Elementare Logik

SS26 — Stefan Näher — Universität Trier

Stand: 11. April 2026

Inhaltsverzeichnis

Literatur

[1] M. Kreuzer / S Kühling: Logik für Informatiker.

[2] Uwe Schöning: Theoretische Informatik - kurzgefasst.

[3] John E. Hopcroft, Jeffrey D. Ullman: Introduction to Automata Theory, Languages and Computation // (übersetzte Version: Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie.)

[4] C. Meinel, M. Mundhenk: Mathematische Grundlagen der Informatik. Mathematisches Denken und Beweisen, eine Einführung.

[5] K. A. Ross, C. R. B. Wright: Discrete Mathematics.

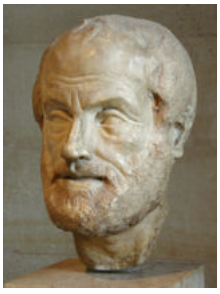
[6] E. Kinber / C. Smith: Theory of Computing. A Gentle Introduction.

- Es gibt ungezählte Bücher zur 'Einführung in die Theoretische Informatik"; finden Sie selbst ein für Sie gut geeignetes Buch!
- Es gibt zahlreiche gute Skripten im Internet...
- Gehen Sie in die Bibliothek oder in eine Buchhandlung und beginnen Sie, in verschiedenen Büchern zur Theoretischen Informatik zu lesen: wenigstens zwei davon sollten Sie über das Studium begleiten...

1 Grundbegriffe der Logik

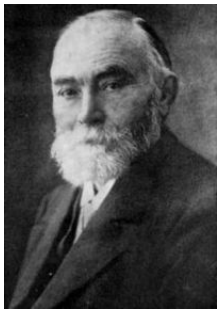
1.1 Begriffe der Aussagenlogik

Logische Ahnengalerie


Aristoteles


Leibniz


Boole


Frege


Gödel

- Aristoteles (384-322 v.Chr.), Begründer der Logik: Aussage, Definition, Beweis
- Gottfried Wilhelm von Leibniz (1646-1716): symbolische Logik in Kalkülform
- George Boole (1815-1864): Aussagenlogik, disjunktive Normalform
- Gottlob Frege (1848-1925): Logik als formale Sprache mit formalen Beweisen
- Kurt Gödel (1906-1978): Unvollständigkeit formaler logischer Systeme

Formen von Logik:

mathematische Logik	formale Logik	klassische Logik
Aussagenlogik	Boole'sche Logik	Prädikatenlogik
intuitionistische Logik	mehrwertige Logik	Kleene Logik
fuzzy logic	temporale Logik	Quantenlogik
philosophische Logik	Modallogik	epistemische Logik

In dieser Vorlesung:

- Boole'sche Logik (=Aussagenlogik) sowie
- ein Einstieg in Prädikatenlogik

Definition 1.1.1. *Eine Aussage (im Sinne der Logik) ist ein (oft sprachliches) Gebilde, bei dem es sinnvoll ist zu fragen, ob es wahr (***w***) oder falsch (***f***) ist.* w *und* f *heißen auch Wahrheitswerte.*

Beispiel 1.1.2.

- *5 ist eine Primzahl. (wahre Aussage)*
- $\sqrt{7}$ *ist rational. (falsche Aussage)*
- e^{π} *ist rational. (Aussage mit unbekanntem Wert)*
- *'Wie geht's?' (Fragen sind keine Aussagen!)*
- *'Dieser Satz ist falsch'. (Antinomie von Russel, keine Aussage!)*

Erklärung für einige benutzte Begriffe:

- *Primitive Begriffe*: Begriffe, die nicht vollständig erklärt werden (Beispiele: Aussage, Menge, Universum, ...)
- *Definierte Begriffe*: Begriffe, die mit Hilfe bekannter Begriffe vollständig erklärt werden (Bsp: Graph, Relation, Funktion....)
- *Notationen*: symbolische Beschreibungen ($\Leftrightarrow$ oder $\equiv$)

Erklärung für einige wichtige verwendete mathematische Begriffe:

- *Axiome*: Aussagen, die grundlegende Annahmen formulieren, die nicht beweisen werden.
- *Sätze*: Aussagen, die bewiesen werden können.
- *Propositionen*: Sätze, deren Beweis relativ einfach ist.
- *Beweise*: vollständige und nachvollziehbare Argumentationen, die zweifelsfrei zeigen, dass eine Aussage gültig ist.

Definition 1.1.3 (Aussageverknüpfungen')**.**

- *Sind* $p_1, \ldots, p_n$ *Aussagen, so können wir diese zu Aussageverknüpfungen* $a(p_1, \ldots, p_n)$ *kombinieren.*
- *Der Wahrheitswert einer extensionalen Aussageverknüpfung* $a(p_1, \ldots, p_n)$ *ist bereits eindeutig durch die Wahrheitswerte ihrer Teilaussagen* p_i *bestimmt.*

Beispiel 1.1.4.

- *Alice ist reich und Bob ist arm. (extensionale Verknüpfung)*
- *Alice ist reich, weil Bob arm ist. (nicht extensional, 'intensional')*

"Extensionalität" bei Aussagenlogik:

- Es werden nur extensionale Verknüpfungen betrachtet.
- Kern ist die Untersuchung möglicher Verknüpfungen.

Definition 1.1.5. *Eine Aussageform (oder Prädikat) ist ein (sprachliches) Gebilde mit möglichen Leerstellen (oder Variablen), welches durch Ersetzen in die Leerstellen zu einer Aussage wird. Die Gesamtheit $\boldsymbol{U}$ der Objekte, die in eine Leerstelle $\boldsymbol{x}$ eingesetzt werden dürfen, muss (vorher) bekannt sein; $\boldsymbol{U}$ heißt auch Universum.*

Beispiel 1.1.6.

- *"$\boldsymbol{x}$ ist durch drei teilbar." ist Aussageform (Universum $\mathbb{Z}$): Für jede ganze Zahl $\boldsymbol{x}$ ergibt sich eine Aussage. Abhängig von der Belegung ist die Aussage wahr oder falsch.*
- *"$(\boldsymbol{x}+\boldsymbol{y})^2 = \boldsymbol{x}^2 + 2\boldsymbol{x}\boldsymbol{y} + \boldsymbol{y}^2$" ist Aussageform (Universum $\mathbb{R}$): Für beliebige reelle $\boldsymbol{x}, \boldsymbol{y}$ ergibt sich eine wahre Aussage.*
- *"$\boldsymbol{p}$ ist wahr" ist Aussageform (Universum: Menge aller Aussagen): "**4 ist Primzahl** ist wahr" ist falsche Aussage, "**5 ist Primzahl** ist wahr" ist wahre Aussage*

Sonderfall: Aussagen sind Aussageformen mit 0 Leerstellen.

Beispiel 1.1.7 (Prädikate in Programmiersprachen)**.**

- *Wahrheitswerte als Datentyp* `bool`*, Werte* `TRUE,FALSE`
- `bool even(int x)`*, Universum* $\mathbb{Z}$
- `bool prim(int x)`*, Universum* $\mathbb{Z}$
- `bool rel_prim(int x,int y)`*, Universum* $\mathbb{Z}^2$
- *Logische Operatoren als Prädikate:*

```
bool operator < (int x,int y)
```

Prädikatenlogik (im Vorgriff...)

Ist $\boldsymbol{A}(\boldsymbol{x})$ eine Aussageform (mit Universum $\boldsymbol{X}$ und Leerstelle $\boldsymbol{x}$), so kann man auch Aussagen gewinnen durch Konstruktionen wie:

- Für alle $\boldsymbol{x}$ gilt: $\boldsymbol{A}(\boldsymbol{x})$.
- Für mindestens zwölf $\boldsymbol{x}$ gilt: $\boldsymbol{A}(\boldsymbol{x})$.
- Für irgendein $\boldsymbol{x}$ gilt: $\boldsymbol{A}(\boldsymbol{x})$.
- Es gibt ein $\boldsymbol{x}$, für das gilt: $\boldsymbol{A}(\boldsymbol{x})$.

Metalogik: Oft redet man mit logischen Sprechweisen über Logik, z.B.

- Für alle Aussagen $\boldsymbol{p}$ gilt: $\boldsymbol{p}$ ist wahr genau dann, wenn $\boldsymbol{p}$ nicht falsch ist.

Es sei $\boldsymbol{p}$ eine Aussageform. Die *Negation* (*Verneinung*, "nicht $\boldsymbol{p}$") von $\boldsymbol{p}$ ist wiederum Aussageform.

Übliche Schreibweisen: $\neg\boldsymbol{p}$, $\overline{\boldsymbol{p}}$, $\tilde{\boldsymbol{p}}$

Definition 1.1.8 (Verknüpfung von Aussageformen I)**.** *Für Aussageformen p wird $\neg p$ genau dann wahr, wenn p falsch wird.*

Wahrheitstafel:

p	$\neg p$
w	f
f	w

Beispiel 1.1.9. *p sei: "3 ist eine Primzahl." $\neg p$ bedeutet: "Es gilt nicht, dass 3 eine Primzahl ist:" oder kürzer: "3 ist keine Primzahl."*

Die Negation ist durch Angabe der Wahrheitstafel eindeutig definiert (Extensionalität der Verknüpfung!).

Es seien p, q Aussageformen. Die *Konjunktion* (*Und-Verknüpfung*: "p und q") von p und q ist wiederum eine Aussageform.

Übliche Schreibweisen: $p \wedge q$, $p \cdot q$, pq, $p\&q$

Definition 1.1.10 (Verknüpfung von Aussageformen II)**.** *Für Aussageformen p und q wird $p \wedge q$ genau dann wahr, wenn sowohl p als auch q wahr werden.*

Wahrheitstafel:

p	q	$p \wedge q$
w	w	w
w	f	f
f	w	f
f	f	f

Beispiel 1.1.11. *p sei: "11 ist eine Primzahl." q sei: "11 ist durch drei teilbar."*

$p \wedge q$ bedeutet: "11 ist eine durch drei teilbare Primzahl."

Es seien p, q Aussageformen. Die *Disjunktion* (*Oder-Verknüpfung*: "p oder q") von p und q ist wiederum eine Aussageform.

Übliche Schreibweisen: $p \vee q$, $p + q$, $p|q$

Definition 1.1.12 (Verknüpfung von Aussageformen III)**.** *Für Aussageformen p und q wird $p \vee q$ genau dann wahr, wenn mindestens eine der zwei Aussagenformen p, q wahr wird.*

Wahrheitstafel:

p	q	$p \vee q$
w	w	w
w	f	w
f	w	w
f	f	f

Beispiel 1.1.13. *p sei: "11 ist eine Primzahl." q sei: "11 ist durch drei teilbar."*

$p \vee q$ bedeutet: "11 ist eine Primzahl oder durch drei teilbar."

Das umgangssprachliche '*und*' und die Konjunktion '$\wedge$' unterscheiden sich:

Vergleiche z.B.

- Lisa stolpert und fällt hin.
- Lisa fällt hin und stolpert.

In der natürlichen Sprache gibt es Beziehungen (hier: zeitliche Abfolge), die durch den formalen Verknüpfungsoperator nicht erfasst werden (und auch nicht erfasst werden sollen!)

Ähnliches gilt für '*oder*'/Disjunktion und '*nicht*'/Negation:

Vergleiche z.B.

- Tauchen ist gefährlich.
- Tauchen ist nicht ungefährlich.

Natürliche Sprache ist meist nicht (oder nicht komplett) extensional!

Satz 1.1.14 (Rechengesetze der Aussagenlogik I)**.**

Konjunktion und Disjunktion sind kommutativ. Dies bedeutet für beliebige Aussagenformen p, q*:*

- $p \wedge q$ *wird wahr genau dann, wenn* $q \wedge p$ *wahr wird.*
- $p \vee q$ *wird wahr genau dann, wenn* $q \vee p$ *wahr wird.*

Wie beweist man eine solche Aussage der Metalogik? mit Hilfe der Extensionalität: $\rightsquigarrow$vollständige Fallunterscheidung $\rightsquigarrow$ Wahrheitstafelmethode

(Eine Wahrheitstafel werden übrigens auch Wahrheitstabelle, Wahrheitswert-Tabelle, Wahrheitsmatrix o.ä. genannt)

Satz 1.1.15 (Rechengesetze der Aussagenlogik II)**.**

Konjunktion und Disjunktion sind assoziativ. Dies bedeutet für beliebige Aussagenformen p, q, r*:*

- $(p \wedge q) \wedge r$ *wird wahr genau dann, wenn* $p \wedge (q \wedge r)$ *wahr wird.*
- $(p \vee q) \vee r$ *wird wahr genau dann, wenn* $p \vee (q \vee r)$ *wahr wird.*

Bei "längeren" $\wedge$ (oder auch $\vee$) Ausdrücken können wir Klammern "einsparen".

Beweis wiederum durch Wahrheitstafelmethode.

p	q	r	$p \wedge q$	$(p \wedge q) \wedge r$	$q \wedge r$	$p \wedge (q \wedge r)$
w	w	w	w	w	w	w
w	w	f	w	f	f	f
w	f	w	f	f	f	f
w	f	f	f	f	f	f
f	w	w	f	f	w	f
f	w	f	f	f	f	f
f	f	w	f	f	f	f
f	f	f	f	f	f	f

Satz 1.1.16 (Rechengesetze der Aussagenlogik III)**.**

Es gelten folgende Distributivgesetze für Konjunktion und Disjunktion:

1. $(p \wedge q) \vee r$ *und* $(p \vee r) \wedge (q \vee r)$ *erhalten stets denselben Wahrheitswert.*
2. $(p \vee q) \wedge r$ *und* $(p \wedge r) \vee (q \wedge r)$ *erhalten stets denselben Wahrheitswert.*

Satz 1.1.17 (Rechengesetze der Aussagenlogik IV)**.**

Regeln zur Behandlung der Negation:

1. $\neg(\neg p)$ *und* p *erhalten stets denselben Wahrheitswert.*
2. $\neg(p \vee q)$ *und* $(\neg p) \wedge (\neg q)$ *erhalten stets denselben Wahrheitswert.*
3. $\neg(p \wedge q)$ *und* $(\neg p) \vee (\neg q)$ *erhalten stets denselben Wahrheitswert.*

(2) und (3) heißen auch *Gesetze von de Morgan*.

Logische Operationen in Programmiersprachen

Frage: Sind die *aussagenlogischen Verknüpfungen* (auch *Junktoren* genannt) "dasselbe" wie die die *logischen Operatoren* in Programmiersprachen ?

JEIN: . . . JA bei seiteneffektfreien Programmen, NEIN im Allgemeinen z.B. in Java: aus Effizienzgründen wird y bei $x||y$ nicht "ausgeführt", falls x schon als wahr bekannt ist. Damit ist die Wirkung von $x||y$ möglicherweise von der von $y||x$ verschieden.

Definition 1.1.18 (Verknüpfung von Aussagen IV)**.** *Weitere Junktoren:*

- *Implikation* $p \to q$ *gilt genau dann, wenn* $(\neg p) \vee q$ *gilt.*
- *Äquivalenz* $p \leftrightarrow q$ *gilt genau dann, wenn* $(p \to q) \wedge (q \to p)$.

Wie lauten also die Wahrheitstafeln? ($\rightsquigarrow$ Selbsttest)

Sprechweisen für $p \to q$:

- "Aus p folgt q.";
- "Wenn p, dann q." oder "Nur wenn q, dann p."
- "p ist *hinreichende Bedingung* oder *Prämisse* für q";
- "q ist *notwendige Bedingung* oder *Konklusion* für p";

Sprechweisen für $p \leftrightarrow q$:

- "p genau dann, wenn q." (Vermischung von Logik und Metalogik...)

weitere Schreibweisen: $p \Rightarrow q$ bzw. $p \Leftrightarrow q$.

Die *(materiale)* **Implikation** — ein schwieriger Junktor. . .

Beispiele:

- Dass es regnet, ist eine hinreichende Bedingung dafür, dass die Straße nass ist.
- Schon wenn es regnet, ist die Straße nass.
- Wenn es regnet, dann ist die Straße nass.
- Nur wenn die Straße nass ist, regnet es.

Die Lesart "wenn...dann" ist insofern problematisch, als man in der natürlichen Sprache vor allem *inhaltliche Zusammenhänge* oder *zeitliche Nähe* dadurch ausdrückt. All das macht die materiale Implikation **nicht**, sie nennt nur den *formalen Zusammenhang*.Zur Frage, warum das eine hinreichende Bedingung ist (ob auf Grund eines kausalen Zusammenhangs oder auch nur rein zufällig), nimmt die materiale Implikation nicht Stellung.

Als *Umkehrschluss* (oder *Kontraposition*) bezeichnet man die Verwendung von $(\neg q) \to (\neg p)$ an Stelle von $p \to q$.

Im Beispiel:

- Wenn es regnet, dann ist die Straße nass. ('Originalformulierung')
- Wenn die Straße nicht nass ist, dann regnet es nicht. ('Umkehrschluss')
- Nur wenn es nicht regnet, ist die Straße nicht nass. ('Umkehrschluss')

Umgangssprachlich lässt man sich gelegentlich zu weiteren (oft falschen!) Aussagen verleiten, z.B.: Weil es nicht regnete, kann die Straße nicht nass sein.

Diese Folgerung ist falsch, da die Straße auch aus anderen Gründen nass werden kann (Rohrbruch, Übung der Feuerwehr ...).

Also: Wenn die Folgerung $p \to q$ wahr ist, aber ebenso auch die Teilaussage $\neg p$, so hat man keine Information über q; q kann wahr oder falsch sein. ("Ex falso (sequitur) quodlibet." — "Aus Falschem folgt Beliebiges.")

Definition 1.1.19 (Tautologien und Kontradiktionen)**.** *Eine Aussageform heißt*

- *Tautologie, wenn sie unabhängig von den Werten der verwendeten Variablen stets wahr ist.*
- *Kontradiktion, wenn sie unabhängig von den Werten der verwendeten Variablen stets falsch ist.*

Beispiel 1.1.20.

- *Kontradiktion:* $p \wedge \neg p$
- *Tautologien:*
  - $p \vee \neg p$ *(Satz vom ausgeschlossenen Dritten, tertium non datur)*
  - $\neg(p \wedge \neg p)$ *(Satz vom ausgeschlossenen Widerspruch)*
  - $((p \to q) \wedge (q \to p)) \leftrightarrow (p \leftrightarrow q)$
- Beweise für das obige Beispiel z.B. über Wahrheitstafeln
- Äquivalenzen beweist man wegen des letzten Beispiels oft durch Nachrechnen zweier Implikationsrichtungen.

Beispiel 1.1.21 (Beispiele für Tautologien)**.**

1. $(p \to \neg p) \to \neg p$
2. $(p \wedge (p \to q)) \to q$
3. $(p \wedge q) \to p\,, \quad (p \wedge q) \to q$
4. $p \to (p \vee q)\,, \quad q \to (p \vee q)$
5. $(p \to q) \to [(q \to r) \to (p \to r)]$
6. $((p \to q) \wedge (q \to r) \wedge p) \to r$

Hinweis: Beweisverfahren fußen oft auf Tautologien.

- Die Negation einer Tautologie ist eine Kontradiktion.
- Die Negation einer Kontradiktion ist eine Tautologie.

Beweise für Tautologien:

Z.B. zu Tautologie 5 $(p \to q) \to [(q \to r) \to (p \to r)]$ mit (kombinierter, gekürzter) Wahrheitstafel

p	q	r	$A:$ $p \to q$	$B:$ $q \to r$	$C:$ $p \to r$	$D:$ $B \to C$	$A \to D$
w	w	w	w	w	w	w	w
w	w	f	w	f	f	w	w
w	f	?	f	?	?	?	w
f	w	w	w	w	w	w	w
f	w	f	w	f	w	w	w
f	f	?	w	w	w	w	w

"Abkürzungsregel": Ist die Prämisse falsch, so ist die Implikation wahr.Zu prüfen ist also nur, ob bei wahrer Prämisse auch die Konklusion wahr ist.

Definition 1.1.22 (Logische Äquivalenz)**.** *Zwei Aussageformen $A(v_1, \ldots, v_n)$ und $B(v_1, \ldots, v_n)$ mit denselben Variablen $v_1, \ldots, v_n$ heißen logisch äquivalent, falls die Verknüpfung*

$$A(v_1, \ldots, v_n) \leftrightarrow B(v_1, \ldots, v_n)$$

eine Tautologie ist. Schreibweise: $A(v_1, \ldots, v_n) \equiv B(v_1, \ldots, v_n)$

Viele Rechenregeln und Tautologien lassen sich so schreiben!

Unterschied zwischen '$\leftrightarrow$' und '$\equiv$':

- Das Symbol '$\leftrightarrow$' gehört zum Alphabet der Sprache der Logik, es 'lebt' innerhalb von Formeln.
- Mit '$\equiv$' vergleichen wir Eigenschaften von Formeln (von 'aussen'), es ist ein 'Metasymbol'.
- Mit '$\equiv$' lassen sich Äquivalenzketten aufschreiben, etwa $A \equiv B \equiv C$ als Abkürzung für $(A \leftrightarrow B) \wedge (B \leftrightarrow C)$

Beispiel 1.1.23 (Logische Äquivalenzen)**.**

1. *Kommutativgesetze:* $(p \wedge q) \equiv (q \wedge p)$; $(p \vee q) \equiv (q \vee p)$.
2. *Assoziativgesetze:* $(p \wedge (q \wedge r)) \equiv ((p \wedge q) \wedge r)$; $(p \vee (q \vee r)) \equiv ((p \vee q) \vee r)$.
3. *Distributivgesetze:* $(p \wedge (q \vee r)) \equiv ((p \wedge q) \vee (p \wedge r))$; $(p \vee (q \wedge r)) \equiv ((p \vee q) \wedge (p \vee r))$.
4. *Gesetze von de Morgan:* $\neg(p \wedge q) \equiv (\neg p \vee \neg q)$; $\neg(p \vee q) \equiv (\neg p \wedge \neg q)$.
5. *Identitätsgesetze:* $(p \vee f) \equiv p$; $(p \wedge w) \equiv p$.
6. *Umkehrschluss:* $(p \to q) \equiv (\neg q \to \neg p)$.
7. *Auflösen der Implikation:* $(p \to q) \equiv (\neg p \vee q)$.
8. *Auflösen der Äquivalenz:* $(p \leftrightarrow q) \equiv ((p \to q) \wedge (q \to p)) \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$.

Beweise für **Logische Äquivalenzen**:

- Wahrheitstafelbeweis (vollständige Fallunterscheidung), z.B. für Umkehrschluss

- Ausnutzung bekannter Tautologien (also durch Umformungen) und der Extensionalitätz.B. für 8b:

$$\begin{array}{rl} & (p \leftrightarrow q) \\ \equiv^{8a} & ((p \rightarrow q) \wedge (q \rightarrow p)) \\ \equiv^{7} & ((\neg p \vee q) \wedge (\neg q \vee p)) \\ \equiv^{3} & ((\neg p \vee q) \wedge \neg q) \vee ((\neg p \vee q) \wedge p) \\ \equiv^{1,3} & ((\neg p \wedge \neg q) \vee (q \wedge \neg q)) \vee ((\neg p \wedge p) \vee (q \wedge p)) \\ \equiv^{1,2,5} & (p \wedge q) \vee (\neg p \wedge \neg q) \end{array}$$

Beispiel 1.1.24 (Weitere logische Äquivalenzen)**.**

1. $((p \rightarrow q) \wedge (r \rightarrow q)) \equiv (p \vee r) \rightarrow q$.
2. $(p \vee (q \wedge \neg q)) \equiv p$; $(p \wedge (q \vee \neg q)) \equiv p$.
3. *Dominanz:* $p \wedge f \equiv f$; $p \vee w \equiv w$.
4. *Negationsgesetze:* $p \wedge \neg p \equiv f$; $p \vee \neg p \equiv w$.
5. *Doppelte Negation:* $\neg\neg p \equiv p$.
6. *Idempotenzgesetze:* $p \wedge p \equiv p$; $p \vee p \equiv p$.
7. *Absorptionsgesetze:* $(p \wedge (p \vee q)) \equiv p$; $(p \vee (p \wedge q)) \equiv p$.

