CNFs with Elementary Symmetric Clauses and their SAT Solving

Masahiko Sakai (Nagoya University)

Joint work with

Yohei Umano and Yoshizane Hino

3rd Austria - Japan Summer Workshop on Term Rewriting,
1 – 7 August 2010, Obergurgl

SAT: Satisfiability Problem

- Instance: Formula with boolean variables and logical connectives: ex. $x \cdot y \cdot \overline{\overline{x} \vee y}$
- Answer whether there exists a variable-assignment that makes the instance true.
- SAT solving
  - NP-complete but current SAT solver is practically efficient in many cases
  - Wide application
  - Dimacs format for CNFs in many solvers

```
p cnf 4 2
-1 2 3 0
-2 -3 4 0
```

$(\overline{x_1} \vee x_2 \vee x_3) \wedge$
$(\overline{x_2} \vee \overline{x_3} \vee x_4)$

This work

- Propose a technique that improves efficiency of SAT solvers
  - Introducing clauses based on Elementary Symmetric Function into CNF
  - Modifying DPLL algorithm on BCP and implication-graph construction
- Implementation and experiments

Elementary Symmetric Function

- **Symmetric function**: function whose value does not change even if inputs are permuted
- **ES-function**: $S_k^n(x_1, \ldots, x_n)$
  - Function that returns true iff exactly k inputs are true
  - Each symmetric function is representable by union of some ES-functions

CNF with ES-clauses

- **ES$_k^n$-clause: set of n literals**
  e.g. $\{x, y, z\}_k$ (interpreted as $S_k^3(x, y, z)$)
  c.f. OR-clause for ordinary clause
- **Intuitional merit of ES-clauses**
  - **ES$_1^n$-clause $\{x, y, z\}_1$ is equal to $O(n^2)$ OR-clauses $\{x, y, z\}$, $\{\overline{x}, \overline{y}\}$, $\{\overline{y}, \overline{z}\}$, $\{\overline{z}, \overline{x}\}$**
  - **Such a pattern is often used in SAT-coding**
- **Extension of dimacs format**
  !2 -2 3 -4 0 stands for ES$_2$-clause $\{\overline{x_2}, x_3, \overline{x_4}\}_2$

This work

- Propose a technique that improves efficiency of SAT solvers
  - Introducing clauses based on Elementary Symmetric Function into CNF
  - Modifying DPLL algorithm on BCP and implication-graph construction
- Implementation and experiments

DPLL algorithm(1/3)

- DPLL [Davis, Putnam, Logemann, Loveland 1960]
- Transition-rule formulation [Nieuwenhuis et al. 2004]
- Configuration $M \parallel F$ where
  - M: a list of (un-)marked literals z^d $\overline{x}$
    - z is true ($\top$), x is false ($\bot$),
    - y is unassigned (?)
  - F: a set of clauses
  - $level$: the number of marked variables
  - $Collision$: M contains (un-)marked l and $\neg l$

DPLL algorithm(2/3)

- **UnitPropagate (BCP)**
  $M \parallel F, C \vee l \Longrightarrow Ml \parallel F, C \vee l$ **if** $M \models \neg C$
- **Decide**
  $M \parallel F \Longrightarrow Ml^d \parallel F$ **if** l :?
- **Fail**
  $M \parallel F \Longrightarrow fail$
  if M **has no marked literals and collision.**
- **Backjump**
  $M *^d N \parallel F \Longrightarrow Ml \parallel F$
  if $\exists C, l\ [F \models C \vee l$ **and** $M \models \neg C]$
- **Learn**
  $M \parallel F \Longrightarrow M \parallel FC$ **if** $F \models C$

DPLL algorithm(3/3)

- **Example.**

$F = (a \vee b)(\overline{b} \vee c \vee d)(\overline{b} \vee x \vee y)(x \vee \overline{y})(\overline{x} \vee y)(\overline{x} \vee \overline{y})$

