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

DPLL-Algorithmus

- DPLL: Davis-Putnam-Logemann-Loveland
  - Erster Algorithmus um 1965
- Entscheidungsverfahren für *SAT-Solving* Problem
  - gegeben Formel *F*: gibt es eine erfüllende Belegung für *F*?
  - Idee: Probiere „intelligent“ 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.

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 := true
    else return false;
}
```

Beispiel DPLL

- $S_0 = \{\{x, y, z\}, \{\neg x, y, z\}, \{\neg x\}, \{z, \neg y\}\}$
  - unit propagation mit ¬x:
- $S_1 = \{\{y, z\}, \{z, \neg y\}\}$
  - Wähle z.B. y als Entscheidungsvariable:
  - Fall 1: setze y=1
  - $S_2 = \{\{y\}, \{y, z\}, \{z, \neg y\}\}$
  - unit propagation y ergibt $S_3 = \{z\}$, $S_4 = \{\ \}$, 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.
  - 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 constraint propagation
- 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 $\in S$ 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 (i.Z. $F \models_\beta \bot$), dann haben wir einen gültigen Constraint C gefunden: $C= \neg\bigwedge\{b_i\}$
  - denn an der Stelle β wird F=0
  - C kann zur CNF hinzugefügt werden
- Eigenschaften von C
  - In C brauchen nur diejenigen Variablen aufzutauchen, die durch freie Entscheidungen belegt wurden (Entscheidungsvariablen)
  - die durch Propagation belegten können wegbleiben
  - dadurch können sich kurze (=mächtige) Constraints ergeben
  - C kann auch durch Resolution hergeleitet werden

Lernen im DPLL-Algorithmus

- Nutzen von C
  - Falls in C eine Variable x *nicht* vorkommt, so wird F=0 für Belegung β und jeden Wert von x.
  - Zufügen von C verhindert, dass der zweite Wert von x auch noch ausprobiert wird.
  - Wiederholte Bearbeitung von Teilen des Suchbaums wird vermieden, die aufgrund derselben Ursache keine Lösung enthalten.
  - Backtracking kann zum Teil unterbleiben.
  - Neue Klauseln werden generiert (→ zusätzliches Wissen über die Probleminstanz)
- Lernen ist der große Durchbruch für DPLL i.d. Praxis

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 = {{¬x}, {x, y}, {¬y, z}, {¬z, x}}
  - unit propagation ¬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 $S_0 = 0$ falls x=0, also C = {x}.
  - Resolutionsbeweis von C siehe 4.2.1