Beispiel 1.1.25 (Abtrennungsregel)**.**

- *Gilt sowohl* p *als auch* $p \rightarrow q$*, so gilt auch* q*. In Zeichen:* $(p \wedge (p \rightarrow q)) \rightarrow q$*. Bezeichnung: modus ponens (Schlussketten) Beweis z.B. mit Wahrheitstafel*
- *Allgemein:*
  - *Sind* $p_1, \ldots, p_n$ *Aussageformen und*
  - *werden* p_1 *und* $p_i \rightarrow p_{i+1}$ *wahr für* $i = 1, \ldots, n-1$*,*
  - *so wird* p_n *wahr.*

  Formaler, als komplexere Tautologie:

$$(p_1 \wedge (p_1 \rightarrow p_2) \wedge (p_2 \rightarrow p_3) \wedge \cdots \wedge (p_{n-1} \rightarrow p_n)) \quad \rightarrow \quad p_n$$

  Oft verwendete Schreibweise (wenn auch nicht ganz korrekt):

$$(p_1 \wedge (p_1 \Rightarrow p_2 \Rightarrow \cdots \Rightarrow p_n)) \Rightarrow p_n$$

  mit Metasymbol '$\Rightarrow$ *für Implikationsketten*

1.2 Syntax der Aussagenlogik

Notwendig für Computer-gestützte Verarbeitung von Logik:

- Reformulierung der Grundbegriffe der Aussagenlogik,
- jetzt aber sauber und streng formal.

Wichtig ist dabei Unterschied zwischen:

- **Syntax**= System von Regeln, nach denen Ausdrücke aus einem Zeichenvorrat gebildet werden dürfen (Über eine inhaltliche Bedeutung der Zeichen wird hier nicht gesprochen!)
- **Semantik**= Festlegung der Bedeutung von Ausdrücken, die syntaktisch richtig aufgebaut sind

Hier zunächst nur: Syntax! (Genaue Definition der Semantik folgt erst in späterem Kapitel) Wähle als Zeichenvorrat z.B. die folgenden 8 Zeichen:

$$\boldsymbol{Z} = \{\ \texttt{A}\ ,\ \texttt{0}\ ,\ \texttt{1}\ ,\ \texttt{(}\ ,\ \texttt{)}\ ,\ \vee\ ,\ \wedge\ ,\ \neg\ \}$$

Definition 1.2.1 (Formalisierte Syntax der Aussagenlogik)**.**

- *Die Menge $\boldsymbol{V}$ der atomaren Formeln (bzw. Aussagenlogischen Variablen) besteht aus allen Zeichenfolgen, die mit `A1` beginnen und danach nur noch `0`en und `1`en enthalten, d.h.* $\boldsymbol{V} := \{\texttt{A1}, \texttt{A10}, \texttt{A11}, \texttt{A100}, \texttt{A101}, \texttt{A110}, \texttt{A111}, ...\}$
- *Die Menge $\boldsymbol{WFF}$ der (wohlgeformten aussagenlogischen) Formeln ist wie folgt induktiv definiert:*
  1. *Alle atomaren Formeln sind Formeln.*
  2. *Ist $\boldsymbol{F}$ Formel, so auch '$\neg\boldsymbol{F}$' (Negation).*
  3. *Sind $\boldsymbol{F_1}$ und $\boldsymbol{F_2}$ Formeln, so auch '$(\boldsymbol{F_1} \wedge \boldsymbol{F_2})$' (Konjunktion).*
  4. *Sind $\boldsymbol{F_1}$ und $\boldsymbol{F_2}$ Formeln, so auch '$(\boldsymbol{F_1} \vee \boldsymbol{F_2})$' (Disjunktion).*

  Nur das, was mit diesen vier Regeln in endlich vielen Schritten aufgebaut werden kann, gehört zu WFF!

'Induktive Definition' bedeutet:

> Nur das, was mit diesen vier Regeln in endlich vielen Schritten aufgebaut werden kann, ist eine Formel.

Wird eine Formel $\boldsymbol{F}$ unter Verwendung einer Formel $\boldsymbol{G}$ aufgebaut, so heißt $\boldsymbol{G}$ Teilformel von $\boldsymbol{F}$.

Für jede Formel $\boldsymbol{F}$ gibt es also eine Zahl $\boldsymbol{n}$ und eine Menge $\{\boldsymbol{F_1}, \boldsymbol{F_2}, \ldots, \boldsymbol{F_n}\}$ von Formeln mit folgenden Eigenschaften:

- Jedes $\boldsymbol{F_i}$ ist atomar oder wie folgt aufgebaut:
  - $\boldsymbol{F_i} = \neg\boldsymbol{F_j}$ für ein $\boldsymbol{j} < \boldsymbol{i}$ oder
  - $\boldsymbol{F_i} = (\boldsymbol{F_j} \wedge \boldsymbol{F_k})$ für ein $\boldsymbol{j} < \boldsymbol{i}$ und ein $\boldsymbol{k} < \boldsymbol{i}$ oder
  - $\boldsymbol{F_i} = (\boldsymbol{F_j} \vee \boldsymbol{F_k})$ für ein $\boldsymbol{j} < \boldsymbol{i}$ und ein $\boldsymbol{k} < \boldsymbol{i}$.
  - Es ist $\boldsymbol{F} = \boldsymbol{F_n}$.
- Stets enthält $\{\boldsymbol{F_1}, \boldsymbol{F_2}, \ldots, \boldsymbol{F_i}\}$ also alle Teilformeln von $\boldsymbol{F_i}$(aber evtl. auch noch weitere Formeln!)

Beispiel: '`(((A1∨A10)∧¬(¬A10∧A11))∧(A1∨A111))`' $\in \boldsymbol{WFF}$, denn:

1. $\boldsymbol{F_1}$ ='`A1`' $\in \boldsymbol{WFF}$ (Regel 1)
2. $\boldsymbol{F_2}$ ='`A10`' $\in \boldsymbol{WFF}$ (Regel 1)
3. $\boldsymbol{F_3}$ ='`A11`' $\in \boldsymbol{WFF}$ (Regel 1)
4. $\boldsymbol{F_4}$ ='`A111`' $\in \boldsymbol{WFF}$ (Regel 1)
5. $\boldsymbol{F_5}$ ='`¬A10`' $\in \boldsymbol{WFF}$ ($\boldsymbol{F_2}$, Regel 2)

6. $\boldsymbol{F_6}$ =‘(A1∨A10)’ $\in \boldsymbol{WFF}$ ($\boldsymbol{F_1}$, $\boldsymbol{F_2}$, Regel 4)
7. $\boldsymbol{F_7}$ =‘(¬A10∧A11)’ $\in \boldsymbol{WFF}$ ($\boldsymbol{F_3}$, $\boldsymbol{F_5}$, Regel 3)
8. $\boldsymbol{F_8}$ =‘(A1∨A111)’ $\in \boldsymbol{WFF}$ ($\boldsymbol{F_1}$, $\boldsymbol{F_4}$, Regel 4)
9. $\boldsymbol{F_9}$ =‘¬(¬A10∧A11)’ $\in \boldsymbol{WFF}$ ($\boldsymbol{F_7}$, Regel 2)
10. $\boldsymbol{F_{10}}$ =‘((A1∨A10)∧¬(¬A10∧A11))’ $\in \boldsymbol{WFF}$ ($\boldsymbol{F_6}$, $\boldsymbol{F_9}$, Regel 3)
11. $\boldsymbol{F_{11}}$ =‘(((A1∨A10)∧¬(¬A10∧A11))∧(A1∨A111))’ $\in \boldsymbol{WFF}$ ($\boldsymbol{F_8}$, $\boldsymbol{F_{10}}$, Regel 3)

Man nutzt die Ziffernfolgen im Namen von Variablen als Binärzahlen und als Index, z.B. A101 wird geschrieben als $\boldsymbol{A_5}$.

Damit ist

$$\boldsymbol{V = \{A_1, A_2, A_3, ...\}}$$

und

(((A1∨A10)∧ ¬(¬A10∧A11))∧(A1∨A111))

entspricht damit in vereinfachter Form:

$$\boldsymbol{(((A_1 \vee A_2) \wedge \neg(\neg A_2 \wedge A_3)) \wedge (A_1 \vee A_7))}$$

Abkürzende Schreibweisen und weitere Bezeichnungen:

- $\boldsymbol{A, B, C, ...}$ an Stelle von $\boldsymbol{A_1, A_2, A_3, ...}$
- $\boldsymbol{(F_1 \rightarrow F_2)}$ an Stelle von $\boldsymbol{(\neg F_1 \vee F_2)}$
- $\boldsymbol{(F_1 \leftrightarrow F_2)}$ an Stelle von $\boldsymbol{((F_1 \wedge F_2) \vee (\neg F_1 \wedge \neg F_2))}$

$\boldsymbol{WFF}$ programmiert als Java-Klasse, erste Fassung:

```
1  public class WFF {
2
3    public static class Var extends WFF {
4      String name;
5      public Var (String name){ this.name = name; }
6      @Override public String toString(){ return name; }
7    }
8
9    public static class Not extends WFF {
10     WFF f;
11     public Not(WFF f){ this.f = f;}
12     @Override public String toString(){ return  "¬"+f; }
13   }
14
15   public static class And extends WFF {
16     WFF f1, f2;
17     public And(WFF f1, WFF f2){this.f1=f1; this.f2=f2;}
18     @Override public String toString(){return "("+f1+"∧"+f2+")";}
19   }
20
21   public static class Or extends WFF {
```

```
    WFF f1, f2;
    public Or(WFF f1, WFF f2){this.f1=f1; this.f2=f2;}
    @Override public String toString(){return "("+f1+"∨"+f2+")";}
  }
```

```
  public static WFF var(String s)       {return new Var(s);}
  public static WFF not(WFF f)          {return new Not(f);}
  public static WFF and(WFF f1, WFF f2) {return new And(f1,f2);}
  public static WFF or (WFF f1, WFF f2) {return new Or (f1,f2);}


  public static void main(String[] args) {
    //¬(X∧(Y∨Z))
    WFF f = not(and(var("X"),or(var("Y"),var("Z"))));
    System.out.println(f);

    //(((A1∨A10)∧¬(¬A10∧A11))∧(A1∨A111))
    WFF f1  = var("A1");     WFF f2  = var("A10");
    WFF f3  = var("A11");    WFF f4  = var("A111");
    WFF f5  = not(f2);       WFF f6  = or (f1,f2);
    WFF f7  = and(f5,f3);    WFF f8  = or (f1,f4);
    WFF f9  = not(f7);       WFF f10 = and(f6,f9);
    WFF f11 = and(f10,f8);
    System.out.println(f11);
  }
}
```

Parser für $\boldsymbol{WFF}$:

```
import java.util.Scanner;
import java.util.StringTokenizer;
import java.util.Stack;

public class WFFparser {

    public static WFF inv_and(WFF f1, WFF f2) {
      return WFF.and(f2,f1);
    }
    public static WFF inv_or (WFF f1, WFF f2) {
      return WFF.or(f2,f1);
    }

    public static WFF parse(String text){
      StringTokenizer t = new StringTokenizer("("+text+")",
                          "¬()∨∧",true);
      Stack<Character> ops = new Stack<>();
      Stack<WFF>  forms = new Stack<>();
```

```
while (t.hasMoreTokens()) {
  String s = t.nextToken();
  char tk= s.charAt(0);
```

```
  if ( tk=='¬' || tk=='(' ){
    ops.push(tk);
    continue;
  }
  if ( tk=='∨' || tk == '∧' || tk==')' ){
    while( !(ops.peek()=='(' || (ops.peek()=='∨' && tk == '∧'))) {
      char op = ops.pop();
      if ( op == '¬' ) forms.push(WFF.not(forms.pop()));
      if ( op == '∨' ) forms.push(inv_or(forms.pop(),forms.pop()));
      if ( op == '∧' ) forms.push(inv_and(forms.pop(),forms.pop()));
    }
  }
  if ( tk == ')' ) {
    ops.pop();
  } else  if ( tk == '∨' || tk == '∧' ){
    ops.push(tk);
  } else {
    forms.push(WFF.var(s));
  }
}
```

```
     WFF f=forms.pop();
      if (!ops.empty() || !forms.empty())
            throw new RuntimeException("error in formula");
      return f;
    }

    public static void test(String s) {
        WFF f = parse(s);
        System.out.println("String: " + s);
        System.out.println("WFF:    " + f);
        System.out.println();
    }

    public static void main(String[] args) {

      test ( "((a1∨a2)∧¬(¬a3∨¬a4))" );
      test ( "¬¬((a1∨a2∨a3∧a4∨a5)∧¬¬¬¬(¬a3∨¬a4))" );
      test ( "a1∨¬a2∨¬a3∧¬¬¬a4∧a5∨a6∨a7" );
      test ( "((a1∨¬a2∨¬a3)∧(¬¬¬a4∧a5∨a6∨a7))" );
    }
}
```

1.3 Begriffe der Prädikatenlogik

Prädikatenlogik ist eine Erweiterung der Aussagenlogik.

Exakte Definitionen von Syntax und Semantik kommen erst am Semesterende, hier zunächst nur vereinfachte Fassung!

Beispiel für einfache pradikatenlogische Formeln

$$P(x) := \text{`}x > 3 \wedge x < 5\text{'}$$

oder

$$Q(x, y) := \text{`}\cos(x) = \sin(y)/2\text{'}$$

(mit einen Universum, in dem '3', '$>$', '$+$', ... existieren)

Eine prädikatenlogische Formel enthält (zumindest) Symbole für:

- Variablen: 'x', 'y', ...
- Konstanten: '2', '3', 'π', 'e', ...
- Funktionen: '$\sin$', '$\cos$', '$\sqrt{}$', ...
- (Funktionen in Infix-Schreibweise: '$+(x, y)$' als '$x + y$')
- elementare Prädikate (meist auch Infix): '$=$', '$<$', '$\geq$', ...
- Aussagenlogische Junktoren: '$\wedge$', '$\vee$', '$\neg$', ...

Prädikate:

- Funktionen, die Wahrheitswerte als Resultate haben,
- elementar wie etwa '$\leq$' (bzw. '$x < y$')
- aber auch über zusammengesetzte Terme wie '$x > 3 \wedge x < 5$'

Bekannte Verwendung:

- Übliche Schreibweise zur Angabe von Mengen über einem Universum U

  $$\{\ x \in U \mid P(x)\ \}$$

  mit einer Variablen x und einem Prädikat P und der Bedeutung:

  Menge aller x aus U, bei denen '$P(x)$' wahr ergibt

- Beispiele (mit Universum $\mathbb{Z}$):

  $$\{\ n \in \mathbb{Z} \mid n > 0\ \} \quad (= \{1, 2, 3, \ldots\})$$
  $$\{\ n \in \mathbb{Z} \mid n \cdot n < 5\ \} \quad (= \{-2, -1, 0, 1, 2\})$$

Zusätzliche Komponenten bei Prädikatenlogik: **Quantoren**

Ziel: Formulierung von Sachverhalten wie

- Für alle ... gilt ...
- Für ein ... gilt ...
- Es gibt mindestens zwei ... mit ...
- Es gibt genau ein ... mit ...
- Es gibt kein ... mit ...

Übliche Symbole dafür: $\forall$ und $\exists$

Beispiel 1.3.1 (Universelle Quantifizierung / Allquantifizierung)**.** *Durch universelle Quantifizierung über eine Variable x wird aus einem Prädikat P ein neues Prädikat, mit folgender Schreibweise:*

$$\forall x \in U \;\; P$$

in Worten: Alle x aus U haben die Eigenschaft P oder Für alle x aus U gilt P.

Enthält '$\forall x \in U \;\; P$' keine weitere 'freie' Variable, so entsteht sogar eine Aussage, deren Wert aber vom Universum U abhängen kann:

Beispiel:

- $U = \mathbb{Z}$, *alle ganzen Zahlen* $\rightsquigarrow \forall x \in U \;\; (x > 1 \to x \geq 2)$ *ist richtig.*
- $U = \mathbb{Q}$, *alle rationalen Zahlen* $\rightsquigarrow \forall x \in U \;\; (x > 1 \to x \geq 2)$ *ist falsch.*

Sprechweisen:

- $\forall$ *wird Allquantor genannt*
- *Bei* $\forall x \in U \ldots$ *ist* x *die quantifizierte Variable*
- *Meist schreiben wir* $P(x)$*, um deutlich zu machen, dass* x *die (bzw. eine) Leerstelle in* P *ist. Für einen Wert* $u \in U$ *bedeutet* $P(u)$*, dass* u *in* P *für* x *eingesetzt werden soll.*
- *Ist das Universum* U *endlich, d.h., ist* $U = \{u_1, \ldots, u_r\}$*, so gilt:*

$$\forall x \in U \;\; P(x) \quad \equiv \quad P(u_1) \wedge P(u_2) \wedge \cdots \wedge P(u_r).$$

  Dann schreibt man auch: $\bigwedge_x P(x)$ *oder* $\bigwedge_{x \in U} P(x)$
- *Weitere übliche Schreibweisen sind: '*$(\forall x \in U) \;\; P(x)$*' und '*$\forall x \in U \;:\; P(x)$*'*
- *Ist* U *eindeutig festgelegt, schreibt man auch kurz '*$\forall x \; P(x)$*'*

Beispiel 1.3.2 (Existentielle Quantifizierung)**.** *Durch existentielle Quantifizierung über eine Variable x wird aus einem Prädikat P ein neues Prädikat, mit folgender Schreibweise*

$$\exists x \in U \;\; P$$

in Worten: Es gibt (mindestens) ein x in U mit der Eigenschaft P.

Enthält '$\exists x \in U \;\; P$' keine weitere 'freie' Variable, so entsteht wiederum eine Aussage, deren Wert aber erneut vom Universum U abhängen kann.

Beispiel:

- $U = \mathbb{Z}$, *alle ganzen Zahlen* $\rightsquigarrow \exists x \in U \;\; (x > 1 \wedge x < 2)$ *ist falsch.*
- $U = \mathbb{Q}$, *alle rationalen Zahlen* $\rightsquigarrow \exists x \in U \;\; (x > 1 \wedge x < 2)$ *ist richtig.*

Sprechweisen:

- $\exists$ *wird Existenzquantor genannt.*
- *Bei* $\exists x \in U \;\; \ldots$ *ist* x *die quantifizierte Variable*

- *Ist die Menge U endlich, d.h. ist $U = \{u_1, \ldots, u_r\}$, so gilt:*

$$\exists x \in U \;\; P(x) \quad \equiv \quad P(u_1) \vee P(u_2) \vee \cdots \vee P(u_r).$$

  Dann schreibt man auch: $\bigvee_x P(x)$ oder $\bigvee_{x \in U} P(x)$

- *Weitere übliche Schreibweisen sind: '$(\exists x \in U) \;\; P(x)$' oder '$\exists x \in U \;:\; P(x)$' bzw. '$\exists x \;\; P(x)$'*

Beispiel 1.3.3 (Werte quantifizierter Aussageformen)**.**

- *$\forall x \in \mathbb{Z} \;\; (x > 1 \to x \geq 2)$ ist richtig, denn das innere Prädikat $(x > 1 \to x \geq 2)$ ist bei jedem $x \in \mathbb{Z}$ erfüllt (d.h. wahr).*
- *$\forall x \in \mathbb{Q} \;\; (x > 1 \to x \geq 2)$ ist falsch, denn das innere Prädikat $(x > 1 \to x \geq 2)$ ist zum Beispiel bei $x = \frac{3}{2}$ verletzt (d.h. falsch).*
- *$\exists x \in \mathbb{Z} \;\; (x > 1 \wedge x < 2)$ ist falsch, da $(x > 1 \wedge x < 2)$ von keinem $x \in \mathbb{Z}$ erfüllt wird.*
- *$\exists x \in \mathbb{Q} \;\; (x > 1 \wedge x < 2)$ ist richtig, da $x = \frac{3}{2}$ das Prädikat erfüllt.*

Satz 1.3.4 (de Morgan)**.** *Für quantifizierte Formeln gelten die folgenden de Morgan'schen Regeln:*

$$\begin{aligned} \neg(\forall x \in X \;\; P(x)) \quad &\equiv \quad \exists x \in X \;\; \neg P(x) \\ \neg(\exists x \in X \;\; P(x)) \quad &\equiv \quad \forall x \in X \;\; \neg P(x) \end{aligned}$$

Definition 1.3.5. *Wird in einer Aussageform P eine Variable x verwendet, so kann diese Verwendung in 'freier' oder 'gebundener' Form auftreten.*

- *Hat P keine Quantoren, so ist jedes Vorkommen jeder Variablen x in P 'frei'.*
- *Ist ein Vorkommen von x in P frei, so ist das Vorkommen von x in $Qx \; P$ nicht mehr frei, sondern an den Quantor Q gebunden.*

Beispiel 1.3.6. *Betrachte ein Prädikat $P(x, y)$ mit zwei Variablen x, y.*

- *$\exists x \in U \;\; P(x, y)$ ist Aussageform, x ist gebunden, y ist frei.*
- *$\forall y \in U \;\; P(x, y)$ ist Aussageform, x ist frei, y ist gebunden.*
- *$\forall x \in U \;\; \exists y \in U \;\; P(x, y)$ ist Aussage, beide Variablen sind gebunden: x an $\forall$ und y an $\exists$.*

Anmerkungen:

- Variablen können sowohl frei als auch gebunden auftreten:

$$x < 1 \vee \exists x \in U \;\; x < 2$$

- Eine Belegung modifiziert nur das freie Vorkommen von Variablen.
- Tritt eine Variable in gebundener Form auf, so können wir ihre Werte dort nicht mehr frei belegen.

  Setzen wir also z.B. 3 für x ein, so erhalten wir (die Aussage...)

$$3 < 1 \vee \exists x \in U \;\; x < 2$$

- Der gleiche Variablenname kann an verschiedenen Stellen einer Aussageform sogar an verschiedene Quantoren gebunden sein:

$$\forall x \in U \quad (x < 1 \vee \exists x \in U \;\; x < 2)$$

- Zur besseren Lesbarkeit sollten (gebundene) Variablen umbenannt werden, wenn sie in unterschiedlicher Form verwendet werden, also besser:

$$\forall x \in U \;\; (x < 1 \vee \exists y \in U \;\; y < 2)$$

Achtung: "$\forall x \exists y \neq \exists y \forall x$", dazu ein Beispiel:

Beispiel 1.3.7. *Aus Analysis bekannt: Eine reelle Zahl a heißt Grenzwert einer Folge (a_n) reeller Zahlen gdw.*

$$\forall \varepsilon \in \mathbb{R}^+ \;\; \exists n_0 \in \mathbb{N}_0 \;\; \forall n \in \mathbb{N}_0 \;\; (n \geq n_0 \to |a_n - a| < \varepsilon)$$

In Worten: Zu jedem $\varepsilon > 0$ gibt es ein n_0, so dass für alle n ...

