

Skriptum zur Vorlesung

Automatisches Beweisen

Sommersemester 2006

Prof. Dr. W. Kuchlin

WSI für Informatik
Universität Tübingen

Skriptum erstellt auf Grundlage des Skriptes von Dr. Carsten Sinz
aus dem SS 2003 unter Mitwirkung von Monika Gehweiler

September 2007

Inhaltsverzeichnis

1	Aussagenlogik	1
1.1	Syntax	2
1.2	Semantik	3
1.3	Äquivalenzen	5
1.4	Normalformen	7
2	Aussagenlogische Entscheidungsverfahren	11
2.1	Resolution	11
2.2	Davis–Putnam Algorithmus	15
2.3	Binäre Entscheidungsdiagramme (BDDs)	18
3	Prädikatenlogik (PL1)	25
3.1	Syntax	25
3.2	Semantik	27
3.3	Substitutionen	29
3.4	Normalformen	30
3.5	Unifikation	33
3.6	Prädikatenlogische Resolution	36

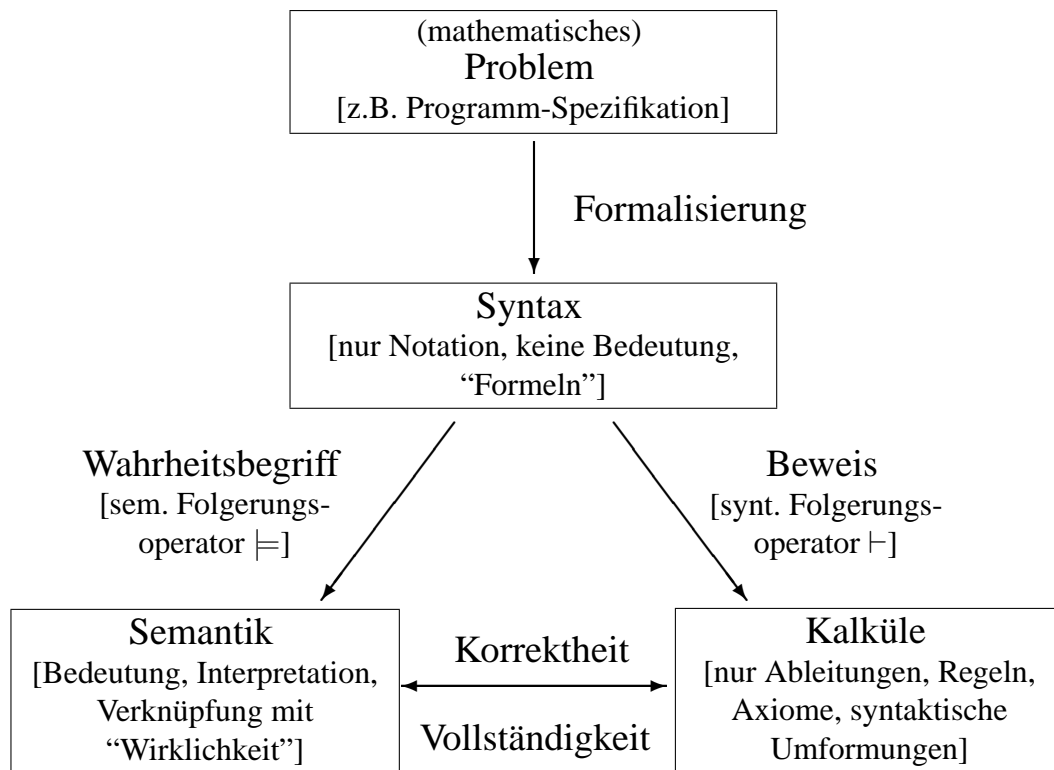


Abbildung 1: Zentrale Begriffe der mathematischen Logik und deren Beziehungen (nach B. Becker)

1 Aussagenlogik

In der Aussagenlogik werden komplexe Aussagen durch Verknüpfungen („und“, „oder“, „nicht“) aus elementaren Aussagen aufgebaut. Elementaraussagen sind entweder wahr oder falsch und nicht weiter zerlegbar.

Der Wahrheitswert einer komplexen Aussage hängt nur von den Wahrheitswerten der Teilaussagen und der gewählten Verknüpfung (dem *Junktor*) ab. Daher ist es z.B. nicht möglich, eine Verknüpfung wie „A solange B“ in der Aussagenlogik auszudrücken.

1.1 Syntax

Definition Die Sprache der Aussagenlogik (AL) besteht aus Symbolen für:

- Atome (atomare Aussagen, Aussagevariablen (propositional variables), (Prädikat-)Variablen): a_1, a_2, a_3, \dots ("genügend viele"); wenn angenehm auch x, y, z, \dots
- Junktoren (logische Verknüpfungsoperatoren):
 - $\vee(\cdot, \cdot)$ („Kerbe“)
 - $\wedge(\cdot, \cdot)$ („Dach“)
 - $\neg(\cdot)$ („Hacken“)
 - \perp („Bottom“)
- Klammern: $(,)$

Formeln sind besondere endliche Zeichenketten über diesen Symbolen. Welche Zeichenketten eine gültige Formel ergeben, wird durch die folgende Definition festgelegt.

Definition 1.1 (aussagenlogische Formel) Sei $\Phi_0 = \{x, y, z, \dots\}$ eine Menge von Aussagevariablen. Die Menge Φ der aussagenlogischen Formeln über Φ_0 ist definiert als die kleinste Menge, für die gilt:

1. $\Phi_0 \subseteq \Phi, \perp \in \Phi$.
2. Falls $F, G \in \Phi$, so auch $(F \vee G) \in \Phi, (F \wedge G) \in \Phi$ und $\neg F \in \Phi$.

Präzedenzregeln (um Klammern zu sparen):

- \wedge bindet stärker als \vee
- \neg bindet am stärksten
- \vee und \wedge sind linksassoziativ:
 - $F \vee G \vee H$ wird verstanden als $(F \vee G) \vee H$,
 - $F \wedge G \wedge H$ wird verstanden als $(F \wedge G) \wedge H$
- Außenklammern können weggelassen werden.

Beispiel:

$$\neg x \vee y \wedge z \vee \perp$$

$$((\neg x) \vee (y \wedge z)) \vee \perp$$

Abgeleitete Operatoren:

$$F \Rightarrow G := \neg F \vee G \quad (\text{Implikation})$$

$$F \Leftrightarrow G := (F \Rightarrow G) \wedge (G \Rightarrow F) \quad (\text{Äquivalenz})$$

$$\top := \neg \perp$$

Für eine Formel $F \in \Phi$ bezeichnen wir mit $Var(F)$ die in F auftretenden Variablen. Ein *Literal* ist ein Atom oder dessen Negation. (z.B. $x, \neg y, \dots$).

Φ_0 ist die Menge der positiven Literale.

$\neg\Phi_0$ ist die Menge der negativen Literale. Dabei gilt: $\neg\Phi_0 := \{\neg x \mid x \in \Phi_0\}$ Die Menge der Literale bezeichnen wir mit L (Es ist $L = \Phi_0 \cup \neg\Phi_0$)

Beispiel:

$(x), ((x))$ ist keine gültige Formel

$\neg(x \vee y)$ ist eine gültige Formel

1.2 Semantik

Die Semantik ordnet den Formeln eine Bedeutung zu.

Definition 1.2 (Variablenbelegung) Eine (Variablen-)Belegung oder Interpretation β_0 ist eine Abbildung $\beta_0 : \Phi_0 \rightarrow \{0, 1\}$, die den Aussagevariablen Wahrheitswerte zuweist (0: falsch, 1: wahr).

Definition 1.3 (Bedeutung einer Formel) Sei β_0 eine Variablenbelegung. Wir erweitern β_0 zur Interpretation β , die auf der Menge Φ der Formeln definiert ist durch

$$\begin{aligned} \beta(x) &= \beta_0(x) \quad \text{für } x \in \Phi_0 & \beta(F \vee G) &= \max(\beta(F), \beta(G)) \\ \beta(\neg F) &= 1 - \beta(F) & \beta(F \wedge G) &= \min(\beta(F), \beta(G)) \\ \beta(\perp) &= 0 \end{aligned}$$

Beispiel:

$$\Phi_0 = \{x, y, z\}, \beta_0 : \{x, y, z\} \rightarrow \{0, 1\}$$

mit $\beta_0(x) = 0, \beta_0(y) = \beta_0(z) = 1$

$$\begin{aligned} &\beta(\neg x \vee (y \wedge z) \vee \perp) = \\ &\max(\beta(\neg x \vee (y \wedge z)), \beta(\perp)) = \\ &\max(\max(1 - \beta_0(x), \min(\beta_0(y), \beta_0(z))), \beta_0(\perp)) = 1 \end{aligned}$$

Definition 1.4 (Erfüllbarkeit, Allgemeingültigkeit) Eine Formel $F \in \Phi$ heißt erfüllbar, wenn es eine Variablenbelegung β_0 gibt mit $\beta(F) = 1$. Ansonsten heißt F unerfüllbar. Gilt für alle Variablenbelegungen β_0 , dass $\beta(F) = 1$, so heißt F (allgemein-)gültig. F wird dann auch eine Tautologie genannt.

Beispiel:

$F = \neg x \vee (y \wedge z) \vee \perp$ ist erfüllbar (β_0 s.o.), nicht allgemeingültig, da $\beta(F) = 0$ für z.B. $\beta_0(x) = 1, \beta_0(y) = 0$.

Definition 1.5 ((semantischer) Folgerungsbegriff) Sei $M \subseteq \Phi$ eine Formelmeng
e und $F \in \Phi$. Dann folgt F (semantisch) aus M , falls für jede Belegung β_0 mit $\beta(G) = 1$ für alle $G \in M$ auch $\beta(F) = 1$ gilt. Wir schreiben dafür auch $M \models F$. Für $\{G\} \models F$ schreiben wir auch $G \models F$, für $\emptyset \models F$ auch einfach $\models F$.

Anmerkung:

$\models F$ ist äquivalent zu F ist eine Tautologie.

Beispiel:

Sei $M = \{x \Rightarrow y, x \Rightarrow z\}$

$F = \neg x \vee (y \wedge z) \vee \perp$

$M \models F$?

x	y	z	$x \Rightarrow z$	$y \Rightarrow z$	F
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	1

Anhand der Tabelle sieht man, dass jede Variablenbelegung, die $x \Rightarrow y$ und $x \Rightarrow z$ wahr macht, auch F wahr macht. Somit gilt $M \models F$.

1.3 Äquivalenzen

Wir schreiben auch $F \equiv G$ für $\models F \Leftrightarrow G$. Es gilt:

$$\neg\neg F \equiv F$$

$$F \vee F \equiv F \quad \text{„Idempotenz“}$$

De Morgan'sche Regeln:

$$\neg(F \vee G) \equiv \neg F \wedge \neg G$$

$$\neg(F \wedge G) \equiv \neg F \vee \neg G$$

Distributivgesetze:

$$(F \wedge G) \vee H \equiv (F \vee H) \wedge (G \vee H)$$

$$(F \vee G) \wedge H \equiv (F \wedge H) \vee (G \wedge H)$$

⋮

Definition 1.6 (Boolesche Algebra) Sei B eine Menge, \vee, \wedge zwei Operatoren und $0, 1 \in B$.

Es gelte:

1. \vee und \wedge sind assoziativ und kommutativ

2. es gelten die Distributivgesetze

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$

und

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

3. $x \wedge 0 = 0, x \wedge 1 = x \quad \forall x$

4. $x \vee 0 = x, x \vee 1 = 1 \quad \forall x$

5. Zu jedem x existiert genau ein x' mit $x \wedge x' = 0$ und $x \vee x' = 1$

Dann heißt $[B; \vee, \wedge, ', 0, 1]$ eine Boolesche Algebra.

Der Beweis von $\neg\neg F \equiv F$ ist jetzt einmal über die Definition und Belegungen möglich, oder auch in der Booleschen Algebra.

Satz 1.7 In einer Booleschen Algebra gilt $\neg\neg F = F$.

Beweis: Nach 2. ex. $\neg\neg x$ mit $\neg x \wedge \neg\neg x = 0$.

