State-of-the-art SAT Solving

Marijn J. H. Heule

University of Texas

April 16, 2012 @ ACL2
The Satisfiability (SAT) problem

\((x_5 \lor x_8 \lor \bar{x}_2) \land (x_2 \lor \bar{x}_1 \lor \bar{x}_3) \land (\bar{x}_8 \lor \bar{x}_3 \lor \bar{x}_7) \land (\bar{x}_5 \lor x_3 \lor x_8)\) \land

\((\bar{x}_6 \lor \bar{x}_1 \lor \bar{x}_5) \land (x_8 \lor \bar{x}_9 \lor x_3) \land (x_2 \lor x_1 \lor x_3) \land (\bar{x}_1 \lor x_8 \lor x_4)\) \land

\((\bar{x}_9 \lor \bar{x}_6 \lor x_8) \land (x_8 \lor x_3 \lor \bar{x}_9) \land (x_9 \lor \bar{x}_3 \lor x_8) \land (x_6 \lor \bar{x}_9 \lor x_5)\) \land

\((x_2 \lor \bar{x}_3 \lor \bar{x}_8) \land (x_8 \lor \bar{x}_6 \lor \bar{x}_3) \land (x_8 \lor \bar{x}_3 \lor \bar{x}_1) \land (\bar{x}_8 \lor x_6 \lor \bar{x}_2)\) \land

\((x_7 \lor x_9 \lor \bar{x}_2) \land (x_8 \lor \bar{x}_9 \lor x_2) \land (\bar{x}_1 \lor \bar{x}_9 \lor x_4) \land (x_8 \lor \bar{x}_1 \lor \bar{x}_2)\) \land

\((x_3 \lor \bar{x}_4 \lor \bar{x}_6) \land (\bar{x}_1 \lor \bar{x}_7 \lor x_5) \land (\bar{x}_7 \lor x_1 \lor x_6) \land (\bar{x}_5 \lor x_4 \lor \bar{x}_6)\) \land

\((\bar{x}_4 \lor x_9 \lor \bar{x}_8) \land (x_2 \lor x_9 \lor x_1) \land (x_5 \lor \bar{x}_7 \lor x_1) \land (\bar{x}_7 \lor \bar{x}_9 \lor \bar{x}_6)\) \land

\((x_2 \lor x_5 \lor x_4) \land (x_8 \lor \bar{x}_4 \lor x_5) \land (x_5 \lor x_9 \lor x_3) \land (\bar{x}_5 \lor \bar{x}_7 \lor x_9)\) \land

\((x_2 \lor \bar{x}_8 \lor x_1) \land (\bar{x}_7 \lor x_1 \lor x_5) \land (x_1 \lor x_4 \lor x_3) \land (\bar{x}_1 \lor \bar{x}_9 \lor \bar{x}_4)\) \land

\((x_3 \lor x_5 \lor x_6) \land (\bar{x}_6 \lor x_3 \lor \bar{x}_9) \land (\bar{x}_7 \lor x_5 \lor x_9) \land (x_7 \lor \bar{x}_5 \lor \bar{x}_2)\) \land

\((x_4 \lor x_7 \lor x_3) \land (x_4 \lor \bar{x}_9 \lor \bar{x}_7) \land (x_5 \lor \bar{x}_1 \lor x_7) \land (x_5 \lor \bar{x}_1 \lor x_7)\) \land

\((x_6 \lor x_7 \lor \bar{x}_3) \land (\bar{x}_8 \lor \bar{x}_6 \lor \bar{x}_7) \land (x_6 \lor x_2 \lor x_3) \land (\bar{x}_8 \lor x_2 \lor x_5)\)

Does there exist an assignment satisfying all clauses?
Search for a satisfying assignment (or proof none exists)

Play the SAT game:
http://www.cril.univ-artois.fr/~roussel/satgame/satgame.php
Motivation

From 100 variables, 200 constraints (early 90s) to 1,000,000 vars. and 20,000,000 cls. in 20 years.

Applications:
Hardware and Software Verification, Planning, Scheduling, Optimal Control, Protocol Design, Routing, Combinatorial problems, Equivalence Checking, etc.

SAT used to solve many other problems!
Motivation

From 100 variables, 200 constraints (early 90s) to \textbf{1,000,000 vars.} and \textbf{20,000,000 cls.} in 20 years.

Applications:
Hardware and Software Verification, Planning, Scheduling, Optimal Control, Protocol Design, Routing, Combinatorial problems, Equivalence Checking, etc.

SAT used to solve many other problems!
Motivation

From 100 variables, 200 constraints (early 90s) to 1,000,000 vars. and 20,000,000 cls. in 20 years.

Applications:
Hardware and Software Verification, Planning, Scheduling, Optimal Control, Protocol Design, Routing, Combinatorial problems, Equivalence Checking, etc.

SAT used to solve many other problems!
Overview

Search for Lemmas
- Learning Lemmas
- Data-structures
- Heuristics

Search for Simplification
- Variable elimination
- Blocked clause elimination
- Unhiding redundancy

Depth-first search

Breadth-first search
Conflict-driven SAT solvers: Search and Analysis

\((x_1 \lor x_4) \land (x_3 \lor \overline{x}_4 \lor \overline{x}_5) \land (\overline{x}_3 \lor \overline{x}_2 \lor \overline{x}_4) \land F_{\text{extra}}\)
Conflict-driven SAT solvers: Search and Analysis

\[ (x_1 \lor x_4) \land (x_3 \lor \bar{x}_4 \lor \bar{x}_5) \land (\bar{x}_3 \lor \bar{x}_2 \lor \bar{x}_4) \land \mathcal{F}_{\text{extra}} \]

\[ x_5 = 1 \]
Conflict-driven SAT solvers: Search and Analysis

\[
(x_1 \lor x_4) \land \\
(x_3 \lor \neg x_4 \lor \neg x_5) \land \\
(\neg x_3 \lor \neg x_2 \lor \neg x_4) \land \\
F_{\text{extra}}
\]

\[
\begin{align*}
x_5 &= 1 \\
x_2 &= 1
\end{align*}
\]
Conflict-driven SAT solvers: Search and Analysis

\[
\begin{align*}
(x_1 \lor x_4) & \land \\
(x_3 \lor \overline{x}_4 \lor \overline{x}_5) & \land \\
(\overline{x}_3 \lor \overline{x}_2 \lor \overline{x}_4) & \land \\
\mathcal{F}_{\text{extra}}
\end{align*}
\]
Conflict-driven SAT solvers: Search and Analysis

\[(x_1 \lor x_4) \land (x_3 \lor \bar{x}_4 \lor \bar{x}_5) \land (\bar{x}_3 \lor \bar{x}_2 \lor \bar{x}_4) \land F_{\text{extra}}\]

State-of-the-art SAT Solving

Marijn J. H. Heule (UT)
Conflict-driven SAT solvers: Search and Analysis

\[(x_1 \lor x_4) \land (x_3 \lor \overline{x}_4 \lor \overline{x}_5) \land (\overline{x}_3 \lor \overline{x}_2 \lor \overline{x}_4) \land F_{\text{extra}}\]

\[
\begin{align*}
x_5 &= 1 \\
x_2 &= 1 \\
x_1 &= 0 \\
x_4 &= 1
\end{align*}
\]
Conflict-driven SAT solvers: Search and Analysis

\[
\begin{align*}
(x_1 \lor x_4) & \land \\
(x_3 \lor \bar{x}_4 \lor \bar{x}_5) & \land \\
(\bar{x}_3 \lor \bar{x}_2 \lor \bar{x}_4) & \land \\
F_{\text{extra}} & 
\end{align*}
\]

Marijn J. H. Heule (UT)

State-of-the-art SAT Solving

April 16, 2012 @ ACL2
Conflict-driven SAT solvers: Search and Analysis

\[(x_1 \lor x_4) \land (x_3 \lor \bar{x}_4 \lor \bar{x}_5) \land (ar{x}_3 \lor \bar{x}_2 \lor \bar{x}_4) \land F_{\text{extra}}\]

Marijn J. H. Heule (UT)
Conflict-driven SAT solvers: Search and Analysis

\[
\begin{align*}
(x_1 \lor x_4) \land & \\
(x_3 \lor \bar{x}_4 \lor \bar{x}_5) \land & \\
(\bar{x}_3 \lor \bar{x}_2 \lor \bar{x}_4) \land & \\
F_{\text{extra}}
\end{align*}
\]

\[
\begin{align*}
& x_1 = 0 \\
& x_4 = 1 \\
& x_2 = 1 \\
& x_3 = 0 \\
& (\bar{x}_2 \lor \bar{x}_4 \lor \bar{x}_5)
\end{align*}
\]