Vertauschen der Quantoren liefert die folgender Aussage (mit vollkommen anderer Bedeutung!):

$$\exists n_0 \in \mathbb{N}_0 \;\; \forall \varepsilon \in \mathbb{R}^+ \;\; \forall n \in \mathbb{N}_0 \;\; (n \geq n_0 \to |a_n - a| < \varepsilon)$$

In Worten: Es gibt ein n_0, so dass für alle $\varepsilon > 0$ und alle n ...

Hinweis:

$$\exists y \;\; \forall x \;\; P(x, y) \quad \text{impliziert} \quad \forall x \;\; \exists y \;\; P(x, y)$$

Beispiel 1.3.8 (Formalisierung eines 'Rätsels')**.** *Ist der folgende Schluss logisch korrekt?*

(A) Weder Samson noch Freunde von Samson schießen ein Tor.
(B) Ein Tor wird von Samson oder Johannes geschossen.
(C) Also ist Johannes kein Freund von Samson.

Formalisierung:

- *Das Universum U ist eine Fußballmannschaft, aus der namentlich S=Samson und J=Johannes bekannt sind.*
- *$T(x)$ ist eine Aussageform mit der Bedeutung: x schießt ein Tor.*
- *$F(x, y)$ ist eine Aussageform mit der Bedeutung: x und y sind Freunde.*
- *damit folgende Formalisierung:*

  (A) $\neg T(S) \wedge \forall x \; (F(x, S) \to \neg T(x))$
  (B) $T(S) \vee T(J)$
  (C) $\neg F(J, S)$

- *Wir haben zu zeigen: Aus (A) und (B) folgt (C).*
- *Als Hilfsaussage zeigen wir mit (A) und (B): Es gilt $T(J)$!*
- *Zunächst: Aus (A) folgt durch Vereinfachung: $\neg T(S)$.*
- *Zusammen mit (B) gilt daher: $\neg T(S) \wedge (T(S) \vee T(J))$.*
- *Das Distributivgesetz liefert: $(\neg T(S) \wedge T(S)) \vee (\neg T(S) \wedge T(J))$.*
- *Identitätsgesetz für Oder $\rightsquigarrow \neg T(S) \wedge T(J)$.*
- *Durch Vereinfachung folgt also: $T(J)$.*
- *Aus dem zweiten Teil der Aussage (A) folgt durch Spezialisierung: $F(J, S) \to \neg T(J)$.*
- *Umkehrschluss (und doppelte Negation) liefert: $T(J) \to \neg F(J, S)$.*
- *Mit der Hilfsaussage folgt durch modus ponens die Behauptung.*

2 Beweistechniken

2.1 einfachste Beweismethoden

Ziel eines Beweises:

Zeige (unmissverständlich!), dass eine Aussage wahr ist!

Probleme dabei:

- Was ist "unmissverständlich"?

  Für Informatiker: Beweis sollte maschinell verifizierbar sein!

  - Coq proof assistant
  - MIZAR project ("Formalized Mathematics")
  - HOL light / ISABELLE

- Was ist eine Aussage?

  Notwendig: Saubere Formalisierung der Logik...

Beispiele für komplexere (fast) formalisierte Beweise:

- Jordanscher Kurvensatz (formal in HOL light, 2007)
- Fließkomma-Arithmetik von INTEL-CPUs, Coquelicot (Coq, 2015)
- Keplers Vermutung über optimale Kugelpackungen (Flyspeck-Projekt, HOL light, 2014)

Künstliche Intelligenz und **korrektes Formulieren**:

- Large language models sind aktuell auf dem Stand eines sehr schwachen Schülers, der nicht in der Lage ist selber zu denken aber sehr viel Hintergrundlektüre betrieben hat.
- Large language models beherrschen Schulmathematik, weil:
  - "Schulmathematik" ein sehr kleiner und abgeschlossener Bereich der Mathematik ist,
  - es einen "Beurteilungsalgorithmus" gibt, nach welchem gelernt werden kann,
  - es Unmengen an Inhalten zu den exakt gleichen Fragen auf dem Internet gibt.
- Ein Mensch kann die Schulmathematik bestehen, ohne wirklich ein Verständnis davon zu haben, was er/sie macht.
- Ein large language model kann das auch!
- folgende Folien: "Unterhaltung" von Prof. Staub mit ChatGTP...

Kannst du mir beweisen, dass es unendlich viele Primzahlen gibt?

Ja, ich kann Ihnen einen Beweis dafür geben, dass es unendlich viele Primzahlen gibt. Dieser Beweis wurde erstmals von dem antiken griechischen Mathematiker Euklid formuliert und wird als der "Beweis durch Widerspruch" bezeichnet.

Annahme: Angenommen, es gäbe nur endlich viele Primzahlen, nämlich p_1, p_2, ..., p_k (wobei k eine endliche Anzahl von Primzahlen ist).

Jetzt können wir eine neue Zahl bilden, die größer ist als alle diese Primzahlen. Wir multiplizieren einfach alle vorhandenen Primzahlen und fügen 1 hinzu:

$N = (p_1 * p_2 * ... * p_k) + 1$

Jetzt müssen wir zwei Fälle betrachten:

1. Wenn N selbst eine Primzahl ist, haben wir eine neue Primzahl gefunden, die nicht in unserer ursprünglichen Liste enthalten ist. Dies wäre ein Widerspruch zu unserer Annahme, dass es nur endlich viele Primzahlen gibt.
2. Wenn N keine Primzahl ist, muss es einen Primfaktor haben. Dieser Primfaktor kann jedoch nicht in unserer ursprünglichen Liste der Primzahlen enthalten sein, da er N teilt und somit größer sein muss als jede dieser Primzahlen. Auch dies wäre ein Widerspruch zu unserer Annahme.

Da in beiden Fällen ein Widerspruch auftritt, muss unsere ursprüngliche Annahme, dass es nur endlich viele Primzahlen gibt, falsch sein. Daher gibt es tatsächlich unendlich viele Primzahlen.

Dieser Beweis zeigt, dass es immer mehr Primzahlen gibt, und es gibt keine Obergrenze für die Anzahl der Primzahlen.

Kannst du beweisen, dass es unendlich viele Primzahlen gibt, die mit einer 7 enden?

Ja, ich kann Ihnen einen Beweis dafür geben, dass es unendlich viele Primzahlen gibt, die mit einer 7 enden. Dies ist ein Beispiel für einen Beweis durch Widerspruch:

Angenommen, es gäbe nur endlich viele Primzahlen, die auf eine 7 enden. Diese könnten wir als $p_1, p_2, ..., p_k$ bezeichnen (wobei k eine endliche Anzahl von solchen Primzahlen ist).

Jetzt können wir eine neue Zahl bilden, die sicherstellt, dass sie nicht durch diese endlichen Primzahlen teilbar ist. Wir betrachten die Zahl:

$N = 10^n + 7$

Hierbei wählen wir n so groß, dass N größer ist als jede der Primzahlen $p_1, p_2, ..., p_k$. Da 10^n immer mit einer 0 endet und wir dann 7 hinzufügen, endet N definitiv mit einer 7.

Jetzt müssen wir zwei Fälle betrachten:

1. Wenn N eine Primzahl ist, haben wir eine neue Primzahl gefunden, die auf eine 7 endet und nicht in unserer ursprünglichen Liste enthalten ist. Dies widerspricht unserer Annahme, dass es nur endlich viele solcher Primzahlen gibt.
2. Wenn N keine Primzahl ist, hat sie einen Primfaktor, der nicht in unserer Liste enthalten ist. Dieser Primfaktor muss selbst auf eine 7 enden, da N mit einer 7 endet. Auch dies widerspricht unserer Annahme.

Da in beiden Fällen ein Widerspruch auftritt, müssen wir schließen, dass unsere ursprüngliche Annahme falsch ist, und daher gibt es tatsächlich unendlich viele Primzahlen, die auf eine 7 enden.

kannst du zeigen, dass es unendlich viele Primzahlen gibt, die auf einer 7 enden und den Teiler 7 haben?

Ja, ich kann Ihnen zeigen, dass es unendlich viele Primzahlen gibt, die auf einer 7 enden und den Teiler 7 haben. Dies kann mit einem klassischen Beweis durch Widerspruch getan werden.

Angenommen, es gäbe nur eine endliche Anzahl solcher Primzahlen. Wir nehmen an, diese Primzahlen seien p_1, p_2, ..., p_n, wobei n eine endliche Anzahl ist. Diese Primzahlen enden alle auf 7 und sind Teiler von 7. Das bedeutet, dass jede dieser Primzahlen durch 7 teilbar ist, was wiederum bedeutet, dass sie in der Form 7k geschrieben werden können, wobei k eine ganze Zahl ist.

Betrachten Sie nun das Produkt P = p_1 * p_2 * ... * p_n. Da jedes p_i in der Form 7k geschrieben werden kann, kann P in der Form 7^nk geschrieben werden, wobei n die Anzahl der Primzahlen ist und k das Produkt der k_i-Werte für jedes p_i ist.

Nun, P ist also eine positive ganze Zahl, die durch 7^nk teilbar ist. Betrachten wir P + 7. Da P durch 7^nk teilbar ist, ist P + 7 nicht durch 7^nk teilbar. Dies bedeutet jedoch, dass P + 7 entweder eine Primzahl ist oder einen Primfaktor hat, der nicht in p_1, p_2, ..., p_n enthalten ist. In beiden Fällen erhalten wir einen Widerspruch zur Annahme, dass p_1, p_2, ..., p_n die einzigen Primzahlen sind, die auf 7 enden und den Teiler 7 haben.

Da die Annahme, dass es nur eine endliche Anzahl solcher Primzahlen gibt, zu einem Widerspruch führt, müssen unendlich viele solcher Primzahlen existieren. Dies beweist, dass es unendlich viele Primzahlen gibt, die auf einer 7 enden und den Teiler 7 haben.

Beweise in der “Praxis”:

- Form der zu beweisenden Aussage meist:
$$p \to q$$
- Manchmal ist p nicht ausdrücklich angegeben, sondern nur (implizit) z.B. durch *Grundannahmen* (*Axiome*) gegeben.
- Es "hilft" auch manchmal die Tautologie $q \equiv (t \to q)$.
- Manchmal sind auch *Äquivalenzaussagen* zu zeigen: $p \leftrightarrow q$: Hierzu verwendet man meist die Tautologie
$$(p \leftrightarrow q) \equiv ((p \to q) \wedge (q \to p))$$
- Oft werden keine Aussagen, sondern Aussageformen angegeben, etwa (für $x \in \mathbb{N}_0$):
$$\mathbf{6} \text{ teilt } x \to \mathbf{3} \text{ teilt } x$$
Dann wird implizit über alle freien Variablen all-quantifiziert, d.h zu zeigen ist dann:
$$\forall x \in \mathbb{N}_0 \ \ (\ \mathbf{6} \text{ teilt } x \to \mathbf{3} \text{ teilt } x)$$

Beispiel 2.1.1. **Einfachste Beweise** *der Form $p \to q$ (z.B. interessant für "Induktionsanfänge")*

- *Inhaltsleerer Beweis: Wir zeigen: p ist stets falsch.*
Beispiel: Betrachte Aussageformen
$$p(n) := (n > 1) \quad , \quad q(n) := (n^3 > n)$$
Für $p(0) \to q(0)$ reicht "inhaltsleerer Beweis" ("Nichts zu zeigen.").
- *Trivialer Beweis: Wir zeigen: q ist wahr, unabhängig von p.*
Beispiel: Betrachte
$$p := (0 < a \leq b) \quad , \quad q(n) := (a^n \leq b^n)$$
Dann $q(0) = (a^0 \leq b^0) = (1 \leq 1)$ ✓ unabhängig von p

Eine Behauptung von der Form $p \to q$ ist nur dann falsch, wenn p wahr ist und q falsch.
Zu untersuchen ist also nur der Fall (die "Annahme"): p ist wahr.
Aus dieser Annahme muss jetzt gefolgert werden, dass q wahr ist...

Beispiel 2.1.2 (Direkte Beweise 1)**.** *Für beliebiges $x \in \mathbb{N}_0$ zeigen wir die Behauptung:*
$$\mathbf{6}|x \to \mathbf{3}|x$$

$y|x$ bedeutet: y ist Teiler von x, d.h. $\exists k \in \mathbb{N}_0 \ \ y \cdot k = x$
Verbale Ausformulierung des Beweises von $\mathbf{6}|x \to \mathbf{3}|x$:

1. *Es sei $\mathbf{6}|x$.*
2. *Dann gibt es ein k mit $\mathbf{6} \cdot k = x$.*
3. *Also gibt es auch ein k mit $\mathbf{3} \cdot (\mathbf{2} \cdot k) = x$.*
4. *Mit $k' = \mathbf{2} \cdot k$ erhalten wir: Es gibt k' mit $\mathbf{3} \cdot k' = x$.*
5. *Es ist daher $\mathbf{3}|x$.* □

Direkte Beweise haben oft die Form von Schlussketten, auf die dann der *modus ponens* angewendet wird:

Beispiel 2.1.3 (Direkte Beweise 2)**.**

Behauptung wieder:

$$\mathbf{6}|x \rightarrow \mathbf{3}|x$$

"Formale" Form des obigen Beweises:

1. *Annahme:* $\mathbf{6}|x$.
2. $\mathbf{6}|x \;\rightarrow\; \exists k \in \mathbb{N}_0 \;\; (\mathbf{6} \cdot k = x)$
3. $\exists k \in \mathbb{N}_0 \;\; (\mathbf{6} \cdot k = x) \;\rightarrow\; \exists k \in \mathbb{N}_0 \;\; (\mathbf{3} \cdot (\mathbf{2} \cdot k) = x)$
4. $\exists k \in \mathbb{N}_0 \;\; (\mathbf{3} \cdot (\mathbf{2} \cdot k) = x) \;\rightarrow\; \exists k \in \mathbb{N}_0 \;\; (\mathbf{3} \cdot k = x)$
5. $\exists k \in \mathbb{N}_0 \;\; (\mathbf{3} \cdot k = x) \;\rightarrow\; \mathbf{3}|x$
6. *Mit Modus Ponens:* $\mathbf{3}|x$. □

Noch formaler:

$$\begin{array}{lrcl} & & & \mathbf{6}|x \\ \wedge(& \mathbf{6}|x & \rightarrow & \exists k \in \mathbb{N}_0 \;\; (\mathbf{6} \cdot k = x) \quad) \\ \wedge(& \exists k \in \mathbb{N}_0 \;\; (\mathbf{6} \cdot k = x) & \rightarrow & \exists k \in \mathbb{N}_0 \;\; (\mathbf{3} \cdot (\mathbf{2} \cdot k) = x)) \\ \wedge(& \exists k \in \mathbb{N}_0 \;\; (\mathbf{3} \cdot (\mathbf{2} \cdot k) = x) & \rightarrow & \exists k \in \mathbb{N}_0 \;\; (\mathbf{3} \cdot k = x) \quad) \\ \wedge(& \exists k \in \mathbb{N}_0 \;\; (\mathbf{3} \cdot k = x) & \rightarrow & \mathbf{3}|x \quad) \end{array}$$

Übliche Schreibweise in Kurzform ('Schlußkette")

$$\begin{array}{rcl} \mathbf{6}|x & \Rightarrow & \exists k \in \mathbb{N}_0 \;\; (\mathbf{6} \cdot k = x) \\ & \Rightarrow & \exists k \in \mathbb{N}_0 \;\; (\mathbf{3} \cdot (\mathbf{2} \cdot k) = x) \\ & \Rightarrow & \exists k \in \mathbb{N}_0 \;\; (\mathbf{3} \cdot k = x) \\ & \Rightarrow & \mathbf{3}|x \end{array}$$

Achtung: '$\Rightarrow$' ist hier Metasymbol und nicht der Junktor der Implikation! Später: Komplexere Beweise in Baum-Form sowie Symbole $\models$ bzw. $\vdash$

2.2 indirekte Beweise

Beispiel 2.2.1 (indirekter Beweis, reductio ad absurdum))**.**

Behauptung:

> *Für jede natürliche Zahl* t *und jede natürliche Zahl* n *gilt: Wenn* $t \geq 2$ *und wenn* t *ein Teiler von* n *ist, dann ist* t *kein Teiler von* $n + 1$.

Mögliche Formalisierung (mit Universum $\mathbb{N}_0$*):*

$$\forall n \;\; \forall t \;\; [t \geq \mathbf{2} \;\; \rightarrow \;\; ((t \mid n) \rightarrow \neg(t \mid n + \mathbf{1}))]$$

Wähle t, n *beliebig mit* $t \geq 2$*. Führe einen Widerspruchsbeweis für die verbleibende Implikation:*

- *Annahme:* $(t \mid n) \wedge (t \mid n + \mathbf{1})$.

- *Dann* $t \mid (n+1) - n$ *(Rechnen mit Teilern)*
- *Damit* $t \mid 1$ *(Rechnen mit Teilern)*
- *Also* $t = 1$ *(Universum* $\mathbb{N}_0$*)*
- *Das ist jedoch Widerspruch zu* $t \geq 2$ *!*

Allgemeine Hinweise zu indirekten Beweisen

Bekannte Aussagen:

- Tautologie $(p \to q) \equiv ((p \wedge \neg q) \to f)$
- klassisches Falsum: $r \wedge \neg r$

Beim indirekten Beweis von $p \to q$ haben wir nun:

- Ist p nicht (explizit) vorhanden, setzen wir $p = t$; um q zu beweisen, zeigen wir daher $\neg q \to f$.
- Ein Beweis durch Kontraposition ist auch ein Widerspruchsbeweis: Mit $\neg q \to \neg p$ gilt auch: $(p \wedge \neg q) \to (p \wedge \neg p)$
- Umgekehrt: Ein Widerspruchsbeweis der Form $\neg q \to f$ ist auch ein Umkehrschluss der Implikation $(t \to q)$, d.h. von q.

2.3 Fallunterscheidungen

Erinnerung: Tautologie $p \equiv (q \to p) \wedge (\neg q \to p)$.

Beispiel 2.3.1 (Beweise mit Fallunterscheidungen)**.**

Wegen der o.a. Tautologie können wir für beliebiges q *die folgenden Fälle getrennt untersuchen: 1.* q *ist wahr; 2.* q *ist falsch.Für* $a \in \mathbb{N}_0$ *beweise Beispiel-Behauptung* p*:* a^2 *geteilt durch vier hat entweder Rest Eins oder Rest Null. Wähle als* q*: '*a *ist gerade', damit ergibt sich:*

- *Fall* a *ist gerade: Dann* $\exists k : (a = 2k)$*, und damit* $a^2 = (2k)^2 = 4k^2$*, d.h.* $4 \mid a^2$*, also ist* p *erfüllt.*
- *Fall* a *ist ungerade: Dann* $\exists k : (a = 2k+1)$*, damit* $a^2 = (2k+1)^2 = 4k^2 + 4k + 1$ *und* $4 \mid (a^2 - 1)$*, also ist wieder* p *erfüllt.*

Da sowohl $(q \to p)$ *als auch* $(\neg q \to p)$ *gilt, folgt also* p*.*

Ist (für beliebige $q_1, \ldots q_n$) die Aussageform $q_1 \vee q_2 \vee \ldots \vee q_n$ immer wahr, dann gilt:

$$\begin{aligned} p &\equiv t \to p \\ &\equiv (q_1 \vee q_2 \vee \ldots \vee q_n) \to p \\ &\equiv \neg(q_1 \vee q_2 \vee \ldots \vee q_n) \vee p \\ &\equiv (\neg q_1 \wedge \neg q_2 \wedge \ldots \wedge \neg q_n) \vee p \\ &\equiv (\neg q_1 \vee p) \wedge (\neg q_2 \vee p) \wedge \ldots \wedge (\neg q_n \vee p) \\ &\equiv (q_1 \to p) \wedge (q_2 \to p) \wedge \ldots \wedge (q_n \to p) \end{aligned}$$

- Fallunterscheidungen können also mehr als zwei Fälle umfassen.
- Beweise mit vielen Fallunterscheidungen sind aber in der Regel nicht sehr elegant (und eignen sich nicht als Beispiele für eine Grundvorlesung).

2.4 Quantoren

Beispiel 2.4.1 (Umgang mit Allquantoren)**.** *Viele mathematische Aussagen haben die Gestalt von Allaussagen:*

$$\forall x \ (p(x) \to q(x)) \qquad (*)$$

Man beweist sie, indem man die Aussage $p(a) \to q(a)$ für jedes a aus dem zugrundeliegenden Universum beweist.

Übliche Formulierungsweisen dabei sind:

- *Startfloskel: "a sei im folgenden beliebig (aus dem Universum) gewählt."*
- *Dann folgt der Beweis von $p(a) \to q(a)$ (wobei a wie eine Variable behandelt wird, d.h. man verwendet eigentlich Aussageformen, aber nicht Aussagen!)*
- *Abschlussfloskel: "Da a beliebig gewählt worden war, folgt damit $(*)$."*

Im vorigen Beispiel war "x beliebig gewählt", bewiesen wurde damit $\forall x \in \mathbb{N}_0 \ ((6|x) \to (3|x))$ (nur die Abschlussfloskel fehlte noch...)

Zusammengesetzt ergäbe sich für die Behauptung

$$\forall x \in \mathbb{N}_0 \ ((6|x) \to (3|x))$$

z.B. der folgende Beweis:

1. *$x \in \mathbb{N}_0$ sei zunächst beliebig gewählt.*
2. *Es reicht, den Fall $6|x$ zu betrachten.*
3. *Dann gibt es ein k mit $6 \cdot k = x$.*
4. *Also gibt es auch ein k mit $3 \cdot (2 \cdot k) = x$.*
5. *Mit $k' = 2 \cdot k$ erhalten wir: Es gibt k' mit $3 \cdot k' = x$.*
6. *Es ist daher $3|x$.*
7. *Da x ansonsten beliebig gewählt war, gilt damit die Behauptung.* □

Beweise von *Existenzaussagen* $\exists x \ q(x)$:

Solche Aussagen sind oft *konstruktiv* beweisbar, d.h. es wird ein x angegeben ("konstruiert"), für dass $q(x)$ gilt.

Beispiel 2.4.2 (Umgang mit Existenzquantoren)**.**

$n \in \mathbb{N}_0$ *heißt zusammengesetzt gdw.* $\exists t \in \mathbb{N}_0 \ ((1 < t < n) \wedge (t|n))$

Beispiel-Behauptung: Zu jedem $n \in \mathbb{N}_0$ gibt es n aufeinander folgende zusammengesetzte Zahlen. Etwas formalere Schreibweise:

$$\forall n \in \mathbb{N}_0 \ \exists k \in \mathbb{N}_0 \ \forall i \in \{1, \ldots, n\} \ (k+i \text{ ist zusammengesetzt})$$

Beweis: :

1. *(Startfloskel:) Sei n beliebig.*
2. *(Konstruktionsschritt:) Wähle $k = (n+1)! + 1$.*

3. *(Beweisschritt(e):) Für dieses* k *gilt:* $\forall i \in \{1,\ldots,n\}\ ((i+1) \mid (k+i))$*, denn stets* $(i+1)|(k-1)$ *und* $(i+1)|(i+1)$.

4. *(Abschlussfloskel:) Da* n *beliebig war, folgt die Behauptung.*

Erinnerung: $(p \to q) \equiv (\neg q \to \neg p)$ ist eine Tautologie!

Beispiel 2.4.3 (Beweis durch Umkehrschluss (Kontraposition))**.**
Behauptung: Ist eine Quadratzahl ungerade, so auch ihre Wurzel.
Ausführliche Schreibweise:

- *Es sei* a *eine natürliche Zahl.*
- *Wenn* a^2 *eine ungerade Zahl ist, dann ist* a *ungerade.*

Formuliert in Prädikatenlogik:

- $p(a) := (a^2$ *ist ungerade*$)$
- $q(a) := (a$ *ist ungerade*$)$
- *Behauptung ist also:* $\forall a \in \mathbb{N}_0\ (p(a) \to q(a))$
- *Kontraposition:* $\forall a \in \mathbb{N}_0\ (\neg q(a) \to \neg p(a))$.

