Unhiding Redundancy in SAT

Armin Biere

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

joint work with Marijn Heule and Matti Järvisalo

Deduction at Scale 2011

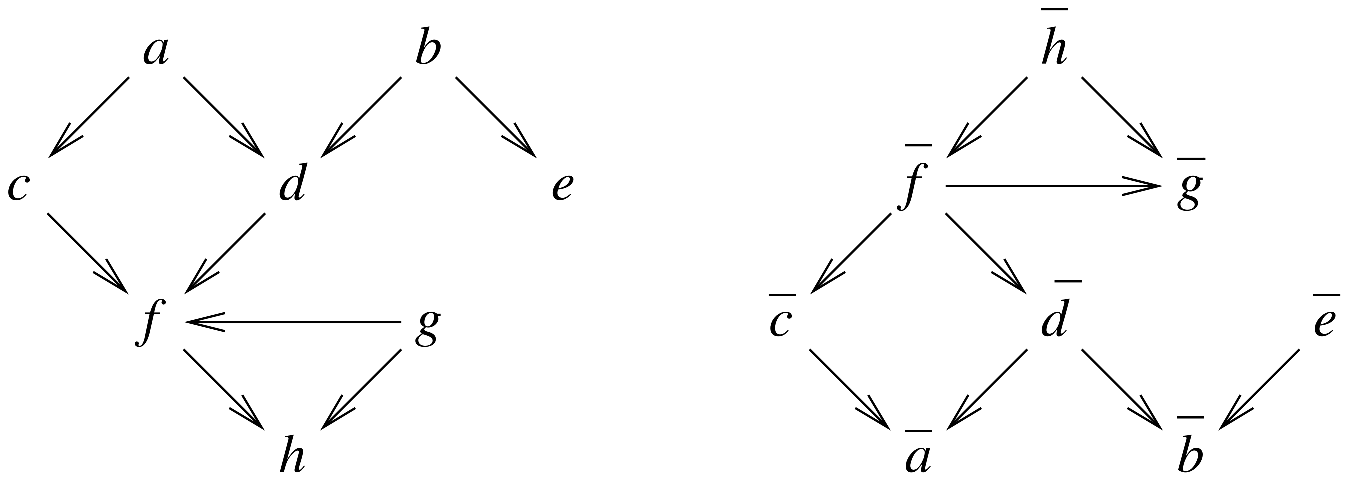
Ringberg Castle, Tegernsee, Germany

Monday, March 7, 2011

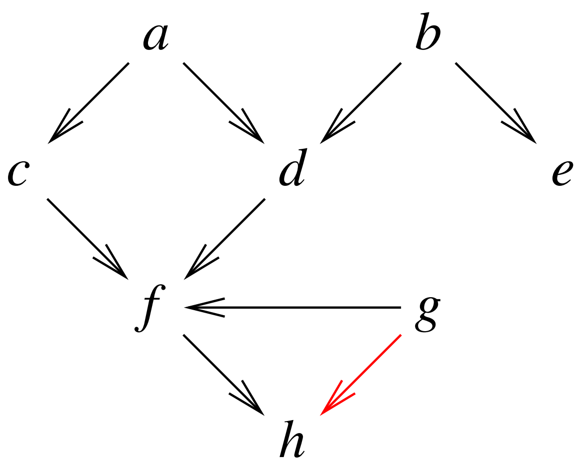
Motivation

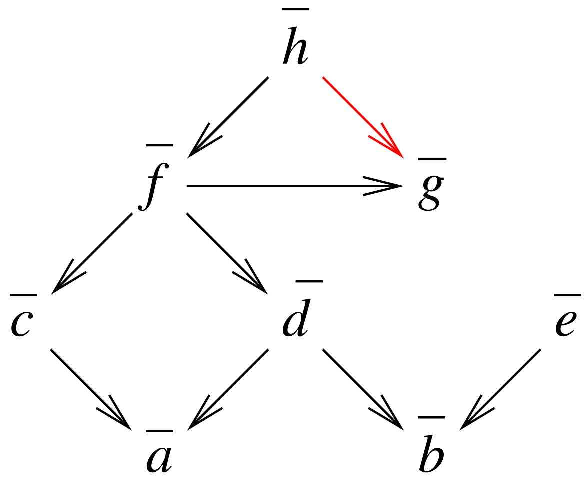
- SAT solvers applied to huge formulas
  - million of variables
  - fastests solvers use preprocessing/inprocessing
  - *need cheap and effective inprocessing techniques for millions of variables*
- this talk:
  - **unhiding** redundancy in large formulas
  - almost linear randomized algorithm
  - using the binary implication graph
  - fast enough to be applied to learned clauses
- paper submitted, available on request

