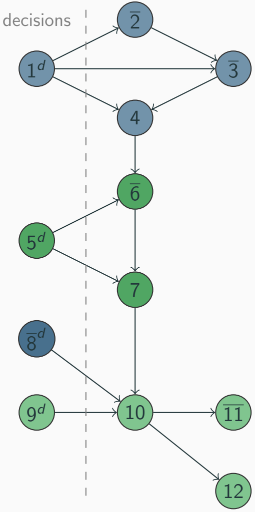
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$
3	$\overline{8}$	decision
4	9	decision
	10	$\overline{7} \vee 8 \vee \overline{9} \vee 10$
	$\overline{11}$	$\overline{10} \vee \overline{11}$

Example: Implication Graph

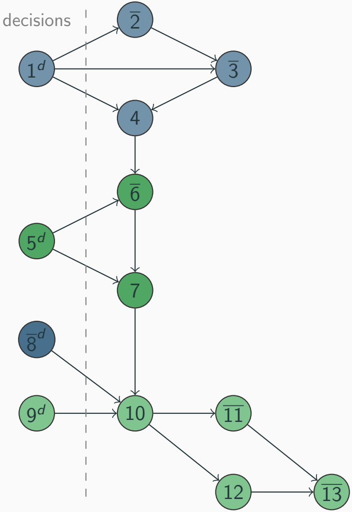
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$
3	$\overline{8}$	decision
4	9	decision
	10	$\overline{7} \vee 8 \vee \overline{9} \vee 10$
	$\overline{11}$	$\overline{10} \vee \overline{11}$
	12	$\overline{10} \vee 12$

Example: Implication Graph

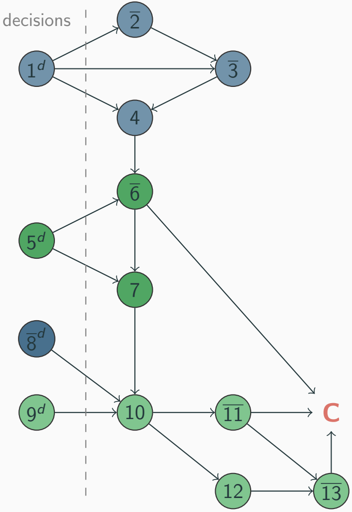
$$\varphi = (\overline{1} \lor \overline{2}) \land (\overline{1} \lor 2 \lor \overline{3}) \land (\overline{1} \lor 3 \lor 4) \land (\overline{4} \lor \overline{5} \lor \overline{6}) \land (\overline{5} \lor 6 \lor 7) \land (\overline{7} \lor 8 \lor \overline{9} \lor 10) \land (\overline{10} \lor \overline{11}) \land (\overline{10} \lor 12) \land (\overline{12} \lor \overline{13}) \land (6 \lor 11 \lor 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \lor \overline{2}$
	$\overline{3}$	$\overline{1} \lor 2 \lor \overline{3}$
	4	$\overline{1} \lor 3 \lor 4$
2	5	decision
	$\overline{6}$	$\overline{4} \lor \overline{5} \lor \overline{6}$
	7	$\overline{5} \lor 6 \lor 7$
3	$\overline{8}$	decision
4	9	decision
	10	$\overline{7} \lor 8 \lor \overline{9} \lor 10$
	$\overline{11}$	$\overline{10} \lor \overline{11}$
	12	$\overline{10} \lor 12$
	$\overline{13}$	$\overline{12} \lor \overline{13}$

Example: Implication Graph

$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$

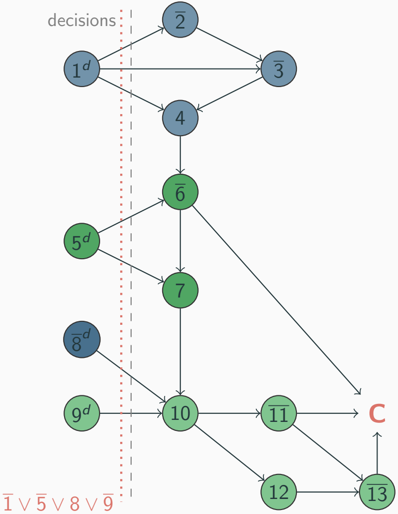


level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$
3	$\overline{8}$	decision
4	9	decision
	10	$\overline{7} \vee 8 \vee \overline{9} \vee 10$
	$\overline{11}$	$\overline{10} \vee \overline{11}$
	12	$\overline{10} \vee 12$
	$\overline{13}$	$\overline{12} \vee \overline{13}$

next

Example: Implication Graph

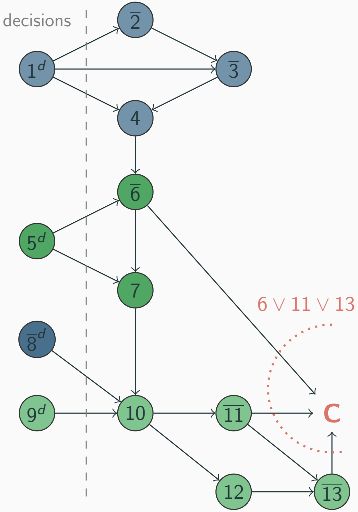
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge (\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$
3	$\overline{8}$	decision
4	9	decision
	10	$\overline{7} \vee 8 \vee \overline{9} \vee 10$
	$\overline{11}$	$\overline{10} \vee \overline{11}$
	12	$\overline{10} \vee 12$
	$\overline{13}$	$\overline{12} \vee \overline{13}$

