Revisiting Decision Diagrams for SAT

Tom van Dijk[1] Rüdiger Ehlers[2] Armin Biere[1]

[1] Johannes Kepler University Linz, Austria
[2] University of Bremen, Germany

PoCR'17

Programatics for Constraint Reasoning
affiliated to CP'17, ILCP'17, and SAT'17

Melbourne, Australia

August 28, 2017

Motivation

- SAT solvers are used almost everywhere, but also …
- … increasing use of SAT solvers for hard combinatorial problems
  - Pythagorean Triples Problem (PTN) (CACM August 2017: The Science of Brute Force)
  - verifying arithmetic circuits
  - cryptanalysis
- features of these hard problems
  - "few variables" in the thousands (PTN has 7825)
  - no short resolution proofs (200 TB)
  - plain CDCL SAT solvers do not work
- binary decision diagrams (BDDs)
  - one fixed variable order / bad on large industrial instances
  - symbolic representation might give exponential speed-ups
  - much more memory and many more cores today
  - new paradigms such as *cube-and-conquer*

Pigeon Hole Problem (PHP$_n$)

- fit $n+1$ pigeons into n holes

$$\bigwedge_{i=1}^{n+1} \bigvee_{j=1}^{n} p_{i,j}$$

each pigeon i in at least one hole j

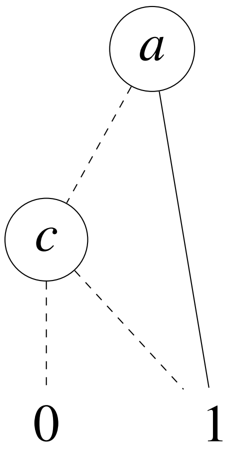
$$\bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n+1} \bigwedge_{k=1}^{n} (\overline{p_{i,k}} \vee \overline{p_{j,k}})$$

pigeon i and pidgeon j not in the same hole k

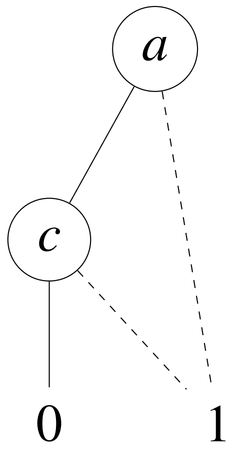
- [Haken'85] showed that all resolution refutations of PHP$_n$ are exponential
  - thus also hard for plain CDCL SAT solving ...
  - ... which in principle is as good as general resolution
  - so is a prototypical benchmark to test new ideas
- can be solved faster than with CDCL SAT solving by
  - directly building BDDs, or performing variable elimination over ZDDs
  - empirical results [ChatalicSimon'03] are old $\Rightarrow$ revisit
- actually trivial to solve by cardinality reasoning

Binary Decision Diagram (BDD) [Bryant'86]

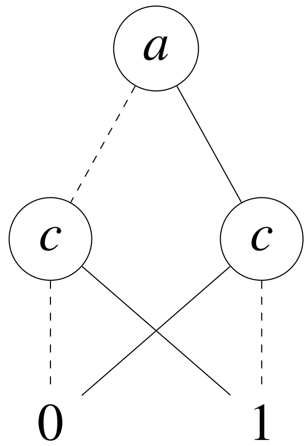
$(a \vee c) \quad \wedge \quad (\bar{a} \vee \bar{c}) \quad = \quad a \oplus c$ XOR



$\wedge$



$=$



$a \,?\, 1 : (c \,?\, 1 : 0)$ $\quad$ $a \,?\, (c \,?\, 0 : 1) : 1$ $\quad$ $a \,?\, (c \,?\, 0 : 1) : (c \,?\, 1 : 0)$

$\Diamond(a, 1, \Diamond(c, 1, 0))$ $\quad$ $\Diamond(a, \Diamond(c, 0, 1), 1)$ $\quad$ $\Diamond(a, \Diamond(c, 0, 1), \Diamond(c, 1, 0))$

BDD Apply Algorithm

$$
\begin{aligned}
0 \wedge 0 &= 0 \\
0 \wedge 1 &= 0 \\
1 \wedge 0 &= 0 \\
1 \wedge 1 &= 1 \\
\Diamond(x, f_1, f_2) \wedge \Diamond(x, g_1, g_2) &= \Diamond(x, (f_1 \wedge g_1), (f_2 \wedge g_2))
\end{aligned}
$$