Nach 2. ex. $\neg x$ mit $x \wedge \neg x = 0$.

Wegen der Kommutativität (1.) gilt nun $x \wedge \neg x = \neg x \wedge x = 0$. Wegen der Eindeutigkeit von $\neg\neg x$ gilt $\neg\neg x = x$.

Weitere Sätze der Booleschen Algebra:

- $\neg\neg x = x$ (wie gezeigt)
- $x \vee x = x \quad x \wedge x = x$
- $x \vee (x \wedge y) = x \quad y \wedge (x \vee y) = y$
- DeMorgan: $\neg(x \vee y) = \neg x \wedge \neg y \quad \neg(x \wedge y) = \neg x \vee \neg y$

Definition 1.8 (Dualität) F heißt dual zu G , falls F aus G durch Vertauschen der Junktoren \vee und \wedge und der Symbole \top und \perp entsteht. Wir schreiben auch F^δ für die zu F duale Formel.

Satz 1.9 Falls $F \equiv G$, so gilt auch: $F^\delta \equiv G^\delta$

Beweis: Übung.

Lemma 1.10 Sei $F \in \Phi$. Es gilt:

1. F erfüllbar $\Leftrightarrow \neg F$ keine Tautologie [F Tautologie $\Leftrightarrow \neg F$ unerfüllbar].
2. $\models F \wedge G$ gdw. $\models F$ und $\models G$.

Beweis: Übung.

Satz 1.11 (Deduktionstheorem) Sei $M \subseteq \Phi$ und $F, G \in \Phi$. Dann gilt:

$$M \cup \{F\} \models G \quad \text{impliziert} \quad M \models F \Rightarrow G.$$

Anwendung zum Nachweis von $M \models F$, wobei $M = \{G_1, \dots, G_n\}$

$M \models F$?

$\{G_1, \dots, G_n\} \models F$

$\{G_1, \dots, G_{n-1}\} \models G_n \Rightarrow F$

\vdots

$\models G_1 \Rightarrow (\dots(G_n \Rightarrow F))$

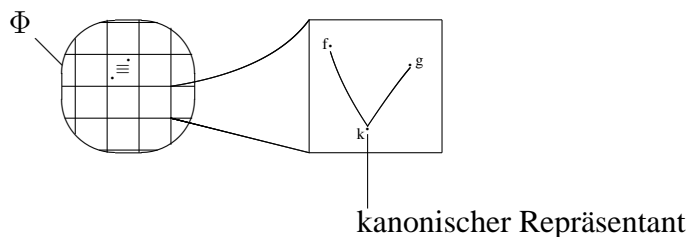
oder $\models (G_1 \wedge \dots \wedge G_n) \Rightarrow F$

1.4 Normalformen

Vorteile von Normalformen: einfachere Algorithmen und Datenstrukturen.

Wichtige Unterscheidung:

- *Normalform* – ein bestimmtes Aussehen (z.B. CNF)
- *kanonische Form* – ein eindeutiger Repräsentant pro Äquivalenzklasse



Definition 1.12 (vereinfachte Formel) Eine Formel $F \in \Phi$ heißt vereinfacht, falls entweder $F = \neg \perp (= \top)$ oder $F = \perp$ oder \perp kommt nicht in F vor.

Lemma 1.13 Zu jeder Formel $F \in \Phi$ gibt es eine äquivalente vereinfachte Formel.

Beweis 1: Ersetze \perp durch $(x \wedge \neg x)$

$$\begin{array}{ll}
 \text{Beweis 2: } F \vee \perp \rightsquigarrow F & F \vee \top \rightsquigarrow \top \\
 \perp \vee F \rightsquigarrow F & \top \vee F \rightsquigarrow \top \\
 F \wedge \perp \rightsquigarrow \perp & F \wedge \top \rightsquigarrow F \\
 \perp \wedge F \rightsquigarrow \perp & \top \wedge F \rightsquigarrow F
 \end{array}$$

Definition 1.14 (Negationsnormalform) Eine Formel $F \in \Phi$ ist in Negationsnormalform (NNF), falls Negationen nur direkt vor Aussagevariablen auftreten. (\perp wird dabei wie eine Aussagevariable behandelt.)

Beispiele zur NNF:

- $\neg(x \vee y)$ nicht in NNF.
- $\neg x \wedge \neg y$ in NNF.
- $\neg \neg x$ nicht in NNF; x ist eine dazu äquivalente Formel in NNF.

Lemma 1.15 Zu jeder Formel $F \in \Phi$ gibt es eine äquivalente Formel in NNF.

Algorithmus für NNF: (Termersetzungssystem)

$$\neg(F \wedge G) \rightsquigarrow \neg F \vee \neg G$$

$$\neg(F \vee G) \rightsquigarrow \neg F \wedge \neg G$$

$$\neg\neg F \rightsquigarrow F$$

(Redex \rightsquigarrow Reduktion)

Beispiel:

$$F = \neg(x \wedge \neg y) \vee \neg(y \vee \perp)$$

$$\stackrel{2*}{\rightsquigarrow} \neg x \vee \neg\neg y \vee (\neg y \wedge T)$$

$$\rightsquigarrow \neg x \vee y \vee \neg y \wedge T$$

Definition 1.16 (Klausel) Eine Disjunktion $F = (l_1 \vee \dots \vee l_n)$ von Literalen l_i ($1 \leq i \leq n$) heißt Klausel. Sind alle Literale positiv (negativ) so heißt F positive (negative) Klausel. Ist höchstens ein Literal in F positiv, so heißt F Horn-Klausel. Eine Klausel mit k Literalen heißt k -Klausel. Eine 1-Klausel wird auch Unit-Klausel genannt. Die leere Klausel (i.Z.: \square) entspricht der leeren Disjunktion, die als \perp interpretiert wird, denn es ist:

$$\bigvee(F_1, \dots, F_n) = \begin{cases} \perp & n = 0 \\ F_1 & n = 1 \\ F_n \vee (F_1 \vee \dots \vee F_{n-1}) & n > 1 \end{cases}$$

Definition 1.17 (konjunktive Normalform) Eine Formel F ist in konjunktiver Normalform (CNF), falls F eine Konjunktion von Klauseln ist, also $F = F_0 \wedge \dots \wedge F_n$ für Klauseln F_i .

Beispiel: $F_0 = (x \vee y \vee \neg z) \wedge (\neg z \vee y) \wedge x$ ist in CNF.

Lemma 1.18 Zu jeder Formel F gibt es eine äquivalente Formel in CNF.

Anmerkungen:

1. Mengenschreibweise für Formeln in CNF:

Klausel $\hat{=}$ Menge von Literalen, Formel in CNF $\hat{=}$ Menge von Klauseln.

$$F_0 = \{\{x, y, \neg z\}, \{\neg z, y\}, \{x\}\}$$

2. Wir lassen zwei Spezialfälle zu:

- (a) $F = \{\}$ entsprechend der leeren Konjunktion, interpretiert als \top .
- (b) $\emptyset \in F : \square$ leere Klausel

Beweisskizze zum Lemma 1.18:

Algorithmus für CNF: (für vereinfachte Formeln in NNF)
Termersetzungssystem:

$$F \vee (G \wedge H) \rightsquigarrow (F \vee G) \wedge (F \vee H)$$

$$(G \wedge H) \vee F \rightsquigarrow (G \vee F) \wedge (H \vee F)$$

Anmerkungen:

1. CNF ist im Allgemeinen nicht eindeutig bestimmt. (D.h. CNF ist keine kanonische Form.)
2. Es gibt Formeln, deren kleinste äquivalente CNF exponentiell in der Größe der Ursprungsformel ist. $C_n = (x_{11} \wedge x_{12}) \vee \dots \vee (x_{n1} \wedge x_{n2}) \rightsquigarrow$ CNF enthält 2^n Klauseln mit je n Variablen.

Definition 1.19 (Erfüllbarkeitsäquivalenz) Seien $F, G \in \Phi$. F und G heißen erfüllbarkeitsäquivalent (in Zeichen: $F \stackrel{\text{SAT}}{\equiv} G$), falls F erfüllbar ist gdw. G erfüllbar ist.

Beispiel: $x \wedge y \stackrel{\text{SAT}}{\equiv} x \stackrel{\text{SAT}}{\equiv} z$

Verfahren von Tseitin zur Generierung von CNF: (Termersetzungssystem)

$$F \vee (G \wedge H) \rightsquigarrow (F \vee x) \wedge (G \vee \neg x) \wedge (H \vee \neg x)$$

$$(G \wedge H) \vee F \rightsquigarrow (F \vee x) \wedge (G \vee \neg x) \wedge (H \vee \neg x)$$

wobei x neue Variable ($x \notin \text{Var}(F) \cup \text{Var}(G) \cup \text{Var}(H)$).

Idee: Verwende x als Abkürzung für $G \wedge H$. Dann

$$F \vee (G \wedge H) \stackrel{\text{SAT}}{\equiv} (x \Leftrightarrow G \wedge H) \wedge (F \vee x)$$

$$\begin{aligned} &\equiv \underbrace{(\neg x \vee G) \wedge (\neg x \vee H)}_{\Rightarrow} \wedge \underbrace{(\neg G \vee \neg H \vee x)}_{\Leftarrow} \wedge (F \vee x) \\ &\stackrel{\text{SAT}}{\equiv} (\neg x \vee G) \wedge (\neg x \vee H) \wedge (F \vee x) \end{aligned}$$

Zu zeigen ist unter Anderem:

$$(\neg x \vee G) \wedge (\neg x \vee H) \wedge (\neg G \vee \neg H \vee x) \wedge (F \vee x) \stackrel{\text{SAT}}{\equiv} (\neg x \vee G) \wedge (\neg x \vee H) \wedge (F \vee x)$$

Beweis:

„ \Rightarrow “: trivial

„ \Leftarrow “: (Bem: Schreibe $\models_{\mu} F$, falls F durch die Belegung μ erfüllt wird.)

Sei μ' eine Belegung mit $\models_{\mu'} (\neg x \vee G) \wedge (\neg x \vee H) \wedge (F \vee x)$.

1. Sei $\mu'(x) = 1$.
Dann $\models_{\mu'} H$. Also erfüllt μ' die obige linke Seite ebenfalls.
2. Sei $\mu'(x) = 0$.
Dann $\models_{\mu'} F$. Jetzt wähle neue Belegung μ'' , die bis auf x mit μ' übereinstimmt. Falls $\models_{\mu'} G \wedge H$, dann wähle $\mu''(x) = 1$, sonst $\mu''(x) = 0$.
Das geht, da $\{x\} \cap (Var(F) \cup Var(G) \cup Var(H)) = \emptyset$.

Anmerkung: Tseitins Verfahren liefert erfüllbarkeitsäquivalente Formeln, die linear in der Größe der Ausgangsformeln sind.

Disjunktive Normalform (DNF)

CNF hat Klauseln, DNF hat Minterme. DNF dual zu CNF.

Algorithmus für DNF:

$$DNF(F) = CNF^{\delta}(F^{\delta})$$

Beispiel: CNF \rightarrow DNF - Transformation kann zu exponentiellem Wachstum führen:

Sei $F = (x_0^1 \vee x_1^1) \wedge (x_0^2 \vee x_1^2) \wedge \dots \wedge (x_0^n \vee x_1^n)$ in CNF.

F repräsentiert die Randbedingung um in n-bit Vektor die Bits zu setzen.

Die DNF zählt dann alle Möglichkeiten auf:

$$\begin{aligned} F_{DNF} &= (x_0^1 \wedge x_0^2 \wedge \dots \wedge x_0^n) \vee \\ &\quad (x_0^1 \wedge x_0^2 \wedge \dots \wedge x_1^n) \vee \\ &\quad \vdots \\ &\quad (x_1^1 \wedge x_1^2 \wedge \dots \wedge x_1^n) \end{aligned}$$

Anmerkung: Das Problem der Erfüllbarkeit für aussagenlogische Formeln in CNF (SAT) ist NP-vollständig (Bewiesen von Cook, ca. 1970). Es ist das erste Problem, dessen NP-Vollständigkeit bewiesen wurde.

