# *Vorlesung*
# *– Automatisches Beweisen –*

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


Prof. Dr. Wolfgang Küchlin

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

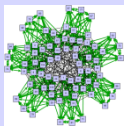
**Arbeitsbereich Symbolisches Rechnen**
**Wilhelm-Schickard-Institut für Informatik**
**Fakultät für Informations- und Kognitionswissenschaften**

**Universität Tübingen**

**Steinbeis Transferzentrum**
**Objekt- und Internet-Technologien (OIT)**

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

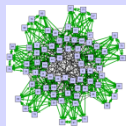

SR



# *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“ 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

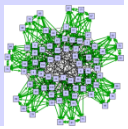
```
boolean DPLL(ClauseSet S){

    //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;
}
```

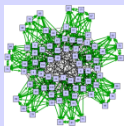




# *Grundlegendes zum DPLL-Algorithmus*

- 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.

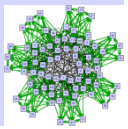




# *Beispiel DPLL*

- $S_0 = \{\{x, y, z\}, \{\neg x, y, z\}, \{\neg x\}, \{z, \neg y\}\}$
  - unit propagation mit ¬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.

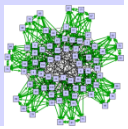




# *DPLL-Algorithmus (3)*

- 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

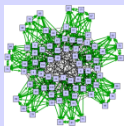




# *Korrektheit des Davis-Putnam-Algorithmus*

- $S|_L := \{C \setminus \{\neg L\} \mid C \in S, L \notin C\}$ entspricht der 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.

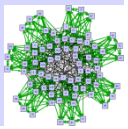




# *Lernen im DPLL-Algorithmus*

- Falls nach einer Kette von Variablenbelegungen $x_i = b_i$, $b_i \in \{0, 1\}$ die Formel F=0 wird (b(F) =0), dann haben wir eine Nullstelle von F gefunden
- Wie üblich ergibt die Negation der Nullstelle eine Klausel C in der CNF von F : $C = \neg(\wedge\{x_i\} \wedge\{\neg y_i\})$, $b(x_i)=1$, $b(y_i)=0$
  - 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 $F \models C$

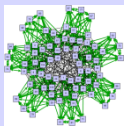




# *Lernen im DPLL-Algorithmus*

- Sei b eine Variablenbelegung, die x belegt und y nicht belegt. Wir schreiben b \ {x} für die Restriktion von b und b ∪ {y=1} für die Erweiterung von b.
- Nutzen von C
  - Sei C mit Literal x, und F mit b(F)=0 gegeben; sei b´= b\{x}.
  - Es ist b(C)=0, aber unter b´ ist C die Unit-Klausel {x}.
  - Falls in C eine Variable y *nicht* vorkommt, so wird b(F)=0 für Belegung b und *jeden* Wert von y.
  - In b´(F∪C) ist x keine Entscheidungsvariable mehr, ebensowenig mit b´∪{y=1} und mit b´∪{y=0}
  - Tendenziell wird durch C eine Entscheidungsvariable durch eine UP Variable ersetzt (falls eine Belegung b´ wieder auftaucht).

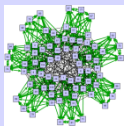




# *Lernen im DPLL-Algorithmus*

- Nutzen von C
  - Falls (z.B. nach Neustart des Verfahrens mit anderer Variablenreihenfolge) b´ als Teil von einem b´´= b´∪{y=1} oder b´´= b´∪{y=0} wieder vorkommt, so wird die Belegung von x durch die Unit-Klausel b´(C) = b´´(C) erzwungen.
  - 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)





# *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