Programm- & Systemverifikation

Satisfiability Checking

Georg Weissenbacher

(some slides from Josef Widder)

184.741

What happened so far

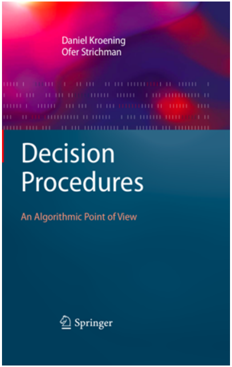
- How bugs come into being:
  - Fault – cause of an error (e.g., mistake in coding)
  - Error – *incorrect* state that may lead to failure
  - Failure – deviation from *desired* behaviour
- We specified *intended* behaviour using assertions
- We proved our programs correct (inductive invariants).
- We learned how to test programs.
- We heard about logical formalisms:
  - Propositional Logic
  - First Order Logic
  - Hoare Logic
- Last time we learned about Bounded Model Checking.

Literature

"*Decision Procedures*"

An Algorithmic Point of View

Daniel Kröning, Ofer Strichman



- Chapter 2.2: SAT Solvers
- Available in "Hauptbibliothek"

Propositional Logic:

$$
\begin{array}{rcl}
\textit{formula} & ::= & \textit{formula} \wedge \textit{formula} \mid \textit{formula} \vee \textit{formula} \mid \\
 & & \neg \textit{formula} \mid (\textit{formula}) \mid \textit{atom} \\
\textit{atom} & ::= & \textit{propositional identifier} \mid \textit{constant} \\
\textit{constant} & ::= & \text{true} \mid \text{false}
\end{array}
$$

- Goal:
  - Find satisfying assignment or
  - show unsatisfiability
- Soundness: Decision Procedure gives correct answer
- Completeness: Decision Procedure always finds an answer

Warm-up: SAT or UNSAT?

$(a \lor b) \land \neg a \land \neg b$

Warm-up: SAT or UNSAT?

$(a \vee b) \wedge \neg a \wedge \neg b$ UNSAT

Warm-up: SAT or UNSAT?

$(a \lor b) \land \neg a \land \neg b$ UNSAT

$(a \lor \neg b) \lor (\neg a \land \neg b)$

Warm-up: SAT or UNSAT?

$(a \vee b) \wedge \neg a \wedge \neg b$ UNSAT

$(a \vee \neg b) \vee (\neg a \wedge \neg b)$ SAT

Warm-up: SAT or UNSAT?

$(a \vee b) \wedge \neg a \wedge \neg b$

UNSAT

$(a \vee \neg b) \vee (\neg a \wedge \neg b)$

SAT

$b =$ false

Warm-up: SAT or UNSAT?

$(a \lor b) \land \neg a \land \neg b$ UNSAT

$(a \lor \neg b) \lor (\neg a \land \neg b)$ SAT
$b =$ false

$(a \lor b) \land c \land (\neg a \lor b \lor \neg c) \land (a \lor c) \land (\neg b \lor c)$

Warm-up: SAT or UNSAT?

$(a \vee b) \wedge \neg a \wedge \neg b$	UNSAT
$(a \vee \neg b) \vee (\neg a \wedge \neg b)$	SAT $b = \text{false}$
$(a \vee b) \wedge c \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee c) \wedge (\neg b \vee c)$	SAT

Warm-up: SAT or UNSAT?

$(a \lor b) \land \neg a \land \neg b$ UNSAT

$(a \lor \neg b) \lor (\neg a \land \neg b)$ SAT
$b = \text{false}$

$(a \lor b) \land c \land (\neg a \lor b \lor \neg c) \land (a \lor c) \land (\neg b \lor c)$ SAT
$b = c = \text{true}$

Warm-up: SAT or UNSAT?

$(a \vee b) \wedge \neg a \wedge \neg b$ UNSAT

$(a \vee \neg b) \vee (\neg a \wedge \neg b)$ SAT
$b = \text{false}$

$(a \vee b) \wedge c \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee c) \wedge (\neg b \vee c)$ SAT
$b = c = \text{true}$

$(a \vee b) \wedge c \wedge (a \vee b \vee c) \wedge (\neg a \vee \neg c) \wedge (\neg b \vee \neg c)$

Warm-up: SAT or UNSAT?

$(a \lor b) \land \neg a \land \neg b$ UNSAT

$(a \lor \neg b) \lor (\neg a \land \neg b)$ SAT
$b = \text{false}$

$(a \lor b) \land c \land (\neg a \lor b \lor \neg c) \land (a \lor c) \land (\neg b \lor c)$ SAT
$b = c = \text{true}$

$(a \lor b) \land c \land (a \lor b \lor c) \land (\neg a \lor \neg c) \land (\neg b \lor \neg c)$ UNSAT

Warm-up: SAT or UNSAT?

$(a \vee b) \wedge \neg a \wedge \neg b$ UNSAT

$(a \vee \neg b) \vee (\neg a \wedge \neg b)$ SAT

$b = \text{false}$

$(a \vee b) \wedge c \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee c) \wedge (\neg b \vee c)$ SAT

$b = c = \text{true}$

$(a \vee b) \wedge c \wedge (a \vee b \vee c) \wedge (\neg a \vee \neg c) \wedge (\neg b \vee \neg c)$ UNSAT

if $c = \text{true}$, then $a = \text{false}$, $b = \text{false}$,
then 1st clause false

Warm-up: SAT or UNSAT?

$(a \vee b) \wedge \neg a \wedge \neg b$ UNSAT

$(a \vee \neg b) \vee (\neg a \wedge \neg b)$ SAT
$b = \text{false}$

$(a \vee b) \wedge c \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee c) \wedge (\neg b \vee c)$ SAT
$b = c = \text{true}$

$(a \vee b) \wedge c \wedge (a \vee b \vee c) \wedge (\neg a \vee \neg c) \wedge (\neg b \vee \neg c)$ UNSAT
if $c = \text{true}$, then $a = \text{false}$, $b = \text{false}$,
then 1st clause false

$(d \vee \neg b) \wedge (\neg a \vee \neg c) \wedge (\neg d \vee e) \wedge (a \vee b) \wedge (e \vee c \vee \neg b) \wedge c \wedge (\neg b \vee \neg c) \wedge (a \vee b \vee c)$

Warm-up: SAT or UNSAT?

$(a \vee b) \wedge \neg a \wedge \neg b$ UNSAT

$(a \vee \neg b) \vee (\neg a \wedge \neg b)$ SAT

$b = \text{false}$

$(a \vee b) \wedge c \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee c) \wedge (\neg b \vee c)$ SAT

$b = c = \text{true}$

$(a \vee b) \wedge c \wedge (a \vee b \vee c) \wedge (\neg a \vee \neg c) \wedge (\neg b \vee \neg c)$ UNSAT

if $c = \text{true}$, then $a = \text{false}$, $b = \text{false}$,
then 1st clause false

$(d \vee \neg b) \wedge (\neg a \vee \neg c) \wedge (\neg d \vee e) \wedge (a \vee b) \wedge (e \vee c \vee \neg b) \wedge c \wedge (\neg b \vee \neg c) \wedge (a \vee b \vee c)$

UNSAT

Warm-up: SAT or UNSAT?