$\overline{a}^d\ b$	$\| F$	**by (Decide), (BCP)** $a \vee b$
$\overline{a}^d\ b\ \overline{c}^d\ d$	$\| F$	**by (Decide), (BCP)** $\overline{b} \vee c \vee d$
$\overline{a}^d\ b\ \overline{c}^d\ d\ \overline{x}^d$	$\| F$	**by (Decide)**
$\overline{a}^d\ b\ \overline{c}^d\ d\ \overline{x}^d\ y\ \overline{y}$	$\| F$	**by (BCP)** $\overline{b} \vee x \vee y$, $x \vee \overline{y}$
$\overline{a}^d\ b\ x$	$\| F\ (a \vee x)$	**by (Learn), (BJump)**
$\overline{a}^d\ b\ x\ y\ \overline{y}$	$\| F\ (a \vee x)$	**by (BCP)** $\overline{x} \vee y$, $\overline{x} \vee \overline{y}$
$\overline{x}$	$\| F\ (a \vee x)\ \overline{x}$	**by (Learn), (BJump)**
$\overline{x}\ \overline{y}\ a\ \overline{b}$	$\| F\ (a \vee x)\ \overline{x}$	**by (BCP)** $x \vee \overline{y}$, $a \vee x$, $b \vee x \vee y$
$\overline{x}\ \overline{y}\ a\ \overline{b}\ \overline{c}^d\ \overline{d}^d$	$\| F\ (a \vee x)\ \overline{x}$	**by (Decide)**

F **is satisfiable by** $x = \bot$, $y = \bot$, $a = \top$, ...

BCP

- **Inference of variable-values and detection of collision**

Condition	Action
$\#_{\perp} = n - 1$ and $\#_{\top} = 0$	sets the unassigned literal to be true
$\#_{\perp} = n$	collision detection

n: number of literals in the clause

$\#_{\perp}$: number of false-literals in the clause

$\#_{\top}$: number of true-literals in the clause

Extension: BCP(1/2)

- Easy according to the meaning of ES-clauses

Condition	Action
$\#_{\perp} = n - k$	sets all unassigned literals to be true
$\#_{\top} = k$	sets all unassigned literals to be false
$\#_{\perp} > n - k$	collision detection
$\#_{\top} > k$	

n: number of literals in the clause

$\#_{\perp}$: number of false-literals in the clause

Extension: BCP(2/2)

- **Example.** $F = \{\overline{x}, y, z\}_1 (y \vee \overline{z})$

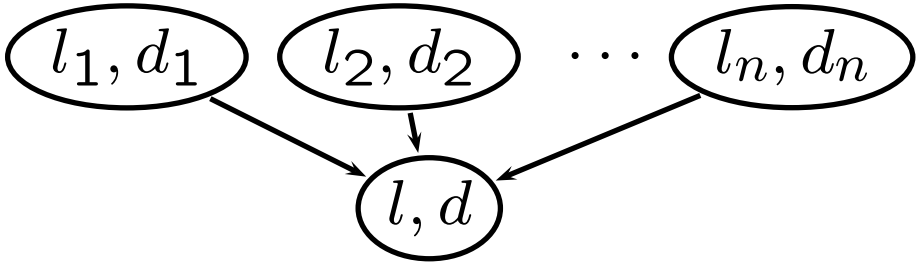
  $\overline{x}^d \quad \| F$ **by (decide)**

  $\overline{x}^d \, \overline{y} \, \overline{z} \, \| F$ **by (BCP)** $\{\overline{x}, y, z\}_1$

  ⋮

Implication graph(1/2)

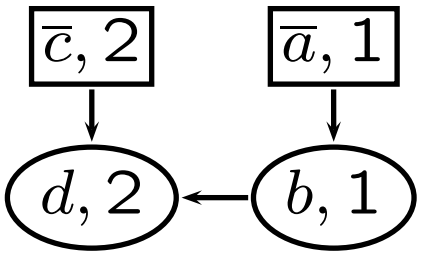
- Representation of history of BCP
- Necessary for (Backjump) and (Learn)
- Implication graph:
  - Node: pair of literal in M and level when created
  - Arrow: $\overline{l_1} \vee \cdots \vee \overline{l_n} \vee l$ iff



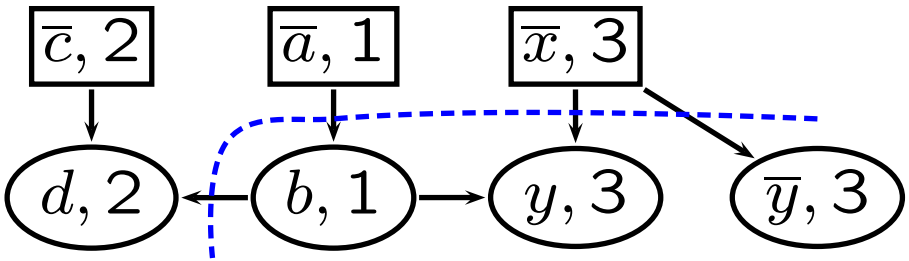
Implication graph(2/2)

