

Unhiding Redundancy in SAT

Armin Biere

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

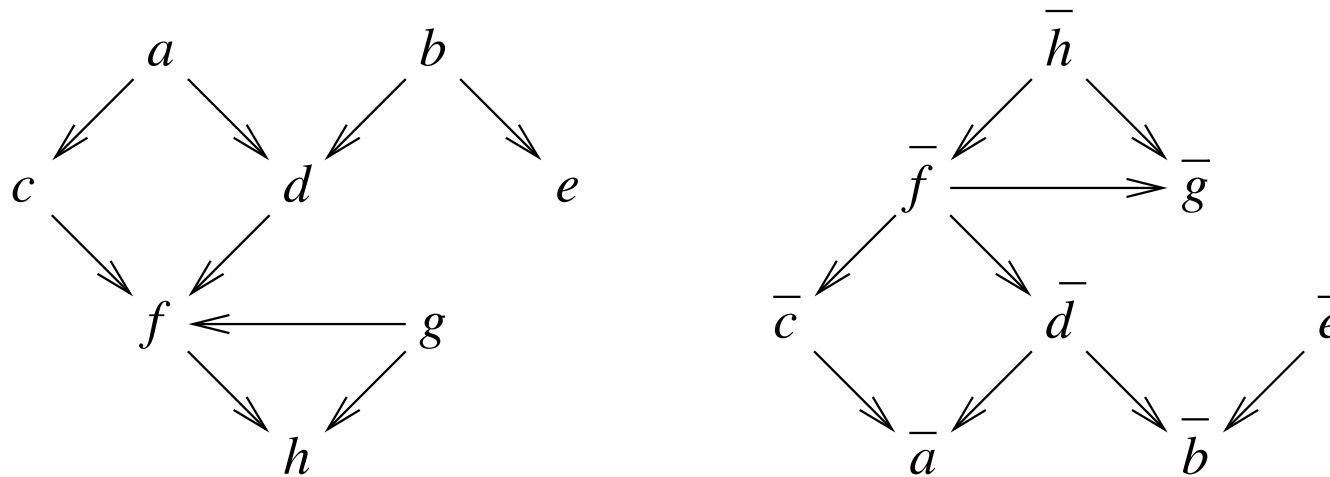
joint work with Marijn Heule and Matti Järvisalo

Deduction at Scale 2011

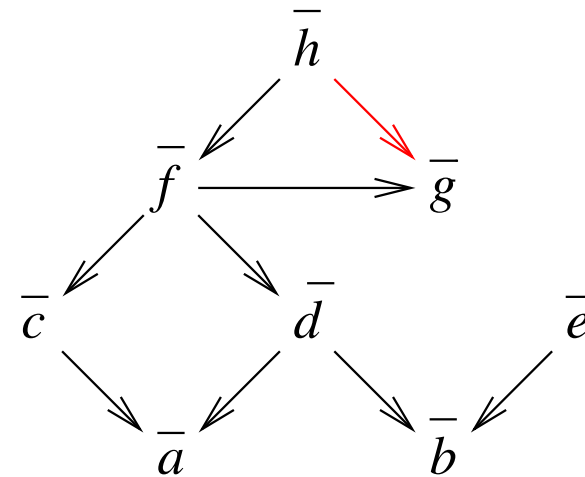
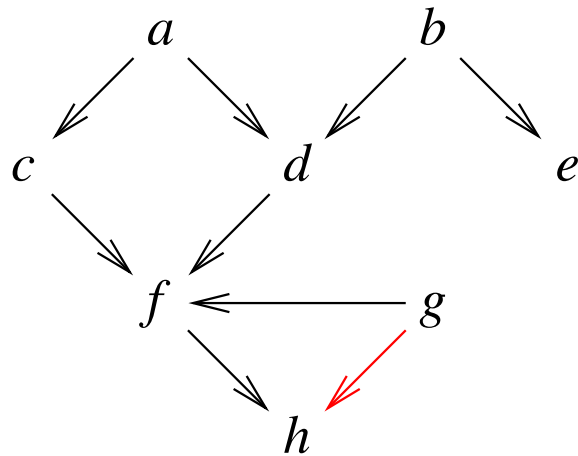
Ringberg Castle, Tegernsee, Germany

