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

WorKer'11

3rd Workshop on Kernelization

TU Vienna, Austria

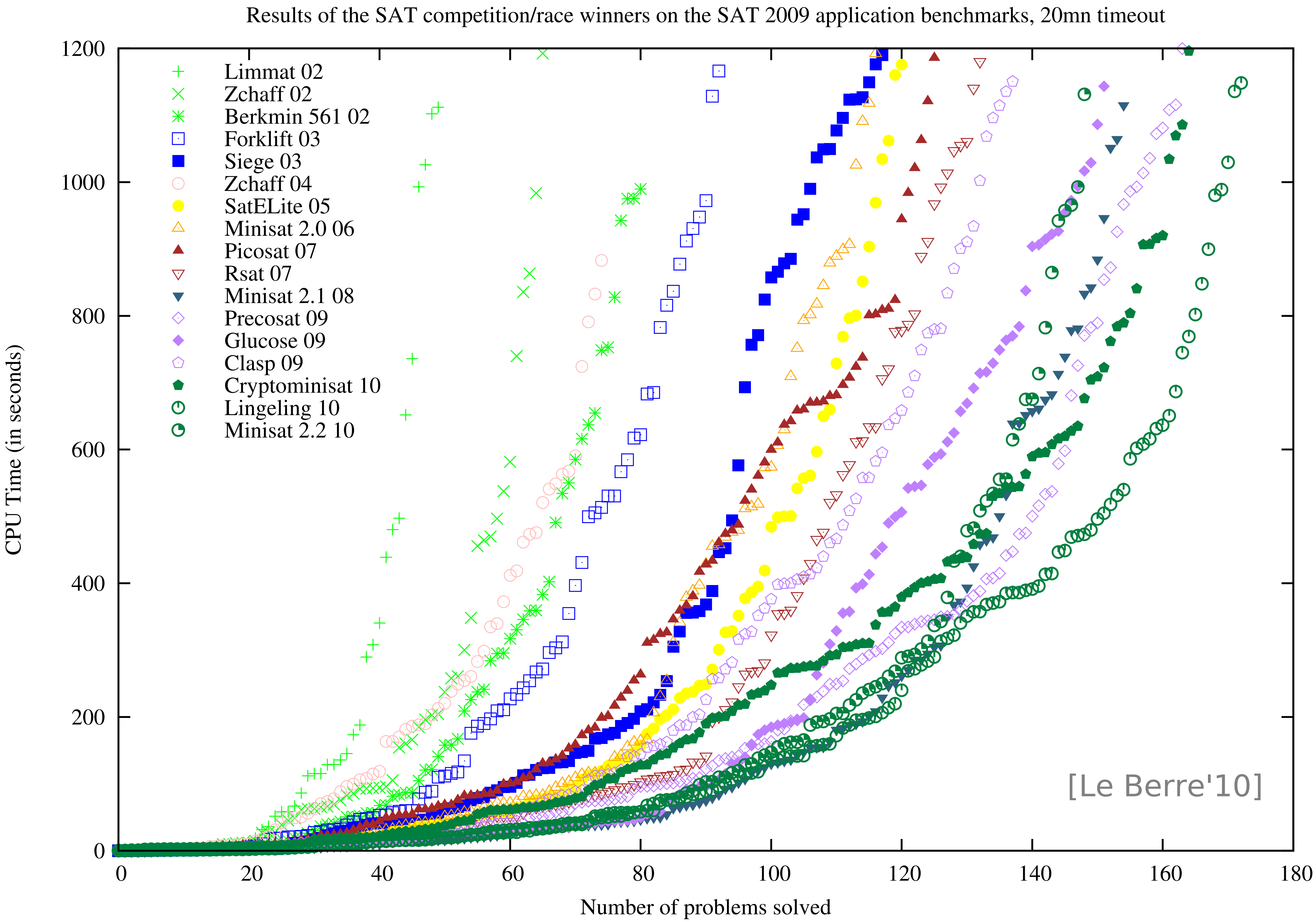
Friday, September 2, 2011

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 $\neg(\textbf{tie} \wedge \textbf{shirt}) \quad \equiv \quad \neg\textbf{tie} \vee \neg\textbf{shirt}$

- 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
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
[Le Berre'10]

Overview

- failed literal *probing*
- variable elimination (VE)
- inprocessing
- lazy hyper binary resolution
- blocked clause elimination (BCE)
- 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)

Additional Techniques related to VE

- 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 (later again)

- 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

Inprocessing: Interleaving Preprocessing and Search

- **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

Benefits of Interleaving Preprocessing and Search

- 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 units / equivalences learned in search 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 $(a \vee \overline{b}), (\overline{a} \vee b) \in f$ then replace a by b in f

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

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

- repeated probing, c.f. $\mathrm{HypBinResFast}$ [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
  - at least with respect to units it is possible to focus on roots
- current algorithms too expensive to run until completion

Observations

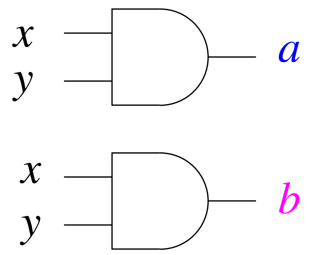
- **time** complexity: seems to be at least quadratic, unfortunately also in practice

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

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

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

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



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

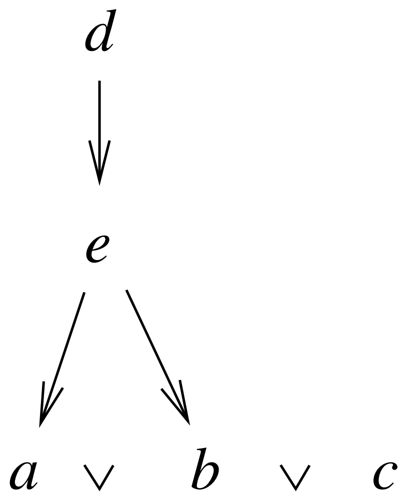
- can not simulate structural hashing of XOR or ITE gates

