Automatisches Beweisen—Vertiefung
Wiederholung Aussagenlogik

Christoph Zengler

Arbeitsbereich Symbolisches Rechnen
Prof. Dr. Wolfgang Küchlin
Universität Tübingen

16. Oktober 2012

Logischer Formalismus

Syntax

Wie werden Formeln gebildet?

- klassisch-mathematisch: bestimmte ausgezeichnete Zeichenreihen
- informatisch: Sprache einer Grammatik

Semantik

Was ist die Bedeutung einer Formel?

- Allgemein: Abbildung in einen (bekannten) Semantik-Bereich

Kalkül

Wie kann der Wahrheitswert einer Formel bestimmt werden?

- Inferenzregeln, Algorithmen

Syntax

EBNF-Grammatik für Formeln

Formel	=	$\top \mid \bot$	Konstanten
	$\mid$	*Variable*	$\in \mathcal{V}$
	$\mid$	$\neg$ Formel	Negation
	$\mid$	Formel $\wedge$ Formel	Konjunktion
	$\mid$	Formel $\vee$ Formel	Disjunktion
	$\mid$	Formel $\rightarrow$ Formel	Implikation
	$\mid$	Formel $\leftrightarrow$ Formel	Äquivalenz/Biimplikation
	$\mid$	Formel $\oplus$ Formel	Xor
	$\mid$	(Formel)	Klammerung

- Alle Operatoren sind rechtsassoziativ
- Priorität der Operatoren (abnehmend): $\neg, \wedge, \vee, \oplus, \rightarrow, \leftrightarrow$
- Literal: positives oder negatives Vorkommen einer Aussagenvariable $\{x, \neg x\}$
- vars(φ): Menge der Variablen, die in Formel φ vorkommen
- lits(φ): Menge der Literale, die in Formel φ vorkommen

Syntax Beispiele

Beispiel (Prioritäten)

$$x \wedge y \vee \neg z \leftrightarrow x \oplus \neg z \wedge \neg w$$

ist klammerfreie Schreibweise für

$$((x \wedge y) \vee (\neg z)) \leftrightarrow (x \oplus ((\neg z) \wedge (\neg w)))$$

Beispiel (Syntaktisch korrekte Formeln)

- x
- $(x \vee y) \wedge (y \oplus z) \rightarrow \neg y$
- vars$((x \vee y) \wedge (y \oplus z) \rightarrow \neg y) = \{x, y, z\}$
- lits$((x \vee y) \wedge (y \oplus z) \rightarrow \neg y) = \{x, y, \neg y, z\}$

Evaluation von Formeln

- Menge der Wahrheitswerte: $\mathbb{B} = \{\mathtt{true}, \mathtt{false}\}$
- $\beta : \mathcal{V} \to \mathbb{B}$ ordnet jeder Variable einen Wahrheitswert zu

Algorithmus: $\mathtt{eval}(\beta, \psi)$

Eingabe: Belegung β, Formel ψ
Ausgabe: Evaluation von ψ unter β ($\mathtt{true}$ oder $\mathtt{false}$)

$$
\begin{aligned}
\mathtt{eval}(\beta, \psi) = \psi\ \mathtt{match}\ & \\
& \top \rightsquigarrow \mathtt{true} \\
|\ & \bot \rightsquigarrow \mathtt{false} \\
|\ & v \in \mathcal{V} \rightsquigarrow \beta(v) \\
|\ & \neg\varphi \rightsquigarrow \mathtt{if\ eval}(\beta, \varphi)\ \mathtt{then\ false\ else\ true} \\
|\ & \varphi_1 \wedge \varphi_2 \rightsquigarrow \mathtt{if\ eval}(\beta, \varphi_1)\ \mathtt{and\ eval}(\beta, \varphi_2)\ \mathtt{then\ true\ else\ false} \\
|\ & \varphi_1 \vee \varphi_2 \rightsquigarrow \mathtt{eval}(\beta, \neg(\neg\varphi_1 \wedge \neg\varphi_2)) \\
|\ & \varphi_1 \to \varphi_2 \rightsquigarrow \mathtt{eval}(\beta, \neg\varphi_1 \vee \varphi_2) \\
|\ & \varphi_1 \leftrightarrow \varphi_2 \rightsquigarrow \mathtt{eval}(\beta, (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)) \\
|\ & \varphi_1 \oplus \varphi_2 \rightsquigarrow \mathtt{eval}(\beta, \neg(\varphi_1 \leftrightarrow \varphi_2))
\end{aligned}
$$