$(a \vee b) \wedge \neg a \wedge \neg b$ UNSAT

$(a \vee \neg b) \vee (\neg a \wedge \neg b)$ SAT
$b =$ false

$(a \vee b) \wedge c \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee c) \wedge (\neg b \vee c)$ SAT
$b = c =$ true

$(a \vee b) \wedge c \wedge (a \vee b \vee c) \wedge (\neg a \vee \neg c) \wedge (\neg b \vee \neg c)$ UNSAT
if $c =$ true, then $a =$ false, $b =$ false,
then 1st clause false

$(d \vee \neg b) \wedge (\neg a \vee \neg c) \wedge (\neg d \vee e) \wedge (a \vee b) \wedge (e \vee c \vee \neg b) \wedge c \wedge (\neg b \vee \neg c) \wedge (a \vee b \vee c)$
UNSAT ... contains all clauses of previous formula

Motivation: Why Do We Need SAT Solvers?

- SAT is the canonical NP-complete problem
  - Other problems in NP can be reduced to SAT
  - Unless $P = NP$, there is no efficient solution

Motivation: Applications of SAT Solvers in Verification

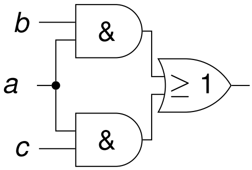
- Test Case Generation
- Bounded Model Checking
- Equivalence Checking

Checking Equivalence of Circuits

Show that, e.g., optimized circuit has the same functionality

Checking Equivalence of Circuits

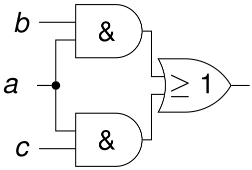
Show that, e.g., optimized circuit has the same functionality



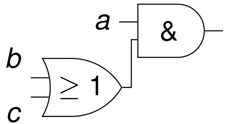
$$C_1 \equiv (a \wedge b) \vee (a \wedge c)$$

Checking Equivalence of Circuits

Show that, e.g., optimized circuit has the same functionality



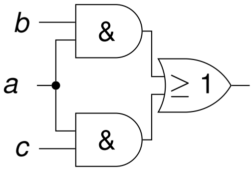
$C_1 \equiv (a \wedge b) \vee (a \wedge c)$

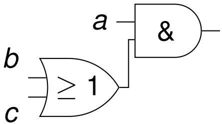


$C_2 \equiv a \wedge (b \vee c)$

Checking Equivalence of Circuits

Show that, e.g., optimized circuit has the same functionality





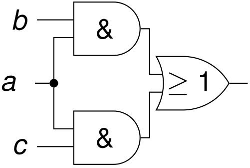
$$C_1 \equiv (a \wedge b) \vee (a \wedge c)$$

$$C_2 \equiv a \wedge (b \vee c)$$

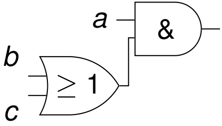
Are C_1 and C_2 functionally equivalent?

Checking Equivalence of Circuits

Show that, e.g., optimized circuit has the same functionality



$C_1 \equiv (a \wedge b) \vee (a \wedge c)$



$C_2 \equiv a \wedge (b \vee c)$

Are C_1 and C_2 functionally equivalent?

Is $\neg(C_1 \Leftrightarrow C2)$ unsatisfiable?
There is no assignment to a, b, and c, such that C_1 and C_2 yield different truth values.

Test Case/Pattern Generation

- Inject bug in software (mutation) or fault in hardware
  - e.g., variable or wire replaced with a constant
- Yields two versions of encoding of software/hardware:
  - Original program/circuit C
  - Faulty program/circuit C'
- If $(C \oplus C')$ is satisfiable
  - satisfying assignment corresponds to test inputs
  - test inputs *reveal* fault (if present in program/circuit)

Conjunctive Normal Form

- CNF formula: A conjunction of clauses (product of sums)

$$\bigwedge_i \bigvee_j \ell_{i,j}\,, \qquad \ell_{i,j} \in \{a, \neg a \mid a \in \text{Variables}\}$$

  e.g.,

$$\neg a_1 \wedge (a_1 \vee \neg a_2) \wedge (\neg a_1 \vee a_2) \wedge a_1$$

- Remember:
  - $\bigvee_{\ell \in \emptyset} \ell \;\equiv\;$ false (we use $\square$ to denote the empty clause)
- Alternative (more compact) notation:

$$(\overline{a}_1)\;(a_1\,\overline{a}_2)\;(\overline{a}_1\,a_2)\;(a_1)$$

- Obtained through Tseitin transformation (see lecture on logic)

Tseitin's Algorithm Revisited: Example

$$(a \Rightarrow (c \wedge d)) \vee (b \Rightarrow (c \wedge e))$$

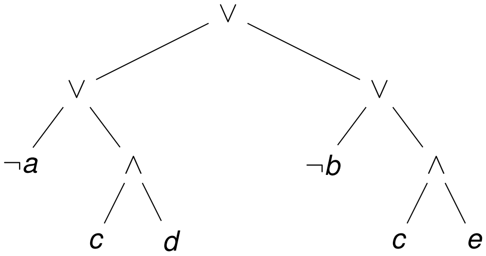
Tseitin's Algorithm Revisited: Example

$$(a \Rightarrow (c \wedge d)) \vee (b \Rightarrow (c \wedge e))$$

$$(\neg a \vee (c \wedge d)) \vee (\neg b \vee (c \wedge e))$$

$$(a \Rightarrow (c \wedge d)) \vee (b \Rightarrow (c \wedge e))$$

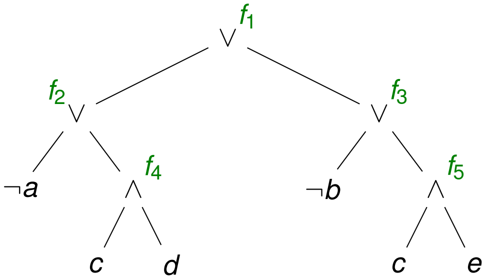
$$(\neg a \vee (c \wedge d)) \vee (\neg b \vee (c \wedge e))$$



Tseitin's Algorithm Revisited: Example

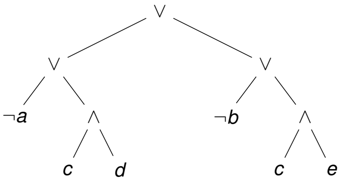
$$(a \Rightarrow (c \wedge d)) \vee (b \Rightarrow (c \wedge e))$$

$$(\neg a \vee (c \wedge d)) \vee (\neg b \vee (c \wedge e))$$



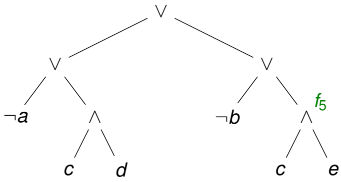
Example for Tseitin continued

$$(\neg a \vee (c \wedge d)) \vee (\neg b \vee (c \wedge e))$$



Example for Tseitin continued

$$(\neg a \vee (c \wedge d)) \vee (\neg b \vee (c \wedge e))$$

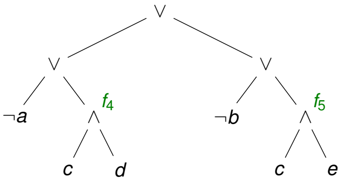


$$\wedge$$

$$(\neg f_5 \vee c) \wedge (\neg f_5 \vee e) \wedge (\neg c \vee \neg e \vee f_5)$$