Marijn J. H. Heule (UT)
State-of-the-art SAT Solving
April 16, 2012 @ ACL2 6 / 55
Conflict-driven SAT solvers: Search and Analysis

\[
\begin{align*}
(x_1 \lor x_4) \land \\
(x_3 \lor \bar{x}_4 \lor \bar{x}_5) \land \\
(\bar{x}_3 \lor \bar{x}_2 \lor \bar{x}_4) \land \\
\mathcal{F}_{\text{extra}}
\end{align*}
\]
Conflict-driven SAT solvers: Search and Analysis

\[(x_1 \lor x_4) \land (x_3 \lor \bar{x}_4 \lor \bar{x}_5) \land (\bar{x}_3 \lor \bar{x}_2 \lor \bar{x}_4) \land \mathcal{F}_{\text{extra}}\]

\[
\begin{align*}
(x_1 & \lor x_4) \land \\
(x_3 & \lor \bar{x}_4 \lor \bar{x}_5) \land \\
(\bar{x}_3 & \lor \bar{x}_2 \lor \bar{x}_4) \land \\
\mathcal{F}_{\text{extra}}&
\end{align*}
\]

Marijn J. H. Heule (UT)
Conflict-driven SAT solvers: Search and Analysis

\[(x_1 \lor x_4) \land (x_3 \lor \bar{x}_4 \lor \bar{x}_5) \land (\bar{x}_3 \lor \bar{x}_2 \lor \bar{x}_4) \land F_{\text{extra}}\]

\[(\bar{x}_2 \lor \bar{x}_4 \lor \bar{x}_5)\]

Marijn J. H. Heule (UT)
State-of-the-art SAT Solving
April 16, 2012 @ ACL2
Conflict-driven SAT solvers: Pseudo-code

1: while TRUE do
2: \( l_{\text{decision}} := \text{GetDecisionLiteral}() \)
3: If no \( l_{\text{decision}} \) then return satisfiable
4: \( \mathcal{F} := \text{Simplify}( \mathcal{F}(l_{\text{decision}} \leftarrow 1) ) \)
5: while \( \mathcal{F} \) contains \( C_{\text{falsified}} \) do
6: \( C_{\text{conflict}} := \text{AnalyzeConflict}( C_{\text{falsified}} ) \)
7: If \( C_{\text{conflict}} = \emptyset \) then return unsatisfiable
8: \( \text{BackTrack}( C_{\text{conflict}} ) \)
9: \( \mathcal{F} := \text{Simplify}( \mathcal{F} \cup \{ C_{\text{conflict}} \} ) \)
10: end while
11: end while
Learning conflict clauses (lemma’s)

Marijn J. H. Heule (UT) State-of-the-art SAT Solving April 16, 2012 @ ACL2
Learning conflict clauses (lemma’s)

\[
7x_6 = 0
\]
\[
7x_{12} = 0
\]
\[
7x_{11} = 1
\]
\[
x_7 = 1
\]
\[
x_2 = 0
\]
\[
x_{10} = 0
\]
\[
1x_4 = 1
\]
\[
3x_8 = 1
\]
\[
x_{17} = 0
\]
\[
3x_1 = 1
\]
\[
x_3 = 1
\]
\[
x_5 = 0
\]
\[
7x_{18} = 1
\]
\[
7x_{18} = 0
\]
\[
2x_{19} = 1
\]

\[
(\neg x_1 \lor \neg x_3 \lor x_5 \lor x_{17} \lor \neg x_{19})
\]

tri-asserting clause
Learning conflict clauses (lemma’s)

\[(x_{10} \lor \neg x_8 \lor x_{17} \lor \neg x_{19})\]

first unique implication point
Learning conflict clauses (lemma’s)

\[(x_2 \lor \neg x_4 \lor \neg x_8 \lor x_{17} \lor \neg x_{19})\]

second unique implication point

Marijn J. H. Heule (UT) State-of-the-art SAT Solving April 16, 2012 @ ACL2 8 / 55
Average Learned Clause Length

![Graph showing the average learned clause length over iterations. The graph has a y-axis ranging from 0 to 600 and an x-axis ranging from 0 to 80. The data points are scattered across the graph, with a trend line indicating an increasing pattern.]
Data-structures

Watch pointers
Simple data structure for unit propagation
Conflict-driven: Watch pointers (1)

\[ \varphi = \{ x_1 = *, x_2 = *, x_3 = *, x_4 = *, x_5 = *, x_6 = * \} \]
Conflict-driven: Watch pointers (1)

\[ \varphi = \{ x_1 = *, x_2 = *, x_3 = *, x_4 = *, x_5 = 1, x_6 = * \} \]
Conflict-driven: Watch pointers (1)

\[ \varphi = \{ x_1 = \ast, x_2 = \ast, x_3 = 1, x_4 = \ast, x_5 = 1, x_6 = \ast \} \]
Conflict-driven: Watch pointers (1)

$$\varphi = \{x_1 = *, x_2 = *, x_3 = 1, x_4 = *, x_5 = 1, x_6 = * \}$$
Conflict-driven: Watch pointers (1)

\[ \varphi = \{ x_1 = 1, x_2 = *, x_3 = 1, x_4 = *, x_5 = 1, x_6 = * \} \]
Conflict-driven: Watch pointers (1)

\[ \varphi = \{ x_1 = 1, x_2 = *, x_3 = 1, x_4 = *, x_5 = 1, x_6 = * \} \]
Conflict-driven: Watch pointers (1)

\[ \varphi = \{ x_1 = 1, x_2 = *, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = * \} \]

![Diagram](image-url)
Conflict-driven: Watch pointers (1)

\[ \varphi = \{ x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = * \} \]
Conflict-driven: Watch pointers (1)

\[ \varphi = \{ x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = 1 \} \]
Conflict-driven: Watch pointers (1)

\[ \varphi = \{ x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = 1 \} \]
Conflict-driven: Watch pointers (2)

Only examine (get in the cache) a clause when both
- a watch pointer gets falsified
- the other one is not satisfied

While backjumping, just unassign variables

Conflict clauses $\rightarrow$ watch pointers

No detailed information available

Not used for binary clauses
Percentage visited clauses with other watched literal true
Heuristics
Most important CDCL heuristics

Variable selection heuristics
- aim: minimize the search space
- plus: could compensate a bad value selection

Value selection heuristics
- aim: guide search towards a solution (or conflict)
- plus: could compensate a bad variable selection, cache solutions of subproblems [PipatsrisawatDarwiche’07]

Restart strategies
- aim: avoid heavy-tail behavior [GomesSelmanCrato’97]
- plus: focus search on recent conflicts when combined with dynamic heuristics
Most important CDCL heuristics

Variable selection heuristics

- aim: minimize the search space
- plus: could compensate a bad value selection

Value selection heuristics

- aim: guide search towards a solution (or conflict)
- plus: could compensate a bad variable selection, cache solutions of subproblems [PipatsrisawatDarwiche’07]

Restart strategies

- aim: avoid heavy-tail behavior [GomesSelmanCrato’97]
- plus: focus search on recent conflicts when combined with dynamic heuristics
Most important CDCL heuristics

Variable selection heuristics

- aim: minimize the search space
- plus: could compensate a bad value selection

Value selection heuristics

- aim: guide search towards a solution (or conflict)
- plus: could compensate a bad variable selection, cache solutions of subproblems [PipatsrisawatDarwiche’07]

Restart strategies

- aim: avoid heavy-tail behavior [GomesSelmanCrato’97]
- plus: focus search on recent conflicts when combined with dynamic heuristics
Variable selection heuristics

Based on the occurrences in the (reduced) formula
- examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads
- not practical for CDCL solver due to watch pointers

Variable State Independent Decaying Sum (VSIDS)
- original idea (zChaff): for each conflict, increase the score of involved variables by 1, half all scores each 256 conflicts
  [MoskewiczMZZM2001]
- improvement (MiniSAT): for each conflict, increase the score of involved variables by $\delta$ and increase $\delta := 1.05\delta$
  [EenSörensson2003]
Variable selection heuristics

Based on the occurrences in the (reduced) formula
- examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads
- not practical for CDCL solver due to watch pointers

Variable State Independent Decaying Sum (VSIDS)
- original idea (zChaff): for each conflict, increase the score of involved variables by 1, half all scores each 256 conflicts
  \[ \delta := 1.05 \delta \]
- improvement (MiniSAT): for each conflict, increase the score of involved variables by \( \delta \) and increase \( \delta := 1.05\delta \)
Visualization of VSIDS in PicoSAT

http://www.youtube.com/watch?v=MOjhFywLre8
Value selection heuristics

Based on the occurrences in the (reduced) formula
- examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads
- not practical for CDCL solver due to watch pointers