Evaluation - Beispiel

Beispiel (Evaluation einer Formel)

- $\beta = \{x \mapsto \mathtt{false}, y \mapsto \mathtt{true}\}$
- $\mathtt{eval}(\beta, x \oplus y)$

$= \mathtt{eval}(\beta, \neg(x \leftrightarrow y))$

$= \mathtt{eval}(\beta, \neg((x \rightarrow y) \wedge (y \rightarrow x)))$

$= \mathtt{eval}(\beta, \neg((\neg x \vee y) \wedge (\neg y \vee x)))$

$= \mathtt{eval}(\beta, \neg(\neg(x \wedge \neg y) \wedge \neg(y \wedge \neg x)))$

$= \mathtt{if}\ \mathtt{eval}(\beta, \neg(x \wedge \neg y) \wedge \neg(y \wedge \neg x))\ \mathtt{then}\ \mathtt{false}\ \mathtt{else}\ \mathtt{true}$

$= \ldots$

$= \mathtt{true}$

Wahrheitstabellen

$$\varphi = x \land \neg(y \to z)$$

x	y	z	$y \to z$	$\neg(y \to z)$	$x \land \neg(y \to z)$
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	0	1	1
1	1	1	1	0	0

- Entscheidungsverfahren, das in 2^n Schritten entscheidet, ob eine gegebene Formel erfüllbar ist, oder nicht.

$\Rightarrow$ **Aussagenlogik ist entscheidbar**

- **Problem:** n Variablen benötigen 2^n Tabellenzeilen, daher nicht für große Formeln geeignet

Wahrheitstabellen—Demo

Demo

```
val phi = "x & ~(y => z)".pl
println(phi.truthTable)
----------------
x y z | formula
----------------
0 0 0 | 0
0 0 1 | 0
0 1 0 | 0
0 1 1 | 0
1 0 0 | 0
1 0 1 | 0
1 1 0 | 1
1 1 1 | 0
----------------
```

Äquivalenzumformungen

- Entfernen von Doppelnegationen

$$\neg\neg\varphi \equiv \varphi$$

- Distributivgesetze

$$\varphi_1 \vee (\varphi_2 \wedge \varphi_3) \equiv (\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)$$

$$\varphi_1 \wedge (\varphi_2 \vee \varphi_3) \equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_3)$$

- Absorptionsgesetze

$$\varphi_1 \vee (\varphi_1 \wedge \varphi_2) \equiv \varphi_1$$

$$\varphi_1 \wedge (\varphi_1 \vee \varphi_2) \equiv \varphi_1$$

- DeMorgansche Gesetze

$$\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2$$

$$\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$$

- Kommutativität und Assoziativität von $\wedge, \vee$
- ...

Erfüllbarkeit und Gültigkeit von Formeln

Definition (Erfüllbarkeit)

Eine Formel φ heißt erfüllbar, wenn eine Variablenbelegung $\beta : \text{vars}(\varphi) \to \mathbb{B}$ *existiert*, so dass $\texttt{eval}(\beta, \varphi) = \texttt{true}$

- β ist ein Modell von φ / φ ist gültig für β
- Notation: $\beta \models \varphi$

Definition (Tautologie)

Eine Formel φ ist eine Tautologie (oder heißt allgemeingültig), falls *für alle* $\beta : \text{vars}(\varphi) \to \mathbb{B}$ gilt, dass $\beta \models \varphi$

- Notation: $\models \varphi$

Definition (Kontradiktion)

Eine Formel φ ist eine Kontradiktion (oder heißt unerfüllbar), falls *kein* $\beta : \text{vars}(\varphi) \to \mathbb{B}$ *existiert*, so dass $\beta \models \varphi$

Schlussfolgerung und Äquivalenz

Definition (Implikation/Konsequenz)

φ impliziert ψ (ψ ist eine Konsequenz von φ) wenn für alle Belegungen β gilt: Wenn $\beta \vDash \varphi$ dann $\beta \vDash \psi$

- Notation: $\varphi \vDash \psi$

Definition (Äquivalenz)

φ und ψ sind äquivalent, wenn für alle Belegungen β gilt, dass $\beta \vDash \varphi$ gdw. $\beta \vDash \psi$

- Notation $\varphi \equiv \psi$

Übungen

Beweisen Sie folgende Aussagen:

1. φ ist eine Tautologie gdw. $\neg\varphi$ eine Kontradiktion ist.
2. $\varphi \vDash \psi$ gdw. $\vDash (\varphi \rightarrow \psi)$
3. $\varphi \equiv \psi$ gdw. $\vDash (\varphi \leftrightarrow \psi)$

Substitution

Definition (Substitution)

Ersetze alle Vorkommen von x in φ durch ψ.