Binary Implication Graph BIG



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





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

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

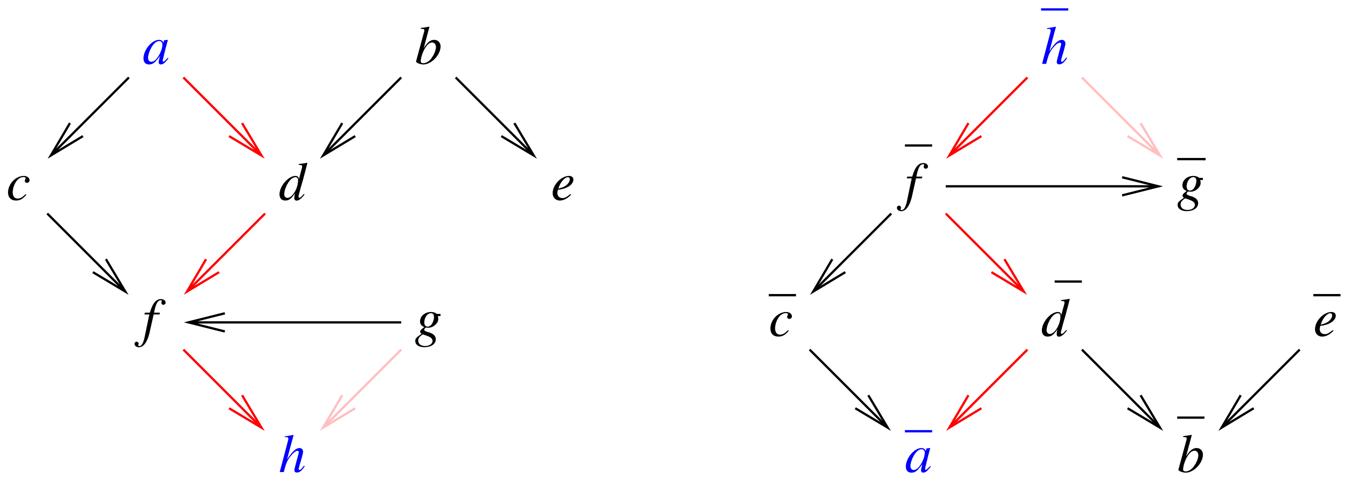
~~$(\bar{g} \vee h)$~~ $\wedge (\bar{a} \vee \bar{e} \vee h) \wedge (\bar{b} \vee \bar{c} \vee h) \wedge (a \vee b \vee c \vee d \vee e \vee f \vee g \vee h)$

TRD

$g \rightarrow f \rightarrow h$

Hidden Tautology Elimination HTE

[HeuleJärvisaloBiere LPAR'2010]



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

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

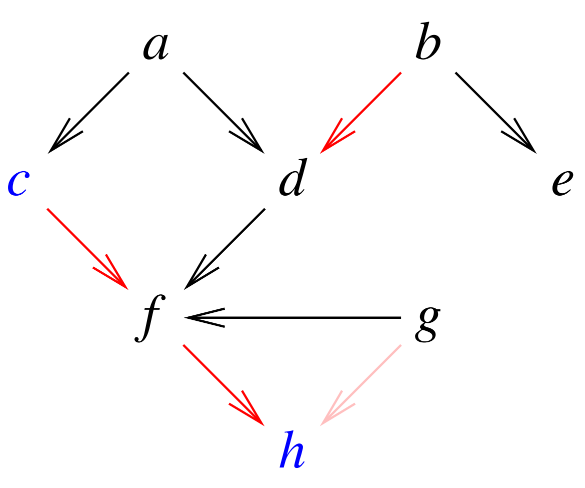
$$\sout{(\bar{a} \vee \bar{e} \vee h)} \wedge (\bar{b} \vee \bar{c} \vee h) \wedge (a \vee b \vee c \vee d \vee e \vee f \vee g \vee h)$$