Based on the encoding / consequently
- negative branching (early MiniSAT) [EenSörensson2003]

Based on the last implied value (phase-saving)
- introduced to CDCL [PipatsrisawatDarwiche2007]
- already used in local search [HirschKojevnikov2001]
Value selection heuristics

Based on the occurrences in the (reduced) formula
- examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads
- not practical for CDCL solver due to watch pointers

Based on the encoding / consequently
- negative branching (early MiniSAT)  [EenSörensson2003]

Based on the last implied value (phase-saving)
- introduced to CDCL  [PipatsrisawatDarwiche2007]
- already used in local search  [HirschKojevnikov2001]
Value selection heuristics

Based on the occurrences in the (reduced) formula
- examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads
- not practical for CDCL solver due to watch pointers

Based on the encoding / consequently
- negative branching (early MiniSAT) [EenSörensson2003]

Based on the last implied value (phase-saving)
- introduced to CDCL [PipatsrisawatDarwiche2007]
- already used in local search [HirschKojevnikov2001]
Heuristics: Phase-saving

Selecting the last implied value remembers solved components

negative branching

phase-saving
Restarts

Restarts in CDCL solvers:
- Counter heavy-tail behavior [GomesSelmanCrato’97]
- Unassign all variables but keep the (dynamic) heuristics

Restart strategies: [Walsh’99, LubySinclairZuckerman’93]
- Geometrical restart: e.g. 100, 150, 225, 333, 500, 750, …
- Luby sequence: e.g. 100, 100, 200, 100, 100, 200, 400, …

Rapid restarts by reusing trail: [vanderTakHeuleRamos’11]
- Partial restart same effect as full restart
- Optimal strategy Luby-1: 1, 1, 2, 1, 1, 2, 4, …
Restarts

Restarts in CDCL solvers:
- Counter heavy-tail behavior
- Unassign all variables but keep the (dynamic) heuristics

Restart strategies:
- Geometrical restart: e.g. 100, 150, 225, 333, 500, 750, . . .
- Luby sequence: e.g. 100, 100, 200, 100, 100, 200, 400, . . .

Rapid restarts by reusing trail:
- Partial restart same effect as full restart
- Optimal strategy Luby-1: 1, 1, 2, 1, 1, 2, 4, . . .
## Restarts

### Restarts in CDCL solvers:
- Counter heavy-tail behavior \[\text{[GomesSelmanCrato’97]}\]
- Unassign all variables but keep the (dynamic) heuristics

### Restart strategies: \[\text{[Walsh’99, LubySinclairZuckerman’93]}\]
- Geometrical restart: e.g. 100, 150, 225, 333, 500, 750, …
- Luby sequence: e.g. 100, 100, 200, 100, 100, 200, 400, …

### Rapid restarts by reusing trail: \[\text{[vanderTakHeuleRamos’11]}\]
- Partial restart same effect as full restart
- Optimal strategy Luby-1: 1, 1, 2, 1, 1, 2, 4, …
Preliminary CDCL solver in ACL2

“Don’t be smart”
Removal of false literals in ACL2

(defun neg (literal) (* -1 literal))

(defun false–literal (assignment literal)
  (member (neg literal) assignment))

(defun one–not–false–literal (assignment clause)
  (cond ((atom clause) nil)
        ((false–literal assignment (car clause))
         (one–not–false–literal assignment (cdr clause)))
        (t clause)))

(defun two–not–false–literals (assignment clause)
  (cond ((atom clause) nil)
        ((false–literal assignment (car clause))
         (two–not–false–literals assignment (cdr clause)))
        (t (cons (car clause)
                 (one–not–false–literal assignment (cdr clause)))))))
Unit clause is member of all-lits in ACL2

(defun all-lits (formula)
  (if (atom formula)
      nil
      (append (car formula) (all-lits (cdr formula))))

(defun reduced-clause-implies-member-car-reduced-clause
  (implies (two-not-false-literals-assignment-clause)
    (member (car (two-not-false-literals-assignment-clause)) clause)))

(defun member-append-member-or
  (iff (member x (append y z))
    (or (member x y) (member x z))))

(defun reduced-clause-implies-member-car-all-lits
  (implies (and (two-not-false-literals-assignment-clause)
    (member clause formula))
    (member (car (two-not-false-literals-assignment-clause))
      (all-lits formula))))
The new get-unit procedure in ACL2

```lisp
(defun get-unit (formula assignment)
  (if (atom formula)
      (mv nil nil)
    (let ((reduced-clause (two-not-false-literals assignment (car formula))))
      (cond ((not reduced-clause) (mv (car formula) nil))
            ((and (car reduced-clause)
                   (not (cdr reduced-clause))
                   (not (member (car reduced-clause) assignment)))
               (mv (car formula) (car reduced-clause)))
            (t (get-unit (cdr formula) assignment))))))

(defthm get-unit-returns-member-of-all-lits
  (implies (cadr (get-unit formula assignment))
   (member (cadr (get-unit formula assignment))
           (all-lits formula))))
```
Old unit propagation code in ACL2

(defun neg (literal) (* –1 literal))

(defun reduce–clause (assignment clause unassigned)
  (cond ((atom clause) unassigned)
        ((member (neg (car clause)) assignment)
         (reduce–clause assignment (cdr clause) unassigned))
        (unassigned (append unassigned clause))
        (t (reduce–clause assignment (cdr clause) (list (car clause))))))

(defun get–unit (formula assignment)
  (if (atom formula)
      (mv nil nil)
      (let ((reduced–clause (reduce–clause assignment (car formula) nil)))
       (if (and (not (cdr reduced–clause)) ; if unit and not satisfied
                (not (member (car reduced–clause) assignment)))
        (mv (car formula) (car reduced–clause))
        (get–unit (cdr formula) assignment))))
Reduction theorem and some defuns in ACL2

(defthm new-element-reduces-difference
  (implies (and (member e y)
                (not (member e x)))
     (< (len (set-difference-equal y (cons e x)))
       (len (set-difference-equal y x))))))

(defun remove-literal (clause literal)
  (cond ((atom clause) clause)
        ((eql (car clause) literal) (cdr clause))
        (t (cons (car clause) (remove-literal (cdr clause) literal))))))

(defun resolve (clause resolvent literal)
  (union-equal (remove-literal clause literal)
               (remove-literal resolvent (neg literal))))

(defun unit-under-assignment (assignment clause)
  (and (car (two-not-false-literals assignment clause))
       (not (cdr (two-not-false-literals assignment clause))))))
(defun implications—or—resolvent (formula assignment implications)
  (declare (xargs :measure (nfix (len
       (set—difference—equal (all—lits formula) implications))))
 (mv—let (clause literal)
      (get—unit formula (append assignment implications))
      (if (not literal) ; end recursion
           (if clause (mv nil clause) (mv implications nil))
      (mv—let (more—implications resolvent)
       (implications—or—resolvent formula assignment
        (cons literal implications))
      (if more—implications
           (mv more—implications nil)
       (if (or (unit—under—assignment assignment resolvent)
                (not (member (neg literal) resolvent)))
            (mv nil resolvent)
       (mv nil (resolve clause resolvent literal))))))))
Old code of first unique implication point in ACL2

(defun implications—or—resolvent (formula assignment implications)
  (mv—let (clause literal)
    (get—unit formula (append assignment implications))
    (if (not literal) ; no unit means either conflict or done
      (mv implications clause)
      (mv—let (more—implications resolvent)
        (implications—or—resolvent formula assignment
         (cons literal implications))
        (if (and (member (neg literal) resolvent)
          (cadr (two—not—false—literals assignment resolvent)))
          (mv nil (resolve clause resolvent literal))
          (mv more—implications resolvent))))))
get-decision in ACL2

(defun get-decision (heuristics assignment)
  (if (atom heuristics)
      nil
      (if (or (member (car heuristics) assignment)
              (member (neg (car heuristics)) assignment))
        (get-decision (cdr heuristics) assignment)
        (list (car heuristics))))

(defthm get-decision-returns-not-member-assignment
  (implies (get-decision heuristics assignment)
           (not (member (car (get-decision heuristics assignment)) assignment))))
car get-decision member of implications in ACL2

(defthm cons–subsetp–lemma
  (implies (subsetp x lst)
    (subsetp x (cons y lst))))

(defthm decision–subsetp–of–implications
  (implies (car (implications–or–resolvent f a d))
    (subsetp d (car (implications–or–resolvent f a d))))))

(defthm subsetp–car–member
  (implies (and (consp x)
      (subsetp x y))
    (member (car x) y)))

