Vorlesung
– Automatisches Beweisen –

Kap. 4.2.2: SAT-Solving mit dem DPLL-Verfahren

Prof. Dr. Wolfgang Küchlin

Dipl.-Inform., Dr. sc. techn. (ETH)

Arbeitsbereich Symbolisches Rechnen
Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften

Universität Tübingen

Steinbeis Transferzentrum
Objekt- und Internet-Technologien (OIT)

Wolfgang.Kuechlin@uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de

SR

DPLL-Algorithmus

- DPLL: Davis-Putnam-Logemann-Loveland
  - Erster Algorithmus um 1960/1962, stark verbessert ab 1996.
- Entscheidungsverfahren für *SAT-Solving* Problem
  - gegeben Formel *F*: gibt es eine erfüllende Belegung für *F*?
  - Idee: Probiere „intelligent“ eine erfüllende Belegung für F zu konstruieren, falls nötig backtracking.
- In aktueller Form das schnellste Verfahren für SAT
  - löst „gutartige“ SAT-Probleme mit Tausenden bis zu Millionen von Variablen
  - Beweise $\mathcal{F} \models F$ über $\mathcal{F} \wedge \neg F \models \bot$, also UnSAT($\mathcal{F} \wedge \neg F$).
  - Große Verbreitung zur Lösung industrieller (Verifikations-) Probleme, z.B. Hardware-, und Protokoll-Verifikation, KFZ-Konfiguration; Software-Verifikation im Kommen.

Historie: Der Davis-Putnam Algorithmus

- Martin Davis, Hilary Putnam: A Computing Procedure for Quantification Theory. *Journal of the ACM 7:201--215* (1960)
  - reduziert Unerfüllbarkeitsproblem für Prädikatenlogik auf Aussagenlogik und gibt dafür Algorithmus mit 3 Regeln an
- Regel 1 (One-Literal Clauses)

a) F ist inkonsistent falls es 2 Unit-Klauseln {p} und {¬p} enthält

b) Andernfalls, wenn F eine Unit-Klausel {p} enthält, dann streiche alle Klauseln, die p enthalten und streiche ¬p von allen Klauseln. Das Ergebnis F´ ist inkonsistent gdw. F inkonsistent.

c) Fall einer Unit-Klausel {¬p}, analog zu (b).

Falls F´ leer ist, dann ist F konsistent.

Historie: Der Davis-Putnam Algorithmus

- Regel 2 (Affirmative-Negative Rule)
  - Falls ein Atom p nur positiv (affirmative) oder nur negativ auftritt, dann lösche alle Klauseln in denen es auftritt.
  - Das Resultat F´ ist inkonsistent gdw. F inkonsistent
  - Falls F´ leer ist, dann ist F konsistent
- Regel 3 (Elimination of Atomic Formulas)
  - Falls ein Atom p sowohl positiv als auch negativ auftritt, dann forme F um zu $F = (A \vee p) \wedge (B \vee \neg p) \wedge R$, sodass p nicht in R vorkommt.
  - F ist inkonsistent gdw. $F' = (A \vee B) \wedge R$ ist inkonsistent
  - Beweis: F inkonsistent gdw. inkonsistent für p=0 *und* für p=1. $F = A \wedge R$ für p=0, und $F = B \wedge R$ für p=1, also ist F inkonsistent gdw. $F' = (A \wedge R) \vee (B \wedge R) = (A \vee B) \wedge R$ inkonsistent.

Historie: Der Davis-Putnam Algorithmus

- Der DP-Algorithmus hat 3 Regeln
  1. One-Literal (Unit Propagation – UP)
  2. Affirmative-Negative (Pure Literal – PL)
  3. Elimination of conflicts (Resolution)
- Beispiel
  - $S_0 = \{\{x, y, z\}, \{\neg x, y, z\}, \{\neg x\}, \{z, \neg y\}\}$
  - 1c (UP of ¬x): $S_1 = \{\{y, z\}, \{z, \neg y\}\}$
  - 3 (resolution on y): $S_2 = \{\{z\}, \{z\}\} = \{\{z\}\}$
  - 2 (PL of z}: $S_3 = \{\ \}$, also konsistent.
- Der DP-Algorithmus ist wegen Regel 3 ineffizient
  - Um das gewählte p zu eliminieren müssen ALLE Resolventen über p gebildet werden, Explosion neuer Klauseln

Historie: Der Davis-Logemann-Loveland Algorithmus

- Martin Davis, George Logemann, and Donald Loveland.