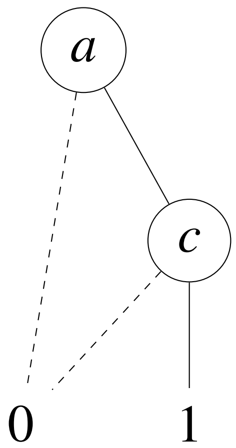
modulo

$$\Diamond(x, f, f) = f$$

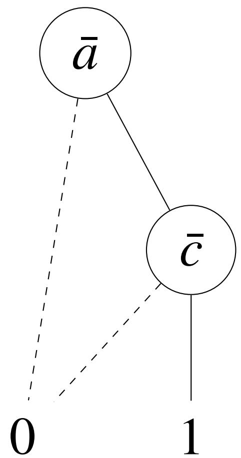
works the same for other boolean operators $\vee$, $\oplus$, $\ldots$

Zero Suppressed Decision Diagram (ZDD) [Minato'93,ChatalicSimon'03]

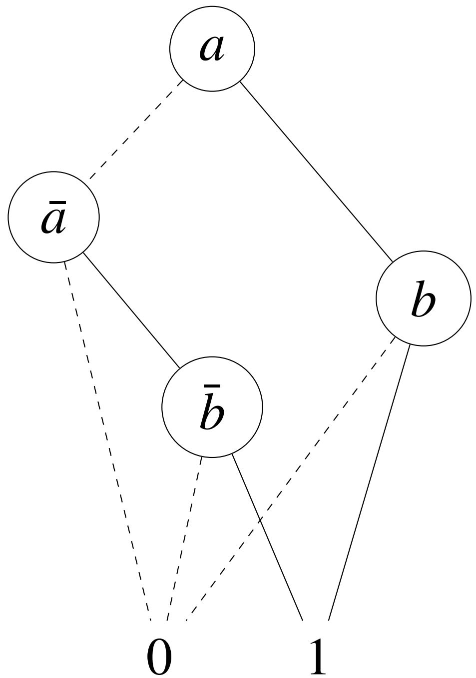
$$\{\{a,c\}\} \quad \cup \quad \{\{\bar{a},\bar{c}\}\} \quad = \quad \{\{a,c\},\{\bar{a},\bar{c}\}\}$$



