Technische Universität Dresden

Theoretische Informatik und Logik

5. Vorlesung: DPLL und SMT

Markus Krötzsch

Professur Wissensbasierte Systeme

TU Dresden, 30. April 2026

Wiederholung: Logisches Schließen

Allgemeine Fragestellungen des logischen Schließens

- Logische Folgerung: Gilt $\mathcal{F} \models G$?
- Allgemeingültigkeit: Gilt $\models F$?
- Unerfüllbarkeit: Gilt $\models \neg F$, d.h. $F \models \bot$?

Einige einfache Methoden des logischen Schließens

- Wahrheitswertetabelle
- Erfüllbarkeitstest mit DNF
- Widerlegbarkeitstest mit KNF
- Erfüllbarkeitstest mit Resolution

$\rightsquigarrow$ Alle schlimmstenfalls exponentiell und oft zu ineffizient

Die DPLL-Methode

Ein effizienteres Verfahren

Idee: Implementiere den Erfüllbarkeitstest als Suche nach einem Modell

- Teste systematisch Belegungen für Atome
- Bilde Ableitungen um die Konsequenzen einer Belegung zu ermitteln
- Erkenne Widersprüche und verwende Backtracking um andere Belegungen zu testen

DPLL arbeitet mit Formeln in Klauselform (muss vorab gebildet werden)

Wir testen Erfüllbarkeit $\rightsquigarrow$ „kleine“ (polynomielle) Klauselform mit Hilfsatomen zulässig

Hintergrund: DPLL steht für die Namen Davis, Putnam, Logemann, Loveland

- Davis & Putnam hatten einen resolutionsbasierten Vorgängeralgorithmus entwickelt [1960]
- Davis, Logemann & Loveland entwickelten daraus die hier vorgestellte Version [1961]
- Weitere Verbesserungen wurden erst viel später entwickelt, aber Grundideen bis heute in Solvern relevant

Rückblick: Vereinfachung von $\top$ und $\bot$

Mit den Äquivalenzen aus Vorlesung 2 erhalten wir folgende Vereinfachungen (anwendbar auf beliebige Teilformeln):

$$\top \wedge F \equiv F \wedge \top \equiv F \qquad F \to \top \equiv \top$$
$$\top \vee F \equiv F \vee \top \equiv \top \qquad \top \to F \equiv F$$
$$\bot \wedge F \equiv F \wedge \bot \equiv \bot \qquad F \to \bot \equiv \neg F$$
$$\bot \vee F \equiv F \vee \bot \equiv F \qquad \bot \to F \equiv \top$$
$$\neg\top \equiv \bot \qquad \top \leftrightarrow F \equiv F \leftrightarrow \top \equiv F$$
$$\neg\bot \equiv \top \qquad \bot \leftrightarrow F \equiv F \leftrightarrow \bot \equiv \neg F$$

Zusammen mit den weiteren Äquivalenzen aus Vorlesung 2 kann jede Formel soweit vereinfacht werden, dass sie entweder $\top$ oder $\bot$ ist, oder keine Vorkommen von $\top$ oder $\bot$ enthält.

Splitting

Idee: Weise einem Atom p einen Wert zu und ermittle (rekursiv) ob die Formel damit erfüllbar ist

Daraus ergibt sich auch schon für beliebigen Formeln ein möglicher Algorithmus:

SplitSat
Eingabe: Formel F (in beliebiger Form)
Ausgabe: „erfüllbar“ oder „unerfüllbar“

(1) Wende die Äquivalenzen zur Vereinfachung von $\top$ und $\bot$ erschöpfend auf F an
(2) Wenn $F = \bot$: gib „unerfüllbar“ aus
(3) Wenn $F = \top$: gib „erfüllbar“ aus
(4) Andernfalls wähle ein Atom p in F:
   - Wenn **SplitSat**($F[p \mapsto \top]$)=„erfüllbar“, dann gib „erfüllbar“ aus
   - Andernfalls gib **SplitSat**($F[p \mapsto \bot]$) aus

Anmerkung: $F[p \mapsto \top]$ ist die Formel, die aus F durch Ersetzung aller Vorkommen von p durch $\top$ entsteht; analog für $F[p \mapsto \bot]$

Splitting: Beispiel

$$(A \leftrightarrow \neg B) \wedge (B \leftrightarrow \neg C) \wedge (C \leftrightarrow (\neg A \wedge \neg B))$$

