2.9.10 Lemma (CDCL Soundness)

In a reasonable CDCL derivation, CDCL can only terminate in two different types of final states: $(M; N; U; k; \top)$ where $M \models N$ and $(M; N; U; k; \bot)$ where N is unsatisfiable.

2.9.11 Proposition (CDCL Soundness)

The rules of the CDCL algorithm are sound: (i) if CDCL terminates from $(\epsilon; N; \emptyset; 0; \top)$ in the state $(M; N; U; k; \top)$, then N is satisfiable, (ii) if CDCL terminates from $(\epsilon; N; \emptyset; 0; \top)$ in the state $(M; N; U; k; \bot)$, then N is unsatisfiable.

2.9.12 Proposition (CDCL Strong Completeness)

The CDCL rule set is complete: for any valuation M with $M \models N$ there is a reasonable sequence of rule applications generating $(M'; N; U; k; \top)$ as a final state, where M and M' only differ in the order of literals.

2.9.13 Proposition (CDCL Termination)

Assume the algorithm CDCL with all rules except Restart and Forget is applied using a reasonable strategy. Then it terminates in a state $(M; N; U; k; D)$ with $D \in \{\top, \bot\}$.

The Overall Picture

Application System + Problem
System Algorithm + Implementation
Algorithm Calculus + Strategy
Calculus Logic + States + Rules
Logic Syntax + Semantics

Algorithm: 5 CDCL(S)

Input : An initial state $(\epsilon; N; \emptyset; 0; \top)$.

Output: A final state $S = (M; N; U; k; \top)$ or $S = (M; N; U; k; \bot)$

while (*any rule applicable*) **do**

    ifrule (**Conflict**(S)) **then**

        while (**Skip**(S) $\|$ **Resolve**(S)) **do**

            update VSIDS on resolved literals;

        update VSIDS on learned clause, **Backtrack**(S);

        if (*forget heuristic*) **then**

            Forget(S), **Restart**(S);

        else

            if (*restart heuristic*) **then**

                Restart(S);

    else

        ifrule (! **Propagate**(S)) **then**

            Decide(S) literal with max. VSIDS score;

return(S);

Implementation: Data Structures

Propagate $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (ML^{C \vee L}; N; U; k; \top)$
provided $C \vee L \in (N \cup U)$, $M \models \neg C$, and L is undefined in M

Conflict $(M; N; U; k; \top) \Rightarrow_{\text{CDCL}} (M; N; U; k; D)$
provided $D \in (N \cup U)$ and $M \models \neg D$

Implementation

- data structures: clauses, trail, and the rules
- heuristics: decision literal, forget, restart
- space efficiency: forget
- quality: restarts
- special cases

Implementation

- data structures: clauses, trail, and the rules
- heuristics: decision literal, forget, restart
- space efficiency: forget
- quality: restarts
- special cases

Implementation

- data structures: clauses, trail, and the rules
- heuristics: decision literal, forget, restart
- space efficiency: forget
- quality: restarts
- special cases

Implementation

- data structures: clauses, trail, and the rules
- heuristics: decision literal, forget, restart
- space efficiency: forget
- quality: restarts
- special cases

Implementation

- data structures: clauses, trail, and the rules
- heuristics: decision literal, forget, restart
- space efficiency: forget
- quality: restarts
- special cases

Data Structures

Idea: Select two literals from each clause for indexing.

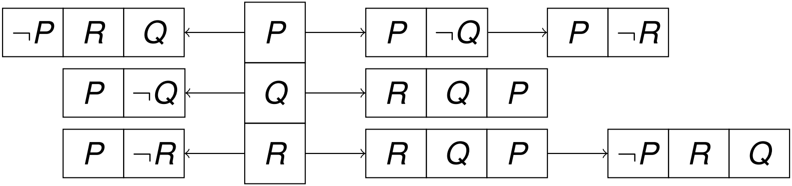
Data Structures

Idea: Select two literals from each clause for indexing.

2.10.1 Invariant (2-Watched Literal Indexing)

If one of the watched literals is false and the other watched literal is not true, then all other literals of the clause are false.

$$N = \{P \vee \neg R,\ P \vee \neg Q,\ R \vee Q \vee P,\ \neg P \vee R \vee Q\}$$



VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

VSIDS: Variable State Independent Decaying Sum

- each propositional variable has a positive *score*, initially 0
- decide the variable with maximal score, remember sign (*phase saving*)
- increment the score of variables involved in resolution by b
- increment the score of variables in learned clauses by b
- initially $b > 0$
- at Backtrack set $b := c * b$ where $2 >> c > 1$, i.e., $b_n = c^n * b$
- take care of overflows, i.e., rescale from time to time
- sometimes pick a variable randomly

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the k% clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the $k\%$ clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Forget

- fix a limit d on the number of learned clauses
- if more than $|U| > d$ start forgetting
- remove redundant clauses
- sort the learned clauses according to a score
- typical elements of the score are clause length, the VSIDS score, dependency on decisions
- remove the k% clauses with minimal score from U
- $d := d + e$ for some e, $e >> 1$
- do a Restart

Restart

- after forgetting do a restart
- if a unit is learned do a restart
- restart often at the beginning of a run
- classics: Luby sequence 1, 1, 2, 1, 1, 2, 4, . . .
  $(u_1, v_1) := (1, 1)$,
  $(u_{n+1}, v_{n+1}) := ((u_n \,\&\, -u_n) = v_n ? (u_n + 1, 1) : (u_n, 2 * v_n))$

Restart

- after forgetting do a restart
- if a unit is learned do a restart
- restart often at the beginning of a run
- classics: Luby sequence 1, 1, 2, 1, 1, 2, 4, . . .
  $(u_1, v_1) := (1, 1)$,
  $(u_{n+1}, v_{n+1}) := ((u_n \& -u_n) = v_n?(u_n + 1, 1) : (u_n, 2 * v_n))$

Restart

- after forgetting do a restart
- if a unit is learned do a restart
- restart often at the beginning of a run
- classics: Luby sequence 1, 1, 2, 1, 1, 2, 4, . . .
  $(u_1, v_1) := (1, 1)$,
  $(u_{n+1}, v_{n+1}) := ((u_n \& -u_n) = v_n?(u_n + 1, 1) : (u_n, 2 * v_n))$

Restart

- after forgetting do a restart
- if a unit is learned do a restart
- restart often at the beginning of a run
- classics: Luby sequence 1, 1, 2, 1, 1, 2, 4, ...
  $(u_1, v_1) := (1, 1)$,
  $(u_{n+1}, v_{n+1}) := ((u_n \,\&\, -u_n) = v_n ? (u_n + 1, 1) : (u_n, 2 * v_n))$

Memory Matters: SPASS-SATT

Forget-Start	800	108800
Restarts	412	369
Conflicts	153640	133403
Decisions	184034	159005
Propagations	17770298	15544812
Time	11	23
Memory	16	41

Propositional Logic Calculi

1. Tableau: classics, natural from the semantics
2. Resolution: classics, first-order, prepares for CDCL
3. CDCL: current prime calculus for propositional logic
4. Superposition: first-order, prepares for first-order