  A Machine Program for Theorem Proving. *Communications of the ACM 5:394—397* (1962).

- Regel 3* (Splitting Rule)
  - Falls p sowohl positiv als auch negativ auftritt, dann forme F um zu $F = (A \vee p) \wedge (B \vee \neg p) \wedge R$, sodass p nicht in R vorkommt.
  - F ist inkonsistent gdw. sowohl $(A \wedge R)$ als auch $(B \wedge R)$ inkonsistent sind
  - Beweis: F inkonsistent gdw. inkonsistent für p=0 *und* für p=1. $F= A \wedge R$ für p=0, und $F= B \wedge R$ für p=1.

- Regel 3* ersetzt Regel 3
  - Keine neuen Klauseln, Problem wird immer kleiner

Historie: Der Davis-Logemann-Loveland Algorithmus

The forms of Rule III are interchangeable; although theoretically they are equivalent, in actual applications each has certain desirable features. We used Rule III* because of the fact that Rule III can easily increase the number and the lengths of the clauses in the expression rather quickly after several applications. This is prohibitive in a computer if ones fast access storage is limited. Also, it was observed that after performing Rule III, many duplicated and thus redundant clauses were present. Some success was obtained by causing the machine to systematically eliminate the redundancy; but the problem of total length increasing rapidly still remained when more complicated problems were attempted. Also use of Rule III can seldom yield new one-literal clauses, whereas use of Rule III* often will.

In programming Rule III*, we used auxiliary tape storage. The rest of the testing for consistency is carried out using only fast access storage. When the "Splitting Rule" is used one of the two formulas resulting is placed on tape. Tape memory records are organized in the cafeterial stack-of-plates scheme: the last record written is the first to be read.

Grundlegender DPLL-Algorithmus

```
boolean DPLL(ClauseSet S){

    //1. Bereinige S (unit constraint propagation)
    while (S contains a unit clause {ℓ}) {
        delete from S clauses containing ℓ;      // unit-subsumption
        delete ¬ℓ from all clauses in S          // unit-resolution mit subsumption
    }

    //2. Trivialfall?
    if (□∈S)  return false;                      // constraint unerfüllbar
    if (S=={}) return true;                      // nichts mehr zu erfüllen

    //3. Fallunterscheidung und Rekursion
    choose a literal ℓ occurring in S;           // Heuristik (Intelligenz) gefragt!
    if( DPLL(S ∪ {ℓ}) ) return true;             // first recursive branch: try ℓ := true
    else if ( DPLL(S ∪ {¬ℓ}) ) return true;      // backtracking: try ℓ := false
    else return false;
}
```

Grundlegendes zum DPLL-Algorithmus

- Keine Deduktion neuer Klauseln
  - keine dynamische Speicher-Allokation
- Algorithmus „lebt“ von der Unit-Propagation
  - UP dominiert die Laufzeit in der Praxis (> 90% UP).
  - Das muss so sein:
    - bei 100 Variablen ohne UP 2^{100} Einzelfälle
    - mit kompletter UP nur 100 Propagationen
    - Praxis-typischer Wert: 90 Propagationen, 2^{10} Einzelfälle
- Fazit für Praxis-Probleme: nur wenige Entscheidungen sind essentiell, der Rest ergibt sich als zwingende Konsequenz.

Beispiel DPLL

- $S_0 = \{\{x, y, z\}, \{\neg x, y, z\}, \{\neg x\}, \{z, \neg y\}\}$
  - unit propagation mit $\neg x$
    - $\{\neg x\}$ subsumiert $\{\neg x, y, z\}$, also $\neg x \wedge (\neg x \vee y \vee z) \equiv \neg x$
    - $\{\neg x\}$ resolviert mit $\{x, y, z\}$ zu $\{y, z\}$, und $\{y, z\}$ subsumiert $\{x, y, z\}$
- $S_1 = \{\{y, z\}, \{z, \neg y\}\}$
  - Wähle z.B. y als Entscheidungsvariable:
  - Fall 1: setze y=1
  - $S_2 = \{\{y\}, \{y, z\}, \{z, \neg y\}\}$
  - unit propagation y ergibt $S_3 = \{z\}$,
  - unit propagation z ergibt $S_4 = \{\ \}$, return true.