Example for Tseitin continued

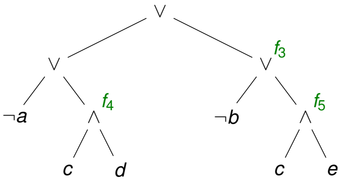
$$(\neg a \vee (c \wedge d)) \vee (\neg b \vee (c \wedge e))$$



$$\begin{gathered}\wedge \\ (\neg f_4 \vee c) \wedge (\neg f_4 \vee d) \wedge (\neg c \vee \neg d \vee f_4) \\ \wedge \\ (\neg f_5 \vee c) \wedge (\neg f_5 \vee e) \wedge (\neg c \vee \neg e \vee f_5)\end{gathered}$$

Example for Tseitin continued

$$(\neg a \vee (c \wedge d)) \vee (\neg b \vee (c \wedge e))$$



$$\wedge$$

$$(f_3 \vee b) \wedge (f_3 \vee \neg f_5) \wedge (\neg b \vee f_5 \vee \neg f_3)$$

$$\wedge$$

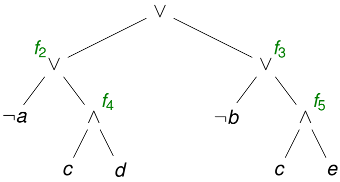
$$(\neg f_4 \vee c) \wedge (\neg f_4 \vee d) \wedge (\neg c \vee \neg d \vee f_4)$$

$$\wedge$$

$$(\neg f_5 \vee c) \wedge (\neg f_5 \vee e) \wedge (\neg c \vee \neg e \vee f_5)$$

Example for Tseitin continued

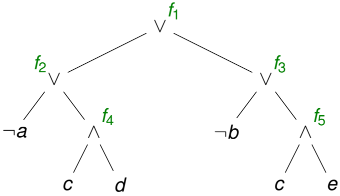
$$(\neg a \vee (c \wedge d)) \vee (\neg b \vee (c \wedge e))$$



$$
\begin{array}{c}
\wedge \\
(f_2 \vee a) \wedge (f_2 \vee \neg f_4) \wedge (\neg a \vee f_4 \vee \neg f_2) \\
\wedge \\
(f_3 \vee b) \wedge (f_3 \vee \neg f_5) \wedge (\neg b \vee f_5 \vee \neg f_3) \\
\wedge \\
(\neg f_4 \vee c) \wedge (\neg f_4 \vee d) \wedge (\neg c \vee \neg d \vee f_4) \\
\wedge \\
(\neg f_5 \vee c) \wedge (\neg f_5 \vee e) \wedge (\neg c \vee \neg e \vee f_5)
\end{array}
$$

Example for Tseitin continued

$$(\neg a \vee (c \wedge d)) \vee (\neg b \vee (c \wedge e))$$



$$f_1$$
$$\wedge$$
$$(f_1 \vee \neg f_2) \wedge (f_1 \vee \neg f_3) \wedge (f_2 \vee f_3 \vee \neg f_1)$$
$$\wedge$$
$$(f_2 \vee a) \wedge (f_2 \vee \neg f_4) \wedge (\neg a \vee f_4 \vee \neg f_2)$$
$$\wedge$$
$$(f_3 \vee b) \wedge (f_3 \vee \neg f_5) \wedge (\neg b \vee f_5 \vee \neg f_3)$$
$$\wedge$$
$$(\neg f_4 \vee c) \wedge (\neg f_4 \vee d) \wedge (\neg c \vee \neg d \vee f_4)$$
$$\wedge$$
$$(\neg f_5 \vee c) \wedge (\neg f_5 \vee e) \wedge (\neg c \vee \neg e \vee f_5)$$

Quantifier Expansion

- Is there a *satisfying* assignment?
  - Naïve algorithm for n variables: $O(2^n)$
- Let's look at a single variable $\mathtt{y}$ first:

$$(\mathtt{x} \vee \mathtt{y}) \;\wedge\; (\mathtt{z} \vee \neg \mathtt{y})$$

Quantifier Expansion

- Is there a *satisfying* assignment?
  - Naïve algorithm for n variables: $O(2^n)$
- Let's look at a single variable $\mathtt{y}$ first:

$$\exists \mathtt{y}\,.\,(\mathtt{x} \vee \mathtt{y}) \;\wedge\; (\mathtt{z} \vee \neg \mathtt{y})$$

Quantifier Expansion

- Is there a *satisfying* assignment?
  - Naïve algorithm for n variables: $O(2^n)$
- Let's look at a single variable $\mathtt{y}$ first:

$$
\begin{aligned}
&\exists \mathtt{y} \,.\, (\mathtt{x} \lor \mathtt{y}) \;\land\; (\mathtt{z} \lor \neg \mathtt{y}) \\
\equiv &((\mathtt{x} \lor \mathtt{y}) \;\land\; (\mathtt{z} \lor \neg \mathtt{y}))[\mathtt{y}/1] \lor ((\mathtt{x} \lor \mathtt{y}) \;\land\; (\mathtt{z} \lor \neg \mathtt{y}))[\mathtt{y}/0]
\end{aligned}
$$

Quantifier Expansion

- Is there a *satisfying* assignment?
  - Naïve algorithm for n variables: $O(2^n)$
- Let's look at a single variable $\mathtt{y}$ first:

$$
\begin{aligned}
& \exists \mathtt{y} \,.\, (\mathtt{x} \lor \mathtt{y}) \;\land\; (\mathtt{z} \lor \neg \mathtt{y}) \\
\equiv & ((\mathtt{x} \lor \mathtt{y}) \;\land\; (\mathtt{z} \lor \neg \mathtt{y}))[\mathtt{y}/1] \lor ((\mathtt{x} \lor \mathtt{y}) \;\land\; (\mathtt{z} \lor \neg \mathtt{y}))[\mathtt{y}/0] \\
\equiv & (\underbrace{(\mathtt{x} \lor 1)}_{1} \;\land\; \underbrace{(\mathtt{z} \lor \neg 1)}_{\mathtt{z}}) \lor (\underbrace{(\mathtt{x} \lor 0)}_{\mathtt{x}} \;\land\; \underbrace{(\mathtt{z} \lor \neg 0)}_{1})
\end{aligned}
$$

Quantifier Expansion

- Is there a *satisfying* assignment?
  - Naïve algorithm for n variables: $O(2^n)$
- Let's look at a single variable $\mathtt{y}$ first:

$$
\begin{aligned}
&\exists \mathtt{y}\,.\,(\mathtt{x} \vee \mathtt{y}) \;\wedge\; (\mathtt{z} \vee \neg\mathtt{y}) \\
\equiv&((\mathtt{x} \vee \mathtt{y}) \;\wedge\; (\mathtt{z} \vee \neg\mathtt{y}))[\mathtt{y}/1] \vee ((\mathtt{x} \vee \mathtt{y}) \;\wedge\; (\mathtt{z} \vee \neg\mathtt{y}))[\mathtt{y}/0] \\
\equiv&(\underbrace{(\mathtt{x} \vee 1)}_{1} \;\wedge\; \underbrace{(\mathtt{z} \vee \neg 1)}_{\mathtt{z}}) \vee (\underbrace{(\mathtt{x} \vee 0)}_{\mathtt{x}} \;\wedge\; \underbrace{(\mathtt{z} \vee \neg 0)}_{1}) \\
\equiv&(\mathtt{x} \vee \mathtt{z})
\end{aligned}
$$