2 Aussagenlogische Entscheidungsverfahren

2.1 Resolution

- Rein syntaktisches Verfahren (\rightarrow Kalkül) zur Widerlegung (Feststellung der Unerfüllbarkeit) einer Formel
- Ursprünglich für die Prädikatenlogik entwickelt von J. A. Robinson (1965).
- arbeitet auf Formeln in CNF.
- nur 1 Ableitungsregel: Resolutionsregel

Definition 2.1 (Resolutionsregel) Sei C eine Klausel mit Literal l und D eine Klausel mit Literal $\neg l$. Dann ist die Resolutionsregel anwendbar:

$$\frac{C \quad D}{(C \setminus \{l\}) \cup (D \setminus \{\neg l\})} \text{Res}$$

$E = (C \setminus \{l\}) \cup (D \setminus \{\neg l\})$ heißt Resolvente von C und D . Man sagt auch, E ist durch Resolution über l (bzw. $\neg l$) aus C und D entstanden. (Weitere Schreibweise: $C, D \vdash_{\text{Res}}^1 E$.)

Anmerkungen

1. $\neg l$ heißt zu l komplementäres Literal.
2. C, D werden Elternklauseln genannt.
3. Alternative Darstellung als Baum:

Definition 2.2 (Resolutionsherleitung) Sei $F = \{C_1, \dots, C_n\}$ eine Formel in CNF und D eine Klausel. Eine Folge E_1, \dots, E_k ist eine Herleitung von D aus F , falls $E_k = D$ und für alle E_i ($1 \leq i \leq k$) existieren Klauseln $A, B \in F \cup \{E_1, \dots, E_{i-1}\}$ mit $A, B \vdash_{\text{Res}}^1 E_i$. D heißt dann auch per Resolution aus F herleitbar, in Zeichen: $F \vdash_{\text{Res}} D$.

Beispiel: Es gilt $F_1 \vdash_{\text{Res}} \{x\}$ aus obigem Beispiel

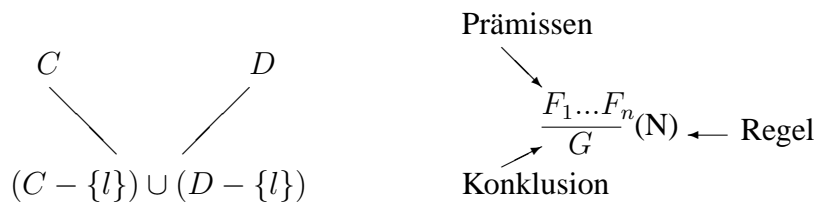
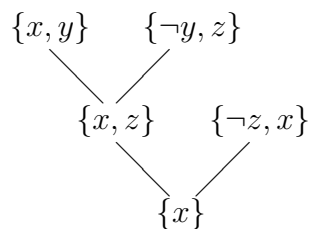


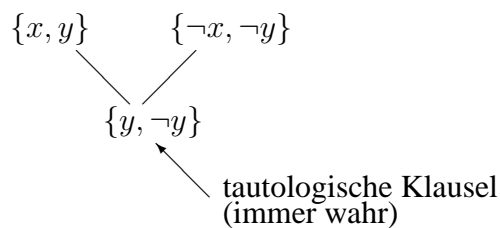
Abbildung 2: Resolutionskalkül

Beispiele:

$$F_1 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$$



$$F_2 = \{\{x, y\}, \{\neg x, \neg y\}\}$$



Definition 2.3 (Resolutionsabschluss) Der Resolutionsabschluss $\text{Res}^*(F)$ einer Formel F in CNF ist definiert durch:

$$\text{Res}^0(F) = F$$

$$\text{Res}^1(F) = F \cup \{E \mid E \text{ ist (nicht-tautologische)}$$

$$\text{Resolvente zweier Klauseln } C, D \in F\}$$

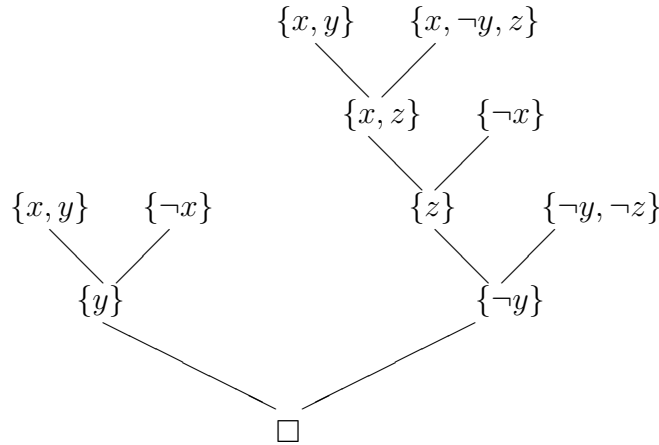
$$\text{Res}^{n+1}(F) = \text{Res}^1(\text{Res}^n(F)) \quad \text{für } n \geq 1.$$

$$\text{Res}^*(F) = \bigcup_{n \geq 0} \text{Res}^n(F)$$

Der Resolutionsabschluss Res^* enthält alle möglichen per Resolution aus F herleitbaren Klauseln.

Beispiel:

$$F_3 = \{\{x, y\}, \{x, \neg y, z\}, \{\neg x\}, \{\neg y, \neg z\}\}$$



Satz 2.4 1. Der Resolutionskalkül ist korrekt (engl. sound). Das heißt es gilt für alle Formeln $F \in \Phi$ und für alle Klauseln D :

$$F \vdash_{Res} D \text{ impliziert } F \models D.$$

2. Der Resolutionskalkül ist widerlegungsvollständig, das heißt für alle $F \in \Phi$ gilt: Falls F unerfüllbar, so gilt $F \vdash_{Res} \square$.

Anmerkung: Vollständigkeit, d.h. $F \models D$ impliziert $F \vdash_{Res} D$ gilt nicht. Z.B. $F = x, D = x \vee y$.

F in Klauselform: $\{\{x\}\}$. Hieraus lässt sich $\{\{x, y\}\}$ per Resolution nicht herleiten.

Beweisskizze zu Satz 2.4:

1. Für $C, D \vdash_{Res}^1 E$ ist $\{C, D\} \models E$ durch einfaches Nachrechnen zu überprüfen.

Allgemeiner Fall: Induktion über Länge der Herleitung.

2. (Widerlegungsvollständigkeit)

Induktion über Länge der zu widerlegenden Formel:

Induktionsanfang: $\{x\}, \{\neg x\}$. Klar.

Induktionsschritt: F unerfüllbar $\Rightarrow F \vdash_{Res} \square$. Betrachte Restriktionen

$F \mid_{x=0}, F \mid_{x=1}$ für eine beliebige Variable x (beide sind nach Voraussetzung unerfüllbar) und deren Resolutionsherleitungen von \square . Diese Herleitungen lassen sich leicht zu Herleitungen von $\{x\}$ bzw. $\{\neg x\}$ ergänzen (sofern sich nicht direkt die leere Klausel \square herleiten lässt). Dann liefert $\frac{\{x\}, \{\neg x\}}{\square} Res$ die gewünschte Widerlegung.

Varianten/Einschränkungen der Resolution

a) Unit-Resolution (\vdash_{URes}):

Mindestens eine Elternklausel ist Unit-Klausel (Klausel, die nur ein Literal enthält).

Anmerkung:

- \vdash_{URes} ist widerlegungsvollständig für Hornklauseln.
- \vdash_{URes} ist in linearer Zeit entscheidbar.

b) Geordnete Resolution (\vdash_{ORes}):

Voraussetzung: Strikte, totale Ordnung \prec auf Variablen Φ_0 .

Resolution ist eingeschränkt: Literal, über das resolviert wird, muss in jeder Elternklausel maximal sein.

$x \prec y$ heißt auch $x \prec \neg y$

Anmerkung: \vdash_{ORes} ist widerlegungsvollständig für beliebige Formeln in CNF.

Beispiel:

$$F_3, x \prec y \prec z$$

$$F_3 = \{\{x, y\}, \{x, \neg y, z\}, \{\neg x\}, \{\neg y, \neg z\}\}$$

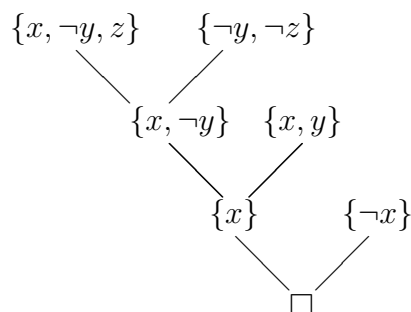


Abbildung 3: Beispiele zur geordneten Resolution

Definition 2.5 (Subsumtion) Seien C und D Klauseln. C subsumiert D , falls $C \subseteq D$, d.h. falls jedes Literal von C auch in D vorkommt.

Lemma 2.6 Falls C subsummiert D , dann gilt $C \models D$.

Anmerkung: Zur Erfüllbarkeitsprüfung in CNF ist es ausreichend, nur solche Klauseln zu betrachten, die von keiner anderen subsumiert werden. \rightsquigarrow subsumierte Klauseln können gelöscht werden.

Beispiel:

$$\{x\} \text{ subsumiert } \{x, y\}$$

$$\{\{x\}, \{x, y\}\} \equiv \{\{x\}\}$$

2.2 Davis–Putnam Algorithmus

- In den 60er Jahren vorgestelltes Verfahren zur Erfüllbarkeitsprüfung von Formeln in CNF.
- Große Verbreitung zur Lösung aussagenlogischer Probleme, z.B. Hardware-Verifikation (BMC), Protokoll-Verifikation, Konfiguration

Grundidee:

Probiere Belegung zu konstruieren, Variable für Variable; backtracking; unit-propagation

```

(1)  boolean DPLL(ClauseSet  $S$ )
(2)  { 1. Bereinige die Klauselmenge
(3)    while (  $S$  contains a unit clause  $\{l\}$  ) {
(4)      delete from  $S$  clauses containing  $l$ ; // unit-subsumption
(5)      delete  $\bar{l}$  from all clauses in  $S$ ; // unit-resolution
(6)    }
(7)    2. Trivialfall?
(8)    if (  $\square \in S$  ) return false; // constraint unerfüllbar
(9)    if (  $S = \emptyset$  ) return true; // nichts mehr zu erfüllen
(10)   3. Fallunterscheidung
(11)   choose a literal  $l$  occurring in  $S$ ; // Heuristik!
(12)   if ( DPLL( $S \cup \{\{l\}\}$ ) ) return true; // first branch
(13)   else if ( DPLL( $S \cup \{\{\bar{l}\}\}$ ) ) return true; // second branch
(14)   else return false;
(15) }

```

Abbildung 4: Davis-Putnam-(Logemann-Loveland-)Algorithmus.

Anmerkung: Die unit-resolution und die unit-subsumption fasst man oft zusammen zu der unit propagation.

Beispiel:

(a) $S = F_3 = \{\{\neg x\}, \{x, y\}, \{x, \neg y, z\}, \{\neg y, \neg z\}\}$
unit propagation mit $l = \neg x$: Klausel $\{\neg x\}$ wird gestrichen, x wird aus allen Formeln gestrichen. $\rightsquigarrow S = \{\{y\}, \{\neg y, z\}, \{\neg y, \neg z\}\}$
unit propagation mit $l = y$ $\rightsquigarrow S = \{\{z\}, \{\neg z\}\}$
unit propagation mit $l = z$ $\rightsquigarrow S = \{\square\}$
 $\square \in S$: false $\implies F_3$ ist unerfüllbar.

(b) $S = \{\{x, y, z\}, \{\neg x, y, z\}, \{\neg x\}, \{z, \neg y\}\}$
unit propagation mit $l = \neg x$ $\rightsquigarrow \{\{y, z\}, \{z, \neg y\}\}$
Fallunterscheidung:
Fall 1: $S \cup \{\{y\}\}$
 $S = \{\{y, z\}, \{z, \neg y\}, \{y\}\}$
unit propagation mit $l = y$ $\rightsquigarrow S = \{\{z\}\}$
unit propagation mit $l = z$ $\rightsquigarrow S = \emptyset$
 $S = \emptyset \implies$ return true
 S ist erfüllbar. Modell von S : $\beta(x) = 0, \beta(y) = \beta(z) = 1$

Fall 2: $S \cup \{\{\neg y\}\}$
wird nicht mehr untersucht.