Monday, March 7, 2011

- SAT solvers applied to huge formulas
 - million of variables
 - fastest 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



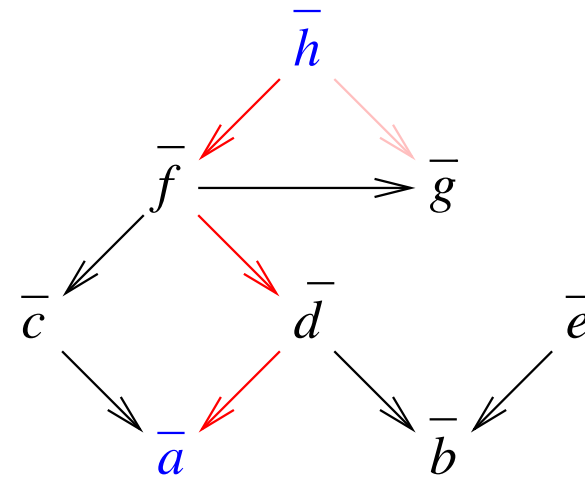
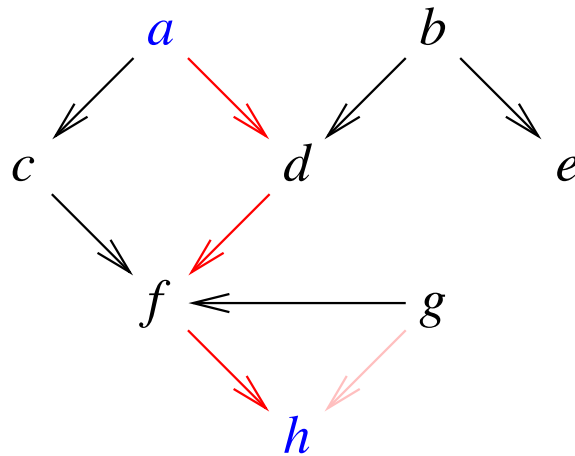
$$\begin{aligned}
 & (\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge \\
 & (\bar{c} \vee f) \wedge (\bar{d} \vee f) \wedge (\bar{g} \vee f) \wedge (\bar{f} \vee h) \wedge \\
 & (\bar{g} \vee h) \wedge \underbrace{(\bar{a} \vee \bar{e} \vee h) \wedge (\bar{b}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



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

TRD

$$g \rightarrow f \rightarrow h$$



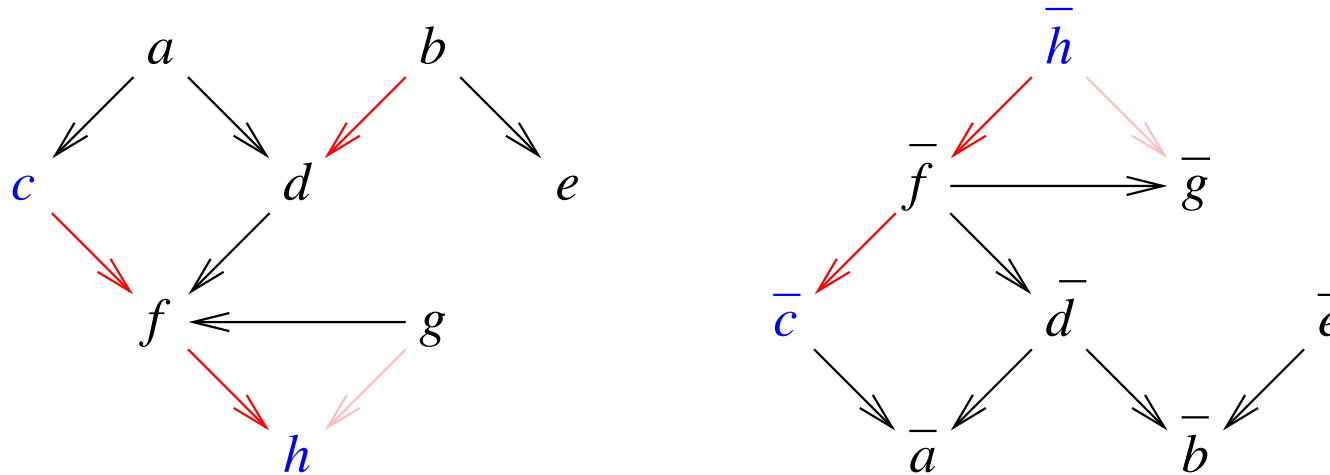
$$(\bar{a} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{b} \vee d) \wedge (\bar{b} \vee e) \wedge$$