Zu zeigen ist also:

> *Es sei* a *eine natürliche Zahl. Wenn* a *keine ungerade Zahl ist, dann ist* a^2 *nicht ungerade.*

Dies ist offensichtlich eine komplizierte Formulierung für:

> *Ist* a *gerade, so auch* a^2*. (dabei: Eine Zahl* a *heißt gerade gdw.* $2 \mid a$*.)*

Beweis dazu: Bei beliebigem $a \in \mathbb{N}_0$ *gilt:*

$$a \text{ gerade } \to\ \exists k\ (a = 2k)\ \to\ \exists k\ (a \cdot a = (2k) \cdot a))\ \to\ \exists k\ (a^2 = 2 \cdot (ka)) \to a^2 \text{ gerade.}$$

Da a *beliebig gewählt war, haben wir damit gezeigt:*

$$\forall a \in \mathbb{N}_0\ (p(a) \to q(a))$$

Beispiel 2.4.4 (Beweis mit Quantoren-Negation)**.**
Satz von Euklid: Es gibt unendlich viele Primzahlen.

- $\mathbb{P} \subseteq \mathbb{N}_0$ *sei Menge der Primzahlen, d.h.*

$$n \in \mathbb{P} \quad \equiv \quad \forall m \in \mathbb{N}_0\ (\ m|n \to (m = 1 \vee m = n))$$

  (O.B.d.A. sei $n = 1$ *ebenfalls Primzahl...)*

- *Formalisierte Fassung des Satzes von Euklid:*

$$\forall k\ \exists n\ ((k < n) \wedge n \in \mathbb{P})$$

- *Wir wollen ihn indirekt beweisen, d.h. die folgende Negation des Satzes widerlegen (de Morgan!):*

$$\exists k \ \forall n \ ((n \leq k) \vee n \notin \mathbb{P})$$

- *Diese Aussage ist äquivalent zu:*

$$\exists k \ \forall n \ (n \in \mathbb{P} \rightarrow (n \leq k))$$

Annahme sei also: $\exists k \ \forall n \ (n \in \mathbb{P} \rightarrow (n \leq k))$ k_0 *sei einer der Werte, deren Existenz angenommen wird, d.h. es gelte:* $\forall n \ (n \in \mathbb{P} \rightarrow (n \leq k_0))$ *(*)*

Anmerkungen:

- k_0 *ist "fest gewählt", also keine Variable, sondern (unbekannter) Wert...*
- *Wir wollen* $\forall n \ (n \in \mathbb{P} \rightarrow (n \leq k_0))$ *widerlegen, d.h. wir wollen zeigen, dass* $\exists n \ (n \in \mathbb{P} \wedge n > k_0)$

(Konstruktionsschritt:) Setze $b := 1 + (1 \cdot 2 \cdot 3 \cdot \ldots \cdot k_0)$. *(Beweisschritte z.B.:) Sei* a *der kleinste Teiler von* b*, der nicht* 1 *ist; damit sicher* $a \in \mathbb{P}$*. Sicher ist auch* $j \nmid b$ *für alle* $2 \leq j \leq k_0$*; damit andererseits* $a > k_0$*. Zusammen* $a \in \mathbb{P} \wedge a > k_0$*, ein Widerspruch zu (*)Die Annahme ist also falsch; damit ist die Negation der Annahme wahr! (Anmerkung: Dieser Beweis wäre auch leicht direkt durchführbar...)*

2.5 Schubfachprinzip

Beispiel 2.5.1 (Das Schubfachprinzip von Dirichlet)**.**

Falls man n *Gegenstände auf* k *Schubfächer verteilt,*
wobei $n > k > 0$ *gilt,*
dann gibt es mindestens ein Fach,
in dem mehr als ein Gegenstand landet.

Indirekter Beweis des Schubfachprinzips:

- *Ansonsten befände sich in jedem Fach nur ein Gegenstand;*
- *damit wäre aber die Zahl* n *der Gegenstände maximal gleich der Zahl* k *der Fächer,*
- *was ein Widerspruch zur Voraussetzung ist, dass es mehr Gegenstände als Schubfächer gibt...*

Beispiel 1:

- *In einer Gruppe von acht Leuten haben (mindestens) zwei am gleichen Wochentag Geburtstag.*
- *dabei:* $n = 8$ *Leute (=Gegenstände) und* $k = 7$ *Wochentage (=Schubfächer)*

Beispiel 2:

- p_1*,* p_2*,* p_3 *seien Aussagen. Unter ihnen gibt es zwei mit gleichem Wahrheitswert.*
- *dabei:* $n = 3$ *Aussagen (=Gegenstände) und* $k = 2$ *Wahrheitswerte (=Schubfächer)*

- *Also haben wenigstens zwei Aussagen den gleichen Wahrheitswert...*

Verallgemeinerung des Schubfachprinzipes:

Lemma 2.5.2. *Werden $n > k$ Gegenstände auf k Fächer verteilt, so gibt es mindestens ein Fach, das mindestens $\lceil \frac{n}{k} \rceil$ Gegenstände enthält.*

Dabei ist für $x \in \mathbb{R}$ der Wert $\lceil x \rceil$ ("obere Gaußklammer") definiert als kleinste ganze Zahl j mit $j \geq x$, also z.B. $\lceil \frac{99}{100} \rceil = 1$ und $\lceil \frac{100}{99} \rceil = 2$.

Indirekter Beweis des Lemma:

Wenn in jedem Fach maximal $\lceil \frac{n}{k} \rceil - 1$ Gegenstände wären, so könnten die Fächer insgesamt nicht mehr Gegenstände enthalten als

$$k \cdot \left(\left\lceil \frac{n}{k} \right\rceil - 1 \right) \;<\; k \cdot \left(\left(\frac{n}{k} + 1 \right) - 1 \right) \;=\; n$$

Widerspruch! □

2.6 Natürliche Zahlen und Induktionsbeweise

Definition 2.6.1 (Peano-Axiome, (1889) Giuseppe Peano)**.**

Die natürlichen Zahlen $\mathbb{N}_0$ sind wie folgt charakterisiert:

(1) 0 *ist eine natürliche Zahl.*

(2) Zu jeder natürlichen Zahl n gibt es genau einen Nachfolger n', der ebenfalls eine natürliche Zahl ist.

(3) Es gibt keine natürliche Zahl, deren Nachfolger 0 ist.

(4) Zwei verschiedene natürliche Zahlen n und m besitzen stets verschiedene Nachfolger n' und m'.

(5) Induktionsaxiom: Enthält eine Menge X die Zahl 0 und mit jeder natürlichen Zahl n auch stets deren Nachfolger n', so enthält X bereits alle natürlichen Zahlen.

- der Nachfolger n' wird i.d.R. mit $n + 1$ bezeichnet...
- eigentlich von Richard Dedekind (1888) in "Was sind und was sollen die Zahlen ?"
- Enthält die im Induktionsaxiom genannte Menge X nur natürliche Zahlen (also $X \subseteq \mathbb{N}_0$), dann gilt sogar $X = \mathbb{N}_0$.

Aus dem Induktionsaxiom ergibt sich:

(nach Dedekind/Peano:)	$\mathbb{N}_0 = \{0, 0', (0')', ((0')')', \dots\}$
(über Addition von 1)	$\mathbb{N}_0 = \{0, 0{+}1, 0{+}1{+}1, 0{+}1{+}1{+}1, \dots\}$
Bequemer/gewohnter:	$\mathbb{N}_0 = \{0, 1, 2, 3, \dots\}$

Durch Axiome nahegelegte Umgangsweisen:

- Induktion: Objekte werden "der Reihe nach" aufgelistet (bei Zahlen: $0, 1, 2, 3, \dots$)
- Rekursion: Objekte sind entweder elementar (wie die Zahl 0) oder zusammengesetzt (wie die Nachfolger von Zahlen).

Hauptanwendung der Induktionsaxioms: Induktive Beweise!

- p sei ein Prädikat mit einer Variablen beim Universum $\mathbb{N}_0$.
- Betrachte Menge X aller Zahlen, für die $p(n)$ wahr ist.

$$X = \{n \in \mathbb{N}_0 \mid p(n)\}$$

- Laut Induktionsaxiom: Falls $0 \in X$ und für jedes $n \in X$ auch $n+1 \in X$, so $X = \mathbb{N}_0$.
- Anders formuliert: Falls $p(0)$ und stets $p(n) \to p(n+1)$, dann $p(n)$ für alle $n \in \mathbb{N}_0$.
- Daraus abgeleitet: Beweisschema für viele Aussagen der Form

$$\forall n \in \mathbb{N}_0 \;\; p(n)$$

Beispiel 2.6.2 (mathematische Induktion)**.** *Ziel ist zu zeigen, dass gilt*

$$\forall n \in \mathbb{N}_0 \;\; p(n)$$

Dabei sei $p(n)$ eine Aussageform, die von $n \in \mathbb{N}_0$ abhängt. Nach dem Induktionsaxiom reicht es aus, die folgende Aussage zu beweisen:

$$p(0) \quad \wedge \quad \forall n \in \mathbb{N}_0 \;\; p(n) \to p(n+1)$$

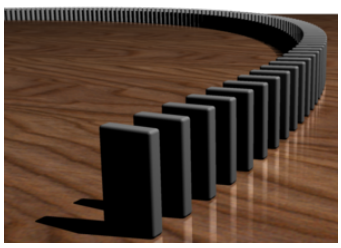
Induktionsbeweise haben also (meist) folgende Teile:

1. *Den Nachweis der Gültigkeit von $p(0)$ (Induktionsanfang, I.A.)*
2. *Den Nachweis der Gültigkeit von $\forall n \in \mathbb{N}_0 \;\; p(n) \to p(n+1)$*
   (Allaussage; meist angegangen wie im Beispiel 2.4.1):
   (a) *Betrachte beliebiges $n \in \mathbb{N}_0$.*
   (b) *Zeige $p(n) \to p(n+1)$ (Induktionsschritt).*
   Oft als direkter Beweis analog zu Beispiel 2.1.2:
   Unter Annahme der Induktionsvoraussetzung (I.V.) $p(n)$ wird die Konklusion (Induktionsbehauptung, I.B.) $p(n+1)$ nachgeweisen.

Schematischer Aufbau eines Beweises von $\forall n \in \mathbb{N}_0 \;\; p(n)$ mit Induktion:

1. *I.A.: Für den Fall $n = 0$ gilt ...(+).*
2. *n sei im folgenden beliebig aus $\mathbb{N}_0$.*
3. *I.V.: Wir nehmen an, dass $p(n)$ (*) gilt.*
4. *I.B.: Zu zeigen ist daher $p(n+1)$ (**).*
5. *Die I.B. gilt, da ... (++).*
6. *Da n beliebig gewählt war, gilt nach dem Induktionsprinzip damit $\forall n \in \mathbb{N}_0 \;\; p(n)$.*

Empfehlung: Schema genau(!) übernehmen, d.h. erst (+) ausfüllen, dann (*) und (**) ausformulieren(!), schließlich (++) ausfüllen.



Der Dominoeffekt als Induktion:

1. I.A.: Der vorderste Dominostein wird umgeworfen.
2. I.S.: Die Aufstellung gewährleistet:

   Wenn der n-te Dominostein in der Reihe fällt, so auch der $n+1$-te.

Folgerung: Alle Steine fallen um....

Zur Gültigkeit des **Induktionprinzips:** WARUM stimmt eine induktiv bewiesene Aussage $\forall n \in \mathbb{N}_0 \ p(n)$?

Naive Antwort:

- $p(0)$ ist wahr: Wegen des Ankers...
- $p(1)$ ist wahr: $p(0)$ ist wahr; $p(0) \rightarrow p(1)$ ist Spezialfall des Induktionsschritts, also folgt $p(1)$ (modus ponens!).
- $p(2)$ ist wahr: $p(1)$ ist wahr; $p(1) \rightarrow p(2)$ ist Spezialfall des Induktionsschritts, also folgt $p(2)$ (modus ponens!).
- usw...

Formal richtige Antwort:

- Das Induktionsaxiom *postuliert* die Gültigkeit für ganz $\mathbb{N}_0$.
- Aber: Das Induktionsaxiom selbst ist nicht beweisbar!!!

Beispiel 2.6.3 (Induktionsbeweis)**.** *Beobachtung:*

$$\begin{array}{lll} 1 & = 1 & = 1^2 \\ 1+3 & = 4 & = 2^2 \\ 1+3+5 & = 9 & = 3^2 \\ 1+3+5+7 & = 16 & = 4^2 \\ \ldots & & \\ 1+3+5+7+\ldots+2n-1 & & = n^2 \end{array}$$

Behauptung:

Für alle $n \in \mathbb{N}_0$ *gilt* $\sum_{i=1}^{n}(2i-1) = n^2 \quad (*)$

Hier ist also $p(n) \equiv (\sum_{i=1}^{n}(2i-1) = n^2)$*, wir müssen zeigen:* $\forall n \in \mathbb{N}_0 \ p(n)$*. Wir wenden das genannte Schema an:*

1. *I.A.: Für den Fall* $n = 0$ *gilt* $\sum_{i=1}^{0}(2i-1) = 0 = 0^2$.

   (Leere Summe ist als 0 *definiert. Tipp: auch Fall* $n = 1$ *testen!)*
2. n *sei im folgenden beliebig aus* $\mathbb{N}_0$.
3. *I.V.: Wir nehmen an, dass* $p(n)$*, also* $\sum_{i=1}^{n}(2i-1) = n^2$ *gilt.*
4. *I.B.: Zu zeigen ist daher* $p(n+1)$*, d.h.* $\sum_{i=1}^{n+1}(2i-1) = (n+1)^2$.
5. *Diese I.B. gilt, da:*

$$\begin{array}{rcll} (n+1)^2 & = & n^2+2n+1 & \textit{(binomischer Lehrsatz)} \\ & = & (\sum_{i=1}^{n}(2i-1))+2n+1 & \textit{(mit I.V.)} \\ & = & \sum_{i=1}^{n+1}(2i-1) & \end{array}$$

6. *Da n beliebig gewählt war, gilt nach dem Induktionsprinzip damit $\forall n \in \mathbb{N}_0 \; p(n)$, d.h. die obige Behauptung $(*)$.*

Beispiel 2.6.4 (Induktion und Computerprogramme)**.** *Ziel: Untersuchung der Eigenschaften eines Computerprogrammes.*

Wähle als Schleifeninvariante $S(i,k) \equiv (k = (i-1)^2)$. Zu zeigen ist, dass nach beliebiger Zahl n von Schleifendurchläufen stets $S(i,k)$ für aktuelle Werte von i und k gilt.

1. *Lies die natürliche Zahl m ein.*
2. *Setze $k := 0$ und $i := 1$. (hier $n = 0$)*
   I.A.: { Für aktuelle i und k gilt $S(i,k)$: $0 = (1-1)^2$) ✓ }
3. **Solange $i \leq m$ wiederhole**: *(Induktionsschritt von n auf $n+1$)*
   I.V.: { Für aktuelle i und k gilt $S(i,k)$, also $k = (i-1)^2$. }
   $k := k + 2i - 1$;
   $i := i + 1$
   Zeige: Für neue i und k gilt wieder $S(i,k)$, d.h. aus
   $S(i,k)$ für 'alte' Werte i, k muss auch folgen:
   $S(i',k')$ für 'neue' Werte $i' = i+1$, $k' = k+2i-1$
   I.S.: $\{k' = k+2i-1 =^{I.V.} (i-1)^2 + 2i - 1 = i^2 = (i'-1)^2\}$
4. *Gib k aus.*
   Beim Schleifenende gilt: $i = m+1$. Also: $k = m^2$

Wichtige Variante der Induktionsbeweise für $\forall n \in \mathbb{N}_0 \; p(n)$

- Integration des Induktionsanfanges $p(0)$ in den Induktionsschritt
- Zugriff auf $p(m)$ für zusätzliche Werte $m < n$ im Induktionsschritt.

Beispiel 2.6.5 (vollständige (mathematische) Induktion)**.** *Der Nachweis von $\forall n \in \mathbb{N}_0 \; p(n)$ wird durch den Nachweis von*

$$\forall n \in \mathbb{N}_0 \; (\; [\; \forall m \in \mathbb{N}_0 \; (\; m < n \to p(m) \;) \;] \to p(n))$$

geführt. Dies ist äquivalent zur 'einfachen' Induktion, jedoch:

- *Kein expliziter Induktionsanfang notwendig.*
- *Induktionsvoraussetzung $\forall m \in \mathbb{N}_0 \; (m < n \to p(m))$ und Induktionsbehauptung $p(n)$ sind von unterschiedlicher Struktur.*
- *Wir haben stärkere(?) Voraussetzungen zum Beweis von $p(n)$.*

Schematischer Aufbau eines Beweises von $\forall n \in \mathbb{N}_0 \; p(n)$ mit Induktionbeweis der Form $\forall m \in \mathbb{N}_0 \; (m < n \to p(m)) \to p(n)$

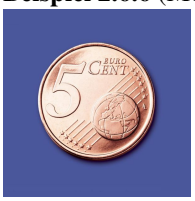
1. *n sei im folgenden beliebig aus $\mathbb{N}_0$.*
2. *I.V.: Wir nehmen an, dass $\forall m \in \mathbb{N}_0 \; (m < n \to p(m))$ (*) gilt.*
3. *I.B.: Zu zeigen ist nun $p(n)$ (**).*
4. *Fallunterscheidung:*
   - *Fall $n = 0$: 'Trivialer' Beweis von $p(0)$.(+)*

- *Fall $n > 0$: (Direkter) Beweis von $p(n)$ unter Verwendung von $p(0)$, $p(1)$, ..., $p(n-1)$.(++)*

5. *Da n beliebig gewählt war, gilt nach dem Induktionsprinzip damit $\forall n \in \mathbb{N}_0 \; p(n)$.*

Empfehlung: Schema genau(!) übernehmen, d.h. erst (*) und (**) ausformulieren(!), dann schließlich (+) und (++) bearbeiten.

Beispiel 2.6.6 (Money, Money, Money, ...)**.**



Behauptung:
Jeder Geldbetrag $n \geq 4$ Cent kann
unter ausschließlicher Verwendung
von 2- und 5-Cent-Münzen ausbezahlt werden.

1. *Sei n beliebig gegeben.*
2. *Jeder Betrag m mit $4 \leq m < n$ sei auszahlbar.*
3. *Zeige, dass dann auch Betrag n auszahlbar ist, falls $n \geq 4$.*
   - *Falls $n \leq 3$: Nichts zu zeigen...*
   - *Falls $n = 4$ oder $n = 5$: Trivial...*
   - *Falls $n \geq 6$: Für den Spezialfall $m = n - 2$ (mit $4 \leq m < n$) kann ausgezahlt werden, mit einer weiteren 2-Cent-Münze ist damit auch n zahlbar...*
4. *Mit Induktion: Jeder Betrag $n \geq 4$ ist auszahlbar.*

Zur Übung:

- *Reichen immer vier 2-Cent-Stücke aus?*
- *Geht es auch mit nur drei 2-Cent-Stücken?*

3 Boolesche Funktionen

3.1 Boolesche Ausdrücke

Ziel: Was ist die Logik, auf der Computer aufgebaut sind?

($\Rightarrow$ enger Zusammenhang mit Aussagenlogik...)

George Boole: The Laws of Thought (1854)

Definition 3.1.1. *Eine n-stellige Boolesche Funktion f ist eine Abbildung, die jedem Tupel $(b_1, \ldots, b_n)$ aus $\{0,1\}^n$ einen Wert aus $\{0,1\}$ ('Bits') zuordnet:*

$$f : \{0,1\}^n \to \{0,1\}$$

Eine n-stellige Boolesche Funktion transformiert also n-Bit-Vektoren in einzelne Bits.

Beispiel 3.1.2. *Angabe von Booleschen Funktionen z.B. über Wertetabelle:*

b_1	b_2	b_3	$f_1(b_1,b_2,b_3)$	$f_2(b_1,b_2,b_3)$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(f_1 ist Übertrag und f_2 ist Summe bei der Addition dreier Binärziffern)

Spezielle Funktionen:

Definition 3.1.3.

- *Boolesches Komplement (Negation, einstellig), Schreibweisen $\overline{b}$ und $\neg b$, mit*

$$\overline{0} = \neg 0 = 1, \quad \overline{1} = \neg 1 = 0$$

- *Boolesches Produkt (zweistellig), Schreibweise: $b_1 \cdot b_2$, mit*

$$0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0, \quad 1 \cdot 1 = 1$$

- *Boolesche Summe (zweistellig), Schreibweise: $b_1 + b_2$ mit*

$$0 + 0 = 0, \quad 0 + 1 = 1 + 0 = 1 + 1 = 1$$

Anmerkung: Identifiziert man 0 mit f und 1 mit w, so entspricht

- das Produkt der Konjunktion und
- die Summe der Disjunktion von Wahrheitswerten.

Anzahl n-stelliger Boolescher Funktionen:

- Es gibt 2^n mögliche Vektoren als Parameter, dabei
- jeweils 2 Möglichkeiten der Werte pro Parametervektor,

- also gibt es $\mathbf{2^{2^n}}$ $\boldsymbol{n}$-stellige Boolesche Funktionen.

$\boldsymbol{n=0}$	$\boldsymbol{2^{2^0}=2}$
$\boldsymbol{n=1}$	$\boldsymbol{2^{2^1}=4}$
$\boldsymbol{n=2}$	$\boldsymbol{2^{2^2}=16}$
$\boldsymbol{n=3}$	$\boldsymbol{256}$
$\boldsymbol{n=4}$	$\boldsymbol{65.536}$
$\boldsymbol{n=5}$	$\boldsymbol{4.294.966.416}$
$\boldsymbol{n=6}$	$\boldsymbol{18.446.744.073.709.551.616}$

- Wie findet man Bezeichnungen für alle diese Funktionen?
- Wie kann man sie möglichst kurz, aber eindeutig aufschreiben?
- Wie kann man sie grafisch veranschaulichen?

Im folgenden: Boolesche Ausdrücke, d.h. textuelle Repräsentation Boolescher Funktionen

Wähle als Zeichenvorrat z.B. die folgenden 8 Zeichen:

$$\boldsymbol{Z} = \{\ \texttt{X}\ ,\ \texttt{0}\ ,\ \texttt{1}\ ,\ \texttt{(}\ ,\ \texttt{)}\ ,\ \texttt{+}\ ,\ \texttt{*}\ ,\ \neg\ \}$$

Definition 3.1.4. • *Die Menge $\boldsymbol{V}$ der 'Booleschen Variablen' besteht aus allen Zeichenfolgen, die mit `X1` beginnen und danach nur noch `0`en und `1`en enthalten, d.h.* $\boldsymbol{V} = \{\texttt{X1}, \texttt{X10}, \texttt{X11}, \texttt{X100}, \texttt{X101}, \texttt{X110}, \texttt{X111}, \ldots\}$

- *Die Menge $\boldsymbol{A}$ der 'Booleschen Ausdrücke' besteht nun aus Zeichenfolgen, die alle wie folgt aufgebaut sind:*

  1. *'`0`' und '`1`' sind Boolesche Ausdrücke.*
  2. *Jede Boolesche Variable $\boldsymbol{v} \in \boldsymbol{V}$ ist ein Boolescher Ausdruck.*
  3. *Ist $\boldsymbol{w}$ Boolescher Ausdruck, so auch '$\neg\boldsymbol{w}$'.*
  4. *Sind $\boldsymbol{v}$ und $\boldsymbol{w}$ Boolesche Ausdrücke, so auch '$(\boldsymbol{v}+\boldsymbol{w})$'.*
  5. *Sind $\boldsymbol{v}$ und $\boldsymbol{w}$ Boolesche Ausdrücke, so auch '$(\boldsymbol{v}*\boldsymbol{w})$'.*

  Nur das, was mit diesen fünf Regeln in endlich vielen Schritten aufgebaut werden kann, ist ein Boolescher Ausdruck!