Lazy Hyper Binary Resolution (LHBR)

- learn binary clauses lazily or on-the-fly
  - in the innermost (!) BCP loop
  - during BCP in search or during BCP in preprocessing (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 $(\bar{e} \vee c)$



Avoid Transitive Edges by Prioritizing Binary Clause Propagation

1. trail contains assigned literals

2. set n_2 and n_3 to the trail level of those literals that still need to be propagated

3. while $0 \leq n_3 \leq n_2 < |\text{trail}|$ and there is no conflict

   (a) if $n_2 < |\text{trail}|$

      i. pick literal l at position n_2, increment n_2 and visit binary clauses with $\bar{l}$

      ii. assign literals forced through these binary clauses first

   (b) otherwise (necessarily $n_3 < |\text{trail}|$)

      i. pick literal l at position n_3, increment n_3 and visit large clauses with $\bar{l}$

      ii. assign literals forced through these large clauses

How-To Check Existence of Dominators

- for each assigned literal l keep **one** dominator $\quad \mathsf{bindom}(l)$

  (in the implication graph restricted to binary clauses)

- thus $\mathsf{bindom}(l)$ is the root of binary implication tree of l

- decisions l set $\quad \mathsf{bindom}(l) = l$

- binary implications $(a_1 \vee c)$ with $\sigma(a_1) = 0$, $\sigma(c) = 1$ set $\quad \mathsf{bindom}(c) = \mathsf{bindom}(\overline{a_1})$

- necessary & sufficient for $\overline{a_i}$ in large $(m \geq 2)$ reasons to have common dominator:

$$(a_1 \vee \cdots \vee a_m \vee c) \qquad\qquad \mathsf{bindom}(\overline{a_1}) = \cdots = \mathsf{bindom}(\overline{a_m})$$

- if condition triggers, actually use least common ancestor (closest dominator)

- use $(\overline{d} \vee c)$ as **new reason** instead of $(a_1 \vee \cdots \vee a_m \vee c)$

PrecoSAT Integration

- interleave search and preprocessing
  - bound time spent in search to roughly 80%
  - measured in number of propagations / resolutions
- during preprocessing / simplification on the top level
  - unit propagation on the top-level does LHBR top-level (1)
  - failed literal probing learns most binary clauses with LHBR probing (2)
- BCP during search learns binary clauses with LHBR search (3)

Experiments

- rerunning SAT'09 competition with competition version 236 of PrecoSAT
  - 900 seconds time out
  - roughly twice as fast machines
- PrecoSAT without LHBR solves 6 less instances
  - 171 instead of 177 out of 292
- statistics
  - LHBR learned 48 million binary clause
  - on 292 instances that is 181 thousand learned binary clauses on average
  - additionally 202 million learned clauses through conflict analysis
  - 19% learned (binary) clauses due to LHBR

Summary of LHBR

- no measurable overhead doing LHBR during BCP
  - so at least not harmful — in contrast to many other “optimizations”
  - implementation in Lingeling became non-trivial
  - unfortunately does not simulate structural hashing in practice (!!)
- *not formally published* but implemented in PrecoSAT and Lingeling
  - source code of PrecoSAT available under MIT license
  - source code of Lingeling available under GPL license
  - extensions in [HanSomenzi-DATE’11]
- performs a limited version of on-the-fly strengthening / subsumption

thus partially simulates distillation / vivification

Blocked Clause Removal

blocked clause $C \in F$

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

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

fix a CNF F

$a \vee b \vee l$

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

since all resolvents of C on l are tautological C can be removed

Proof

assignment σ satisfying $F \backslash C$ but not C

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

Blocked Clauses [Kullmann'99]

Definition 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.

Definition [Blocked Clause] A clause is blocked if has a literal that blocks it.

Definition [Blocked Literal] A literal is blocked if it blocks a clause.

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

only first clause is not blocked.

second clause contains two blocked literals: a and $\bar{c}$.

literal c in the last clause is blocked.

after removing either $(a \vee \bar{b} \vee \bar{c})$ or $(\bar{a} \vee c)$, the clause $(a \vee b)$ becomes blocked
actually all clauses can be removed

Relating Blocked Clauses and Encoding / Preprocessing Techniques

[JärvisaloBiereHeule-TACAS10]