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

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

HTE

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



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

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

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

HTE

$$c \rightarrow f \rightarrow h$$

$$\frac{C \vee l \quad D \vee \bar{l}}{D} \quad C \subseteq D$$

$$\frac{a \vee b \vee l \quad a \vee b \vee c \vee \bar{l}}{a \vee b \vee c}$$

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

assume given CNF contains both antecedents

$$\dots (a \vee b \vee l) (a \vee b \vee c \vee \bar{l}) \dots$$

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

$$\Downarrow$$

which in essence *removes* \bar{l} from $D \vee \bar{l}$

$$\dots (a \vee b \vee l) (a \vee b \vee c) \dots$$

used in SATeLite preprocessor

now common in many SAT solvers

hidden literal addition (HLA) uses SSR in reverse order

$$\frac{C \vee l \quad D \vee \bar{l}}{D} \quad C \subseteq D$$

$$\frac{a \vee b \vee l \quad a \vee b \vee c \vee \bar{l}}{a \vee b \vee c}$$

assume given CNF contains resolvent and first antecedent

$$\dots (a \vee b \vee l)(a \vee b \vee c) \dots$$

we can replace D by $D \vee \bar{l}$

$$\dots (a \vee b \vee l)(a \vee b \vee c \vee \bar{l}) \dots$$

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

better explained on binary implication graph

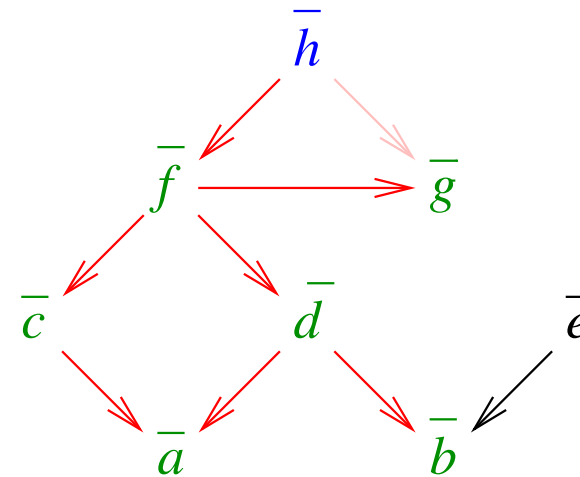
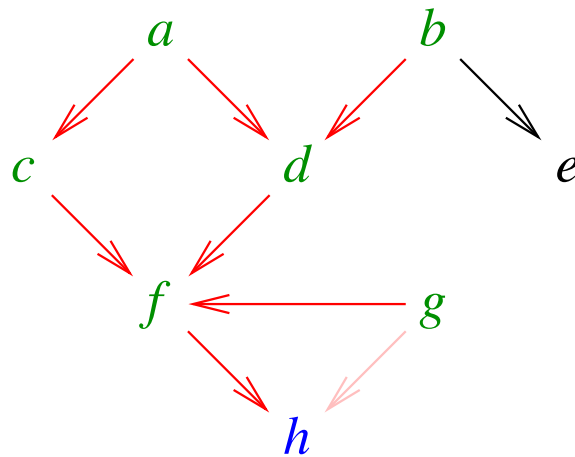
remove literal from a clause which implies another literal in the clause

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

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!



