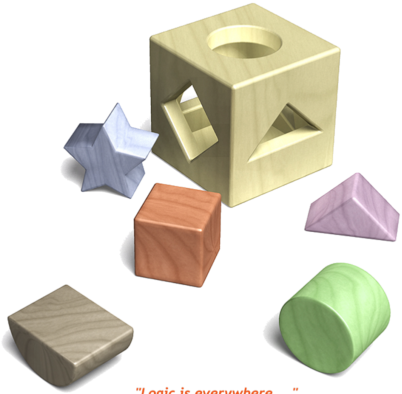
SAT Solving – Extensions

Steffen Hölldobler
International Center for Computational Logic
Technische Universität Dresden
Germany

- **Heuristics**
- **Polynomial Sub-Classes**
- **Backdoors**
- **Simplification**
- Implementation
- Combining Systematic and Stochastic Solvers
- Parallelization



"Logic is everywhere ..."

Variable Selection Heuristics – VSIDS

- **Variable State Independent Decay Sum**
  Moskewicz, Madigan, Zhao, Zhang, Malik: Chaff: Engineering an Efficient SAT Solver. In: Proceedings of the 38th Design Automation Conference: 2001
- **To each variable A an activity $activity(A)$ is assigned**
- **Initialization**
  - **random, frequency of occurrence in given formula, or 1**
- **Parameter $decay \geq 1$ (often $decay := 1/0.95$)**
- **Increment value inc (initially set to $inc := 1$)**
- **At each conflict do**
  - $activity(A) := activity(A) \times inc$
    for each A occurring in the derivation of the learned clause
  - $inc = inc \times decay$
- **Pick the variable with the highest activity**

Variable Selection Heuristics – VMTF

- **Variable Move to Front**
  Ryan: Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University: 2004
- **Like VSIDS except the following**
  - **At each conflict do**
    - $\boldsymbol{activity(A) := inc}$
      for each A occurring in the derivation of the learned clause
    - $\boldsymbol{inc := inc \times decay}$

Variable Selection Heuristics – BerkMin

- **Berkeley-Minsk SAT solver**
  Goldberg, Novikov: BerkMin: A Fast and Robust SAT Solver. In: Proceedings of the Conference on Design, Automation and Test in Europe: 2002
- **Store conflict clauses in a stack**
- **For each variable A**
  - $activity(A)$ **counts the number of conflict clauses in which A occurs**
  - $activity(A)$ **is periodically divided by a small constant ≥ 1**
- **Selection**
  - **Select a variable with the highest activity occurring in the top-most unsatisfied clause of the stack**
  - **If no such clause exists, select a variable with the highest activity**

Variable Selection Heuristics – MOMS

- **Maximum number of Occurrences on clauses of Minimum Size**
  Buro, Kleine-Büning: Report on a SAT competition. Technical report, University of Paderborn: 1992
- **Let m be the minimum clause length of formula F**
- **For each literal L**
  let $h(L)$ be the number of occurrences of L in clauses of length m
- **Pick a literal with highest h-value**

Polarity Selection Heuristics

- **Random**
  - **Pick a random polarity**
- **Ratio Heuristics**
  - **Pick the polarity according to a predefined ratio between positive and negative literals**
- **Jeroslaw-Wang Heuristics**
  - **For any literal L occurring in F let $h(L) = \sum_{C \in F, L \in C} 2^{-|C|}$**
  - **Select polarity which leads to higher h-value**
  - Jeroslaw, Wang: Solving Propositional Satisfiability Problems. Annals of Mathematics and Artificial Intelligence 1, 167-187: 1990
- **Phase Saving / Progress Saving**
  - **Use the last polarity the variable had before it was backtracked**
  - **Use any other heuristics if the variable was not assigned before**
  - Pipatsrisawat, Darwiche: A Lightweight Component Caching Schema for Satisfiability Solvers. In: Proc. SAT, 294-299: 2007

Restart-Schedule Heuristics – Geometric Series

- **Geometric series**
  - **Let *decay* ≥ 1**
  - **Let *c* be a counter (usually initialized with a value in [100, 1000])**
  - **Schedule a restart after the next *c* conflicts**
  - **When a restart is scheduled set $c := c \times decay$**
  - Eén, Sörensson: MiniSAT v1.13: A SAT Solver with Conflict Clause Minimization. In: Proc. SAT Competition: Solver Descriptions: 2005
- **Nested Geometric Series**
  - **Use two geometric series, where the outer is used as limit for the inner one**
  - **Use the inner series as above until it exceeds the current value of the outer**
  - **Reset the inner and increase the limit of the outer series**
  - Biere: Picosat Essentials. JSAT 4, 75-97: 2008

Restart-Schedule Heuristics – Luby Series

- **Consider the Luby series**

$$1\ 1\ 2\ 1\ 1\ 2\ 4\ 1\ 1\ 2\ 4\ 8\ \ldots$$

- **Let f be a factor (usually set to a value in $[1, 512]$)**
- **Let c be a counter (initially set to 0)**
- **Let r be a couter (initially set to 1)**
- **At each conflict do**
  - $c := c + 1$
  - **restart if $c > f \times Luby[r]$**
- **At each restart do**
  - $r := r + 1$
- Huang: The Effect of Restarts on the Efficiency of Clause Learning
  In: Proc. IJCAI, 2318-2323: 2007

Remove Heuristics

- **Usually, short clauses are not removed at all**
  - **Let C be a clause.**
  - **C is not removed if $|C| < n$, where $n \in \mathbb{N}$**
  - **Often $n = 3$**
- Eén, Sörensson: MiniSAT v1.13: A SAT Solver with Conflict Clause Minimization. In: Proc. SAT Competition: Solver Descriptions: 2005

Remove Heuristics – Activity Removal

- **Like VSIDS except the following**
  - **to each clause an activity is assigned**
  - **the activity is updated whenever the clause is used in a linear resolution derivation of a learnt clause**
- Eén, Sörensson: MiniSAT v1.13: A SAT Solver with Conflict Clause Minimization. In: Proc. SAT Competition: Solver Descriptions: 2005