(defthm car–get–decision–member–car–implications
  (implies (and (consp d)
      (car (implications–or–resolvent f a d)))
    (member (car d) (car (implications–or–resolvent f a d))))))
get-decision-and-implication-reduce-set-difference in ACL2

(defthm member-not-member-reduce-set-difference
  (implies (and (member (car get-d) h)
                 (member (car get-d) i)
                 (not (member (car get-d) a)))
           (< (len (set-difference-equal h (append a i)))
               (len (set-difference-equal h a))))))

(defthm get-decision-and-implication-reduce-set-difference
  (implies (and (get-decision h a)
                 (car (implications-or-resolvent f a (get-decision h a))))
           (and (member (car (get-decision h a)) h)
                (member (car (get-decision h a))
                         (car (implications-or-resolvent f a (get-decision h a))))
                (not (member (car (get-decision h a)) a)))
           (< (len (set-difference-equal h (append a (car (implications-or-resolvent f a (get-decision h a))))))
               (len (set-difference-equal h a))))))
(defun assign-rec (f h a)
  (declare (xargs :measure (nfix (len (set-difference-equal h a)))))
  (let ((decision (get-decision h a)))
    (if (not decision)
      (mv assignment nil) ; found a solution -> satisfiable
      (mv-let (implications resolvent)
        (implications-or-resolvent f a decision)
        (if implications
          (assign-rec f h (append a implications))
          (mv nil resolvent))))))

(defun solution-or-resolvent (formula heuristics)
  (mv-let (assignment resolvent)
    (implications-or-resolvent formula nil nil)
    (if resolvent
      (mv nil nil) ; found refutation -> unsatisfiable
      (assign-rec formula heuristics assignment))))
Top level structure CDCL in ACL2

\textbf{(defun heuristics-init \{formula\})}
\hspace{1cm} (all-lits formula))

(skip-proofs)
\textbf{(defun cdcl-rec \{formula\} \{heuristics\} ; returns solution or unsatisfiable)}
\hspace{1cm} (mv-let (solution resolvent)
\hspace{2cm} (solution-or-resolvent \{formula\} \{heuristics\})
\hspace{2cm} (cond (resolvent (cdcl-rec (cons resolvent \{formula\}) \{heuristics\}))
\hspace{3cm} (solution solution) ; found solution
\hspace{3cm} (t 'unsatisfiable))))) ; found refutation
)

\textbf{(defun cdcl \{formula\})}
\hspace{1cm} (cdcl-rec \{formula\} (heuristics-init \{formula\})))
Search for Simplification
Variable Elimination
Variable Elimination [DavisPutnam’60]

Definition (Resolution)
Given two clauses $C = (x \lor a_1 \lor \cdots \lor a_i)$ and $D = (\overline{x} \lor b_1 \lor \cdots \lor b_j)$, the resolvent of $C$ and $D$ on variable $x$ (denoted by $C \otimes_x D$) is $(a_1 \lor \cdots \lor a_i \lor b_1 \lor \cdots \lor b_j)$

Resolution on sets of clauses $F_x$ and $F_{\overline{x}}$ (denoted by $F_x \otimes_x F_{\overline{x}}$) generates all (non-tautological) resolvents of $C \in F_x$ and $D \in F_{\overline{x}}$.

Definition (Variable elimination (VE))
Given a CNF formula $F$, variable elimination (or DP resolution) removes a variable $x$ by replacing $F_x$ and $F_{\overline{x}}$ by $F_x \otimes_x F_{\overline{x}}$

Proof procedure [DavisPutnam60]
VE is a complete proof procedure. Applying VE until fixpoint results in the empty formula (satisfiable) or empty clause (unsatisfiable)
Variable Elimination [DavisPutnam’60]

Definition (Resolution)

Given two clauses \( C = (x \lor a_1 \lor \cdots \lor a_i) \) and \( D = (\overline{x} \lor b_1 \lor \cdots \lor b_j) \), the \textit{resolvent} of \( C \) and \( D \) on variable \( x \) (denoted by \( C \otimes_x D \)) is

\[
(a_1 \lor \cdots \lor a_i \lor b_1 \lor \cdots \lor b_j)
\]

Resolution on sets of clauses \( F_x \) and \( F_{\overline{x}} \) (denoted by \( F_x \otimes_x F_{\overline{x}} \)) generates all (non-tautological) resolvents of \( C \in F_x \) and \( D \in F_{\overline{x}} \).

Definition (Variable elimination (VE))

Given a CNF formula \( F \), \textit{variable elimination} (or DP resolution) removes a variable \( x \) by replacing \( F_x \) and \( F_{\overline{x}} \) by \( F_x \otimes_x F_{\overline{x}} \).

Proof procedure [DavisPutnam60]

VE is a complete proof procedure. Applying VE until fixpoint results in the empty formula (satisfiable) or empty clause (unsatisfiable).
Variable Elimination [DavisPutnam’60]

Definition (Resolution)

Given two clauses $C = (x \lor a_1 \lor \cdots \lor a_i)$ and $D = (\overline{x} \lor b_1 \lor \cdots \lor b_j)$, the resolvent of $C$ and $D$ on variable $x$ (denoted by $C \otimes_x D$) is $(a_1 \lor \cdots \lor a_i \lor b_1 \lor \cdots \lor b_j)$

Resolution on sets of clauses $F_x$ and $F_{\overline{x}}$ (denoted by $F_x \otimes_x F_{\overline{x}}$) generates all (non-tautological) resolvents of $C \in F_x$ and $D \in F_{\overline{x}}$.

Definition (Variable elimination (VE))

Given a CNF formula $F$, variable elimination (or DP resolution) removes a variable $x$ by replacing $F_x$ and $F_{\overline{x}}$ by $F_x \otimes_x F_{\overline{x}}$

Proof procedure [DavisPutnam60]

VE is a complete proof procedure. Applying VE until fixpoint results in the empty formula (satisfiable) or empty clause (unsatisfiable)
Example VE by clause distribution [DavisPutnam’60]

**Definition (Variable elimination (VE))**

Given a CNF formula $F$, *variable elimination* (or DP resolution) removes a variable $x$ by replacing $F_x$ and $F_{\bar{x}}$ by $F_x \otimes_x F_{\bar{x}}$

**Example of clause distribution**

$$
\begin{array}{ccc}
\hline
F_x & \hline
(x \lor c) & (x \lor \bar{d}) & (x \lor \bar{a} \lor \bar{b}) \\
\hline
F_{\bar{x}} \left\{ \\
(\bar{x} \lor a) & (a \lor c) & (a \lor d) & (a \lor \bar{a} \lor \bar{b}) \\
(\bar{x} \lor b) & (b \lor c) & (b \lor d) & (b \lor \bar{a} \lor \bar{b}) \\
(\bar{x} \lor \bar{e} \lor f) & (c \lor \bar{e} \lor f) & (d \lor \bar{e} \lor f) & (\bar{a} \lor \bar{b} \lor \bar{e} \lor f) \\
\hline
\end{array}
$$

example: $|F_x \otimes F_{\bar{x}}| > |F_x| + |F_{\bar{x}}|$; in general: exponential growth of clauses
Example VE by clause distribution [DavisPutnam’60]

Definition (Variable elimination (VE))

Given a CNF formula $F$, variable elimination (or DP resolution) removes a variable $x$ by replacing $F_x$ and $F_{\bar{x}}$ by $F_x \otimes_x F_{\bar{x}}$

Example of clause distribution

<table>
<thead>
<tr>
<th>$F_x$</th>
<th>$F_{\bar{x}}$</th>
<th>$\times$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x \lor c)$</td>
<td>$(\bar{x} \lor a)$</td>
<td>$(\bar{x} \lor b)$</td>
</tr>
<tr>
<td>$(x \lor \bar{d})$</td>
<td>$(a \lor c)$</td>
<td>$(b \lor c)$</td>
</tr>
<tr>
<td>$(x \lor \bar{a} \lor \bar{b})$</td>
<td>$(a \lor d)$</td>
<td>$(b \lor d)$</td>
</tr>
<tr>
<td></td>
<td>$(a \lor \bar{a} \lor \bar{b})$</td>
<td>$(b \lor \bar{a} \lor \bar{b})$</td>
</tr>
<tr>
<td></td>
<td>$(\bar{a} \lor \bar{b} \lor \bar{e} \lor f)$</td>
<td></td>
</tr>
</tbody>
</table>

example: $|F_x \otimes F_{\bar{x}}| > |F_x| + |F_{\bar{x}}|$; in general: exponential growth of clauses
Example VE by clause distribution [DavisPutnam’60]

Definition (Variable elimination (VE))