COI Cone-of-Influence reduction

MIR Monontone-Input-Reduction

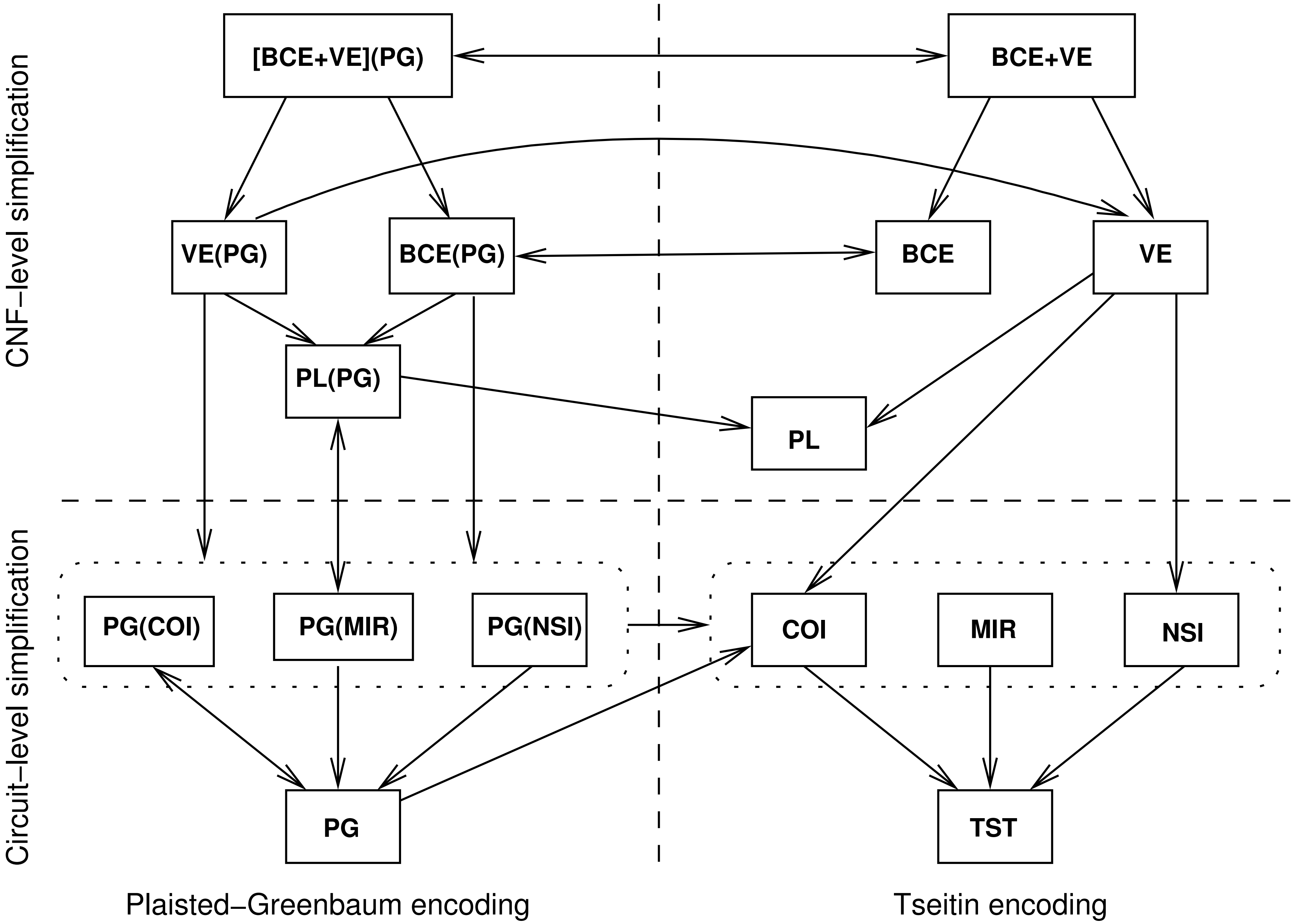
NSI Non-Shared Inputs reduction

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

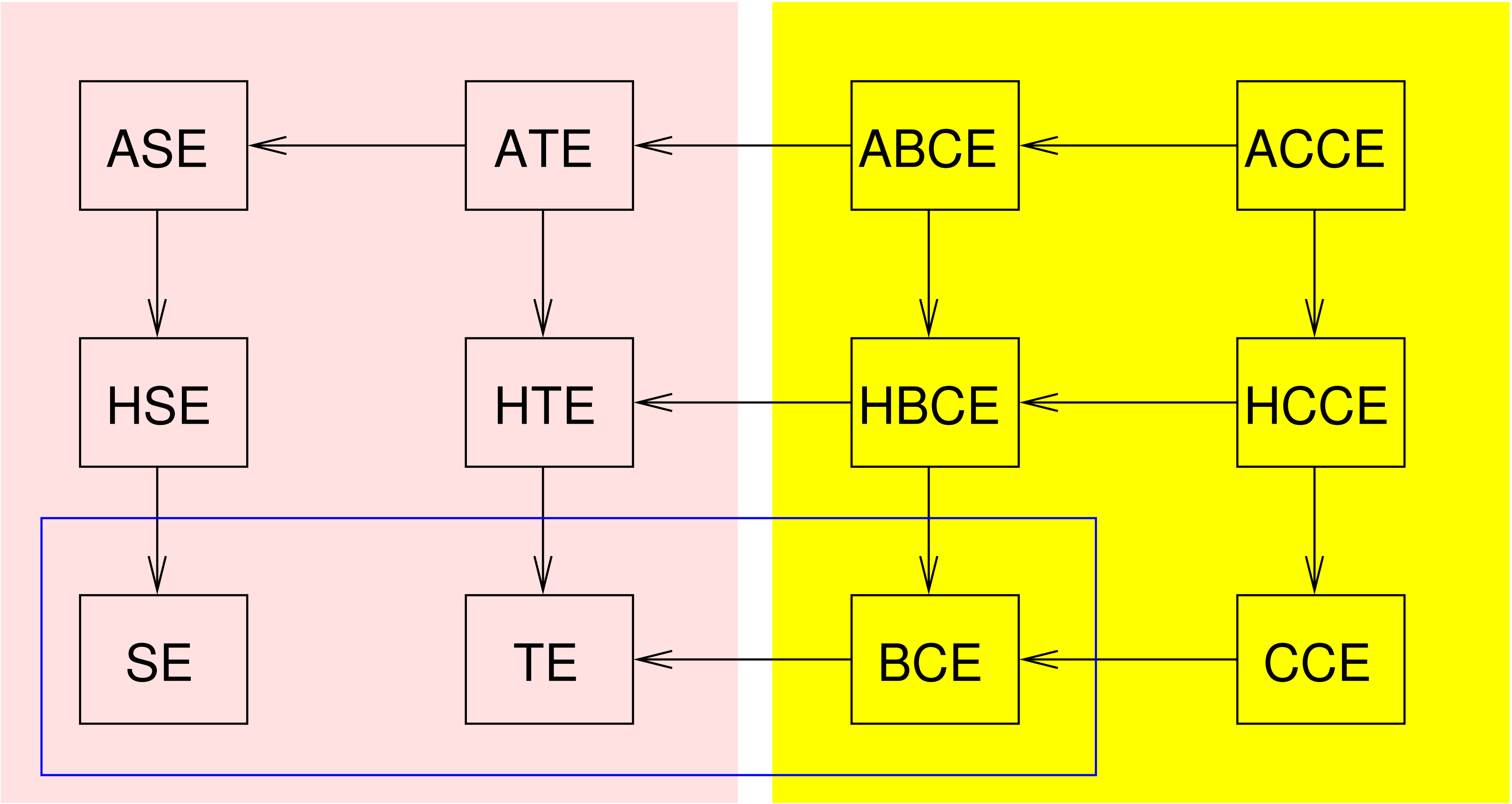
S	=	Sat competition	T	=	plain Tseitin encoding
A	=	AIG competition	P	=	Plaisted Greenbaum
H	=	HW model checking competition	M	=	MiniCirc encoding
B	=	bit-vector SMT competition	N	=	NiceDAGs