Anmerkung:

1. Ursprünglich nach der while-Schleife der Zeilen (3) bis (6) zusätzliche Vereinfachung: *pure literal rule*:
Falls ein Literal l nur positiv (oder nur negativ) in S vorkommt, so können alle Klauseln, in denen es vorkommt gelöscht werden. Denn oBdA kann $\beta(l) = 1$ gewählt werden.
2. Für das zu l komplementäre Literal schreiben wir auch \bar{l} , d. h. $\bar{x} = \neg x$ und $\neg\bar{x} = x$ für alle $x \in \Phi_0$

Korrektheit des Davis–Putnam-Algorithmus: $S \upharpoonright_l := \{C - \{\bar{l}\} \mid C \in S, l \notin C\}$ wobei S Klauselmenge und l Literal.

Vereinfachung von S durch Setzen von l auf 1, entsprechend Zeilen (4) und (5) des Algorithmus.

Lemma 2.7 1. Falls $\{l\} \in S$, so ist S erfüllbar gdw. $S \upharpoonright_l$ erfüllbar ist.

2. S ist erfüllbar, gdw. $S \cup \{\{l\}\}$ oder $S \cup \{\{\bar{l}\}\}$ erfüllbar ist für beliebiges Literal l .

Beweis:

1. „ \Rightarrow “ Sei β_0 eine erfüllende Belegung (Modell) von S . Also muss auch $\beta(l) = 1$ gelten (da $\{l\} \in S$) und $\beta(\bar{l}) = 0$. Dann ist $\beta(C) = \beta(C \setminus \{\bar{l}\})$ für alle $C \in S$. Und somit ist β_0 auch Modell von $S \upharpoonright_l$.

„ \Leftarrow “ Sei β_0 ein Modell von $S \upharpoonright_l$ und $\{l\} \in S$. Erweiterung von β_0 zu β'_0 , so dass $\beta'_0(l) = 1$ ist immer möglich, da weder l noch \bar{l} in $S \upharpoonright_l$ vorkommt. Da $\beta(D) = 1$ für alle $D \in S \upharpoonright_l$, gilt natürlich auch $\beta'(D \cup \{\bar{l}\}) = 1$. Außerdem ist $\beta'(E) = 1$ für alle E mit $l \in E$. Also ist β'_0 ein Modell von S .

2. Übung.

Zur Termination: Die Anzahl n der in S vorkommenden Variablen (vor Schritt 2) nimmt bei jedem rekursiven Aufruf ab, so dass letztendlich $\square \in S$ oder $S = \emptyset$ gelten muss.

Lernen im Davis–Putnam–Algorithmus:

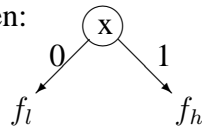
- Wiederholte Bearbeitung von Teilen des Suchbaums, die aufgrund derselben Ursachen keine Lösung enthalten, wird vermieden.
- Neue Klauseln werden generiert (\rightarrow zusätzliches Wissen über die Probleminstanz).
- Backtracking (Untersuchung des zweiten Falls) kann zum Teil unterbleiben (\rightarrow nichtchronologisches Backtracking).

2.3 Binäre Entscheidungsdiagramme (BDDs)

- Graphen–basierter Formalismus zur eindeutigen Darstellung boolescher Funktionen: $f : B^n \rightarrow B$, $B = \{0, 1\}$.
- In der heutigen Form von Bryant 1986 entwickelt.
- Aussagenlogische Formeln repräsentiert als gerichteter, azyklischer Graph (DAG, „directed acyclic graph“):

Terminalknoten: $\boxed{0}$, $\boxed{1}$ (entsprechend \perp und \top)

innere Knoten:



- $x \in \Phi_0$
- zwei Nachfolger: f_l und f_h (über 0– bzw. 1–Kanten)
interpretiert als $(\neg x \Rightarrow f_l) \wedge (x \Rightarrow f_h)$.
„if — then — else“: if x then f_h else f_l .

Abbildung 5: BDDs

Definition 2.8 (Restriktion) Sei F eine aussagenlogische Formel, $x \in \Phi_0$ und

$b \in \{0, 1\}$. Die Restriktion $F|_{x=b}$ ist dann rekursiv definiert durch:

$$\begin{aligned} \perp|_{x=b} &= \perp \\ y|_{x=b} &= \begin{cases} \top & \text{falls } x = y \text{ und } b = 1, \\ \perp & \text{falls } x = y \text{ und } b = 0, \\ y & \text{sonst.} \end{cases} \\ (\neg G)|_{x=b} &= \neg(G|_{x=b}) \\ (G \vee H)|_{x=b} &= G|_{x=b} \vee H|_{x=b} \\ (G \wedge H)|_{x=b} &= G|_{x=b} \wedge H|_{x=b} \end{aligned}$$

Shannon–Expansion: (einer Funktion f , dargestellt als Formel)

$$\begin{aligned} f &\equiv (\neg x \Rightarrow f|_{x=0}) \wedge (x \Rightarrow f|_{x=1}) \\ &\equiv (x \vee f|_{x=0}) \wedge (\neg x \vee f|_{x=1}) \\ &\equiv (x \wedge f|_{x=1}) \vee (\neg x \wedge f|_{x=0}) \end{aligned}$$

$\longrightarrow f_l \hat{=} f|_{x=0}, f_h \hat{=} f|_{x=1}$

(Reduced) Ordered BDD (ROBDD):

- Variablenordnung \prec (strikt, total)
- BDDs gebildet durch Shannon–Expansion
- Variablen eines Kindknotens immer kleiner als Variablen des Elternknotens (bezüglich \prec)
- (in Implementationen: Knoten eindeutig dargestellt \longrightarrow DAG)
- reduziert mittels Vereinfachungsregel:

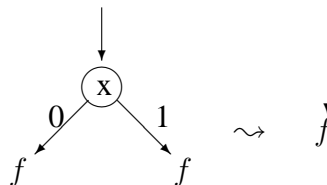


Abbildung 6: Vereinfachungsregel

Beispiel: $f = (x \vee y) \wedge (x \vee \neg y \vee \neg z) \wedge \neg z$ Variablenordnung: $z \prec y \prec x$

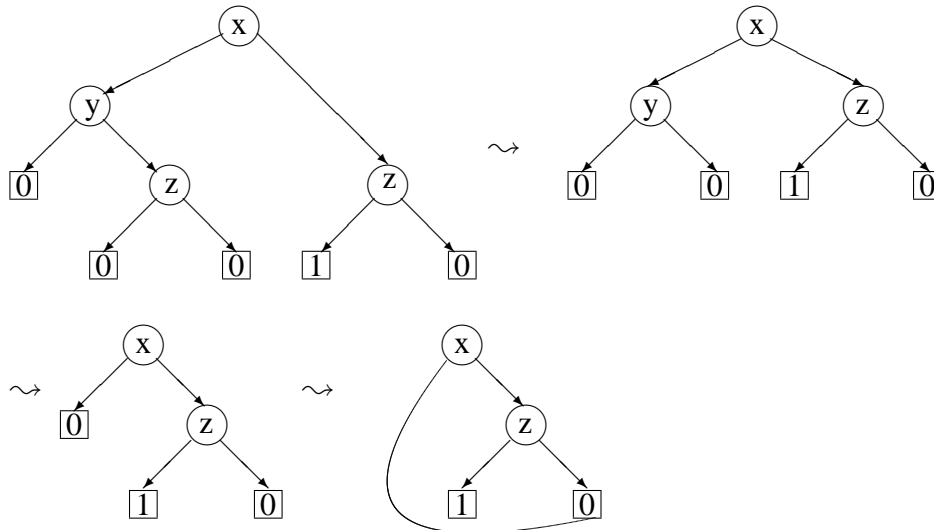


Abbildung 7: Beispiel für Reduktion von BDDs

Lemma 2.9 (Canonicity) Gegeben sei eine boolesche Funktion $f : \mathbb{B}^n \rightarrow \mathbb{B}$ und eine Variablenordnung $x_1 < x_2 < \dots < x_n$. Dann gibt es genau einen ROBDD u mit $f^u = f(x_1, \dots, x_n)$. (Dabei bezeichnet f^u , die von u repräsentierte Funktion)

Konsequenzen:

- $f \equiv \top$ gdw. $\text{ROBDD}(f) = \boxed{1}$
- $f \equiv \perp$ gdw. $\text{ROBDD}(f) = \boxed{0}$
- f erfüllbar gdw. $\text{ROBDD}(f) \neq \boxed{0}$
- $f \equiv g$ gdw. $\text{ROBDD}(f) = \text{ROBDD}(g)$

Konstruktion und Manipulation von ROBDDs

1. Aufbau eines ODT (ordered decision tree)
Sei t ein boolescher Ausdruck in n Variablen

$\text{Build}(t,n) \equiv \text{Build}'(t,1,n)$

$\text{Build}'(t,i,n) \equiv$

if $(i > n)$ then if t then return 1 else return 0 fi
 else $\text{MKNod}(i, \text{Build}'(t[0/x_i], i + 1, n), \text{Build}'(t[1/x_i], i + 1, n))$
 fi

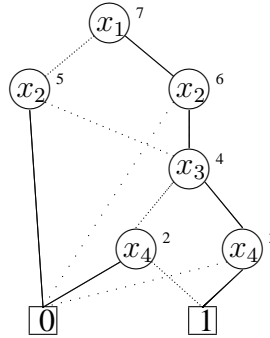
2. Aufbau eines ROBDDs

Jeder Knoten darf nur ein einziges Mal konstruiert werden.

Wir benutzen eine Tabellenrepräsentation, um existierende Knoten schnell zu finden.

Tabelle T der Knoten bzw. des Graphen:

node	Var	l	h
#	#		
0	n+1		
1	n+1		
2	4	1	0
3	4	0	1
4	3	2	3
5	2	4	0
6	2	0	4
7	1	5	6



Operationen auf T :

$init(T)$: füge 0 und 1 in leeres T ein

$u \leftarrow add(i, l, h)$: neuer Knoten

$var(u), low(u), high(u)$: accessors

Zum Finden existierender Knoten benutzen wir eine inverse Tabelle H :

$init(H)$: leeres H anlegen

$b \leftarrow member(i, l, h)$: ist (i, l, h) in H vorhanden?