- Let C, D be clauses (disjunctions of literals)

$$\frac{(C \vee a) \qquad (D \vee \overline{a})}{C \vee D} \quad [\text{Res}]$$

- For instance:

$$(\overline{a}_1)\ (a_1\ \overline{a}_2)\ (\overline{a}_1\ a_2)\ (a_1)$$

$$\begin{tikzcd}
& & a_1\overline{a}_2 \arrow[ddr] & & \overline{a}_1 a_2 \arrow[dr] & & a_1 \arrow[dl] \\
& & & & & a_2 \arrow[dll] & \\
& \overline{a}_1 \arrow[dr] & & a_1 \arrow[dl] & & & \\
& & \square & & & &
\end{tikzcd}$$

Unit Resolution

- In particular:

$$\frac{(C \vee a) \qquad (\overline{a})}{C} \quad \text{[Res]}$$

- "Unit Clause Rule"
- Example revisited:

$$(\overline{a}_1)\ (a_1\ \overline{a}_2)\ (\overline{a}_1\ a_2)\ (a_1)$$

$$\frac{\overline{a}_1 a_2 \qquad a_1}{a_2}$$

Unit Resolution

- In particular:

$$\frac{(C \vee a) \qquad (\overline{a})}{C} \quad [\text{Res}]$$

- "Unit Clause Rule"
- Example revisited:

$$(\overline{a}_1)\ (a_1\ \overline{a}_2)\ (\overline{a}_1\ a_2)\ (a_1)$$

$$\begin{tikzcd}
a_1\overline{a}_2 \arrow[ddr] & \overline{a}_1 a_2 \arrow[dr] & & a_1 \arrow[dl] \\
 & & a_2 \arrow[dl] & \\
 & a_1 & &
\end{tikzcd}$$

Unit Resolution

- In particular:

$$\frac{(C \vee a) \qquad (\overline{a})}{C}\ \text{[Res]}$$

- "Unit Clause Rule"
- Example revisited:

$$(\overline{a}_1)\ (a_1\ \overline{a}_2)\ (\overline{a}_1\ a_2)\ (a_1)$$

$$\begin{tikzcd}[column sep=small]
 & a_1\overline{a}_2 \arrow[ddr] & & \overline{a}_1 a_2 \arrow[dr] & & a_1 \arrow[dl] \\
 & & & & a_2 \arrow[dll] & \\
\overline{a}_1 \arrow[dr] & & a_1 \arrow[dl] & & & \\
 & \square & & & &
\end{tikzcd}$$

- In particular:

$$\frac{(C \vee a) \qquad (\overline{a})}{C}\ [\text{Res}]$$

- "Unit Clause Rule"
- Example revisited:

$$(\overline{a}_1)\ (a_1\ \overline{a}_2)\ (\overline{a}_1\ a_2)\ (a_1)$$

$$\begin{tikzcd}
& a_1\overline{a}_2 \arrow[ddrr] & & \overline{a}_1 a_2 \arrow[dr] & & a_1 \arrow[dl] \\
& & & & a_2 \arrow[dl] & \\
& \overline{a}_1 \arrow[dr] & & a_1 \arrow[dl] & & \\
& & \square & & &
\end{tikzcd}$$

- Unit clause *propagation*: Efficient

Decision Making

- What if there are no unit clauses?
  - Progress by making decisions about variables:

$$(a_1\ \overline{a}_2)$$
$$\{a_1 \mapsto 1,\ \ldots\}$$

Decision Making

- What if there are no unit clauses?
  - Progress by making decisions about variables:

$$(a_1\ \overline{a}_2)$$
$$\{a_1 \mapsto 1,\ \ldots\}$$

- Partial assignment: Not all variables assigned

$$\{x_1 \mapsto 1,\ x_2 \mapsto 0,\ x_4 \mapsto 1\}$$

  - $(x_1 \vee x_3 \vee \neg x_4)$ is *satisfied*
    One or more literal satisfied (clause can be ignored)
  - $(\neg x_1 \vee x_2)$ is *conflicting*:
    All literals assigned but not satisfied
  - $(\neg x_1 \vee \neg x_4 \vee x_3)$ is *unit*:
    All but one literal assigned, but not satisfied
  - $(\neg x_1 \vee x_3 \vee x_5)$ is *unresolved*

Decision Levels

- Decision may result in *unit clauses*

$$\{\mathtt{x}_1 \mapsto 1,\ \mathtt{x}_4 \mapsto 1\}$$
$$(\neg \mathtt{x}_1 \vee \neg \mathtt{x}_4 \vee \mathtt{x}_3)$$

- Results in unit clause:
  - $\{\mathtt{x}_1 \mapsto 1,\ \mathtt{x}_4 \mapsto 1\}$ AND $(\neg \mathtt{x}_1 \vee \neg \mathtt{x}_4 \vee \mathtt{x}_3)$ implies $\mathtt{x}_3$
  - Antecedent of $\mathtt{x}_3$ is $(\neg \mathtt{x}_1 \vee \neg \mathtt{x}_4 \vee \mathtt{x}_3)$
  - Leads to unit propagation!
- Each decision is associated with a *decision level*

$$\{\underbrace{\mathtt{x}_1 \mapsto 1}_{1},\ \underbrace{\mathtt{x}_4 \mapsto 1}_{2},\ \ldots\}$$

- Implications of a decision associated with same decision level:
  - $\mathtt{x}_4$ and $\mathtt{x}_3$ above have decision level 2, denoted by $\neg \mathtt{x}_4 @ 2$ and $\mathtt{x}_3 @ 2$

Decision Levels (continued)

dl	**Assignment**	**Clauses**
0	–	$(\overline{x}_1\ \overline{x}_4\ x_3)(\overline{x}_3\overline{x}_2)$

Decision Levels (continued)

dl	**Assignment**	**Clauses**	
0	–	$(\overline{x}_1\ \overline{x}_4\ x_3)(\overline{x}_3\overline{x}_2)$	
1	$\{x_1 \mapsto 1\}$	$(\overline{x}_1\ \overline{x}_4\ x_3)(\overline{x}_3\overline{x}_2)$	x_1@1

Decision Levels (continued)

dl	**Assignment**	**Clauses**	
0	–	$(\overline{x}_1\ \overline{x}_4\ x_3)(\overline{x}_3\overline{x}_2)$	
1	$\{x_1 \mapsto 1\}$	$(\overline{x}_1\ \overline{x}_4\ x_3)(\overline{x}_3\overline{x}_2)$	x_1@1
2	$\{x_1 \mapsto 1, x_4 \mapsto 1\}$	$(\overline{x}_1\ \overline{x}_4\ x_3)(\overline{x}_3\overline{x}_2)$	x_4@2
	$\{x_1 \mapsto 1, x_4 \mapsto 1, x_3 \mapsto 1\}$	$(x_3)(\overline{x}_3\overline{x}_2)$	x_3@2

Decision Levels (continued)

dl	**Assignment**	**Clauses**	
0	–	$(\bar{x}_1\ \bar{x}_4\ x_3)(\bar{x}_3\bar{x}_2)$	
1	$\{x_1 \mapsto 1\}$	$(\bar{x}_1\ \bar{x}_4\ x_3)(\bar{x}_3\bar{x}_2)$	x_1@1
2	$\{x_1 \mapsto 1, x_4 \mapsto 1\}$	$(\bar{x}_1\ \bar{x}_4\ x_3)(\bar{x}_3\bar{x}_2)$	x_4@2
	$\{x_1 \mapsto 1, x_4 \mapsto 1, x_3 \mapsto 1\}$	$(x_3)(\bar{x}_3\bar{x}_2)$	x_3@2
	$\{x_1 \mapsto 1, x_4 \mapsto 1, x_3 \mapsto 1, x_2 \mapsto 0\}$	$(\bar{x}_2)$	$\neg x_2$@2

