Practical SAT Solving
Lecture 6
Carsten Sinz, Tomáš Balyo | May 29, 2018
Lecture Outline

- Stålmarck’s Method
- Advanced Techniques in DPLL
  - Restarts
  - Phase Saving
Stålmarck’s Method [1]

- **Input**: Arbitrary formula \( F \) in propositional logic (need not be in CNF, \( \Rightarrow \) and \( \Leftrightarrow \) also allowed)
- **Goal**: Show unsatisfiability of \( F \)
- **Preprocessing**: Decompose formula tree into simple equations (triplets) \( T \) and a literal equivalence class \( R \).
  \[ R \subseteq L^0 \times L^0 \text{ where } L^0 = L \cup \{0, 1\}, \text{ } R \text{ ‘consistent’} \]
- **Basic processing steps**: \( k \)-saturation (\( k = 0, 1, \ldots \))
  - 0-saturation: simplification with triplet rules
  - \( k \)-saturation (\( k \geq 1 \)): case distinction, breadth-first search
- Developed by Gunnar Stålmarck (~1989), patented
Decomposition into Triplets

\[ F = ((x \land y) \lor \neg y) \land (z \iff y) \]

Formula tree:

 Initial equival. class: \( \{ n_1 = 1 \} \)
(to show unsatisfiability of \( F \) by contradiction)
Stålmarck’s Method: 0-Saturation

Given set of triplets $T$ and literal equivalence class $R$ apply derivation rules (deriving new literal equivalences):

\[
\begin{align*}
\frac{p = q \land r}{r = 1} & \quad \frac{p = q \land r}{p = 0} \quad (A) \\
\frac{q = 1}{p = 1} & \quad \frac{q = 0}{p = 0} \\
\frac{p = q \land r}{q = 0} & \quad \frac{q = r}{p = q} \quad (B) \\
\frac{p = r}{p = 1} & \quad \frac{q = \neg r}{p = 0} \quad (C) \\
\frac{q = 1}{p = r} & \quad \frac{q = \neg r}{p = 0} \quad (D) \\
\end{align*}
\]
Stålmarck’s Method: $k$-Saturation

Given formula $F$, represented as $(T, R)$ (triplets and equiv. rel.)
procedure saturate extends equivalence relation $R$:

EquivRel saturate(int $k$, TripletSet $T$, EquivRel $R$) {
    if ($k = 0$) return zero-saturate($T$, $R$)
    forall $x \in \text{Var}(T)$ not fixed in $R$ do {
        $R_0 =$ saturate($k - 1$, $T$, $R \cup \{x = 0\}$)
        $R_1 =$ saturate($k - 1$, $T$, $R \cup \{x = 1\}$)
        $R = R_0 \cap R_1$
    }
    return $R$
}

(zero-saturate returns all-relation if inconsistency was found)
$k$-Saturation: Graphical Illustration

0-saturation (simplification rules)

1-saturation

2-saturation

merge relations → new $R$
Summary: Stålmarck’s Algorithm

**Input:** Formula $F$ represented as set of triplets $T$
(with $n_1$ representing top of formula tree)

**Output:** $F$ satisfiable?

```java
boolean stalmarckSAT(TripletSet T) {
    k = 0; R = \{n_1 = 1\}
    do {
        R = saturate(k, T, R)
        if (R = all-relation) return false
        else if (R satisfies all triplets T) return true
        else k = k + 1
    }
}
```
Restarts

- What is a restart?

Clear the partial assignment
Unassign all the variables
Backtrack to level 0

Why would anybody want to do restarts in DPLL?
To recover from bad branching decisions
You solve more instances
Might decrease performance on easy instances
What is a restart?
- Clear the partial assignment
- Unassign all the variables
- Backtrack to level 0
Restarts

- What is a restart?
  - Clear the partial assignment
  - Unassign all the variables
  - Backtrack to level 0
- Why would anybody want to do restarts in DPLL?
Restarts

- What is a restart?
  - Clear the partial assignment
  - Unassign all the variables
  - Backtrack to level 0

- Why would anybody want to do restarts in DPLL?
  - To recover from bad branching decisions
  - You solve more instances
  - Might decrease performance on easy instances
Restarts: Why?

Heavy-tail distribution
\[ P[X > x] \sim C \cdot x^{-\alpha} \]
(for \( 0 < \alpha < 2, \ C > 0 \))

Standard distribution
\[ P[X > x] \sim \frac{1}{x\sqrt{2\pi}}e^{-x^2/2} \]
(Figures from Gomes et al., 2000)
When to Restart?

- After a given number of decisions
- The number of decision between restarts should grow
  - To guarantee completeness
- How much increase?
  - Linear increase – too slow
  - Exponential increase – ok with small exponent
  - MiniSat: $k$-th restart happens after $100 \times 1.1^k$
**Inner/Outer Restart Scheduling**

**Inner/Outer Restart Algorithm**

```plaintext
int inner = 100
int outer = 100

forever do
  . . . do DPLL for inner conflicts . . .
  restarts++
  if inner >= outer then
    outer *= 1.1
    inner = 100
  else
    inner *= 1.1
```

Carsten Sinz, Tomáš Balyo – SAT Solving

Stålmarck's Method Advanced DPLL

May 29, 2018 13/18
Luby Sequence

\[ Luby = u \cdot (t_i)_{i \in \mathbb{N}} \]

\[ t_i = \begin{cases} 
2^{k-1} & \text{if } i = 2^k - 1 \\
2^{k-1} + 1 & \text{if } 2^{k-1} \leq i \leq 2^k - 1 
\end{cases} \]

1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots
### Luby Sequence Algorithm

```c
unsigned luby(unsigned i) {
    for (unsigned k = 1; k < 32; k++) {
        if (i == (1 << k) - 1) return 1 << (k - 1);
    }
    for (k = 1;; k++) {
        if ((1 << (k - 1)) <= i && i < (1 << k) - 1) return luby(i - (1 << (k-1)) + 1);
    }
    limit = 512 * luby(++restarts);
    ... // run SAT core loop for limit conflicts
}
```

- Complicated, not trivial to compute
Reluctant Doubling

- A more efficient implementation of the Luby sequence
- Use the $v_n$ of the following pair

\[
(u_1, v_1) = (1, 1) \tag{1}
\]

\[
(u_{n+1}, v_{n+1}) = u_n \& - u_n = v_n ? (u_n + 1, 1) : (u_n, 2v_n) \tag{2}
\]

- Example: $(1,1), (2,1), (2,2), (3,1), (4,1), (4,2), (4,4), (5,1), ...$
- Invented by Donald Knuth
Phase Saving

First implemented in RSAT (2006)
http://reasoning.cs.ucla.edu/rsat/

- Observation: Frequent *Restarts* decrease performance on some SAT instances
- Goal: Cache partial solutions to subsets of the formula and reuse them after a restart
- Idea: Remember last assignment of each variable and use it *first* in branching
- Result: *Phase Saving* stabilizes positive effect of restarts; best results in combination with *non-chronological backtracking* (later in this lecture)

Example: $A$ and $B$ are satisfied, search works on component $C$
URL http://dx.doi.org/10.1023/A:1008725524946