- Notation: $\varphi[\psi/x]$

Theorem

Für eine beliebige Belegung β, Formeln φ und ψ mit $\mathtt{eval}(\beta, \varphi) = \mathtt{eval}(\beta, \psi)$, eine Variable x und eine Formel τ gilt

$$\mathtt{eval}(\beta, \tau[x/\varphi]) = \mathtt{eval}(\beta, \tau[x/\psi])$$

Theorem (Substitutionssatz)

Wenn $\varphi \equiv \psi$ dann gilt für beliebige Formeln τ:

$$\mathtt{eval}(\beta, \tau[x/\varphi]) = \mathtt{eval}(\beta, \tau[x/\psi]).$$

Im Speziellen heißt dies:

$$\models \tau[x/\varphi] \textit{ gdw. } \models \tau[x/\psi]$$

Normalformen

Warum Normalformen?

- Einheitliche Darstellung von Formeln
- Einfachere Datenstrukturen zum Speichern von Formeln
- Einfachere Dateiformate zum Speichern von Formeln
- Einfachere Inferenzmechanismen

Im Folgenden

- Negationsnormalform (NNF)
- Disjunktive Normalform (DNF)
- Konjunktive Normalform (CNF)

Bemerkung

Vor allem CNF spielt wichtige Rolle im Bereich SAT-Solving

Negationsnormalform (NNF)

Definition (NNF)

φ ist in NNF, wenn

1. nur die Operatoren $\neg$, $\wedge$ und $\vee$ in φ vorkommen
2. $\neg$ nur vor Variablen steht (nicht vor komplexen Formeln)

Beispiel (Negationsnormalform)

- $\neg(x \vee (y \wedge \neg z))$ nicht in NNF (wegen $\neg(x\ldots)$)
- $\neg x \wedge (\neg y \vee z)$ ist in NNF

Komplexität

Beim Auflösen von $\varphi_1 \leftrightarrow \varphi_2$ und $\varphi_1 \oplus \varphi_2$ werden φ_1 und φ_2 verdoppelt.

$\Rightarrow$ wort-case: aus n Operatoren werden über 2^n

NNF Algorithmus

Algorithmus: $\mathtt{nnf}(\psi)$

Eingabe: Formel ψ
Ausgabe: NNF von ψ

$\mathtt{nnf}(\psi) = \psi$ match

$\varphi_1 \wedge \varphi_2 \rightsquigarrow \mathtt{nnf}(\varphi_1) \wedge \mathtt{nnf}(\varphi_2)$

$\mid \varphi_1 \vee \varphi_2 \rightsquigarrow \mathtt{nnf}(\varphi_1) \vee \mathtt{nnf}(\varphi_2)$

$\mid \varphi_1 \rightarrow \varphi_2 \rightsquigarrow \mathtt{nnf}(\neg\varphi_1) \vee \mathtt{nnf}(\varphi_2)$

$\mid \varphi_1 \leftrightarrow \varphi_2 \rightsquigarrow (\mathtt{nnf}(\varphi_1) \wedge \mathtt{nnf}(\varphi_2)) \vee (\mathtt{nnf}(\neg\varphi_1) \wedge \mathtt{nnf}(\neg\varphi_2))$

$\mid \varphi_1 \oplus \varphi_2 \rightsquigarrow (\mathtt{nnf}(\varphi_1) \wedge \mathtt{nnf}(\neg\varphi_2)) \vee (\mathtt{nnf}(\neg\varphi_1) \wedge \mathtt{nnf}(\varphi_2))$

$\mid \neg\neg\varphi \rightsquigarrow \mathtt{nnf}(\varphi)$ *(Doppelnegation)*

$\mid \neg(\varphi_1 \wedge \varphi_2) \rightsquigarrow \mathtt{nnf}(\neg\varphi_1) \vee \mathtt{nnf}(\neg\varphi_2)$ *(De Morgan)*