Schreibweisen zur besseren Lesbarkeit Boolescher Ausdrücke:

- Indizes werden wieder binär interpretiert
- Um Klammern zu sparen, nutzt man die Regel 'Punktrechnung vor Strichrechnung'. Außerdem interpretiert man Operationen auf gleichem Level als von links her geklammert.
- Statt einer Negation $\neg\boldsymbol{w}$ wird oft auch $\overline{\boldsymbol{w}}$ geschrieben. (Dann kann man auch eine innere Klammer weglassen.)
- Das Produktsymbol $*$ wird meist nicht aufgeschrieben.

Damit also '`(((X1+X10)*¬(¬X10*X11))*(X1+1))`' schreibbar als:

$$(\boldsymbol{x_1}+\boldsymbol{x_2})\overline{\overline{\boldsymbol{x_2}}\boldsymbol{x_3}}(\boldsymbol{x_1}+\boldsymbol{1})$$

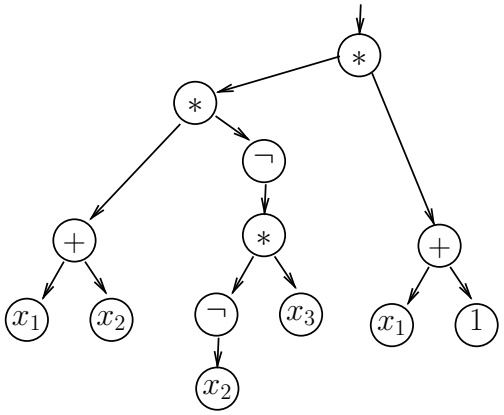
Die obigen Booleschen Ausdrücke entsprechen also fast exakt den wohlgeformten Formeln:

- '$+$' und '$*$' ersetzen '$\vee$' und '$\wedge$'.
- Bits $\{0, 1\}$ ersetzen Wahrheitswerte $\{f, w\}$.
- Konstanten 0 und 1 sind explizit in Ausdrücken erlaubt.

Grafische Darstellung des Aufbaus des Ausdruckes

$$(x_1 + x_2)\overline{\overline{x_2}x_3}(x_1 + 1)$$

in einem 'Syntaxbaum':



Vom Booleschen Ausdruck (Syntax) zur Booleschen Funktion (Semantik):

> Enthält ein Boolescher Ausdruck w nur Variablen mit Index bis höchstens n, so kann er als n-stellige Boolesche Funktion $f_w^{(n)} : \{0,1\}^n \to \{0,1\}$ interpretiert werden!

Definition 3.1.5. *Die von w repräsentierte n-stellige Boolesche Funktion $f_w^{(n)}$ ist wie folgt definiert:*

> *Es ist $f_w^{(n)}(b_1, ..., b_n) = b$ genau dann, wenn bei Einsetzen von b_i für x_i in w und anschließender Auswertung mit Negation, Boolescher Summe und Produkt als Resultat b entsteht.*

Damit können wieder Wahrheitswertetafeln verwendet werden, um einen Überblick über den Werteverlauf der Booleschen Funktion zu erhalten!

Beim Beispiel $w := (x_1 + x_2)\overline{\overline{x_2}x_3}(x_1 + 1)$ erhält man als 3-stellige Boolesche Funktion $f_w^{(3)}$:

x_1	x_2	x_3	(x_1+x_2)	$\cdot$	$\neg(\overline{x_2}\cdot x_3)$			$\cdot(x_1+1)$		$f_w^{(3)}(x_1, x_2, x_3)$
0	0	0	0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1	0	1	0
0	1	0	1	1	1	0	0	1	1	1
0	1	1	1	1	1	0	0	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	0	1	0
1	1	0	1	1	1	0	0	1	1	1
1	1	1	1	1	1	0	0	1	1	1

Definition 3.1.6. *Zwei Boolesche Ausdrücke v und w heißen äquivalent ($v \equiv w$), wenn beide die gleiche n-stellige Boolesche Funktion f darstellen, also $f_v^{(n)}(b_1, .., b_n) = f_w^{(n)}(b_1, .., b_n)$ für alle Bitvektoren $(b_1, .., b_n)$ gilt. Dabei wird n als maximaler Index der in v oder w vorkommenden Variablen gewählt.*

Es ist dabei nicht wichtig, welche Stelligkeit bei f verwendet wird, solange sie ausreichend groß ist, dass alle Variablen belegt werden.

Offensichtlich gilt nämlich:

$$f_v^{(n)}(b_1, .., b_n) = f_w^{(n)}(b_1, .., b_n)$$

dann auch

$$f_v^{(m)}(b_1, .., b_m) = f_w^{(m)}(b_1, .., b_m)$$

für jedes $m \geq n$.

Beispiele:

$$(x_1 + x_2)x_3 \equiv (x_1x_3) + (x_2x_3)$$
$$x_2 + \overline{x}_2 \equiv 1$$

Die logischen Äquivalenzen aus den Beispielen 1.1.23 und 1.1.24 gelten (in angepasster Schreibweise):

Satz 3.1.7 (Boolesche Äquivalenzen)**.** *Für beliebige Boolesche Ausdrücke p, q, r gilt:*

1. *Kommutativgesetze:* $p+q \equiv q+p$; $\quad pq \equiv qp$.
2. *Assoziativgesetze:* $p+(q+r) \equiv (p+q)+r$; $\quad p(qr) \equiv (pq)r$.
3. *Distributivgesetze:* $p(q+r) \equiv pq+pr$; $\quad p+qr \equiv (p+q)(p+r)$.
4. *Gesetze von de Morgan:* $\overline{pq} \equiv \overline{p}+\overline{q}$; $\quad \overline{p+q} \equiv \overline{p}\,\overline{q}$.
5. *Identitätsgesetze:* $p\,1 \equiv p$; $\quad p+0 \equiv p$.
6. $p+q\overline{q} \equiv p$; $\quad p(q+\overline{q}) \equiv p$.
7. *Dominanz:* $p\,0 \equiv 0$; $\quad p+1 \equiv 1$.
8. *Negationsgesetze:* $p\overline{p} \equiv 0$; $\quad p+\overline{p} \equiv 1$.
9. *Doppelte Negation:* $\overline{\overline{p}} \equiv p$.
10. *Idempotenzgesetze:* $pp \equiv p$; $\quad p+p \equiv p$.
11. *Absorptionsgesetze:* $p(p+q) \equiv p$; $\quad p+pq \equiv p$.

3.2 Normalformen

Zu Booleschen Ausdrücken w hatten wir entsprechende Boolesche Funktionen f_w definiert, aber offen ist noch:

Gibt es zu jeder Booleschen Funktion f einen passenden Booleschen Ausdruck w mit $f_w = f$?

Beispiel: Gesucht ist ein Boolescher Ausdruck zu folgendem f:

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	0
1	0	0
1	1	1

Vorgehensweise: Bilde Teilausdrücke für alle 1-en im Werteverlauf, die dann mit Boolescher Summe verknüpft werden.

Im Beispiel: f hat Wert 1 nur bei $(0, 0)$ und $(1, 1)$, damit wird f vom folgenden Ausdruck dargestellt:

$$(\overline{x_1}\,\overline{x_2}) + (x_1x_2)$$

Definition 3.2.1.
- *Eine Boolescher Ausdruck, der nur aus einer Variablen oder einer negierten Variablen besteht, wird als 'Literal' bezeichnet.*
- *Für $b \in \{0,1\}$ und eine Boolesche Variable v definiere das Literal v^b durch*

$$v^b := \begin{cases} v, \text{ falls } b = 1 \\ \overline{v}, \text{ falls } b = 0 \end{cases}$$

- *Für $b = (b_1, ..., b_n) \in \{0,1\}^n$ definiere den Ausdruck $M(b)$ durch*

$$M(b) := x_1^{b_1} \cdot x_2^{b_2} \cdot ... \cdot x_n^{b_n}$$

$M(b)$ wird als der zu b gehörende 'Minterm' bezeichnet.

Beispiel:

$$M(1,0,1,0,1) = x_1\, \overline{x_2}\, x_3\, \overline{x_4}\, x_5$$

Lemma 3.2.2. *Für die durch einen einzelnen Minterm $M(b_1, ..., b_n)$ repräsentierte Boolesche Funktion f gilt:*

$$f(a_1, ..., a_n) = \begin{cases} 1, \text{ falls } a_1 = b_1 \wedge a_2 = b_2 \wedge ... \wedge a_n = b_n, \\ 0, \text{ sonst.} \end{cases}$$

Satz 3.2.3. *Sei f eine n-stellige Boolesche Funktion und $b^{(1)}, ..., b^{(t)}$ seien die Parametervektoren, für die $f(b) = 1$ gilt.*

Dann wird f durch den Booleschen Ausdruck $M(b^{(1)}) + ... + M(b^{(t)})$ repräsentiert.

Im vorigen Beispiel: f hatte Wert 1 nur bei $(0,0)$ und $(1,1)$, Minterme sind also $M(0,0)$ und $M(1,1)$, damit ergibt sich folgender Ausdruck:

$$(\overline{x_1}\, \overline{x_2}) + (x_1 x_2)$$

Definition 3.2.4.
- *Ein Boolescher Ausdruck, der Boolesche Summe von Booleschen Produkten von Literalen ist, wird als 'Disjunktive Normalform' (DNF) bezeichnet.*
- *Ein Boolescher Ausdruck, der Boolesches Produkt von Booleschen Summen von Literalen ist, wird als 'Konjunktive Normalform' (KNF) bezeichnet.*

Beipiel:

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	0
1	0	0
1	1	1

passende DNF: $\overbrace{(\overline{x_1}\, \overline{x_2})}^{\text{zu Zeile 1}} + \overbrace{(x_1 x_2)}^{\text{zu Zeile 4}}$

passende KNF: $\underbrace{(\overline{x_1} + x_2)}_{\text{zu Zeile 3}} \underbrace{(x_1 + \overline{x_2})}_{\text{zu Zeile 2}}$

Satz 3.2.3 sagt also aus:

Satz 3.2.5 (Disjunktive Normalform)**.** *Für jede Boolesche Funktion f gibt es einen Booleschen Ausdruck w in disjunktiver Normalform, der sie repräsentiert, d.h. $f = f_w$.*

Analog gilt:

Satz 3.2.6 (Konjunktive Normalform)**.** *Für jede Boolesche Funktion f gibt es einen Booleschen Ausdruck w in konjunktiver Normalform, der sie repräsentiert, d.h. $f = f_w$.*

Beweis dazu auf zwei Arten möglich:

- durch Verwendung der Wertetabelle (d.h. i.W. über die Semantik der Ausdrücke)
- durch Umformung des Ausdruckes (d.h. i.W. Syntax-basierte Vorgehensweise)

Hier: Erzeugung der KNF durch Umformung von Ausdrücken.

Wende dazu die folgenden fünf Umformungsregeln auf Teile des zu bearbeitenden Ausdruckes an, solange es möglich ist:

$$\begin{array}{ll} \mathbf{1.)} & \\ \mathbf{2.)} & \\ \mathbf{3.)} & \text{Ersetze} \left\{ \begin{array}{c} \neg\neg w \\ \neg(v * w) \\ \neg(v + w) \\ w + (u * v) \\ (u * v) + w \end{array} \right\} \text{durch} \left\{ \begin{array}{c} w \\ (\neg v + \neg w) \\ (\neg v * \neg w) \\ (w + u) * (w + v) \\ (u + w) * (v + w) \end{array} \right\}. \\ \mathbf{4.)} & \\ \mathbf{5.)} & \end{array}$$

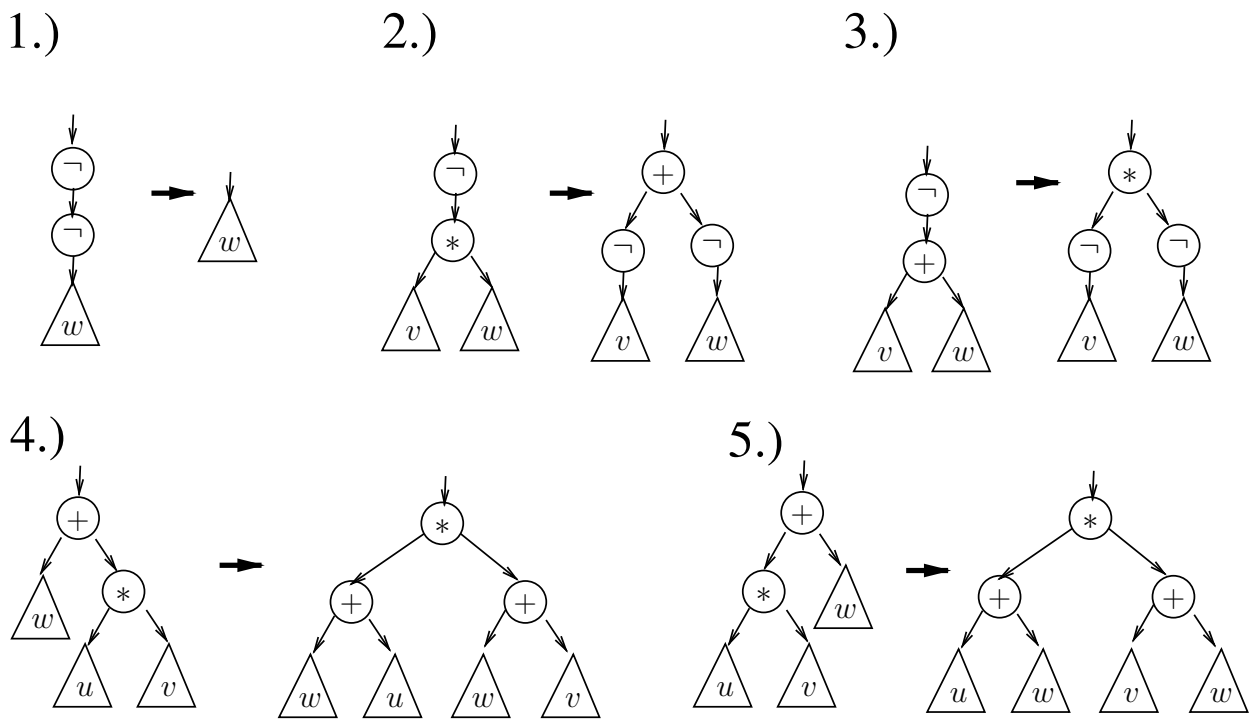
Es wird also bei der Ersetzung immer ein äquivalenter Ausdruck gebildet.

An Ende der Umformung gilt:

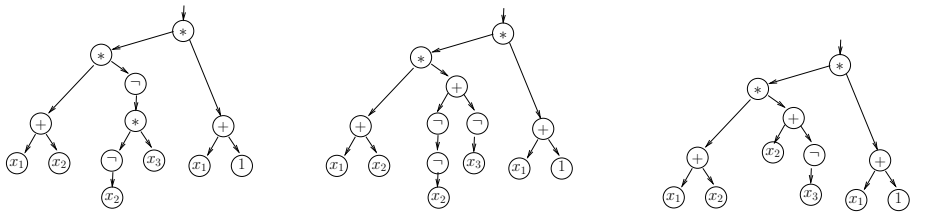
- Negation kann nur noch auf Variablen angewendet werden.
- Boolesche Summen können keine Booleschen Produkte enthalten.

Das Endprodukt der Umformung ist also Boolesches Produkt von Booleschen Summen von Literalen, d.h. eine KNF.

Graphische Darstellung der Umformungsregeln:



Im Beispiel $(x_1 + x_2)\overline{\overline{x}_2 x_3}(x_1 + 1)$ reichen bereits zwei Umformungsschritte zur Erzeugung einer KNF!



$(x_1 + x_2)\overline{\overline{x_2}x_3}(x_1 + 1) \;\rightsquigarrow\; (x_1 + x_2)(\overline{\overline{x_2}} + \overline{x_3})(x_1 + 1) \;\rightsquigarrow\; (x_1 + x_2)(x_2 + \overline{x_3})(x_1 + 1)$

Weiteres Beispiel:

$$\begin{array}{rc} & \overline{x_1}\,\overline{x_2} + x_1x_2 \\ \overset{4.)}{\rightsquigarrow} & (\overline{x_1}\,\overline{x_2} + x_1)(\overline{x_1}\,\overline{x_2} + x_2) \\ \overset{5.)}{\rightsquigarrow} & (\overline{x_1} + x_1)(\overline{x_2} + x_1)(\overline{x_1}\,\overline{x_2} + x_2) \\ \overset{5.)}{\rightsquigarrow} & (\overline{x_1} + x_1)(\overline{x_2} + x_1)(\overline{x_1} + x_2)(\overline{x_2} + x_2) \end{array}$$

Dieser Ausdruck ist bereits KNF, könnte aber noch vereinfacht werden:

$$\begin{array}{rc} \equiv & 1\,(\overline{x_2} + x_1)(\overline{x_1} + x_2)\,1 \\ \equiv & (\overline{x_2} + x_1)(\overline{x_1} + x_2) \end{array}$$

JAVA-Programm zur Konversion eines Ausdrucks in KNF

- Basis: JAVA-Klasse WFF aus Folie 37
- Rekursive Zerlegung einer gegebenen WFF entsprechend ihres Aufbaus (über zwei Stufen!)
- Eingabe immer noch mit '$\wedge$', '$\vee$' und '$\neg$'.

```
1  public class WFFtoCNF {
2    public static WFF toCNF (WFF f) {
3      if ( f instanceof WFF.Not ){
4        WFF.Not notF = (WFF.Not) f; WFF g = notF.f;
5        if ( g instanceof WFF.Not ){// f == not(not(...))
6          WFF.Not notG = (WFF.Not) g;
7          return toCNF(notG.f);
8        }
9        if ( g instanceof WFF.And ){// f == not(and(...,...))
10         WFF.And andG = (WFF.And) g;
11         WFF g1 = andG.f1; WFF g2 = andG.f2;
12         return toCNF(WFF.or(WFF.not(g1),WFF.not(g2)));
13       }
14       if ( g instanceof WFF.Or ){//  f == not(or(...,...))
15         WFF.Or orG = (WFF.Or) g;
16         WFF g1 = orG.f1; WFF g2 = orG.f2;
17         return toCNF(WFF.and(WFF.not(g1),WFF.not(g2)));
18       }
19     }
```

```
    if ( f instanceof WFF.Or ){
        WFF.Or orF = (WFF.Or) f;
        WFF g1 = orF.f1; WFF g2 = orF.f2;
        if ( g2 instanceof WFF.And ){// f == or(...,and(...,...))
          WFF.And andG2 = (WFF.And) g2;
          WFF g21 = andG2.f1; WFF g22 = andG2.f2;
          return toCNF(WFF.and(WFF.or(g1,g21),WFF.or(g1,g22)));
        }
        if ( g1 instanceof WFF.And){// f == or(and(...,...),...))
          WFF.And andG1 = (WFF.And) g1;
          WFF g11 = andG1.f1; WFF g12 = andG1.f2;
          return toCNF(WFF.and(WFF.or(g11,g2),WFF.or(g12,g2)));
        }
        // andere Fälle mit f == or(...,...): Teile umwandeln
        WFF h1= toCNF(g1); WFF h2= toCNF(g2);
        if (h1!=g1 || h2!=g2) return toCNF(WFF.or(h1,h2));
    }
    if ( f instanceof WFF.And ){  // f == and(...,...)
        WFF g1 = ((WFF.And) f).f1; WFF g2 = ((WFF.And) f).f2;
        WFF h1 = toCNF(g1); WFF h2 = toCNF(g2);
        if (h1 !=g1 || h2!= g2) return toCNF(WFF.and(h1,h2));
    }
    return f;
  }
```

```
    public static void test(String s) {
        WFF f = WFFparser.parse(s);
        System.out.println("String:␣" + s);
        System.out.println("WFF:␣␣␣␣" + f);
        System.out.println("CNF:␣␣␣␣" + toCNF(f));
        System.out.println();
    }

   public static void main(String[] args) {
      test ( "((x1∨x2)∧¬(¬x2∧x3)∧(x1∨1))" );
      test ( "((¬x1∧¬x2)∨(x1∧x2))" );
      test ( "((a1∨a2)∧¬(¬a3∨¬a4))" );
      test ( "¬¬((a1∨a2∨a3∧a4∨a5)∧¬¬¬¬(¬a3∨¬a4))" );
      test ( "a1∨¬a2∨¬a3∧¬¬¬a4∧a5∨a6∨a7" );
      test ( "((a1∨¬a2∨¬a3)∧(¬¬¬a4∧a5∨a6∨a7))" );
      test ( "(a1∧a2)∨(a3∧a4)∨(a5∧a6)∨(a7∧a8)∨(a9∧a10)" );
   }
}
```

Anmerkungen:

- Eine wichtige Fragestellung ist oft, ob ein Ausdruck erfüllbar ist, d.h. ob es eine Variablenbelegung gibt, die zu 1 ausgewertet wird.
- Bei einer DNF ist dies ganz einfach feststellbar...

- Eine KNF $\boldsymbol{w}$ hingegen ist Produkt von Summen:
  - Jede Summe stellt eine Bedingung (Klausel) dar.
  - Damit $\boldsymbol{w}$ erfüllt ist, muss jede dieser Bedingungen erfüllt sein!
  - Also: In jeder Klausel muss mindestens ein Literal erfüllt sein!

  Auf KNF-Ausdrücke trifft man daher sehr oft!
- Ein trivialer Algorithmus zum Erfüllbarkeitstest einer KNF könnte bei $\boldsymbol{n}$ Variablen theoretisch alle $\boldsymbol{2^n}$ Möglichkeiten ausprobieren.
- Dies dauert aber bereits für kleine $\boldsymbol{n}$ (etwa $\boldsymbol{n = 100}$) zu lange: Ein 2-GHz-Rechner bräuchte ca. 20 Billionen Jahre für $\boldsymbol{2^{100}}$ Tests.
- Ob es wesentlich schnellere Algorithmen ('polynomiale' Laufzeit, d.h. $\boldsymbol{n^k}$ statt $\boldsymbol{2^n}$) gibt, ist ein ungelöstes Problem der Informatik (das 'P=NP'-Problem).