$\cup$



$=$



$a.(c.1 \cup 0) \cup 0$ $\qquad$ $\bar{a}.(\bar{c}.1 \cup 0) \cup 0$

with $0 = \{\}$, $1 = \{\{\}\}$, $x.P = \{\{x\} \cup S \mid S \in P\}$

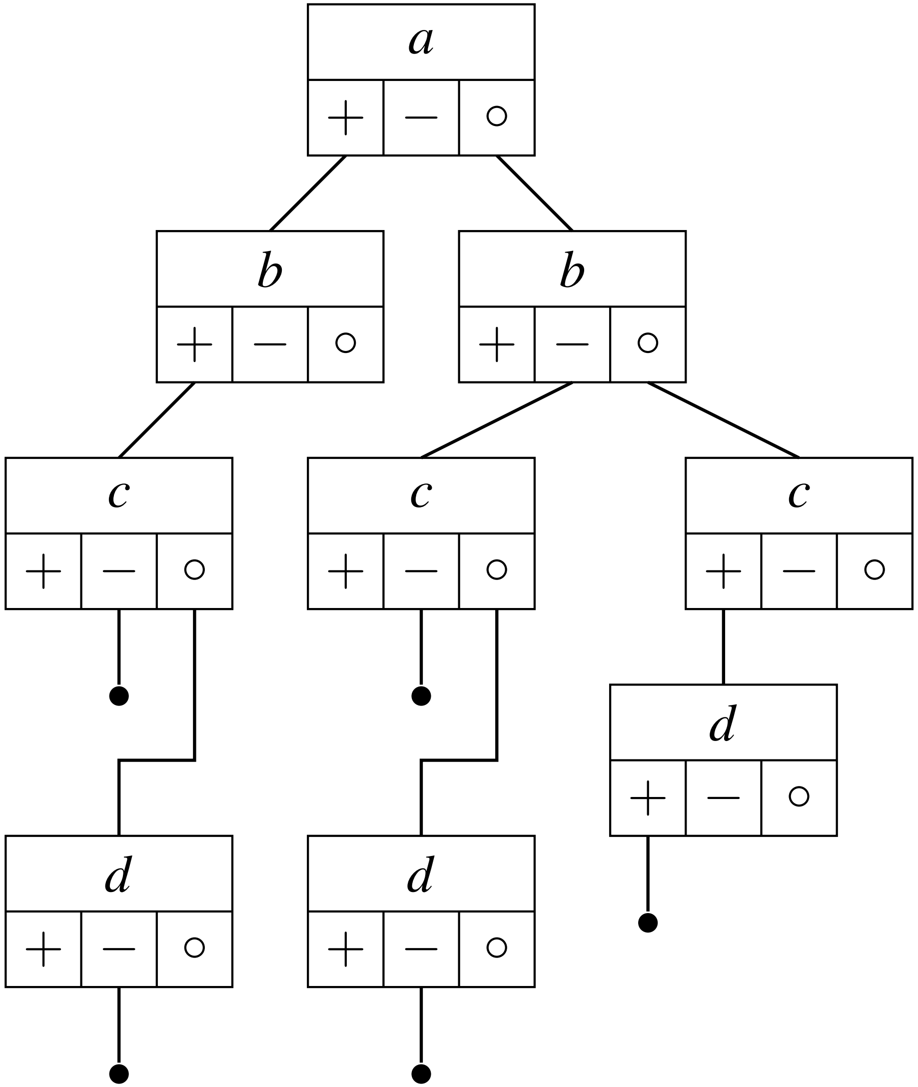
$\triangle(a, \triangle(c,1,0),0)$ $\qquad$ $\triangle(\bar{a}, \triangle(\bar{c},1,0),0)$ $\qquad$ $\triangle(a, \triangle(b,1,0), \triangle(\bar{a}, \triangle(\bar{b},1,0),0))$

CNF $\rightarrow$ ZDD (1/4)

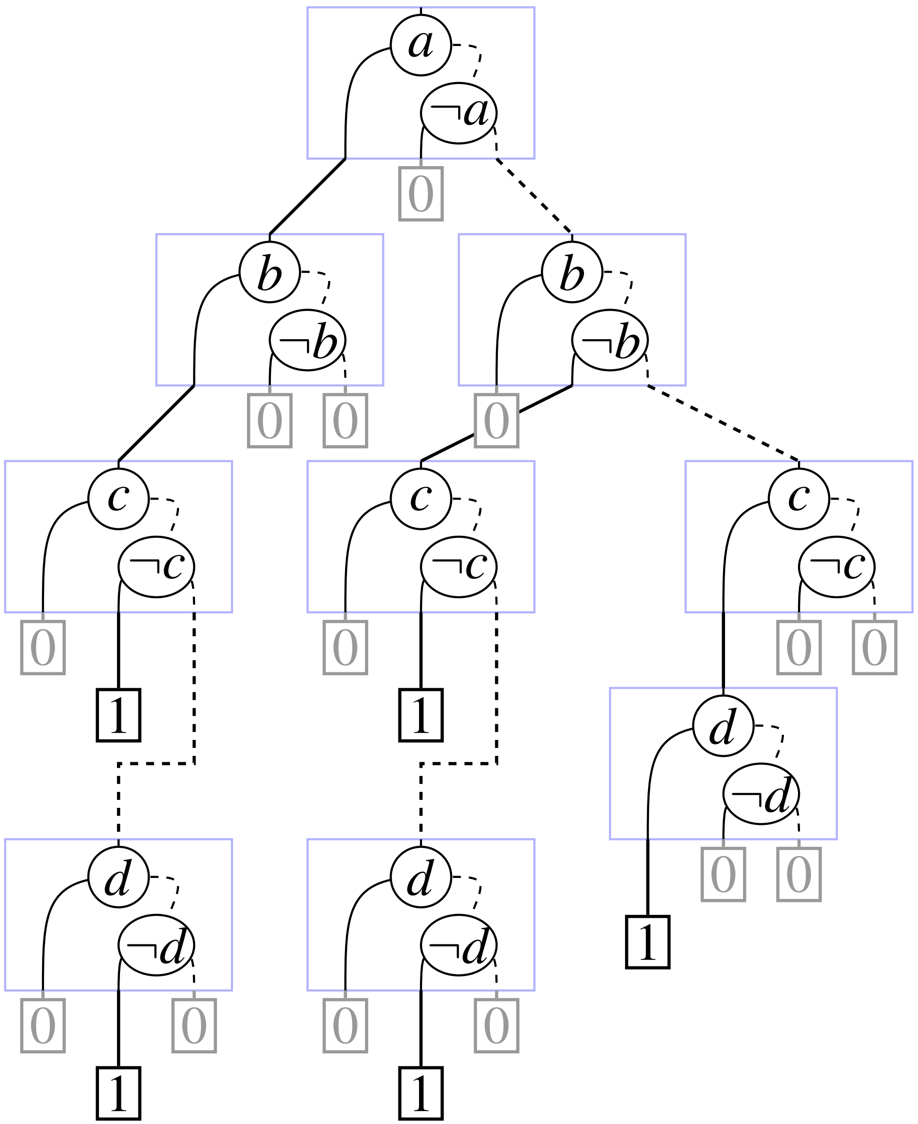
Example clauses:

- $a \vee b \vee \neg c$
- $a \vee b \vee \neg d$
- $\neg b \vee \neg c$
- $\neg b \vee \neg d$
- $c \vee d$

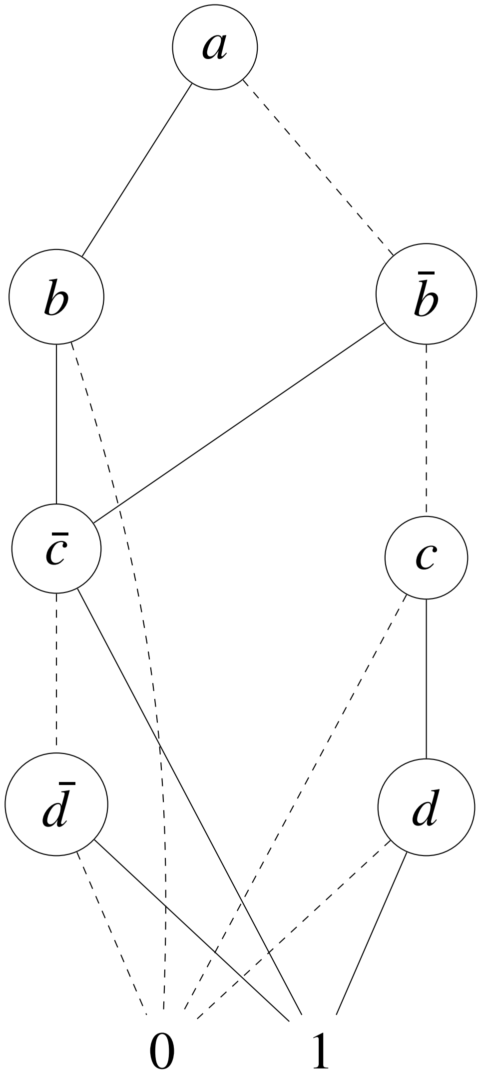
CNF → ZDD (2/4)



CNF → ZDD (3/4)



CNF → ZDD (4/4)



ZDD Apply Algorithm

$$
\begin{aligned}
0 \cup 0 &= 0 \\
0 \cup 1 &= 1 \\
1 \cup 0 &= 1 \\
1 \cup 1 &= 1 \\
\triangle(x, f_1, f_2) \cup \triangle(x, g_1, g_2) &= \triangle(x, (f_1 \cup g_1), (f_2 \cup g_2))
\end{aligned}
$$

modulo

$$
\triangle(x, 0, f) = 0
$$

works the same for other set operations $\cap$, $\backslash$, $\ldots$

again with $0 = \{\}$ and $1 = \{\{\}\}$

CNF $\rightarrow$ BDD

- parse CNF and build individual BDD for each clause
- keep a BDD representing conjunction of all previously read clauses
- add BDD for new clause with (parallelized) BDD apply algorithm