Remove Heuristics – Literal Block Distance

- **Let C be a learned clause.**
- **Let $L(C) = \{\ell \mid \ell$ is the level of some literal occurring in $C\}$**
- **Let $n \in \mathbb{N}$**
- **Clause C is removed if $|L(C)| \geq n$**
- Audemard, Simon: Glucose: A Solver that Predicts Learnt Clause Quality. In: SAT Competitive Event Booklet: 2009

Remove Heuristics – Progress Saving Measure

- Is based on the phase saving polarity heuristics
- Let C be a clause
- Let $\mathcal{P}$ be the set of saved literal polarities, ie. for each atom A we find $A \in \mathcal{P}$ if the last used polarity of A was positive and $\overline{A} \in \mathcal{P}$ otherwise
- Let $\text{psm}_{\mathcal{P}}(C) = |\mathcal{P} \cap C|$
- Remove clauses with a high psm-value
- Audemard, Lagniez, Mazure, Sais: On Freezing and Reactivating Learnt Clauses. In Proc. SAT, 188-200: 2011

Heuristics – Parameter Selection

- Hutter, Hoos, Leyton-Brown, Stützle: Automatic Algorithm Configuration Framework. Journal of Artificial Intelligence Research 36, 267-306: 2009
- **Idea** **Use a stochastic local search algorithm on the parameter space of a SAT-solver to find a *good* parameter setting**

Polynomial Sub-Classes

- **2SAT** F is in 2SAT iff each clause occurring in F has at most two literals
- **Horn** F is in Horn iff each clause has at most one positive literal
- **AHorn**
  F is in AHorn (anti Horn) iff each clause has at most one negative literal
- **RHorn** F is in RHorn (renamable Horn) iff there is a mapping Φ from variables to literals such that after applying Φ and modulo double negation F is in Horn
  - Let $F = \langle[1, 2], [\overline{3}, \overline{4}]\rangle$
  - Let $\Phi = \{1 \mapsto \overline{5}, 3 \mapsto \overline{6}\}$
  - After applying Φ and modulo double negation we obtain $F = \langle[\overline{5}, 2], [6, \overline{4}]\rangle$
- **UP+PL** F is in UP+PL iff it can be solved by applying only UNIT and PURE

2SAT – Backtrack Once

- del Val: On 2-SAT and Renamable Horn. In: Proc. AAAI, 279-284: 2000
- **Let F be a 2SAT-formula and J a partial interpretation**
- **Procedure *unitPropagate*$(F :: J)$**
  - **computes the closure of $F :: J$ under UNIT**
- **Procedure *BTOSAT*$(F :: J)$**

  while $[\,] \not\in F|_J$ and $F|_J \neq \langle\,\rangle$ do
  - **choose an unassigned literal L**
  - $F :: J' :=$ ***unitPropagate***$(F :: J, \dot{L})$
  - **if $[\,] \in F|_{J'}$ then $F :: J :=$ *unitPropagate*$(F :: J, \overline{L})$ else $F :: J := F :: J'$**

  if $[\,] \in F|_J$ then return *unsatisfiable* else return J

BTOSAT – Examples

- Suppose literals are assigned in their natural order
- Suppose positive literals are prefered
- What happens if *BTOSAT* is applied to
  - $F = \langle [\overline{1}, 2], [1, 3], [\overline{2}, \overline{4}], [4, \overline{3}]$?
  - $F = \langle [\overline{1}, 2], [\overline{2}, 3], \ldots, [\overline{8}, 9], [\overline{8}, \overline{9}], [8, \overline{9}], [8, 9] \rangle$? $\rightsquigarrow$ next slide
  - $F = \langle [\overline{1}, 2], [\overline{2}, 3], \ldots, [\overline{9}, 10], [\overline{10}, 11], \ldots, [\overline{18}, 19], [\overline{1}, 20], [\overline{1}, \overline{20}], \ldots, [\overline{9}, 20], [\overline{9}, \overline{20}] \rangle$? $\rightsquigarrow$ Exercise
- The worst-case complexity of *BTOSAT* is $O(nm)$, where
  - n is the number of variables and
  - m is the number of clauses in F

BTOSAT – A Derivation

- Let $F = \langle [\overline{1}, 2], [\overline{2}, 3], \ldots, [\overline{8}, 9], [\overline{8}, \overline{9}], [8, \overline{9}], [8, 9] \rangle$ and $J = ()$
- We obtain $F :: ()$

$$
\begin{array}{lll}
\leadsto_{DECIDE} & F :: (\dot{1}) & F|_{(\dot{1})} = \langle [2], [\overline{2}, 3], \ldots, [\overline{8}, 9], [\overline{8}, \overline{9}], [8, \overline{9}], [8, 9] \rangle \\
\leadsto_{UNIT} & F :: (\dot{1}, 2) & F|_{(\dot{1},2)} = \langle [3], \ldots, [\overline{8}, 9], [\overline{8}, \overline{9}], [8, \overline{9}], [8, 9] \rangle \\
\ldots & & \\
\leadsto_{UNIT} & F :: (\dot{1}, 2, \ldots, 8) & F|_{(\dot{1},2,\ldots,8)} = \langle [9], [\overline{9}] \rangle \\
\leadsto_{UNIT} & F :: (\dot{1}, 2, \ldots, 8, 9) & F|_{(\dot{1},2,\ldots,8,9)} = \langle [\,] \rangle \\
\leadsto_{NB} & F :: (\overline{1}) & F|_{(\overline{1})} = \langle [\overline{2}, 3], \ldots, [\overline{8}, 9], [\overline{8}, \overline{9}], [8, \overline{9}], [8, 9] \rangle \\
\leadsto_{DECIDE} & F :: (\overline{1}, \dot{2}) & F|_{(\overline{1},\dot{2})} = \langle [3], \ldots, [\overline{8}, 9], [\overline{8}, \overline{9}], [8, \overline{9}], [8, 9] \rangle \\
\ldots & & \\
\leadsto_{UNIT} & F :: (\overline{1}, \dot{2}, \ldots, 8, 9) & F|_{(\overline{1},\dot{2},\ldots,8,9)} = \langle [\,] \rangle \\
\leadsto_{NB} & F :: (\overline{1}, \overline{2}) & F|_{(\overline{1},\overline{2})} = \langle [\overline{3}, 4], \ldots, [\overline{8}, 9], [\overline{8}, \overline{9}], [8, \overline{9}], [8, 9] \rangle \\
\ldots & & \\
\leadsto_{NB} & F :: (\overline{1}, \overline{2}, \ldots, \overline{8}) & F|_{(\overline{1},\overline{2},\ldots,\overline{8})} = \langle [\overline{9}], [9] \rangle \\
\leadsto_{UNIT} & F :: (\overline{1}, \overline{2}, \ldots, \overline{8}, 9) & F|_{(\overline{1},\overline{2},\ldots,\overline{8},9)} = \langle [\,] \rangle \\
\leadsto_{UNSAT} & F :: \text{UNSAT} &
\end{array}
$$