Further Clause Elimination Procedures

H = hidden, A = asymmetric,

SE = subsumption elimination, T = tautology elimination

BC = blocked clause elimination, CC = covered clause elimination



logically equivalent

satisfiability equivalent

Self-Subsuming Resolution and Asymmetric Literal Addition

Definition [self-subsuming resolution] [EénBiere SAT'05]

if $A \vee x$ and $B \vee \bar{x}$ are two clauses in a CNF and $\frac{A \vee x \quad \bar{x} \vee B}{A}$ a valid resolution, e.g. $B \subseteq A$, then replace the clause $A \vee x$ by A, in essence **removing** x from $A \vee x$

example: if both $a \vee b \vee x$ and $b \vee \bar{x}$ are in a CNF remove x from first clause

Definition [asymmetric literal addition] [HeuleJärvisaloBiere LPAR'10]

if A and $B \vee \bar{x}$ are two clauses in a CNF and $\frac{A \vee x \quad \bar{x} \vee B}{A}$ a valid resolution, e.g. $B \subseteq A$, then replace the clause A by $A \vee x$, in essence **adding** x to A

example: if both $a \vee b$ and $b \vee \bar{x}$ are in a CNF add x to first clause

Asymmetric and Hidden Tautologies and Blocked Clauses

Definition [asymmetric tautology / blocked clause] [HeuleJärvisaloBiere LPAR'10]

apply asymmetric literal addition to a clause w.r.t. to fixed CNF as long as possible, if the result is a tautological / blocked then remove clause (otherwise keep original)

Definition hidden = only use binary side clauses $\bar{x} \vee B$

Fact AHTE can be simulated by asymmetric branching / distillation and BCP

Fact HTE can be implemented much faster by iterating over all literals instead of iterating over all clauses (partially implemented in Lingeling)

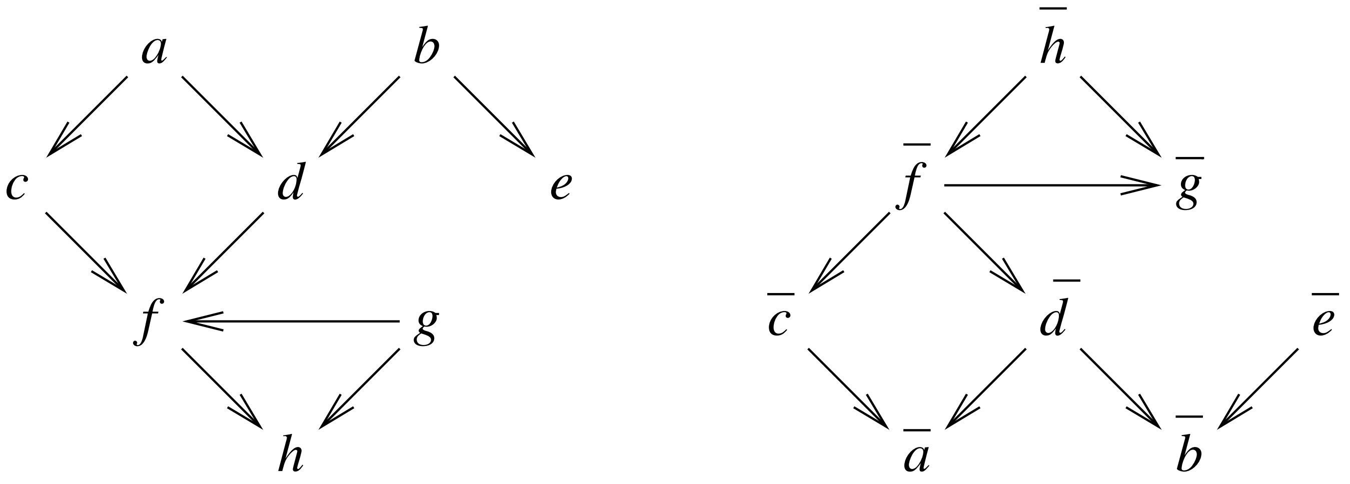
Fact HTE after equivalence reasoning and failed literal probing until completion on binary clauses only is confluent and BCP preserving

see our long and short LPAR'10 papers for more details

Motivation for Unhiding

- SAT solvers applied to huge formulas
  - million of variables
  - 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
  - using the binary implication graph
  - fast enough to be applied to learned clauses
- see our SAT'11 paper for more details