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

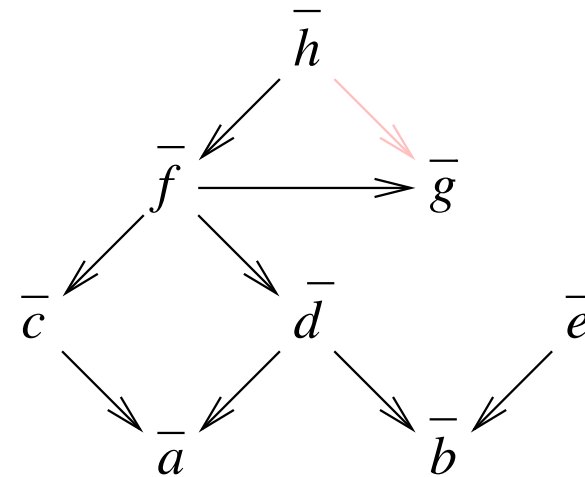
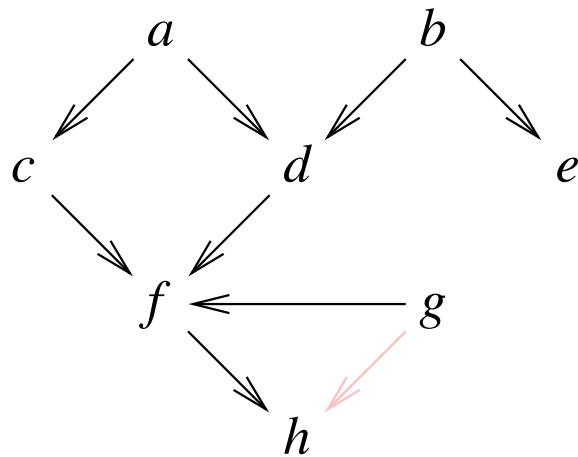
$$(\bar{c} \vee f) \wedge (\bar{d} \vee f) \wedge (\bar{g} \vee f) \wedge (\bar{f} \vee h) \wedge$$

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

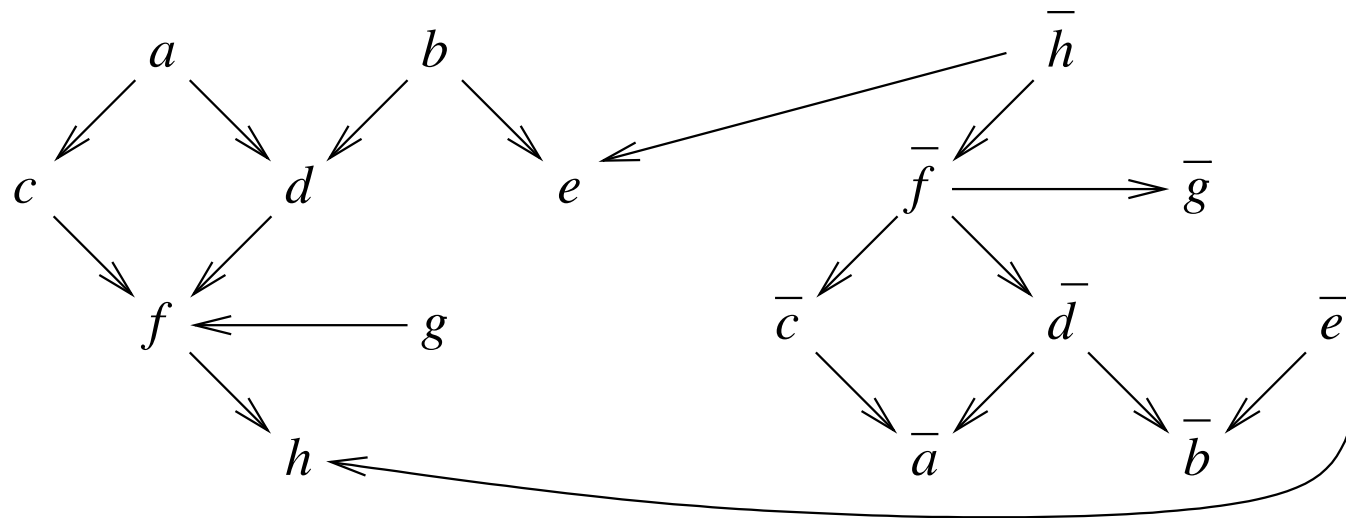
HLE

all but e imply h

also b implies e



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



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

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]

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

subsumed by running one HLA until completion

decompose BIG into strongly connect components (SCCs)