Given a CNF formula $F$, variable elimination (or DP resolution) removes a variable $x$ by replacing $F_x$ and $F_{\bar{x}}$ by $F_x \otimes_x F_{\bar{x}}$

Example of clause distribution

<table>
<thead>
<tr>
<th>$F_{\bar{x}}$</th>
<th>$F_x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>($\bar{x} \lor a$)</td>
<td>($x \lor c$)</td>
</tr>
<tr>
<td>($\bar{x} \lor b$)</td>
<td>($x \lor \bar{d}$)</td>
</tr>
<tr>
<td>($\bar{x} \lor \bar{e} \lor f$)</td>
<td>($x \lor \bar{a} \lor \bar{b}$)</td>
</tr>
<tr>
<td>($a \lor c$)</td>
<td>($a \lor d$)</td>
</tr>
<tr>
<td>($b \lor c$)</td>
<td>($b \lor d$)</td>
</tr>
<tr>
<td>($\bar{a} \lor \bar{b} \lor \bar{e} \lor f$)</td>
<td>($a \lor \bar{a} \lor \bar{b}$)</td>
</tr>
<tr>
<td>($b \lor \bar{a} \lor \bar{b}$)</td>
<td></td>
</tr>
</tbody>
</table>

Example: $|F_x \otimes F_{\bar{x}}| > |F_x| + |F_{\bar{x}}|; \text{ in general: exponential growth of clauses}$
Definition (Variable elimination (VE))

Given a CNF formula $F$, variable elimination (or DP resolution) removes a variable $x$ by replacing $F_x$ and $F_{\overline{x}}$ by $F_x \otimes F_{\overline{x}}$

Example of clause distribution

<table>
<thead>
<tr>
<th>$F_{\overline{x}}$</th>
<th>$F_x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\overline{x} \lor a)$</td>
<td>$(x \lor c)$</td>
</tr>
<tr>
<td>$(\overline{x} \lor b)$</td>
<td>$(x \lor \overline{d})$</td>
</tr>
<tr>
<td>$(\overline{x} \lor \overline{e} \lor f)$</td>
<td>$(x \lor \overline{a} \lor \overline{b})$</td>
</tr>
</tbody>
</table>

Example: $|F_x \otimes F_{\overline{x}}| > |F_x| + |F_{\overline{x}}|$; in general: exponential growth of clauses
General idea

Detect gates (or definitions) \( x = \text{GATE}(a_1, \ldots, a_n) \) in the formula and use them to reduce the number of added clauses.

Possible gates

<table>
<thead>
<tr>
<th>gate</th>
<th>( G_x )</th>
<th>( G_{\overline{x}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND ((a_1, \ldots, a_n))</td>
<td>((x \lor \overline{a}_1 \lor \cdots \lor \overline{a}_n))</td>
<td>((\overline{x} \lor a_1), \ldots, (\overline{x} \lor a_n))</td>
</tr>
<tr>
<td>OR ((a_1, \ldots, a_n))</td>
<td>((x \lor \overline{a}_1), \ldots, (x \lor \overline{a}_n))</td>
<td>((\overline{x} \lor a_1 \lor \cdots \lor a_n))</td>
</tr>
<tr>
<td>ITE ((c, t, f))</td>
<td>((x \lor \overline{c} \lor \overline{t}), (x \lor c \lor \overline{f}))</td>
<td>((\overline{x} \lor \overline{c} \lor t), (\overline{x} \lor c \lor f))</td>
</tr>
</tbody>
</table>

Variable elimination by substitution [EenBiere07]

Let \( R_x = F_x \setminus G_x \); \( R_{\overline{x}} = F_{\overline{x}} \setminus G_{\overline{x}} \).

Replace \( F_x \land F_{\overline{x}} \) by \( G_x \otimes_x R_{\overline{x}} \land G_{\overline{x}} \otimes_x R_x \).
VE by substitution \cite{EenBi07}

**General idea**
Detect gates (or definitions) \( x = \text{GATE}(a_1, \ldots, a_n) \) in the formula and use them to reduce the number of added clauses

**Possible gates**

<table>
<thead>
<tr>
<th>gate</th>
<th>( G_x )</th>
<th>( G_{\overline{x}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{AND}(a_1, \ldots, a_n)</td>
<td>( (x \lor \overline{a}_1 \lor \cdots \lor \overline{a}_n) )</td>
<td>( (\overline{x} \lor a_1), \ldots, (\overline{x} \lor a_n) )</td>
</tr>
<tr>
<td>\text{OR}(a_1, \ldots, a_n)</td>
<td>( (x \lor \overline{a}_1), \ldots, (x \lor \overline{a}_n) )</td>
<td>( (\overline{x} \lor a_1 \lor \cdots \lor a_n) )</td>
</tr>
<tr>
<td>\text{ITE}(c, t, f)</td>
<td>( (x \lor \overline{c} \lor \overline{t}), (x \lor c \lor \overline{f}) )</td>
<td>( (\overline{x} \lor \overline{c} \lor t), (\overline{x} \lor c \lor f) )</td>
</tr>
</tbody>
</table>

**Variable elimination by substitution \cite{EenBi07}**

Let \( R_x = F_x \setminus G_x \); \( R_{\overline{x}} = F_{\overline{x}} \setminus G_{\overline{x}} \).

Replace \( F_x \land F_{\overline{x}} \) by \( G_x \otimes_x R_{\overline{x}} \land G_{\overline{x}} \otimes_x R_x \).
VE by substitution [EenBiere07]

General idea

Detect gates (or definitions) $x = GATE(a_1, \ldots, a_n)$ in the formula and use them to reduce the number of added clauses.

Possible gates

<table>
<thead>
<tr>
<th>gate</th>
<th>$G_x$</th>
<th>$G_{\bar{x}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND($a_1, \ldots, a_n$)</td>
<td>$(x \lor \bar{a}_1 \lor \cdots \lor \bar{a}_n)$</td>
<td>$(\bar{x} \lor a_1), \ldots, (\bar{x} \lor a_n)$</td>
</tr>
<tr>
<td>OR($a_1, \ldots, a_n$)</td>
<td>$(x \lor \bar{a}_1), \ldots, (x \lor \bar{a}_n)$</td>
<td>$(\bar{x} \lor a_1 \lor \cdots \lor a_n)$</td>
</tr>
<tr>
<td>ITE($c, t, f$)</td>
<td>$(x \lor \bar{c} \lor \bar{t}), (x \lor c \lor \bar{f})$</td>
<td>$(\bar{x} \lor \bar{c} \lor t), (\bar{x} \lor c \lor f)$</td>
</tr>
</tbody>
</table>

Variable elimination by substitution [EenBiere07]

Let $R_x = F_x \setminus G_x$; $R_{\bar{x}} = F_{\bar{x}} \setminus G_{\bar{x}}$.

Replace $F_x \land F_{\bar{x}}$ by $G_x \otimes_x R_{\bar{x}} \land G_{\bar{x}} \otimes_x R_x$. 

VE by substitution [EenBiere’07]

Example of gate extraction: $x = \text{AND}(a, b)$

$$F_x = (x \lor c) \land (x \lor \bar{d}) \land (x \lor \bar{a} \lor \bar{b})$$

$$F_{\bar{x}} = (\bar{x} \lor a) \land (\bar{x} \lor b) \land (\bar{x} \lor \bar{e} \lor f)$$

Example of substitution

<table>
<thead>
<tr>
<th>$R_x$</th>
<th>$G_x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x \lor c)$</td>
<td>$(x \lor \bar{d})$</td>
</tr>
<tr>
<td>$(a \lor c)$</td>
<td>$(a \lor d)$</td>
</tr>
<tr>
<td>$(b \lor c)$</td>
<td>$(b \lor d)$</td>
</tr>
<tr>
<td>$\bar{a} \lor \bar{b} \lor \bar{e} \lor f$</td>
<td></td>
</tr>
</tbody>
</table>

using substitution: $|F_x \otimes F_{\bar{x}}| < |F_x| + |F_{\bar{x}}|$
**Example of gate extraction:** \( x = \text{AND}(a, b) \)

\[
F_x = (x \lor c) \land (x \lor \bar{d}) \land (x \lor \bar{a} \lor \bar{b}) \\
F_{\bar{x}} = (\bar{x} \lor a) \land (\bar{x} \lor b) \land (\bar{x} \lor \bar{e} \lor f)
\]