Binary Implication Graph BIG

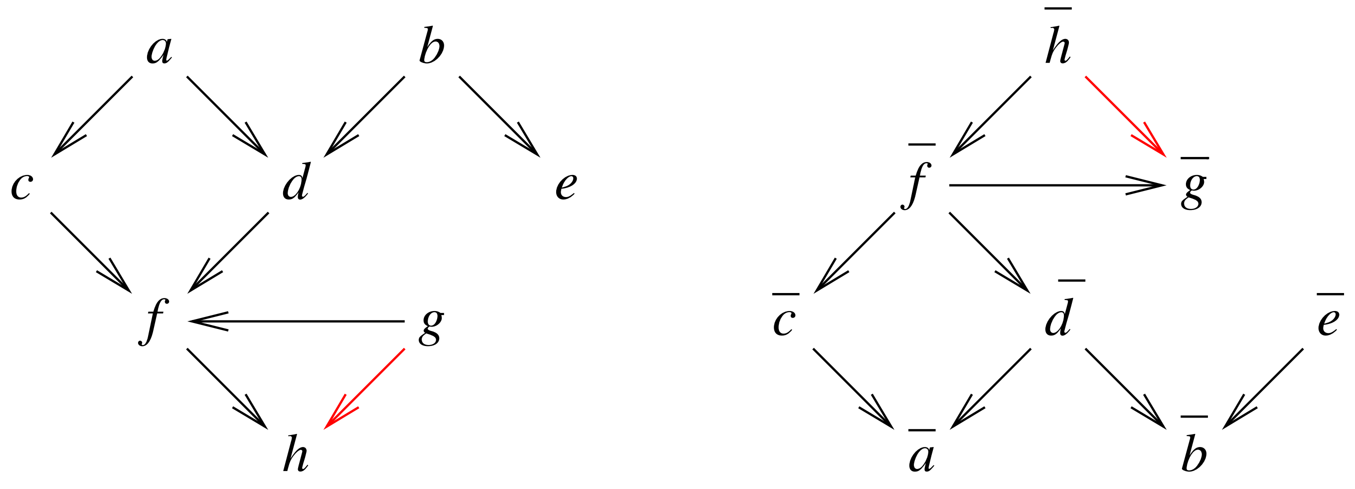


$$(\bar a \vee c) \wedge (\bar a \vee d) \wedge (\bar b \vee d) \wedge (\bar b \vee e) \wedge$$

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

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

Transitive Reduction TRD



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

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

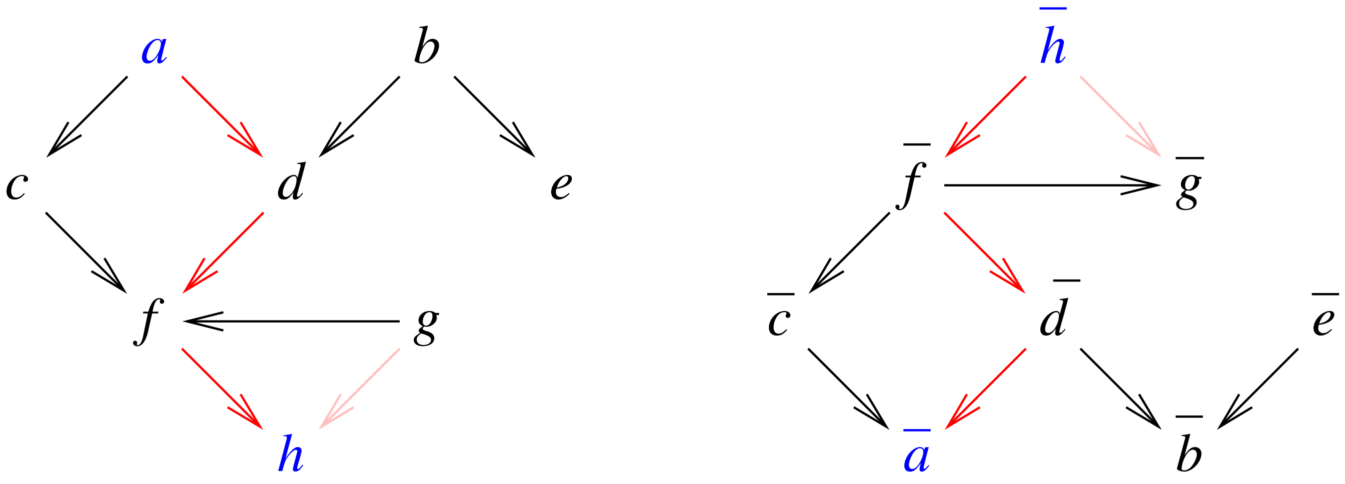
~~$(\bar g \vee h)$~~ $\wedge (\bar a \vee \bar e \vee h) \wedge (\bar b \vee \bar c \vee h) \wedge (a \vee b \vee c \vee d \vee e \vee f \vee g \vee h)$

TRD

$g \rightarrow f \rightarrow h$

Hidden Tautology Elimination HTE

[HeuleJärvisaloBiere LPAR'2010]



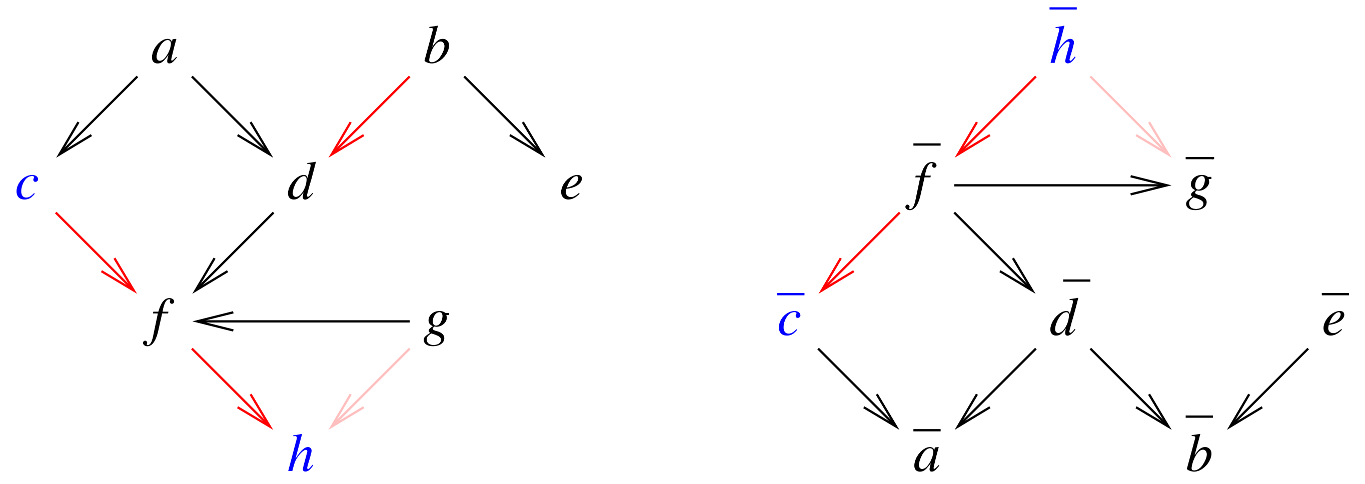
$$(\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge$$
$$(\bar{c} \vee f) \wedge (\bar{d} \vee f) \wedge (\bar{g} \vee f) \wedge (\bar{f} \vee h) \wedge$$
$$\cancel{(\bar{a} \vee \bar{e} \vee h)} \wedge (\bar{b} \vee \bar{c} \vee h) \wedge (a \vee b \vee c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