$\mid \neg(\varphi_1 \vee \varphi_2) \rightsquigarrow \mathtt{nnf}(\neg\varphi_1) \wedge \mathtt{nnf}(\neg\varphi_2)$ *(De Morgan)*

$\mid \neg(\varphi_1 \rightarrow \varphi_2) \rightsquigarrow \mathtt{nnf}(\varphi_1) \wedge \mathtt{nnf}(\neg\varphi_2)$

$\mid \neg(\varphi_1 \leftrightarrow \varphi_2) \rightsquigarrow (\mathtt{nnf}(\varphi_1) \wedge \mathtt{nnf}(\neg\varphi_2)) \vee (\mathtt{nnf}(\neg\varphi_1) \wedge \mathtt{nnf}(\varphi_2))$

$\mid \neg(\varphi_1 \oplus \varphi_2) \rightsquigarrow (\mathtt{nnf}(\varphi_1) \wedge \mathtt{nnf}(\varphi_2)) \vee (\mathtt{nnf}(\neg\varphi_1) \wedge \mathtt{nnf}(\neg\varphi_2))$

$\mid \_ \rightsquigarrow \psi$

Disjunktive Normalform (DNF)

- Formel ist eine Disjunktion von Konjunktionen (Minterme)

Beispiel (DNF)

- $(x_1 \wedge y_1 \wedge z_1) \vee (x_2 \wedge y_2) \vee z_2$

Vorteile

- Positive Aufzählung der Eins-Stellen („was geht“)
- Für Erfüllbarkeitstest (SAT) muss nur ein Minterm erfüllbar sein
- Einfacher Check auf Kontradiktion

Nachteil

Kann exponentielles Wachstum gegenüber der Originalformel haben

DNF Algorithmen

- *Theoretisch:* Ablesen aus der Wahrheitstabelle, jedoch praktisch nicht möglich
- *Besser:* Anwendung des Distributivgesetzes

$$\varphi_1 \land (\varphi_2 \lor \varphi_3) \rightsquigarrow (\varphi_1 \land \varphi_2) \lor (\varphi_1 \land \varphi_3)$$

Algorithmus: $\mathtt{dnf}(\psi)$

Eingabe: Formel ψ
Ausgabe: DNF von ψ

$\mathtt{dnf}(\psi) = \mathtt{nnf}(\psi)\ \mathtt{match}$

$\quad \varphi_1 \land \varphi_2 \rightsquigarrow \mathtt{distrib}(\mathtt{dnf}(\varphi_1) \land \mathtt{dnf}(\varphi_2))$

$\mid \varphi_1 \lor \varphi_1 \rightsquigarrow \mathtt{dnf}(\varphi_1) \lor \mathtt{dnf}(\varphi_2)$

$\mid \_\_ \rightsquigarrow \mathtt{nnf}(\psi)$

$\mathtt{distrib}(\psi) = \psi\ \mathtt{match}$

$\quad \varphi_1 \land (\varphi_2 \lor \varphi_3) \rightsquigarrow (\mathtt{distrib}(\varphi_1 \land \varphi_2)) \lor (\mathtt{distrib}(\varphi_1 \land \varphi_3))$

$\mid (\varphi_1 \lor \varphi_2) \land \varphi_3 \rightsquigarrow (\mathtt{distrib}(\varphi_1 \land \varphi_3)) \lor (\mathtt{distrib}(\varphi_2 \land \varphi_3))$

$\mid \_\_ \rightsquigarrow \psi$

Konjunktive Normalform (CNF)

- Formel ist eine Konjunktion von Disjunktionen (Klauseln)

Beispiel (CNF)

- $\neg x \wedge (y \vee z) \wedge (\neg y \vee w \vee \neg x)$

Vorteile

- Negative Aufzählung der Nullstellen („was geht nicht“)
- Klauseln sind Randbedingungen *(constraints)* der Erfüllbarkeit
- Einfacher Check auf Tautologie

Nachteil

Kann exponentielles Wachstum gegenüber der Originalformel haben

CNF Algorithmen

- *Theoretisch:* Ablesen aus der Wahrheitstabelle, jedoch praktisch nicht möglich
- *Besser:* Anwendung des Distributivgesetzes

$$\varphi_1 \lor (\varphi_2 \land \varphi_3) \rightsquigarrow (\varphi_1 \lor \varphi_2) \land (\varphi_1 \lor \varphi_3)$$