$u \leftarrow lookup(i, l, h)$: finde $H(i, l, h)$

$insert(i, l, h, u)$: repräsentiere Abb. $(i, l, h) \mapsto u$ durch Eintrag in H

MakeNode für ROBDD:

$MkNod[T, H](i, l, h) ::=$

if $(l = h)$ **then return** l

else if $member(i, l, h)$ **then return** $lookup(i, l, h)$

else $u = add(i, l, h);$

$insert(i, l, h, u);$

return $u;$

fi

fi

Apply

$Apply(op, u_1, u_2)$ berechnet zu den ROBDDs u_1 und u_2 das ROBDD für

den Ausdruck $t^{u_1} \text{ op } t^{u_2}$. (Dabei ist t^u der zu BDD u gehörige Ausdruck.)
 Grundlagen:

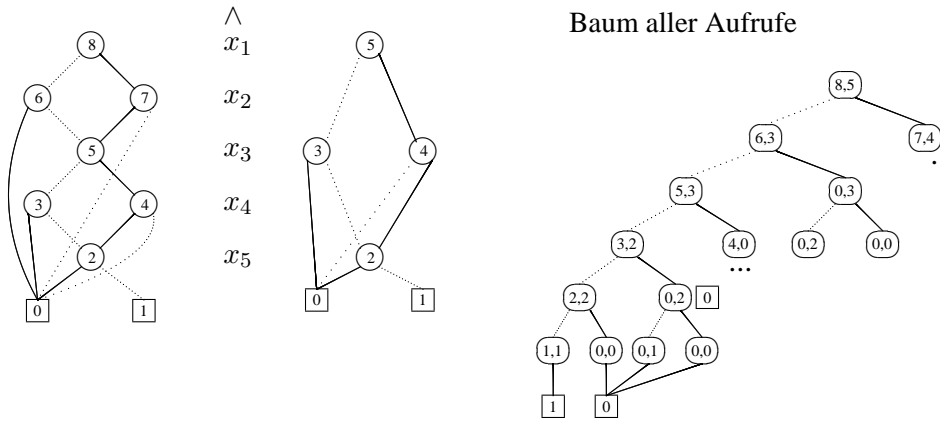
- (1) **(if x then t_1 else fi t_2) op (if x then t'_1 else fi t'_2)** \equiv
if x then ($t_1 \text{ op } t'_1$) else ($t_2 \text{ op } t'_2$) fi
 für alle booleschen Operatoren
- (2) **(if x_i then t_1 else t_2 fi) op t** \equiv **if x_i then $t_1 \text{ op } t$ else $t_2 \text{ op } t$ fi**
 gilt immer

$Apply[T, H](\text{op}, u_1, u_2) \equiv \text{init}(G); App(u_1, u_2)$

```

App(u1, u2)  $\equiv$ 
  if  $G(u_1, u_2) \neq \text{empty}$  then return  $G(u_1, u_2)$ 
  else if  $u_1 \in \{0, 1\}$  and  $u_2 \in \{0, 1\}$  then  $u \leftarrow \text{op}(u_1, u_2)$ 
    else if  $\text{var}(u_1) = \text{var}(u_2)$  then
       $u \leftarrow \text{MkNod}(\text{var}(u_1), \text{App}(\text{low}(u_1), \text{low}(u_2)),$ 
         $\text{App}(\text{high}(u_1), \text{high}(u_2)))$ 
    else if  $\text{var}(u_1) < \text{var}(u_2)$  then
       $u \leftarrow \text{MkNod}(\text{var}(u_1), \text{App}(\text{low}(u_1), u_2),$ 
         $\text{App}(\text{high}(u_1), u_2))$ 
    else  $u \leftarrow \text{MkNod}(\text{var}(u_2), \text{App}(u_1, \text{low}(u_2)),$ 
       $\text{App}(u_1, \text{high}(u_2)))$ 
    fi
  fi
  fi
   $G(u_1, u_2) \leftarrow u$ 
return( $u$ )
  
```

Bsp zu *Apply*:



Existentielle Quantifizierung

$$\exists x.t \equiv t[0/x] \vee t[1/x]$$

Dabei bedeutet $t[t'/x]$: alle Vorkommisse von x in t durch t' ersetzt.

Warnung: Es wird auch $t[x/t']$ verwendet, in der Bedeutung "replace x by t' in t ".

Ebenso wird $t[x \leftarrow t']$ verwendet

Sat Count Berechnung der Anzahl erfüllender Belegungen für F aus ROBDD(F)

$$\begin{aligned} SatCount(u) &= 2^{var(low(u)) - var(u) - 1} * SatCount(low(u)) + \\ &2^{var(high(u)) - var(u) - 1} * SatCount(high(u)) \end{aligned}$$

Model Checking mit BDDs (Symbolic) Prüfe Eigenschaften eines Zustands-Übergangssystems

Idee: Repräsentiere Zustände durch Formel (implizit) $F_R(\vec{x}) = 1$, falls $x \in R$

Repräsentiere Zustandsübergänge durch Formel

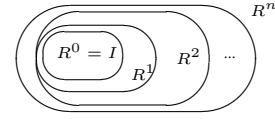
$$T(\vec{x}, \vec{x}') = \begin{cases} 1 & \text{falls } \vec{x} \rightarrow \vec{x}' \\ 0 & \text{sonst} \end{cases}$$

Erhalte Beschreibung aller erreichbaren Zustände durch Fixpunktberechnung ausgehend von Initialzustand I .

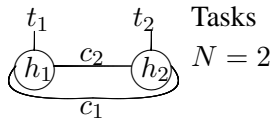
$$R = I$$

repeat

$$\begin{aligned}
& R' = R; \\
& R = R \vee (\exists \vec{x}. R(\vec{x}) \wedge T(\vec{x}, \vec{x}'))[\vec{x}/\vec{x}'] \\
& \mathbf{until} \ R = R'
\end{aligned}$$



Beispiel: Milner's Scheduler



N cyclers

N tasks sollen laufen; dürfen nur in Reihenfolge gestartet werden

\exists token h_i heißt: i hat token

Cycler mit token darf task starten

start: cycler i ändert t_i von 0 zu 1

cycler i nimmt token auf: c_i von 1 zu 0, h_i von 0 zu 1

cycler i legt token ab: c_{i+1} von 0 zu 1

Transitionen für cycler i : **if** $c_i = 1 \wedge t_i = 0$ **then** $t_i, c_i, h_i := 1, 0, 1$
if $h_i = 1$ **then** $c_{(i \bmod N)+1}, h_i := 1, 0$

In Boolescher Logik: $P_i = (c_i \wedge \neg t_i \wedge t'_i \wedge \neg c'_i \wedge h'_i) \vee (h_i \wedge c'_{(i \bmod N)+1} \wedge \neg h'_i)$

Tasks können abstürzen: **if** $t_i = 1$ **then** $t'_i = 0$
 $E_i = t_i \wedge \neg t'_i$

Initialzustand: $\neg \vec{t} \wedge \neg \vec{h} \wedge c_1 \wedge \neg c_2 \wedge \neg c_3 \wedge \dots \wedge \neg c_N$
(Dabei ist $\neg \vec{t} = \neg t_1 \wedge \neg t_2 \wedge \dots \wedge \neg t_N$)

Beispiel: $N = 2$:

$$T \begin{cases} P_1 & = (c_1 \wedge \neg t_1 \wedge t'_1 \wedge \neg c'_1 \wedge h'_1) \vee (h_1 \wedge c'_2 \wedge \neg h'_1) \\ P_2 & = (c_2 \wedge \neg t_2 \wedge \dots) \vee (\dots) \\ E_1 & = t_1 \wedge \neg t'_1 \\ E_2 & = t_2 \wedge \neg t'_2 \end{cases}$$

$$I = \neg t_1 \wedge \neg t_2 \wedge \neg h_1 \wedge \neg h_2 \wedge c_1 \wedge \neg c_2$$

$$\begin{aligned}
R^0 &= I \\
R' &= R = I \\
R &= I \vee (\exists \vec{x}. T \wedge R)[\vec{x}/\vec{x}'] \\
&= I \vee (\exists \vec{x}. \bigvee_{i=1}^2 (P_i \vee E_i) \wedge I)[\vec{x}/\vec{x}'] \\
&= I \vee (t'_1 \wedge \neg c'_1 \wedge h'_1)[\vec{x}/\vec{x}'] \\
&= I \vee (t_1 \wedge \neg c_1 \wedge h_1) \\
&= R^1
\end{aligned}$$

3 Prädikatenlogik (PL1)

Im Gegensatz zur Aussagenlogik sind hier die elementaren Aussagen nicht atomar, sondern können aus Relationen, Funktionen und Variablen zusammengesetzt sein. Die Prädikatenlogik erster Stufe (PL1) erlaubt die Beschreibung mathematischer Strukturen (z.B. Gruppen, Körper, ...) und die Bezugnahme auf einzelne enthaltene Elemente („Formalisierte Sprache der Mathematik“).

3.1 Syntax

Die Sprache der PL1 besteht aus Symbolen für

- (Element-, Individuen-) Variablen
 $v_0, v_1, v_2, \text{ etc.}$
- aussagenlogische Junktoren:
 $\vee, \wedge, \neg, \perp$
- Quantoren:
universeller Quantor („für alle“): \forall
existentieller Quantor („es gibt ein“): \exists
- Klammern: $(,)$
- Relationssymbole: R, S, T, \dots (jeweils mit Stelligkeit $n \geq 0$)
- Funktionssymbole: f, g, h, \dots (jeweils mit Stelligkeit $n \geq 0$)

Anmerkung: Wir schreiben die Stelligkeiten z.T. als Index, also z.B. f_2 für 2-stelliges f .

Definition 3.1 (PL1-Term) Sei \mathcal{V} eine Menge von (Element-)Variablen, \mathcal{F} eine Menge von Funktionssymbolen. Die Menge $T(\mathcal{V}, \mathcal{F})$ (oder einfach auch T) der Terme (über \mathcal{V} und \mathcal{F}) ist definiert als die kleinste Menge mit

1. $\mathcal{V} \subseteq T$.
2. Falls $f \in \mathcal{F}$ (mit Stelligkeit n), und $t_1, \dots, t_n \in T$, so auch $ft_1 \dots t_n \in T$.

Definition 3.2 (PL1-Formel) Sei \mathcal{V} eine Menge von Variablen, \mathcal{F} eine Menge von Funktionssymbolen und \mathcal{R} eine Menge von Relationsymbolen. Die Menge $\Phi(\mathcal{V}, \mathcal{F}, \mathcal{R})$ (oder einfach Φ) der PL1-Formeln (über \mathcal{V} , \mathcal{F} und \mathcal{R}) ist dann definiert als die kleinste Menge mit

1. $\perp \in \Phi$.
2. Falls $R \in \mathcal{R}$ (mit Stelligkeit n) und t_1, \dots, t_n in $T(\mathcal{V}, \mathcal{F})$, so ist $Rt_1 \dots t_n \in \Phi$.
3. Falls $F, G \in \Phi$, so auch $(F \vee G) \in \Phi$, $(F \wedge G) \in \Phi$ und $\neg F \in \Phi$.
4. Falls $x \in \mathcal{V}$ und $F \in \Phi$, so auch $\exists x F \in \Phi$ und $\forall x F \in \Phi$.

Anmerkung:

- Für einen Term t bezeichnet $Var(t)$ die Menge der im Term auftretenden Elementarvariablen.
- Präzedenzregeln zum Einsparen von Klammern wie üblich.

Beispiel:

$$F = \forall x (\exists y Pxy \wedge Qfxgxy)$$

$$\mathcal{V} = \{x, y\}, \mathcal{F} = \{f_2, g_1\}, \mathcal{R} = \{P_2, Q_2\}$$

Beispiel: (freie Gruppe)

$\exists x \forall y : x \cdot y = y$ “Eins“ (entspricht: $\exists x \forall y = \cdot xy y$)

$\forall y \exists x : x \cdot y = 1$ “inverses Element“

$\forall x \text{ forally for all } z : x \cdot (y \cdot z) = (x \cdot y) \cdot z$ “Assoziativgesetz“

$\mathcal{V} = \{x, y, z\}, \mathcal{F} = \{\cdot, 1_0\}, \mathcal{R} = \{=, _2\}$

$\forall y \forall y : (x = y) \Rightarrow (y = x)$ “symmetrisch“

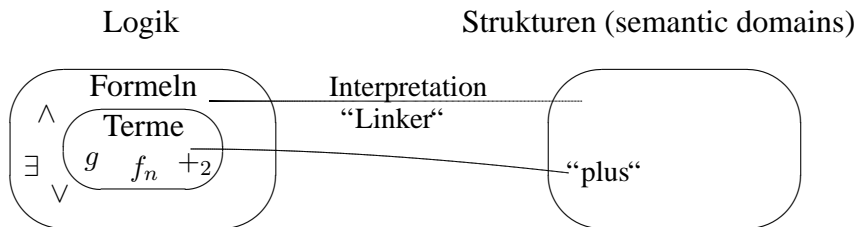
$\forall x : (x = x)$ “reflexiv“

$\forall x \forall y \forall z : ((x = y) \wedge (y = z)) \Rightarrow (x = z)$ “transitiv“

(Peanos Axiomen - natürliche Zahlen)

$$\begin{array}{ll} x + 0 = x & \exists y \forall x : x + y = x \\ x + S(y) = S(x + y) & \forall x \forall y : x + S(y) = S(x + y) \\ x \cdot 1 = x & \exists y \forall x : x \cdot y = x \\ x \cdot S(y) = (x \cdot y) + x & \forall x \forall y : x \cdot S(y) = (x \cdot y) + x \end{array}$$

3.2 Semantik



Definition 3.3 ((\mathcal{F}, \mathcal{R})-Struktur) Sei \mathcal{F} eine Menge von Funktionssymbolen und \mathcal{R} eine Menge von Relationssymbolen. Eine $(\mathcal{F}, \mathcal{R})$ -Struktur ist ein Tupel $\mathcal{A} = (A, a)$, wobei $A \neq \emptyset$ eine Menge, das Universum von \mathcal{A} , ist und a eine Funktion, die jedem $f \in \mathcal{F}$ (der Stelligkeit n) eine n -stellige Funktion $a(f) : A^n \rightarrow A$ und jedem $R \in \mathcal{R}$ (der Stelligkeit n) eine n -stellige Relation $a(R) \subseteq A^n$ zuordnet.