Example: Implication Graph

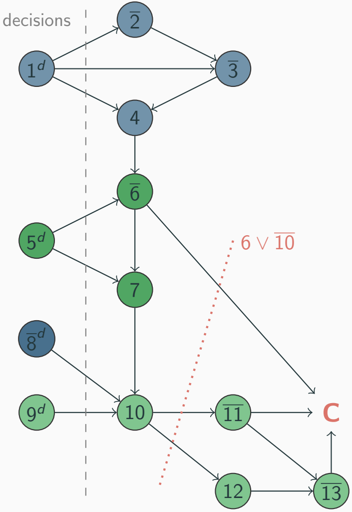
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge (\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$
3	$\overline{8}$	decision
4	9	decision
	10	$\overline{7} \vee 8 \vee \overline{9} \vee 10$
	$\overline{11}$	$\overline{10} \vee \overline{11}$
	12	$\overline{10} \vee 12$
	$\overline{13}$	$\overline{12} \vee \overline{13}$

Example: Implication Graph

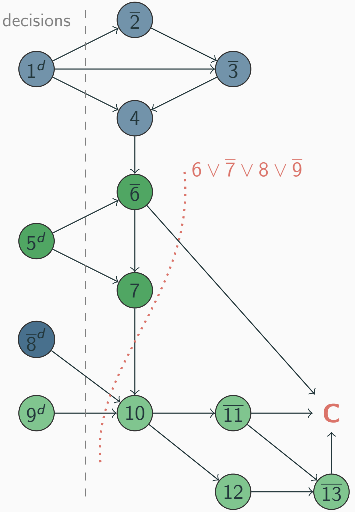
$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$
3	$\overline{8}$	decision
4	9	decision
	10	$\overline{7} \vee 8 \vee \overline{9} \vee 10$
	$\overline{11}$	$\overline{10} \vee \overline{11}$
	12	$\overline{10} \vee 12$
	$\overline{13}$	$\overline{12} \vee \overline{13}$

Example: Implication Graph

$$\varphi = (\overline{1} \vee \overline{2}) \wedge (\overline{1} \vee 2 \vee \overline{3}) \wedge (\overline{1} \vee 3 \vee 4) \wedge (\overline{4} \vee \overline{5} \vee \overline{6}) \wedge (\overline{5} \vee 6 \vee 7) \wedge$$
$$(\overline{7} \vee 8 \vee \overline{9} \vee 10) \wedge (\overline{10} \vee \overline{11}) \wedge (\overline{10} \vee 12) \wedge (\overline{12} \vee \overline{13}) \wedge (6 \vee 11 \vee 13)$$



level	literal	reason
1	1	decision
	$\overline{2}$	$\overline{1} \vee \overline{2}$
	$\overline{3}$	$\overline{1} \vee 2 \vee \overline{3}$
	4	$\overline{1} \vee 3 \vee 4$
2	5	decision
	$\overline{6}$	$\overline{4} \vee \overline{5} \vee \overline{6}$
	7	$\overline{5} \vee 6 \vee 7$
3	$\overline{8}$	decision
4	9	decision
	10	$\overline{7} \vee 8 \vee \overline{9} \vee 10$
	$\overline{11}$	$\overline{10} \vee \overline{11}$
	12	$\overline{10} \vee 12$
	$\overline{13}$	$\overline{12} \vee \overline{13}$

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right

Key Observations

- if $l_1 \to l'_1, \dots, l_k \to l'_k$ are cut edges then $l_1^c \vee \dots \vee l_k^c$ is entailed clause

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right

Key Observations

- if $l_1 \to l'_1, \ldots, l_k \to l'_k$ are cut edges then $l_1^c \vee \cdots \vee l_k^c$ is entailed clause

Example

- $\overline{1} \vee \overline{5} \vee 8 \vee \overline{9}$

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right

Key Observations

- if $l_1 \to l_1', \ldots, l_k \to l_k'$ are cut edges then $l_1^c \vee \cdots \vee l_k^c$ is entailed clause

Example

- $\overline{1} \vee \overline{5} \vee 8 \vee \overline{9}$
- $6 \vee 11 \vee 13$

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right

Key Observations

- if $l_1 \to l'_1, \dots, l_k \to l'_k$ are cut edges then $l_1^c \vee \cdots \vee l_k^c$ is entailed clause

Example

- $\overline{1} \vee \overline{5} \vee 8 \vee \overline{9}$
- $6 \vee 11 \vee 13$
- $6 \vee \overline{10}$

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right

Key Observations

- if $l_1 \to l'_1, \ldots, l_k \to l'_k$ are cut edges then $l_1^c \vee \cdots \vee l_k^c$ is entailed clause

Example

- $\overline{1} \vee \overline{5} \vee 8 \vee \overline{9}$
- $6 \vee 11 \vee 13$
- $6 \vee \overline{10}$
- $6 \vee \overline{7} \vee 8 \vee \overline{9}$

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right
- literal l in implication graph is unique implication point (UIP) if all paths from last decision literal to conflict node go through l

Key Observations

- if $l_1 \to l_1', \ldots, l_k \to l_k'$ are cut edges then $l_1^c \vee \cdots \vee l_k^c$ is entailed clause

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right
- literal l in implication graph is unique implication point (UIP) if all paths from last decision literal to conflict node go through l

Key Observations

- if $l_1 \to l_1', \dots, l_k \to l_k'$ are cut edges then $l_1^c \vee \cdots \vee l_k^c$ is entailed clause

Example

- UIPs are 9 and 10

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right
- literal l in implication graph is unique implication point (UIP) if all paths from last decision literal to conflict node go through l
- first UIP is UIP closest to conflict node

Key Observations

- if $l_1 \to l_1', \dots, l_k \to l_k'$ are cut edges then $l_1^c \vee \dots \vee l_k^c$ is entailed clause

Example

- UIPs are 9 and 10

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right
- literal l in implication graph is unique implication point (UIP) if all paths from last decision literal to conflict node go through l
- first UIP is UIP closest to conflict node

Key Observations

- if $l_1 \to l_1', \ldots, l_k \to l_k'$ are cut edges then $l_1^c \vee \cdots \vee l_k^c$ is entailed clause

Example

- UIPs are 9 and 10
- first UIP is 10

What to Learn from That?