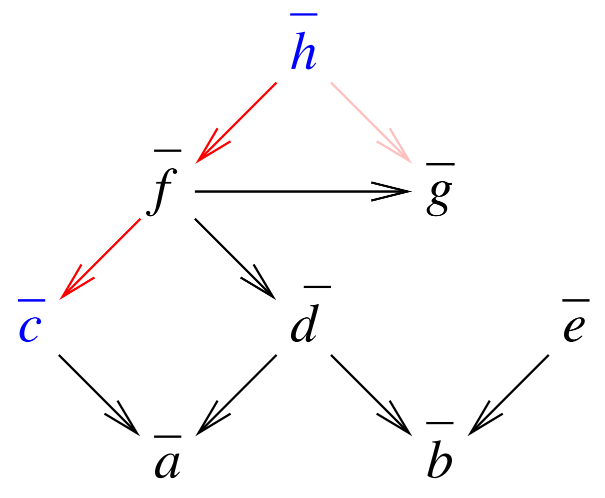
HTE

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

Hidden Tautology Elimination HTE

[HeuleJärvisaloBiere LPAR'2010]





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

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

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

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

[HeuleJärvisaloBiere submitted]

better explained on binary implication graph

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

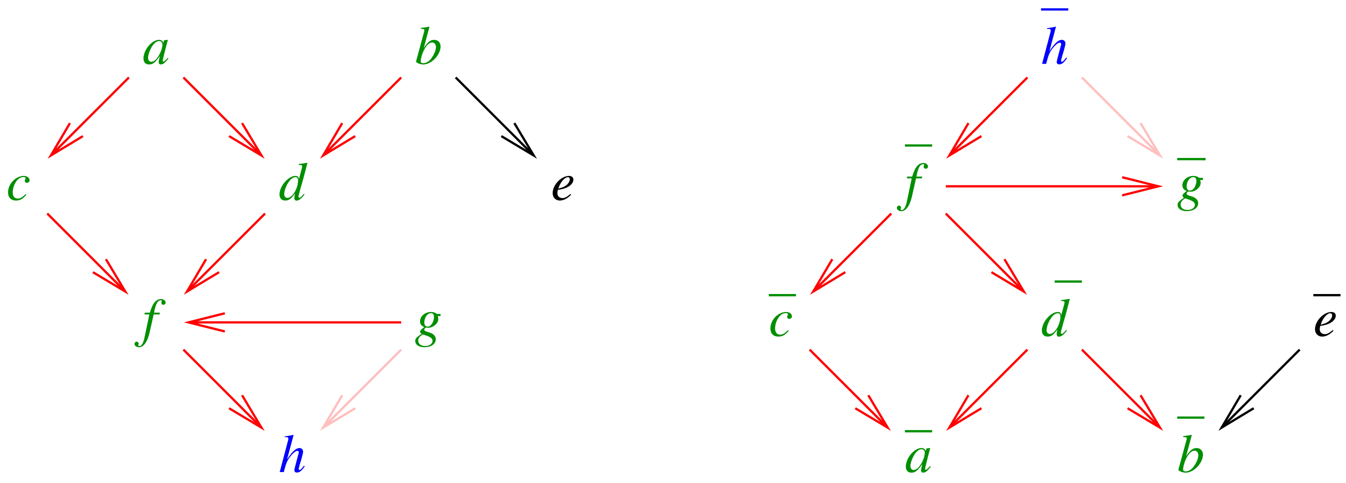
$$\ldots(\bar{a}\vee b)(\bar{b}\vee c)(a\vee c\vee d)\ldots \quad\Rightarrow\quad \ldots(\bar{a}\vee b)(\bar{b}\vee c)(c\vee d)\ldots$$

related work before all uses BCP:

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

HTE/HLE only uses the binary implication graph!

[HeuleJärvisaloBiere submitted]



