Preprocessing and Inprocessing Techniques in SAT

Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University, Linz, Austria

joint work with

Marijn Heule and Matti Järvisalo on SAT preprocessing
Florian Lonsing and Martina Seidl on QBF preprocessing

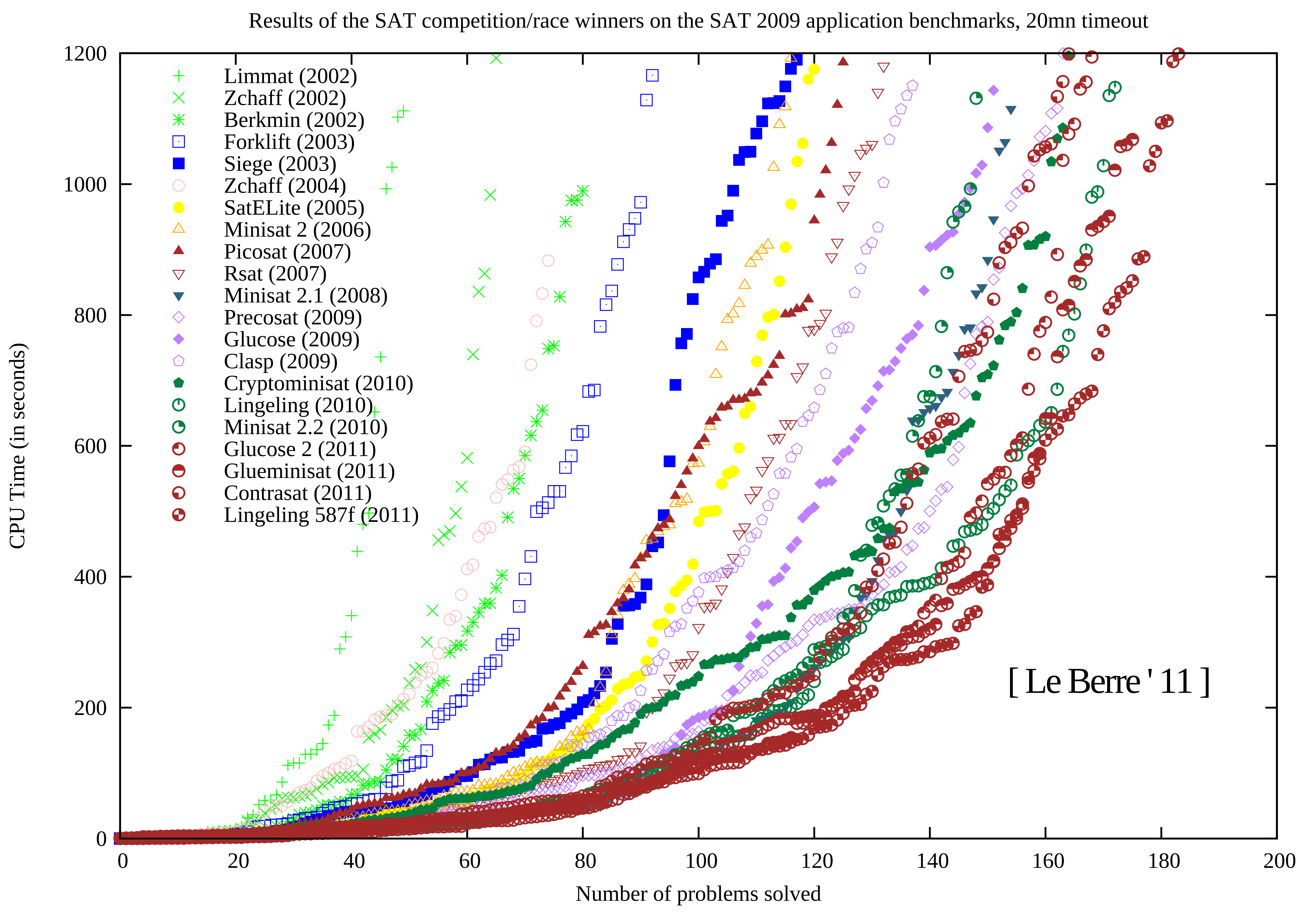
Haifa Verification Conference
Thursday, December 8, 2011
Haifa, Israel

Dress Code as Satisfiability Problem

- propositional logic:
  - variables **tie** **shirt**
  - negation $\neg$ (not)
  - disjunction $\vee$ disjunction (or)
  - conjunction $\wedge$ conjunction (and)

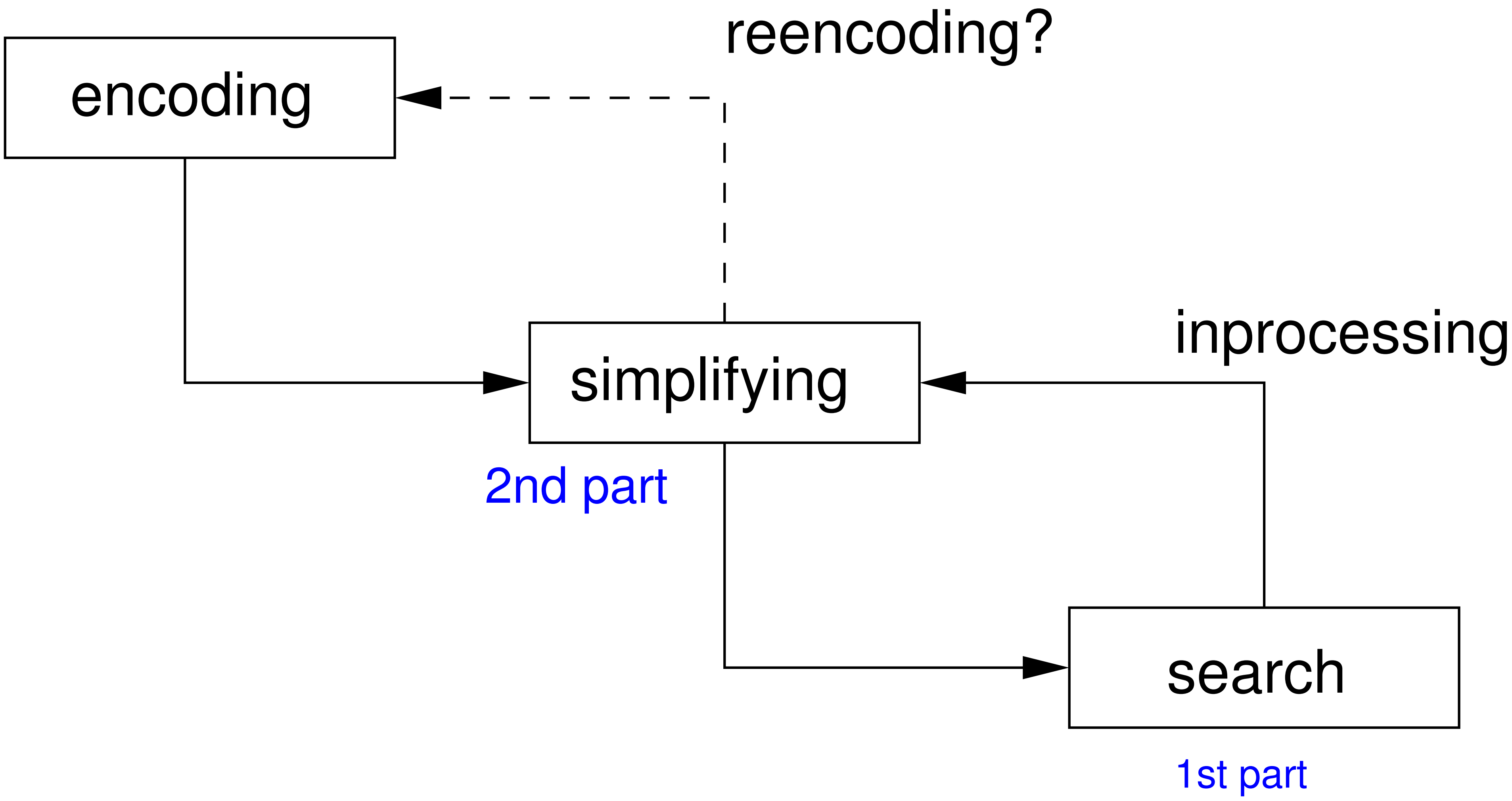
- three conditions / clauses:
  - clearly one should not wear a **tie** without a **shirt** $\neg \textbf{tie} \vee \textbf{shirt}$
  - not wearing a **tie** nor a **shirt** is impolite $\textbf{tie} \vee \textbf{shirt}$
  - wearing a **tie** and a **shirt** is overkill $\neg(\textbf{tie} \wedge \textbf{shirt}) \equiv \neg \textbf{tie} \vee \neg \textbf{shirt}$

- is the formula $(\neg \textbf{tie} \vee \textbf{shirt}) \wedge (\textbf{tie} \vee \textbf{shirt}) \wedge (\neg \textbf{tie} \vee \neg \textbf{shirt})$ satisfiable?



Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout
CPU Time (in seconds)
Number of problems solved
0
200
400
600
800
1000
1200
0
20
40
60
80
100
120
140
160
180
200
Limmat (2002)
Zchaff (2002)
Berkmin (2002)
Forklift (2003)
Siege (2003)
Zchaff (2004)
SatELite (2005)
Minisat 2 (2006)
Picosat (2007)
Rsat (2007)
Minisat 2.1 (2008)
Precosat (2009)
Glucose (2009)
Clasp (2009)
Cryptominisat (2010)
Lingeling (2010)
Minisat 2.2 (2010)
Glucose 2 (2011)
Glueminisat (2011)
Contrasat (2011)
Lingeling 587f (2011)
[Le Berre ' 11]

What is Practical SAT Solving?



DPLL Procedure

[DavisLogemannLoveland'62]

$DPLL(F)$

$F := BCP(F)$ boolean constraint propagation

if $F = \top$ **return** *satisfiable*

if $\bot \in F$ **return** *unsatisfiable*

pick remaining variable x and literal $l \in \{x, \neg x\}$

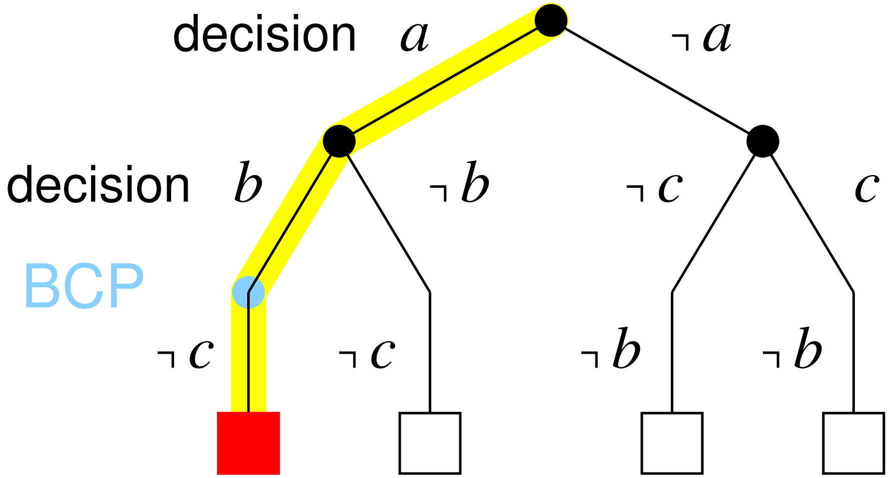
if $DPLL(F \wedge \{l\})$ returns *satisfiable* **return** *satisfiable*

return $DPLL(F \wedge \{\neg l\}$

$a = 1$

$b = 1$

$c = 0$



clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

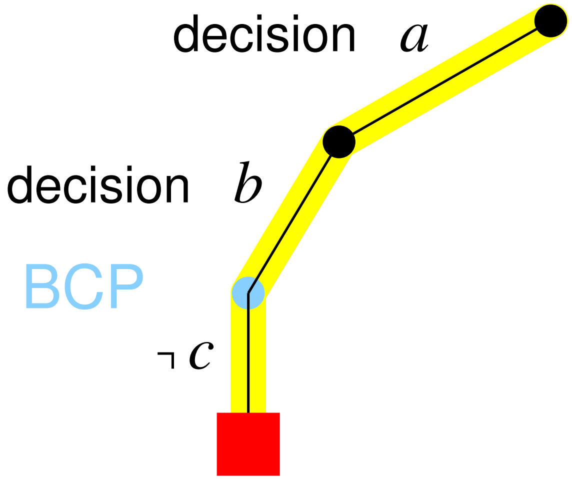
Conflict Driven Clause Learning (CDCL)

Grasp [MarquesSilvaSakallah'96]

$a = 1$

$b = 1$

$c = 0$



clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

learn $\neg a \vee \neg b$

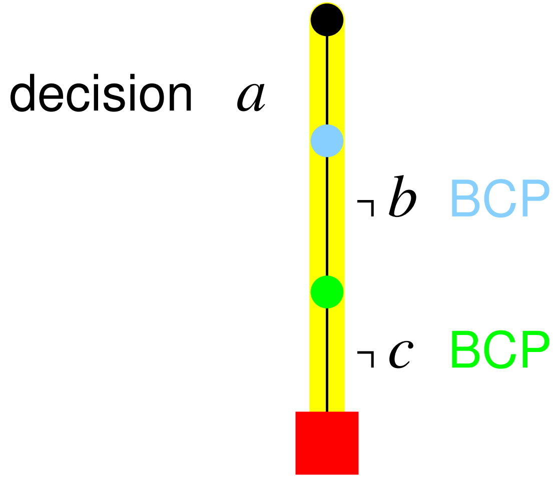
Conflict Driven Clause Learning (CDCL)

Grasp [MarquesSilvaSakallah'96]

$a = 1$

$b = 0$

$c = 0$



clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

$\neg a \vee \neg b$

learn $\neg a$

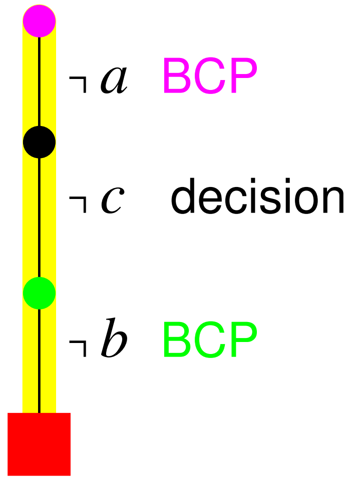
Conflict Driven Clause Learning (CDCL)

Grasp [MarquesSilvaSakallah'96]

$a = 1$

$b = 0$

$c = 0$



clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

$\neg a \vee \neg b$

$\neg a$

learn c

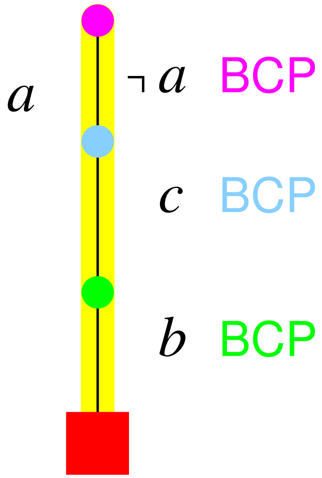
Conflict Driven Clause Learning (CDCL)

Grasp [MarquesSilvaSakallah'96]

$a = 1$

$b = 0$

$c = 0$



clauses

$\neg a \vee \neg b \vee \neg c$
$\neg a \vee \neg b \vee c$
$\neg a \vee b \vee \neg c$
$\neg a \vee b \vee c$
$a \vee \neg b \vee \neg c$
$a \vee \neg b \vee c$
$a \vee b \vee \neg c$
$a \vee b \vee c$

$\neg a \vee \neg b$
$\neg a$
c

learn $\perp$

empty clause

- failed literal *probing*
- variable elimination (VE)
- inprocessing
- lazy hyper binary resolution
- blocked clause elimination (BCE)
  - for SAT
  - for QBF
- hidden tautologies elimination (HTE)
- unhiding

Failed Literal Probing

we are still working on tracking down the origin before [Freeman'95] [LeBerre'01]

- key technique in look-ahead solvers such as Satz, OKSolver, March
  - failed literal probing at all search nodes
  - used to find the best decision variable and phase
- simple algorithm
  1. assume literal l, propagate (BCP), if this results in conflict, add unit clause $\neg l$
  2. continue with all literals l until *saturation* (nothing changes)
- quadratic to cubic complexity
  - BCP linear in the size of the formula — 1st linear factor
  - each variable needs to be tried — 2nd linear factor
  - and tried again if some unit has been derived — 3rd linear factor

Extensions

- lifting
  - complete case split: literals implied in all cases become units
  - similar to Stålmark's method and Recursive Learning [PradhamKunz'94]
- asymmetric branching
  - assume all but one literal of a clause to be false
  - if BCP leads to conflict remove originally remaining unassigned literal
  - implemented for a long time in MiniSAT but switched off by default
- generalizations:
  - vivification [PietteHamadiSais ECAI'08]
  - distillation [JinSomenzi'05][HanSomenzi DAC'07] probably most general (+ tries)

Bounded Variable Elimination (VE)

[Biere'04][SubbarayanPradhan'04][EénBiere SAT'05]