JAVA-Programm zur Erstellung einer Wahrheitswertetafel bzw. einem einfachen Tests auf Erfüllbarkeit:

```
1  import java.util.ArrayList;
2
3  public class WFFtt {
```

Teil (1): Bestimmung der Variablen in einer Formel:

```
5     public static void variables(WFF f, ArrayList<String> v) {
6          if ( f instanceof WFF.Var)  {
7              String s = ((WFF.Var) f).name;
8              if (! v.contains(s) ) v.add(s);
9          } else if ( f instanceof WFF.Not ) {
10             variables(((WFF.Not) f).f,v);
11         } else if ( f instanceof WFF.And ) {
12             variables(((WFF.And) f).f1,v);
13             variables(((WFF.And) f).f2,v);
14         } else if ( f instanceof WFF.Or) {
15             variables(((WFF.Or)  f).f1,v);
16             variables(((WFF.Or)  f).f2,v);
17         }
18         return;
19     }
20     public static ArrayList<String> variables(WFF f) {
21         ArrayList<String> v =  new ArrayList<>();
22         variables(f,v);
23         return v;
24     }
```

Teil (2): Auswertung einer Zeile der Wahrheitswertetafel:

```
1  public static boolean eval(WFF f, ArrayList<String> v, boolean[] values) {
2      if ( f instanceof WFF.Not ) return !eval(((WFF.Not) f).f,v,values);
3      if ( f instanceof WFF.And )
4         return eval(((WFF.And) f).f1,v,values)&& eval(((WFF.And) f).f2,v,values);
5      if ( f instanceof WFF.Or  )
6         return eval(((WFF.Or ) f).f1,v,values)|| eval(((WFF.Or ) f).f2,v,values);
7      String s= ((WFF.Var) f).name;
8      for (int i=0; i< values.length; i++) if (v.get(i).equals(s)) return values[i];
9      return false;
10 }
```

Teil (3): Auswertung und Ausgabe der gesamten Tabelle:

```
1  public static void evalRecursive(WFF f,ArrayList<String> v,boolean[] values,int n) {
2      if (n==values.length-1) {
3          System.out.print("|");
4          for (int i=0; i< values.length; i++) System.out.printf("␣%5s␣|",values[i]);
5          System.out.printf("|␣%5s␣|\n",   eval(f,v,values)   );
6          return;
7      }
8      n=n+1;
9      values[n]=false; evalRecursive(f,v,values,n);
10     values[n]=true;  evalRecursive(f,v,values,n);
11 }
12 public static void truthTable(WFF f) {
13     ArrayList<String> v = variables(f);
14     System.out.print("|");
15     for(String s:v) System.out.printf("␣%5s␣|",s);
16     System.out.printf("|␣␣␣f␣␣␣|\n");
17     boolean[] values = new boolean[v.size()];
18     evalRecursive(f,v,values,-1);
19 }
```

Teil (4): Auswertung OHNE Ausgabe der Tabelle:

```
public static boolean satisfiableRecursive(WFF f,
                ArrayList<String> v, boolean[] values, int n) {
    if (n==values.length-1) return eval(f,v,values);
    n=n+1;
    values[n]=false;  if (satisfiableRecursive(f,v,values,n))return true;
    values[n]=true;   return satisfiableRecursive(f,v,values,n);
}

public static boolean satisfiable(WFF f) {
    ArrayList<String> v = variables(f);
    boolean[] values = new boolean[v.size()];
    return satisfiableRecursive(f,v,values,-1);
}
```

Teil (6): Ein paar Tests mit Zeitmessung:

```
    public static void test1(String s) {
        WFF f = WFFparser.parse(s);
        System.out.println("String: " + s);
        System.out.println("WFF:    " + f);
        truthTable(f);
        long start = System.currentTimeMillis();
        System.out.print(" satisfiable: " + satisfiable(f));
        long end = System.currentTimeMillis();
        System.out.println(" in "+(end-start)+"ms");
        System.out.println();
    }

    public static void test2(String s) {
        WFF f = WFFparser.parse(s);
        System.out.println("String: " + s);
        System.out.println("WFF:    " + f);
        long start = System.currentTimeMillis();
        System.out.print("satisfiable: " + satisfiable(f));
        long end = System.currentTimeMillis();
        System.out.println(" in "+(end-start)+"ms");
        System.out.println();
    }

    public static void main(String[] args) {
      test1 ( "¬a1" );
      test1 ( "a1∨a2" );
      test1 ( "a1∧a2" );
      test1 ( "¬a1∨a2" );
      test1 ( "¬a1∧a1" );
      test1 ( "(a1∨a2)∧¬(¬a3∨¬a4)" );
      test1 ( "(a1∨a2)∧¬(a1∨a2)" );
      String s ="a1";
      for (int i=2;i<=4; i++){ s = s + "∧a"+i; test1 (s); }
      for (int i=5;i<=50;i++){ s = s + "∧a"+i; test2 (s); }
   }
}
```

Einige der Testresultate mit Wertetabelle:

```
String: ¬a1
WFF:    ¬a1
|    a1 ||   f   |
| false ||  true |
|  true || false |
 satisfiable: true in 4ms

String: a1∨a2
WFF:    (a1∨a2)
|    a1 |    a2 ||   f   |
| false | false || false |
| false |  true ||  true |
|  true | false ||  true |
|  true |  true ||  true |
 satisfiable: true in 0ms

String: a1∧a2
WFF:    (a1∧a2)
|    a1 |    a2 ||   f   |
| false | false || false |
| false |  true || false |
|  true | false || false |
|  true |  true ||  true |
 satisfiable: true in 0ms
```

Testresultate ohne Tabelle (gemessene/geschätzte Zeiten):

- Formeln der Form `a1∧a2∧a3∧...∧a_n` für verschiedene `n`
- Formel sind erfüllbar, aber erst in letzter Tabellenzeile ...

n	Dauer [ms]
5	**0**
10	**1**
15	**10**
20	**71**
25	**2616**
29	**46700**
30	**96915**
35	$\sim$ **50**min
40	$\sim$ **1**day
45	$\sim$ **1**month
50	$\sim$ **3**years
55	$\sim$ **100**years

Weitere Folgerungen und Anmerkungen zum DNF-Satz:

- Jede Boolesche Funktion ist mit $\neg$,+ und $\star$ repräsentierbar, d.h. das System $\{\neg, +, \star\}$ ist 'funktional vollständig'.
- Das System $\{\neg, +\}$ ist ebenfalls funktional vollständig, da

$$x_1 x_2 \equiv \overline{\overline{x_1} + \overline{x_2}}$$

- Auch $\{\neg, \star\}$ ist funktional vollständig,wegen

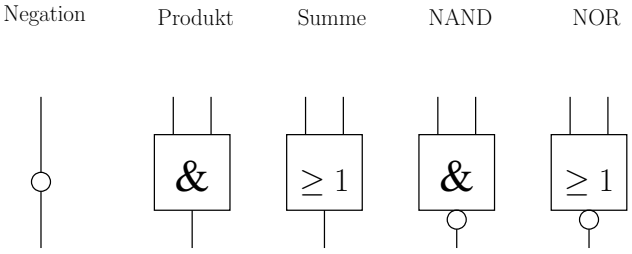
$$x_1 + x_2 \equiv \overline{\overline{x_1}\,\overline{x_2}}$$

- Es reichen sogar bereits die einzelnen(!) Funktionen
  - $NAND(x_1, x_2)$ (mit Bedeutung $\overline{x_1 x_2}$) oder
  - $NOR(x_1, x_2)$ (mit Bedeutung $\overline{x_1 + x_2}$).
- Das System $\{+, \star\}$ ist hingegen nicht funktional vollständig,die Negation kann nicht darüber ausgedrückt werden.

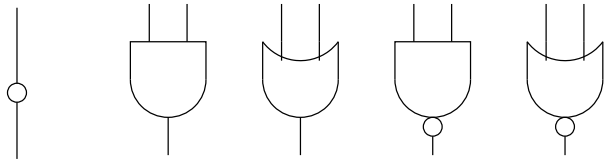
3.3 Kombinatorische Schaltkreise

Technische Realisierung von Booleschen Funktionen mit Logikgattern:

Wichtigste Gatter:

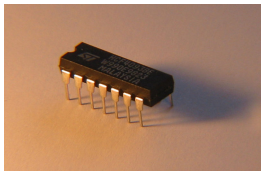


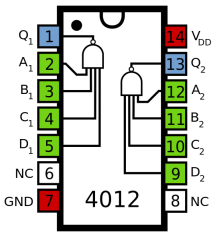
Schreibweise nach IEC 60617-12

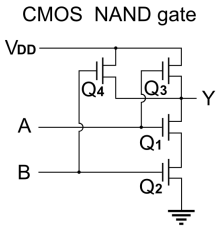


Schreibweise nach ANSI 91-1984

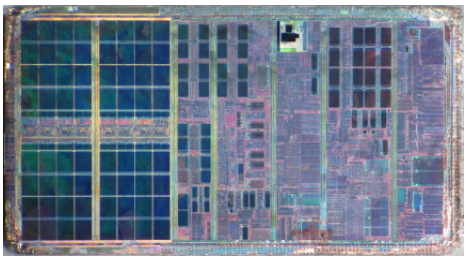
NAND-Gatter im DIL-Gehäuse, mögliche Realisierung







AMD Athlon XP microprocessor mit Barton-Kern (Jahr 2003, Fläche ca 100mm^2, ca. $54 \cdot 10^6$ Transistoren)



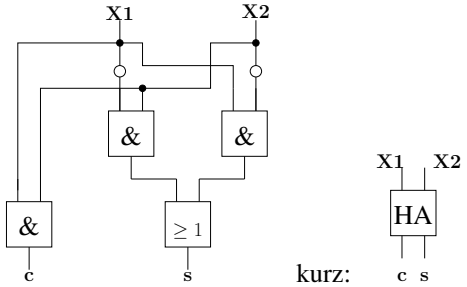
(Pauli Rautakorpi, CC BY 3.0)

Beispiel 1: ‘Halbaddierer’ für binäre Addition:

- zwei Eingänge x_1 und x_2 und
- zwei Ausgänge s (Summe) und c (Übertrag, Carry)

d.h. $c \equiv x_1 \cdot x_2$ und $s \equiv x_1\overline{x_2} + \overline{x_1}x_2$:

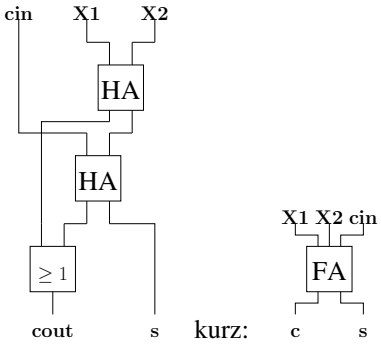
x_1	x_2	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Beispiel 2: ‘Volladdierer’ für binäre Addition mit Übertrag:

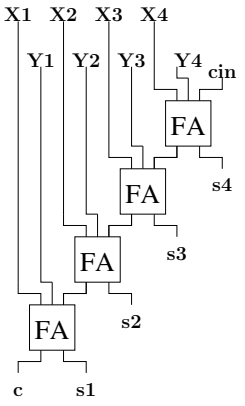
- drei Eingänge x_1, x_2, c_{in} und
- zwei Ausgänge s (Summe) und c_{out} (Übertrag, Carry)

x_1	x_2	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Beispiel 3: ‘4-Bit-Addierer’ für binäre Addition mit Übertrag:

- 9 Eingänge $x_1x_2x_3x_4$, $y_1y_2y_3y_4$, c_{in} und
- fünf Ausgänge $s_1s_2s_3s_4$ (Summe) und c_{out} (Übertrag, Carry)



$$z.B.: \quad \begin{array}{rr} x_1x_2x_3x_4 = & 1\,1\,0\,1 \\ y_1y_2y_3y_4 = & 1\,0\,1\,1 \\ & {\scriptstyle 1\;1\;1\;1}\;0 = c_{in} \\ \hline c_{out}s_1s_2s_3s_4 = & 1\,1\,0\,0\,0 \end{array}$$

4 Aussagenlogik

4.1 Semantik der Aussagenlogik

Definition 4.1.1 (Semantik der Aussagenlogik).

- *Die Werte* $\mathbf{0}$ *und* $\mathbf{1}$ *heißen Wahrheitswerte.*
- *Im folgenden sei* $D \subseteq V$ *eine Menge atomarer Formeln.*
- $[D]$ *sei die Menge aller der Formeln, die nur mit atomaren Formeln aus* D *aufgebaut sind ("Formeln über* D*").*
- *Eine Belegung ist eine Abbildung* $\alpha\colon D \to \{0,1\}$ *(die also jeder atomaren Formel einen Wahrheitswert zuordnet).*

Anmerkung:

- D ist oft endlich, z.B: $D = \{A_1, A_2, A_3\}$,
- aber auch eine unendliche Menge D ist möglich, z.B. alle atomaren Formeln mit ungeradem Index:

$$D = \{A_1, A_3, A_5, A_7, A_9, \ldots\}$$

- α *wird zu einer Abbildung* $\tilde{\alpha}\colon [D] \to \{0,1\}$ *erweitert; dabei wird* $\tilde{\alpha}(F)$ *für* $F \in [D]$ *wie folgt definiert:*
  - *Ist* $F \in D$*, d.h.* F *ist sogar atomare Formel, so sei* $\tilde{\alpha}(F) := \alpha(F)$.
  - *Ist* $F = \neg G$ *und* $\tilde{\alpha}(G)$ *bereits bekannt, so sei*

$$\tilde{\alpha}(F) := \begin{cases} 1 \text{ falls } \tilde{\alpha}(G) = 0 \\ 0 \text{ sonst.} \end{cases}$$

  - *Ist* $F = (G \wedge H)$ *und sind* $\tilde{\alpha}(G)$, $\tilde{\alpha}(H)$ *bereits bekannt, so sei*

$$\tilde{\alpha}(F) := \begin{cases} 1 \text{ falls } \tilde{\alpha}(G) = 1 \text{ und } \tilde{\alpha}(H) = 1 \\ 0 \text{ sonst.} \end{cases}$$

  - *Ist* $F = (G \vee H)$ *und sind* $\tilde{\alpha}(G)$, $\tilde{\alpha}(H)$ *bereits bekannt, so sei*

$$\tilde{\alpha}(F) := \begin{cases} 0 \text{ falls } \tilde{\alpha}(G) = 0 \text{ und } \tilde{\alpha}(H) = 0 \\ 1 \text{ sonst.} \end{cases}$$

- Da $\tilde{\alpha}$ eine Erweiterung von α auf $[D]$ ist, wird i.d.R. $\tilde{\alpha}$ vereinfacht als α geschrieben.
- Die Definition enthält offensichtlich die übliche Festlegung von Negation, Konjunktion und Disjunktion.
- Eine Belegung α legt zu einer Formel F den Wahrheitswert fest, d.h. aus einer Formel wird durch Belegung eine 'Aussage'.

Lemma 4.1.2. $\alpha(F)$ *ist für jedes* $F \in [D]$ *wohldefiniert, d.h. die Definition legt auf eindeutige Weise* $\alpha(F)$ *fest.*

Zum Beweis des Lemma muss gezeigt werden:

(a) Für jede Formel F aus $[D]$ ist $\alpha(F)$ definiert.

(b) Es wird jeder Formel F aus $[D]$ nur ein Wert zugeordnet.

Beweis dazu per Induktion über die Zahl n der Schritte beim Aufbau der Formel.

Nach 2.6.5 gehen wir bei gegebenem n wie folgt vor:

- Induktionsvoraussetzung ist, dass $\alpha(F)$ für jede Formel F wohldefiniert ist, die in weniger als n Schritten erzeugt werden kann.
- Zu zeigen ist, dass dann $\alpha(F)$ auch für jede Formel F wohldefiniert ist, die in genau n Schritten erzeugt werden kann.

Sei daher F eine Formel, die in genau n Schritten aufgebaut werden kann, d.h. es gibt Formeln $F_1, F_2, \ldots, F_n$ mit $F = F_n$; dabei ist jedes F_i atomar oder aus vorherigen F_j, F_k aufgebaut.

Folgende (sich gegenseitig ausschließende!) Fälle sind möglich:

1. F_n ist atomar. Dann ist $F = F_n$ aus D und damit $\alpha(F)$ eindeutig definiert.
2. F entsteht durch Negation, d.h. $F = \neg F_j$ mit $j < n$. Dabei ist F_j eindeutig festgelegt! Zudem ist $F_j \in [D]$ und in weniger als n Schritten aufbaubar. Nach Ind.vor. ist also $\alpha(F_j)$ eindeutig definiert. Damit ist auch $\alpha(F)$ eindeutig definiert.
3. F entsteht durch Konjunktion: , d.h. $F = (F_i \wedge F_j)$ mit $i, j < n$. Dabei sind F_i und F_j eindeutig festgelegt (Klammern!). Zudem sind beide in $[D]$ und in weniger als n Schritten aufbaubar. Nach Ind.vor. sind also $\alpha(F_i)$ und $\alpha(F_j)$ eindeutig definiert. Damit ist auch $\alpha(F)$ eindeutig definiert.
4. F entsteht durch Disjunktion: analog...

Definition 4.1.3. *Sei F eine Formel, $D \subseteq V$ eine Menge atomarer Formeln und $\alpha\colon D \to \{0,1\}$ eine Belegung.*

- *Ist $F \in [D]$, so heißt α zu F passend.*
- *Passt α zu F und ist $\alpha(F) = 1$, so sagt man "F gilt unter α" und schreibt $\alpha \models F$. α heißt Modell für F.*
- *Ist $\mathcal{F}$ eine Menge von Formeln und $\alpha \models F$ für jedes $F \in \mathcal{F}$, so heißt α Modell von $\mathcal{F}$, mit der Schreibweise $\alpha \models \mathcal{F}$.*
- *Eine Menge $\mathcal{F}$ von Formeln heißt erfüllbar, wenn $\mathcal{F}$ mindestens ein Modell hat. Ansonsten heißt $\mathcal{F}$ unerfüllbar.*
- *Eine Formel F heißt allgemeingültig oder Tautologie, wenn jede zu F passende Belegung ein Modell ist.*

Anmerkung: Eine Formel F ist genau dann eine Tautologie, wenn $\neg F$ unerfüllbar ist.

Definition 4.1.4. *Zwei Formeln F und G heißen (semantisch) äquivalent, wenn für alle Belegungen α, die zu F und G passen, stets $\alpha(F) = \alpha(G)$ gilt, mit Schreibweise $F \equiv G$.*

Satz 4.1.5 (Ersetzungssatz)**.**

- *Es sei G eine Formel mit einer Teilformel F.*
- *F' sei eine Formel mit $F \equiv F'$.*
- *G' sei eine Formel, die entsteht, wenn man beim induktiven Aufbau von G an einer Stelle F' statt F verwendet.*

Dann gilt $G \equiv G'$.

Formale Beschreibung der Ersetzung von F durch F':

- G sei erzeugt mit $F_1, F_2, \ldots F_n$ mit $F_n = G$
- Ist F Teilformel von G , dann $F = F_i$ für ein $i \leq n$.

- Beim Aufbau von G werde F zur Konstruktion eines F_j verwendet, d.h. $j > i$ und F_j ist Negation von F oder Disjunktion/Konjunktion von F mit einer weiteren Formel F_ν mit Index $\nu < j$.
- F' sei erzeugt durch $F'_1, F'_2, ..., F'_m$ mit $F'_m = F'$.
- Dann kann G' erzeugt werden durch

  $$F'_1, F'_2, ..., F'_m, F''_1, F''_2, ..., F''_{j-1}, F''_j, F''_{j+1}, ..., F''_n$$

  mit neuen Formeln $F''_1, ..., F''_n$, wenn wir
  - für $k < j$ setzen: $F''_k := F_k$,
  - für $k = j$ setzen:

    $$F''_j := \begin{cases} \neg F' & \text{, falls } F_j = \neg F \\ F' \wedge F_\nu & \text{, falls } F_j = F \wedge F_\nu \\ F' \vee F_\nu & \text{, falls } F_j = F \vee F_\nu \\ F_\nu \wedge F' & \text{, falls } F_j = F_\nu \wedge F \\ F_\nu \vee F' & \text{, falls } F_j = F_\nu \vee F \end{cases}$$

  - und für $k > j$ setzen:

    $$F''_k := \begin{cases} \neg F''_\nu & \text{, falls } F_k = \neg F_\nu \\ F''_\mu \wedge F''_\nu & \text{, falls } F_k = F_\mu \wedge F_\nu \\ F''_\mu \vee F''_\nu & \text{, falls } F_k = F_\mu \vee F_\nu \end{cases}$$

Beweis des Ersetzungssatzes durch Induktion über Zahl n der Schritte beim Aufbau von G.

Sei daher n im folgenden beliebig gewählt, G sei eine Formel, die in genau n Schritten erzeugt werden kann.

- Induktionsvoraussetzung: Für jede Formel, die in weniger als n Schritten erzeugt werden kann, gilt der Ersetzungssatz. Zu zeigen ist, dass dann der Ersetzungssatz auch für G gilt.
- Ist G atomar, dann ist $F = G$ und $F' = G'$, also $G \equiv G'$.
- Ist $F = G$, so liefert die Ersetzung $F' = G'$, also $G \equiv G'$. Im folgenden sei also $F \neq G$.
- Ist G von der Form $G = \neg H$, so ist H in $< n$ Schritten erzeugbar und F ist Teilformel von H. Die Ersetzung von F durch F' liefert dann $G' = \neg H'$ mit $H' \equiv H$, damit auch $G' \equiv G$.
- Ist G Disjunktion oder Konjunktion: analog...

Damit gilt der Ersetzungssatz auch für Formeln G, die in n Schritten erzeugbar sind..

Per Induktion gilt er also für alle Formeln.

4.2 Der Resolutionskalkül der Aussagenlogik

- Ein Kalkül ist eine Sammlung von Regeln zur Umformung von Formeln.
- Normalerweile sind diese Regeln algorithmisch in einer Programmiersprache implementierbar.
- Ziel der Umformungen ist z.B.
  - zu einer Formel F ein Modell α zu finden, oder
  - die Unerfüllbarkeit von F nachzuweisen.

  Dabei ist der Nachweis der Unerfüllbarkeit ein wenig einfacher...

- Ein Kalkül sollte korrekt sein: Wenn der Kalkül die Unerfüllbarkeit von F meldet, dann muss F unerfüllbar sein.
- Ferner sollte der Kalkül vollständig sein: Wenn F unerfüllbar ist, wird dieses Resultat auch vom Kalkül gefunden.

Im Folgenden:

Kalkül für den Nachweis der Unerfüllbarkeit einer Formel F.

Damit auch möglich:

- Überprüfung, ob F eine Tautologie ist: Teste $\neg F$ auf Unerfüllbarkeit
- Überprüfung, ob eine Formel G aus einer Menge $\{F_1, ..., F_n\}$ von Formeln folgt: Teste $F_1 \wedge ... \wedge F_n \wedge \neg G$ auf Unerfüllbarkeit.

Weitere wichtige Schreibweisen der Aussagenlogik:

- $(\bigwedge_{i=1}^{n} F_i)$ an Stelle von $(...((F_1 \wedge F_2) \wedge F_3) \wedge ...) \wedge F_n)$
- $(\bigvee_{i=1}^{n} F_i)$ an Stelle von $(...((F_1 \vee F_2) \vee F_3) \vee ...) \vee F_n)$
- Ein Literal ist eine atomare Formel (positives Literal) oder die Negation einer atomaren Formel (negatives Literal).
- Eine Formel F ist in konjunktiver Normalform (KNF), wenn

$$F = (\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} L_{ij}))$$

und in disjunktiver Normalform (DNF), wenn

$$F = (\bigvee_{i=1}^{n} (\bigwedge_{j=1}^{m_i} L_{ij}))$$

für gewisse Literale L_{ij} ist, wobei $n \geq 1$ und auch jedes $m_i \geq 1$.

Beispiel für die Schreibweise $F = (\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} L_{ij}))$:

- Betrachte die Formel

$$(A \vee C) \wedge (B \vee C \vee \overline{A}) \wedge B$$

- Hier ist $n = 3$ und

$m_1 = 2$	$L_{11} = A, L_{12} = C$
$m_2 = 3$	$L_{21} = B, L_{22} = C, L_{23} = \neg A$
$m_3 = 1$	$L_{31} = B$

Im folgenden nutzen wir, dass wir F bereits in KNF umformen können.

Definition 4.2.1.
- *Eine (endliche) Menge $K := \{L_1, ..., L_j\}$ von Literalen nennt man Klausel.*
- *F sei eine Formel in KNF, d.h.*

$$F = (\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} L_{ij}))$$

für ein $n \geq 1$, gewisse $m_i \geq 1$ und gewisse Literale L_{ij}.