**Example of substitution**

<table>
<thead>
<tr>
<th>( G_{\bar{x}} )</th>
<th>( R_x )</th>
<th>( G_x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(( \bar{x} \lor a )) \smallblue{(x \lor \bar{d})} \smallgreen{(x \lor \bar{a} \lor \bar{b})}</td>
<td>(a \lor c) \smallblue{(a \lor d)}</td>
<td>(a \lor d)</td>
</tr>
<tr>
<td>(( \bar{x} \lor b ))</td>
<td>(b \lor c)</td>
<td>(b \lor d)</td>
</tr>
<tr>
<td>(( \bar{x} \lor \bar{e} \lor f ))</td>
<td></td>
<td>(( \bar{a} \lor \bar{b} \lor \bar{e} \lor f ))</td>
</tr>
</tbody>
</table>

using substitution: \( |F_x \otimes F_{\bar{x}}| < |F_x| + |F_{\bar{x}}| \)
**VE by substitution** [EenBiere’07]

### Example of gate extraction: $x = \text{AND}(a, b)$

\[
F_x = (x \lor c) \land (x \lor \overline{d}) \land (x \lor \overline{a} \lor \overline{b}) \\
F_{\overline{x}} = (\overline{x} \lor a) \land (\overline{x} \lor b) \land (\overline{x} \lor \overline{e} \lor f)
\]

### Example of substitution

<table>
<thead>
<tr>
<th>$G_{\overline{x}}$</th>
<th>$R_{\overline{x}}$</th>
<th>$G_x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\overline{x} \lor a)$</td>
<td>$(\overline{x} \lor \overline{a})$</td>
<td>$(x \lor \overline{a} \lor \overline{b})$</td>
</tr>
<tr>
<td>$(\overline{x} \lor b)$</td>
<td>$(\overline{x} \lor b)$</td>
<td>$(x \lor \overline{a} \lor \overline{b})$</td>
</tr>
<tr>
<td>$(\overline{x} \lor \overline{e} \lor f)$</td>
<td>$(\overline{x} \lor \overline{e} \lor f)$</td>
<td>$(x \lor \overline{a} \lor \overline{b})$</td>
</tr>
</tbody>
</table>

Using substitution: $|F_x \otimes F_{\overline{x}}| < |F_x| + |F_{\overline{x}}|$
Blocked Clause Elimination
**Blocked Clauses** [Kullmann’99]

**Definition (Blocking literal)**

A literal \( l \) in a clause \( C \) of a CNF \( F \) blocks \( C \) w.r.t. \( F \) if for every clause \( C' \in F \) with \( \overline{l} \in C' \), the resolvent \((C \setminus \{l\}) \cup (C' \setminus \{l\})\) obtained from resolving \( C \) and \( C' \) on \( l \) is a tautology.

With respect to a fixed CNF and its clauses we have:

**Definition (Blocked clause)**

A clause is blocked if it contains a literal that blocks it.

**Example**

Consider the formula \((a \lor b) \land (a \lor \overline{b} \lor \overline{c}) \land (\overline{a} \lor c)\).

First clause is not blocked.

Second clause is blocked by both \( a \) and \( \overline{c} \). Third clause is blocked by \( c \).

**Proposition**

Removal of an arbitrary blocked clause preserves satisfiability.
**Blocked Clauses** [Kullmann’99]

**Definition (Blocking literal)**

A literal \( l \) in a clause \( C \) of a CNF \( F \) blocks \( C \) w.r.t. \( F \) if for every clause \( C' \in F \) with \( \bar{l} \in C' \), the resolvent \((C \setminus \{l\}) \cup (C' \setminus \{\bar{l}\})\) obtained from resolving \( C \) and \( C' \) on \( l \) is a tautology.

With respect to a fixed CNF and its clauses we have:

**Definition (Blocked clause)**

A clause is blocked if it contains a literal that blocks it.

**Example**

Consider the formula \((a \lor b) \land (a \lor \bar{b} \lor \bar{c}) \land (\bar{a} \lor c)\).

First clause is not blocked.
Second clause is blocked by both \( a \) and \( \bar{c} \). Third clause is blocked by \( c \).

**Proposition**

Removal of an arbitrary blocked clause preserves satisfiability.
Blocked Clauses  [Kullmann’99]

Definition (Blocking literal)

A literal $l$ in a clause $C$ of a CNF $F$ blocks $C$ w.r.t. $F$ if for every clause $C' \in F$ with $\bar{l} \in C'$, the resolvent $(C \setminus \{l\}) \cup (C' \setminus \{\bar{l}\})$ obtained from resolving $C$ and $C'$ on $l$ is a tautology.

With respect to a fixed CNF and its clauses we have:

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Example

Consider the formula $(a \lor b) \land (a \lor \bar{b} \lor \bar{c}) \land (\bar{a} \lor c)$.  
First clause is not blocked.  
Second clause is blocked by both $a$ and $\bar{c}$.  
Third clause is blocked by $c$.

Proposition

Removal of an arbitrary blocked clause preserves satisfiability.
Blocked Clause Elimination (BCE)

**Definition (BCE)**

While there is a blocked clause $C$ in a CNF $F$, remove $C$ from $F$.

**Example**

Consider $(a \lor b) \land (a \lor \overline{b} \lor \overline{c}) \land (\overline{a} \lor c)$. After removing either $(a \lor \overline{b} \lor \overline{c})$ or $(\overline{a} \lor c)$, the clause $(a \lor b)$ becomes blocked (no clause with either $\overline{b}$ or $\overline{a}$). An extreme case in which BCE removes all clauses of a formula!

**Proposition**

BCE is confluent, i.e., has a unique fixpoint

- Blocked clauses stay blocked w.r.t. removal
Blocked Clause Elimination (BCE)

**Definition (BCE)**
While there is a blocked clause $C$ in a CNF $F$, remove $C$ from $F$.

**Example**
Consider $(a \lor b) \land (a \lor \lnot b \lor \lnot c) \land (\lnot a \lor c)$. After removing either $(a \lor \lnot b \lor \lnot c)$ or $(\lnot a \lor c)$, the clause $(a \lor b)$ becomes blocked (no clause with either $\lnot b$ or $\lnot a$). An extreme case in which BCE removes all clauses of a formula!

**Proposition**
BCE is confluent, i.e., has a unique fixpoint

- Blocked clauses stay blocked w.r.t. removal
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[
\begin{align*}
(c_1) &\quad (t_1 \lor \neg t_3 \lor \neg c_0) \\
(\neg c_1 \lor t_1 \lor \neg t_2) &\quad (\neg t_1 \lor t_3) \\
(c_1 \lor \neg t_1) &\quad (\neg t_1 \lor c_0) \\
(c_1 \lor \neg t_2) &\quad (t_2 \lor \neg a_0 \lor \neg b_0) \\
(\neg o_0 \lor t_3 \lor c_0) &\quad (\neg t_2 \lor a_0) \\
(\neg o_0 \lor \neg t_3 \lor \neg c_0) &\quad (\neg t_2 \lor b_0) \\
(o_0 \lor t_3 \lor \neg c_0) &\quad (\neg t_3 \lor \neg a_0 \lor \neg b_0) \\
(o_0 \lor \neg t_3 \lor c_0) &\quad (\neg t_3 \lor a_0 \lor \neg b_0) \\
(\neg o_0 \lor \neg t_3 \lor \neg c_0) &\quad (t_3 \lor a_0 \lor \neg b_0) \\
(o_0 \lor \neg t_3 \lor c_0) &\quad (t_3 \lor \neg a_0 \lor b_0)
\end{align*}
\]
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[
\begin{align*}
(c_1) & \quad (t_1 \lor \neg t_2 \lor \neg c_0) \\
(\neg c_1 \lor t_1 \lor \neg t_2) & \quad (\neg t_1 \lor t_3) \\
(c_1 \lor \neg t_1) & \quad (\neg t_1 \lor c_0) \\
(c_1 \lor t_2) & \quad (t_2 \lor \neg a_0 \lor \neg b_0) \\
(\neg o_0 \lor t_3 \lor c_0) & \quad (\neg t_3 \lor \neg a_0 \lor \neg b_0) \\
(\neg o_0 \lor \neg t_3 \lor \neg c_0) & \quad (t_3 \lor a_0 \lor \neg b_0) \\
(o_0 \lor t_3 \lor \neg c_0) & \quad (\neg t_3 \lor \neg a_0 \lor \neg b_0) \\
(o_0 \lor \neg t_3 \lor c_0) & \quad (t_3 \lor a_0 \lor b_0) \\
\end{align*}
\]
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[
\begin{align*}
(c_1) & \quad (t_1 \lor \neg t_3 \lor \neg c_0) \\
(\neg c_1 \lor t_1 \lor \neg t_2) & \quad (\neg t_1 \lor t_3) \\
(c_1 \lor \neg t_1) & \quad (\neg t_1 \lor c_0) \\
(c_1 \lor t_2) & \quad (t_2 \lor \neg a_0 \lor \neg b_0) \\
(\neg o_0 \lor t_3 \lor \neg c_0) & \quad (\neg t_3 \lor \neg a_0 \lor \neg b_0) \\
(\neg o_0 \lor \neg t_3 \lor \neg c_0) & \quad (\neg t_3 \lor \neg a_0 \lor \neg b_0) \\
(o_0 \lor t_3 \lor \neg c_0) & \quad (t_3 \lor a_0 \lor \neg b_0) \\
(o_0 \lor \neg t_3 \lor \neg c_0) & \quad (t_3 \lor \neg a_0 \lor b_0)
\end{align*}
\]
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[(c_1)\]
\[(\neg c_1 \lor t_1 \lor \neg t_2)\]
\[(c_1 \lor \neg t_1)\]
\[(c_1 \lor t_2)\]
\[(\neg o_0 \lor t_3 \lor c_0)\]
\[(\neg o_0 \lor \neg t_3 \lor \neg c_0)\]
\[(o_0 \lor t_3 \lor \neg c_0)\]
\[(o_0 \lor \neg t_3 \lor c_0)\]