- goes back to original Davis & Putnam algorithm [DP'60]
  - eliminate variable x by adding all resolvents with x as pivot ...
  - ... and removing all clauses in which x or $\neg x$ occurs
  - eliminating one variable is in the worst case quadratic
- bounded = apply only if increment in size is small
  - Quantor [Biere'03,Biere'04] bound increase in terms of literals (priority queue)
  - NiVER [SubbarayanPradhan'04] do non increase number of clauses (round-robin)
  - SatELite [EénBiere'05] do not increase number of clauses (priority queue)

- fast *subsumption* and *strengthening* [Biere'04][EénBiere'05]
  - backward subsumption: traverse clauses of least occurring literal
  - forward subsumption: one-watched literal scheme [Zhang'05]
  - 1st and 2nd level signatures $=$ Bloom-filters for faster checking
  - strengthen clauses through *self-subsuming resolution*

- *functional substitution*
  - if x has a functional dependency, e.g. Tseitin translation of a gate
  - then only resolvents using exactly one "gate clause" need to be added

$$\overbrace{(\bar{x} \vee a)(\bar{x} \vee b)(x \vee \bar{a} \vee \bar{b})}^{x=a\wedge b}(x \vee c)(x \vee d)(\bar{x} \vee e)(\bar{x} \vee f)$$ 7 clauses

$$(a \vee c)(a \vee d)(b \vee c)(b \vee d)(\bar{a} \vee \bar{b} \vee e)(\bar{a} \vee \bar{b} \vee f)\cancel{(c \vee e)(c \vee f)(d \vee e)(d \vee f)}$$ 6 + 4 clauses

- **preprocessing** can be extremely beneficial
  - most SAT competition solvers use variable elimination (VE) [EénBiere SAT'05]
  - equivalence & XOR reasoning beneficial
  - probing / failed literal preprocessing / hyper binary resolution useful
  - however, even though polynomial, can not be run until completion

- **inprocessing**: simple idea to benefit from full preprocessing without penalty
  - "preempt" preprocessors after some time
  - resume preprocessing between restarts
  - limit preprocessing time in relation to search time

- allows to use costly preprocessors
  - without increasing run-time “much” in the worst-case
  - still useful for benchmarks where these costly techniques help
  - good examples: probing and distillation even VE can be costly
- additional benefit:
  - makes learned units / equivalences / implications available to preprocessing
  - particularly interesting if preprocessing simulates encoding optimizations
- danger of hiding “bad” implementation though ...
- ... and hard(er) to debug

Background on Hyper Binary Resolution (HBR)

- one Hyper Binary Resolution step [Bacchus-AAAI02]

$$\frac{(l \vee l_1 \vee \cdots \vee l_n) \quad (\overline{l_1} \vee l') \quad \cdots \quad (\overline{l_n} \vee l')}{(l \vee l')}$$

  - combines multiple resolution steps into one
  - special case “hyper unary resolution” where $l = l'$
  - HBR stronger than unit propagation if it is repeated until (confluent) closure
  - equality reduction: if $(a \vee \overline{b}), (\overline{a} \vee b) \in f$ replace a by b in f substitution

- can be simulated by unit propagation [BacchusWinter-SAT03]

if $(l \vee l') \in \text{HypBinRes}(f)$ then $l' \in \text{UnitProp}(f \wedge \overline{l})$ or vice versa

- repeated probing, c.f. HypBinResFast [GershmanStrichman-SAT05]

Previous Optimizations

[BacchusWinter-SAT03][GershmanStrichman-SAT05]

- maintain acyclic and transitively-reduced binary implication graph
  - acyclic: decomposition in *strongly connected components* (SCCs)

    $(\overline{a} \vee b)(\overline{b} \vee c)(\overline{c} \vee a) \wedge R$ equisatisfiable to $R[a/b, a/c]$
  - transitively-reduced: remove resp. do not add transitive edges

- not all literals have to be probed
  - if $l \in \mathrm{UnitProp}(r)$ and $\mathrm{UnitProp}(r)$ does not produce anything

    $\Rightarrow$ no need to probe l until next unit or implication is found
  - at least with respect to units it is possible to focus on roots
  - tree based probing in March

- current algorithms too expensive to run until completion we are working on this

Observations

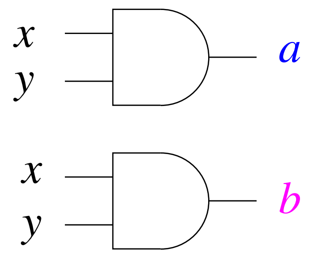
- **time** complexity: seems to be at least quadratic, unfortunately also in practice

- **space** complexity: unclear, at most quadratic, linear?

- hyper binary resolution simulates structural hashing for AND gates a and b

$$F \quad \equiv \quad (\overline{a} \vee x)(\overline{a} \vee y)(a \vee \overline{x} \vee \overline{y}) \quad (\overline{b} \vee x)(\overline{b} \vee y)(b \vee \overline{x} \vee \overline{y}) \qquad \cdots$$

$$\frac{(\overline{a} \vee x)(\overline{a} \vee y)(b \vee \overline{x} \vee \overline{y})}{(\overline{a} \vee b)} \qquad \frac{(\overline{b} \vee x)(\overline{b} \vee y)(a \vee \overline{x} \vee \overline{y})}{(\overline{b} \vee a)}$$



  can also be seen by $b \in \mathrm{UnitProp}(F \wedge a)$ and $a \in \mathrm{UnitProp}(F \wedge b)$

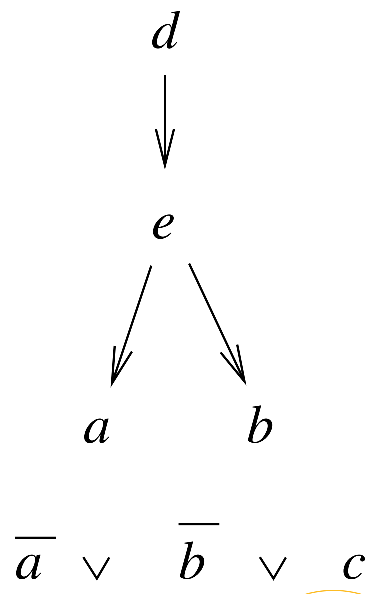
- can not simulate structural hashing of ITE or (binary) XOR gates
  - need equivalence reasoning and/or double look ahead

Lazy Hyper Binary Resolution (LHBR)

- learn binary clauses lazily or on-the-fly
  - in the innermost (!) BCP loop
  - actually only necessary during failed literal probing

- whenever a **large** clause $(a_1 \vee \cdots \vee a_m \vee c)$ with $m \geq 2$ becomes a reason for c
  - partial assignment σ with $\sigma(a_i) = 0$ and $\sigma(c) = 1$
  - check whether exists literal d dominating all $\overline{a_i}$
  - in implication graph restricted to binary clauses
  - which is a tree !

- learn $(\overline{d} \vee c)$ if such a dominator exists better $(\bar{e} \vee c)$



Status of (L)HBR

- theory
  - at least as strong as structural hashing with AIGs
  - might derive additional important implication
- practice
  - empirically proven that simulation of structural hashing really works
  - but current algorithms are far slower (100x)
  - example: combinational miter for intel048 from HWMCC ($>$ 200k gates) with itself can not be solved by Lingeling in a day, with structural hashing in half a second
- even in combination with advanced probing techniques
  - such as *tree based lookahead* as implemented by Marijn Heule in March
  - probably need eager/online substitution current hypothesis

Blocked Clause Elimination (BCE)

one clause $C \in F$ with l

all clauses in F with $\bar{l}$

fix a CNF F

$$\bar{l} \vee \bar{a} \vee c$$

$$a \vee b \vee l$$

$$\bar{l} \vee \bar{b} \vee d$$

all resolvents of C on l are tautological $\Rightarrow$ C can be removed

Proof assume assignment σ satisfies $F \backslash C$ but not C

can be extended to a satisfying assignment of F by flipping value of l

[JärvisaloBiereHeule-TACAS10] [JärvisaloBiereHeule-JAR1X]