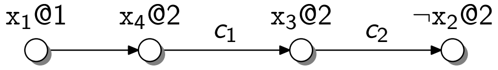
Decision Levels (continued)

dl	**Assignment**	**Clauses**	
0	–	$(\overline{x}_1\ \overline{x}_4\ x_3)(\overline{x}_3\overline{x}_2)$	
1	$\{x_1 \mapsto 1\}$	$(\overline{x}_1\ \overline{x}_4\ x_3)(\overline{x}_3\overline{x}_2)$	$x_1@1$
2	$\{x_1 \mapsto 1, x_4 \mapsto 1\}$	$(\overline{x}_1\ \overline{x}_4\ x_3)(\overline{x}_3\overline{x}_2)$	$x_4@2$
	$\{x_1 \mapsto 1, x_4 \mapsto 1, x_3 \mapsto 1\}$	$(x_3)(\overline{x}_3\overline{x}_2)$	$x_3@2$
	$\{x_1 \mapsto 1, x_4 \mapsto 1, x_3 \mapsto 1, x_2 \mapsto 0\}$	$(\overline{x}_2)$	$\neg x_2@2$

- $\{x_1 \mapsto 1, x_4 \mapsto 1, x_3 \mapsto 1, x_2 \mapsto 0\}$ satisfies $(\overline{x}_1\ \overline{x}_4\ x_3)(\overline{x}_3\overline{x}_2)$

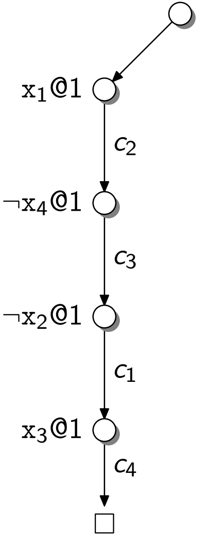
Boolean Constraint Propagation, Implication Graph

$$c_1 \equiv (\bar{x}_1\ \bar{x}_4\ x_3)\ ,\ c_2 \equiv (\bar{x}_3 \bar{x}_2)$$



- Nodes labelled with *decisions*
- Edges labelled with *antecedents*

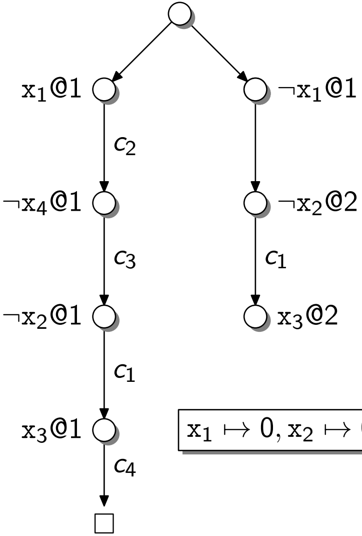
What if a decision is wrong?



c_1	$(x_2\, x_3)$
c_2	$(\overline{x}_1 \overline{x}_4)$
c_3	$(\overline{x}_2 x_4)$
c_4	$(\overline{x}_1 x_2 \overline{x}_3)$

Backtracking

What if a decision is wrong?



$\boxed{x_1 \mapsto 0, x_2 \mapsto 0, x_3 \mapsto 1}$

c_1	$(x_2\, x_3)$
c_2	$(\overline{x}_1 \overline{x}_4)$
c_3	$(\overline{x}_2 x_4)$
c_4	$(\overline{x}_1 x_2 \overline{x}_3)$

Davis-Putnam-Loveland-Logeman (DPLL)

- Decide
  Choose a variable and make a decision
- Propagate
  Propagate implications
- Backtrack
  “Undo” decisions which lead to conflict

Conflict-Driven Backtracking

- How can we do *systematic* backtracking?

Definition (Partial Implication Graph)

Sub-graph of an implication graph illustrating binary constraint propagation (BCP) at a specific implication level

Definition (Conflict Graph)

An implication graph in which BCP has reached a conflict

$$c_1 \equiv (\mathtt{x}_2\,\mathtt{x}_3),\ c_2 \equiv (\overline{\mathtt{x}}_1\overline{\mathtt{x}}_4),\ c_3 \equiv (\overline{\mathtt{x}}_2\mathtt{x}_4),\ c_4 \equiv (\overline{\mathtt{x}}_1\mathtt{x}_2\overline{\mathtt{x}}_3)$$

$$\mathtt{x}_1@1 \xrightarrow{c_2} \neg\mathtt{x}_4@1 \xrightarrow{c_3} \neg\mathtt{x}_2@1 \xrightarrow{c_1} \mathtt{x}_3@1 \xrightarrow{c_4} \square$$

Conflict-Driven Backtracking, Learning

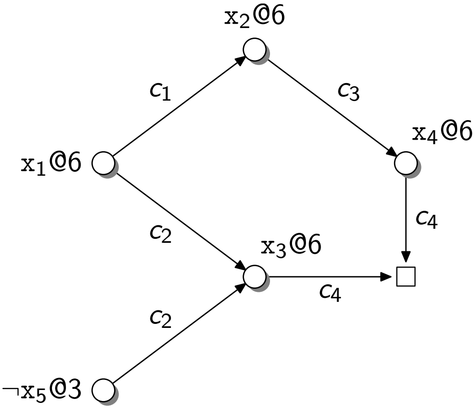
$$c_1 \equiv (\mathtt{x}_2\,\mathtt{x}_3),\ c_2 \equiv (\overline{\mathtt{x}}_1\overline{\mathtt{x}}_4),\ c_3 \equiv (\overline{\mathtt{x}}_2\mathtt{x}_4),\ c_4 \equiv (\overline{\mathtt{x}}_1\mathtt{x}_2\overline{\mathtt{x}}_3)$$

$$\mathtt{x}_1@1 \xrightarrow{c_2} \neg\mathtt{x}_4@1 \xrightarrow{c_3} \neg\mathtt{x}_2@1 \xrightarrow{c_1} \mathtt{x}_3@1 \xrightarrow{c_4} \Box$$

- Analyse conflict
- Add *learnt conflict clause* (($\overline{\mathtt{x}}_1$) in our example)
- Backtrack
  - to highest decision level in conflict clause that's not the current decision level
  - to 0, if we learnt a unit clause

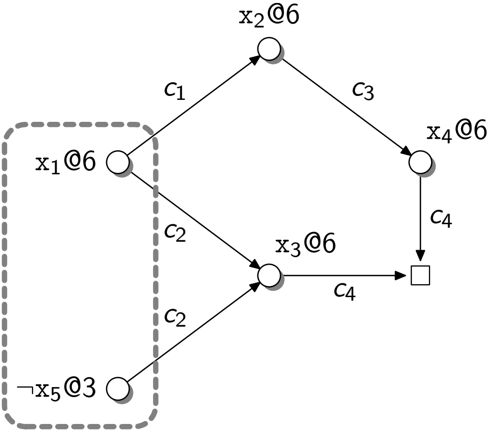
Example: Learning Conflict Clauses

$$
\begin{aligned}
c_1 &= (\overline{x}_1 x_2)\\
c_2 &= (\overline{x}_1 x_3 x_5)\\
c_3 &= (\overline{x}_2 x_4)\\
c_4 &= (\overline{x}_3 \overline{x}_4)
\end{aligned}
$$