- *Alternative*—Ausnutzen von De Morgan:

$$\neg\varphi \leftrightarrow \bigvee_{i=1}^{m} \bigwedge_{j=1}^{n} \varphi_{ij} \rightsquigarrow \varphi \leftrightarrow \bigwedge_{i=1}^{m} \bigvee_{j=1}^{n} \neg\varphi_{ij}$$

Algorithmus: $\mathtt{cnf}(\psi)$

Eingabe: Formel ψ
Ausgabe: CNF von ψ

$$\mathtt{cnf}(\psi) = \mathtt{nnf}(\neg\mathtt{dnf}(\neg\psi))$$

Simplifikation von CNF und DNF

CNF und DNF sind dual zueinander

Komplementäre Literale

- Enthält ein Minterm komplementäre Literale x und $\neg x$, so evaluiert er zu `false`
- Enthält eine Klausel komplementäre Literale x und $\neg x$, so evaluiert sie zu `true`

Redundante Klauseln/Minterme

- Gilt für zwei Minterme m_1 und m_2, dass lits$(m_1) \subseteq$ lits(m_2), so absorbiert m_1 m_2 und m_2 ist redundant.
- Gilt für zwei Klauseln c_1 und c_2, dass lits$(c_1) \subseteq$ lits(c_2), so subsumiert m_1 m_2 und m_2 ist redundant.

Definitional CNF

Idee

Führe neue Variablen für Teilformeln ein (Definitionen) und verhindere somit exponentiellen Blow-Up der Formel.

- Idee geht zurück auf Tseitin (1968)
- Abwandlung: Plaisted-Greenbaum (1986)
- Entstehende CNF ist nicht mehr äquivalent zur Originalformel, sondern erfüllbarkeitsäquivalent

Literaturhinweis

- *G. S. Tseitin.* **On the complexity of derivation in propositional calculus.** Leningrad Seminar on Mathematical Logic, 1970.
- *D. A. Plaisted and S. Greenbaum.* **A Structure-Preserving Clause Form Translation.** Journal of Symbolic Computation, 1986.

Definitional CNF—Beispiel

Beispiel (Tseitin Verfahren)

- Originalformel: $(p \vee (q \wedge \neg r)) \wedge s$
- Führe neue Variable p_1 für $q \wedge \neg r$ ein (Definition von p_1)

$$(p \vee p_1) \wedge s \wedge (p_1 \leftrightarrow q \wedge \neg r)$$

- p_2 definiert $p \vee p_1$

$$(p_1 \leftrightarrow q \wedge \neg r) \wedge (p_2 \leftrightarrow p \vee p_1) \wedge p_2 \wedge s$$

- p_3 definiert $p_2 \wedge s$

$$(p_1 \leftrightarrow q \wedge \neg r) \wedge (p_2 \leftrightarrow p \vee p_1) \wedge (p_3 \leftrightarrow p_2 \wedge s) \wedge p_3$$

- Jedes Konjunkt kann nun mit Standard-Methoden in CNF übersetzt werden:

$$(\neg p_1 \vee q) \wedge (\neg p_1 \vee \neg r) \wedge (p_1 \vee \neg q \vee r) \wedge$$
$$(\neg p_2 \vee p \vee p_1) \wedge (p_2 \vee \neg p) \wedge (p_2 \vee \neg p_1) \wedge$$
$$(\neg p_3 \vee p_2) \wedge (\neg p_3 \vee \neg s) \wedge (p_3 \vee \neg p_2 \vee \neg s) \wedge p_3$$

Normalformen—Demos

Demo

```
val phi = val phi = "p => (r & ~q & ~s | r   s)".pl
```

```
phi.nnf
... = (¬p ∨ (r ∧ ¬q ∧ ¬s) ∨ (¬r ∧ s))
```

```
phi.cnf
... = ((¬r ∨ r ∨ ¬p) ∧ (s ∨ r ∨ ¬p) ∧ (¬r ∨ ¬q ∨ ¬p) ∧
(s ∨ ¬q ∨ ¬p) ∧ (¬r ∨ ¬s ∨ ¬p) ∧ (s ∨ ¬s ∨ ¬p))
```

```
phi.cnf.simplify(NaiveCNFSimplificationStrategy)
... = (s ∨ r ∨ ¬p) ∧ (¬r ∨ ¬q ∨ ¬p) ∧ (s ∨ ¬q ∨ ¬p) ∧
(¬r ∨ ¬s ∨ ¬p)
```

```
phi.dnf
... = (¬p ∨ (r ∧ ¬q ∧ ¬s) ∨ (¬r ∧ s))
```