DPLL-Algorithmus (3)

- Heuristikbeispiele für die Literalauswahl:
  - Wähle das Literal, das am häufigsten vorkommt.
    - dann schmilzt die Formel durch Evaluation an vielen Stellen
  - Wähle ein Literal L aus einer 2er-Klausel {K, ¬L}.
    - Dann K=1 falls L=1, bzw. L=0 falls K=0.
    - geschickte Wahl von K bzw. L führt unmittelbar zu einer Unit-Propagation
  - Wähle ein Literal aus einer kürzesten Klausel.
    - dann ergibt sich bald eine Unit-Klausel
  - Teste für jedes Literal L, wie sich S mittels der unit-propagation vereinfachen lässt. Wähle dasjenige L, für das S am einfachsten wird.
    - dann schmilzt die Formel vor der nächsten Fallunterscheidung insgesamt am schnellsten

Korrektheit des Davis-Putnam-Algorithmus

- $S|_L := \{C \setminus \{\neg L\} \mid C \in S, L \notin C\}$ entspricht der unit constraint propagation
  - unit Klausel {L} subsumiert alle Klauseln C mit $L \in C$.
  - unit Klausel {L} resolviert mit allen Klauseln $C = C_1 \cup \{\neg L\}$ zu C_1, und C_1 wiederum subsumiert C.
- Falls $\{L\} \in S$, so ist S erfüllbar gdw. $S|_L$ erfüllbar ist.
- S erfüllbar gdw. $S \cup \{\{L\}\}$ oder $S \cup \{\{\neg L\}\}$ erfüllbar ist, für ein beliebiges Literal L.
- Termination: Anzahl n der in S vorkommenden Variablen (vor Schritt 2) nimmt bei jedem rekursiven Aufruf ab, so dass letztendlich $\square \in S$ oder $S = \{\}$ gelten muss.

Lernen im DPLL-Algorithmus

- Falls nach einer Kette von Variablenbelegungen $x_i = b_i$, $b_i \in \{0, 1\}$ die Formel F=0 wird ($\beta(F) = 0$), dann haben wir eine Nullstelle N von F gefunden.
  - N ist gegeben durch den Term $N=(\bigwedge\{x_i\} \wedge\{\neg y_i\})$, $\beta(x_i)=1$, $\beta(y_i)=0$
- Die Negation ergibt eine Klausel C = ¬N in der CNF von F
  - C kann deshalb zu F hinzugefügt werden
- Eigenschaften von C
  - In C brauchen nur diejenigen Variablen aufzutauchen, die durch Entscheidungen belegt wurden (Entscheidungsvariablen)
  - die durch Propagation erzwungen belegten können wegbleiben
  - dadurch können sich kurze (=mächtige) Constraints ergeben
  - C kann auch durch Resolution hergeleitet werden, denn $F \wedge N \models \bot$

Beispiel DPLL mit Lernen

- $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$
  - Wähle z.B. x als Entscheidungsvariable:
  - Fall 1: setze x=0
  - S1 = $\{\{\neg x\}, \{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$
  - unit propagation $\neg x$
  - $S_2 = \{\{y\}, \{\neg y, z\}, \{\neg z\}\}$
  - unit propagation y
  - $S_3 = \{\{z\}, \{\neg z\}\}$
  - unit propagation z
  - $S_4 = \{\{\ \}\}$, return false
- Wir lernen, dass $S_0 = 0$ falls x=0, also C = {x}.
  - Resolutionsbeweis von C siehe 4.2.1

Beispiel DPLL mit Lernen

- Analyse des Beispiels $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$

x	y	z	S_0
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

- Aus den Variablenbelegungen im vorangehenden Lauf von DPLL lernen wir: es wurde die Nullstelle N=(x=0, y=1, z=1) getroffen
- Da hierbei nur x eine Entscheidungsvariable war, lernen wir weiter, dass N Teil eines Clusters aus 4 Nullstellen ist