2SAT – BinSAT – Preliminaries

- **Let F be a 2SAT-formula**
- **We distinguish between permanent and temporary partial interpretations**
- $F|_{permVal}$
  denotes the reduct of F wrt a permanent partial interpretation
- $F|^{tempVal}_{permVal}$
  denotes the reduct of F wrt a permanent and a temporary partial interpretation, where the permanent interpretation overrules the partial one

2SAT – BinSAT

- **Procedure** $unitPropagate(F)$ computes the closure of F under UNIT with respect to and setting $permVal(A)$ and $permVal(\overline{A})$ accordingly
- **Procedure** $BinSAT(F)$

  for each atom A occurring in F do

  - $tempVal(A), tempVal(\overline{A}), permVal(A), permVal(\overline{A}) := nil$

  $F := unitPropagate(F)$

  while $[\,] \not\in F|_{permVal}$ and $(\exists L)\ permVal(L) = tempVal(L) = nil$ do

  - $tempUnitPropagate(L)$

  If $[\,] \in F|^{tempVal}_{permVal}$ then return *unsatisfiable* else return *satisfiable*

2SAT – BinSAT – TempUnitPropagate

- Let F be a 2SAT-formula
- Let L be an unassigned or a temporarily assigned literal
- Procedure $tempUnitPropagate(L)$

  if $tempVal(L) = \bot$ (a conflict has occurred) then do
  - $F := unitPropagate(F \wedge [L])$
  - return

  $tempVal(L) := \top;\ tempVal(\overline{L}) = \bot$

  for each $[\overline{L}, L'] \in F$ do
  - if $[\,] \in F|_{permVal}$ then return
  - if $tempVal(L') \neq \top$ then $tempUnitPropagate(L')$

2SAT – BinSAT – Model Generation

- ***BinSAT*** **returns only *satisfiable***
- **In this case, a model can be generated as follows**
- **For each A do**
  - **if $\mathit{permVal}(A) \neq \mathit{nil}$ then A is assigned $\mathit{permVal}(A)$**
  - **otherwise, A is assigned $\mathit{tempVal}(A)$**

2SAT – BinSAT – Examples

- Suppose literals assigned in their natural order.
- Suppose positive literals are prefered.
- What happens if *BinSAT* is applied to
  - $F = \langle[\overline{1}, 2], [1, 3], [\overline{2}, \overline{4}], [4, \overline{3}]$? $\leadsto$ next slide
  - $F = \langle[\overline{1}, 2], [\overline{2}, 3], \ldots, [\overline{8}, 9], [\overline{8}, \overline{9}], [8, \overline{9}], [8, 9]\rangle$? $\leadsto$ next but one slide
  - $F = \langle [\overline{1}, 2], [\overline{2}, 3], \ldots, [\overline{9}, 10], [\overline{10}, 11], \ldots, [\overline{18}, 19],$ $[\overline{1}, 20], [\overline{1}, \overline{20}], \ldots, [\overline{9}, 20], [\overline{9}, \overline{20}]\rangle$? $\leadsto$ Exercise
- The worst-case complexity of *BinSAT* is $O(m)$, where
  - m is the number of clauses in F

2SAT – BinSAT – A Derivation

- **Notation**
  - $F :: permVal :: tempVal$
    denotes a formula with a permanent and a temporary partial interpretation
- **Let** $F = \langle [\overline{1}, 2], [1, 3], [\overline{2}, \overline{4}], [4, \overline{3}] \rangle$.
- **We obtain**

$$
\begin{array}{lll}
 & F :: () :: () & \\
\leadsto_{tempDECIDE} & F :: () :: (\dot{1}) & \{[\overline{1}, 2]\} \\
\leadsto_{tempUNIT} & F :: () :: (\dot{1}, 2) & \{[\overline{2}, \overline{4}]\} \\
\leadsto_{tempUNIT} & F :: () :: (\dot{1}, 2, \overline{4}) & \{[4, \overline{3}]\} \\
\leadsto_{tempUNIT} & F :: () :: (\dot{1}, 2, \overline{4}, \overline{3}) & \emptyset \\
\leadsto_{SAT} & F :: \text{SAT} &
\end{array}
$$

2SAT – BinSAT – Another Derivation

- Let $F = \langle[\overline{1}, 2], [\overline{2}, 3], \ldots, [\overline{8}, 9], [\overline{8}, \overline{9}], [8, \overline{9}], [8, 9]\rangle$
- We obtain $F :: () :: ()$

$$
\begin{array}{lll}
\leadsto_{tempDECIDE} & F :: () :: (\dot{1}) & \{[\overline{1}, 2]\} \\
\leadsto_{tempUNIT} & F :: () :: (\dot{1}, 2) & \{[\overline{2}, 3]\} \\
\ldots & & \\
\leadsto_{tempUNIT} & F :: () :: (\dot{1}, 2, \ldots, 8) & \{[\overline{8}, 9], [\overline{8}, \overline{9}]\} \\
\leadsto_{tempUNIT} & F :: () :: (\dot{1}, 2, \ldots, 8, 9) & \{[\overline{9}, 8], [\overline{9}, \overline{8}]\} \\
\leadsto & F :: (\overline{8}) :: (\dot{1}, 2, \ldots, 8, 9) & \\
\leadsto_{UNIT} & F :: (\overline{8}, 9) :: (\dot{1}, 2, \ldots, 8, 9) & [\,] \in F|_{(\overline{8},9)} \\
\leadsto_{UNSAT} & F :: \mathrm{UNSAT} &
\end{array}
$$

