Inprocessing in SAT solvers
Presentation at SAT/SMT Summer School 2015

Mate Soos
Cisco

16th of July 2011
Outline

Motivation

Clause massaging

Variable massaging

Takeaways
The problem

- CDCL is nice but doesn’t do everything
- No satisfied clause removal
- No replacement when $a = (\neg)b$
- No removal of trivially satisfiable clauses
- etc.
The solution

SIMPLIFY

ALL THE THINGS
Outline

Motivation

Clause massaging

Variable massaging

Takeaways
Concept: literal occurrence list

Note: misnomer, it’s really an array of arrays. Array provides better cache locality.
Tautologies

- No point in having a clause $a \lor b \lor \neg b$
- We remove it during parsing of CNF
- Sort clause $\rightarrow O(n \log n)$
- Check linearly if previous literal is inverse of current one
- Mem need: 1 lit + clause for in-place sort
Unit clauses

- Keep an array of assignments: Undefined, True, False
- Unit clauses are only stored here
- Remove clauses containing a literal set to True
- Remove from clauses literals set to False
- Time: $O(\sum lits)$
- Note: we keep 1B to store the 3 values. 2b would be enough, but it’s accessed often and x86/amd64 is slow at bit manipulation
Pure literal rule

- If variable $x$ only appears in positive or only in negative form, all clauses it appears in can be trivially satisfied.
- Time: $O(\sum \text{lits})$, Mem: $O(\text{num vars})$
- Alternatively, once occur list is built, we have the info already so time is $O(nvars)$
Subsumption [EenBiere’07]

- $a \lor b$ is stronger than $a \lor b \lor c$. No need for former
- Use occurrence list of clauses
- Backward subsumtion: check if $a \lor b$ subsumes anything
- Cheaper, because otherwise we don’t know which literal(s) from $a \lor b \lor c$ are missing and so we have to try them all
- Also, we can start with shorter clauses and ignore longer ones
- Tricks used: bloom filter, fingerprint stored inside occurrence list (no dereferencing of ptr)
- Time need: go through occurrence lits of lit with smallest occur size for each clause
- Note: unit clause removing clauses is subsumption
Self-subsuming resolution [EenBiere’07]

- $a \lor \neg b \lor c$ resolved with $a \lor b$ is $a \lor c$ which subsumes former
- Use occurrence list of clauses
- We do it backwards for same reason as before: take small $A$ and find all longer clauses $B$ that can be shortened to $B'$
- Sort the clauses’ literals, go through linearly, abort if any variable is in $A$ but not in $B$, only allow one inverted variable.
- Tricks used: bloom filter, fingerprint stored inside occur list (no dereferencing of ptr), if none are inverted it’s subsumed
- Time need: just like subsumption, but have to go through both the lit’s and its inverse’s occur list
- Note: unit clause removing literals from clauses is self-subsuming resolution
Concept: binary implication graph (BIG)

CNF clauses
- \( \neg a \lor b, \neg a \lor \neg c \)
- \( \neg b \lor \neg e, c \lor d \)
- \( e \lor f, \neg d \lor f \)
- \( \neg f \lor a \)

Transitive clauses:
- \( \neg a \lor \neg e \)
- \( \neg a \lor d \)
- \( \neg a \lor f \)
- \( c \lor f \)
- \( \ldots \)
% Stampin'[HelueJarvisaloBiere’11]

\[ \neg a \lor b, \neg a \lor \neg c, c \lor d \]

- DFS through BIG, note “START, END” times
- Dec sort clauses’ lits according to START, if previous lit’s END is smaller than current, remove current lit
- Inc sort clauses’ lits according to START of inverted lit, if previous inverted lit’s END is larger than the current’s, remove current lit

\[ g \lor b \lor a \]. 1st case, sorted: 
\[ b \lor a \lor g \]
Outline

Motivation

Clause massaging

Variable massaging

Takeaways
Equivalent literal substitution [WarnersMaaren’1999]

- Find strongly connected component (SCC) in BIG
- SCC: every node can be reached from every node
- It’s an equivalence class
- Use Tarjan’s algorithm to find it, linear time
- Replace equivalent literals
- Some clauses become tautologies, some become shorter
- Ex.: $a = \neg b$, so $a \lor b \lor c$ is a tautology
- Ex.: $a = \neg b$, so $a \lor \neg b \lor d \rightarrow a \lor d$
- Note: above two can be simulated by respectively subsumption and self-subsuming resolution if we take transitive binary clauses into account
Failed Literal Probing [Freeman’95], [Stalmarck’92]

- Enqueue $a$, see if it fails. If fails $\rightarrow$ set $\neg a$
- Enqueue $\neg a$. If both $a$ and $\neg a$ propagate $b$, set $b$
- Keep an array of variables set for $a$, check against $\neg a$
- Can be very expensive — we need to roll back every time
- Trick: build stack from binary clauses, enqueue iteratively, reuse old propagations
- Example $a \rightarrow b \rightarrow c$ So enqueue $b$, propagate. Then enqueue $a$ and propagate
- No need to roll back so often!
Bounded variable elimination [EenBierré’05]

- Through clause distribution
- \((a \lor b) \land (\neg a \lor c, \neg a \lor d)\)
- Becomes \(b \lor c, b \lor d\)
- Note: lots of resolvents are Tautologies
- Bounded: don’t increase num cl, abort if resolvent too large
- Call backward subsumption with resolved clause
- Order matters, e.g. order according to least occurred variable
- Trick: try to reverse strengthen to remove resolvent clause
- Note: simulates pure literal rule
Bounded variable addition [HeuleJarvisaloBiere’13]

Replace

\[(a \lor d) (a \lor e) \]
\[(b \lor d) (b \lor e) \]
\[(c \lor d) (c \lor e) \]

by

\[(-x \lor a) (-x \lor b) (-x \lor c) \]
\[(x \lor d) (x \lor e) \]

- Started off with 6 clauses and 12 literals
- Finished with 5 clauses and 10 literals
- We can have any number of “columns” (here: 2)
- We can have any set of literals (here: \(d, e\))
Variable renumbering

- Variables are set, eliminated, replaced
- Renumber them to pack them tightly, use a simple map
- Lowers memory footprint
- More tightly packed structs $\rightarrow$ better cache usage
- But maintaining the map with BVE-(un)eliminated, BVA-added (and later BVE-removed...), replaced, and new variables can be complicated
Takeaways

- We can make the work of SAT solvers easier
- Improving the clause set
- Improving the variable set
- Thereby improving efficiency of heuristics, memory footprint, cache locality, etc.
- Some methods can be (partially) simulated by other, more general methods — but runtimes will be *vastly* different