HTE

$a \rightarrow d \rightarrow f \rightarrow h$

Hidden Tautology Elimination HTE

[HeuleJärvisaloBiere LPAR'2010]



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

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

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

HTE

$c \rightarrow f \rightarrow h$

Self-Subsuming Resolution SSR

[EénBiere-SAT'05]

$$\frac{C \vee l \qquad D \vee \bar{l}}{D} \quad C \subseteq D \qquad\qquad \frac{a \vee b \vee l \qquad a \vee b \vee c \vee \bar{l}}{a \vee b \vee c}$$

resolvent D subsumes second antecedent $D \vee \bar{l}$

assume given CNF contains both antecedents $\ldots(a \vee b \vee l)(a \vee b \vee c \vee \bar{l})\cdots$

if D is added to CNF then $D \vee \bar{l}$ can be removed $\Downarrow$

which in essence *removes* $\bar{l}$ from $D \vee \bar{l}$ $\ldots(a \vee b \vee l)(a \vee b \vee c \quad)\ldots$

used in SATeLite preprocessor

now common in many SAT solvers

HTE = Reverse SSR + Tautology Elimination

[HeuleJärvisaloBiere LPAR'2010]

hidden literal addition (HLA) uses SSR in reverse order

$$\frac{C \vee l \qquad D \vee \bar{l}}{D} \quad C \subseteq D \qquad\qquad \frac{a \vee b \vee l \qquad a \vee b \vee c \vee \bar{l}}{a \vee b \vee c}$$

assume given CNF contains resolvent and first antecedent $\ldots(a \vee b \vee l)(a \vee b \vee c)\cdots$

we can replace D by $D \vee \bar{l}$ $\ldots(a \vee b \vee l)(a \vee b \vee c \vee \bar{l})\ldots$

which in essence *adds* $\bar{l}$ to D, repeat HLA until fix-point

keep remaining non-tautological clauses *after removing added literals again*

HTE = assume $C \vee l$ is a binary clauses more general versions in the paper

remove clauses with a literal implied by negation of another literal in the clause

HTE confluent and BCP preserving modulo equivalent variable renaming

Hidden Literal Elimination HLE

[HeuleJärvisaloBiere SAT'11]

better explained on binary implication graph

remove literal from a clause which implies another literal in the clause

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

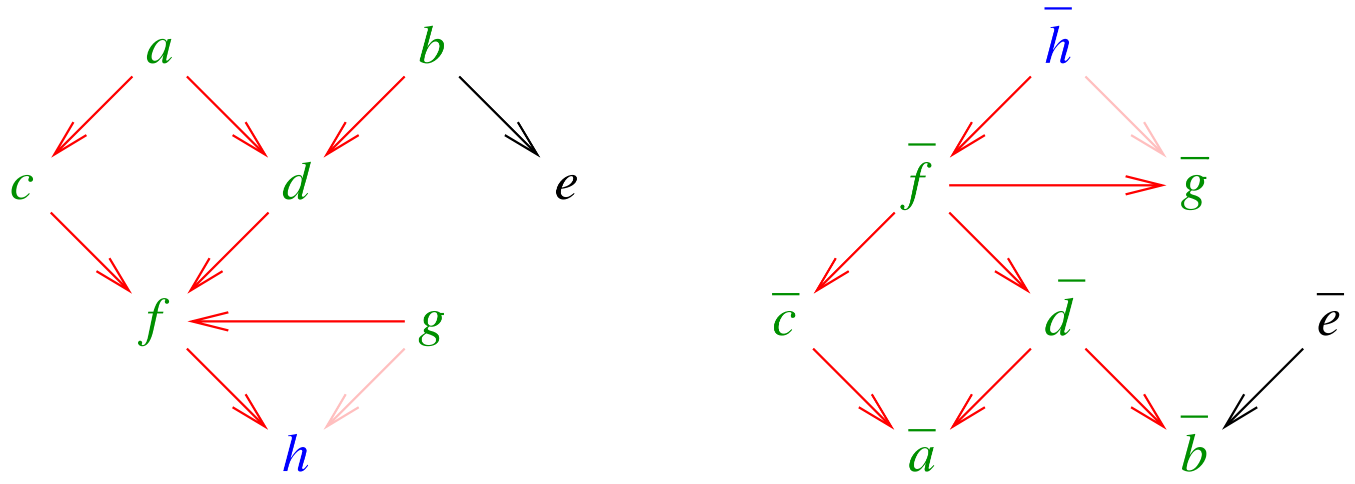
related work before all uses BCP:

- asymmetric branching — implemented in MiniSAT but switched off by default
- **distillation** — [JinSomenzi'05][HanSomenzi DAC'07]
- vivification — [PietteHamadiSais ECAI'08]
- caching technique in CryptoMiniSAT

HTE/HLE only uses the binary implication graph!