- **Note**
  - $\leadsto$ denotes the call of *unitPropagate* within *tempUnitPropagate*
  - In general, temporary partial interpretations are kept
    but they may be overwritten by the permanent ones

Backdoors

- Williams, Gomes, Selman: Backdoors to Typical Case Complexity.
  In: Proc. IJCAI, 1173-1178: 2003
- **Why show SAT solvers such a good scaling behavior although SAT is in NP?**
  - **Practical combinatorial problem instances have a substantial amout of (hidden) tractable sub-structure**
  - **New algorithmic techniques exploit such tractable structure**

Sub-Solver

- Let F be a SAT-instance.
- A sub-solver S given F as input satisfies the following
  - Trichotomy S either rejects F or *determines* F correctly
  - Efficiency S runs in polynomial time
  - Trivial solvability S can determine if F is trivially true (i.e. is empty) or trivially false (i.e. contains the empty clause)
  - Self-reducibility If S determines F, then it determines $F|_J$ for any (partial) interpretation J

Sub-Solver – Example

- **2SAT**
  - **Trichotomy** S must reject a formula F if it is not 2SAT, i.e. if F contains a clause with more than 2 literals
  - **Efficiency** S must solve 2SAT in polynomial time like eg. BTOSAT or BinSAT
  - **Trivial Solvability** obvious
  - **Self-reducibility** If F is in 2SAT, then $F|_J$ is also in 2SAT

Backdoors

- Let S be a sub-solver and F a SAT-instance
- A non-empty subset $\mathcal{B} \subseteq \text{atoms}(F)$ is a **weak backdoor in F for S** if for some $J : \mathcal{B} \to \{\top, \bot\}$, S returns a satisfying assignment for $F|_J$
- A non-empty subset $\mathcal{B} \subseteq \text{atoms}(F)$ is a **strong backdoor in F for S** if for all $J : \mathcal{B} \to \{\top, \bot\}$, S returns a satisfying assignment for $F|_J$

Backdoors – Example 1

- Consider $F = \langle [\overline{2}, 3], [1, \overline{3}, \overline{4}], [\overline{1}, 6], [\overline{1}, 5, \overline{6}], [4, \overline{5}], [2, \overline{5}, \overline{6}] \rangle$
- $\mathcal{B} = \{1\}$ is a weak backdoor in F for UP+PL
  - Let $F' = F|_1 = \langle [\overline{2}, 3], [6], [5, \overline{6}], [4, \overline{5}], [2, \overline{5}, \overline{6}] \rangle$
  - We obtain

$$
\begin{array}{lll}
 & F' :: () & \\
\leadsto_{UNIT} & F' :: (6) & F'|_{(6)} = \langle [\overline{2}, 3], [5], [4, \overline{5}], [2, \overline{5}] \rangle \\
\leadsto_{UNIT} & F' :: (6, 5) & F'|_{(6,5)} = \langle [\overline{2}, 3], [4], [2] \rangle \\
\leadsto_{UNIT} & F' :: (6, 5, 2) & F'|_{(6,5,2)} = \langle [3], [4] \rangle \\
\leadsto_{UNIT} & F' :: (6, 5, 2, 3) & F'|_{(6,5,2,3)} = \langle [4] \rangle \\
\leadsto_{UNIT} & F' :: (6, 5, 2, 3, 4) & F'|_{(6,5,2,3,4)} = \langle\, \rangle \\
\leadsto_{SAT} & F' :: \mathrm{SAT} &
\end{array}
$$

Backdoors – Example 2

- Reconsider $F = \langle[\overline{2}, 3], [1, \overline{3}, \overline{4}], [\overline{1}, 6], [\overline{1}, 5, \overline{6}], [4, \overline{5}], [2, \overline{5}, \overline{6}]\rangle$
- $\mathcal{B} = \{1, 2\}$ is a strong backdoor in F for 2SAT because
  - $F_1 = F|_{1,2} = \langle[3], [6], [5, \overline{6}], [4, \overline{5}]\rangle$ and $(4, 5, 6, 3) \models F_1$
  - $F_2 = F|_{\overline{1},2} = \langle[3], [\overline{3}, \overline{4}], [4, \overline{5}]\rangle$ and $(\overline{5}, \overline{4}, 3) \models F_2$
  - $F_3 = F|_{1,\overline{2}} = \langle[6], [5, \overline{6}], [4, \overline{5}]\rangle$ and $(4, 5, 6) \models F_3$
  - $F_4 = F|_{\overline{1},\overline{2}} = \langle[\overline{3}, \overline{4}], [4, \overline{5}], [\overline{5}, \overline{6}]\rangle$ and $(\overline{5}, \overline{4}, \dot{3}) \models F_4$

Minimal and Smallest Backdoors

- **Let S be a sub-solver and F a SAT-instance**
- **A (weak/strong) backdoor $\mathcal{B}$ in F for S is said to be minimal iff no proper subset of $\mathcal{B}$ is a (weak/strong) backdoor in F for S**
- **A (weak/strong) backdoor $\mathcal{B}$ in F for S is said to be smallest iff it is minimal and $|\mathcal{B}| \leq |\mathcal{B}'|$ for any minimal (weak/strong) backdoor $\mathcal{B}'$ in F for S**

Minimal and Smallest Backdoors – Examples

- Let $G = \langle [\overline{1}, \overline{2}, 4], [\overline{4}, 6], [\overline{4}, \overline{6}, 7], [\overline{4}, \overline{6}, \overline{7}], [\overline{5}, 6], [\overline{5}, \overline{6}, 7], [\overline{5}, \overline{6}, \overline{7}] \rangle$
  $F = \langle [1, 3], [2, 3], [\overline{3}, 4, 5] \rangle \wedge G$
  - G can be solved by a Horn sub-solver
  - $(\overline{1}, \overline{4}, \overline{5}) \models G$
- $\mathcal{B}_1 = \{3, 4\}$ is a strong backdoor in F for Horn $\rightsquigarrow$ Exercise
  - Neither $\{3\}$ nor $\{4\}$ are strong backdoors in F for Horn
  - $\mathcal{B}_1 = \{3, 4\}$ is a minimal strong backdoor in F for Horn
- $\mathcal{B}_2 = \{1, 3, 4\}$ is another strong backdoor in F for Horn
  - $\mathcal{B}_2$ is not a minimal strong backdoor in F for Horn
- $\mathcal{B}_3 = \{1, 2, 4\}$ is yet another strong backdoor in F for Horn. $\rightsquigarrow$ Exercise
  - $\mathcal{B}_3$ is not a smallest strong backdoor in F for Horn
  - $\mathcal{B}_1$ is a smallest strong backdoor in F for Horn $\rightsquigarrow$ Exercise
- $\mathcal{B}_4 = \{3, 5\}$ is another smallest strong backdoor in F for Horn $\rightsquigarrow$ Exercise