Erfüllbarkeitsprüfung (SAT)

Warum Erfüllbarkeitsprüfung

Ein Algorithmus, der Erfüllbarkeit testen kann, kann für eine Formel φ folgende Ausgaben liefern:

- φ erfüllbar (mit Angabe eines Modells)
- φ nicht erfüllbar (Kontradiktion)
- φ allgemeingültig (wenn $\neg\varphi$ unerfüllbar)

Welche Methoden gibt es? *(AB Grundlagen)*

- Wahrheitstabellen
- Resolutionskalkül
- SAT Solver
- Tableaus
- BDDs

Resolutionskalkül

Das Resolutionskalkül arbeitet auf einer Menge Γ von Klauseln:

Resolution

$$\frac{\Delta \cup \{\varphi \vee x\} \cup \{\psi \vee \neg x\}}{\Delta \cup \{\varphi \vee x\} \cup \{\psi \vee \neg x\} \cup \{\varphi \vee \psi\}}$$

Faktorisierung

$$\frac{\Delta \cup \{\varphi \vee x \vee x\}}{\Delta \cup \{\varphi \vee x \vee x\} \cup \{\varphi \vee x\}}$$

Subsumption

$$\frac{\Delta \cup \{\varphi\} \cup \{\psi\} \qquad \varphi \subseteq \psi}{\Delta \cup \{\varphi\}}$$

Ersetzende Resolution

$$\frac{\Delta \cup \{\varphi \vee x\} \cup \{\psi \vee \neg x\} \qquad \varphi \subseteq \psi}{\Delta \cup \{\varphi \vee x\} \cup \{\psi\}}$$

i Resolution ist korrekt und vollständig

Aussagenlogische Resolution ist korrekt und vollständig: Γ ist unerfüllbar gdw. die leere Klausel mit obigen Regeln aus Γ geschlussfolgert werden kann.

DPLL Algorithmus—Unit Propagation

- Algorithmus zur Erfüllbarkeitsprüfung von Formeln in CNF
- Formel wird als Klauselmenge betrachtet, Klausel als Literalmenge

Idee

Wechsle Suche und Deduktion miteinander ab.

Definition (Unit Propagation)

Befindet sich in der Klauselmenge eine Unit Klausel (Klausel mit nur einem Literal) $\{l\}$, dann

1. entferne alle Vorkommen von $-l$ aus der Formel
2. enfterne alle Klauseln, die l enthalten

Beispiel (Unit Propagation)

- $\{\{a, b, c\}, \{\neg a, \neg b\}, \{c, \neg d\}, \{a\}\}$
- Propagiere a: $\{\{\neg b\}, \{c, \neg d\}\}$
- Propagiere $\neg b$: $\{\{c, \neg d\}\}$

DPLL Algorithmus

Algorithmus: $\mathtt{dpll}(\varphi, \beta)$

Eingabe: Formel φ, Belegung β
Ausgabe: `true` wenn die Formel erfüllbar ist, `false` sonst

$\mathtt{UnitPropagation}(\varphi)$
if $(\mathtt{eval}(\beta, \varphi))$ **then**
  return `true`
if $\mathtt{not}(\mathtt{eval}(\beta, \varphi)$ **then**
  return `false`

wähle eine noch nicht belegte Variable $x \in \mathrm{vars}(\varphi)$

if $\mathtt{dpll}(\varphi \cup \{\{\neg x\}\}, \beta \cup [x \mapsto \mathtt{false}])$ **then**
  return `true`
else
  return $\mathtt{dpll}(\varphi \cup \{\{x\}\}, \beta \cup [x \mapsto \mathtt{true}])$

SAT Solving—Demo

Andbindung des C Solvers `Picosat` über `JNA`.