$A \mapsto \top$: $(\top \leftrightarrow \neg B) \wedge (B \leftrightarrow \neg C) \wedge (C \leftrightarrow (\neg\top \wedge \neg B))$

$\equiv \neg B \wedge (B \leftrightarrow \neg C) \wedge \neg C$

- $B \mapsto \top$: $\neg\top \wedge (\top \leftrightarrow \neg C) \wedge \neg C \equiv \bot$
- $B \mapsto \bot$: $\neg\bot \wedge (\bot \leftrightarrow \neg C) \wedge \neg C \equiv C \wedge \neg C$
  - $C \mapsto \top$: $\top \wedge \neg\top \equiv \bot$
  - $C \mapsto \bot$: $\bot \wedge \neg\bot \equiv \bot$

$A \mapsto \bot$: $(\bot \leftrightarrow \neg B) \wedge (B \leftrightarrow \neg C) \wedge (C \leftrightarrow (\neg\bot \wedge \neg B))$

$\equiv B \wedge (B \leftrightarrow \neg C) \wedge (C \leftrightarrow \neg B)$

- $B \mapsto \top$: $\top \wedge (\top \leftrightarrow \neg C) \wedge (C \leftrightarrow \neg\top) \equiv \neg C$
  - $C \mapsto \top$: $\neg\top \equiv \bot$
  - $C \mapsto \bot$: $\neg\bot \equiv \top$

Splitting: Beobachtungen

SplitSat auf allgemeinen Formeln:

- rekursive Implementierung der systematischen Betrachtung aller Belegungen
  $\leadsto$ vollständig, korrekt, möglicherweise ineffizient
- Reihenfolge der Belegung der Atome kann Performance beeinflussen
- Backtracking in Rekursion eingebaut; kann als Suchbaum illustriert/vorgestellt werden

Splitting in DPLL:

- Bei Formeln in Klauselform genügt eine einzige Form der Vereinfachung: Unit propagation
- Weitere sinnvolle Optimierung: Pure literal elimination

Unit Propagation

Idee: Klauseln mit genau einem Literal („Unit Clauses“) legen den Wahrheitswert ihres Atoms fest

- Wir können diesen Wert immer sofort festlegen
- Alle anderen Klauseln können entsprechend vereinfacht werden ($\rightsquigarrow$ Resolution)

Für ein Atom A definieren wir $\overline{A} = \neg A$ und $\overline{\neg A} = A$.

Unit Propagation
Eingabe: Formel $\mathcal{F}$ in Klauselform
Ausgabe: Vereinfachte $\mathcal{F}$ ohne einelementige Klauseln

Solange es eine Klausel $\{L\} \in \mathcal{F}$ gibt:

(1) Lösche alle Klauseln $K \in \mathcal{F}$ mit $L \in K$.

(2) Entferne $\overline{L}$ aus allen Klauseln in $K \in \mathcal{F}$ mit $\overline{L} \in K$.

Pure Literal Elimination

Idee: Atome, die entweder nur nicht-negiert oder nur negiert vorkommen („Pure Literals“) sind ohne Probleme erfüllbar

- Wir können den Wert dieser Atome immer sofort festlegen
- Alle anderen Klauseln können entsprechend vereinfacht werden ($\rightsquigarrow$ Resolution)

Pure Literal Elimination
Eingabe: Formel $\mathcal{F}$ in Klauselform
Ausgabe: Vereinfachte $\mathcal{F}$ ohne Pure Literals

Solange in $\mathcal{F}$ ein Pure Literal L vorkommt:

(1) Lösche alle Klauseln $K \in \mathcal{F}$ mit $L \in K$.

Äquivalent: „Füge Klausel $\{L\}$ ein und wende Unit Propagation an.“

DPLL: Vollständiger Algorithmus

DPLLSat
Eingabe: Formel $\mathcal{F}$ in Klauselform
Ausgabe: „erfüllbar“ oder „unerfüllbar“

(1) Wende Unit Propagation auf $\mathcal{F}$ an
(2) Wende Pure Literal Elimination auf $\mathcal{F}$ an
(3) Wenn $\mathcal{F} = \emptyset$: gib „erfüllbar“ aus
(4) Wenn $\mathcal{F}$ die leere Klausel enthält: gib „unerfüllbar“ aus
(5) Andernfalls wähle ein Atom p in $\mathcal{F}$:
   - Wenn **DPLLSat**$(\mathcal{F} \cup \{\{p\}\})$=„erfüllbar“, dann gib „erfüllbar“ aus
   - Andernfalls gib **DPLLSat**$(\mathcal{F} \cup \{\{\neg p\}\})$ aus