Backdoors – Remarks

- Backdoors exist for each F
- Given a weak backdoor $\mathcal{B}$ in F
  - The search cost for solving F is of order $2^{|\mathcal{B}|}$
- The size of backdoors in practical problem instances may be surprisingly small

instance	# vars	# clauses	backdoor	fraction
logstics.c	6783	437431	12	0.0018
3bitadd_32	8704	32316	53	0.0061
pipe_01	7736	26087	23	0.0030
qg_30_1	1235	8523	14	0.0113
qg_35_1	1597	10658	15	0.0094

- Given F with $|\text{var}(F)| = n$, $k \geq 0$, and sub-solver S. The problem whether there exists a (weak/strong) backdoor in F for S of size k is $\mathcal{NP}$-hard

Backdoors and Parameterized Complexity

- Gario: Backdoors for SAT. EMCL Master Thesis. TU Dresden: 2011
- de Haan: Parameterized Complexity in the Polynomial Hierarchy. PhD Thesis TU Wien: 2016

Simplification – Warm Up (1)

- **When are two formulas F and G semantically equivalent ($F \equiv G$)?**
- **Let $G = F \setminus \{C\}$**
  - **Is $G \equiv F$?**
  - **Under which condition is $G \equiv F$?**
  - **How is the check performed?**
  - **How complex is this check?**
- **How is $F \equiv_{SAT} G$ defined?**
- **Are there other redundancies which can be eliminated?**

Simplification – Warm Up (2)

- **Which of the following statements is true?**
  - $F \wedge L \equiv F|_{(L)}$
  - $F \wedge L \equiv_{SAT} F|_{(L)}$
  - $F \wedge L \models F|_{(L)}$
  - Let C and D be clauses with $C \subset D$ in $F \wedge D \models F \wedge C$
  - Let C and D be clauses with $C \subset D$ in $F \wedge C \models F \wedge D$

Simplification – Warm Up (3)

- How many models has the formula $F = (a \lor \bar{b}) \land (\bar{a} \lor b)$?
- Enumerate the models!
- How many models has the formula $F = (a \lor \bar{a}) \land (\bar{a} \lor a)$?
- Enumerate the models!
- Do you see a connection?

Equivalence Preserving Techniques

- A clause C is a tautology iff it contains a complementary pair of literals
  - $(a \vee b \vee \bar{b})$
  - Tautologies can be removed
- Clause C subsumes clause D iff $C \subseteq D$
  - $(a \vee b)$ subsumes $(a \vee b \vee \bar{c})$
  - Subsumed clauses can be removed

Resolution

- **Remember** Let C_1 be a clause containing L and C_2 be a clause containing $\overline{L}$
  The (propositional) resolvent of C_1 and C_2 with respect to L is the clause

$$(C_1 \setminus \{L\}) \cup (C_2 \setminus \{\overline{L}\})$$

  C is said to be a resolvent of C_1 and C_2 iff
  there exists a literal L such that C is the resolvent of C_1 and C_2 wrt L

  - We will write $C_1 \otimes_L C_2$ to denote the resolvent of C_1 and C_2 wrt L

- Is the addition of resolvents an equivalence preserving technique?

- Shall we apply it?

Self-Subsuming Resolution

- **Suppose** $C \vee L \otimes_L D \vee \bar{L} = D$
- **Example** $(a \vee b \vee d) \otimes_d (a \vee b \vee c \vee \bar{d}) = (a \vee b \vee c)$
- **Observe** the resolvent subsumes one of its parent clauses
- **Example (continued)** Suppose, a CNF contains both parent clauses

$$\ldots (a \vee b \vee d),\ (a \vee b \vee c \vee \bar{d}) \ldots$$

  - If D is added, then $D \vee \bar{L}$ can be removed
  - which in essence removes $\bar{L}$ from $D \vee \bar{L}$

$$\ldots (a \vee b \vee d),\ (a \vee b \vee c) \ldots$$

- Initially in the SATeLite preprocessor
  - now common in most solvers (i.e., as pre- and inprocessing)

Self-Subsuming Resolution – Example

- **Self-Subsuming Resolution** $C \lor L \otimes_L D \lor \bar{L} = D$
- **Example**

$$
\begin{array}{llll}
(\quad b \lor c) & \land & (\bar{a} \lor b \lor c) & \land \\
(\quad) & \land & (a \quad) & \land \\
(\bar{a} \lor \bar{b} \quad) & \land & (\bar{a} \lor \bar{b} \lor \bar{d}) & \land \\
(a \lor \bar{c} \quad) & \land & (a \lor \bar{c} \lor \bar{d}) &
\end{array}
$$

Probing (1)

- **Idea** use unit propagation do derive extra information
- **Vivification** of a clause $C = (L_1 \vee \cdots \vee L_n)$, $C \in F$
  - Unit propagation results in the empty clause
    1. $F :: (\overline{L_1}, \ldots, \overline{L_i}) \leadsto^*_{UNIT} F :: J$, where $[] \in F|_J$, $i < n$
  - Unit propagation implies another literal of the clause C
    2. $F :: (\overline{L_1}, \ldots, \overline{L_i}) \leadsto^*_{UNIT} F :: J$, where $L_j \in J$, $i < j \leq n$
  - Unit propagation implies another negated literal of the clause C
    3. $F :: (\overline{L_1}, \ldots, \overline{L_i}) \leadsto^*_{UNIT} F :: J$, where $\overline{L_j} \in J$, $i < j \leq n$
- **Exploit** $F \models ((\bar{L}_1 \wedge \cdots \wedge \bar{L}_i) \rightarrow L)$ and, hence, $F \equiv F \wedge (L_1 \vee \cdots \vee L_i \vee L)$
  - By the above statements and self-subsuming resolution, replace C with
    1. $(L_1 \vee \cdots \vee L_i)$
    2. $(L_1 \vee \cdots \vee L_i \vee L_j)$
    3. $C \setminus \{L_j\}$