\[(t_1 \lor \neg t_3 \lor c_0)\]
\[(\neg t_1 \lor t_3)\]
\[(\neg t_1 \lor c_0)\]
\[(t_2 \lor \neg a_0 \lor \neg b_0)\]
\[(\neg t_2 \lor a_0)\]
\[(\neg t_2 \lor b_0)\]

\[\neg (t_1 \lor \neg t_3 \lor \neg c_0)\]
\[\neg (t_1 \lor \neg t_3 \lor c_0)\]
\[\neg (t_2 \lor \neg a_0 \lor \neg b_0)\]
\[\neg (t_2 \lor a_0 \lor \neg b_0)\]
\[\neg (t_2 \lor a_0 \lor b_0)\]
\[\neg (t_3 \lor \neg a_0 \lor \neg b_0)\]
\[\neg (t_3 \lor a_0 \lor \neg b_0)\]
\[\neg (t_3 \lor a_0 \lor b_0)\]
\[t_1 \lor \neg t_3 \lor \neg c_0\]
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[
\begin{align*}
(c_1) & \quad (t_1 \lor \neg t_3 \lor \neg c_0) \\
(\neg c_1 \lor t_1 \lor \neg t_2) & \quad (\neg t_1 \lor t_3) \\
(c_1 \lor \neg t_1) & \quad (\neg t_1 \lor c_0) \\
(c_1 \lor t_2) & \quad (t_2 \lor \neg a_0 \lor \neg b_0) \\
(\neg o_0 \lor t_3 \lor c_0) & \quad (\neg t_2 \lor a_0) \\
(\neg o_0 \lor \neg t_3 \lor \neg c_0) & \quad (\neg t_2 \lor b_0) \\
(o_0 \lor t_3 \lor c_0) & \quad (\neg o_0 \lor \neg t_3 \lor c_0) \\
(o_0 \lor t_3 \lor \neg c_0) & \quad (\neg o_0 \lor t_3 \lor \neg c_0) \\
(o_0 \lor \neg t_3 \lor c_0) & \quad (\neg t_3 \lor a_0 \lor \neg b_0) \\
(o_0 \lor \neg t_3 \lor \neg c_0) & \quad (\neg t_3 \lor \neg a_0 \lor \neg b_0) \\
(t_3 \lor a_0 \lor \neg b_0) & \quad (t_3 \lor \neg a_0 \lor b_0) \\
(t_3 \lor \neg a_0 \lor \neg b_0) & \quad (t_3 \lor \neg a_0 \lor b_0)
\end{align*}
\]
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[
\begin{align*}
(c_1) & \quad (t_1 \lor t_3 \lor \neg c_0) \\
(\neg c_1 \lor t_1 \lor \neg t_2) & \quad (\neg t_1 \lor t_3) \\
(c_1 \lor \neg t_1) & \quad (\neg t_1 \lor c_0) \\
(c_1 \lor t_2) & \quad (t_2 \lor \neg a_0 \lor \neg b_0) \\
(\neg o_0 \lor t_3 \lor c_0) & \quad (\neg t_2 \lor a_0) \\
(\neg o_0 \lor \neg t_3 \lor \neg c_0) & \quad (\neg t_2 \lor b_0) \\
(o_0 \lor t_3 \lor \neg c_0) & \quad (\neg t_3 \lor \neg a_0 \lor \neg b_0) \\
(o_0 \lor \neg t_3 \lor \neg c_0) & \quad (t_3 \lor \neg a_0 \lor \neg b_0) \\
(t_3 \lor \neg a_0 \lor b_0) & \quad (t_3 \lor a_0 \lor \neg b_0) \\
\end{align*}
\]
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[
\begin{align*}
(c_1) & \\
(\neg c_1 \lor t_1 \lor \neg t_2) & \\
(c_1 \lor \neg t_1) & \\
(c_1 \lor t_2) & \\
(\neg o_0 \lor t_3 \lor c_0) & \\
(\neg o_0 \lor \neg t_3 \lor \neg c_0) & \\
(o_0 \lor t_3 \lor \neg c_0) & \\
(o_0 \lor \neg t_3 \lor c_0) & \\
\end{align*}
\]

\[
\begin{align*}
(t_1 \lor t_3 \lor \neg c_0) & \\
(\neg t_1 \lor t_3) & \\
(\neg t_1 \lor c_0) & \\
(t_2 \lor \neg a_0 \lor \neg b_0) & \\
(\neg t_2 \lor a_0) & \\
(\neg t_2 \lor b_0) & \\
(\neg t_3 \lor a_0 \lor b_0) & \\
(\neg t_3 \lor \neg a_0 \lor \neg b_0) & \\
(t_3 \lor a_0 \lor \neg b_0) & \\
(t_3 \lor \neg a_0 \lor b_0) & \\
\end{align*}
\]
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[
\begin{align*}
(c_1) & \quad (t_1 \lor t_3 \lor c_0) \\
(\neg c_1 \lor t_1 \lor t_2) & \quad (\neg t_1 \lor t_3) \\
(c_1 \lor \neg t_1) & \quad (\neg t_1 \lor c_0) \\
(c_1 \lor t_2) & \quad (t_2 \lor a_0 \lor \neg b_0) \\
(\neg o_0 \lor t_3 \lor c_0) & \quad (\neg t_3 \lor a_0 \lor \neg b_0) \\
(\neg o_0 \lor \neg t_3 \lor \neg c_0) & \quad (\neg t_3 \lor \neg a_0 \lor \neg b_0) \\
(o_0 \lor t_3 \lor \neg c_0) & \quad (t_3 \lor a_0 \lor \neg b_0) \\
(o_0 \lor \neg t_3 \lor \neg c_0) & \quad (t_3 \lor a_0 \lor \neg b_0)
\end{align*}
\]
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[
\begin{align*}
(c_1) & \quad (t_1 \lor t_3 \lor \neg c_0) \\
(\neg c_1 \lor t_1 \lor t_2) & \quad (\neg t_1 \lor t_3) \\
(c_1 \lor \neg t_1) & \quad (\neg t_1 \lor c_0) \\
(c_1 \lor t_2) & \quad (t_2 \lor \neg a_0 \lor \neg b_0) \\
(\neg o_0 \lor t_3 \lor c_0) & \quad (\neg t_3 \lor a_0 \lor \neg b_0) \\
(\neg o_0 \lor \neg t_3 \lor \neg c_0) & \quad (\neg t_3 \lor \neg a_0 \lor \neg b_0) \\
(o_0 \lor t_3 \lor \neg c_0) & \quad (t_3 \lor a_0 \lor \neg b_0) \\
(o_0 \lor \neg t_3 \lor \neg c_0) & \quad (t_3 \lor \neg a_0 \lor b_0)
\end{align*}
\]
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[
\begin{align*}
(c_1) & \quad \quad (t_1 \lor t_3 \lor c_0) \\
(\neg c_1 \lor t_1 \lor t_2) & \quad \quad (\neg t_1 \lor t_3) \\
(c_1 \lor \neg t_1) & \quad \quad (\neg t_1 \lor c_0) \\
(c_1 \lor t_2) & \quad \quad (t_2 \lor \neg a_0 \lor \neg b_0) \\
(\neg o_0 \lor t_3 \lor c_0) & \quad \quad (\neg t_3 \lor a_0 \lor \neg b_0) \\
(\neg o_0 \lor \neg t_3 \lor \neg c_0) & \quad \quad (\neg t_3 \lor \neg a_0 \lor \neg b_0) \\
(o_0 \lor t_3 \lor \neg c_0) & \quad \quad (t_3 \lor a_0 \lor \neg b_0) \\
(o_0 \lor \neg t_3 \lor \neg c_0) & \quad \quad (t_3 \lor a_0 \lor \neg b_0)
\end{align*}
\]
BCE very effective on circuits [JärvisaloBiereHeule’10]