Anmerkung: bei Ausgabe „erfüllbar“ ergeben die Werte der nacheinander in Unit Propagation und Pure Literal Elimination eliminierten Atome ein Modell

DPLL: Beispiel

$$\{\{\neg A, \neg B\}, \{A, B\}, \{\neg B, \neg C\}, \{B, C\}, \{\neg C, \neg A\}, \{A, B, C\}\}$$

$A \mapsto \top$

$$\{\{A\}, \{\neg A, \neg B\}, \{A, B\}, \{\neg B, \neg C\}, \{B, C\}, \{\neg C, \neg A\}, \{A, B, C\}\}$$

$$\equiv \{\{\neg B\}, \{\neg B, \neg C\}, \{B, C\}, \{\neg C\}\}$$

$$\equiv \{\{C\}, \{\neg C\}\}$$

$$\equiv \{\{\}\} \equiv \{\bot\} \equiv \bot$$

$A \mapsto \bot$

$$\{\{\neg A\}, \{\neg A, \neg B\}, \{A, B\}, \{\neg B, \neg C\}, \{B, C\}, \{\neg C, \neg A\}, \{A, B, C\}\}$$

$$\equiv \{\{B\}, \{\neg B, \neg C\}, \{B, C\}\}$$

$$\equiv \{\{\neg C\}\}$$

$$\equiv \{\} \equiv \top$$

DPLL terminiert

Satz: Der Algorithmus DPLLSat terminiert für jede Eingabe.

Beweis: Die Zahl der Atome in $\mathcal{F}$ verringert sich in jedem Durchlauf von Unit Propagation oder Pure Literal Elimination.
$\rightsquigarrow$ die Schleifen dieser Funktionen terminieren

Rekursive Aufrufe sind nur unter Hinzunahme von Unit Clauses möglich
$\rightsquigarrow$ mit jeder Rekursion sinkt die Zahl der Atome mindestens um 1
$\rightsquigarrow$ die maximale Rekursionstiefe ist linear □

Aus der bestimmten maximalen Rekursionstiefe folgt:
DPLLSat führt zu maximal 2^k rekursiven Aufrufen (k = Zahl der Atome in $\mathcal{F}$).

DPLL ist korrekt

Satz: Das Ergebnis von DPLLSat($\mathcal{F}$) ist korrekt.

Beweis: Die Vereinfachungen produzieren äquivalente Formeln (unter den gewählten Belegungen), weil das Löschen von Klauseln oder Literalen Sonderformen der bereits bekannten Äquivalenumformungen sind.

Korrektheit folgt durch Induktion über Zahl k der Atome in $\mathcal{F}$ den Vereinfachungen:

- Für $k = 0$ Atome ist die Klauselform $\{\}$ oder $\{\{\}\}$. In beiden Fällen ist das Ergebnis korrekt.
- Bei $k > 0$ Atomen führt Splitting zu rekursiven Aufrufen für $\mathcal{F} \cup \{p\}$ und $\mathcal{F} \cup \{\neg p\}$.In beiden Fällen wird sich die Zahl der Atome durch Unit Propagation um mindestens eins verringern.Per Induktionsannahme ist das Ergebnis der rekursiven Aufrufe also korrekt. Korrektheit für k folgt, weil $\mathcal{F}$ genau dann erfüllbar ist, wenn $\mathcal{F} \cup \{p\}$ oder $\mathcal{F} \cup \{\neg p\}$ erfüllbar sind. □

DPLL in der Praxis

DPLL ist bereits ein praktikables Verfahren

- oft wenige Splittings nötig, speziell bei (leicht) erfüllbaren Formeln
- Eingaben mit vielen Horn-Formeln fast komplett mit Unit Propagation lösbar
- insgesamt viel effizienter als Resolution

Moderne Solver verwenden weitere Verbesserungen:

- heuristische Strategien zur Wahl der Atome für das Splitting
- Backjumping: Rückgängigmachung vieler Wertzuweisungen in einem Schritt, überspringt „irrelevante“ Entscheidungen im Backtracking
- Conflict-driven clause learning (CDCL): Erkenntnisse aus Fehlschlägen in der Suche werden in zusätzlichen Klauseln „abgespeichert“ um die weitere Suche abzukürzen

SMT-Solving

Satisfiability Modulo Theories

Aussagenlogisches Problem:

Gibt es Werte für p, q, r, s, t aus der Menge {1,0}, so dass gilt
$(p \vee r) \wedge (q \vee s) \wedge (p \vee q) \wedge \neg t$?