Hidden Literal Elimination HLE

[HeuleJärvisaloBiere SAT’11]



$$(\bar{a} \lor c) \land (\bar{a} \lor d) \land (\bar{b} \lor d) \land (\bar{b} \lor e) \land$$

$$(\bar{c} \lor f) \land (\bar{d} \lor f) \land (\bar{g} \lor f) \land (\bar{f} \lor h) \land$$

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

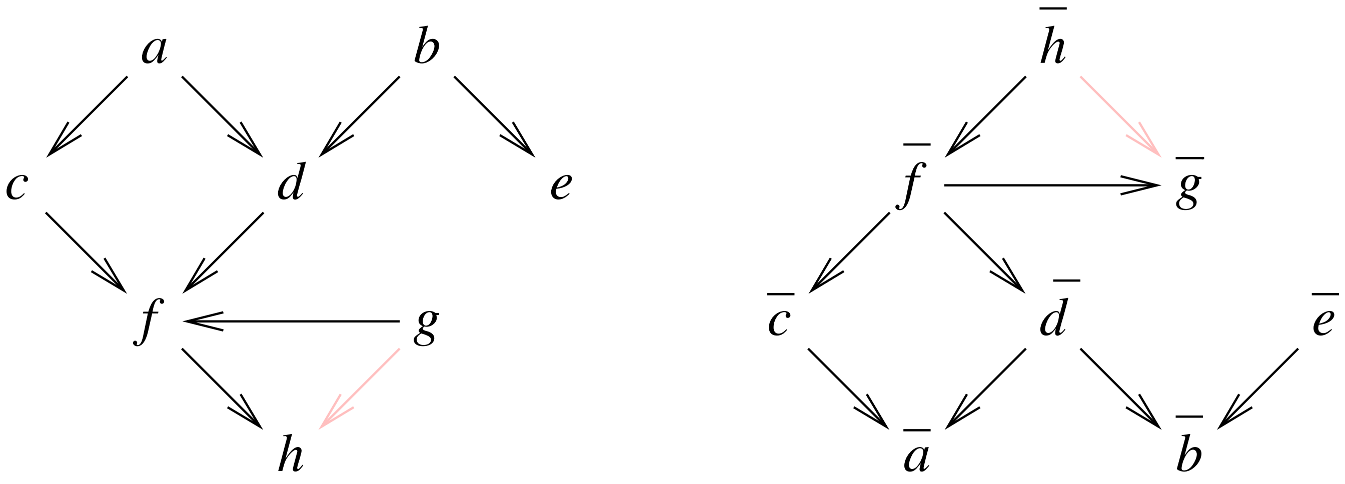
HLE

all but e imply h

also b implies e

Hidden Literal Elimination HLE

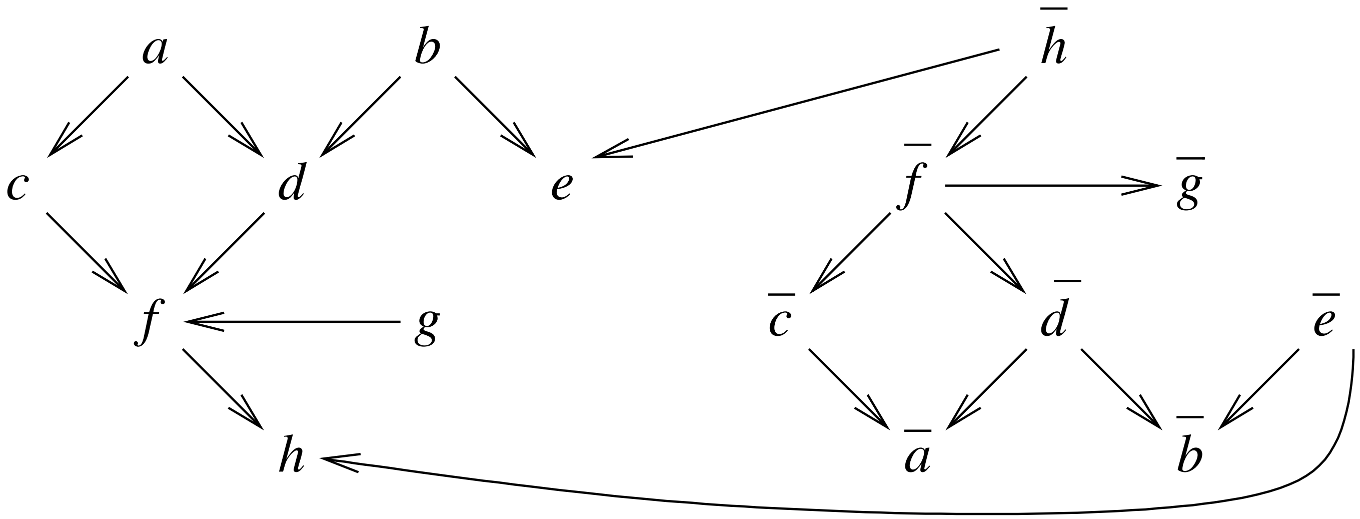
[HeuleJärvisaloBiere SAT'11]



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

TRD + HTE + HLE

[HeuleJärvisaloBiere SAT'11]



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

Failed Literal Elimination FL

actually quite old technique ... [Freeman PhdThesis'95] [LeBerre'01] ...