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

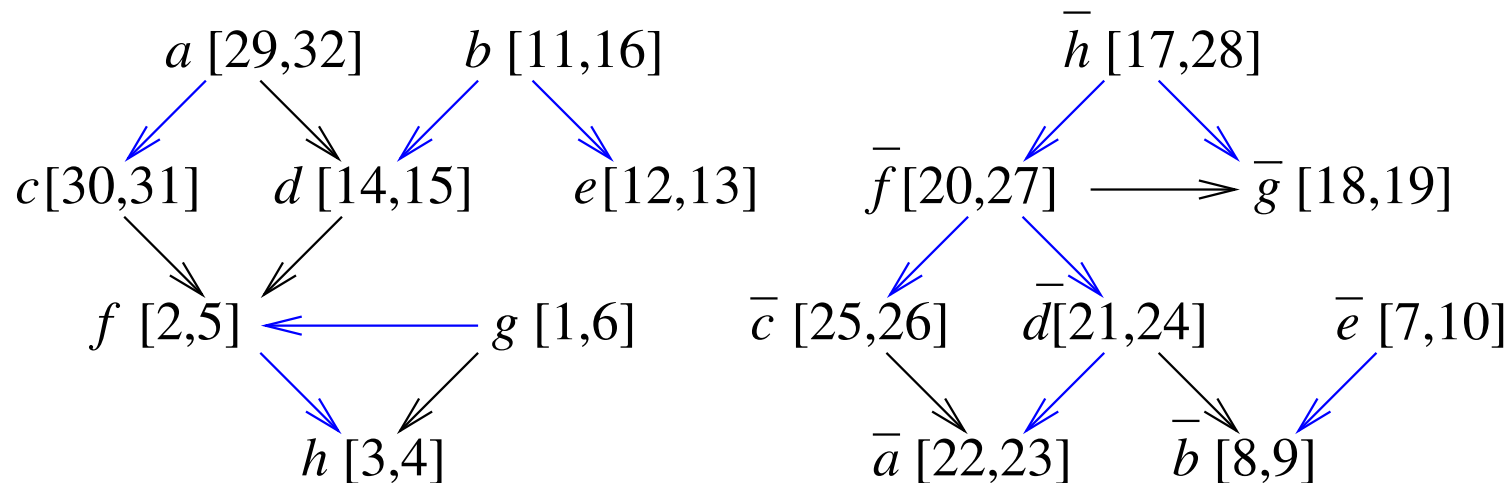
otherwise replace all literals by a “representative”

linear algorithm can be applied routinely during garbage collection

but as with failed literal preprocessing may generate new binary clauses

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

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



tree edges

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

ancestor relationship gives necessary conditions for (transitive) implication:

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

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

- time stamping in previous example does not cover $b \rightarrow h$
 $[11, 16] = [\text{dsc}(b), \text{fin}(b)] \not\subseteq [\text{dsc}(h), \text{fin}(h)] = [3, 4]$
 $[17, 28] = [\text{dsc}(\bar{h}), \text{fin}(\bar{h})] \not\subseteq [\text{dsc}(\bar{b}), \text{fin}(\bar{b})] = [8, 9]$
- in example still both HTE “unhidden”, HLE works too (since $b \rightarrow e$)
- “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 (formula F)

```

1   $stamp := 0$ 
2  foreach literal  $l$  in  $BIG(F)$  do
3     $dsc(l) := 0; fin(l) := 0$ 
4     $prt(l) := l; root(l) := l$ 
5  foreach  $r \in RTS(F)$  do
6     $stamp := Stamp(r, stamp)$ 
7  foreach literal  $l$  in  $BIG(F)$  do
8    if  $dsc(l) = 0$  then
9       $stamp := Stamp(l, stamp)$ 
10 return  $Simplify(F)$ 

```

Stamp (literal l , integer $stamp$)

```

1   $stamp := stamp + 1$ 
2   $dsc(l) := stamp$ 
3  foreach  $(\bar{l} \vee l') \in F_2$  do
4    if  $dsc(l') = 0$  then
5       $prt(l') := l$ 
6       $root(l') := root(l)$ 
7       $stamp := Stamp(l', stamp)$ 
8   $stamp := stamp + 1$ 
9   $fin(l) := stamp$ 
10 return  $stamp$ 