Probing (2)

- **Failed Literal** test for some literal L
  - $F :: L \leadsto^*_{UNIT} F :: J$, where $[] \in F|_J$, then add the unit clause $\bar{L}$
  - Could also apply conflict analysis
  - Then: learn all UIP clauses (have to be units)
- Test for **entailed literals** (also backbones, necessary assignments), and **equivalent literals** wrt F
  - $F :: L \leadsto^*_{UNIT} F :: J_L$ where J_L is the set of all implied literals of L
  - $F :: \bar{L} \leadsto^*_{UNIT} F :: J_{\bar{L}}$ where $J_{\bar{L}}$ is the set of all implied literals of $\bar{L}$
  - L' is an **entailed literal** if $L' \in J_L \cap J_{\bar{L}}$,
  - L' and L are **equivalent** if $L' \in J_L$ and $\bar{L}' \in J_{\bar{L}}$

Simplification – Equivalence Preserving Techniques

- Unit propagation
- Subsumption
- Resolution, hyper binary resolution
- Self-subsuming resolution
- Hidden tautology elimination
- Asymmetric tautology elimination
- Probing
  - Clause vivification
  - Necessary assignments
  - Failed literals
- Adding and removing transitive implications (binary clauses)
- Higher reasoning: Gaussian elimination, Fourier-Motzkin method
- No need to construct a model, the found model can be used

Simplification – Equisatisfiability Preserving Techniques

- **Model needs to be constructed**
- **Information required for model construction can be stored on a stack**
- **Reason** $F \leadsto_{bad} F' \leadsto_{bad} F'' \leadsto_{bad} F''' \ldots$
- **Reconstruction processes this chain in the opposite direction**
- $\ldots J''' \rightarrow J'' \rightarrow J' \rightarrow J$
- **Thus, techniques can be run in any order, and mixed with the good ones**
- **For all currently used techniques, this process is polynomial (linear in the stack)**

Equivalent Literal Substitution

- Given a formula F such that $F \models (L_1 \leftrightarrow L_2)$
  - then replace each occurrence of L_1 and $\overline{L_1}$ in F by L_2 and $\overline{L_2}$, respectively
  - remove double negation
- How to find equivalent literals?
  - By probing
  - By analyzing the binary implication graph (each SCC is an equivalence)
    - $F \models (a \rightarrow b) \wedge (b \rightarrow c) \wedge (c \rightarrow a)$, then $F \models a \leftrightarrow b \leftrightarrow c$
  - By structural hashing
    - $F \models (L_1 \leftrightarrow (a \wedge b)) \wedge (L_2 \leftrightarrow (a \wedge b)$, then $F \models (L_1 \leftrightarrow L_2)$
    - Works for many other gate types and variable definitions
    - Weakness   definitions have to be found (structurally or semantically)
- How to construct the model J from J'?
  - If $L_2 \in J'$, then $J := (J' \setminus \{L_1, \bar{L}_1\}) \cup \{L_1\}$
  - If $\bar{L}_2 \in J'$, then $J := (J' \setminus \{L_1, \bar{L}_1\}) \cup \{\bar{L}_1\}$

Variable Elimination by Clause Distribution

- Given a formula F in CNF and a literal L
  - $F_L = \{C \in F \mid L \in C\}$
  - $F_L \otimes_L F_{\overline{L}} = \{C \otimes_L D \mid C \in F_L \text{ and } D \in F_{\overline{L}}\}$
- Given a formula F in CNF, variable elimination (or DP resolution) removes a variable X by replacing $F_X \cup F_{\overline{X}}$ by $F_X \otimes_X F_{\overline{X}}$
- Example

$F_{\overline{X}} \backslash F_X$	$(X \vee c)$	$(X \vee \overline{d})$	$(X \vee \overline{a} \vee \overline{b})$
$(\overline{X} \vee a)$	$(a \vee c)$	$(a \vee d)$	$(a \vee \overline{a} \vee \overline{b})$
$(\overline{X} \vee b)$	$(b \vee c)$	$(b \vee d)$	$(b \vee \overline{a} \vee \overline{b})$
$(\overline{X} \vee \overline{e} \vee f)$	$(c \vee \overline{e} \vee f)$	$(d \vee \overline{e} \vee f)$	$(\overline{a} \vee \overline{b} \vee \overline{e} \vee f)$

  - Observe $|F_X \otimes_X F_{\overline{X}}| > |F_X| + |F_{\overline{X}}|$
  - Exponential growth of clauses in general

Variable Elimination by Substitution

- **Idea** Detect gates (or definitions) $X \leftrightarrow \mathrm{GATE}(a_1, \ldots, a_n)$ in the formula and use them to reduce the number of added clauses
- **Possible gates**

gate	G_X	$G_{\bar{X}}$
$\mathrm{AND}(a_1, \ldots, a_n)$	$(X \vee \bar{a}_1 \vee \cdots \vee \bar{a}_n)$	$(\bar{X} \vee a_1), \ldots, (\bar{x} \vee a_n)$
$\mathrm{OR}(a_1, \ldots, a_n)$	$(X \vee \bar{a}_1), \ldots, (X \vee \bar{a}_n)$	$(\bar{X} \vee a_1 \vee \cdots \vee a_n)$
$\mathrm{ITE}(c, t, f)$	$(X \vee \bar{c} \vee \bar{t}), (X \vee c \vee \bar{f})$	$(\bar{X} \vee \bar{c} \vee t), (\bar{X} \vee c \vee f)$

- **Variable elimination by substitution**
  - Let $R_X = F_X \setminus G_X$ and $R_{\bar{X}} = F_{\bar{X}} \setminus G_{\bar{X}}$
  - Replace $F_X \cup F_{\bar{X}}$ by $G_X \otimes_X R_{\bar{X}} \cup G_{\bar{X}} \otimes_X R_X$
- Always less than $F_x \otimes_x F_{\bar{x}}$