Beispiel:

$$\begin{aligned} \mathcal{A} &= (\mathbb{N}, a) \\ a(f) : \mathbb{N} \times \mathbb{N} &\rightarrow \mathbb{N} : (x, y) \mapsto x + y \\ a(g) : \mathbb{N} &\rightarrow \mathbb{N} : x \mapsto x + 1 \\ a(P) &\subseteq \mathbb{N}^2 : (x, y) \in a(P) \Leftrightarrow x = y \\ a(Q) &\subseteq \mathbb{N}^2 : (x, y) \in a(Q) \Leftrightarrow x < y \end{aligned}$$

Definition 3.4 (Variablenbelegung (in einer Struktur)) Sei \mathcal{V} eine Variablenmenge und $\mathcal{A} = (A, a)$ eine $(\mathcal{F}, \mathcal{R})$ -Struktur. Eine Variablenbelegung (für \mathcal{V} in \mathcal{A}) ist eine Abbildung $\beta : \mathcal{V} \rightarrow A$. Für eine Belegung β , $a \in A$ und $x \in \mathcal{V}$ schreiben wir $\beta[x/a]$ für die an der Stelle x auf a abgeänderte Funktion β , also

$$\beta[x/a](y) = \begin{cases} a & \text{falls } y = x, \\ \beta(y) & \text{sonst} \end{cases}$$

Definition 3.5 (Interpretation) Eine Interpretation \mathcal{I} (zu gegebenen \mathcal{V} , \mathcal{F} und \mathcal{R}) ist ein Tupel (\mathcal{A}, β) bestehend aus einer $(\mathcal{F}, \mathcal{R})$ -Struktur \mathcal{A} und einer Variablenbelegung β für \mathcal{V} in \mathcal{A} .

Definition 3.6 (Interpretation eines Terms) Sei $\mathcal{I} = (\mathcal{A}, \beta)$ eine Interpretation. Für einen Term t definieren wir dessen Interpretation $\mathcal{I}(t)$ (in \mathcal{A}) rekursiv durch:

- $\mathcal{I}(t) = \beta(t)$ falls $t \in \mathcal{V}$.
- $\mathcal{I}(ft_1 \dots t_n) = a(f)(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n))$.

Beispiel: (zu Definition 3.6)

$$\mathcal{I}(fx, gx) = +(2, +_1(2)) = 2 + (2 + 1) = 5$$

bezüglich obigem \mathcal{A} und $\beta(x) = 2, a(f) = +, a(g) = +_1 = +(x, 1)$

Definition 3.7 (Erfüllbarkeitsrelation) Sei $\mathcal{I} = (\mathcal{A}, \beta)$ eine Interpretation. Wir definieren die Erfüllbarkeitsrelation $\mathcal{I} \models F$ für Formeln F durch:

$$\begin{aligned} \mathcal{I} \not\models \perp \\ \mathcal{I} \models Rt_1 \dots t_n & \text{ gdw. } a(R)(\mathcal{I}(t_1), \dots, \mathcal{I}(t_n)) \\ \mathcal{I} \models F \vee G & \text{ gdw. } \mathcal{I} \models F \text{ oder } \mathcal{I} \models G \\ \mathcal{I} \models F \wedge G & \text{ gdw. } \mathcal{I} \models F \text{ und } \mathcal{I} \models G \\ \mathcal{I} \models \neg F & \text{ gdw. } \mathcal{I} \not\models F \\ \mathcal{I} \models \forall x F & \text{ gdw. } \mathcal{I}[x/a] \models F \text{ für alle } a \in A \\ \mathcal{I} \models \exists x F & \text{ gdw. es gibt ein } a \in A \text{ mit } \mathcal{I}[x/a] \models F \end{aligned}$$

Wir sagen ‘ \mathcal{I} erfüllt F ’ oder ‘ F gilt in \mathcal{I} ’ oder ‘ \mathcal{I} ist ein Modell von F ’, falls $\mathcal{I} \models F$. Falls es eine Interpretation \mathcal{I} mit $\mathcal{I} \models F$ gibt, so heißt F erfüllbar. Falls $\mathcal{I} \models F$ für alle Interpretationen \mathcal{I} (mit passendem \mathcal{V} , \mathcal{F} und \mathcal{R}), so heißt F allgemeingültig (in Zeichen: $\models F$).

Anmerkung: (zu Definition 3.7) Ist $\mathcal{I} = (\mathcal{A}, \beta)$ eine Interpretation, so schreiben wir $\mathcal{I}[x/a]$ für das Tupel $(\mathcal{A}, \beta[x/a])$.

3.3 Substitutionen

Definition 3.8 (freie/gebundene Variablen) Die Menge der freien Variablen $\text{Fr}(F)$ und der gebundenen Variablen $\text{Bd}(F)$ einer Formel $F \in \Phi$ ist rekursiv definiert wie folgt:

$$\begin{array}{ll}
 \text{Fr}(\perp) = \emptyset & \text{Bd}(\perp) = \emptyset \\
 \text{Fr}(Rt_1 \dots t_n) = \text{Var}(t_1) \cup \dots \cup \text{Var}(t_n) & \text{Bd}(Rt_1 \dots t_n) = \emptyset \\
 \text{Fr}(F \vee G) = \text{Fr}(F) \cup \text{Fr}(G) & \text{Bd}(F \vee G) = \text{Bd}(F) \cup \text{Bd}(G) \\
 \text{Fr}(F \wedge G) = \text{Fr}(F) \cup \text{Fr}(G) & \text{Bd}(F \wedge G) = \text{Bd}(F) \cup \text{Bd}(G) \\
 \text{Fr}(\neg F) = \text{Fr}(F) & \text{Bd}(\neg F) = \text{Bd}(F) \\
 \text{Fr}(\exists x F) = \text{Fr}(F) \setminus \{x\} & \text{Bd}(\exists x F) = \text{Bd}(F) \cup \{x\} \\
 \text{Fr}(\forall x F) = \text{Fr}(F) \setminus \{x\} & \text{Bd}(\forall x F) = \text{Bd}(F) \cup \{x\}
 \end{array}$$

Beispiel:

$$F = \forall x(\exists y Pxyz \vee Qf_1u) \wedge \exists z Ra_0x$$

$$\text{Fr}(F) = \{z, u, x\}$$

$$\text{Bd}(F) = \{x, y, z\}$$

Anmerkung: Eine Formel F mit $\text{Fr}(F) = \emptyset$ heißt Satz.

Definition 3.9 (Substitutionen) Die (simultane) Substitution $t[x_1, \dots, x_r/t_1, \dots, t_r]$ bzw. $F[x_1, \dots, x_r/t_1, \dots, t_r]$ ist für Terme t, t_1, \dots, t_r , paarweise verschiedene Variablen x_1, \dots, x_r und Formeln F definiert durch (wir schreiben \vec{x} für x_1, \dots, x_r und \vec{t} für t_1, \dots, t_r):

$$x[\vec{x}/\vec{t}] = \begin{cases} x & \text{falls } x \notin \{x_1, \dots, x_r\} \\ t_i & \text{falls } x = x_i \end{cases}$$

$$(fs_1 \dots s_n)[\vec{x}/\vec{t}] = fs_1[\vec{x}/\vec{t}] \dots s_n[\vec{x}/\vec{t}]$$

$$\begin{aligned}
 \perp[\vec{x}/\vec{t}] &= \perp \\
 (Rs_1 \dots s_n)[\vec{x}/\vec{t}] &= Rs_1[\vec{x}/\vec{t}] \dots s_n[\vec{x}/\vec{t}] \\
 (F \otimes G)[\vec{x}/\vec{t}] &= F[\vec{x}/\vec{t}] \otimes G[\vec{x}/\vec{t}] \quad \text{für } \otimes \in \{\vee, \wedge\} \\
 (\neg F)[\vec{x}/\vec{t}] &= \neg(F[\vec{x}/\vec{t}]) \\
 (QxF)[\vec{x}/\vec{t}] &= Qu(F[x_{i_1}, \dots, x_{i_s}, x/t_{i_1}, \dots, t_{i_s}, u]) \quad \text{mit } Q \in \{\exists, \forall\},
 \end{aligned}$$

wobei in der letzten Zeile x_{i_1}, \dots, x_{i_s} genau die Variablen x_i aus \vec{x} sind, für die $x_i \in \text{Fr}(QxF)$ und $x_i \neq t_i$ gilt, und u eine neue Variable mit $u \notin \text{Fr}(F) \cup \text{Var}(t_{i_1}) \cup \dots \cup \text{Var}(t_{i_s})$ ist. Falls $x \notin \text{Var}(t_{i_1}) \cup \dots \cup \text{Var}(t_{i_s})$, kann auch $u = x$ gewählt werden.

Beispiel:

1. $fyz[y, z, u/z, x, y] = fzx$
2. $\forall xPxy[y/x] = \forall u(Pxy[x, y/x, u] = \forall uPux$
3. $(\exists xPxfyz)[x, z/u, fyy] = \exists xPxfyfy$

Anmerkung: Eine Formel F' , die aus F durch endlich viele Anwendungen der Äquivalenz

$$QxA \equiv QyA[x/y]$$

wobei $y \notin \text{Fr}(A)$, $Q \in \{\forall, \exists\}$, entsteht heißt *gebundene Umbenennung* von F .

Beispiel: $\exists(Pxy \forall y Rxfy)$ ist gebundene Umbenennung von $\exists z(Pzy \forall x Rzfz)$.

3.4 Normalformen

Definition 3.10 (Negationsnormalform) Eine Formel ist in *Negationsnormalform* (NNF), falls Negationen nur vor Relationssymbolen auftreten. (\perp wird dabei wie ein Relationssymbol behandelt.)

Lemma 3.11 Zu jeder Formel $F \in \Phi$ gibt es eine äquivalente Formel in NNF.

Algorithmus für NNF (Termersetzungssystem)

$$\neg \forall x F \rightsquigarrow \exists x \neg F$$

$$\neg \exists x F \rightsquigarrow \forall x \neg F$$

Außerdem gelten die Regeln der AL–NNF–Transformation, z.B.

$$\neg(F \wedge G) \rightsquigarrow \neg F \vee \neg G.$$

Definition 3.12 (Pränexe Normalform) Eine Formel $F \in \Phi$ ist in pränexer Normalform (PNF), falls sie die Form

$$Q_1x_1 \dots Q_nx_n F_0$$

besitzt, wobei $Q_i \in \{\exists, \forall\}$ für $1 \leq i \leq n$ und F_0 quantorenfrei ist. $Q_1x_1 \dots Q_nx_n$ heißt Präfix und F_0 Matrix von F .

Lemma 3.13 Zu jeder Formel $F \in \Phi$ gibt es eine äquivalente Formel in pränexer Normalform.

Algorithmus für PNF: (Termersetzungs-system für Formeln in NNF)

$$F \vee \exists x G \rightsquigarrow \exists y (F \vee G[x/y]), \text{ wobei } y \notin \text{Var}(F) \vee \text{Fr}(G)$$

$$F \wedge \forall x G \rightsquigarrow \forall y (F \wedge G[x/y]), \text{ wobei } y \notin \text{Var}(F) \vee \text{Fr}(G)$$

$$F \vee \forall x G \rightsquigarrow \forall y (F \vee G[x/y]), \text{ wobei } y \notin \text{Fr}(F) \vee \text{Fr}(G)$$

$$F \wedge \exists x G \rightsquigarrow \exists y (F \wedge G[x/y]), \text{ wobei } y \notin \text{Fr}(F) \vee \text{Fr}(G)$$

Beispiel:

$$\exists x \underbrace{(Pxy)}_F \vee \underbrace{\forall y Rxfy}_G \rightsquigarrow \underbrace{\exists x \forall z}_{\text{Präfix}} \underbrace{(Pxy \vee Rxfz)}_{\text{Matrix}}$$

Ziel: Elimination von Existenzquantoren

Beispiel: $F = \forall x \exists y Pxy$

Dann gibt es eine Funktion f , die für alle x das „passende“ y berechnet:
 $\forall x Pxf(x)$ ist damit erfüllbar, falls F erfüllbar ist, also eine Interpretation \mathcal{J} existiert mit $\mathcal{J} \models F$. Dieses \mathcal{J} kann man um obige Funktion f erweitern.