CNF $\rightarrow$ ZDD

- parse whole CNF into integer array
- divide-and-conquer recursive union of clauses as ZDD (parallelized)
- base case is to build a ZDD for individual clauses

ZDD $\rightarrow$ BDD

- build BDDs recursively over whole ZDD

  $$\mathrm{zdd2bdd}(\triangle(x, f_1, f_2)) \;=\; \mathrm{zdd2bdd}(f_2) \vee \Diamond(x, \mathrm{zdd2bdd}(f_1), 0)$$

- again using work-stealing and task parallelism in “$\vee$” and “$\mathrm{zdd2bdd}(\cdots)$”

Experiments CNF → ZDD → BDD for PHP (adding clauses one by one)

cores	ph10	ph11	ph12	ph13	ph14	ph15	ph16	ph17	ph18	ph19	ph20
1	0.43	0.46	0.50	0.61	0.90	1.54	3.02	6.31	18.18	37.15	58.95
2	0.62	0.63	0.73	0.82	1.04	1.19	2.20	4.29	9.61	22.27	33.38
4	0.61	0.63	0.67	0.72	0.87	1.17	1.77	3.15	8.45	18.55	143.05
6	0.37	0.41	0.39	0.47	0.59	0.83	1.31	2.44	5.87	12.50	98.28
8	0.70	0.71	0.76	0.81	0.87	1.06	1.45	2.45	5.04	10.98	132.32
10	0.79	0.84	0.87	0.94	1.06	1.30	1.98	2.88	5.63	10.52	93.82
12	0.26	0.29	0.32	0.42	0.54	0.83	1.28	2.35	5.27	10.06	108.49
14	0.34	0.37	0.42	0.48	0.63	0.86	1.34	2.31	4.83	9.22	102.03
16	0.81	0.81	0.86	0.91	1.05	1.31	1.79	2.72	4.92	9.01	57.23

Experiments CNF → ZDD → BDD for PHP (recusive ZDD computation)

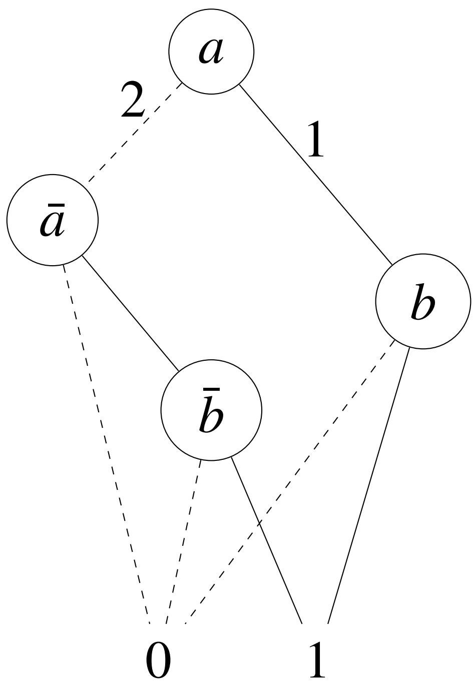
cores	ph10	ph11	ph12	ph13	ph14	ph15	ph16	ph17	ph18	ph19	ph20
1	0.93	0.96	1.00	1.11	1.39	2.04	3.52	6.80	18.80	38.46	54.50
2	0.94	0.94	1.00	1.08	1.27	1.67	2.53	4.62	11.27	23.04	37.13
4	0.94	0.96	0.98	1.04	1.17	1.47	2.02	3.61	8.99	16.36	29.42
6	0.47	0.47	0.48	0.53	0.63	0.95	1.37	2.44	6.96	14.07	105.88
8	0.18	0.17	0.20	0.23	0.35	0.59	1.00	1.83	5.39	11.08	138.65
10	0.54	0.50	0.55	0.60	0.68	0.91	1.47	2.76	5.63	11.57	174.56
12	0.52	0.54	0.57	0.63	0.73	1.00	1.48	2.79	5.84	11.90	91.31
14	0.49	0.49	0.53	0.56	0.69	0.89	1.32	2.62	5.68	10.90	94.71
16	0.41	0.46	0.47	0.56	0.65	0.89	1.29	2.34	5.15	10.74	19.01