Definitions

- cut of implication graph has at least all decision literals on the left, and at least the conflict node on the right
- literal l in implication graph is unique implication point (UIP) if all paths from last decision literal to conflict node go through l
- first UIP is UIP closest to conflict node

Key Observations

- if $l_1 \to l_1', \ldots, l_k \to l_k'$ are cut edges then $l_1^c \vee \cdots \vee l_k^c$ is entailed clause
- last decision literal is UIP

Example

- UIPs are 9 and 10
- first UIP is 10

Definition (Implication Graph)

Consider DPLL derivation to $\| F \quad \Longrightarrow^*_{\mathcal{B}} M \| F'$.

Definition (Implication Graph)

Consider DPLL derivation to $\| F \quad \Longrightarrow_{\mathcal{B}}^{*} M \| F'$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for every decision literal l in M

Definition (Implication Graph)

Consider DPLL derivation to $\| F \quad \Longrightarrow_{\mathcal{B}}^{*} M \| F'$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for every decision literal l in M
- repeat until there is no change:
  if $\exists$ clause $l_1 \vee \ldots l_m \vee l'$ in F' such that there are already nodes $l_1^c, \ldots, l_m^c$

Definition (Implication Graph)

Consider DPLL derivation to $\parallel F \quad \Longrightarrow_{\mathcal{B}}^{*} M \parallel F'$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for every decision literal l in M
- repeat until there is no change:
  if $\exists$ clause $l_1 \vee \ldots l_m \vee l'$ in F' such that there are already nodes $l_1^c, \ldots, l_m^c$
  - add node l' if not yet present

Definition (Implication Graph)

Consider DPLL derivation to $\| F \quad \Longrightarrow^*_{\mathcal{B}} M \| F'$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for every decision literal l in M
- repeat until there is no change:
  if $\exists$ clause $l_1 \vee \ldots l_m \vee l'$ in F' such that there are already nodes $l_1^c, \ldots, l_m^c$
  - add node l' if not yet present
  - add edges $l_i^c \to l'$ for all $1 \leqslant i \leqslant m$ if not yet present

Definition (Implication Graph)

Consider DPLL derivation to $\parallel F \quad \Longrightarrow_{\mathcal{B}}^{*} M \parallel F'$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for every decision literal l in M
- repeat until there is no change:

  if $\exists$ clause $l_1 \vee \dots l_m \vee l'$ in F' such that there are already nodes $l_1^c, \dots, l_m^c$
  - add node l' if not yet present
  - add edges $l_i^c \to l'$ for all $1 \leqslant i \leqslant m$ if not yet present
- if $\exists$ clause $l_1' \vee \cdots \vee l_k'$ in F' such that there are nodes $l_1'^c, \dots, l_k'^c$

Definition (Implication Graph)

Consider DPLL derivation to $\| F \quad \Longrightarrow_{\mathcal{B}}^{*} M \| F'$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for every decision literal l in M
- repeat until there is no change:
  if $\exists$ clause $l_1 \vee \dots l_m \vee l'$ in F' such that there are already nodes $l_1^c, \dots, l_m^c$
  - add node l' if not yet present
  - add edges $l_i^c \to l'$ for all $1 \leqslant i \leqslant m$ if not yet present
- if $\exists$ clause $l_1' \vee \cdots \vee l_k'$ in F' such that there are nodes $l_1'^c, \dots, l_k'^c$
  - add conflict node labeled C

Definition (Implication Graph)

Consider DPLL derivation to $\parallel F \quad \Longrightarrow^*_{\mathcal{B}} M \parallel F'$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for every decision literal l in M
- repeat until there is no change:
  if $\exists$ clause $l_1 \vee \dots l_m \vee l'$ in F' such that there are already nodes $l_1^c, \dots, l_m^c$
  - add node l' if not yet present
  - add edges $l_i^c \to l'$ for all $1 \leqslant i \leqslant m$ if not yet present
- if $\exists$ clause $l_1' \vee \cdots \vee l_k'$ in F' such that there are nodes $l_1'^c, \dots, l_k'^c$
  - add conflict node labeled C
  - add edges $l_i'^c \to C$

Definition (Implication Graph)

Consider DPLL derivation to $\| F \quad \Longrightarrow_{\mathcal{B}}^{*} M \| F'$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for every decision literal l in M
- repeat until there is no change:
  if $\exists$ clause $l_1 \vee \dots l_m \vee l'$ in F' such that there are already nodes $l_1^c, \dots, l_m^c$
  - add node l' if not yet present
  - add edges $l_i^c \to l'$ for all $1 \leqslant i \leqslant m$ if not yet present
- if $\exists$ clause $l_1' \vee \dots \vee l_k'$ in F' such that there are nodes $l_1'^c, \dots, l_k'^c$
  - add conflict node labeled C
  - add edges $l_i'^c \to C$

Lemma

if edges intersected by cut are $l_1 \to l_1', \dots, l_k \to l_k'$ *then* $F' \vDash l_1^c \vee l_k^c$

Definition (Implication Graph)

Consider DPLL derivation to $\| F \quad \Longrightarrow^*_{\mathcal{B}} M \| F'$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for every decision literal l in M
- repeat until there is no change:
  if $\exists$ clause $l_1 \vee \dots l_m \vee l'$ in F' such that there are already nodes $l_1^c, \dots, l_m^c$
  - add node l' if not yet present
  - add edges $l_i^c \to l'$ for all $1 \leqslant i \leqslant m$ if not yet present
- if $\exists$ clause $l_1' \vee \dots \vee l_k'$ in F' such that there are nodes $l_1'^c, \dots, l_k'^c$
  - add conflict node labeled C
  - add edges $l_i'^c \to C$

potential backjump clause

Lemma

if edges intersected by cut are $l_1 \to l_1', \dots, l_k \to l_k'$ *then* $F' \vDash l_1^c \vee l_k^c$

Definition (Implication Graph)

Consider DPLL derivation to $\| F \quad \Longrightarrow_{\mathcal{B}}^{*} M \| F'$.