Example: Learning Conflict Clauses

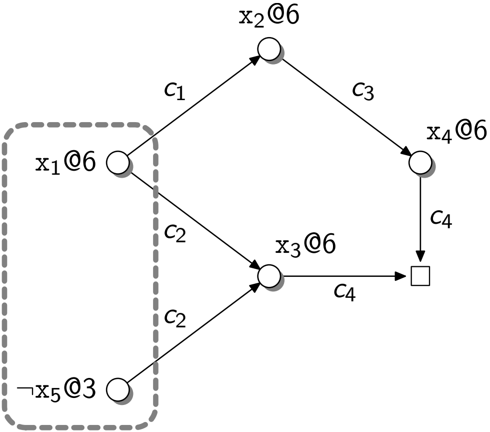
$$
\begin{aligned}
c_1 &= (\overline{\mathtt{x}}_1 \mathtt{x}_2) \\
c_2 &= (\overline{\mathtt{x}}_1 \mathtt{x}_3 \mathtt{x}_5) \\
c_3 &= (\overline{\mathtt{x}}_2 \mathtt{x}_4) \\
c_4 &= (\overline{\mathtt{x}}_3 \overline{\mathtt{x}}_4)
\end{aligned}
$$



- Conflict clause: $(\overline{\mathtt{x}}_1\ \mathtt{x}_5)$

Example: Learning Conflict Clauses

$$
\begin{aligned}
c_1 &= (\overline{x}_1 x_2) \\
c_2 &= (\overline{x}_1 x_3 x_5) \\
c_3 &= (\overline{x}_2 x_4) \\
c_4 &= (\overline{x}_3 \overline{x}_4)
\end{aligned}
$$



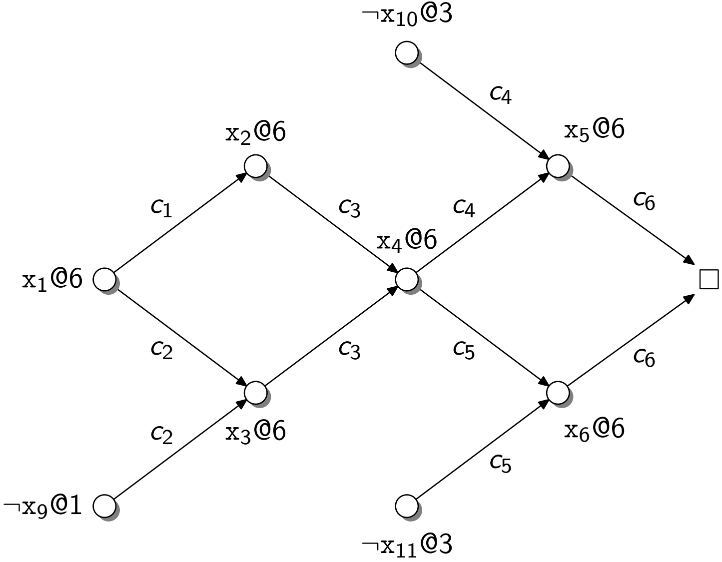
- Conflict clause: $(\overline{x}_1\ x_5)$
- Backtracking level: 3
  - Erase all decisions from decision level 4 onwards

Asserting Clauses

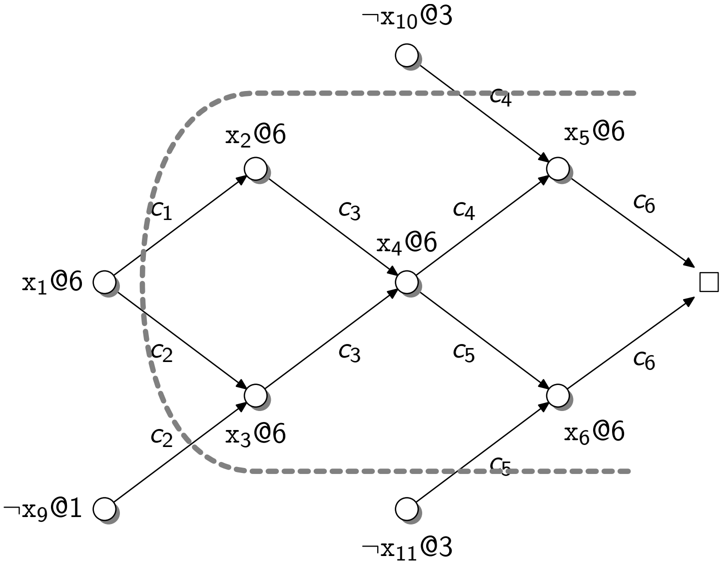
$$
\begin{array}{lcl}
c_1 & = & (\overline{\mathtt{x}}_1 \mathtt{x}_2) \\
c_2 & = & (\overline{\mathtt{x}}_1 \mathtt{x}_3 \mathtt{x}_5) \\
c_3 & = & (\overline{\mathtt{x}}_2 \mathtt{x}_4) \\
c_4 & = & (\overline{\mathtt{x}}_3 \overline{\mathtt{x}}_4) \\
\hline
c_5 & = & (\overline{\mathtt{x}}_1 \ \mathtt{x}_5)
\end{array}
$$

- We backtracked to decision level of $\mathtt{x}_5$
- Since $\mathtt{x}_5 \mapsto 0$, $(\overline{\mathtt{x}}_1 \ \mathtt{x}_5)$ forces an immediate implication
- Such a clause is called **asserting clause**

Choosing Conflict Clauses

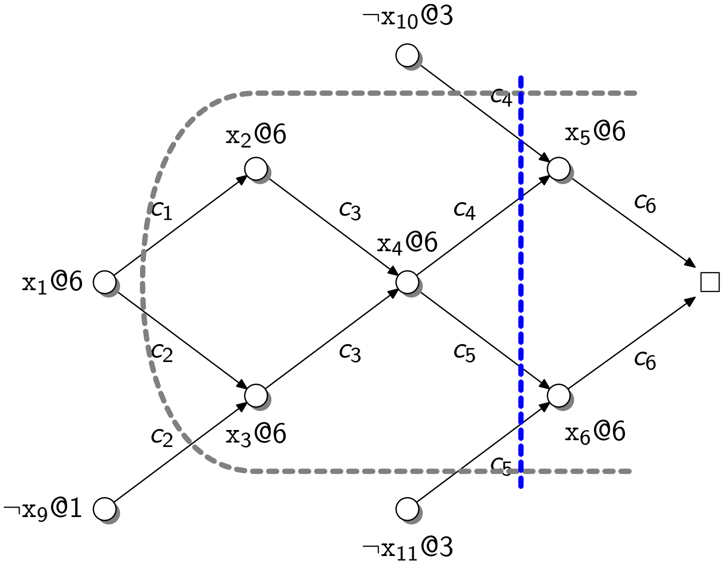


Choosing Conflict Clauses



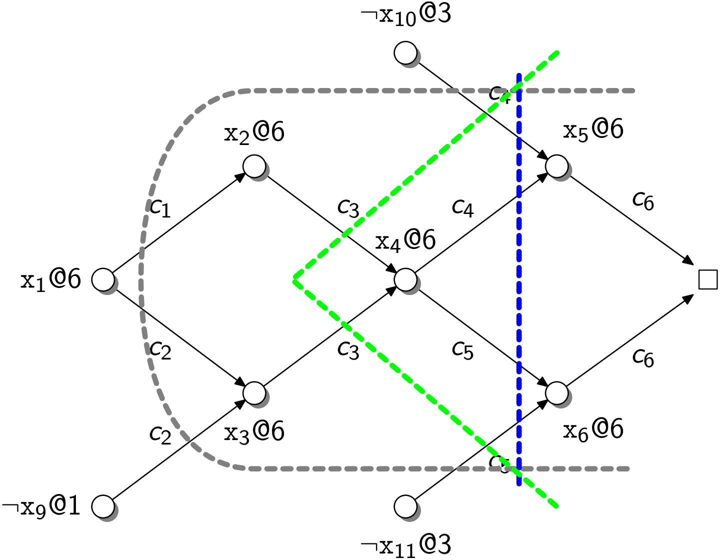
1.) $(x_{10}\ \overline{x}_1\ x_9\ x_{11})$