- **Example.**

$$F = (a \vee b)(\overline{b} \vee c \vee d)(\overline{b} \vee x \vee y)(x \vee \overline{y})(\overline{x} \vee y)(\overline{x} \vee \overline{y})$$



$\overline{a}^d \; b \; \overline{c}^d \; d \; || \; F$



$\overline{a}^d \; b \; \overline{c}^d \; d \; \overline{x}^d \; y \; \overline{y} \; || \; F$

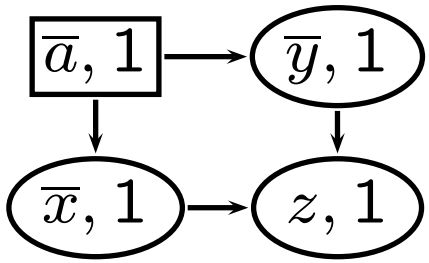
The cut produces a clause $a \vee x$ for (Learn) and also indicates literal for (Backjump)

$\overline{a}^d \; b \; x \; || \; F \; (a \vee x)$

Extension: implication graph

- No explicit extension is necesarry for implication graph
- Natural extension of construction is possible

$$F = \{\overline{a}, x, y\}_1 \; \{x, y, z\}_1$$



$$\overline{a}^d \; \overline{x} \; \overline{y} \; z \parallel F$$

This work

- Propose a technique that improves efficiency of SAT solvers
  - Introducing clauses based on Elementary Symmetric Function into CNF
  - Modifying DPLL algorithm on BCP and implication-graph construction
- Implementation and experiments

Implementation

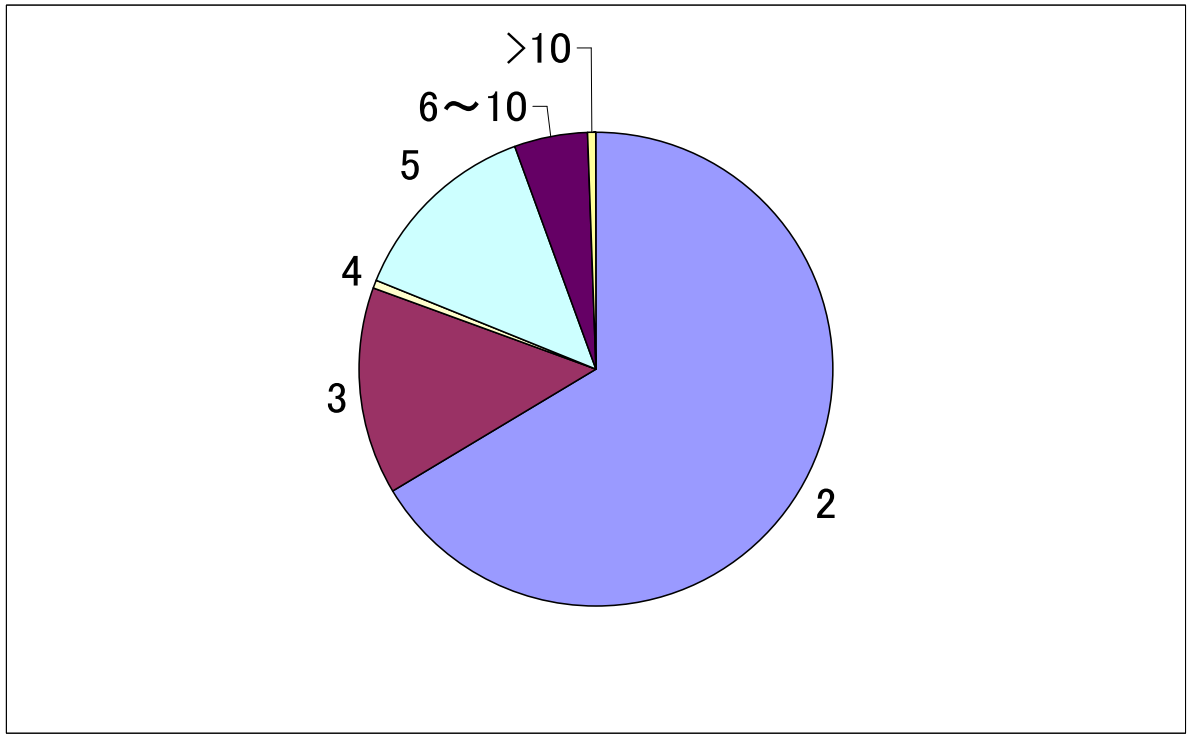
- Underlined SAT solver (nanosat_cnf): Constructed from almost scratch and with
  - DPLL algorithm with two counter method
  - Clause learning and forgetting
  - non-chronological backtrack (Backjump)
  - VSIDS heuristic for variable selection
- Extended SAT solver (nanosat$_k$)
  - ES$_k$-clauses
  - Extraction of ES$_1$-clauses from CNF