COI Cone-of-Influence reduction

MIR Monontone-Input-Reduction

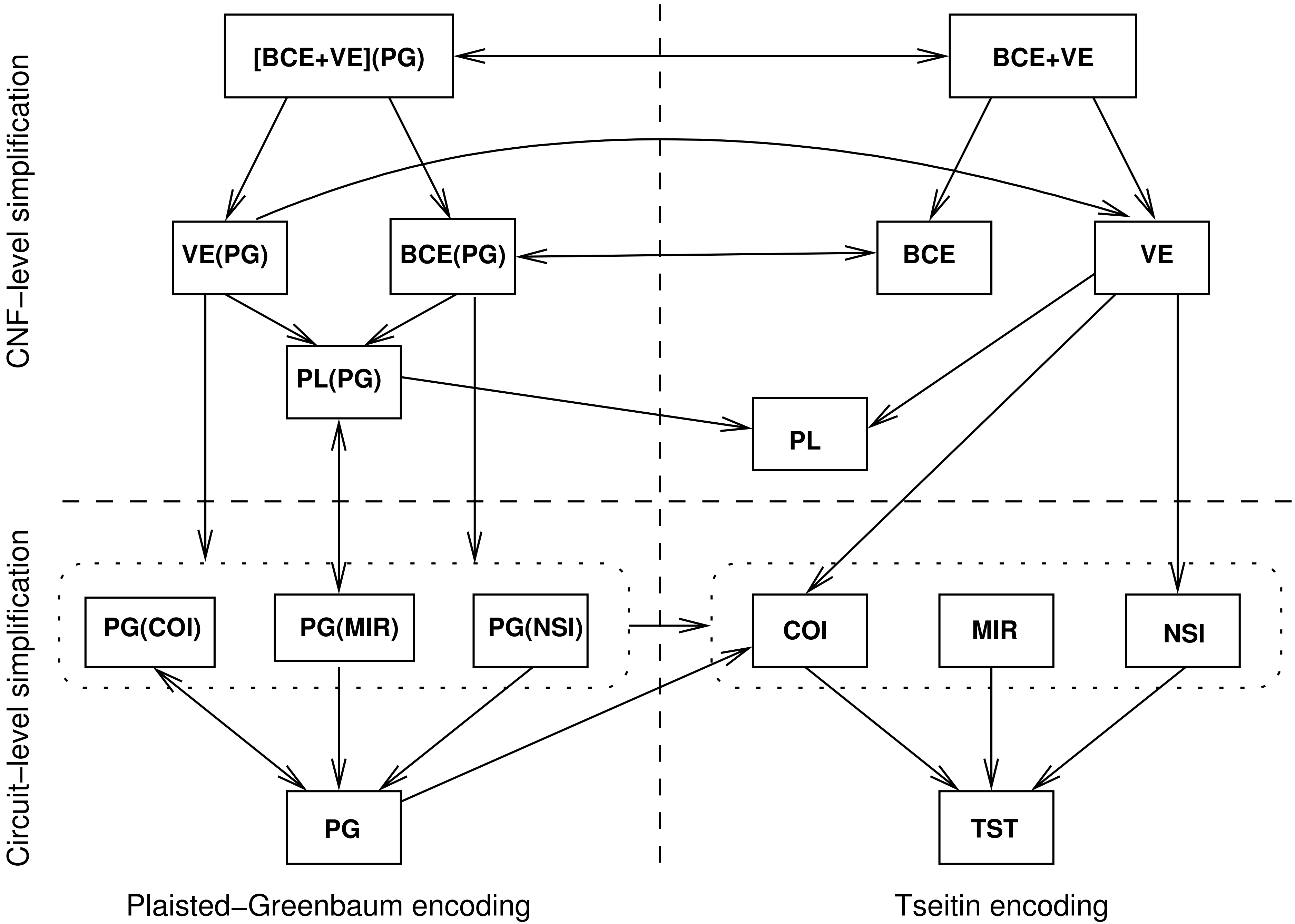
NSI Non-Shared Inputs reduction

PG Plaisted-Greenbaum polarity based encoding

TST standard Tseitin encoding

VE Variable-Elimination as in DP / Quantor / SATeLite

BCE Blocked-Clause-Elimination


CNF−level simplification
Circuit−level simplification
[BCE+VE](PG)
BCE+VE
VE(PG)
BCE(PG)
BCE
VE
PL(PG)
PL
PG(COI)
PG(MIR)
PG(NSI)
COI
MIR
NSI
PG
TST
Plaisted−Greenbaum encoding
Tseitin encoding

	encoding			b			be			beb			bebe			e		
	T	V	C	T	V	C	T	V	C	T	V	C	T	V	C	T	V	C
SU	0	46	256	2303	29	178	1042	11	145	1188	11	145	569	11	144	2064	11	153
AT	12	9	27	116	7	18	1735	1	8	1835	1	6	34	1	6	244	1	9
AP	10	9	20	94	7	18	1900	1	6	36	1	6	34	1	6	1912	1	6
AM	190	1	8	42	1	7	178	1	7	675	1	7	68	1	7	48	1	8
AN	9	3	10	50	3	10	1855	1	6	36	1	6	34	1	6	1859	1	6
HT	147	121	347	1648	117	277	2641	18	118	567	18	118	594	18	116	3240	23	140
HP	130	**121**	286	1398	117	277	2630	18	118	567	18	118	595	**18**	116	2835	19	119
HM	6961	**16**	91	473	16	84	621	12	78	374	12	77	403	**12**	76	553	15	90
HN	134	34	124	573	34	122	1185	17	102	504	17	101	525	17	100	1246	17	103
BT	577	442	1253	5799	420	1119	7023	57	321	1410	56	310	1505	52	294	8076	64	363
BP	542	**442**	1153	5461	420	1119	7041	57	321	1413	56	310	1506	**52**	294	7642	57	322
BM	10024	**59**	311	1252	58	303	1351	53	287	1135	53	286	1211	**52**	280	1435	55	303
BN	13148	196	643	2902	193	635	4845	108	508	2444	107	504	2250	105	500	5076	114	518

S	=	Sat competition	T	=	plain Tseitin encoding
A	=	AIG competition	P	=	Plaisted Greenbaum
H	=	HW model checking competition	M	=	MiniCirc encoding
B	=	bit-vector SMT competition	N	=	NiceDAGs

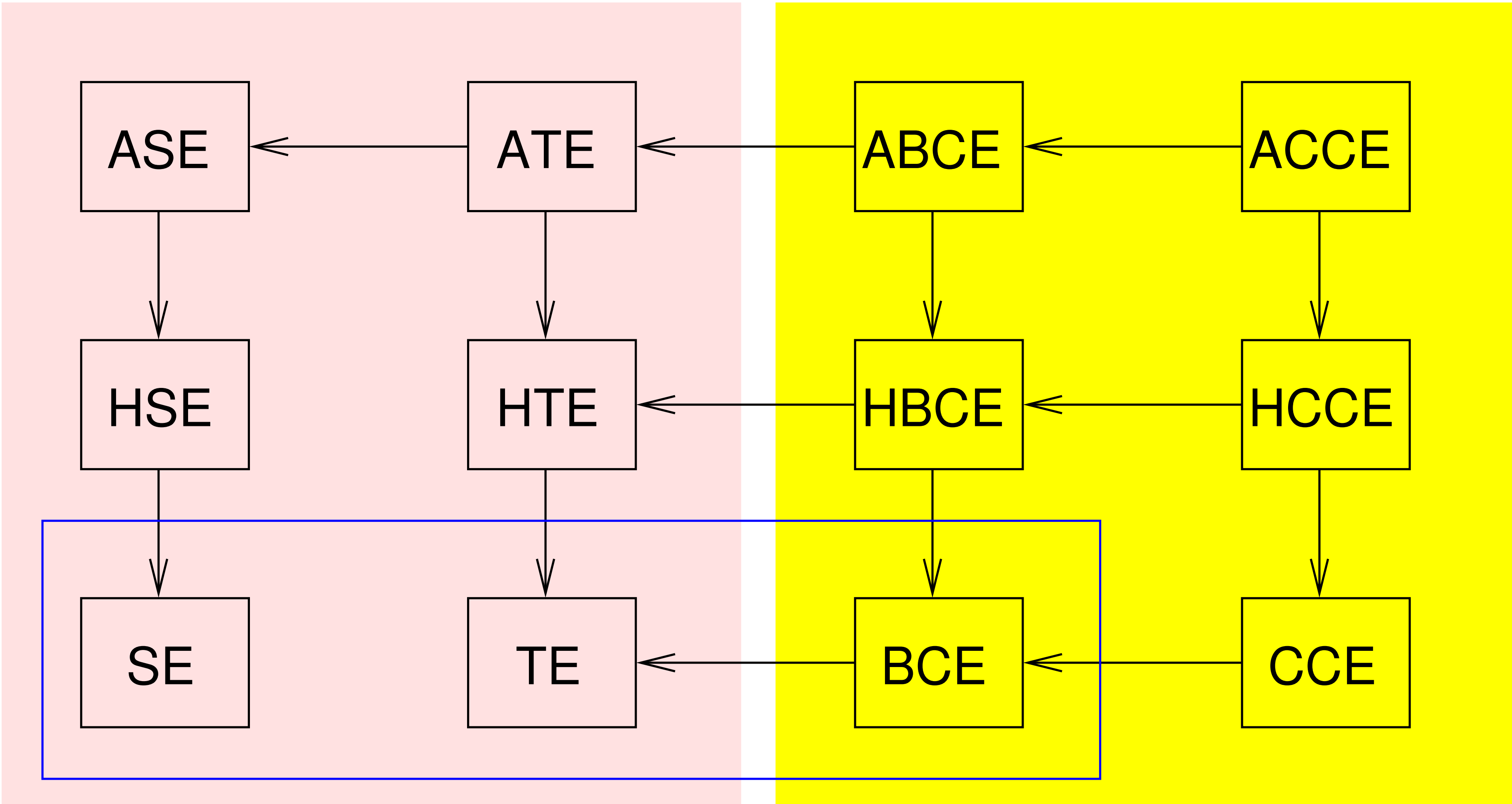
Further Clause Elimination Procedures

[HeuleJärvisaloBiere LPAR'10]