```

Simplify (formula F)

```

1  foreach  $C \in F$ 
2     $F := F \setminus \{C\}$ 
3    if  $UHTE(C)$  then continue
4     $F := F \cup \{UHLE(C)\}$ 
5  return  $F$ 

```

UHTE (clause C)

```

1    $l_{\text{pos}} :=$  first element in  $S^+(C)$ 
2    $l_{\text{neg}} :=$  first element in  $S^-(C)$ 
3   while true
4     if  $\text{dsc}(l_{\text{neg}}) > \text{dsc}(l_{\text{pos}})$  then
5       if  $l_{\text{pos}}$  is last element in  $S^+(C)$  then return false
6        $l_{\text{pos}} :=$  next element in  $S^+(C)$ 
7     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
8       if  $l_{\text{neg}}$  is last element in  $S^-(C)$  then return false
9        $l_{\text{neg}} :=$  next element in  $S^-(C)$ 
10  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 $\text{dsc}()$

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

UHLE (clause C)

```
1  finished := finish time of first element in  $S_{\text{rev}}^+(C)$ 
2  foreach  $l \in S_{\text{rev}}^+(C)$  starting at second element
3      if  $\text{fin}(l) > \textit{finished}$  then  $C := C \setminus \{l\}$ 
4      else  $\textit{finished} := \text{fin}(l)$ 
5  finished := finish time of first element in  $S^-(C)$ 
6  foreach  $\bar{l} \in S^-(C)$  starting at second element
7      if  $\text{fin}(\bar{l}) < \textit{finished}$  then  $C := C \setminus \{l\}$ 
8      else  $\textit{finished} := \text{fin}(\bar{l})$ 
9  return  $C$ 
```

$S_{\text{rev}}^+(C)$ reverse of $S^+(C)$

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

```

Stamp (literal  $l$ , integer  $stamp$ )
1 BSC    $stamp := stamp + 1$ 
2 BSC    $dsc(l) := stamp; obs(l) := stamp$ 
3 ELS    $flag := true$  //  $l$  represents a SCC
4 ELS    $S.push(l)$  // push  $l$  on SCC stack
5 BSC   for each  $(\bar{l} \vee l') \in F_2$ 
6 TRD   if  $dsc(l) < obs(l')$  then  $F := F \setminus \{(\bar{l} \vee l')\}$ ; continue
7 FLE   if  $dsc(\text{root}(l)) \leq obs(\bar{l}')$  then
8 FLE    $l_{\text{failed}} := l$ 
9 FLE   while  $dsc(l_{\text{failed}}) > obs(\bar{l}')$  do  $l_{\text{failed}} := \text{prt}(l_{\text{failed}})$ 
10 FLE   $F := F \cup \{(\bar{l}_{\text{failed}})\}$ 
11 FLE  if  $dsc(\bar{l}')$   $\neq 0$  and  $\text{fin}(\bar{l}')$   $= 0$  then continue
12 BSC  if  $dsc(l') = 0$  then
13 BSC   $\text{prt}(l') := l$ 
14 BSC   $\text{root}(l') := \text{root}(l)$ 
15 BSC   $stamp := \text{Stamp}(l', stamp)$ 
16 ELS  if  $\text{fin}(l') = 0$  and  $dsc(l') < dsc(l)$  then
17 ELS   $dsc(l) := dsc(l')$ ;  $flag := false$  //  $l$  is equivalent to  $l'$ 
18 OBS   $obs(l') := stamp$  // set last observed time attribute
19 ELS  if  $flag = true$  then // if  $l$  represents a SCC
20 BSC   $stamp := stamp + 1$ 
21 ELS  do
22 ELS   $l' := S.pop()$  // get equivalent literal
23 ELS   $dsc(l') := dsc(l)$  // assign equal discovered time
24 BSC   $\text{fin}(l') := stamp$  // assign equal finished time
25 ELS  while  $l' \neq l$ 
26 BSC  return  $stamp$ 

```

- 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 particularly effective for learned clauses
- beside UHTE and UHLE we also have added hyper binary resolution UHBR
not useful in practice

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