New Compact Notation for ZDD encoding CNF

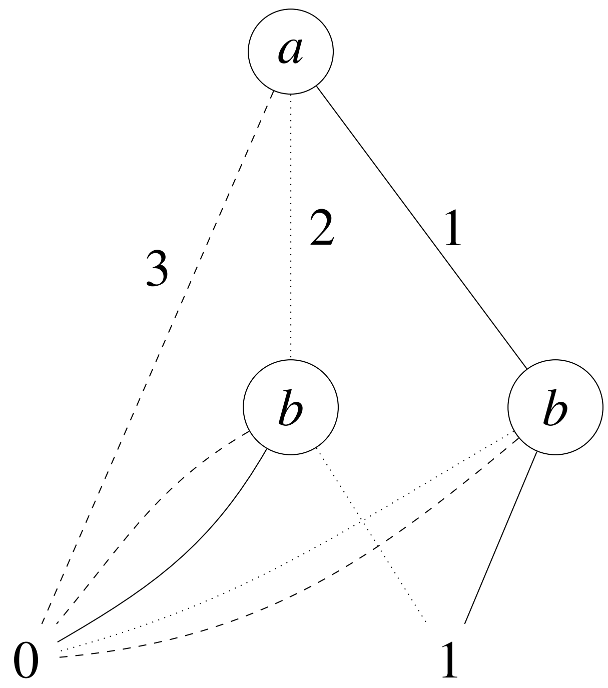
$$\bigtriangledown(v, f_1, f_2, f_3) = \bigtriangleup(v, f_1, \bigtriangleup(\bar{v}, f_2, f_3))$$

assuming no node $\bigtriangleup(v, \bigtriangleup(\bar{v}, \ldots, \ldots), \ldots)$ exists

corresponds to CNF containing trivial clauses with both v and $\bar{v}$



$\bigtriangleup(a, \bigtriangleup(b, 1, 0), \bigtriangleup(\bar{a}, \bigtriangleup(\bar{b}, 1, 0), 0))$



$\bigtriangledown(a, \bigtriangledown(b, 1, 0, 0), \bigtriangledown(\bar{b}, 1, 0, 0))$

Union with New Notation

$$
\begin{aligned}
0 \cup 0 &= 0 \\
0 \cup 1 &= 1 \\
1 \cup 0 &= 1 \\
1 \cup 1 &= 1 \\
\triangle(x, f_1, f_2, f_3) \cup \triangle(x, g_1, g_2, g_3) &= \bigtriangledown(x, (f_1 \cup g_1), (f_2 \cup g_2), (f_3 \cup g_3))
\end{aligned}
$$

modulo

$$
\bigtriangledown(u, f_1, f_2, f_3) \cup \bigtriangledown(v, g_1, g_2, g_3) = \bigtriangledown(u, f_1, f_2, f_3) \cup \bigtriangledown(u, 0, 0, \bigtriangledown(v, g_1, g_2, g_3))
$$

if $u < v$

Subsumption with New Notation

$$
\begin{aligned}
0 \setminus\!\!\setminus f &= 0 \\
1 \setminus\!\!\setminus 1 &= 1
\end{aligned}
$$

$$
\begin{aligned}
1 \setminus\!\!\setminus \bigtriangledown(x, f_1, f_2, f_3) &= 1 \\
\bigtriangledown(x, f_1, f_2, f_3) \setminus\!\!\setminus 1 &= 1 \\
\bigtriangledown(x, f_1, f_2, f_3) \setminus\!\!\setminus 0 &= \bigtriangledown(x, f_1, f_2, f_3)
\end{aligned}
$$

$$
\bigtriangledown(x, f_1, f_2, f_3) \setminus\!\!\setminus \bigtriangledown(x, g_1, g_2, g_3) = \bigtriangledown(x,\ (f_1 \setminus\!\!\setminus g_1) \setminus\!\!\setminus g_3,\ (f_2 \setminus\!\!\setminus g_2) \setminus\!\!\setminus g_3,\ f_3 \setminus\!\!\setminus g_3)
$$