Choosing Conflict Clauses



1.) $(x_{10}\ \overline{x}_1\ x_9\ x_{11})$ 2.) $(x_{10}\ \overline{x}_4\ x_{11})$

Choosing Conflict Clauses



1.) $(x_{10}\ \overline{x}_1\ x_9\ x_{11})$ 2.) $(x_{10}\ \overline{x}_4\ x_{11})$ 3.) $(x_{10}\ \overline{x}_2\ \overline{x}_3\ x_{11})$

Conflict Clauses: Unique Implication Point

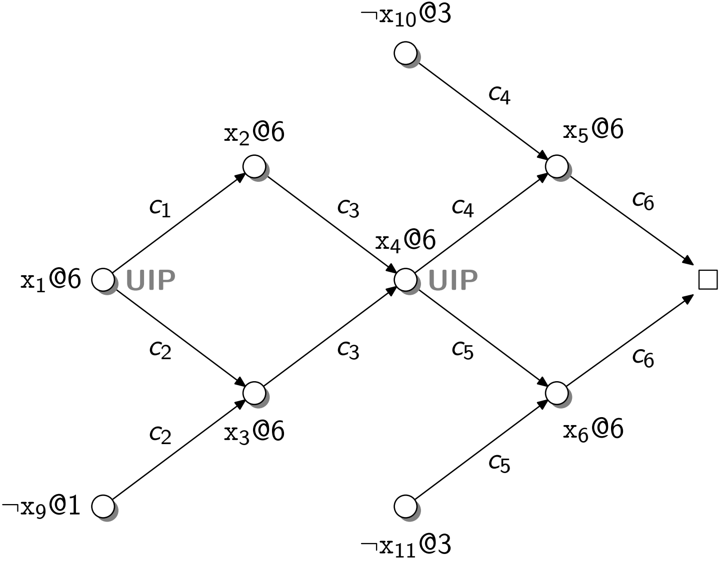
Definition (Unique Implication Point)

Any node (other than the conflict node) in the partial conflict graph which is on *all paths*

- from the decision node
- to the conflict node

Note: The decision node is a UIP by definition.

Conflict Clauses: Unique Implication Point (ctd.)



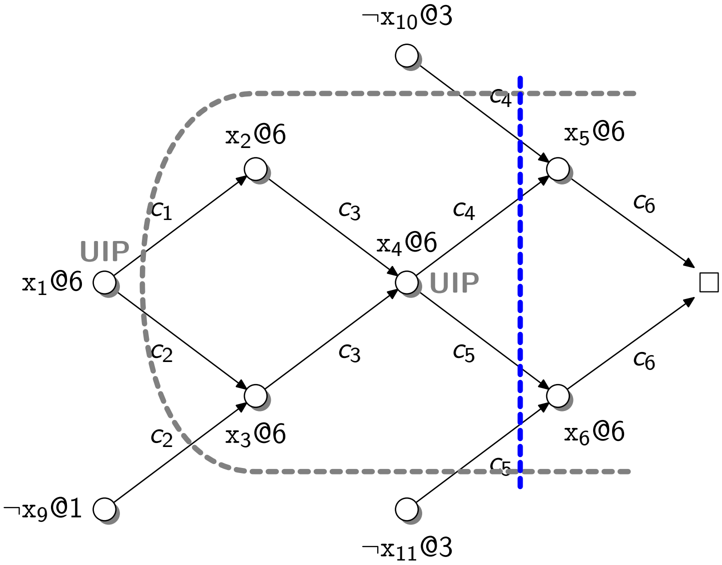
Conflict Clauses: Unique Implication Point (ctd.)

Definition (First Unique Implication Point)

The UIP that's closest to the conflict node

- Choose conflict clause that contains *First UIP* as only literal at the current decision level
- Advantages:
  - Clause is an assertion clause
  - Backtracks to *lowest decision level*
    Why? Clause with First UIP "*subsumes*" other UIPs

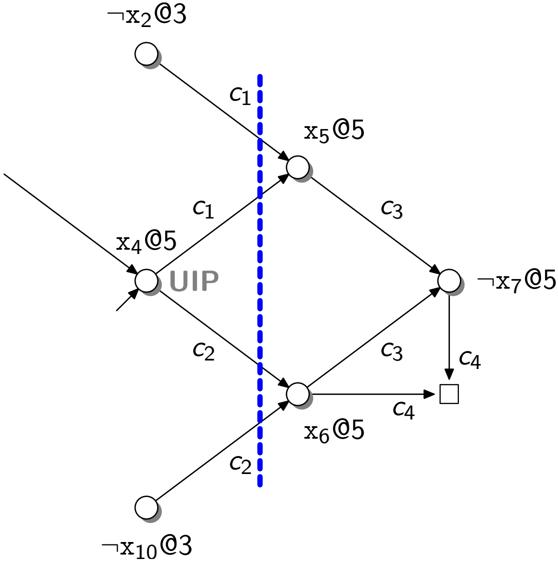
Conflict Clauses: Unique Implication Point (ctd.)



1.) $(x_{10}\ \overline{x}_1\ x_9\ x_{11})$ 2.) $(x_{10}\ \overline{x}_4\ x_{11})$

Conflict Clauses and Resolution

$$\begin{array}{lcl} c_1 & = & (\overline{x}_4\ x_2\ x_5) \\ c_2 & = & (\overline{x}_4\ x_{10}\ x_6) \\ c_3 & = & (\overline{x}_5\ \overline{x}_6\ \overline{x}_7) \\ c_4 & = & (\overline{x}_6\ x_7) \\ \hline c_5 & = & (x_2\ \overline{x}_4\ x_{10}) \end{array}$$



Conflict Clauses and Resolution

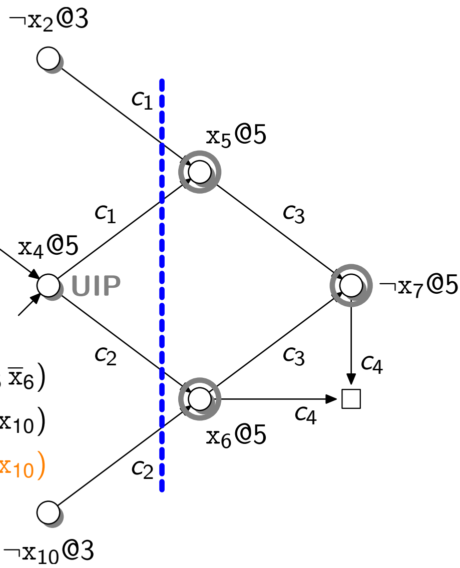
$$
\begin{array}{lcl}
c_1 & = & (\overline{x}_4\ x_2\ x_5) \\
c_2 & = & (\overline{x}_4\ x_{10}\ x_6) \\
c_3 & = & (\overline{x}_5\ \overline{x}_6\ \overline{x}_7) \\
c_4 & = & (\overline{x}_6\ x_7) \\
\hline
c_5 & = & (x_2\ \overline{x}_4\ x_{10})
\end{array}
$$