Demo

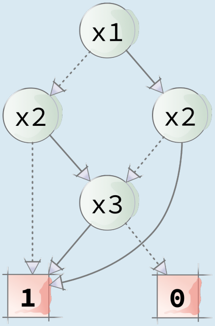
```
import org.warthog.pl.decisionprocedures.satsolver.impl.picosat._
import org.warthog.pl.generators.PigeonHoleGenerator

val solver = new Picosat
solver.init

val f = PigeonHoleGenerator.generate(6).cnf

println(f.stats)
solver.add(f)
solver.sat(5000)
... = -1
```

BDDs

- Knowledge Compilation Format
- Datenstruktur DAG
  - Knoten sind Variablen
  - Blätter sind die Terminale $\boxed{0}$ (`false`) und $\boxed{1}$ (`true`)
  - Kanten sind Belegung der Variable mit `true` oder `false`

Beispiel (BDD)

Formel: $(x_1 \leftrightarrow x_2) \vee x_3$, Variablenordnung $x_1 < x_2 < x_3$



ROBDDs

- Normalerweise betrachtet man immer ROBDDs:
  - reduziert: keine gleichen Teilbäume im BDD, bei keinem Knoten ist der rechte gleich dem linken Teilbaum
  - geordnet: auf allen Pfaden gilt die gleiche Variablenreihenfolge
- Kanonische Normalform für aussagenlogische Formeln

  $\Rightarrow$ Tautologie entspricht BDD $\boxed{1}$

  $\Rightarrow$ Kontradiktion entspricht BDD $\boxed{0}$

  $\Rightarrow$ Erfüllbare Formel entspricht BDD ungleich $\boxed{0}$
- Graphalgorithmen können benutzt werden
  - Finden einer erfüllenden Belegung $\Rightarrow$ ein Pfad zu $\boxed{1}$
  - Auflisten aller erfüllenden Belegungen $\Rightarrow$ alle Pfade zu $\boxed{1}$
  - ...

ROBDDs—Demo

In `Warthog` werden BDDs (wie z.B. in `CUDD`) mit komplementären Kanten gespeichert.

Demo

```
import org.warthog.pl.knowledgecompilation.bdd._

val f = "~a | b & (~b | c)".pl

val man = new BDDManager
val bdd = man.mkBDD(f)

man.isTautology(bdd)
... = false

man.isContradiction(bdd)
... = false
```

Kompaktheitssatz

Definition (Erfüllbarkeit von Formelmengen)

Eine Menge von Formeln Γ ist erfüllbar gdw. eine Belegung β existiert, die gleichzeitig alle Formeln in Γ erfüllt.

Theorem (Kompaktheitssatz)

Für eine beliebige Menge Γ von aussagenlogischen Formeln gilt: Γ ist erfüllbar gdw. jede endliche Teilmenge $\Delta \subseteq \Gamma$ erfüllbar ist.

Beweis.

- Hinrichtung $\Rightarrow$ ist offensichtlich
- Annahme: Menge der Variablen ist abzählbar: $p_1, p_2, \ldots$
- Vorgehensweise: Konstruktion einer Belegung β, die Γ erfüllt
    - Dazu: Belegung je einer Variablen p_i (eine nach der anderen)

Beweis des Kompaktheitsatzes—1

Wenn ...

- ... es Wahrheitswerte $t_1, t_2, \ldots, t_n$ gibt, so dass
- jedes endliche $\Delta \subseteq \Gamma$ mit der Belegung $\{p_1 \mapsto t_1, \ldots, p_n \mapsto t_n\}$ erfüllbar ist (1)

dann ...

- gibt es einen Wahrheitswert t_{n+1}, so dass
- jedes endliche $\Delta \subseteq \Gamma$ auch mit der Belegung $\{p_1 \mapsto t_1, \ldots, p_n \mapsto t_n, p_{n+1} \mapsto t_{n+1}\}$ erfüllbar ist

---

Angenommen diese Aussage ist falsch, dann ...

- ... existiert eine endliche Teilmenge $\Delta_0 \subseteq \Gamma$, die durch keine Belegung $\{p_1 \mapsto t_1, \ldots, p_n \mapsto t_n, p_{n+1} \mapsto \mathtt{false}\}$ erfüllt ist
- ... existiert eine endliche Teilmenge $\Delta_1 \subseteq \Gamma$, die durch keine Belegung $\{p_1 \mapsto t_1, \ldots, p_n \mapsto t_n, p_{n+1} \mapsto \mathtt{true}\}$ erfüllt ist

$\Rightarrow$ $\Delta_0 \cup \Delta_1$ ist nicht erfüllbar durch $\{p_1 \mapsto t_1, \ldots, p_n \mapsto t_n\}$

