EECS 219C: Computer-Aided Verification

Boolean Satisfiability Solving Part II: DLL-based Solvers

Sanjit A. Seshia
EECS, UC Berkeley

With thanks to Lintao Zhang (MSR)

Announcements

- Paper readings will be up on the webpage by the weekend
  - Readings will be assigned by me based on your feedback on what interests you
- Suggested project topics will be announced next week
  - Welcome to pick your own, but talk to me first

A Classification of SAT Algorithms

- Davis-Putnam (DP)
  - Based on **resolution**
- Davis-Logemann-Loveland (DLL/DPLL)
  - Search-based
  - Basis for current most successful solvers
- Stalmarck's algorithm
  - "Different" kind of search, proprietary algorithm
- Stochastic search
  - Local search, hill climbing, etc.
  - Unable to prove unsatisfiability (incomplete)

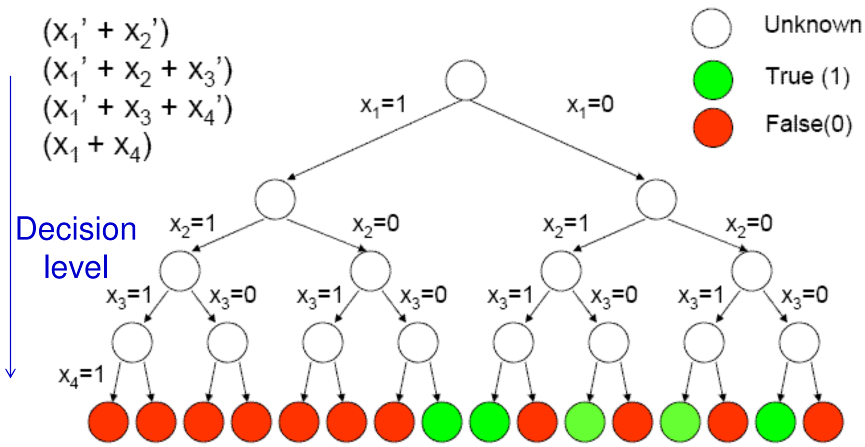
DLL Algorithm: General Ideas

- Iteratively set variables until you find a satisfying assignment or reach a conflict
- Two main rules:
  - *Unit Literal Rule*: If an unsatisfied clause has all but 1 literal set to 0, the remaining literal must be set to 1

    (a + b + c) (d' + e) (a + c' + d)
  - *Conflict Rule*: If all literals in a clause have been set to 0, the formula is unsatisfiable along the current assignment path

Search Tree

$(x_1' + x_2')$
$(x_1' + x_2 + x_3')$
$(x_1' + x_3 + x_4')$
$(x_1 + x_4)$



DLL Example 1

DLL Algorithm Pseudo-code

```
DLL_iterative()
{
    status = preprocess();
    if (status!=UNKNOWN)
        return status;
    while(1) {
        decide_next_branch();
        while (true)
        {
            status = deduce();
            if (status == CONFLICT)
            {
                blevel = analyze_conflict();
                if (blevel < 0)
                    return UNSATISFIABLE;
                else
                    backtrack(blevel);
            }
            else if (status == SATISFIABLE)
                return SATISFIABLE;
            else break;
        }
    }
}
```

Main Steps:

Pre-processing

Branching

Unit propagation (apply unit rule)

Conflict Analysis & Backtracking

Pre-processing: Pure Literal Rule

- If a variable appears in only one phase throughout the problem, then you can set the corresponding literal to 1
- Why?

DLL Algorithm Pseudo-code

```
DLL_iterative()
{
    status = preprocess();
    if (status!=UNKNOWN)
        return status;
    while(1) {
        decide_next_branch();
        while (true)
        {
            status = deduce();
            if (status == CONFLICT)
            {
                blevel = analyze_conflict();
                if (blevel < 0)
                    return UNSATISFIABLE;
                else
                    backtrack(blevel);
            }
            else if (status == SATISFIABLE)
                return SATISFIABLE;
            else break;
        }
    }
}
```

Main Steps:

Pre-processing

Branching

Unit propagation
(apply unit rule)

Conflict Analysis
& Backtracking

Conflicts & Backtracking

- Chronological Backtracking
  - Proposed in original DLL paper
  - Backtrack to highest decision level that has not been tried with both values
    - But does this decision level have to be the reason for the conflict?