H = hidden, A = asymmetric,

SE = subsumption elimination, T = tautology elimination

BC = blocked clause elimination, CC = covered clause elimination



logically equivalent

satisfiability equivalent

Quantified Blocking Literal Given PCNF $\psi := Q_1 S_1 \ldots Q_n S_n.\ \phi$, a literal l in a clause $C \in \psi$ is a *quantified blocking literal* if for every clause C' with $\neg l \in C'$, $C \otimes C'$ is tautologous wrt. some variable k such that $k \leq l$ in prefix ordering.

Quantified Blocked Clause Given PCNF $Q_1 S_1 \ldots Q_n S_n.\ (\phi \wedge C)$. Clause C is *quantified blocked* if it contains a quantified blocking literal. Removing C preserves satisfiability.

$$Q_1 S_1 \ldots Q_n S_n.\ (\phi \wedge C) \overset{sat}{\equiv} Q_1 S_1 \ldots Q_n S_n.\ \phi.$$

All clauses blocked: $\forall x \exists y ((x \vee \neg y) \wedge (\neg x \vee y))$.

No clause blocked: $\exists x \forall y ((x \vee \neg y) \wedge (\neg x \vee y))$.

Implemented in our QBF preprocessor Bloqqer

Experimental Results for QBCE

[BiereLonsingSeidl-CADE11]

		# formulas (total 568)			run time (sec)	
	preprocessing	*solved*	*sat*	*unsat*	*avg*	*med*
DepQBF	sQueeze/Bloqqer	482 (+29%)	234	248	180	5
	Bloqqer	467 (+25%)	224	243	198	5
	Bloqqer/sQueeze	452 (+21%)	213	239	258	19
	sQueeze	435 (+16%)	201	234	231	6
	none	373	167	206	332	26
Qube	sQueeze/Bloqqer	454 (+36%)	207	247	227	7
	Bloqqer	444 (+33%)	200	244	246	5
	Bloqqer/sQueeze	421 (+26%)	183	238	307	27
	sQueeze	406 (+22%)	181	225	313	31
	none	332	135	197	426	258
Quantor	Bloqqer	288 (+39%)	145	143	468	34
	sQueeze/Bloqqer	285 (+38%)	147	138	472	39
	Bloqqer/sQueeze	270 (+31%)	131	139	486	34
	sQueeze	222 (+7%)	106	116	561	49
	none	206	100	106	587	38

QBFEVAL 2010 benchmark set, 568 formulae, 7 GB / 900 sec. limits

Current Status of BCE and Friends

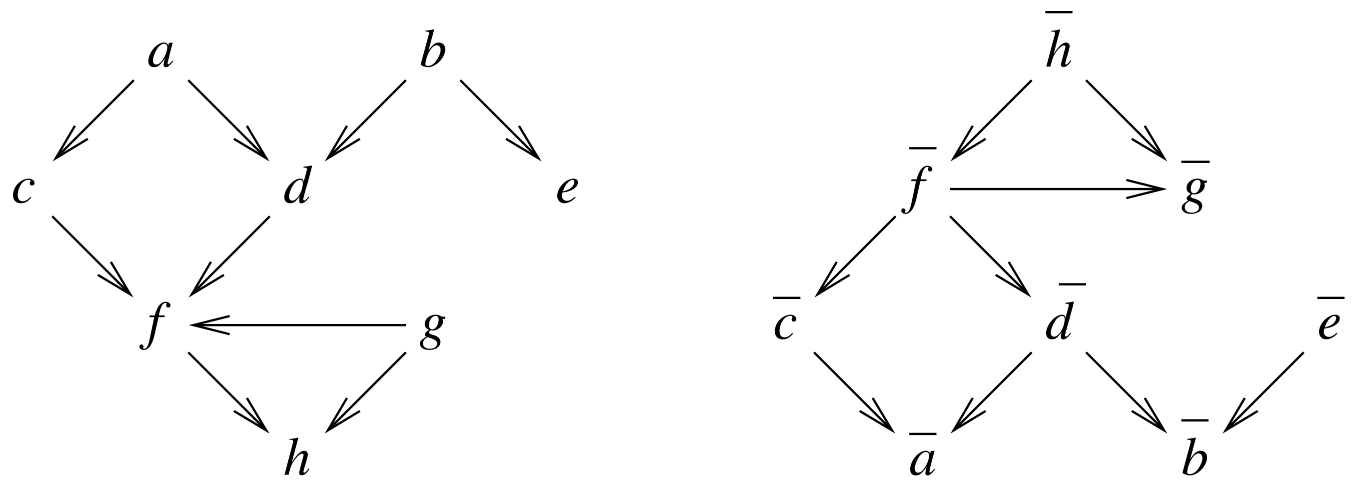
- there are instances which can be solved (only) cheaply with BCE
- most of the time only modest additional size reduction after VE
- BCE implementation very similar to implementation of VE
- as VE needs freeze/melt (freeze/thaw) interface
- we have an unpublished theory to treat redundant clauses as learned clauses …
- … and an unpublished solution reconstruction for CCE as well
- extended to QBF [BiereLonsingSeidl-CADE11]

Motivation for Unhiding

- SAT solvers applied to huge formulas
  - fastests solvers use preprocessing/inprocessing
  - need cheap and effective inprocessing techniques for millions of variables
- this talk:
  - **unhiding** redundancy in large formulas
  - almost linear randomized algorithm $O(|F|\log|C|)$
  - using the binary implication graph
  - fast enough to be applied to learned clauses
- see our SAT'11 paper for more details

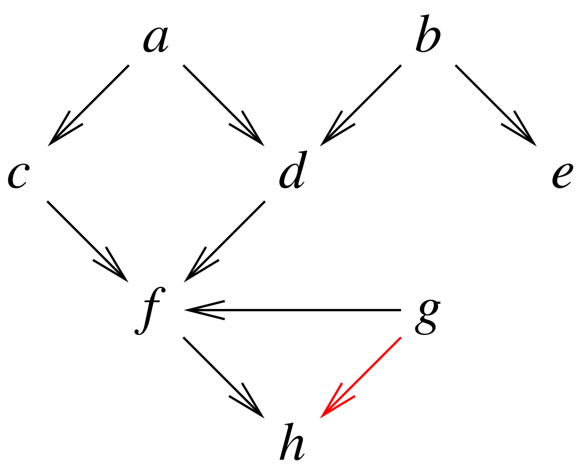
RiSE

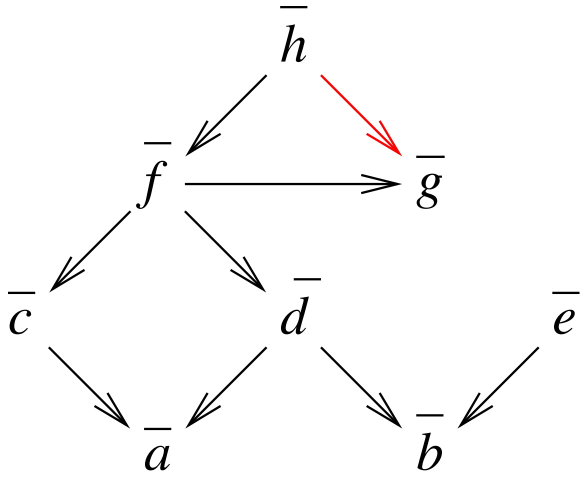
Binary Implication Graph (BIG)



$$
\begin{aligned}
&(\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge \\
&(\bar{c} \vee f) \wedge (\bar{d} \vee f) \wedge (\bar{g} \vee f) \wedge (\bar{f} \vee h) \wedge \\
&(\bar{g} \vee h) \wedge \underbrace{(\bar{a} \vee \bar{e} \vee h) \wedge (\bar{b} \vee \bar{c} \vee h) \wedge (a \vee b \vee c \vee d \vee e \vee f \vee g \vee h)}_{\text{non binary clauses}}
\end{aligned}
$$