Experiment 1 (1/3)
– Extraction of ES_1-clauses –

- **SAT Competition 2003–2005,2007**
  - **1915 problems in industrial or crafted classes**
- **Number of clauses by extraction of ES_1-clauses**
  - **($\geq 40\%$) compression for 10% problems [A]**
  - **($\geq 10\%$) compression for 20% problems [B]**

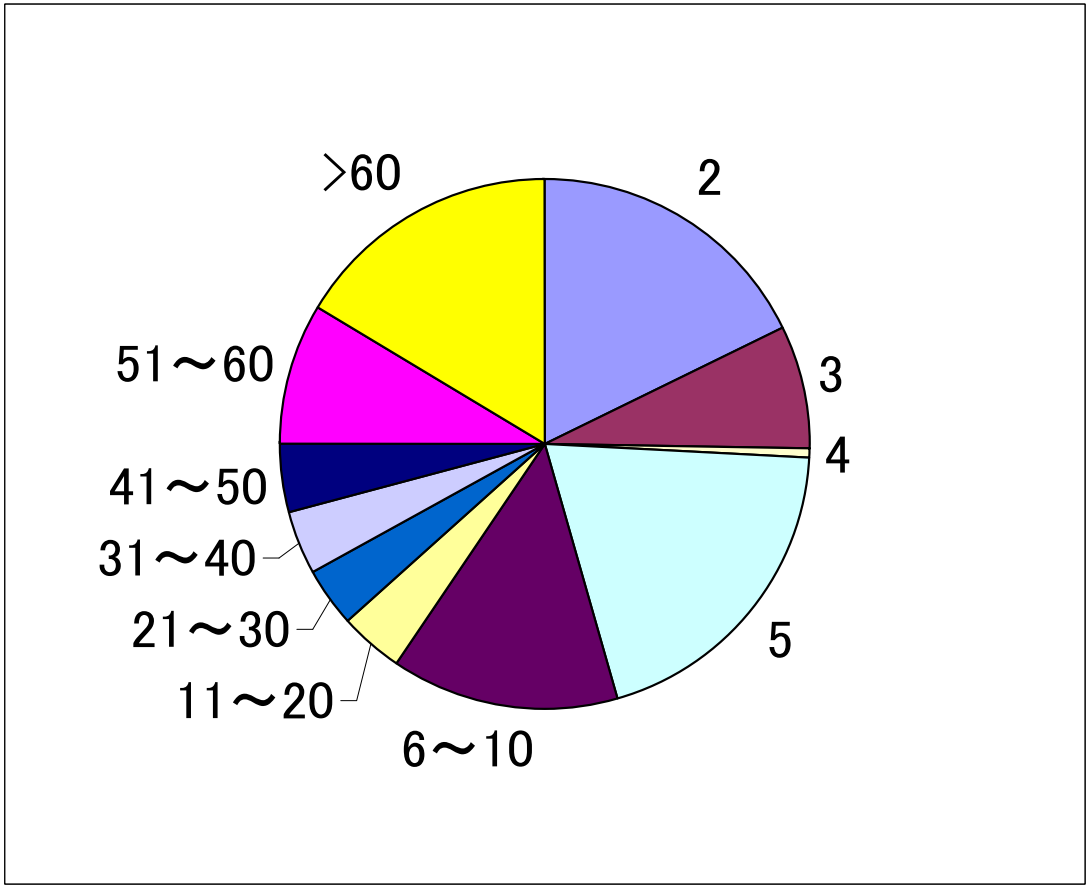
Experiment 1 (2/3)