Mathematisches Problem mit aussagenlogischen Operatoren:

Gibt es Werte für x, y, z aus der Menge der natürlichen Zahlen, so dass gilt
$(x < y \vee x = y) \wedge (y < z \vee y = z) \wedge (x < y \vee y < z) \wedge \neg(x < z)$?

Satisfiability Modulo Theories (SMT):

- Verallgemeinerung von Aussagenlogik mit Atomen, die sich auf andere (nicht-boolesche) Domänen beziehen
- typische Datentypen: ganze Zahlen, reelle Zahlen, Listen, Strings, Enums usw.
- Operationen je nach Typ: Arithmetik, Stringoperationen, Gleicheiten und Ungleichheiten usw.

Ziel: Ermittle Erfüllbarkeit und finde gegebenenfalls ein Modell

Beispiel: Z3 als SMT-Solver

Die Erfüllbarkeit von $(x < y \vee x = y) \wedge (y < z \vee y = z) \wedge (x < y \vee y < z) \wedge \neg(x < z)$ kann in Z3 wie folgt getestet werden:

```
from z3 import *

x,y,z = Ints('x y z')

s = Solver()
s.add(And( Or(x<y, x==y), Or(y<z, y==z), Or(x<y, y<z), Not(x<z)))

print(s.check())
```

SMT und DPLL

Ein wichtiger Ansatz zum Lösen von SMT-Problemen
(am Beispiel $x \leq y \wedge (x > y \vee x = y)$)

- Abstrahiere die Eingabe in eine aussagenlogische Formel
  Beispiel: $x \leq y \wedge (x > y \vee x = y)$ wird zu $p \wedge (q \vee r)$
- Suche mit DPLL nach erfüllenden Belegungen
  Beispiel: $p \mapsto 1, q \mapsto 1, r \mapsto 0$
- Übersetze (partielle) Belegungen zurück in die Ausgangssprache und prüfe dort ihre Erfüllbarkeit $\leadsto$ benötigt spezifische Verfahren
  Beispiel: $x \leq y, x > y, x \neq y$ unerfüllbar
- Falls erfüllbar: gib Modell aus
- Falls unerfüllbar: lerne zusätzliche logische Bedingungen
  Beispiel: Formel wird um Klausel $(\neg p \vee \neg q)$ erweitert

Das T in SMT steht für Theory

Theory bezeichnet die logisch-mathematische Theorie für einen Datentyp und eine Menge unterstützter Operationen

Beispiele:

- Reelle Zahlen mit $*$, $+$, $-$, $/$
- Ganze Zahlen mit lineraren Gleichungssystemen
- Strings mit Verkettungsoperationen und lexikographischer Ordnung
- IEEE Gleitkommazahlen mit Gleitkomma-Arithmetik
- ...

Das Schließen in einer Theorie kann sehr kompliziert sein

- Jede Theorie verlangt eigene Methoden
- Viele Theorien sind unentscheidbar $\leadsto$ unvollständige Erfüllbarkeitstests möglich, aber keine Erfolgsgarantie
- Die Vermischung von Theorien in einer Formel ist nicht trivial

SMT-Solving: Ausblick

Die „erweiterten aussagenlogischen Formeln“ in SMT gehören schon in den Bereich der Prädikatenlogik $\rightsquigarrow$ später mehr dazu

„DPLL mit Theorien“ ist bei weitem nicht die einzige SMT-Methode, aber eine der wichtigsten

SMT-Solver wie Z3 benötigen weitere Techniken, unter anderem

- Normalisierung und Modifikation von logischen Ausdrücken
- Vereinfachungen und Äquivalenzumformungen für Ausdrücke aller Theorien

Z3 (und andere SMT-Solver) in der Praxis:

- vielfältig anwendbar, auch ohne detaillierte Kenntnisse
- oft viele verschiedene Problemkodierungen möglich
- Performance in der Regel nicht theoretisch vorhersagbar (aber Hintergrundkenntnisse und Erfahrung helfen)

