# Lazy Hyper Binary Resolution

Armin Biere

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

Dagstuhl Seminar 09461

Algorithms and Applications for
Next Generation SAT Solvers

Schloss Dagstuhl, Germany

November 2009

- 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 is 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 $l' \in \mathrm{UnitProp}(f \wedge \bar{l})$ or vice versa

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

# Previous Optimizations



[BacchusWinter-SAT03][GershmanStrichman-SAT05]

- maintain acyclic and transitively-reduced binary implication graph
  - acyclic: (incremental) 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 $l \in \text{UnitProp}(r)$ and $\text{UnitProp}(r)$ does not produce anything $\Rightarrow$ no need to probe $l$
  - at least with respect to units it is possible to focus on roots

- still as with failed literal probing, too expensive to run until completion

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

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

  – Are there CNFs where one transitively reduced hyper binary resolution closure is quadratic in size with respect to the size of the original CNF?

  – where size = #clauses or size = #literals

- hyber 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 $b \in \text{UnitProp}(F \wedge a)$ and $a \in \text{UnitProp}(F \wedge b)$

- can not simulate structural hashing of XOR or ITE gates

- learn binary clauses lazily or on-the-fly
  - in BCP
  - during preprocessing with failed literal probing
  - or during search

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

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

# Avoid Transitive Edges by Prioritizing Binary Clauses 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 $l$ calculate **one** dominator $\quad \mathrm{bindom}(l)$

- in the implication graph restricted to binary clauses

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

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

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

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

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

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

- interleave search and preprocessing
  - bound time spent in search to roughly 80%
  - measured in number of propagations / resolutions
- BCP during search learns binary clauses through LHBR (search LHBR)
- during preprocessing / simplification on the top level
  - unit propagation on the top-level does LHBR (top-level LHBR)
  - failed literal probing learns most binary clauses through LHBR (probing LHBR)
- effectiveness of LHBR reduced due to “lifting” in failed literal probing (also in PicoSAT)

- 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

- no measurable overhead doing LHBR during BCP
  - so at least not harmful (in contrast to many other “optimizations”)
  - except for rare cases where it produces too many clauses
  - most recent version limits number of learned binary clauses to 5 million
- unpublished but implemented in PrecoSAT
  - source code of PrecoSAT available under MIT license
- it actually performs a limited version of on-the-fly strengthening / subsumption

  [HanSomenzi-SAT09] [HamadiJabbourSais’09]

# Future Work



- how to run LHBR and / or failed literal probing until completion
- transitive reduction (initially and after equality reduction)
- incremental SCC decomposition and equality reduction
- determine time / space complexity for the problem
- how to simulate structural hashing of XOR / ITE gates
- get more experimental data on
  - how often this actually happens and for which benchmarks
  - the empirical relation to on-the-fly subsumption