- Zukünftige Treffer in den anderen Nullstellen des Clusters werden durch die Zusatzklausel C = {x} vermieden

Verbessertes Lernen im DPLL-Algorithmus

- Das Lernen aus Belegungen der Entscheidungsvariablen führt i.A. zu sehr langen (und wenig nützlichen) Klauseln
  - da i.A. schon sehr viele Entscheidungen getroffen wurden
- Verbesserung: Lerne ausgehend von Konflikt-Klausel K
  - K schlägt fehl (Fehler-Klausel), d.h. β(K)=false, immer wegen einer Unit-Propagation.
  - Ausgehend von einer Entscheidung wurde eine Klausel R (sog. „reason“-Klausel) zur Unit und somit ein Literal L=1 erzwungen
  - Dadurch wurde in K ein Literal L´=false.
  - Also sind L und L´ komplementäre Literale
  - Also haben R und F eine Resolvente ohne dieses Literal
- Nun iteriere diesen Prozess …

Verbessertes Lernen im DPLL-Algorithmus

- Angenommen, K enthält die letzte Entscheidungsvariable x sowie einige Literale, die als Folge der letzten Entscheidung durch UP gesetzt wurden.
- Ausgehend von K
  - ersetze in F sukzessive durch Resolution alle UP(x)-Literale.
  - Die so gelernte Klausel K´ enthält nur noch
    - die letzte Entscheidungsvariable x
    - eine Teilmenge E der vorhergehenden Entscheidungsvariablen
    - durch UP als Folge der Entscheidungen aus E gesetzte Literale
    - andere Literale kann es nicht geben, denn diese wären unbelegt.
- Momentan ist $\beta(K´)$=false

Verbessertes Lernen im DPLL-Algorithmus

- Nun hebt man in β alle Belegungen der letzten Entscheidungsebene auf
  - also die Entscheidung x und die Belegung aller UP(x)-Variablen
- Durch Aufheben der Belegungen wird K´ eine unit
  - Durch die unit K´ wird die Alternativbelegung von x erzwungen
  - ersetzt das backtracking
  - Bem.: falls x nicht in K vorkommt, lässt man eine UP(x)-Variable u in K´ übrig, und diese übernimmt die Rolle von x.
- Die so gelernten Klauseln sind viel kürzer und mächtiger
- Das verbesserte Lernen war der große Durchbruch für DPLL i.d. Praxis

Beispiel DPLL mit verbessertem Lernen

- $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$
  - Wähle z.B. x als Entscheidungsvariable:
  - Fall 1: setze x=0
  - S1 = {{¬x}, {x, y}, {¬y, z}, {¬z, x}}
  - unit propagation ¬x (Reason für ¬x ist Entscheidung x=0)
  - $S_2 = \{\{y\}, \{\neg y, z\}, \{\neg z\}\}$
  - unit propagation y (Reason für y ist R_1={x, y}).
  - $S_3 = \{\{z\}, \{\neg z\}\}$.
  - unit propagation z (Reason für z ist R_2={¬y, z}))
  - $S_4 = \{\{\ \}\}$, return false (Fehler-Klausel ist K={¬z, x})
- Es ist $\beta(\{\neg z, x\}) = 0$. Resolution mit $R_2 = \{\neg y, z\}$ ergibt {¬y, x}, weitere Resolution mit $R_1 = \{x, y\}$ ergibt K´={x}.

Beispiel DPLL mit verbessertem Lernen

- Analyse des Beispiels $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$

x	y	z	S_0
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

- Aus der Fehlerklausel $K=\{\neg z, x\}$ im Lauf von DPLL lernen wir: es wurde der Nullstellen-*Cluster* N=(x=0, z=1) getroffen
- Nach Res. von K mit $R_2=\{\neg y, z\}$ lernen wir $\{\neg y, x\}$, d.h. dass N´= (x=0, y=1) ein ``benachbartes´´ Nullstellencluster ist.
- Nach Res. mit $R_1=\{x, y\}$ lernen wir {x} mit Cluster N´´= (x=0).

- Zukünftige Treffer in den anderen Nullstellen des Clusters werden durch die Zusatzklausel K´ = {x} vermieden