Definition 3.14 (Skolem-Normalform) Eine Formel ist in Skolem-Normalform (SNF), wenn sie in pränexer Normalform ist und ihr Präfix nur universelle Quantoren (\forall) enthält.

Idee: Falls $\exists x Px$, dann gibt es eine Funktion, die dieses existierende Element bezeichnet. Das Element hängt ggf. von anderen Elementen ab:

$$\forall x \exists y Py \rightsquigarrow Pf(x)$$

$$\exists y Py \rightsquigarrow P(c)$$

Satz 3.15 Zu jeder Formel $F \in \Phi$ gibt es eine Formel $G \in \Phi$ für die gilt:

- G ist in pränexer Normalform.
 - G enthält keine Existenzquantoren.
- } G ist in Skolem-Normalform
- F und G sind erfüllbarkeitsäquivalent.

Anmerkung: G kann zusätzliche, nicht in F vorkommende Funktionssymbole enthalten. Diese werden Skolemfunktionen bzw. Skolemkonstanten genannt.

Algorithmus für SNF: Eingabe F in PNF, d.h. $F = Q_1x_1 \dots Q_mx_m F_0$ mit F_0 quantorenfrei.

\exists -Elimination durch Anwendung der Regel:

$$\begin{aligned} & \forall x_1 \dots \forall x_k \exists x_{k+1} Q_{k+2}x_{k+2} \dots Q_mx_m F_0 \rightsquigarrow \\ & \forall x_1 \dots \forall x_k (Q_{k+2}x_{k+2} \dots Q_mx_m F_0)[x_{k+1}/f x_1 \dots x_k] \end{aligned}$$

wobei f ein neues k -stelliges Funktionssymbol ist.

Beispiel:

$$\begin{aligned} & \exists x \forall y \exists z (Pxy \wedge Ryz) \\ & \xrightarrow{k=0} (\forall y \exists z (Pxy \wedge Ryz))[x/c_0] = \forall y \exists z (Pc_0y \wedge Ryz) \\ & \xrightarrow{k=1} \forall y (Pc_0y \wedge Ryz)[z|f_1y] = \forall (Pc_0y \wedge Ryf_1y) \end{aligned}$$

Anmerkung: $\models F$ gdw. $\models \forall x F$.

Algorithmus: Generierung einer erfüllbarkeitsäquivalenten Formel in Klausel-darstellung

Eingabe: $F \in \emptyset$ mit $\text{Fr}(F) = \Phi$ (d.h. F ist ein Satz).

1. Berechne NNF F_1 von F .
2. Berechne PNF F_2 von F_1 .
3. Berechne SNF F_3 von F_2 .
4. Bringe Matrix von F_3 in konjunktive Normalform $\rightsquigarrow F_4$.

Ausgabe: Matrix von F_4 in Klauseldarstellung.

Beispiel: $F = \exists x \forall y Rxy \wedge \neg \exists z \forall u Rzu$

1. NNF: $F_1 = \exists x \forall y Rxy \wedge \forall z \exists u \neg Rzu$
2. PNF: $F_2 = \exists x \forall z \exists u \forall y (Rxy \wedge \neg Rzu)$
3. SNF: $F_3 = \forall z \forall y (R_{c_0}y \wedge \neg Rzf_1z)$
4. CNF: $F_4 = F_3$

Ausgabe: $\{\{Rc_0y\}, \{\neg Rzf_1z\}\}$

3.5 Unifikation

Hintergrund: (syntaktisches) Lösen von Termgleichungssystemen

Beispiel: $\{f(x, y) = z, g(y) = g(g(x))\}$

Durch welche Terme müssen die Variablen x, y, z ersetzt werden, damit (syntaktische) Gleichheit erreicht wird?

$\{y \mapsto g(x), z \mapsto f(x, g(x))\}$ oder $\{x \mapsto a, y \mapsto g(a), z \mapsto f(a, g(a))\}$ (wobei a eine Konstante ist) sind Lösungen des Beispiels.

Definition 3.16 (Substitutor) Ein Substitutor ist eine Abbildung $\sigma : \mathcal{V} \rightarrow T(\mathcal{V}, \mathcal{F})$, so dass $\sigma(x) = x$ für fast alle x . Wir schreiben auch $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ für die Abbildung

$$\sigma(x) = \begin{cases} t_i & \text{falls } x = x_i \text{ für ein } i \text{ mit } 1 \leq i \leq n, \\ x & \text{falls } x \notin \{x_1, \dots, x_n\}, \end{cases}$$

wobei die x_i paarweise verschieden sind.

Definition 3.17 (Substitutionsanwendung) Die Substitutionsanwendung $t\sigma$ bzw. $F\sigma$ für einen Term t , eine Formel F und einen Substitutor $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ ist definiert durch

$$\begin{aligned} t\sigma &:= t[x_1, \dots, x_n/t_1, \dots, t_n] \\ F\sigma &:= F[x_1, \dots, x_n/t_1, \dots, t_n] \end{aligned} .$$

Definition 3.18 (Separator) Seien K_1 und K_2 Klauseln. Eine Umbenennung ξ heißt Separator von K_1 und K_2 , falls $\text{Fr}(K_1\xi) \cap \text{Fr}(K_2) = \emptyset$.

Anmerkung:

1. Ein bijektiver Substitutor $\sigma : \mathcal{V} \rightarrow \mathcal{V}$ wird *Umbenennung* genannt.
2. Sei σ ein Substitutor. σ kann zu einer Funktion $T(\mathcal{V}, \mathcal{F}) \rightarrow T(\mathcal{V}, \mathcal{F})$ erweitert werden durch

$$(f(t_1 \dots t_n))\sigma = f(t_1\sigma \dots t_n\sigma)$$

3. Für zwei Substitutoren σ und τ definieren wir deren Komposition $\sigma\tau$ durch

$$x(\sigma\tau) := (x\sigma)\tau$$

(Postfixnotation, d.h. $\sigma\tau(x) = \tau(\sigma(x))$)

4. $K\xi := \{l\xi \mid l \in K\}$

Ziel von Separatoren: keine gleichen Variablen in K_1 und K_2 .

Beispiel: $\xi = \{z \mapsto v, u \mapsto w, v \mapsto z, w \mapsto u\} =: \{z \leftrightarrow v, u \leftrightarrow w\}$ ist ein Separator von $\{Pxz, Puz\}$ und $\{Qy, Pzu\}$.

Definition 3.19 (Unifikator) Sei L eine Menge von Literalen. L heißt unifizierbar, falls es einen Substitutor σ gibt, für den $L\sigma$ aus nur einem Element besteht. σ heißt dann Unifikator von L .

Beispiel: $L = \{x, a\}$; $L = \{R(x, g(y)), R(g(a), g(a))\}$ unifizierbar
 $L = \{R(y, y), R(g(x), x)\}$ nicht unifizierbar

Definition 3.20 (Allgemeinster Unifikator (MGU)) Ein Unifikator σ einer Literalmenge L heißt allgemeinster Unifikator (most general unifier, MGU) von L , falls es zu jedem anderen Unifikator σ' von L einen Substitutor τ gibt, so dass $\sigma\tau = \sigma'$.

Anmerkung: $L = \emptyset$ ist nicht unifizierbar.

Beispiel: Sei $L = \{P(f(x, y), g(y)), P(z, g(g(x)))\}$.
 L ist unifizierbar mit $\sigma = \{y \mapsto g(x), z \mapsto f(x, g(x))\}$ als Unifikator, denn $L\sigma =$

$$\{P(f(x, g(x)), g(g(x))), P(f(x, g(x)), g(g(x)))\} = \{P(f(x, g(x)), g(g(x)))\}$$

$$\begin{aligned} &\{R(x, g(y)), R(u, v)\} \\ \text{mgu } \mu &:= \{x \mapsto u, v \mapsto g(y)\} \\ \sigma &:= \{x \mapsto a, u \mapsto a, v \mapsto g(y)\} \\ \text{mgu } \mu' &:= \{u \mapsto u, v \mapsto g(y)\} \end{aligned}$$

Anmerkung: Vergleiche Lösen von Termgleichungssystemen

Unifikationsalgorithmus (nach J.A.Robinson):

Berechnet allgemeinsten Unifikator einer Literalmenge L, sofern dieser existiert.

1. Falls L nicht von der Form $\{Pt_1^1 \dots t_n^1, Pt_1^2 \dots t_n^2, \dots, Pt_1^k \dots t_n^k\}$ für ein n-stelliges Prädikat P und Terme t_i^j oder falls $L = \emptyset$: STOP mit Ausgabe „L ist nicht unifizierbar“.
2. Setze $i := 0$ und $\sigma_i := id$ (Identitätsfunktion).
3. Falls $L\sigma_i$ einelementig: STOP mit „ σ_i ist allgemeinsten Unifikator“
4. Wähle $F_1, F_2 \in L\sigma_i$ mit $F_1 \neq F_2$. Seien s_1 und s_2 die ersten in F_1 und F_2 unterschiedlichen Symbole.
5. Falls sowohl s_1 als auch s_2 Funktionssymbole sind: STOP mit „L ist nicht unifizierbar“.
6. Falls s_1 oder s_2 Variablensymbol (angenommen $s_1 = x$): Bestimme Term t in F_2 , der an der Position von s_2 beginnt.
7. Falls $x \in Var(t)$: STOP mit „L ist nicht unifizierbar“.
8. Setze $\sigma_{i+1} := \sigma_i\{x \mapsto t\}$ und $i := i + 1$.
9. Gehe nach 3.

Anmerkung: Der Unifikationsalgorithmus kann exponentielle Größe annehmen, z.B.:

$$\begin{aligned} L &= \{Pfx_0 \dots x_n, Pfgx_1x_1gx_2x_2 \dots gx_{n-1}x_{n-1}\} \\ \mathcal{F} &= \{f_n, g_2\} \\ \sigma_1 &= \{x_0 \mapsto gx_1x_1\} \\ \sigma_2 &= \sigma_1\{x_1 \mapsto gx_2x_2\} = \{x_0 \mapsto ggx_2x_2gx_2x_2, x_1 \mapsto gx_2x_2\} \end{aligned}$$

3.6 Prädikatenlogische Resolution

Erweiterung der aussagenlogischen Resolution auf PL1.

Definition 3.21 (Resolution) Seien K_1, K_2 und K Klauseln. K heißt Resolvente von K_1 und K_2 , falls es einen Separator ξ von K_1 und K_2 gibt und Mengen $M_1, L_1 \subseteq K_1, M_2, L_2 \subseteq K_2$, so dass

- (i) $L_1 \neq \emptyset$ und $L_2 \neq \emptyset$.
- (ii) $L_1\xi \cup \neg L_2$ ist unifizierbar mit allgemeinstem Unifikator (MGU) μ .
- (iii) $K_1 = M_1 \dot{\cup} L_1, K_2 = M_2 \dot{\cup} L_2$ und $K = (M_1\xi \cup M_2)\mu$.