Non-Chronological Backtracking

- Jump back to a decision level "higher" than the last one
- Also combined with "conflict-driven learning"
  - Keep track of the reason for the conflict
- Proposed by Marques-Silva and Sakallah in 1996
  - Similar work by Bayardo and Schrag in '97

DLL Example 2

DLL Algorithm Pseudo-code

```
DLL_iterative()
{
    status = preprocess();
    if (status!=UNKNOWN)
        return status;
    while(1) {
        decide_next_branch();
        while (true)
        {
            status = deduce();
            if (status == CONFLICT)
            {
                blevel = analyze_conflict();
                if (blevel < 0)
                    return UNSATISFIABLE;
                else
                    backtrack(blevel);
            }
            else if (status == SATISFIABLE)
                return SATISFIABLE;
            else break;
        }
    }
}
```

Main Steps:

Pre-processing

Branching

Unit propagation (apply unit rule)

Conflict Analysis & Backtracking

Branching

- Which variable (literal) to branch on (set)?
- This is determined by a “decision heuristic”
- What makes a “decision heuristic” good?

Decision Heuristic Desiderata

- If the problem is satisfiable
  - Find a short partial satisfying assignment
  - GREEDY: If setting a literal will satisfy many clauses, it might be a good choice
- If the problem is unsatisfiable
  - Reach conflicts quickly (rules out bigger chunks of the search space)
  - Similar to above: need to find a short partial falsifying assignment
- Also: Heuristic must be cheap to compute!

Sample Decision Heuristics

- RAND
  - Pick a literal to set at random
  - What's good about this? What's not?
- Dynamic Largest Individual Sum (DLIS)
  - Let cnt(l) = number of occurrences of literal l in unsatisfied clauses
  - Set the l with highest cnt(l)
  - What's good about this heuristic?
  - Any shortcomings?

DLIS: A Typical Old-Style Heuristic

- Advantages
  - Simple to state and intuitive
  - Targeted towards satisfying many clauses
  - Dynamic: Based on current search state
- Disadvantages
  - Very expensive!
  - Each time a literal is set, need to update counts for all other literals that appear in those clauses
  - Similar thing during backtracking (unsetting literals)
- Even though it is dynamic, it is “Markovian” – somewhat static
  - Is based on current state, without any knowledge of the search path to that state

VSIDS: The Chaff SAT solver heuristic

- Variable State Independent Decaying Sum
  - For each literal l, maintain a VSIDS score
  - Initially: set to cnt(l)
  - Increment score by 1 each time it appears in an added (conflict) clause
  - Divide all scores by a constant (2) periodically (every N backtracks)
- Advantages:
  - Cheap: Why?
  - Dynamic: Based on search history
  - Steers search towards variables that are common reasons for conflicts (and hence need to be set differently)

Current State of Heuristics

- VSIDS has been improved upon, but mostly minor improvements
- MiniSat (current champion) decays score after each conflict by a smaller fraction (5%)

S. A. Seshia 19

Key Ideas so Far

- Data structures: Implication graph
- Conflict Analysis: Learn (using cuts in implication graph) and use non-chronological backtracking
- Decision heuristic: must be dynamic, low overhead, quick to conflict/solution

- Principle: Keep #(memory accesses)/step low
  - A step → a primitive operation for SAT solving, such as a branch

S. A. Seshia 20

DLL Algorithm Pseudo-code

```
DLL_iterative()
{
    status = preprocess();
    if (status!=UNKNOWN)
        return status;
    while(1) {
        decide_next_branch();
        while (true)
        {
            status = deduce();
            if (status == CONFLICT)
            {
                blevel = analyze_conflict();
                if (blevel < 0)
                    return UNSATISFIABLE;
                else
                    backtrack(blevel);
            }
            else if (status == SATISFIABLE)
                return SATISFIABLE;
            else break;
        }
    }
}
```

Main Steps:

Pre-processing

Branching

Unit propagation (apply unit rule)

Conflict Analysis & Backtracking

Unit Propagation

- Also called Boolean constraint propagation (BCP)
- Set a literal and propagate its implications
  - Find all clauses that become unit clauses
  - Detect conflicts
- Backtracking is the reverse of BCP
  - Need to unset a literal and 'rollback'
- In practice: Most of solver time is spent in BCP
  - Must optimize!

BCP

- Suppose literal l is set. How much time will it take to propagate just that assignment?
- How do we check if a clause has become a unit clause?
- How do we know if there's a conflict?

- Introductory BCP slides

Detecting when a clause becomes unit

- Watch only two literals per clause. Why does this work?
- If one of the watched literals is assigned 0, what should we do?
- A clause has become unit if
  - Literal assigned 0 *must* continue to be watched, other watched literal unassigned
- What if other watched literal is 0?
- What if a watched literal is assigned 1?

- Lintao's BCP example

2-literal Watching

- In a L-literal clause, L $\geq$ 3, which 2 literals should we watch?

Next Class

- Finishing up SAT: incremental, proof gen.
- Start BDDs