Order: x_4, x_5, x_6, x_7

$$t_1 = \mathrm{Res}(c_4, c_3, x_7) = (\overline{x}_5\ \overline{x}_6)$$

$$t_2 = \mathrm{Res}(t_1, c_2, x_6) = (\overline{x}_4\ \overline{x}_5\ x_{10})$$

$$t_3 = \mathrm{Res}(t_2, c_1, x_5) = (x_2\ \overline{x}_4\ x_{10})$$



Conflict Clauses and Resolution

- Start with currently conflicting clause (c_4 in example)
- Choose last assigned literal ($\mathtt{x}_7$ in example)
- $\mathtt{x}_7$ follows from c_3
- Phase of $\mathtt{x}_7$ in c_4 differs from c_3
- $t_1 = \mathrm{Res}(c_4, c_3, \mathtt{x}_7)$
- Iterate until we reach UIP

  (i.e., t_i contains UIP as single literal at current decision level)

  In our example:

$$
\begin{aligned}
t_1 &= \mathrm{Res}(c_4, c_3, \mathtt{x}_7) = & (\overline{\mathtt{x}}_5\, \overline{\mathtt{x}}_6) \\
t_2 &= \mathrm{Res}(t_1, c_2, \mathtt{x}_6) = & (\overline{\mathtt{x}}_4\, \overline{\mathtt{x}}_5\, \mathtt{x}_{10}) \\
t_3 &= \mathrm{Res}(t_2, c_1, \mathtt{x}_5) = & (\mathtt{x}_2\, \overline{\mathtt{x}}_4\, \mathtt{x}_{10})
\end{aligned}
$$

Conflict Clauses

- Each conflict clause consequence
  - of F and
  - previously derived conflict clauses
- Derived using resolution
- Therefore, conflict clause is implied by original CNF formula F
- Therefore, SAT-solver can be used to **find resolution proofs**!

DPLL Completed

① If conflict at decision level 0 $\rightarrow$ UNSAT

② Repeat:

- ❶ if all variables assigned return SAT
- ❷ Make decision
- ❸ Propagate constraints
- ❹ No conflict? Go to ❶
- ❺ If decision level = 0 return UNSAT
- ❻ Analyse conflict
- ❼ Add conflict clause
- ❽ Backtrack and go to ❸

DPLL Completed

① If conflict at decision level 0 $\rightarrow$ UNSAT

② Repeat:

   ❶ if all variables assigned return SAT
   ❷ Make decision
   ❸ Propagate constraints
   ❹ No conflict? Go to ❶
   ❺ If decision level = 0 return UNSAT
   ❻ Analyse conflict
   ❼ Add conflict clause
   ❽ Backtrack and go to ❸

Termination argument:

- Solver never enters same decision level with same partial assignment

DIMACS Format

```
c A sample .cnf file
p cnf 3 2
1 -3 0
2 3 -1 0
```

DIMACS = Discrete Mathematics and Theoretical Computer Science, a collaboration Rutgers & Princeton, to determine practical algorithm performance on computationally hard problems

Unsatisfiable Core

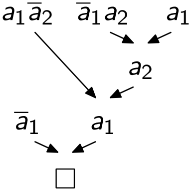
- If instance unsatisfiable, SAT-solver derives $\Box$
- Follow resolution edges starting from $\Box$
  - we obtain a resolution refutation proof
  - does *not* necessarily contain all clauses visited during SAT-run
  - represents *unsatisfiable core*

Definition (Unsatisfiable Core)

Any unsatisfiable subset of the original set of clauses

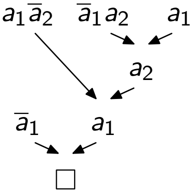
Variable Order

- Does the order in which we assign variables matter?
- How about the values we choose?



Variable Order

- Does the order in which we assign variables matter?
- How about the values we choose?



Probably the most important element in SAT solving!

Decision Heuristics: DLIS

Dynamic Largest Individual Sum –
choose assignment s.t. number of satisfied clauses is maximised

- $p_{\mathtt{x}}$ . . . # of unresolved clauses containing $\mathtt{x}$
- $n_{\mathtt{x}}$ . . . # of unresolved clauses containing $\overline{\mathtt{x}}$
- Let $\mathtt{x}$ be variable for which $p_{\mathtt{x}}$ is maximal
- Let $\mathtt{y}$ be variable for which $n_{\mathtt{y}}$ is maximal
- If $p_{\mathtt{x}} > n_{\mathtt{y}}$ choose $\mathtt{x} \mapsto 1$
- Otherwise, choose $\mathtt{y} \mapsto 0$

Disadvantage: High overhead

Decision Heuristics: VSDIS

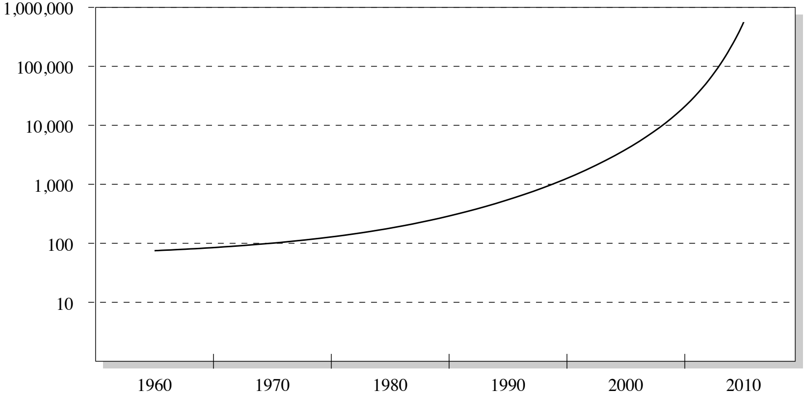
Variable State Independent Decaying Sum –
favour literals in recently added conflict clauses

- Each literal has counter initialised to 0
- When clause is added, literals in clause are *boosted*
- Periodically, all counters divided by constant
- Choose unassigned literal with highest counter

- Implemented in CHAFF
  - Maintain list of unassigned literals sorted by counter
  - Update list when adding conflict clauses
  - Decision in $O(1)$
- Improved performance by order of magnitude

Performance of SAT Solvers

- Scales to hundreds of thousands of variables
  - for "benign" problems
  - challenges:
    - *pigeon hole problems* (size of resolution proof exponential)
    - chains of $\oplus$



BDDs vs SAT

	BDD	**SAT**
Variables	Hundreds	hundreds of thousands
Complexity	PSPACE-complete	NP-complete
Assignments	$O(n)$	SAT-run
Canonical	Yes	No
Equality check	$O(1)$ (hashing)	SAT-run ($F \oplus G$)
Quantifier elimination	Yes	Co-Factoring

Summary

- Efficient SAT checking for propositional logic
- Next time: Satisfiability Modulo Theories (SMT)
  - Theory-specific reasoning