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

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

$(\sout{a \vee b \vee c \vee d \vee} \; e \vee \sout{f \vee g \vee} \; h)$

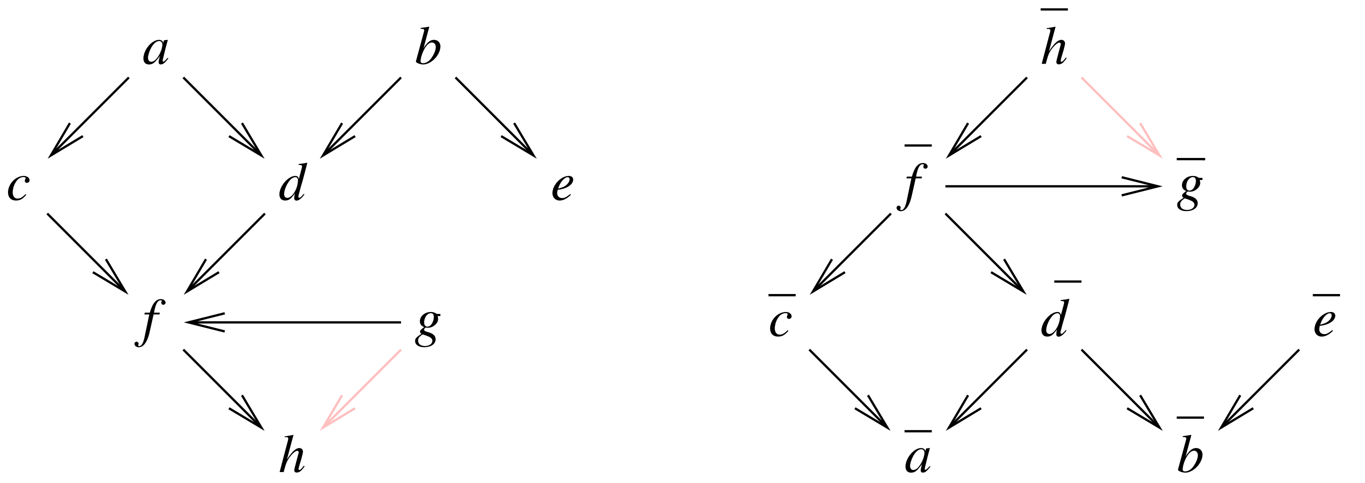
HLE

all but e imply h

also b implies e

Hidden Literal Elimination HLE

[HeuleJärvisaloBiere submitted]

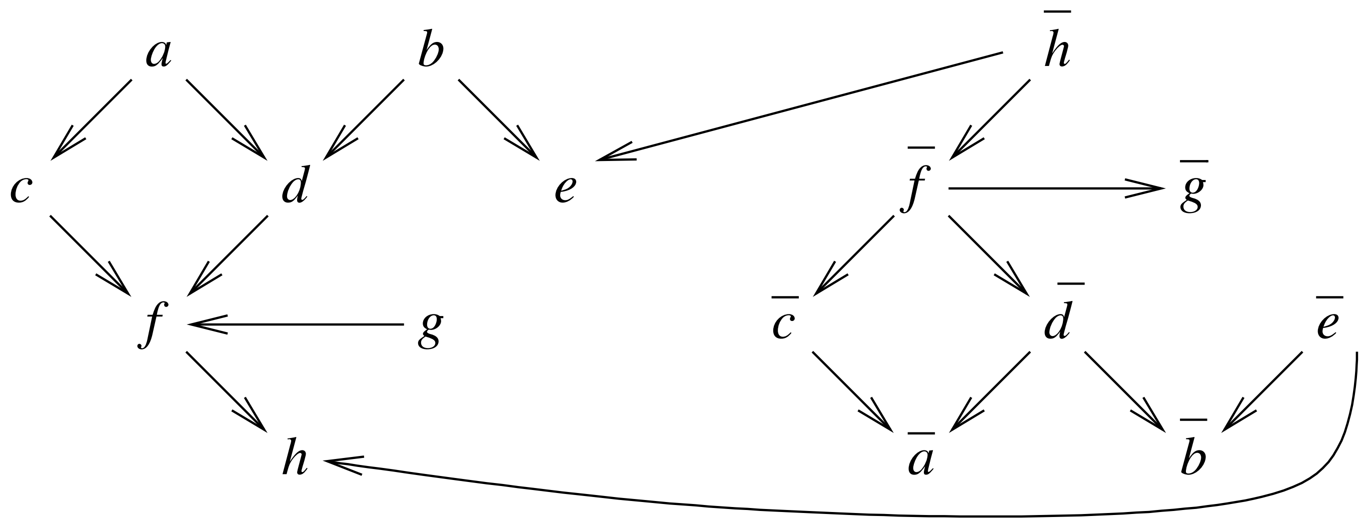


$$(\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge$$

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

$$(\quad e \vee \quad {\color{blue}h})$$

[HeuleJärvisaloBiere submitted]