- *Jede Disjunktion* $(\bigvee_{j=1}^{m_i} L_{ij})$ *in* F *definiert eine (nichtleere!) Klausel*

$$K_i := \{L_{i1}, ..., L_{im_i}\}$$

  (für $1 \leq i \leq n$*).*

- *Die (nichtleere!) Menge* $\mathcal{K}(F) := \{K_1, ..., K_n\}$ *heißt die* ***F zugeordnete Klauselmenge****.*

Anmerkung: Eine Belegung α ist Modell einer KNF genau dann, wenn α in jeder Klausel von F mindestens ein Literal erfüllt.

Beispiele:

- $F_1 = (A_1 \vee A_2) \wedge (A_2 \vee \neg A_3)$:
  - Klauseln $K_1 = \{A_1, A_2\}$ und $K_2 = \{A_2, \neg A_3\}$
  - $\mathcal{K}(F_1) = \{\ \{A_1, A_2\}\ ,\ \{A_2, \neg A_3\}\ \}$
- $F_2 = (\neg A_3 \vee A_2) \wedge (A_2 \vee A_1 \vee A_2)$:
  - Klauseln $K_1 = \{\neg A_3, A_2\}$ und $K_2 = \{A_2, A_1\}$
  - $\mathcal{K}(F_2) = \{\ \{\neg A_3, A_2\}\ ,\ \{A_2, A_1\}\ \}\ (= \mathcal{K}(F_1))$
- Verschiedene Formeln können also die gleiche Klauselmenge $\mathcal{K}$ haben!
- Andererseits:
  - Ist eine endliche Klauselmenge $\mathcal{K} = \{K_1, ..., K_n\}$ nicht leer (also $n \geq 1$),
  - und jede enthaltene Klausel K_i ebenfalls nicht leer, d.h. $K_i = \{L_{i1}, ..., L_{im_i}\}$ mit $m_i \geq 1$,
  - so gibt es eine Formel F, die $\mathcal{K}$ als Klauselmenge hat, und zwar z.B.:

$$F = (\bigwedge_{i=1}^{n}(\bigvee_{j=1}^{m_i} L_{ij}))$$

Beispiel:

- Betrachte $\mathcal{K} = \{\ \{A_1, A_2, \neg A_3\}\ ,\ \{A_1, \neg A_2, A_3\}\ ,\ \{\neg A_1, A_2, A_3\}\ \}$
- 3 Klauseln, jeweils mit 3 Literalen
- also passende Formel z.B.:

$$(A_1 \vee A_2 \vee \neg A_3) \wedge (A_1 \vee \neg A_2 \vee A_3) \wedge (\neg A_1 \vee A_2 \vee A_3)$$

  aber auch:

$$(A_2 \vee A_1 \vee \neg A_3) \wedge (A_1 \vee \neg A_2 \vee A_3) \wedge (\neg A_1 \vee A_2 \vee A_3 \vee A_2) \wedge (A_3 \vee \neg A_2 \vee A_1)$$

Erzeugung der Klauselmenge als Java-Programm:

```
1  import java.util.ArrayList;
2  import java.util.HashSet;
3  import java.util.Arrays;
4
5  public class CS {
6
7      public HashSet<HashSet<Integer>> clauses = null;
8      public String[] var = new String[0];
9
10     public static String clauseToString(HashSet<Integer> c){
11         String s=("{");
12         for (Integer i : c) s+=("␣"+i);
13         s += "␣}";
```

```
        return s;
    }

    @Override public String toString(){
        String s="variables:␣";
        for (int i=0; i < var.length; i++){
                if (i>0) s+=("␣");
                s+=(i+1)+":"+var[i];
            }
        s += "\nclause␣set:␣{␣";
        for (HashSet<Integer> clause: clauses){
            s += clauseToString(clause);
        }
        return s+"␣}";
    }
```

Erzeugung der Klauselmenge aus einer Formel, mit kleinem Test:

```
   public CS (WFF f)     {
        f = WFFtoCNF.toCNF(f);
        ArrayList<String> v = WFFtt.variables(f);
        var = v.toArray(var);
        clauses = new HashSet<>();
        addClauses(f,clauses);
    }

    public static void test(String s) {
        System.out.println("String:␣" + s);
        WFF f = WFFparser.parse(s);
        System.out.println("WFF:␣␣␣␣" + f);
        WFF g = WFFtoCNF.toCNF(f);
        System.out.println("CNF:␣␣␣␣" + g);
        CS cs= new CS(f);
        System.out.println("formula␣as␣clause␣set:\n" + cs);
        System.out.println();
    }


    public static void main(String[] args) {
        test ( "¬a1" );
        test ( "a1∨a2" );
        test ( "a1∧a2" );
        test ( "a1∧a1∧a1∧a1" );
        test ( "¬a1∨a2" );
        test ( "(a1∨a2)∧¬(¬a3∨¬a4)" );
        test ( "(a1∨a2∨a2∨a2∨a2∨a2)∧¬(a1∨a2)" );
    }
}
```

Kern des Aufbaus der Klauselmenge:

```

    public void addClauses(WFF f, HashSet<HashSet<Integer>> cl) {
         if ( f instanceof WFF.And ){
            addClauses(((WFF.And) f).f1,cl);
            addClauses(((WFF.And) f).f2,cl);
         } else {
            HashSet<Integer> l = new HashSet<>();
            addLiterals(f,l);
            if (!cl.contains(l)) cl.add(l);
         }

    public int getVariable(WFF.Var f) {
        String name = f.name;
        for (int i=0; i< var.length; i++) if (var[i].equals(name)) return i+1;
        return 0;
    }

    public void addLiterals(WFF f, HashSet<Integer> l) {
        if ( f instanceof WFF.Var ){
            l.add(getVariable(((WFF.Var) f)));
        }
        if ( f instanceof WFF.Not ){
            WFF g = ((WFF.Not) f).f;
            if ( !(g instanceof WFF.Var) )
                throw new RuntimeException("argument␣is␣not␣CNF");
            l.add( - getVariable(((WFF.Var) g)));
        }
        if ( f instanceof WFF.Or  ) {
            addLiterals(((WFF.Or ) f).f1,l);
            addLiterals(((WFF.Or ) f).f2,l);
        }
        if ( f instanceof WFF.And )
              throw new RuntimeException("argument␣is␣not␣CNF");
    }
```

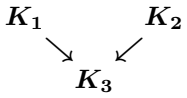
Definition 4.2.2. *Seien* K_1, K_2 *und* K_3 *Klauseln.* K_3 *heißt Resolvente von* K_1 *und* K_2*, wenn es eine atomare Formel* A *gibt, die eine der folgenden Bedingungen erfüllt:*

- $A \in K_1$, $\neg A \in K_2$ *und* $K_3 = (K_1 \setminus \{ A \}) \cup (K_2 \setminus \{\neg A\})$

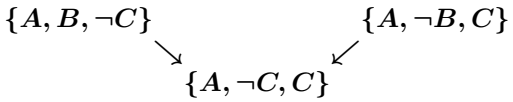
- $\neg A \in K_1,\ A \in K_2$ *und* $K_3 = (K_1 \setminus \{\neg A\}) \cup (K_2 \setminus \{A\})$

Es ist dabei $K_3 = \emptyset$ *möglich (leere Klausel), z.B. wenn* $K_1 = \{A\}$ *und* $K_2 = \{\neg A\}$*).*
Eine Menge von Klauseln, die $\emptyset$ *enthält, wird als unerfüllbar bezeichnet.*

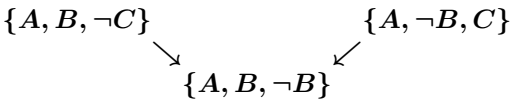
Graphische Darstellung:



Beispiel: Klauselmenge $\{\{A, B, \neg C\}, \{A, \neg B, C\}\}$ mit möglichen Resolventen:



und



Lemma 4.2.3 (Resolutionslemma)**.** *Sei* F *eine KNF mit Klauselmenge* $\mathcal{K}(F)$*.*
Ist K_3 *Resolvente zweier Klauseln* K_1 *und* K_2 *aus* $\mathcal{K}(F)$*, so ist* F *äquivalent zu jeder KNF* $\tilde{F}$ *mit Klauselmenge* $\mathcal{K}(F) \cup \{K_3\}$*.*

Beweis:

- Sei α passend zu F. (Dann passt α auch zu $\tilde{F}$.)
- Gilt $\alpha \models \tilde{F}$, dann auch $\alpha \models F$.
- Sei also nun $\alpha \models F$, wir wollen zeigen: $\alpha \models \tilde{F}$.
- Bis auf die Klausel K_3 werden alle Klauseln von $\tilde{F}$ offensichtlich bereits erfüllt; insbesondere auch K_1 und K_2.
- Sei A die atomare Formel, die die Basis der Resolution ist, o.B.d.A. sei $A \in K_1$ und $\neg A \in K_2$. Dann ist $K_3 = (K_1 \setminus \{A\}) \cup (K_2 \setminus \{\neg A\})$.
- Ist $\alpha(A) = 0$, so muss K_1 ein Literal $L \neq A$ enhalten mit $\alpha(L) = 1$: Wegen $L \in K_3$ ist dann K_3 also auch erfüllt.
- Ist $\alpha(A) = 1$, so muss K_2 ein Literal $L \neq \neg A$ enhalten mit $\alpha(L) = 1$: Wegen $L \in K_3$ ist dann K_3 also auch erfüllt.

Definition 4.2.4.
- *Für jede Klauselmenge* $\mathcal{K}$ *setzen wir*

$$Res(\mathcal{K}) = \mathcal{K} \cup \{R \mid R \text{ Resolvente zweier Klauseln aus } \mathcal{K}\}$$

- *Außerdem setzen wir*

$$Res^0(\mathcal{K}) = \mathcal{K}$$

- *Für* $n \geq 1$ *setzen wir*

$$Res^n(\mathcal{K}) = Res(Res^{n-1}(\mathcal{K}))$$

- *Zudem sei*

$$\boldsymbol{Res^{\infty}(\mathcal{K}) = \bigcup_{n \geq 0} Res^{n}(\mathcal{K})}$$

$\boldsymbol{Res^{n}(\mathcal{K})}$ *heißt Menge der Resolventen* $\boldsymbol{n}$*-ter Stufe.*

- Damit ist z.B. $\boldsymbol{Res^{1}(\mathcal{K}) = Res(Res^{0}(\mathcal{K})) = Res(\mathcal{K})}$
- Stets gilt $\boldsymbol{Res^{n-1}(\mathcal{K}) \subseteq Res^{n}(\mathcal{K})}$
- $\boldsymbol{Res^{\infty}(\mathcal{K})}$ enthält also alle Klauseln, die sich überhaupt durch Resolution aus $\mathcal{K}$ ergeben können.

Satz 4.2.5 (Resolutionssatz der Aussagenlogik)**.** *Eine KNF* $\boldsymbol{F}$ *ist genau dann unerfüllbar, wenn* $\boldsymbol{\emptyset \in Res^{\infty}(\mathcal{K}(F))}$ *gilt.*

Beweis: Sei zunächst $\boldsymbol{\emptyset \in Res^{\infty}(\mathcal{K}(F))}$.

- Dann ist $\boldsymbol{\emptyset \in Res^{n}(\mathcal{K}(F))}$ für eine natürliche Zahl $\boldsymbol{n}$.
- Wähle $\boldsymbol{n}$ minimal mit dieser Eigenschaft, d.h. $\boldsymbol{n = 0}$ oder aber $\boldsymbol{\emptyset \notin Res^{n-1}(\mathcal{K}(F))}$.
- $\boldsymbol{n = 0}$ ist hier nicht möglich, da $\boldsymbol{Res^{0}(\mathcal{K}(F)) = \mathcal{K}(F)}$ nur nicht-leere Klauseln enthält.
- Also entsteht $\boldsymbol{\emptyset}$ als Resolvente zweier $\boldsymbol{K_1, K_2}$ aus $\boldsymbol{Res^{n-1}(\mathcal{K}(F))}$.
- O.B.d.A. also $\boldsymbol{K_1 = \{A\}}$ und $\boldsymbol{K_2 = \{\neg A\}}$ für ein Literal $\boldsymbol{A}$.
- Damit ist $\boldsymbol{F}$ äquivalent zu $\boldsymbol{(A \wedge \neg A)}$, d.h. $\boldsymbol{F}$ ist unerfüllbar.

Für die andere Beweisrichtung: Induktion über die Anzahl $\boldsymbol{n}$ der verwendeten verschiedenen atomaren Formeln.

- Induktionsbehauptung:

  Benutzt $\boldsymbol{F}$ maximal $\boldsymbol{n}$ verschiedene atomare Formeln und ist $\boldsymbol{F}$ unerfüllbar, so ist $\boldsymbol{\emptyset \in Res^{\infty}(\mathcal{K}(F))}$.

- Induktionsanfang: Bei $\boldsymbol{n = 1}$ benutzt $\boldsymbol{F}$ also nur eine atomare Formel $\boldsymbol{A}$. Damit sind nur die drei Klauseln $\boldsymbol{\{A\}}$, $\boldsymbol{\{\neg A\}}$ und $\boldsymbol{\{A, \neg A\}}$ möglich, weshalb es nur 7 Fälle für $\boldsymbol{\mathcal{K}(F)}$ gibt (die leere Menge als Klausel bzw. Klauselmenge fällt jeweils weg):
  - Fall 1, $\boldsymbol{\mathcal{K}(F) = \{\{A\}\}}$:
    Dann ist $\boldsymbol{F}$ erfüllbar, d.h.: Nichts zu zeigen!
  - Fall 2, $\boldsymbol{\mathcal{K}(F) = \{\{\neg A\}\}}$:
    Dann ist $\boldsymbol{F}$ erfüllbar, d.h.: Nichts zu zeigen!
  - Fall 3, $\boldsymbol{\mathcal{K}(F) = \{\{A\}, \{\neg A\}\}}$:
    Dann $\boldsymbol{F}$ ist unerfüllbar und zudem $\boldsymbol{\emptyset \in Res^{\infty}(\mathcal{K}(F))}$.
  - Die vier anderen Fälle für $\boldsymbol{\mathcal{K}(F)}$ lassen sich analog behandeln: $\boldsymbol{\{\{A, \neg A\}\}}$, $\boldsymbol{\{\{A\}, \{A, \neg A\}\}}$, $\boldsymbol{\{\{\neg A\}, \{A, \neg A\}\}}$ (alle erfüllbar) $\boldsymbol{\{\{A\}, \{\neg A\}, \{A, \neg A\}\}}$ (unerfüllbar)

- Induktionsvoraussetzung: Die Behauptung gelte bei maximal $\boldsymbol{n}$ atomaren Formeln.
- $\boldsymbol{F}$ sei eine unerfüllbare Formel mit exakt $\boldsymbol{n + 1}$ atomaren Formeln (ansonsten: nichts zu zeigen).
- Sei $\boldsymbol{A}$ eine beliebige der atomaren Formeln von $\boldsymbol{F}$.

- Da $\boldsymbol{F}$ unerfüllbar ist, können nicht alle Klauseln $\boldsymbol{A}$ enthalten
  (sonst $\boldsymbol{F}$ erfüllbar mit $\boldsymbol{A} \mapsto \mathbf{1}$).
- Da $\boldsymbol{F}$ unerfüllbar ist, können nicht alle Klauseln $\neg\boldsymbol{A}$ enthalten
  (sonst $\boldsymbol{F}$ erfüllbar mit $\boldsymbol{A} \mapsto \mathbf{0}$).
- Bestimme aus $\mathcal{K}(\boldsymbol{F})$ zwei Klauselmengen $\mathcal{K}_\mathbf{1}$ und $\mathcal{K}_\mathbf{2}$ wie folgt:
- $\mathcal{K}_\mathbf{1}$ entsteht wie folgt:
  - Entferne alle Klauseln aus $\mathcal{K}(\boldsymbol{F})$, die $\boldsymbol{A}$ enthalten ($\rightsquigarrow \mathcal{K}'_\mathbf{1}$)
  - Aus jeder Klausel in $\mathcal{K}'_\mathbf{1}$ entferne $\neg\boldsymbol{A}$ (falls vorhanden) ($\rightsquigarrow \mathcal{K}_\mathbf{1}$)
- $\mathcal{K}_\mathbf{2}$ entsteht wie folgt:
  - Entferne alle Klauseln aus $\mathcal{K}(\boldsymbol{F})$, die $\neg\boldsymbol{A}$ enthalten ($\rightsquigarrow \mathcal{K}'_\mathbf{2}$)
  - Aus jeder Klausel in $\mathcal{K}'_\mathbf{2}$ entferne $\boldsymbol{A}$ (falls vorhanden) ($\rightsquigarrow \mathcal{K}_\mathbf{2}$)
- Sowohl $\mathcal{K}_\mathbf{1}$ als auch $\mathcal{K}_\mathbf{2}$ sind nichtleer (enthalten aber evtl. $\emptyset$)
- Dann gilt $\emptyset \in \boldsymbol{Res}^\infty(\mathcal{K}_\mathbf{1})$:
  - Ansonsten gäbe es (per Induktion!) eine KNF $\boldsymbol{F}_\mathbf{1}$ zu $\mathcal{K}_\mathbf{1}$ mit Modell $\boldsymbol{\alpha}_\mathbf{1}$ (ohne Verwendung von $\boldsymbol{A}$).
  - $\boldsymbol{\alpha}_\mathbf{1}$ würde dann mit der Erweiterung $\boldsymbol{A} \mapsto \mathbf{1}$ ein Modell für $\boldsymbol{F}$.
- Führt man die Resolutionsschritte, die $\emptyset$ bei $\mathcal{K}_\mathbf{1}$ erzeugen, analog bei $\mathcal{K}(\boldsymbol{F})$ durch, so entsteht dort dabei entweder ebenfalls $\emptyset$ oder aber stattdessen die Klausel $\{\neg\boldsymbol{A}\}$. Also:

$$\emptyset \in \boldsymbol{Res}^\infty(\mathcal{K}(\boldsymbol{F})) \quad \text{oder} \quad \{\neg\boldsymbol{A}\} \in \boldsymbol{Res}^\infty(\mathcal{K}(\boldsymbol{F}))$$

- Analog gilt $\emptyset \in \boldsymbol{Res}^\infty(\mathcal{K}_\mathbf{2})$. Damit also analog

$$\emptyset \in \boldsymbol{Res}^\infty(\mathcal{K}(\boldsymbol{F})) \quad \text{oder} \quad \{\boldsymbol{A}\} \in \boldsymbol{Res}^\infty(\mathcal{K}(\boldsymbol{F}))$$

- Insgesamt also $\emptyset \in \boldsymbol{Res}^\infty(\mathcal{K}(\boldsymbol{F}))$, und zwar direkt oder aber indirekt durch Resolution aus $\{\boldsymbol{A}\}$ und $\{\neg\boldsymbol{A}\}$.

Aus dem Resolutionssatz ergibt sich folgender Algorithmus:

Algorithmus 4.2.6 (Resolutionskalkül)**.** *1. Zu einer Formel $\boldsymbol{F}$ in KNF bestimme $\mathcal{K} := \mathcal{K}(\boldsymbol{F})$*

2. *Bestimme iterativ solange Werte $\mathcal{K}_n := \boldsymbol{Res}^n(\mathcal{K})$ bis erstmals*
   (a) $\emptyset \in \mathcal{K}_n$
   oder
   (b) $\mathcal{K}_{n-1} = \mathcal{K}_n$
   gilt.
3. *Im Fall (a) melde '$\boldsymbol{F}$ ist unerfüllbar'*
4. *im Fall (b) melde '$\boldsymbol{F}$ ist erfüllbar'.*

Tritt Fall (a) zuerst ein, so ist nach dem Resolutionsatz $\boldsymbol{F}$ unerfüllbar, d.h. die Ausgabe des Algorithmus ist dann korrekt.

Tritt Fall (b) zuerst ein, so ist einerseits $\emptyset \notin \mathcal{K}_n$,aber auch $\mathcal{K}_{n-1} = \mathcal{K}_n = \mathcal{K}_{n+1} = ...$, d.h. $\boldsymbol{Res}^{\infty}(\mathcal{K}) = \mathcal{K}_n$.Nach dem Resolutionssatz ist dann also $\boldsymbol{F}$ erfüllbar, d.h. die Ausgabe des Algorithmus ist ebenfalls korrekt.

Noch offen: Endet der Algorithmus überhaupt in jedem Fall? Dazu:

Es gibt für jede in $\boldsymbol{F}$ benutzte atomare Formel $\boldsymbol{A}$ und jede Klausel $\boldsymbol{K}$ aus $\boldsymbol{Res}^{\infty}(\mathcal{K})$ folgende 4 Möglichkeiten, wie $\boldsymbol{A}$ in $\boldsymbol{K}$ vorkommen kann:

- $\boldsymbol{A}$ kommt in $\boldsymbol{K}$ positiv vor.
- $\boldsymbol{A}$ kommt in $\boldsymbol{K}$ negativ vor.
- $\boldsymbol{A}$ kommt in $\boldsymbol{K}$ positiv und negativ vor.
- $\boldsymbol{A}$ kommt in $\boldsymbol{K}$ gar nicht vor.

Hat $\boldsymbol{F}$ also $\boldsymbol{k}$ atomare Formeln, so enthält $\boldsymbol{Res}^{\infty}(\mathcal{K})$ höchstens 4^k Klauseln. Wegen

$$\mathcal{K}_0 \subseteq \mathcal{K}_1 \subseteq \mathcal{K}_2 ...$$

kann also maximal 4^k-fach eine neue Klausel gefunden werden.

Spätestens nach $n = 4^k$ Schritten muss der Algorithmus anhalten!

(Es gibt Beispiele, wo der Algorithmus exponentiell lange läuft!)

Anmerkungen:

- Exponentielle Laufzeit eines Algorithmus bedeutet, dass er in der Praxis nur für relativ kleine Probleme eingesetzt werden kann.
- Unter `http://www.satcompetition.org/` finden sich Resultate von Wettbewerben um die schnellste Lösung der Frage nach der Erfüllbarkeit einer KNF.
- Es ist ein ungelöstes Problem der Informatik ('P=NP'), ob es generell (un-)möglich ist, die Erfüllbarkeit einer KNF schneller als in exponentieller Zeit festzustellen.

