SAT-Solver

SAT-Solver ist Implementierung eines Algorithmus für das Erfüllbarkeitsproblem der Aussagenlogik (SAT)

Obwohl dies i.A. exponentielle (in $|Vars(\varphi)|$) Laufzeit braucht, gibt es mittlerweile einige SAT-Solver, die in der Praxis erstaunlich gut funktionieren.

- MINISAT `http://minisat.se/`
- PICOSAT `http://fmv.jku.at/picosat/`
- BERKMIN `http://eigold.tripod.com/BerkMin.html`
- RSAT `http://reasoning.cs.ucla.edu/rsat/`
- ZCHAFF `http://www.princeton.edu/~chaff/zchaff.html`
- ...

siehe auch SATLive-Webseite `http://www.satlive.org/`

Das DIMACS-Format

SAT-Solver verlangen typischerweise eine Eingabe in KNF.

Standardisiertes Format: DIMACS

- Variablen sind natürliche Zahlen ≥ 1
- Literale werden durch Integer bezeichnet, z.B. $A_7 = 7$, $\neg A_4 = -4$
- Klausel ist Liste von Integern, `0` markiert Klauselende
- KNF ist Liste von Klauseln
- Kommentare im Header (`c ...`)
- spezielle Headerzeile (`p cnf ...`) gibt Anzahl verwendeter Klauseln und Variablen an

Beispiel

Die KNF

$$(\neg A \vee B \vee C) \wedge (B \vee \neg C) \wedge \neg D \wedge (A \vee D) \wedge (\neg B \vee \neg C \vee \neg D)$$

kann im DIMACS-Format so repräsentiert werden:

```
c Beispielformel aus der Vorlesung
c Autor: Martin Lange
p cnf 4 5
-1 2 3 0
2 -3 0
-4 0
1 4 0
-2 -3 -4 0
```

SAT-Solver im Einsatz

Clevere Heuristiken und jahrelanges Tuning haben dazu geführt, dass moderne SAT-Solver typischerweise Instanzen der Größenordnung

- $\sim 10^5$ Variablen
- $\sim 10^6$ Klauseln

lösen können.

Vorsicht! Es gibt natürlich auch (im Vergleich dazu) sehr kleine Instanzen, an denen sie sich die Zähne ausbeissen.

typischer Einsatz von SAT-Solvern (nicht annähernd vollständig):

- Hardware-Verifikation
- Planungsprobleme in der KI
- Constraint-Solving
- ...

SAT4J

SAT4J ist in Java implementierter SAT-Solver; leicht als Library statt über DIMACS-Eingabe zu bedienen

`http://www.sat4j.org/`

Beispiel: $(N \leftrightarrow M) \wedge (M \leftrightarrow \neg B) \wedge (B \leftrightarrow (M \leftrightarrow N))$

1. Übersetzung in KNF
2. Kodierung der Variablen als `int`
3. Übergabe der KNF an SAT-Solver
4. lösen lassen, evtl. Modell dekodieren

Ersetzung von Literalen

Def.: Sei $\mathcal{C}$ Klauselmenge (= Menge von Mengen von Literalen). Mit $\mathcal{C}[A \mapsto 1]$ bezeichnen wir die Menge von Klauseln, die dadurch entsteht, dass man

1. jede Klausel, die das Literal A enthält, aus $\mathcal{C}$ entfernt, und
2. das Literal $\neg A$ aus jeder Klausel in $\mathcal{C}$ entfernt.

Für $\mathcal{C}[A \mapsto 0]$ gilt das entsprechend duale.

Bsp.:

$$\begin{aligned} \mathcal{C} &= \{\{A, \neg B\}, \{\neg A, \neg B\}, \{\neg A, B\}\} \\ \mathcal{C}[A \mapsto 1] &= \\ \mathcal{C}[B \mapsto 0] &= \end{aligned}$$

Lemma: Sei $\mathcal{C}$ Klauselmenge (als KNF aufgefasst), A Variable. $\mathcal{C}$ erfüllbar gdw. $\mathcal{C}[A \mapsto 1]$ oder $\mathcal{C}[A \mapsto 0]$ erfüllbar.

Beweis: (Übung)

Unit-Propagation

Lemma: Sei $\mathcal{C}$ Klauselmenge, A Variable, so dass $\{A\} \in \mathcal{C}$. Dann ist $\mathcal{C}$ erfüllbar gdw. $\mathcal{C}[A \mapsto 1]$ erfüllbar ist.

Beweis: "$\Leftarrow$" folgt sofort aus Lemma davor.
"$\Rightarrow$" Sei $\mathcal{C}$ erfüllbar. Wegen Lemma davor müssen wir lediglich zeigen, dass $\mathcal{C}[A \mapsto 0]$ unerfüllbar ist. Dies ist der Fall, denn da $\{A\} \in \mathcal{C}$ gilt $\emptyset \in \mathcal{C}[A \mapsto 0]$, und wegen KNF steht $\emptyset$ für $\mathtt{ff}$, und $\mathtt{ff} \wedge \varphi \equiv \mathtt{ff}$, was unerfüllbar ist. □

entsprechendes Lemma für Fall $\{\neg A\} \in \mathcal{C}$

$\rightsquigarrow$ Algorithmus Unit-Propagation($\mathcal{C}$) führt sukzessive diese Ersetzungsschritte durch, solange noch Singleton-Klauseln in $\mathcal{C}$ vorhanden sind.

Der DPLL-Algorithmus

Alle modernen SAT-Solver basieren auf dem DPLL-Algorithmus (nach Davis, Putnam, Logemann, Loveland).

DPLL($\mathcal{C}$) =
    Unit-Propagation($\mathcal{C}$)
    `if` $\mathcal{C} = \emptyset$ `then return` erfüllbar
    `if` $\emptyset \in \mathcal{C}$ `then return` unerfüllbar
    wähle Variable A, die noch in $\mathcal{C}$ vorkommt
    `if` DPLL($\mathcal{C}[A \mapsto 1]$) = erfüllbar `then return` erfüllbar
    `return` DPLL($\mathcal{C}[A \mapsto 0]$)

Bem.: Algorithmus DPLL terminiert immer, ist korrekt (wenn er "erfüllbar" sagt, dann war die Eingabe auch erfüllbar) und vollständig (wenn die Eingabe erfüllbar ist, dann sagt er auch "erfüllbar"), aber wieso?