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

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

assume literal l, 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 $\Rightarrow$ *unsatisfiable*

otherwise replace all literals by a "representative"

linear algorithm can be applied routinely during garbage collection

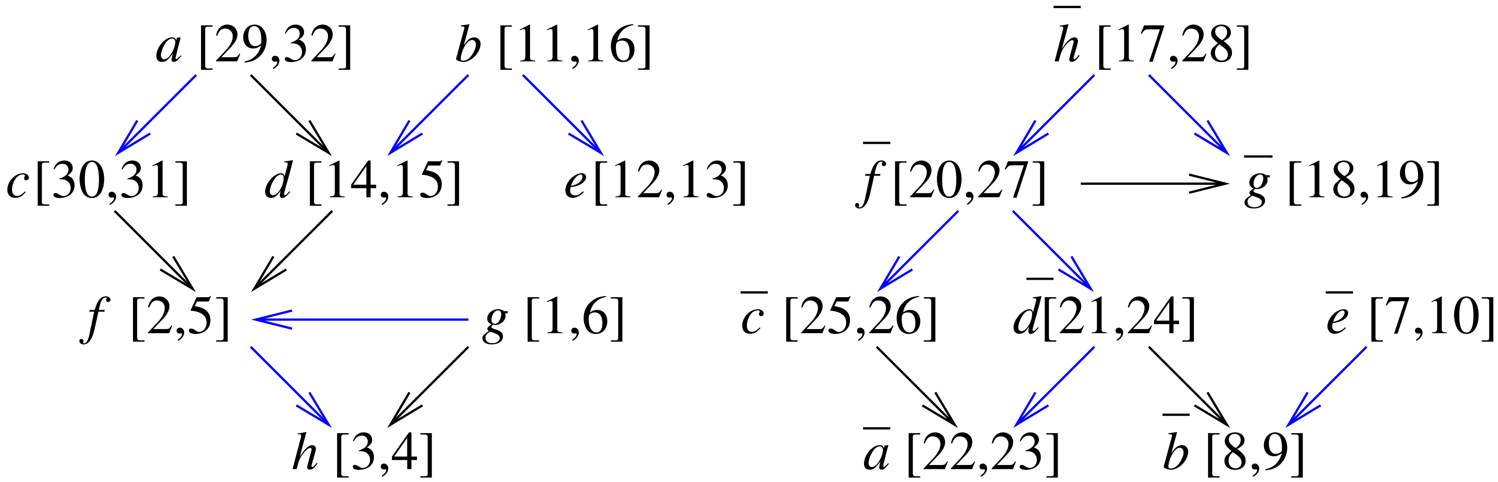
but as with failed literal preprocessing may generate new binary clauses

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

Time Stamping

[HeuleJärvisaloBiere submitted]

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 $l \rightarrow k$

[HeuleJärvisaloBiere submitted]

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

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

- “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 submitted]

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 submitted]

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 submitted]

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 $S^+(C)$

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

[HeuleJärvisaloBiere submitted]

```
    Stamp (literal l, integer stamp)
BSC   stamp := stamp + 1
BSC   dsc(l) := stamp; obs(l) := stamp
ELS   flag := true                                   // l represents a SCC
ELS   S.push(l)                                      // push l on SCC stack
BSC   for each (l̄ ∨ l′) ∈ F2
TRD     if dsc(l) < obs(l′) then F := F \ {(l̄ ∨ l′)}; continue
FLE     if dsc(root(l)) ≤ obs(l̄′) then
FLE       l_failed := l
FLE       while dsc(l_failed) > obs(l̄′) do l_failed := prt(l_failed)
FLE       F := F ∪ {(l̄_failed)}
FLE       if dsc(l̄′) ≠ 0 and fin(l̄′) = 0 then continue
BSC     if dsc(l′) = 0 then
BSC       prt(l′) := l
BSC       root(l′) := root(l)
BSC       stamp := Stamp(l′, stamp)
ELS     if fin(l′) = 0 and dsc(l′) < dsc(l) then
ELS       dsc(l) := dsc(l′); flag := false           // l is equivalent to l′
OBS     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       dsc(l′) := dsc(l)                          // assign equal discovered time
BSC       fin(l′) := stamp                           // assign equal finished time
ELS     while l′ ≠ l
BSC   return stamp
```

[HeuleJärvisaloBiere submitted]

- 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

[HeuleJärvisaloBiere submitted]

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