Self-Subsumption Makes ZDD / CNF Subsumption-Free

$$
\begin{aligned}
SF(0) &= 0 \\
SF(1) &= 1
\end{aligned}
$$

$$
SF(\bigtriangledown(x, f_1, f_2, f_3)) = \bigtriangledown(x,\ SF(f_1) \setminus\!\!\setminus SF(f_3),\ SF(f_2) \setminus\!\!\setminus SF(f_3),\ SF(f_3))
$$

Subsumption-Free Union (Logical Conjunction)

$$
\begin{aligned}
0 \cup_S f &= f \\
1 \cup_S f &= 1 \\
f \cup_S 0 &= f \\
f \cup_S 1 &= 1
\end{aligned}
$$

$$
\begin{aligned}
\bigtriangledown(x, f_1, f_2, f_3) \cup_S \bigtriangledown(x, g_1, g_2, g_3) &= \\
&\bigtriangledown(x,\ (f_1 \cup_S g_1) \setminus (f_3 \cup_S g_3),\ (f_2 \cup_S g_2) \setminus (f_3 \cup_S g_3),\ (f_3 \cup_S g_3))
\end{aligned}
$$

Subsumption-Free Clause Distribution (Logical Disjunction)

$$
\begin{aligned}
0 \times_S f &= 0 \\
1 \times_S f &= f \\
f \times_S 0 &= 0 \\
f \times_S 1 &= f
\end{aligned}
$$

$$
\begin{aligned}
\bigtriangledown(x, f_1, f_2, f_3) \times_S \bigtriangledown(x, g_1, g_2, g_3) &= \\
&\bigtriangledown(x,\ ((f_1 \times_S g_1) \cup_S (f_1 \times_S g_3) \cup_S (f_3 \times_S g_1)) \setminus (f_3 \times_S g_3), \\
&\quad ((f_2 \times_S g_2) \cup_S (f_2 \times_S g3) \cup_S (f_3 \times_S g_2)) \setminus (f_3 \times_S g_3),\ (f_3 \times_S g_3))
\end{aligned}
$$

Clause Distribution – Davis Putnam Procedure (DP)

- eliminate variables from CNF one-by-one [DavisPutnam'58]
- resolve all clauses with variable x with all clauses with $\bar{x}$
- add resolvents after removing clauses with x and $\bar{x}$

Symbolic Variable Elimination

- DP but on ZDD encoded CNF [ChatalicSimon'03]
- was combined with subsumption removal and solves PHP problems
- high compression ratio #clauses / #nodes

Bounded Variable Elimination (BVE)

- only eliminate variables if CNF size does not increase [EenBiere'05]
- combined with subsumption removal
- most effective preprocessing

Bounded Symbolic Variable Elimination

- symbolic variable elimination / clause distribution
- eagerly eliminate variables which do not increase size
- if all variable increase size eliminate one with smallest increase

Experiments

strategy	cores	ph10	ph11	ph12	ph13	ph14	ph15	ph16	ph17	ph18	ph19	ph20
original	1	40	218	1231	–	–	–	–	–	–	–	–
	8	5	21	117	673	–	–	–	–	–	–	–
	16	3	13	64	357	2002	–	–	–	–	–	–
node	1	2	3	6	10	15	26	43	64	99	147	209
	8	2	4	7	11	17	26	44	66	100	146	206
	16	3	5	9	13	22	35	55	83	124	182	260
clause	1	53	351	2108	–	–	–	–	–	–	–	–
	8	7	39	215	1305	–	–	–	–	–	–	–
	16	5	24	120	713	–	–	–	–	–	–	–

substantial amount of time spent in “trial elimination attempts” for “node” and “clause” size bounding

Conclusion

- started to revisit both BDD and ZDD based SAT solving
- contribution: simpler notation, parallel, bounded symbolic variable elimination
- but did not cover parallelization in BDD library **Sylvan** by Tom van Dijk

Things we tried without Success

- played with different variable orderings for PHP
- stronger (more expensive) simplifiers (not just subsumption)
- other similar hard combinatorial examples

Future Work

- symbolic cube-and-conquer
- low-level parallelization of CDCL SAT solvers
- unit propagation on BDDs / ZDDs