Transitive Reduction (TRD)





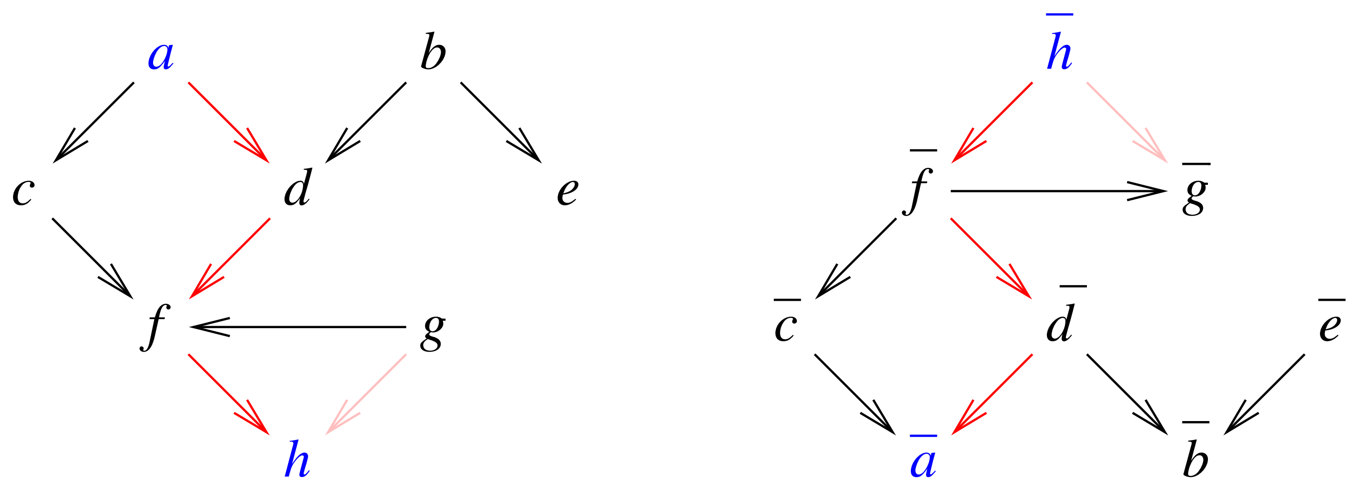
$$(\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge$$
$$(\bar{c} \vee f) \wedge (\bar{d} \vee f) \wedge (\bar{g} \vee f) \wedge (\bar{f} \vee h) \wedge$$
$$\cancel{(\bar{g} \vee h)} \wedge (\bar{a} \vee \bar{e} \vee h) \wedge (\bar{b} \vee \bar{c} \vee h) \wedge (a \vee b \vee c \vee d \vee e \vee f \vee g \vee h)$$

TRD

$g \rightarrow f \rightarrow h$

Hidden Tautology Elimination (HTE)

[HeuleJärvisaloBiere LPAR'2010]



$$(\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge$$

$$(\bar{c} \vee f) \wedge (\bar{d} \vee f) \wedge (\bar{g} \vee f) \wedge (\bar{f} \vee h) \wedge$$

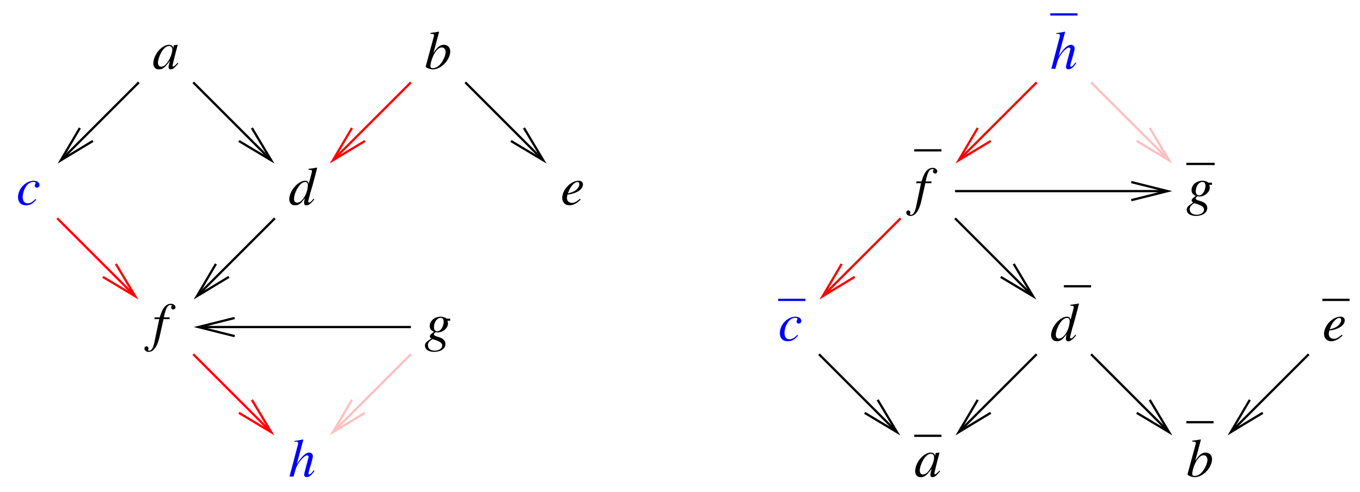
$$\cancel{(\bar{a} \vee \bar{e} \vee h)} \wedge (\bar{b} \vee \bar{c} \vee h) \wedge (a \vee b \vee c \vee d \vee e \vee f \vee g \vee h)$$

HTE

$a \to d \to f \to h$

Hidden Tautology Elimination (HTE)

[HeuleJärvisaloBiere LPAR'2010]



$$(\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge$$

$$(\bar{c} \vee f) \wedge (\bar{d} \vee f) \wedge (\bar{g} \vee f) \wedge (\bar{f} \vee h) \wedge$$

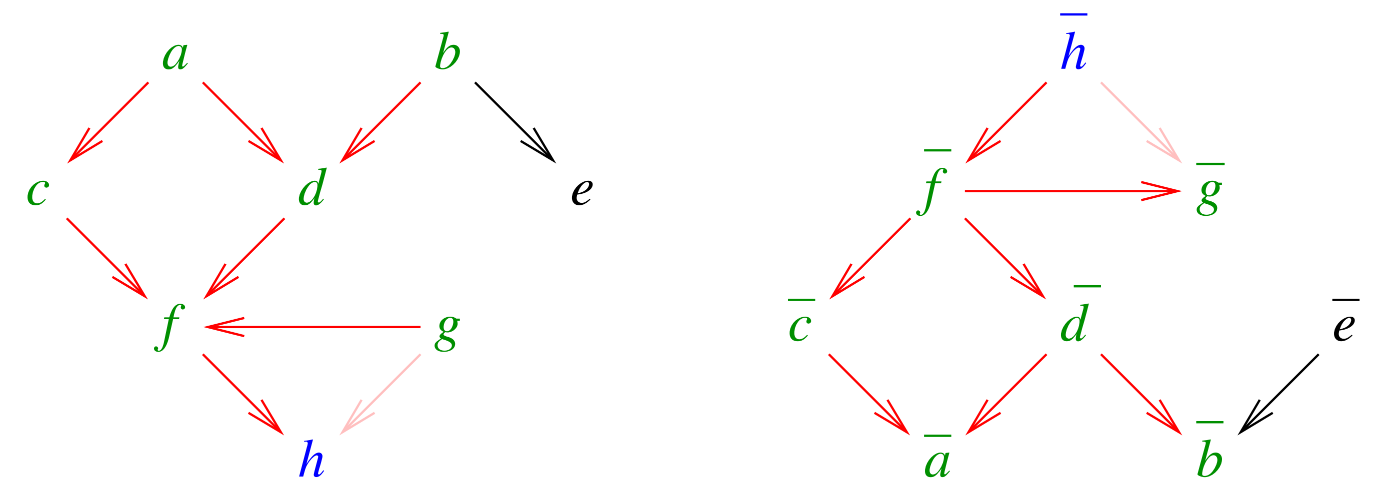
$$\cancel{(\bar{b} \vee \bar{c} \vee h)} \wedge (a \vee b \vee c \vee d \vee e \vee f \vee g \vee h)$$

HTE

$c \rightarrow f \rightarrow h$

Hidden Literal Elimination (HLE)