- **Classification of ES_1-clauses by their length**



Experiment 1 (3/3)

- Classification of deleted OR-clauses by the length of produced ES_1-clauses



Experiment 2 (1/2)
– CNFs with ES_1-clauses –

- Comparation on speed
  - nanosat: including extraction time
  - nanosat_cnf
  - minisat
- Environment
  - cygwin gcc 4.3.2
  - Intel E8200 (2.66GHz x 2)
  - 2GB main memory
  - 30 sec timeout

Experiment 2 (2/2)

- Number of solved problems and execution time for SAT-competition problems

System	Problems [A]		Problems [B]	
	Solved	Time	Solved	Time
nanosat	21	3438	42	9482
	(conv 113 sec)		(conv 341 sec)	
nanosat_cnf	17	3517	40	9588
(minisat)	(19)	(3447)	(51)	(9127)

including timeouts

Exp. 3 – CNFs with ES_1-clauses –

	size	nanosat	nanosat_cnf	(minisat)	number of clauses
Sudoku	25^2	0.1	2.3	(0.4)	0.33%
	36^2	0.3	9.9	(2.2)	0.16%
	49^2	0.8	36.2	(7.6)	0.08%
	64^2	3.9	Err	(31.4)	0.05%
	81^2	8.0	Err	(Err)	0.03%
Magic square	4^2	0.2	0.3	(0.3)	34%
	5^2	1513.0	513.0	(Err)	30%
Picture logic	40×50	0.7	3.4	(0.2)	30%
	50×60	1.3	2.7	(2.0)	30%
Knight Patrol	6^2	0.3	0.4	(0.3)	68%
	7^2	0.7	1.0	(0.1)	69%
	8^2	78.7	77.0	(16.8)	69%
	9^2	6.5	7.9	(0.5)	69%
Hanoi	4	0.5	0.9	(0.1)	70%

direct generatation of ES_1-CNF

Experiment 4(1/2)
– CNFs with ES_k-clauses –

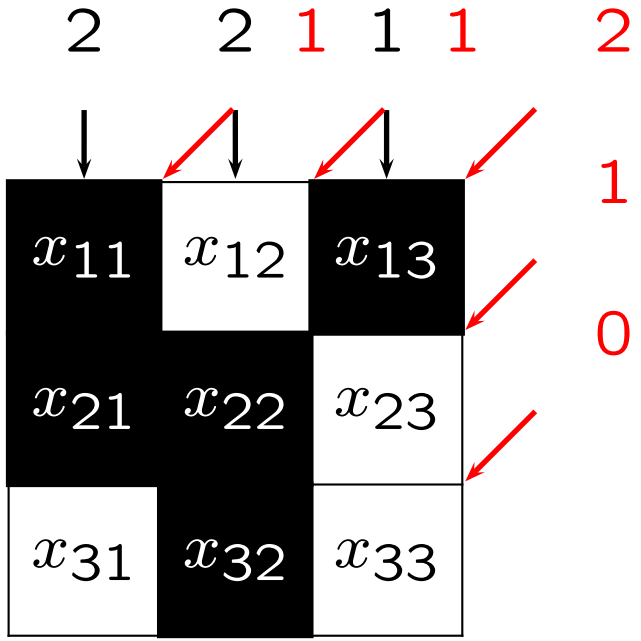
- Discrete computer tomography:
  - Instance: 4 sequences of numbers of black cells
  - Solution: The picture
  - ES_k-coding

    $\{x_{11}, x_{21}, x_{31}\}_2$

    $\{x_{12}, x_{22}, x_{32}\}_2$

    $\{x_{13}, x_{23}, x_{33}\}_1$

    ⋮



Experiment 4(2/2)

CT size		Clauses	Solved	Time
5×5	nanosat	28	100/100	0.05
	nanosat_cnf	212	100/100	(2.32+)0.20
10×10	nanosat	58	100/100	0.80
	nanosat_cnf	8818	100/100	(132+)10.38
15×15	nanosat	88	65/100	15735
	nanosat_cnf	320044	13/100	(4959+)28301

direct generatation of ES_k-CNF

Future work

- Implementation based on two pointer method
- Other useful clauses