Implication graph is a directed acyclic graph constructed as follows:

- add node labelled l for … M
- repeat until there is no change:
  if $\exists$ clause $l_1 \vee \ldots l_m \vee l'$ in F' such that there are already nodes $l_1^c, \ldots, l_m^c$
  - add node l' if not yet present
  - add edges $l_i^c \to l'$ for all $1 \leqslant i \leqslant m$ if not yet present
- if $\exists$ clause $l'_1 \vee \cdots \vee l'_k$ in F' such that there are nodes $l_1'^c, \ldots, l_k'^c$
  - add conflict node labeled C
  - add edges $l_i'^c \to C$

or former backjump clause

Lemma

if edges intersected by cut are $l_1 \to l'_1, \ldots, l_k \to l'_k$ then $F' \vDash l_1^c \vee l_k^c$

Resolution

Remarks

- keeping track of implication graph is too expensive in practice
- compute clauses associated with cuts by resolution instead

Resolution

Remarks

- keeping track of implication graph is too expensive in practice
- compute clauses associated with cuts by resolution instead

Definition (Resolution)

$$\frac{C \vee l \qquad C' \vee \neg l}{C \vee C'}$$

(assuming literals in clauses can be reordered)

Resolution

Remarks

- keeping track of implication graph is too expensive in practice
- compute clauses associated with cuts by resolution instead

Definition (Resolution)

$$\frac{C \vee l \qquad C' \vee \neg l}{C \vee C'}$$

(assuming literals in clauses can be reordered)

Example

$$\frac{6 \vee 11 \vee 13 \qquad \overline{12} \vee \overline{13}}{6 \vee 11 \vee \overline{12}}$$

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

Example

- $C_0 = 6 \vee 11 \vee 13$

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

Example

- $C_0 = 6 \vee 11 \vee 13$ $\qquad \dfrac{6 \vee 11 \vee 13 \qquad \overline{12} \vee \overline{13}}{6 \vee 11 \vee \overline{12}}$

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

Example

- $C_0 = 6 \vee 11 \vee 13$
- $C_1 = 6 \vee 11 \vee \overline{12}$

$$\frac{6 \vee 11 \vee 13 \qquad \overline{12} \vee \overline{13}}{6 \vee 11 \vee \overline{12}}$$

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

Example

- $C_0 = 6 \vee 11 \vee 13$
- $C_1 = 6 \vee 11 \vee \overline{12}$

$$\dfrac{\dfrac{6 \vee 11 \vee 13 \qquad \overline{12} \vee \overline{13}}{6 \vee 11 \vee \overline{12}} \qquad \overline{10} \vee 12}{6 \vee 11 \vee \overline{10}}$$

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

Example

- $C_0 = 6 \vee 11 \vee 13$
- $C_1 = 6 \vee 11 \vee \overline{12}$
- $C_2 = 6 \vee 11 \vee \overline{10}$

$$\cfrac{\cfrac{6 \vee 11 \vee 13 \qquad \overline{12} \vee \overline{13}}{6 \vee 11 \vee \overline{12}} \qquad \overline{10} \vee 12}{6 \vee 11 \vee \overline{10}}$$

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

Example

- $C_0 = 6 \vee 11 \vee 13$
- $C_1 = 6 \vee 11 \vee \overline{12}$
- $C_2 = 6 \vee 11 \vee \overline{10}$

$$\dfrac{\dfrac{\dfrac{6 \vee 11 \vee 13 \qquad \overline{12} \vee \overline{13}}{6 \vee 11 \vee \overline{12}} \qquad \overline{10} \vee 12}{6 \vee 11 \vee \overline{10}} \qquad \overline{10} \vee \overline{11}}{6 \vee \overline{10}}$$

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

Example

- $C_0 = 6 \vee 11 \vee 13$
- $C_1 = 6 \vee 11 \vee \overline{12}$
- $C_2 = 6 \vee 11 \vee \overline{10}$
- $C_3 = 6 \vee \overline{10}$

$$\dfrac{\dfrac{\dfrac{6 \vee 11 \vee 13 \qquad \overline{12} \vee \overline{13}}{6 \vee 11 \vee \overline{12}} \qquad \overline{10} \vee 12}{6 \vee 11 \vee \overline{10}} \qquad \overline{10} \vee \overline{1}1}{6 \vee \overline{10}}$$

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

Example

- $C_0 = 6 \vee 11 \vee 13$
- $C_1 = 6 \vee 11 \vee \overline{12}$
- $C_2 = 6 \vee 11 \vee \overline{10}$
- $C_3 = 6 \vee \overline{10}$

$$\frac{6 \vee 11 \vee 13 \qquad \overline{12} \vee \overline{13}}{6 \vee 11 \vee \overline{12}}$$

$$\frac{6 \vee 11 \vee \overline{12} \qquad \overline{10} \vee 12}{6 \vee 11 \vee \overline{10}}$$

$$\frac{6 \vee 11 \vee \overline{10} \qquad \overline{10} \vee \overline{11}}{6 \vee \overline{10}}$$

$$\frac{6 \vee \overline{10} \qquad \overline{7} \vee 8 \vee \overline{9} \vee 10}{6 \vee \overline{7} \vee 8 \vee \overline{9}}$$

How to Derive Backjump Clause by Resolution

- set C_0 to conflict clause
- let l be last assigned literal such that l^c is in C_0
- while l is no decision literal:
  - C_{i+1} is resolvent of C_i and clause D that led to assignment of l
  - let l be last assigned literal such that l^c is in C_{i+1}

Observation

every C_i corresponds to cut in implication graph

Example

- $C_0 = 6 \vee 11 \vee 13$
- $C_1 = 6 \vee 11 \vee \overline{12}$
- $C_2 = 6 \vee 11 \vee \overline{10}$
- $C_3 = 6 \vee \overline{10}$
- $C_4 = 6 \vee \overline{7} \vee 8 \vee \overline{9}$

$$\dfrac{\dfrac{\dfrac{\dfrac{6 \vee 11 \vee 13 \qquad \overline{12} \vee \overline{13}}{6 \vee 11 \vee \overline{12}} \qquad \overline{10} \vee 12}{6 \vee 11 \vee \overline{10}} \qquad \overline{10} \vee \overline{1}1}{6 \vee \overline{10}} \qquad \overline{7} \vee 8 \vee \overline{9} \vee 10}{6 \vee \overline{7} \vee 8 \vee \overline{9}}$$