Variable Elimination by Substitution

- Example of gate extraction: $X \leftrightarrow \mathrm{AND}(a, b)$
  - $F_X = (X \vee c) \wedge (X \vee \bar{d}) \wedge (X \vee \bar{a} \vee \bar{b})$
  - $F_{\bar{X}} = (\bar{X} \vee a) \wedge (\bar{X} \vee b) \wedge (\bar{X} \vee \bar{e} \vee f)$
- Example of substitution

$$
\begin{array}{ll|ccc}
 & & \multicolumn{2}{c}{\overbrace{\hspace{6cm}}^{R_X}} & \overbrace{\hspace{3cm}}^{G_X} \\
 & & (X \vee c) & (X \vee \bar{d}) & (X \vee \bar{a} \vee \bar{b}) \\
\hline
G_{\bar{X}} & (\bar{X} \vee a) & (a \vee c) & (a \vee \bar{d}) & \\
 & (\bar{X} \vee b) & (b \vee c) & (b \vee \bar{d}) & \\
R_{\bar{X}} & (\bar{X} \vee \bar{e} \vee f) & & & (\bar{a} \vee \bar{b} \vee \bar{e} \vee f)
\end{array}
$$

  - **Observe** $|F_X \otimes F_{\bar{X}}| < |F_X| + |F_{\bar{X}}|$

Variable Elimination

- **How can the model be reconstructed?**
- **Given F, we picked literal X, removed F_X and $F_{\bar{X}}$, and added $F_X \otimes_X F_{\bar{X}}$**
- **A model J does not contain a value for X**
- **How does it work?**

Bounded Variable Addition

- Given a CNF formula F
- **Idea** Can we construct a logically equivalent or equisatisfiable formula F' by introducing a new variable $X \notin VAR(F)$ such that $|F'| < |F|$?
  - Reverse of variable elimination
- **Example** Replace the clauses

$$
\begin{array}{ccc}
(a \vee c) & (a \vee d) & \\
(b \vee c) & (b \vee d) & \\
(c \vee \bar{e} \vee f) & (d \vee \bar{e} \vee f) & (\bar{a} \vee \bar{b} \vee \bar{e} \vee f)
\end{array}
$$

by

$$
\begin{array}{ccc}
(\bar{X} \vee a) & (\bar{X} \vee b) & (\bar{X} \vee \bar{e} \vee f) \\
(X \vee c) & (X \vee d) & (X \vee \bar{a} \vee \bar{b})
\end{array}
$$

- **Challenge** How to find suitable patterns for replacement?

Factoring Out Subclauses

- **Example** Replace

$$(a \lor b \lor c \lor d) \quad (a \lor b \lor c \lor e) \quad (a \lor b \lor c \lor f)$$

by

$$(X \lor d) \quad (X \lor e) \quad (X \lor f) \quad (\bar{X} \lor a \lor b \lor c)$$

- Adds 1 variable and 1 clause
- Reduces number of occurrences of literals by 2
- Not compatible with variable elimination, which would eliminate X immediately
- So this does not work . . .

Bounded Variable Addition

- **Example** Smallest pattern that is compatible: replace

$$\begin{array}{ll} (a \vee d) & (a \vee e) \\ (b \vee d) & (b \vee e) \\ (c \vee d) & (c \vee e) \end{array}$$

by

$$\begin{array}{lll} (\bar{X} \vee a) & (\bar{X} \vee b) & (\bar{X} \vee c) \\ (X \vee d) & (X \vee e) & \end{array}$$

- Adds 1 variable
- Removes 1 clause

Bounded Variable Addition

- **Possible Patterns**

$$\begin{array}{ccc} (X_1 \vee L_1) & \ldots & (X_1 \vee L_k) \\ \vdots & & \vdots \\ (X_n \vee L_1) & \ldots & (X_n \vee L_k) \end{array} \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k} (X_i \vee L_j)$$

- **Replaced by**

$$\bigwedge_{i=1}^{n} (Y \vee X_i) \wedge \bigwedge_{j=1}^{k} (\bar{Y} \vee L_j)$$

where Y is a new variable

  - **Suppose, k clauses share a literal L_j**
  - **Suppose, n literals X_i appear in the clauses**
  - **Then, $nk - n - k$ clauses are removed**

Bounded Variable Addition on AtMostOneZero (1)

- **Example** Encoding of AtMostOneZero($x_1, \ldots, x_n$)

$$
\begin{array}{l}
(x_1 \vee x_2) \wedge (x_9 \vee x_{10}) \wedge (x_8 \vee x_{10}) \wedge (x_7 \vee x_{10}) \wedge (x_6 \vee x_{10}) \wedge \\
(x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_8 \vee x_9) \wedge (x_7 \vee x_9) \wedge (x_6 \vee x_9) \wedge \\
(x_1 \vee x_4) \wedge (x_2 \vee x_4) \wedge (x_3 \vee x_4) \wedge (x_7 \vee x_8) \wedge (x_6 \vee x_8) \wedge \\
(x_1 \vee x_5) \wedge (x_2 \vee x_5) \wedge (x_3 \vee x_5) \wedge (x_4 \vee x_5) \wedge (x_6 \vee x_7) \wedge \\
(x_1 \vee x_6) \wedge (x_2 \vee x_6) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6) \wedge \\
(x_1 \vee x_7) \wedge (x_2 \vee x_7) \wedge (x_3 \vee x_7) \wedge (x_4 \vee x_7) \wedge (x_5 \vee x_7) \wedge \\
(x_1 \vee x_8) \wedge (x_2 \vee x_8) \wedge (x_3 \vee x_8) \wedge (x_4 \vee x_8) \wedge (x_5 \vee x_8) \wedge \\
(x_1 \vee x_9) \wedge (x_2 \vee x_9) \wedge (x_3 \vee x_9) \wedge (x_4 \vee x_9) \wedge (x_5 \vee x_9) \wedge \\
(x_1 \vee x_{10}) \wedge (x_2 \vee x_{10}) \wedge (x_3 \vee x_{10}) \wedge (x_4 \vee x_{10}) \wedge (x_5 \vee x_{10})
\end{array}
$$