[HeuleJärvisaloBiere LPAR'2010]



$$(\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge$$

$$(\bar{c} \vee f) \wedge (\bar{d} \vee f) \wedge (\bar{g} \vee f) \wedge (\bar{f} \vee h) \wedge$$

$$(\cancel{a \vee b \vee c \vee d \vee}\, e \vee \cancel{f \vee g \vee}\, h)$$

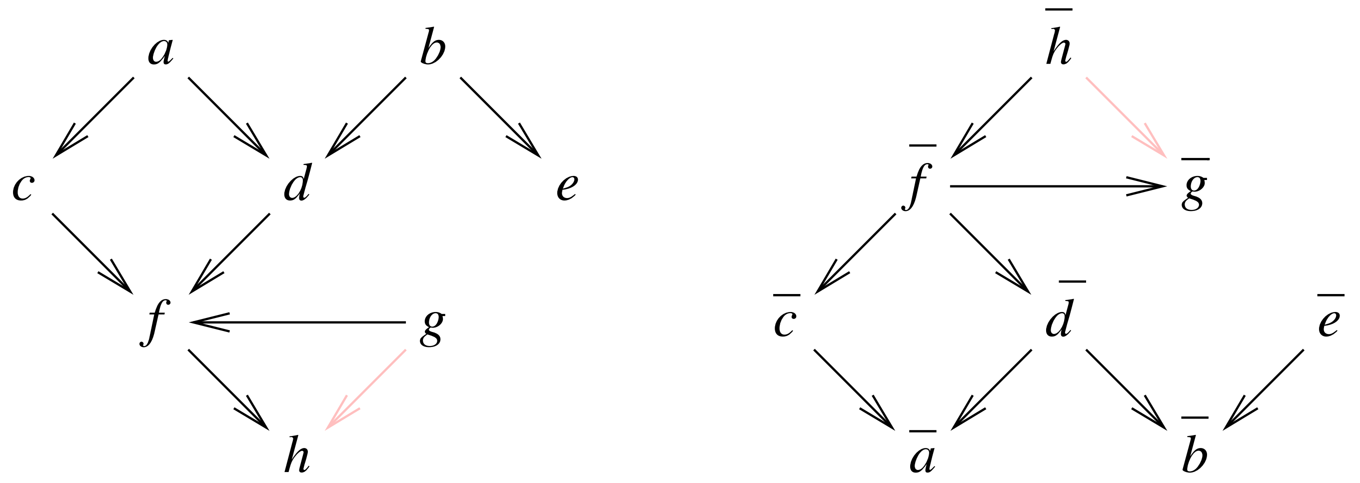
HLE

all but e imply h

also b implies e

Hidden Literal Elimination (HLE)

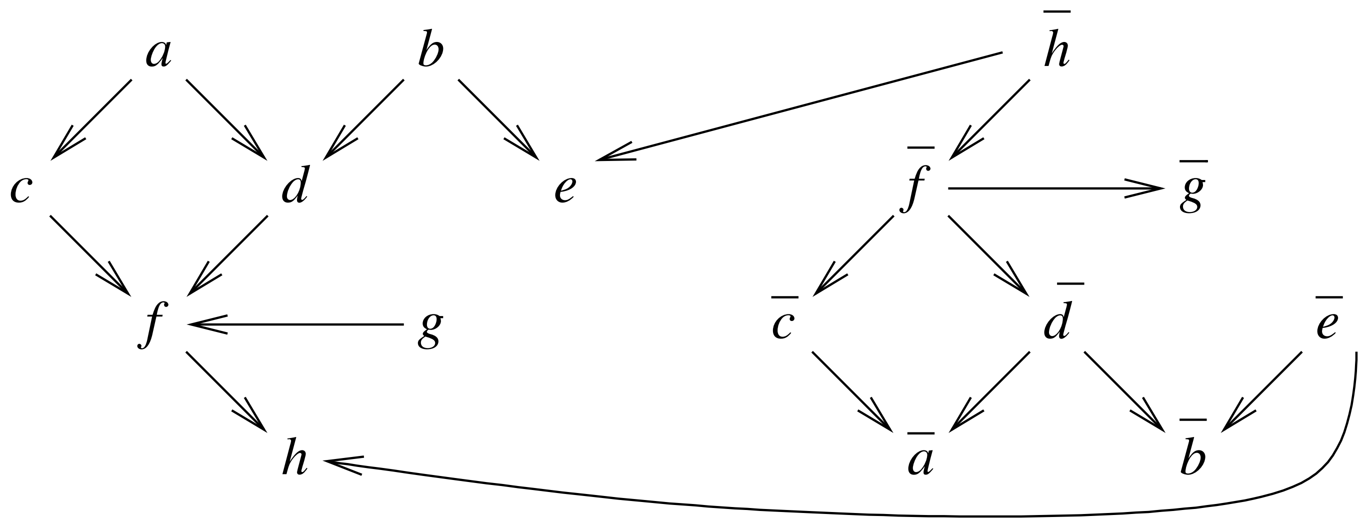
[HeuleJärvisaloBiere LPAR'2010]



$$(\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge$$
$$(\bar{c} \vee f) \wedge (\bar{d} \vee f) \wedge (\bar{g} \vee f) \wedge (\bar{f} \vee h) \wedge$$
$$(e \vee h)$$

TRD + HTE + HLE

[HeuleJärvisaloBiere LPAR'2010]

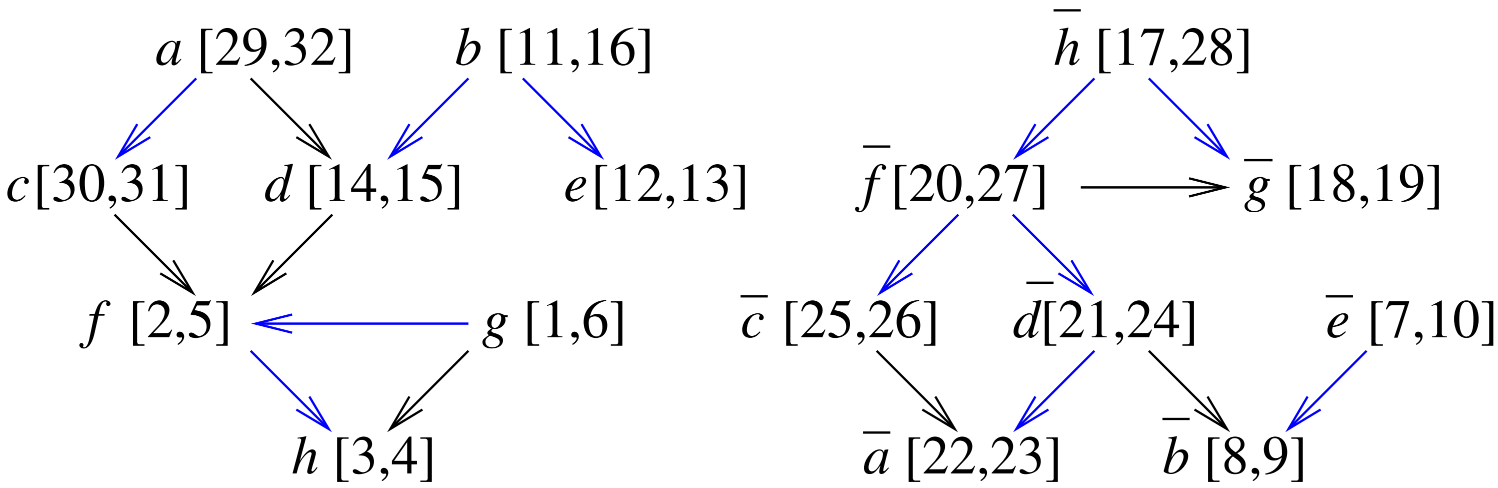


$$(\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge$$
$$(\bar{c} \vee f) \wedge (\bar{d} \vee f) \wedge (\bar{g} \vee f) \wedge (\bar{f} \vee h) \wedge$$
$$(e \vee h)$$

Time Stamping

[HeuleJärvisaloBiere SAT'11]

DFS tree with discovered and finished times: $[\mathrm{dsc}(l), \mathrm{fin}(l)]$



tree edges

parenthesis theorem: l ancestor in DFS tree of k iff $[\mathrm{dsc}(k), \mathrm{fin}(k)] \subseteq [\mathrm{dsc}(l), \mathrm{fin}(l)]$

well known

ancestor relationship gives necessary conditions for **transitive** implication:

if $[\mathrm{dsc}(k), \mathrm{fin}(k)] \subseteq [\mathrm{dsc}(l), \mathrm{fin}(l)]$ then $l \rightarrow k$

if $[\mathrm{dsc}(\bar{l}), \mathrm{fin}(\bar{l})] \subseteq [\mathrm{dsc}(\bar{k}), \mathrm{fin}(\bar{k})]$ then $l \rightarrow k$