- Da jedoch Δ_0 und Δ_1 jeweils endlich sind, ist auch $\Delta_0 \cup \Delta_1$ endlich

$\Rightarrow$ Widerspruch zur Annahme (1)

Beweis des Kompaktheitsatzes—2

Definiere rekursiv eine unendliche Reihe von Wahrheitswerten (t_i), so dass

- für ein beliebiges $n \in \mathbb{N}$ gilt: jedes endliche $\Delta \subseteq \Gamma$ ist erfüllbar durch $\{p_1 \mapsto t_1, \ldots, p_n \mapsto t_n\}$

Behauptung: $\beta = \{p_1 \mapsto t_1, p_2 \mapsto t_2, \ldots\}$ erfüllt Γ, d.h. erfüllt jede Formel $\gamma \in \Gamma$

Für jedes γ

- ist $\mathrm{vars}(\gamma)$ endlich
- d.h. es gibt ein N, so dass jedes $p_n \in \mathrm{vars}(\gamma)$ ein $n < N$ hat
- Per Konstruktion sind alle endlichen Teilmengen von Γ, im Speziellen $\{\gamma\}$, erfüllbar durch eine Belegung β', wobei $\beta'(p_n) = t_n = \beta(p_n)$ für $n \leq N$
- Belegungen von Variablen, die nicht in γ vorkommen, sind irrelevant

$\Rightarrow$ γ wird durch β erfüllt

Craig Interpolation

Theorem (Craig's Interpolation Theorem)

- *φ und ψ sind aussagenlogische Formeln*

Wenn $\models \varphi \land \psi \to \bot$, dann gibt es eine Interpolante γ mit $\mathsf{vars}(\gamma) \subseteq \mathsf{vars}(\varphi) \cap \mathsf{vars}(\psi)$, so dass $\models \varphi \to \gamma$ und $\models \psi \to \neg\gamma$.

Induktion über die Anzahl an Elementen in $\Delta = \mathsf{vars}(\varphi) \setminus \mathsf{vars}(\psi)$:

- Leere Menge: Wähle φ als Interpolante
  - $\models \varphi \to \varphi$ gilt
  - Wegen $\models \varphi \land \psi \to \bot$ gilt auch $\models \psi \to \neg\varphi$
- Andernfalls: Wähle eine Variable x aus Δ und konstruiere $\varphi' = \varphi[\mathtt{true}/x] \lor \varphi[\mathtt{false}/x]$ (eliminiere p)
  - $|\mathsf{vars}(\varphi') \setminus \mathsf{vars}(\psi)| < |\mathsf{vars}(\varphi) \setminus \mathsf{vars}(\psi)|$

  $\Rightarrow$ es gibt eine Interpolante γ mit $\models \varphi' \to \gamma$ und $\models \psi \to \neg\gamma$. (IH)
  - Es gilt jedoch auch $\models \varphi \to \varphi'$ und damit auch $\models \varphi \to \gamma$
  - $\mathsf{vars}(\gamma) \subseteq \mathsf{vars}(\varphi') \cap \mathsf{vars}(\psi)$ und $\mathsf{vars}(\varphi') = \mathsf{vars}(\varphi) \setminus \{x\} \subseteq \mathsf{vars}(\varphi)$

Craig Interpolation—Demo

Algorithmus: `pinterpolate`(φ, ψ)

Eingabe: Formeln φ und ψ mit $\vDash \varphi \wedge \psi \rightarrow \bot$
Ausgabe: Craig Interpolante von φ und ψ

$\texttt{pinterpolate}(\varphi,\psi) = \left|\mathrm{vars}(\varphi) \setminus \mathrm{vars}(\psi)\right|\ \texttt{match}$

$\quad 0 \rightsquigarrow \varphi$

$\quad | \ \_ \rightsquigarrow \texttt{pinterpolate}(\varphi[\mathtt{true}/p] \vee \varphi[\mathtt{false}/p], \psi)$

$\quad$ *mit* $p \in \mathrm{vars}(\varphi) \setminus \mathrm{vars}(\psi)$

Demo

```
import org.warthog.pl.algorithms.CraigInterpolation

val phi = "(a => b) & (b => c)".pl
val psi = ä & ~c".pl
val gamma = CraigInterpolation.pinterpolate(phi,psi)
... = (¬a ∨ c)
```