- **Replace** $(x_i \vee x_j)$ **with** $i \in \{1..5\}, j \in \{6..10\}$ **by** $(x_i \vee y)$, $(x_j \vee \bar{y})$

Bounded Variable Addition on AtMostOneZero (2)

- **Example** Encoding of AtMostOneZero($x_1, \ldots, x_n$)

$$
\begin{array}{l}
(x_1 \vee x_2) \wedge (x_9 \vee x_{10}) \wedge (x_8 \vee x_{10}) \wedge (x_7 \vee x_{10}) \wedge (x_6 \vee x_{10}) \wedge \\
(x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_8 \vee x_9) \wedge (x_7 \vee x_9) \wedge (x_6 \vee x_9) \wedge \\
(x_1 \vee x_4) \wedge (x_2 \vee x_4) \wedge (x_3 \vee x_4) \wedge (x_7 \vee x_8) \wedge (x_6 \vee x_8) \wedge \\
(x_1 \vee x_5) \wedge (x_2 \vee x_5) \wedge (x_3 \vee x_5) \wedge (x_4 \vee x_5) \wedge (x_6 \vee x_7) \wedge \\
(x_1 \vee y) \wedge (x_2 \vee y) \wedge (x_3 \vee y) \wedge (x_4 \vee y) \wedge (x_5 \vee y) \wedge \\
(x_6 \vee \bar{y}) \wedge (x_7 \vee \bar{y}) \wedge (x_8 \vee \bar{y}) \wedge (x_9 \vee \bar{y}) \wedge (x_{10} \vee \bar{y})
\end{array}
$$

- **Replace matched pattern by**

$$
(x_1 \vee z) \wedge (x_2 \vee z) \wedge (x_3 \vee z) \wedge (x_4 \vee \bar{z}) \wedge (x_5 \vee \bar{z}) \wedge (y \vee \bar{z})
$$

Bounded Variable Addition on AtMostOneZero (3)

- **Example** Encoding of AtMostOneZero($x_1, \ldots, x_n$)

$$
\begin{array}{l}
(x_1 \vee x_2) \wedge (x_9 \vee x_{10}) \wedge (x_8 \vee x_{10}) \wedge (x_7 \vee x_{10}) \wedge (x_6 \vee x_{10}) \wedge \\
(x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_8 \vee x_9) \wedge (x_7 \vee x_9) \wedge (x_6 \vee x_9) \wedge \\
(x_1 \vee z) \wedge (x_2 \vee z) \wedge (x_3 \vee z) \wedge (x_7 \vee x_8) \wedge (x_6 \vee x_8) \wedge \\
(x_4 \vee \bar{z}) \wedge (x_5 \vee \bar{z}) \wedge (y \vee \bar{z}) \wedge (x_4 \vee x_5) \wedge (x_6 \vee x_7) \wedge \\
(x_4 \vee y) \wedge (x_5 \vee y) \wedge (x_6 \vee \bar{y}) \wedge (x_7 \vee \bar{y}) \wedge (x_8 \vee \bar{y}) \\
(x_9 \vee \bar{y}) \wedge (x_{10} \vee \bar{y})
\end{array}
$$

- **Replace matched pattern by**

$$
(x_6 \vee w) \wedge (x_7 \vee w) \wedge (x_8 \vee w) \wedge (x_9 \vee \bar{w}) \wedge (x_{10} \vee \bar{w}) \wedge (\bar{y} \vee \bar{w})
$$

Bounded Variable Addition

- **How can the model be reconstructed?**

Blocked Clauses

- A literal L in a clause C of a CNF F **blocks** C in F if for every clause $D \in F_{\bar{L}}$, the resolvent $(C \setminus \{L\}) \cup (D \setminus \{\bar{L}\})$ obtained from resolving C and D on L is a tautology
- A clause is **blocked** if it contains a literal that blocks it
- **Example** Consider the formula $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$
  - 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)

- **BCE** While there is a blocked clause C in a CNF F, remove C from F
- **Example** Consider $(a \lor b) \land (a \lor \bar{b} \lor \bar{c}) \land (\bar{a} \lor c)$
  - After removing either $(a \lor \bar{b} \lor \bar{c})$ or $(\bar{a} \lor c)$, the clause $(a \lor b)$ becomes blocked
    - no clause with either $\bar{b}$ or $\bar{a}$
  - An extreme case in which BCE removes all clauses
- **Proposition** BCE is confluent

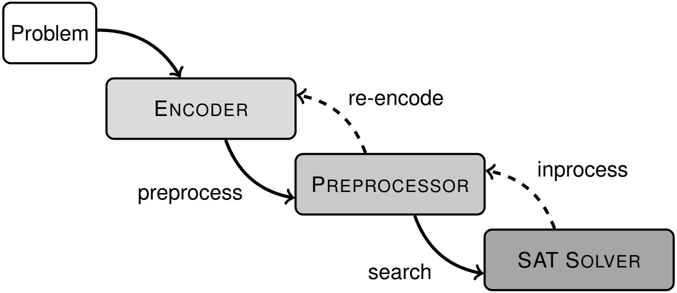
Blocked Clause Elimination

- **How can a model be reconstructed?**
- **Given F, we picked clause C with blocking literal L**
- **C was blocked with respect to $F_{\bar{L}}$**
- **A model J might falsify C**
- **How can it work?**

Simplification Techniques – The Bad and Powerful

- **Equisatisfiability Preserving Techniques**
  - (Bounded) variable elimination
  - Bounded variable addition
  - Blocked clause elimination
  - Covered clause elimination
  - Equivalent literal substitution
    - based on SCCs in binary implication graphs
    - based on structural hashing
    - based on Probing
  - Resolution asymmetric tautology elimination
- Need to store extra information to construct the model
- Not discussed here
  - Adding redundant clauses
  - Minimizing redundant clauses

Solving a Problem with SAT



- **Research topics:**
  - **encode problems into CNF**
  - **simplify the problem**
  - **and search for a solution or prove there does not exist one**
  - **simplification during search**
  - **automatically translate naive encodings into sophisticated encodings**