- time stamping in previous example does not cover $b \rightarrow h$

  $$[11,16] = [\mathrm{dsc}(b), \mathrm{fin}(b)] \not\subseteq [\mathrm{dsc}(h), \mathrm{fin}(h)] = [3,4]$$
  $$[17,28] = [\mathrm{dsc}(\bar{h}), \mathrm{fin}(\bar{h})] \not\subseteq [\mathrm{dsc}(\bar{b}), \mathrm{fin}(\bar{b})] = [8,9]$$

  in example still both HTE "unhidden", HLE works too (since $b \rightarrow e$)

- "coverage" heavily depends on DFS order
  - as solution we propose multiple **randomized DFS** rounds/phases
  - approximate quadratic problem (BIG reachability) randomly by a linear algorithm

- if BIG is a tree *one* time stamping covers everything

Unhiding through Time Stamping

[HeuleJärvisaloBiere SAT'11]

Unhiding (formula F)

$stamp := 0$
foreach literal l in BIG(F) **do**
    $\text{dsc}(l) := 0$; $\text{fin}(l) := 0$
    $\text{prt}(l) := l$; $\text{root}(l) := l$
foreach $r \in \text{RTS}(F)$ **do**
    $stamp :=$ *Stamp*$(r, stamp)$
foreach literal l in BIG(F) **do**
    if $\text{dsc}(l) = 0$ **then**
        $stamp :=$ *Stamp*$(l, stamp)$
return *Simplify*(F)

Stamp (literal l, integer $stamp$)

$stamp := stamp + 1$
$\text{dsc}(l) := stamp$
foreach $(\bar{l} \vee l') \in F_2$ **do**
    if $\text{dsc}(l') = 0$ **then**
        $\text{prt}(l') := l$
        $\text{root}(l') := \text{root}(l)$
        $stamp :=$ *Stamp*$(l', stamp)$
$stamp := stamp + 1$
$\text{fin}(l) := stamp$
return $stamp$

Simplify (formula F)

foreach $C \in F$
    $F := F \setminus \{C\}$
    if *UHTE*(C) **then continue**
    $F := F \cup \{$*UHLE*$(C)\}$
return F

Unhiding HTE

[HeuleJärvisaloBiere SAT'11]

UHTE (clause C)

$l_{\text{pos}} :=$ first element in $S^{+}(C)$
$l_{\text{neg}} :=$ first element in $S^{-}(C)$
while *true*
  if $\operatorname{dsc}(l_{\text{neg}}) > \operatorname{dsc}(l_{\text{pos}})$ **then**
    if l_{pos} is last element in $S^{+}(C)$ **then return** false
    $l_{\text{pos}} :=$ next element in $S^{+}(C)$
  else if $\operatorname{fin}(l_{\text{neg}}) < \operatorname{fin}(l_{\text{pos}})$ **or** $(|C| = 2$ **and** $(l_{\text{pos}} = \bar{l}_{\text{neg}}$ **or** $\operatorname{prt}(l_{\text{pos}}) = l_{\text{neg}}))$ **then**
    if l_{neg} is last element in $S^{-}(C)$ **then return** false
    $l_{\text{neg}} :=$ next element in $S^{-}(C)$
  else return true

$S^{+}(C)$	sequence of literals in C ordered by $\operatorname{dsc}()$
$S^{-}(C)$	sequence of negations of literals in C ordered by $\operatorname{dsc}()$

$$O(|C| \log |C|)$$

[HeuleJärvisaloBiere SAT'11]

UHLE (clause C)

$\mathit{finished}$:= finish time of first element in $S^{+}_{\mathrm{rev}}(C)$
foreach $l \in S^{+}_{\mathrm{rev}}(C)$ starting at second element
  if $\mathrm{fin}(l) > \mathit{finished}$ **then** $C := C \setminus \{l\}$
  else $\mathit{finished} := \mathrm{fin}(l)$
$\mathit{finished}$:= finish time of first element in $S^{-}(C)$
foreach $\bar{l} \in S^{-}(C)$ starting at second element
  if $\mathrm{fin}(\bar{l}) < \mathit{finished}$ **then** $C := C \setminus \{l\}$
  else $\mathit{finished} := \mathrm{fin}(\bar{l})$
return C

$S^{+}_{\mathrm{rev}}(C)$ reverse of $S^{+}(C)$

$$O(|C| \log |C|)$$

Advanced Time Stamping

[HeuleJärvisaloBiere SAT'11]

Stamp (literal l, integer $stamp$)

BSC $stamp := stamp + 1$
BSC $\mathrm{dsc}(l) := stamp$; $\mathrm{obs}(l) := stamp$
ELS $flag :=$ true   // l represents a SCC
ELS S.push(l)   // push l on SCC stack
BSC   **for each** $(\bar{l} \vee l') \in F_2$
TRD   **if** $\mathrm{dsc}(l) < \mathrm{obs}(l')$ **then** $F := F \setminus \{(\bar{l} \vee l')\}$; **continue**
FLE   **if** $\mathrm{dsc}(\mathrm{root}(l)) \leq \mathrm{obs}(\bar{l}')$ **then**
FLE $l_{\mathrm{failed}} := l$
FLE   **while** $\mathrm{dsc}(l_{\mathrm{failed}}) > \mathrm{obs}(\bar{l}')$ **do** $l_{\mathrm{failed}} := \mathrm{prt}(l_{\mathrm{failed}})$
FLE $F := F \cup \{(\bar{l}_{\mathrm{failed}})\}$
FLE   **if** $\mathrm{dsc}(\bar{l}') \neq 0$ **and** $\mathrm{fin}(\bar{l}') = 0$ **then continue**
BSC   **if** $\mathrm{dsc}(l') = 0$ **then**
BSC $\mathrm{prt}(l') := l$
BSC $\mathrm{root}(l') := \mathrm{root}(l)$
BSC $stamp :=$ *Stamp*$(l', stamp)$
ELS   **if** $\mathrm{fin}(l') = 0$ **and** $\mathrm{dsc}(l') < \mathrm{dsc}(l)$ **then**
ELS $\mathrm{dsc}(l) := \mathrm{dsc}(l')$; $flag :=$ false   // l is equivalent to l'
OBS $\mathrm{obs}(l') := stamp$ // set last observed time attribute
ELS   **if** $flag =$ true **then**   // if l represents a SCC
BSC $stamp := stamp + 1$
ELS   **do**
ELS $l' := S$.pop()   // get equivalent literal
ELS $\mathrm{dsc}(l') := \mathrm{dsc}(l)$ // assign equal discovered time
BSC $\mathrm{fin}(l') := stamp$ // assign equal finished time
ELS   **while** $l' \neq l$
BSC   **return** $stamp$

RiSE

Implementation

[HeuleJärvisaloBiere SAT'11]

- implemented as one inprocessing phase in our SAT solver Lingeling
  beside variable elimination, distillation, blocked clause elimination, probing, ...

- bursts of randomized DFS rounds and sweeping over the whole formula

- fast enough to be applicable to large learned clauses as well
  unhiding is particullary effective for learned clauses

- beside UHTE and UHLE we also have added hyper binary resolution UHBR
  not useful in practice

Lingeling 571 on SAT'09 Competition Application Benchmarks

[HeuleJärvisaloBiere SAT'11]

configuration	solved	sat	uns	unhd	simp	elim
adv.stamp (no uhbr)	188	78	110	7.1%	33.0%	16.1%
adv.stamp (w/uhbr)	184	75	109	7.6%	32.8%	15.8%
basic stamp (no uhbr)	183	73	110	6.8%	32.3%	15.8%
basic stamp (w/uhbr)	183	73	110	7.4%	32.8%	15.8%
no unhiding	180	74	106	0.0%	28.6%	17.6%

configuration	hte	stamp	redundant	hle	redundant	units	stamp
adv.stamp (no uhbr)	22	64%	59%	291	77.6%	935	57%
adv.stamp (w/uhbr)	26	67%	70%	278	77.9%	941	58%
basic stamp (no uhbr)	6	0%	52%	296	78.0%	273	0%
basic stamp (w/uhbr)	7	0%	66%	288	76.7%	308	0%
no unhiding	0	0%	0%	0	0.0%	0	0%

similar results for crafted and SAT'10 Race instances

- **search**: conflict driven clause learning (CDCL)
  - steady progress in capacity
  - how and when to restart is active research area
- **preprocessing** / **inprocessing** gives considerable reduction
  - new preprocessing algorithms
  - even quadratic algorithms are typically too expensive
- **parallel** SAT solving
  - port-folio versus splitting — see our ”cube & conquer” paper at HVC’11
  - SIMD algorithms — for many cores incl. GPUs
  - parallel preprocessing / inprocessing

RiSE