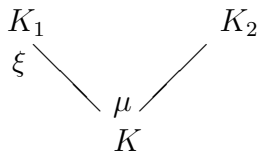


Abbildung 8: Schreibweise des Resolutionskalküls für PL1

Beispiel: $K_1 = \{\neg Pxyz, Rygfx\}, K_2 = \{\neg Rfxgy\}, \mathcal{F} = \{c_0, f_1, g_1\}, \mathcal{R} = \{P_3, R_2\}$

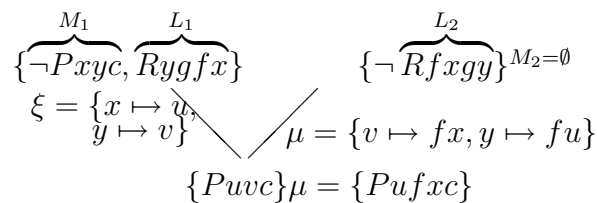


Abbildung 9: Beispiel zur Resolution

$L_1\xi = \{Rvgfu\}, M_1\xi = \{\neg Puv\}$
 Dabei ist $\neg L = \{\neg l \mid l \in L\}$

Verfahren zur Generierung einer Resolventen (Gegeben: Klauselmenge S)

1. Wähle zwei Klauseln K_1, K_2 zur Resolution.
2. Benenne Variablen in K_1 um, so dass K_1 und K_2 variablenfremd sind (liefert Separator ξ).
3. Lege Literale fest ($L_1 \subseteq K_1, L_2 \subseteq K_2$) über die resolviert werden soll.
4. Bestimme allgemeinsten Unifikator μ von $L_1\xi \cup \neg L_2$.
5. Berechne Resolvente:
$$\left(\underbrace{(K_1\xi - L_1\xi)}_{M_1\xi} \cup \underbrace{(K_2 - L_2)}_{M_2} \right) \mu$$

Anmerkung: Resolutionsherleitung (\vdash_{Res}), Resolutionsabschluss (Res^*) lassen sich direkt vom aussagenlogischen Fall übertragen. Definition von \models überträgt sich ebenfalls.

Satz 3.22 *Der prädikatenlogische Resolutionskalkül ist korrekt und widerlegungsvollständig, d.h. es gilt für alle $F \in \Phi$ (in Klauseldarstellung) und alle Klauseln C :*

1. *Korrektheit:* $F \vdash_{Res} C$ impliziert $F \models C$.
2. *Widerlegungsvollständigkeit:* F unerfüllbar impliziert $F \vdash_{Res} \square$.

Beweis: siehe z.B. Gallier, „Logic for Computer Science“.

Anmerkung:

1. Für PL1-Resolution gibt es ähnliche Beweisstrategien wie für die AL-Resolution (geordnete Resolution, Hyperresolution, Set-of-Support Resolution).
2. Zur Lösung eines Folgeungsproblems $F \models C$ für eine PL1-Formel F und eine Klausel C , $C = \{l_1, \dots, l_k\}$, kann man folgende Äquivalenz verwenden:
 $F \models C$ gdw. $F \wedge \neg C$ unerfüllbar
gdw $F \cup \{\{\neg l_1\}, \dots, \{\neg l_k\}\} \vdash_{Res} \square$.

Das Herbrand Universum

Sei S eine Menge von Klauseln.

$H(S)$ ist wie folgt definiert:

1. Die Menge aller Konstanten-Symbole $\{f_i^0\}$, die in S vorkommen ist in $H(S)$. (Falls $\{f_i^0\} = \emptyset$, dann sei ein beliebiges Symbol a in $H(S)$.)
2. Falls die Terme t_1, \dots, t_n in $H(S)$, dann ist auch $f_i^n(t_1, \dots, t_n)$ in $H(S)$. Dabei sind f_i^n n -stellige Funktionssymbole in S .
3. Nichts sonst ist in $H(S)$.

Beispiel: Sei $S = \{P(x) \vee Q(a) \vee \neg P(f(x)) \vee \neg Q(b) \vee P(g(x, y))\}$

$$H(S) = \{a, b, f(a), f(b), g(a, a), g(a, b), g(b, a), g(b, b), f(f(a)), f(f(b)), \dots, g(a, f(a)), \dots\}$$

Die Herbrand Basis

Die Herbrand-Basis besteht aus allen Grund-Instanzen aller atomaren Formeln aus S , wobei die Grundinstanzen durch Einsetzen des Herbrand-Universums in die Variablen gebildet werden. Die Elemente der Herbrand-Basis heißen *Atome*.

Beispiel: Sie P ein Prädikatensymbol.

Die Herbrand-Basis entsteht durch Einsetzen des Herbrand-Universums in die Argumente von P .

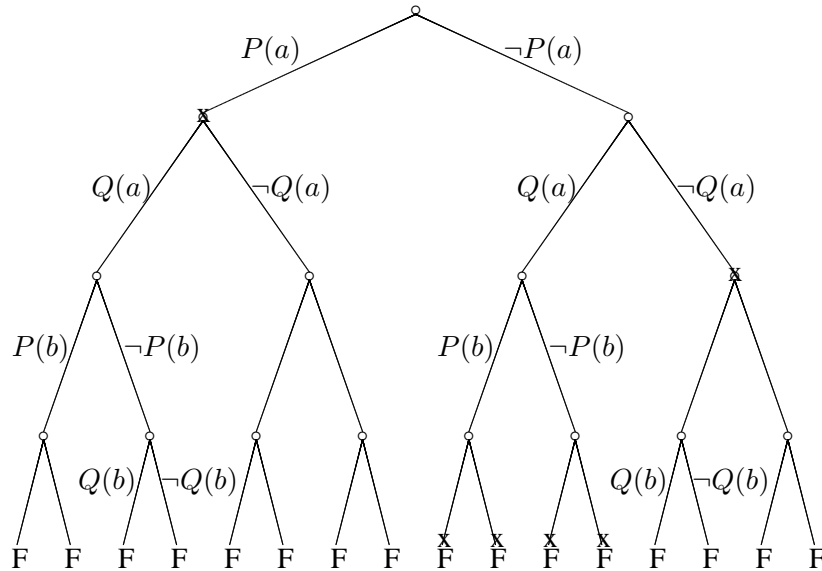
$$P(x, y) \rightsquigarrow P(a, b), P(a, a), \dots$$

Semantische Bäume

Beispiel: $S := \{P(x) \vee Q(y), \neg P(a), \neg Q(b)\}$

$$H(S) = \{a, b\}$$

$$\text{H-B}(S) = \{P(a), P(b), Q(a), Q(b)\}$$



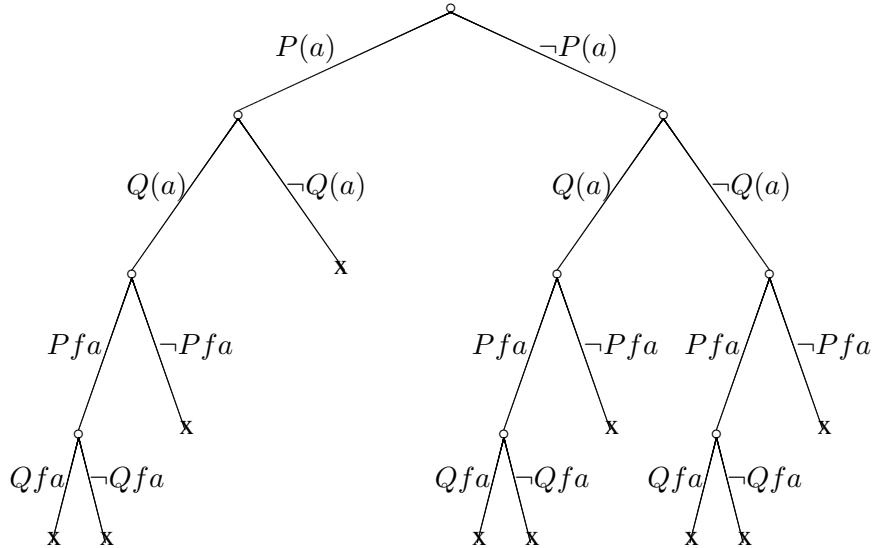
Jeder Pfad im Baum definiert eine Interpretation I . I kann die Klauselmenge wahr oder falsch machen.

Ein Pfad von der Wurzel zu einem Blatt bestimmt eine Interpretation, ein Modell. Ein Modell falsifiziert eine Klausel K , wenn es eine Grundinstanz von K gibt, die unter dem Modell den Wert F erhält. Ein Knoten an dem eine Klausel K zum ersten Mal falsifiziert wird, heißt *Fehlerknoten* (*failure node*) von K . Ein semantischer Baum heißt *geschlossen für eine Klauselmenge S* , falls alle Pfade im Baum in Fehlerknoten enden.

Satz 3.23 *Ein semantischer Baum für eine unerfüllbare Klauselmenge S ist für S geschlossen und enthält eine endliche Menge von Knoten oberhalb der Fehlerknoten.*

Definition 3.24 *Ein Knoten, dessen Kinder Fehlerknoten sind, heißt Inferenzknoten.*

Beispiel: $S = \{\neg Px \vee Qx, Pfy, Qfy\}$



Korrektheit (soundness) und Vollständigkeit (completeness) der Resolution

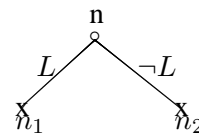
Satz 3.25 (Korrektheit) Die Resolvente ist eine logische Konsequenz ihrer Eltern.

Satz 3.26 (Vollständigkeit) Aus jeder unerfüllbaren Klauselmengemenge S lässt sich mit Resolution die leere Klausel ableiten.

Beweis: Sei T_τ ein geschlossener semantischer Baum von S ; sei n ein Interferenzknoten mit Kindern n_1 und n_2 . Klausel $\{A_i\}$ werde an n_1 falsifiziert und $\{B_i\}$ an n_2 .

Wir zeigen: es existiert eine Resolvente $\{C_i\}$, die bei n (oder höher) falsifiziert wird.

Sei L das Literal, über das an n entschieden wird: L ist wahr an n_1 und falsch an n_2 .



Betrachte $\{A_i\}$, das an n_1 falsifiziert wird. Es muss eine unifizierbare Teilmenge $\{a_i\} \subseteq \{A_i\}$ geben, die $\neg L$ als Grundinstanz hat: $\{a_i\}_\sigma = L$. $\{A_i\} \setminus \{a_i\}$ wird an n oder höher falsifiziert.

Analog für $\{B_i\}$ mit $\{b_i\}$ und n_2 und Unifier τ .

Wir kombinieren σ und τ zu ω .

Da $\{a_i\}_\omega = L$ und $\{b_i\}_\omega = \neg L$, haben $\{A_i\}$ und $\{B_i\}$ eine Resolvente $[\{A_i\} \setminus \{a_i\}]_\lambda \cup [\{B_i\} \setminus \{b_i\}]_\lambda$, wobei λ ein mgu ist und $[\{A_i\} \setminus \{a_i\}]_\omega$ eine Instanz von $[\{A_i\} \setminus \{a_i\}]_\lambda$. (Da $\{a_i\}_\omega = L$ und $\{b_i\}_\omega = \neg L$, haben $\{A_i\}_\omega$ und $\{B_i\}_\omega$ eine Resolvente R_ω . Dann gibt es aber auch eine Resolvente R_λ von $\{A_i\}_\lambda$ und $\{B_i\}_\lambda$ mit mgu λ .)

Sowohl $[\{A_i\} \setminus \{a_i\}]_\omega$ also auch $[\{B_i\} \setminus \{b_i\}]_\omega$ werden an n falsifiziert. Das gilt auch für $[\{A_i\} \setminus \{a_i\}]_\lambda$ und $[\{B_i\} \setminus \{b_i\}]_\lambda$, also auch für $[\{A_i\} \setminus \{a_i\}]_\lambda \cup [\{B_i\} \setminus \{b_i\}]_\lambda$. Das ist die Resolvente! Also werden nach endlich vielen Resolutionsschritten alle Inferenzknoten zu Fehlerknoten, inklusive der Wurzel. Dort schlägt die leere Klausel fehl. \square

Literatur

- [A98] H. R. Andersen: *An Introduction to Binary Decision Diagrams*. <http://www.it-c.dk/people/hra/bdd97.ps>.
- [EFT94] H.-D. Ebbinghaus, J. Flum, W. Thomas: *Mathematical Logic*. Springer-Verlag, 1994.
- [G86] J. Gallier: *Logic for Computer Science* Harper& Row, 1986.
- [HKT00] D. Harel, D. Kozen, and J. Tiuryn: *Dynamic Logic*. MIT Press, 2000.
- [KBL94] H. Kleine Büning, T. Lettmann: *Aussagenlogik: Deduktion und Algorithmen*. Teubner, 1994.