Conflict Driven Clause Learning

Observations

- adding backjump clause to formula helps to prune search space (learning)
- if progress is too slow according to some measure, restart procedure:
  due to learned clauses heuristics will lead to different proof search

Observations

- adding backjump clause to formula helps to prune search space (learning)
- if progress is too slow according to some measure, restart procedure: due to learned clauses heuristics will lead to different proof search

Definition (DPLL with Learning and Restarts)

DPLL with learning and restarts $\mathcal{R}$ extends system $\mathcal{B}$ by following three rules:

Observations

- adding backjump clause to formula helps to prune search space (learning)
- if progress is too slow according to some measure, restart procedure: due to learned clauses heuristics will lead to different proof search

Definition (DPLL with Learning and Restarts)

DPLL with learning and restarts $\mathcal{R}$ extends system $\mathcal{B}$ by following three rules:

- learn $\quad M \parallel F \quad \Longrightarrow \quad M \parallel F,\ C$
  if $F \vDash C$ and all atoms of C occur in M or F

Observations

- adding backjump clause to formula helps to prune search space (learning)
- if progress is too slow according to some measure, restart procedure:
  due to learned clauses heuristics will lead to different proof search

Definition (DPLL with Learning and Restarts)

DPLL with learning and restarts $\mathcal{R}$ extends system $\mathcal{B}$ by following three rules:

- learn $\quad M \parallel F \quad \Longrightarrow \quad M \parallel F,\ C$
  if $F \vDash C$ and all atoms of C occur in M or F

- forget $\quad M \parallel F,\ C \quad \Longrightarrow \quad M \parallel F$
  if $F \vDash C$

Observations

- adding backjump clause to formula helps to prune search space (learning)
- if progress is too slow according to some measure, restart procedure: due to learned clauses heuristics will lead to different proof search

Definition (DPLL with Learning and Restarts)

DPLL with learning and restarts $\mathcal{R}$ extends system $\mathcal{B}$ by following three rules:

- learn $\quad M \parallel F \implies M \parallel F, C$
  if $F \vDash C$ and all atoms of C occur in M or F
- forget $\quad M \parallel F, C \implies M \parallel F$
  if $F \vDash C$
- restart $\quad M \parallel F \implies \parallel F$

Theorem (Termination)

any derivation $\| F \Longrightarrow_{\mathcal{R}} S_1 \Longrightarrow_{\mathcal{R}} S_2 \Longrightarrow_{\mathcal{R}} \ldots$ *is finite if*

- *it contains no infinite subderivation of learn and forget steps, and*

Theorem (Termination)

any derivation $\| F \Longrightarrow_{\mathcal{R}} S_1 \Longrightarrow_{\mathcal{R}} S_2 \Longrightarrow_{\mathcal{R}} \ldots$ *is finite if*

- *it contains no infinite subderivation of learn and forget steps, and*
- *restart is applied with increasing periodicity*

Theorem (Termination)

any derivation $\| F \Longrightarrow_{\mathcal{R}} S_1 \Longrightarrow_{\mathcal{R}} S_2 \Longrightarrow_{\mathcal{R}} \dots$ *is finite if*

- *it contains no infinite subderivation of learn and forget steps, and*
- *restart is applied with increasing periodicity*

Theorem (Correctness)

for derivation with final state S_n*:*

$$\| F \Longrightarrow_{\mathcal{R}} S_1 \Longrightarrow_{\mathcal{R}} S_2 \Longrightarrow_{\mathcal{R}} \dots \Longrightarrow_{\mathcal{R}} S_n$$

- *if* $S_n = \mathrm{FailState}$ *then* F *is unsatisfiable*

Theorem (Termination)

any derivation $\| F \Longrightarrow_{\mathcal{R}} S_1 \Longrightarrow_{\mathcal{R}} S_2 \Longrightarrow_{\mathcal{R}} \dots$ *is finite if*

- *it contains no infinite subderivation of learn and forget steps, and*
- *restart is applied with increasing periodicity*

Theorem (Correctness)

for derivation with final state S_n*:*

$$\| F \Longrightarrow_{\mathcal{R}} S_1 \Longrightarrow_{\mathcal{R}} S_2 \Longrightarrow_{\mathcal{R}} \dots \Longrightarrow_{\mathcal{R}} S_n$$

- *if* $S_n = \text{FailState}$ *then* F *is unsatisfiable*
- *if* $S_n = M \| F'$ *then* F *is satisfiable and* $M \models F$

Bit Vectors

Outline

- Summary of Last Week
- Conflict Analysis
- Conflict Driven Clause Learning
- Bit Vectors

Application: Collision Attacks using SAT

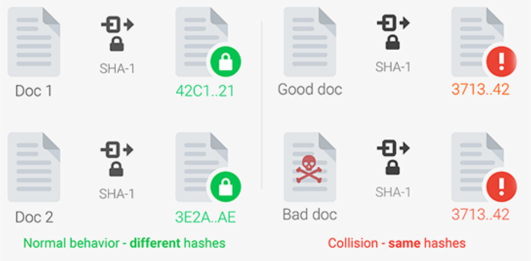
Cryptographic Hash Functions

- cryptographic hash function f is one-way hash function (SHA-1, MD5, ...)
- considered infeasible to invert, and to find messages with same hash

Application: Collision Attacks using SAT

Cryptographic Hash Functions

- cryptographic hash function f is one-way hash function (SHA-1, MD5, ...)
- considered infeasible to invert, and to find messages with same hash
- problem: hash collisions