Logisches Schließen als Wortproblem

Schließen als Entscheidungsproblem

Wir wissen bereits, dass Schließen ein Entscheidungsproblem ist, z.B.:

Erfüllbarkeit:

- **Eingabe:** Formel F
- **Ausgabe:** „ja“ wenn F erfüllbar ist; sonst „nein.“

Wie wir wissen, können solche Probleme als Definition einer Sprache angesehen werden:

$$\textbf{SAT} = \{\text{enc}(F) \mid F \text{ ist erfüllbar}\}$$

wobei $\text{enc}(F)$ eine (vernünftige) Kodierung der Formel F als Wort über einem endlichen Alphabet (z.B. $\{0, 1\}$) ist. Vor allem müssen wir beliebig viele Atome mit nur endlich vielen Zeichen kodieren.

Was für eine Sprache ist **SAT**?
Durch welches Automatenmodell wird sie erkannt?

Schließen als Sprache (1)

Was für eine Sprache ist **SAT**?
Durch welches Automatenmodell wird sie erkannt?

Wir wissen, dass **SAT** entscheidbar ist und daher von Typ 0 ist.
Der folgende naive Algorithmus entscheidet **SAT** auf einer TM:

Naiver Erfüllbarkeitstest (Skizze):

- Wir prüfen systematisch jede Wertzuweisung auf den Atomen der gegebenen Formel
- Dazu iteriert die TM über alle Wertzuweisungen und berechnet für jede rekursiv den Wahrheitswert der Formel
- Falls eine erfüllende Zuweisung gefunden wird, dann hält die TM und akzeptiert
- Falls alle Wertzuweisungen ohne Erfolg durchlaufen worden sind, dann hält die TM und verwirft

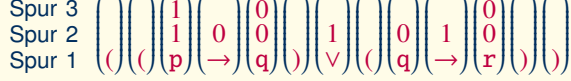
Schließen als Sprache (2)

Was für eine Sprache ist **SAT**?
Durch welches Automatenmodell wird sie erkannt?

Der naive Erfüllbarkeitstest kann in linearem Speicher ablaufen, d.h. auf einem LBA $\rightsquigarrow$
SAT ist von Typ 1

Naive Erfüllbarkeit auf LBA (Skizze):

- Wir verwenden ein Arbeitsalphabet mit mehreren „Spuren“: (1) Eingabe, (2) aktuell ermittelte Wahrheitswerte aller Teilformeln, (3) aktuell getestete Wertzuweisung
- Skizze der Kodierung (ohne Binärkodierung der Formel):



- Man kann nun systematisch Spur 3 iterieren, Spur 2 rekursiv berechnen und den Wert der Gesamtformel prüfen (machbar, wir verzichten auf die Details)

Welches Berechnungsmodell passt zu **SAT**?

Erkenntnisse bisher:

- LBAs sind stark genug für **SAT** (d.h. **SAT** ist von Typ 1)
- **SAT** ist keine reguläre Sprache, da FAs korrekte Klammerung nicht erkennen können (d.h. **SAT** ist nicht von Typ 3)

Man kann außerdem zeigen:

SAT ist nicht kontextfrei.

(zumindest wenn eine offensichtliche Kodierung verwendet wird; siehe `http://cstheory.stackexchange.com/questions/37322/is-sat-a-context-free-language`)

Können wir **SAT** noch genauer charakterisieren als mit LBAs?

$\rightsquigarrow$ Ja, mithilfe anderer Arten von eingeschränkten TMs.

Zusammenfassung und Ausblick

DPLL ist ein effizientes Verfahren zur Ermittlung der Erfüllbarkeit aussagenlogischer Formeln, basierend auf systematischer Modellsuche

Wichtige darin vorkommende Methoden sind Backtracking (und Splitting), Unit Propagation und die Eliminierung von Pure Literals

DPLL kann auf nicht-boolesche Domänen erweitert werden, was wichtig für SMT-Solving ist, aber Resolution ist besser auf ausdrucksstarke Logiken übertragbar (später mehr dazu)

Offene Fragen:

- Welches Sprach- oder Berechnungsmodell passt wirklich zu Aussagenlogik?
- Unsere Ansätze zum logischen Schließen sind alle schlimmstenfalls exponentiell – muss das sein?