BCE converts the Tseiting encoding to Plaisted Greenbaum
BCE simulates Pure literal elimination, Cone of influence and much more

Example of circuit simplification by BCE on CNF

\[(c_1)\]
\[(\neg c_1 \lor t_1 \lor t_2)\]
\[(c_1 \lor \neg t_1)\]
\[(c_1 \lor t_2)\]
\[(\neg o_0 \lor t_3 \lor c_0)\]
\[(\neg o_0 \lor \neg t_3 \lor \neg c_0)\]
\[(o_0 \lor t_3 \lor \neg c_0)\]
\[(o_0 \lor \neg t_3 \lor \neg c_0)\]

\[(t_1 \lor t_3 \lor c_0)\]
\[(\neg t_1 \lor t_3)\]
\[(\neg t_1 \lor c_0)\]
\[(t_2 \lor \neg a_0 \lor \neg b_0)\]
\[(\neg t_2 \lor a_0)\]
\[(\neg t_2 \lor b_0)\]
\[(\neg t_3 \lor a_0 \lor \neg b_0)\]
\[(\neg t_3 \lor \neg a_0 \lor b_0)\]
\[(t_3 \lor a_0 \lor \neg b_0)\]
\[(t_3 \lor \neg a_0 \lor b_0)\]
Unhiding redundancy
Redundancy

Redundant clauses:
- Removal of $C \in F$ preserves unsatisfiability of $F$
- Assign $l \in C$ to false and check for a conflict in $F \setminus \{C\}$

Redundant literals:
- Removal of $l \in C$ preserves satisfiability of $F$
- Assign $l' \in C \setminus \{l\}$ to false and check if $l$ is forced to false

Redundancy elimination during pre- and in-processing
- Distillation [JinSomenzi2005]
- ReVivAl [PietteHamadíSaís2008]
- Unhiding [HeuleJärvisaloBiere2011]
Redundancy

Redundant clauses:
- Removal of $C \in F$ preserves unsatisfiability of $F$
- Assign $l \in C$ to false and check for a conflict in $F \setminus \{C\}$

Redundant literals:
- Removal of $l \in C$ preserves satisfiability of $F$
- Assign $l' \in C \setminus \{l\}$ to false and check if $l$ is forced to false

Redundancy elimination during pre- and in-processing
- Distillation [JinSomenzi2005]
- ReVivAl [PietteHamadiSaïs2008]
- Unhiding [HeuleJärvisaloBiere2011]
Redundancy

Redundant clauses:
- Removal of \( C \in F \) preserves unsatisfiability of \( F \)
- Assign \( I \in C \) to false and check for a conflict in \( F \setminus \{C\} \)

Redundant literals:
- Removal of \( l \in C \) preserves satisfiability of \( F \)
- Assign \( l' \in C \setminus \{l\} \) to false and check if \( l \) is forced to false

Redundancy elimination during pre- and in-processing
- Distillation \([\text{JinSomenzi2005}]\)
- ReVivAl \([\text{PietteHamadiSaîs2008}]\)
- Unhiding \([\text{HeuleJärvisaloBiere2011}]\)
Unhide: Binary implication graph (BIG)

unhide: use the binary clauses to detect redundant clauses and literals

\[(\text{\bar{a}} \lor c) \land (\text{\bar{a}} \lor d) \land (\text{\bar{b}} \lor d) \land (\text{\bar{b}} \lor e) \land \]

\[(\text{\bar{c}} \lor f) \land (\text{\bar{d}} \lor f) \land (\text{\bar{g}} \lor f) \land (\text{\bar{f}} \lor h) \land \]

\[(\text{\bar{g}} \lor h) \land (\text{\bar{a}} \lor \text{\bar{e}} \lor h) \land (\text{\bar{b}} \lor \text{\bar{c}} \lor h) \land (a \lor b \lor c \lor d \lor e \lor f \lor g \lor h)\]

non binary clauses
Unhide: Transitive reduction (TRD)

transitive reduction: remove shortcuts in the binary implication graph

\[(\overline{a} \lor c) \land (\overline{a} \lor d) \land (\overline{b} \lor d) \land (\overline{b} \lor e) \land \]
\[(\overline{c} \lor f) \land (\overline{d} \lor f) \land (\overline{g} \lor f) \land (\overline{f} \lor h) \land \]
\[(\overline{g} \lor h) \land (\overline{a} \lor \overline{e} \lor h) \land (\overline{b} \lor \overline{c} \lor h) \land (a \lor b \lor c \lor d \lor e \lor f \lor g \lor h)\]

TRD

\[g \rightarrow f \rightarrow h\]
Unhide: Hidden tautology elimination (HTE) (1)

HTE removes clauses that are subsumed by an implication in BIG

\[(\overline{a} \lor c) \land (\overline{a} \lor d) \land (\overline{b} \lor d) \land (\overline{b} \lor e) \land \]
\[(\overline{c} \lor f) \land (\overline{d} \lor f) \land (\overline{g} \lor f) \land (\overline{f} \lor h) \land \]
\[\overline{a} \lor \overline{e} \lor h \land (\overline{b} \lor \overline{c} \lor h) \land (a \lor b \lor c \lor d \lor e \lor f \lor g \lor h)\]

HTE

\[a \rightarrow d \rightarrow f \rightarrow h\]
Unhide: Hidden tautology elimination (HTE) (2)

HTE removes clauses that are subsumed by an implication in BIG

\[(\bar{a} \lor c) \land (\bar{a} \lor d) \land (\bar{b} \lor d) \land (\bar{b} \lor e) \land (\bar{c} \lor f) \land (\bar{d} \lor f) \land (\bar{g} \lor f) \land (\bar{f} \lor h) \land (\bar{b} \lor \bar{c} \lor h) \land (a \lor b \lor c \lor d \lor e \lor f \lor g \lor h)\]

HTE

\[c \rightarrow f \rightarrow h\]
Unhide: Hidden literal elimination (HLE)

HLE removes literal using the implication in BIG

\[(\overline{a} \lor c) \land (\overline{a} \lor d) \land (\overline{b} \lor d) \land (\overline{b} \lor e) \land (\overline{c} \lor f) \land (\overline{d} \lor f) \land (\overline{g} \lor f) \land (\overline{f} \lor h) \land (a \lor b \lor c \lor d \lor e \lor f \lor g \lor h)\]

HLE
all but e imply h
also b implies e
Conclusions: state-of-the-art SAT solver

Key contributions to SAT search engine:
- adding conflict clauses (grasp) [Marques-Silva’96]
- restart strategies [GomesSC’97,LubySZ’93]
- 2-watch pointers and VSIDS (zChaff) [MoskewiczMZZM’01]
- efficient implementation (Minisat) [EenSörensson’03]
- variable elimination (SatElite) [EenBiere’05]
- phase-saving (Rsat) [PipatsrisawatDarwiche’07]

Recent progress: pre- and in-processing
- removal of redundant clauses and literals [JinSomenzi’05]
- removal of blocked clauses [JärvisaloBiereHeule’10]
- unhiding redundancy [HeuleJärvisaloBiere’11]
Conclusions: state-of-the-art SAT solver

Key contributions to SAT search engine:

- adding conflict clauses (grasp) [Marques-Silva’96]
- restart strategies [GomesSC’97, LubySZ’93]
- 2-watch pointers and VSIDS (zChaff) [MoskewiczMZZM’01]
- efficient implementation (Minisat) [EenSörensson’03]
- variable elimination (SatElite) [EenBiere’05]
- phase-saving (Rsat) [PipatsrisawatDarwiche’07]

Recent progress: pre- and in-processing

- removal of redundant clauses and literals [JinSomenzi’05]
- removal of blocked clauses [JärvisaloBiereHeule’10]
- unhiding redundancy [HeuleJärvisaloBiere’11]
Cactus plot: Lingeling [Biere’10] contains all features

Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

- Limmat 02
- Zchaff 02
- Berkmin 561 02
- Forklift 03
- Siege 03
- Zchaff 04
- SatELite 05
- Minisat 2.0 06
- Picosat 07
- Rsat 07
- Minisat 2.1 08
- Precosat 09
- Glucose 09
- Clasp 09
- Cryptominisat 10
- Lingeling 10
- Minisat 2.2 10
State-of-the-art SAT Solving

Marijn J. H. Heule

University of Texas

April 16, 2012 @ ACL2