Application: Collision Attacks using SAT

Cryptographic Hash Functions

- cryptographic hash function f is one-way hash function (SHA-1, MD5, ...)
- considered infeasible to invert, and to find messages with same hash
- problem: hash collisions

Collision Attack Scenario

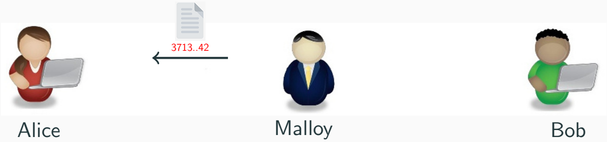
- Malloy aims to send malicious document to Bob pretending it be from Alice

Application: Collision Attacks using SAT

Cryptographic Hash Functions

- cryptographic hash function f is one-way hash function (SHA-1, MD5, ...)
- considered infeasible to invert, and to find messages with same hash
- problem: hash collisions

Collision Attack Scenario



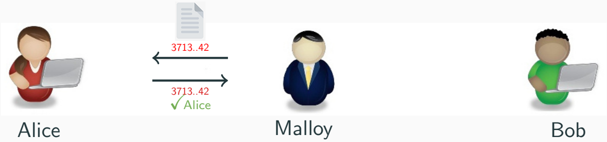
- Malloy aims to send malicious document to Bob pretending it be from Alice

Application: Collision Attacks using SAT

Cryptographic Hash Functions

- cryptographic hash function f is one-way hash function (SHA-1, MD5, ...)
- considered infeasible to invert, and to find messages with same hash
- problem: hash collisions

Collision Attack Scenario



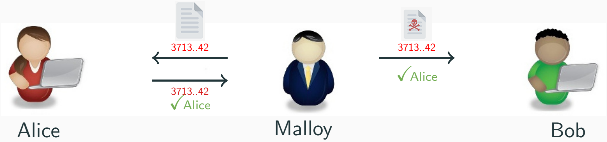
- Malloy aims to send malicious document to Bob pretending it be from Alice

Application: Collision Attacks using SAT

Cryptographic Hash Functions

- cryptographic hash function f is one-way hash function (SHA-1, MD5, ...)
- considered infeasible to invert, and to find messages with same hash
- problem: hash collisions

Collision Attack Scenario



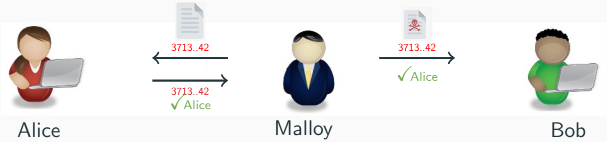
- Malloy aims to send malicious document to Bob pretending it be from Alice

Application: Collision Attacks using SAT

Cryptographic Hash Functions

- cryptographic hash function f is one-way hash function (SHA-1, MD5, ...)
- considered infeasible to invert, and to find messages with same hash
- problem: hash collisions

Collision Attack Scenario



- Malloy aims to send malicious document to Bob pretending it be from Alice

SAT-Based Collision Finding

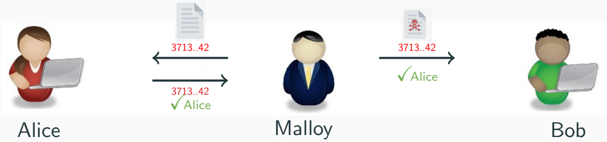
- encode f as operation on bit vectors x, y representing strings

Application: Collision Attacks using SAT

Cryptographic Hash Functions

- cryptographic hash function f is one-way hash function (SHA-1, MD5, ...)
- considered infeasible to invert, and to find messages with same hash
- problem: hash collisions

Collision Attack Scenario



- Malloy aims to send malicious document to Bob pretending it be from Alice

SAT-Based Collision Finding

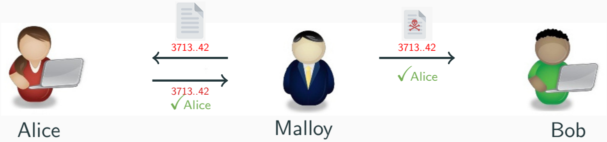
- encode f as operation on bit vectors x, y representing strings
- assert $x \neq y \wedge f(x) = f(y)$: if satisfiable obtain values for x and y

Application: Collision Attacks using SAT

Cryptographic Hash Functions

- cryptographic hash function f is one-way hash function (SHA-1, MD5, ...)
- considered infeasible to invert, and to find messages with same hash
- problem: hash collisions

Collision Attack Scenario



- Malloy aims to send malicious document to Bob pretending it be from Alice

SAT-Based Collision Finding

- encode f as operation on bit vectors x, y representing strings
- assert $x \neq y \wedge f(x) = f(y)$: if satisfiable obtain values for x and y
- collisions for (flawed) MD4, MD5 already found in 2005, using SAT

More Cryptanalysis using SAT

- collision attacks (preimage attacks) for current hash functions such as MD4, MD5, SHA-256, CryptoHash, Keccak, ...

More Cryptanalysis using SAT

- collision attacks (preimage attacks) for current hash functions such as MD4, MD5, SHA-256, CryptoHash, Keccak, ...
- exhibit classes of weak keys (or prove their absence)
  for block ciphers such as IDEA, WIDEA-n, or MESH-8

More Cryptanalysis using SAT

- collision attacks (preimage attacks) for current hash functions such as MD4, MD5, SHA-256, CryptoHash, Keccak, ...
- exhibit classes of weak keys (or prove their absence) for block ciphers such as IDEA, WIDEA-*n*, or MESH-8
- solve inversion problems, e.g. for 20 bit DES key

More Cryptanalysis using SAT

- collision attacks (preimage attacks) for current hash functions such as MD4, MD5, SHA-256, CryptoHash, Keccak, ...
- exhibit classes of weak keys (or prove their absence)
  for block ciphers such as IDEA, WIDEA-*n*, or MESH-8
- solve inversion problems, e.g. for 20 bit DES key
- reason about crypto primitives