$\boxed{\textit{assume} \text{ literal } l\text{, BCP, if conflict, add unit } \bar{l}}$

rather costly to run until completion conjecture: at least quadratic

one BCP is linear and also in practice can be quite expensive

need to do it for all variables and restart if new binary clause generated

useful in practice: lift common implied literals for assumption l and assumption $\bar{l}$

even on BIG (FL2) conjectured to be quadratic [VanGelder'05]

$\ldots(\bar{a} \vee b)(\bar{b} \vee c)(\bar{c} \vee d)(\bar{d} \vee \bar{a})\ldots \quad \Rightarrow$ add unit clause $\bar{a}$

subsumed by running one HLA until completion

Equivalent Literal Substitution ELS

[AspvallPlassTarjan'79] [Li'00] [Val'01] [Brafman'04] [VanGelder'05]

decompose BIG into strongly connect components (SCCs)

if there is an l with l and $\bar{l}$ in the same component ⇒ *unsatisfiable*

otherwise replace all literals by a "representative"

linear algorithm can be applied routinely during garbage collection

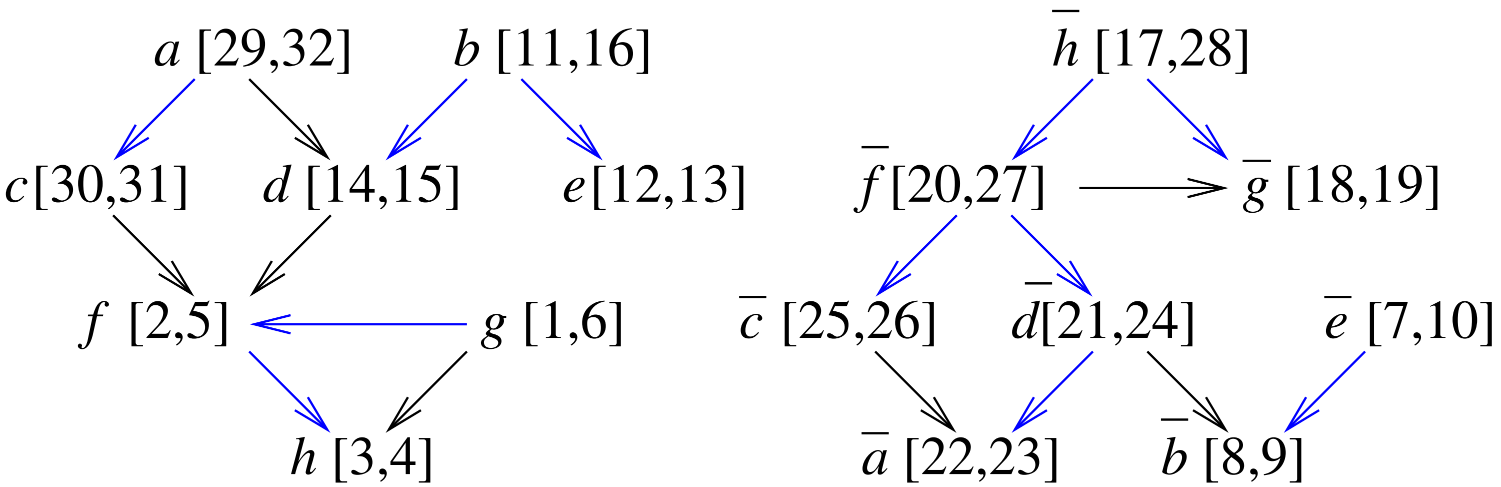
but as with failed literal preprocessing may generate new binary clauses

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

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 $l \rightarrow k$

Unhiding: Applying Time Stamping to TRD/HTE/HLE/FL2/…

[HeuleJärvisaloBiere SAT'11]

- time stamping in previous example does not cover $b \to h$
  $[11,16] = [\text{dsc}(b), \text{fin}(b)] \not\subseteq [\text{dsc}(h), \text{fin}(h)] = [3,4]$
  $[17,28] = [\text{dsc}(\bar{h}), \text{fin}(\bar{h})] \not\subseteq [\text{dsc}(\bar{b}), \text{fin}(\bar{b})] = [8,9]$

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

- "coverage" heavily depends on DFS order

- as solution we propose multiple **randomized DFS** rounds/phases

- so we approximate a quadratic problem (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**
        $\mathrm{dsc}(l) := 0$; $\mathrm{fin}(l) := 0$
        $\mathrm{prt}(l) := l$; $\mathrm{root}(l) := l$
    **foreach** $r \in \mathrm{RTS}(F)$ **do**
        $stamp :=$ *Stamp*$(r, stamp)$
    **foreach** literal $l$ in BIG$(F)$ **do**
        **if** $\mathrm{dsc}(l) = 0$ **then**
            $stamp :=$ *Stamp*$(l, stamp)$
    **return** *Simplify*$(F)$
```

```
*Stamp* (literal $l$, integer $stamp$)
    $stamp := stamp + 1$
    $\mathrm{dsc}(l) := stamp$
    **foreach** $(\bar{l} \vee l') \in F_2$ **do**
        **if** $\mathrm{dsc}(l') = 0$ **then**
            $\mathrm{prt}(l') := l$
            $\mathrm{root}(l') := \mathrm{root}(l)$
            $stamp :=$ *Stamp*$(l', stamp)$
    $stamp := stamp + 1$
    $\mathrm{fin}(l) := stamp$
    **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** $\text{dsc}(l_{\text{neg}}) > \text{dsc}(l_{\text{pos}})$ **then**
      **if** $l_{\text{pos}}$ is last element in $S^{+}(C)$ **then return** false
      $l_{\text{pos}} :=$ next element in $S^{+}(C)$
    **else if** $\text{fin}(l_{\text{neg}}) < \text{fin}(l_{\text{pos}})$ **or** $(|C| = 2$ **and** $(l_{\text{pos}} = \bar{l}_{\text{neg}}$ **or** $\text{prt}(l_{\text{pos}}) = l_{\text{neg}}))$ **then**
      **if** $l_{\text{neg}}$ 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 $\text{dsc}()$
$S^{-}(C)$	sequence of negations of literals in C ordered by $\text{dsc}()$

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

Unhiding HLE

[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 $stamp$
```

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	sol	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

Conclusions

- preprocessing is important for SAT solvers
- hard kernels do occur in practice
- inprocessing provides additional benefits
- new class of clause elimination procedures
- even quadratic algorithms are most of the time too expensive