More Cryptanalysis using SAT

- collision attacks (preimage attacks) for current hash functions such as MD4, MD5, SHA-256, CryptoHash, Keccak, ...
- exhibit classes of weak keys (or prove their absence) for block ciphers such as IDEA, WIDEA-*n*, or MESH-8
- solve inversion problems, e.g. for 20 bit DES key
- reason about crypto primitives
- help prove complexity bounds of certain operations

More Cryptanalysis using SAT

- collision attacks (preimage attacks) for current hash functions such as MD4, MD5, SHA-256, CryptoHash, Keccak, ...
- exhibit classes of weak keys (or prove their absence)
  for block ciphers such as IDEA, WIDEA-*n*, or MESH-8
- solve inversion problems, e.g. for 20 bit DES key
- reason about crypto primitives
- help prove complexity bounds of certain operations

Tools for SAT/SMT-Based Cryptanalysis

- CryptoMiniSat
- CryptoSMT
- Transalg
- ...

Bit Blasting

Representing Characters as Bit Vectors

- ASCII character a is representable as bit vector for `char` of 8 bits:
  using propositional variables $a_0, \ldots, a_6$ write a as

0	a_6	a_5	a_4	a_3	a_2	a_1	a_0

Bit Blasting

Representing Characters as Bit Vectors

- ASCII character a is representable as bit vector for `char` of 8 bits: using propositional variables $a_0, \ldots, a_6$ write a as

0	a_6	a_5	a_4	a_3	a_2	a_1	a_0

- a_0 is least significant bit, so e.g. $\texttt{C} = \texttt{ascii}(67)$ is

0	1	0	0	0	0	1	1

Bit Blasting

Representing Characters as Bit Vectors

- ASCII character a is representable as bit vector for `char` of 8 bits: using propositional variables $a_0, \ldots, a_6$ write a as

0	a_6	a_5	a_4	a_3	a_2	a_1	a_0

- a_0 is least significant bit, so e.g. $\texttt{C} = \texttt{ascii}(67)$ is

0	1	0	0	0	0	1	1

Ingredients of (Cryptographic) Hash Functions

- hash functions operate on character strings

Bit Blasting

Representing Characters as Bit Vectors

- ASCII character a is representable as bit vector for `char` of 8 bits: using propositional variables $a_0, \ldots, a_6$ write a as

0	a_6	a_5	a_4	a_3	a_2	a_1	a_0

- a_0 is least significant bit, so e.g. $\texttt{C} = \texttt{ascii}(67)$ is

0	1	0	0	0	0	1	1

Ingredients of (Cryptographic) Hash Functions

- hash functions operate on character strings
- string of length n representable as bit vector of n `chars` ($8n$ bits)

Bit Blasting

Representing Characters as Bit Vectors

- ASCII character *a* is representable as bit vector for `char` of 8 bits: using propositional variables $a_0, \ldots, a_6$ write *a* as

0	a_6	a_5	a_4	a_3	a_2	a_1	a_0

- a_0 is least significant bit, so e.g. $\mathtt{C} = \mathtt{ascii}(67)$ is

0	1	0	0	0	0	1	1

Ingredients of (Cryptographic) Hash Functions

- hash functions operate on character strings
- string of length *n* representable as bit vector of *n* `chars` ($8n$ bits)
- common operations in hash functions are
  - bitwise XOR $\oplus$
  - addition $+$
  - shifts

Bit Blasting

Representing Characters as Bit Vectors

- ASCII character a is representable as bit vector for `char` of 8 bits: using propositional variables $a_0, \ldots, a_6$ write a as

0	a_6	a_5	a_4	a_3	a_2	a_1	a_0

- a_0 is least significant bit, so e.g. $\texttt{C} = \texttt{ascii}(67)$ is

0	1	0	0	0	0	1	1

Ingredients of (Cryptographic) Hash Functions

- hash functions operate on character strings
- string of length n representable as bit vector of n `chars` ($8n$ bits)
- common operations in hash functions are
  - bitwise XOR $\oplus$
  - addition $+$
  - shifts

Example (Rotation Hash)

```
def rotation_hash(s):
  h = 0
  for i in range(0,len(s)):
    h = (h << 4) ^ (h >> 28) ^ ord(s[i])
  return h
```

Z3

common open source SAT/SMT solver

Z3

common open source SAT/SMT solver

`z3`: Python interface to Z3

- from `https://github.com/Z3Prover/z3` or via `pip install z3`
- API: `https://z3prover.github.io/api/html/namespacez3py.html`

Z3

common open source SAT/SMT solver

`z3`: Python interface to Z3

- from `https://github.com/Z3Prover/z3` or via `pip install z3`
- API: `https://z3prover.github.io/api/html/namespacez3py.html`

Building Formulas

- `True`, `False`   boolean constants

Z3

common open source SAT/SMT solver

`z3`: Python interface to Z3

- from `https://github.com/Z3Prover/z3` or via `pip install z3`
- API: `https://z3prover.github.io/api/html/namespacez3py.html`

Building Formulas

- `True`, `False` boolean constants
- `Bool(`*name*`)` propositional variable named *name*
  (calling `Bool(`*name*`)` twice yields same variable)

Z3

common open source SAT/SMT solver

`z3`: Python interface to Z3

- from `https://github.com/Z3Prover/z3` or via `pip install z3`
- API: `https://z3prover.github.io/api/html/namespacez3py.html`

Building Formulas

- `True`, `False` boolean constants
- `Bool(`*name*`)` propositional variable named *name*
  (calling `Bool(`*name*`)` twice yields same variable)
- `FreshBool(`*pre*`)` new propositional variable with name prefix *name*
  (calling `FreshBool(`*pre*`)` twice does not yield same variable!)

Z3

common open source SAT/SMT solver

`z3`: Python interface to Z3

- from `https://github.com/Z3Prover/z3` or via `pip install z3`
- API: `https://z3prover.github.io/api/html/namespacez3py.html`

Building Formulas

- `True`, `False` boolean constants
- `Bool(`*name*`)` propositional variable named *name*
  (calling `Bool(`*name*`)` twice yields same variable)
- `FreshBool(`*pre*`)` new propositional variable with name prefix *name*
  (calling `FreshBool(`*pre*`)` twice does not yield same variable!)
- `And(`$a_1, \dots, a_n$`)` conjunction with arbitrarily many arguments

Z3

common open source SAT/SMT solver

`z3`: Python interface to Z3

- from `https://github.com/Z3Prover/z3` or via `pip install z3`
- API: `https://z3prover.github.io/api/html/namespacez3py.html`

Building Formulas

- `True`, `False` boolean constants
- `Bool(`*name*`)` propositional variable named *name*
  (calling `Bool(`*name*`)` twice yields same variable)
- `FreshBool(`*pre*`)` new propositional variable with name prefix *name*
  (calling `FreshBool(`*pre*`)` twice does not yield same variable!)
- `And(`$a_1, \ldots, a_n$`)` conjunction with arbitrarily many arguments
- `Or(`$a_1, \ldots, a_n$`)` disjunction with arbitrarily many arguments

Z3

common open source SAT/SMT solver

`z3`: Python interface to Z3

- from `https://github.com/Z3Prover/z3` or via `pip install z3`
- API: `https://z3prover.github.io/api/html/namespacez3py.html`

Building Formulas

- `True`, `False` — boolean constants
- `Bool(`*name*`)` — propositional variable named *name*
  (calling `Bool(`*name*`)` twice yields same variable)
- `FreshBool(`*pre*`)` — new propositional variable with name prefix *name*
  (calling `FreshBool(`*pre*`)` twice does not yield same variable!)
- `And(`$a_1, \ldots, a_n$`)` — conjunction with arbitrarily many arguments
- `Or(`$a_1, \ldots, a_n$`)` — disjunction with arbitrarily many arguments
- `Not(`a`)` — negation

Z3

common open source SAT/SMT solver

z3: Python interface to Z3

- from `https://github.com/Z3Prover/z3` or via `pip install z3`
- API: `https://z3prover.github.io/api/html/namespacez3py.html`

Building Formulas

- `True`, `False` boolean constants
- `Bool(`*name*`)` propositional variable named *name* (calling `Bool(`*name*`)` twice yields same variable)
- `FreshBool(`*pre*`)` new propositional variable with name prefix *name* (calling `FreshBool(`*pre*`)` twice does not yield same variable!)
- `And(`$a_1, \dots, a_n$`)` conjunction with arbitrarily many arguments
- `Or(`$a_1, \dots, a_n$`)` disjunction with arbitrarily many arguments
- `Not(`a`)` negation
- `Implies(`a, b`)` implication

Z3

common open source SAT/SMT solver

z3: Python interface to Z3

- from https://github.com/Z3Prover/z3 or via pip install z3
- API: https://z3prover.github.io/api/html/namespacez3py.html

Building Formulas

- `True`, `False` boolean constants
- `Bool(`*name*`)` propositional variable named *name* (calling `Bool(`*name*`)` twice yields same variable)
- `FreshBool(`*pre*`)` new propositional variable with name prefix *name* (calling `FreshBool(`*pre*`)` twice does not yield same variable!)
- `And(`$a_1, \ldots, a_n$`)` conjunction with arbitrarily many arguments
- `Or(`$a_1, \ldots, a_n$`)` disjunction with arbitrarily many arguments
- `Not(`a`)` negation
- `Implies(`a, b`)` implication
- `Xor(`a, b`)` exclusive or

Solving Formulas

- `Solver()` create new solver object

Solving Formulas

- `Solver()` create new solver object
- `Solver.add(`$\varphi_1, \ldots, \varphi_n$`)` require constraints $\varphi_1, \ldots, \varphi_n$ to be true

Solving Formulas

- `Solver()` create new solver object
- `Solver.add(`$\varphi_1, \ldots, \varphi_n$`)` require constraints $\varphi_1, \ldots, \varphi_n$ to be true
- `Solver.check()` check for satisfiability

Solving Formulas

- `Solver()` create new solver object
- `Solver.add(`$\varphi_1, \ldots, \varphi_n$`)` require constraints $\varphi_1, \ldots, \varphi_n$ to be true
- `Solver.check()` check for satisfiability
- `Solver.model()` returns valuation (after successful call of `check`)

Solving Formulas

- `Solver()` create new solver object
- `Solver.add(`$\varphi_1, \ldots, \varphi_n$`)` require constraints $\varphi_1, \ldots, \varphi_n$ to be true
- `Solver.check()` check for satisfiability
- `Solver.model()` returns valuation (after successful call of `check`)

Moreover ...

- `simplify(`φ`)` simplifies formula φ

Solving Formulas

- `Solver()` create new solver object
- `Solver.add(`$\varphi_1, \ldots, \varphi_n$`)` require constraints $\varphi_1, \ldots, \varphi_n$ to be true
- `Solver.check()` check for satisfiability
- `Solver.model()` returns valuation (after successful call of `check`)

Moreover ...

- `simplify(`φ`)` simplifies formula φ
- `Solver.statistics()` is map of solving statistics

Example

```
from z3 import *
p = Bool('p') # create variable named 'p'
foo1 = FreshBool('foo') # create new variables prefixed 'foo'
foo2 = FreshBool('foo')

phi = Or(p, p, And(foo2, Xor(foo1, Not(foo1)), True), False)
print(phi) # Or(p, p, And(foo!1, Xor(foo!0, Not(foo!0)), True), False)
psi = simplify(phi)
print(psi) # Or(p, foo!1)

solver = Solver()
solver.add(psi) # assert that psi should be true
solver.add(Implies(foo1,p), Or(foo1, foo2)) # assert something else

print solver # [Or(p, foo!1), Implies(foo!0, p), Or(foo!0, foo!1)]
result = solver.check() # check for satisfiability

if result:
  model = solver.model() # get valuation
  print model[p], model[foo1], model[foo2] # False False True
```