Beispielimplementierung des Resolutionskalüls:

```
1  import java.util.ArrayList;
2  import java.util.HashSet;
3  import java.util.Arrays;
4
5  public class CSresolution {
6      public static final HashSet<Integer> emptyClause= new HashSet<Integer>();
7
8      public static HashSet<HashSet<Integer>>
9                  resolve (HashSet<Integer> k1, HashSet<Integer>k2) {
10         HashSet<HashSet<Integer>> r = new HashSet<HashSet<Integer>>();
11         for (Integer i : k1) for (Integer j : k2){
12             if (i.equals(-j) ) {
13                 HashSet<Integer> k1_i = new HashSet<Integer>(k1);
14                 HashSet<Integer> k2_j = new HashSet<Integer>(k2);
15                 k1_i.remove(i);
16                 k2_j.remove(j);
17                 k1_i.addAll(k2_j);
18                 r.add(k1_i);
19             }
20         }
21         return r;
22     }
23
24    public static void res (HashSet<HashSet<Integer>> kappa) {
25         ArrayList<HashSet<Integer>> cl = new  ArrayList<HashSet<Integer>> (kappa);
26         for (int i=1;i< cl.size(); i++)for (int j=0;j<i; j++){
27             kappa.addAll(resolve(cl.get(i), cl.get(j)));
28         }
29    }
```

```
   public static void resInfty (HashSet<HashSet<Integer>> kappa) {
        int count;
        do {
            count= kappa.size();
            res(kappa);
        } while (count != kappa.size() );
   }

    public static void test(String s) {
        System.out.println("String: " + s);
        WFF f = WFFparser.parse(s);
        CS cs= new CS(f);
        System.out.println(cs);
        resInfty(cs.clauses);
        System.out.println(cs);
        System.out.println("satisfiable: " +
                  !cs.clauses.contains(emptyClause));
        System.out.println();
    }


    public static void main(String[] args) {
        test ( "(¬a1∨a2)∧(a1∨a2)∧(a1∨¬a2)" );
        test ( "(¬a1∨a2)∧(a1∨a2)∧(a1∨¬a2)" );
        test ( "(¬a1∨a2)∧a1∧¬a2" );
        String s ="a1";
        for (int i=2;i<=6;i++) s=s+"∧a"+i;
        test (s);
        s = s + "∧¬("+s+")";
        test (s);
    }
}
```

Als größeres Beispiel zur Umgang mit Aussagenlogik und Resolution: betrachte das Sudoku-Spiel:

- Das Spielfeld hat hier $n \times n$ Blöcke aus je $n \times n$ Zellen, d.h. n^2 Zeilen, n^2 Spalten und n^4 Zellen.
- Bezeichne die Zelle in der i-ten Zeile und j-ten Spalte mit F_{ij}.
- Einträge in die Zellen sind Werte von 1 bis n^2.

Beispiele für Spielfelder:

$n = 3$, Standard

	7		8		2			3
	6	9			3		5	
		4		7			6	
7			1					
		2				7		
					2			5
	4				5		3	
		3		6			8	5
5		8		2			4	

$n = 2$

1	
	2
3	
	4

Zellennamen:

F_{11} F_{12}	F_{13} F_{14}
F_{21} F_{22}	F_{23} F_{24}
F_{31} F_{32}	F_{33} F_{34}
F_{41} F_{42}	F_{43} F_{44}

Modellierung des Spiels als aussagenlogische Formel:

Setze $m = n^2$ und verwende m^3 Boolesche Variablen $x_{i,j,k}$ mit Bedeutung

$$x_{i,j,k} \equiv \text{‘in Zelle } F_{ij} \text{ steht Zahl } k\text{’}$$

wobei $i, j, k \in \{1, \ldots, m\}$.

Dann müssen folgende Bedingungen in der Formel (bzw. Klauselmenge) codiert werden:

- zu Beginn sind gewisse Zellen bereits gesetzt,
- in jeder der m Zeilen kommt jede Ziffer k vor,
- in jeder der m Spalten kommt jede Ziffer k vor,

- in jedem der m Teilblöcke kommt jede Ziffer k vor, und
- in jeder der m^2 Zellen steht nur je eine Zahl

Also erhalten wir als Klauseln:

- für den Start einelementige 'Tatsachen'-Klauseln $\{x_{i,j,k}\}$ für alle schon belegten Zellen,
- für die Vollständigkeit der Zeilen m^2 Klauseln mit m Elementen:

  $$\{x_{i,1,k}, x_{i,2,k}, \ldots, x_{i,m,k}\}$$

  für alle m Zeilen i und m mögliche Einträge k,
- für die Vollständigkeit der Spalten m^2 Klauseln mit m Elementen:

  $$\{x_{1,j,k}, x_{2,j,k}, \ldots, x_{m,j,k}\}$$

  für alle m Spalten j und m mögliche Einträge k,
- für die Vollständigkeit der Blöcke m^2 Klauseln mit m Elementen:

  $$\{x_{3i+1,3j+1,k}, \ldots, x_{3i+n,3j+n,k}\}$$

  für alle m Blöcke mit $0 \leq i \leq n-1$,$j \leq n-1$ und $1 \leq k \leq m$,
- und 2-elementige Ausschlussklauseln gegen Mehrfachbelegung:

  $$\{\overline{x_{i,j,k}}, \overline{x_{i,j,k'}}\}$$

  für alle i, j, k, k' mit $1 \leq k < k' \leq m$.

Der letzte Fall ergibt $m \cdot m \cdot \frac{m \cdot (m-1)}{2}$ Klauseln mit je zwei Literalen.
Insgesamt erhalten wir also

- $m^3 \frac{m-1}{2} + 3m^2$ Klauseln
- $m^3 \cdot (m-1) + 3m^3$ Literale
- zzgl. der $\leq m^2$ Klauseln/Literale der Anfangsbelegung.

Dies ergibt z.B. (ohne Startbelegung):

- bei $n = 3$ und damit $m = 9$:
  $m^3 = 729$ Variablen, 3159 Klauseln, 8019 Literale
- bei $n = 2$ und damit $m = 4$:
  $m^3 = 64$ Variablen, 144 Klauseln, 384 Literale

Beispiel bei $n = 2$:

1	2
3	4

Die Anfangsbelegung liefert folgende Zusatzklauseln:

$$\{x_{121}\}, \{x_{232}\}, \{x_{323}\}, \{x_{434}\}$$

(abhängig vom konkreten Spielfeld).

Alle anderen Klauseln sind allgemein verwendbar!

Zunächst die Klauseln für ‘Zeilen’:

$\{x_{111}, x_{121}, x_{131}, x_{141}\}, \{x_{112}, x_{122}, x_{132}, x_{142}\}, \{x_{113}, x_{123}, x_{133}, x_{143}\}, \{x_{114}, x_{124}, x_{134}, x_{144}\},$
$\{x_{211}, x_{221}, x_{231}, x_{241}\}, \{x_{212}, x_{222}, x_{232}, x_{242}\}, \{x_{213}, x_{223}, x_{233}, x_{243}\}, \{x_{214}, x_{224}, x_{234}, x_{244}\},$
$\{x_{311}, x_{321}, x_{331}, x_{341}\}, \{x_{312}, x_{322}, x_{332}, x_{342}\}, \{x_{313}, x_{323}, x_{333}, x_{343}\}, \{x_{314}, x_{324}, x_{334}, x_{344}\},$
$\{x_{411}, x_{421}, x_{431}, x_{441}\}, \{x_{412}, x_{422}, x_{432}, x_{442}\}, \{x_{413}, x_{423}, x_{433}, x_{443}\}, \{x_{414}, x_{424}, x_{434}, x_{444}\}$

für ‘Spalten’:

$\{x_{111}, x_{211}, x_{311}, x_{411}\}, \{x_{112}, x_{212}, x_{312}, x_{412}\}, \{x_{113}, x_{213}, x_{313}, x_{413}\}, \{x_{114}, x_{214}, x_{314}, x_{414}\},$
$\{x_{121}, x_{221}, x_{321}, x_{421}\}, \{x_{122}, x_{222}, x_{322}, x_{422}\}, \{x_{123}, x_{223}, x_{323}, x_{423}\}, \{x_{124}, x_{224}, x_{324}, x_{424}\},$
$\{x_{131}, x_{231}, x_{331}, x_{431}\}, \{x_{132}, x_{232}, x_{332}, x_{432}\}, \{x_{133}, x_{233}, x_{333}, x_{433}\}, \{x_{134}, x_{234}, x_{334}, x_{434}\},$
$\{x_{141}, x_{241}, x_{341}, x_{441}\}, \{x_{142}, x_{242}, x_{342}, x_{442}\}, \{x_{143}, x_{243}, x_{343}, x_{443}\}, \{x_{144}, x_{244}, x_{344}, x_{444}\}$

für ‘Teilblöcke’:

$\{x_{111}, x_{211}, x_{121}, x_{221}\}, \{x_{311}, x_{411}, x_{321}, x_{421}\}, \{x_{131}, x_{231}, x_{141}, x_{241}\}, \{x_{331}, x_{431}, x_{341}, x_{441}\},$
$\{x_{112}, x_{212}, x_{122}, x_{222}\}, \{x_{312}, x_{412}, x_{322}, x_{422}\}, \{x_{132}, x_{232}, x_{142}, x_{242}\}, \{x_{332}, x_{432}, x_{342}, x_{442}\},$
$\{x_{113}, x_{213}, x_{123}, x_{223}\}, \{x_{313}, x_{413}, x_{323}, x_{423}\}, \{x_{133}, x_{233}, x_{143}, x_{243}\}, \{x_{333}, x_{433}, x_{343}, x_{443}\},$
$\{x_{114}, x_{214}, x_{124}, x_{224}\}, \{x_{314}, x_{414}, x_{324}, x_{424}\}, \{x_{134}, x_{234}, x_{144}, x_{244}\}, \{x_{334}, x_{434}, x_{344}, x_{444}\}$

und als Ausschlussklauseln:

$\{\overline{x_{111}}, \overline{x_{112}}\}, \{\overline{x_{111}}, \overline{x_{113}}\}, \{\overline{x_{111}}, \overline{x_{114}}\}, \{\overline{x_{112}}, \overline{x_{113}}\}, \{\overline{x_{112}}, \overline{x_{114}}\}, \{\overline{x_{113}}, \overline{x_{114}}\},$
$\{\overline{x_{121}}, \overline{x_{122}}\}, \{\overline{x_{121}}, \overline{x_{123}}\}, \{\overline{x_{121}}, \overline{x_{124}}\}, \{\overline{x_{122}}, \overline{x_{123}}\}, \{\overline{x_{122}}, \overline{x_{124}}\}, \{\overline{x_{123}}, \overline{x_{124}}\},$
$\{\overline{x_{131}}, \overline{x_{132}}\}, \{\overline{x_{131}}, \overline{x_{133}}\}, \{\overline{x_{131}}, \overline{x_{134}}\}, \{\overline{x_{132}}, \overline{x_{133}}\}, \{\overline{x_{132}}, \overline{x_{134}}\}, \{\overline{x_{133}}, \overline{x_{134}}\},$
$\{\overline{x_{141}}, \overline{x_{142}}\}, \{\overline{x_{141}}, \overline{x_{143}}\}, \{\overline{x_{141}}, \overline{x_{144}}\}, \{\overline{x_{142}}, \overline{x_{143}}\}, \{\overline{x_{142}}, \overline{x_{144}}\}, \{\overline{x_{143}}, \overline{x_{144}}\},$
$\{\overline{x_{211}}, \overline{x_{212}}\}, \{\overline{x_{211}}, \overline{x_{213}}\}, \{\overline{x_{211}}, \overline{x_{214}}\}, \{\overline{x_{212}}, \overline{x_{213}}\}, \{\overline{x_{212}}, \overline{x_{214}}\}, \{\overline{x_{213}}, \overline{x_{214}}\},$
$\{\overline{x_{221}}, \overline{x_{222}}\}, \{\overline{x_{221}}, \overline{x_{223}}\}, \{\overline{x_{221}}, \overline{x_{224}}\}, \{\overline{x_{222}}, \overline{x_{223}}\}, \{\overline{x_{222}}, \overline{x_{224}}\}, \{\overline{x_{223}}, \overline{x_{224}}\},$
$\{\overline{x_{231}}, \overline{x_{232}}\}, \{\overline{x_{231}}, \overline{x_{233}}\}, \{\overline{x_{231}}, \overline{x_{234}}\}, \{\overline{x_{232}}, \overline{x_{233}}\}, \{\overline{x_{232}}, \overline{x_{234}}\}, \{\overline{x_{233}}, \overline{x_{234}}\},$
$\{\overline{x_{241}}, \overline{x_{242}}\}, \{\overline{x_{241}}, \overline{x_{243}}\}, \{\overline{x_{241}}, \overline{x_{244}}\}, \{\overline{x_{242}}, \overline{x_{243}}\}, \{\overline{x_{242}}, \overline{x_{244}}\}, \{\overline{x_{243}}, \overline{x_{244}}\},$
$\{\overline{x_{311}}, \overline{x_{312}}\}, \{\overline{x_{311}}, \overline{x_{313}}\}, \{\overline{x_{311}}, \overline{x_{314}}\}, \{\overline{x_{312}}, \overline{x_{313}}\}, \{\overline{x_{312}}, \overline{x_{314}}\}, \{\overline{x_{313}}, \overline{x_{314}}\},$
$\{\overline{x_{321}}, \overline{x_{322}}\}, \{\overline{x_{321}}, \overline{x_{323}}\}, \{\overline{x_{321}}, \overline{x_{324}}\}, \{\overline{x_{322}}, \overline{x_{323}}\}, \{\overline{x_{322}}, \overline{x_{324}}\}, \{\overline{x_{323}}, \overline{x_{324}}\},$
$\{\overline{x_{331}}, \overline{x_{332}}\}, \{\overline{x_{331}}, \overline{x_{333}}\}, \{\overline{x_{331}}, \overline{x_{334}}\}, \{\overline{x_{332}}, \overline{x_{333}}\}, \{\overline{x_{332}}, \overline{x_{334}}\}, \{\overline{x_{333}}, \overline{x_{334}}\},$
$\{\overline{x_{341}}, \overline{x_{342}}\}, \{\overline{x_{341}}, \overline{x_{343}}\}, \{\overline{x_{341}}, \overline{x_{344}}\}, \{\overline{x_{342}}, \overline{x_{343}}\}, \{\overline{x_{342}}, \overline{x_{344}}\}, \{\overline{x_{343}}, \overline{x_{344}}\},$
$\{\overline{x_{411}}, \overline{x_{412}}\}, \{\overline{x_{411}}, \overline{x_{413}}\}, \{\overline{x_{411}}, \overline{x_{414}}\}, \{\overline{x_{412}}, \overline{x_{413}}\}, \{\overline{x_{412}}, \overline{x_{414}}\}, \{\overline{x_{413}}, \overline{x_{414}}\},$
$\{\overline{x_{421}}, \overline{x_{422}}\}, \{\overline{x_{421}}, \overline{x_{423}}\}, \{\overline{x_{421}}, \overline{x_{424}}\}, \{\overline{x_{422}}, \overline{x_{423}}\}, \{\overline{x_{422}}, \overline{x_{424}}\}, \{\overline{x_{423}}, \overline{x_{424}}\},$
$\{\overline{x_{431}}, \overline{x_{432}}\}, \{\overline{x_{431}}, \overline{x_{433}}\}, \{\overline{x_{431}}, \overline{x_{434}}\}, \{\overline{x_{432}}, \overline{x_{433}}\}, \{\overline{x_{432}}, \overline{x_{434}}\}, \{\overline{x_{433}}, \overline{x_{434}}\},$
$\{\overline{x_{441}}, \overline{x_{442}}\}, \{\overline{x_{441}}, \overline{x_{443}}\}, \{\overline{x_{441}}, \overline{x_{444}}\}, \{\overline{x_{442}}, \overline{x_{443}}\}, \{\overline{x_{442}}, \overline{x_{444}}\}, \{\overline{x_{443}}, \overline{x_{444}}\}$

1	
	2
3	
	4

Aufgabe

2	**1**	*3*	*4*
3	*4*	**2**	*1*
4	**3**	*1*	*2*
1	*2*	**4**	*3*

Modell

Wir zeigen nun mit Resolution, dass hier eine $\mathbf{1}$ ins Feld $\boldsymbol{F_{24}}$ gehört:

Mit Tatsache $\{x_{121}\}$ und Ausschluss $\{\overline{x_{121}}, \overline{x_{122}}\}$ ergibt sich $\{\overline{x_{122}}\}$.

Analog erhalten wir $\{\overline{x_{123}}\}$ und $\{\overline{x_{124}}\}$.

(Auf Feld F_{12} liegen also weder 2, 3 noch 4.)

Über die Zeilen-Klauseln $\{x_{112}, x_{122}, x_{132}, x_{142}\}$, $\{x_{113}, x_{123}, x_{133}, x_{143}\}$, $\{x_{114}, x_{124}, x_{134}, x_{144}\}$ ergeben sich daraus neue, zusätzliche Klauseln $\{x_{112}, x_{132}, x_{142}\}$, $\{x_{113}, x_{133}, x_{143}\}$, $\{x_{114}, x_{134}, x_{144}\}$

(D.h. 2, 3 und 4 müssen auf die drei freien Felder in Zeile 1. 'Offensichtlich' ist damit dort kein Platz mehr für die 1...)

Aber: Können wir das 'Offensichtliche' auch mit Resolution zeigen?

Die obigen neuen Klauseln $\{x_{112}, x_{132}, x_{142}\}$, $\{x_{113}, x_{133}, x_{143}\}$, $\{x_{114}, x_{134}, x_{144}\}$ und die Ausschlussklauseln $\{\overline{x_{112}}, \overline{x_{113}}\}, \{\overline{x_{112}}, \overline{x_{114}}\}, \{\overline{x_{113}}, \overline{x_{114}}\}$ liefern zunächst die drei Klauseln $\{\overline{x_{113}}, x_{132}, x_{142}\}$, $\{\overline{x_{114}}, x_{133}, x_{143}\}$, $\{\overline{x_{112}}, x_{134}, x_{144}\}$ und durch deren Kombination auch die drei Klauseln $\{x_{133}, x_{143}, x_{132}, x_{142}\}$, $\{x_{134}, x_{144}, x_{133}, x_{143}\}$, $\{x_{132}, x_{142}, x_{134}, x_{144}\}$

(Damit ist zunächst F_{11} aus dem Spiel...)

Die Ausschlussklauseln für F_{13} liefern dann aus $\{x_{133}, x_{143}, x_{132}, x_{142}\}$, $\{x_{134}, x_{144}, x_{133}, x_{143}\}$, $\{x_{132}, x_{142}, x_{134}, x_{144}\}$ die Klauseln $\{\overline{x_{134}}, x_{143}, x_{132}, x_{142}\}$, $\{\overline{x_{132}}, x_{144}, x_{133}, x_{143}\}$, $\{\overline{x_{133}}, x_{142}, x_{134}, x_{144}\}$

Durch Kombination erhalten wir daraus $\{x_{143}, x_{142}, x_{132}, x_{144}\}$ und $\{x_{144}, x_{133}, x_{143}, x_{142}\}$.

Mit einer weiteren Ausschlussklausel für F_{13} erhalten wir $\{x_{143}, x_{142}, \overline{x_{133}}, x_{144}\}$ und schließlich $\{x_{144}, x_{143}, x_{142}\}$.

(Damit ist auch F_{13} aus dem Spiel...)

Mit den Ausschlussklauseln für F_{14} ergibt sich dann aus $\{x_{144}, x_{143}, x_{142}\}$ schrittweise $\{x_{144}, x_{143}, \overline{x_{141}}, \}$, weiter $\{x_{144}, \overline{x_{141}}\}$ und schließlich $\{\overline{x_{141}}\}$.

In einer Lösung kann also auf F_{14} keine 1 stehen.

Analog erhalten wir $\{\overline{x_{131}}\}$.

Aus der Tatsache $\{x_{232}\}$ ergibt sich $\{\overline{x_{231}}\}$.

Eine Lösung hat also ebenfalls keine 1 auf F_{13} oder F_{23}.

Mit den Klauseln $\{\overline{x_{131}}\}$, $\{\overline{x_{141}}\}$, $\{\overline{x_{231}}\}$ und der Blockklausel $\{x_{131}, x_{231}, x_{141}, x_{241}\}$ erhalten wir dann schrittweise erst $\{x_{231}, x_{141}, x_{241}\}$, dann $\{x_{141}, x_{241}\}$ und endlich $\{x_{241}\}$.

Jede Lösung hat also auf F_{24} die Zahl 1.

Insgesamt bilden wir dazu 34 Resolventen, hier noch einmal explizit:

1. $\{x_{121}\}, \{\overline{x_{121}}, \overline{x_{122}}\} \rightsquigarrow \{\overline{x_{122}}\}$
2. $\{x_{121}\}, \{\overline{x_{121}}, \overline{x_{123}}\} \rightsquigarrow \{\overline{x_{123}}\}$
3. $\{x_{121}\}, \{\overline{x_{121}}, \overline{x_{124}}\} \rightsquigarrow \{\overline{x_{124}}\}$
4. $\{x_{112}, x_{122}, x_{132}, x_{142}\}, \{\overline{x_{122}}\} \rightsquigarrow \{x_{112}, x_{132}, x_{142}\}$
5. $\{x_{113}, x_{123}, x_{133}, x_{143}\}, \{\overline{x_{123}}\} \rightsquigarrow \{x_{113}, x_{133}, x_{143}\}$
6. $\{x_{114}, x_{124}, x_{134}, x_{144}\}, \{\overline{x_{124}}\} \rightsquigarrow \{x_{114}, x_{134}, x_{144}\}$
7. $\{x_{112}, x_{132}, x_{142}\}, \{\overline{x_{112}}, \overline{x_{113}}\} \rightsquigarrow \{\overline{x_{113}}, x_{132}, x_{142}\}$
8. $\{x_{113}, x_{133}, x_{143}\}, \{\overline{x_{113}}, \overline{x_{114}}\} \rightsquigarrow \{\overline{x_{114}}, x_{133}, x_{143}\}$
9. $\{x_{114}, x_{134}, x_{144}\}, \{\overline{x_{112}}, \overline{x_{114}}\} \rightsquigarrow \{\overline{x_{112}}, x_{134}, x_{144}\}$
10. $\{\overline{x_{113}}, x_{132}, x_{142}\}, \{x_{113}, x_{133}, x_{143}\} \rightsquigarrow \{x_{133}, x_{143}, x_{132}, x_{142}\}$
11. $\{\overline{x_{114}}, x_{133}, x_{143}\}, \{x_{114}, x_{134}, x_{144}\} \rightsquigarrow \{x_{134}, x_{144}, x_{133}, x_{143}\}$
12. $\{\overline{x_{112}}, x_{134}, x_{144}\}, \{x_{112}, x_{132}, x_{142}\} \rightsquigarrow \{x_{132}, x_{142}, x_{134}, x_{144}\}$
13. $\{x_{133}, x_{143}, x_{132}, x_{142}\}, \{\overline{x_{133}}, \overline{x_{134}}\} \rightsquigarrow \{\overline{x_{134}}, x_{143}, x_{132}, x_{142}\}$

14. $\{x_{134}, x_{144}, x_{133}, x_{143}\}, \{\overline{x_{132}}, \overline{x_{134}}\} \rightsquigarrow \{\overline{x_{132}}, x_{144}, x_{133}, x_{143}\}$
15. $\{\overline{x_{132}}, x_{144}, x_{133}, x_{143}\}, \{x_{133}, x_{143}, x_{132}, x_{142}\} \rightsquigarrow \{x_{144}, x_{133}, x_{143}, x_{142}\}$
16. $\{\overline{x_{134}}, x_{143}, x_{132}, x_{142}\}, \{x_{132}, x_{142}, x_{134}, x_{144}\} \rightsquigarrow \{x_{143}, x_{142}, x_{132}, x_{144}\}$
17. $\{x_{143}, x_{142}, x_{132}, x_{144}\}, \{\overline{x_{132}}, \overline{x_{133}}\} \rightsquigarrow \{x_{143}, x_{142}, \overline{x_{133}}, x_{144}\}$
18. $\{x_{143}, x_{142}, \overline{x_{133}}, x_{144}\}, \{x_{144}, x_{133}, x_{143}, x_{142}\} \rightsquigarrow \{x_{144}, x_{143}, x_{142}\}$
19. $\{x_{144}, x_{143}, x_{142}\}, \{\overline{x_{141}}, \overline{x_{142}}\} \rightsquigarrow \{x_{144}, x_{143}, \overline{x_{141}}\}$
20. $\{x_{144}, x_{143}, \overline{x_{141}}\}, \{\overline{x_{141}}, \overline{x_{143}}\} \rightsquigarrow \{x_{144}, \overline{x_{141}}\}$
21. $\{x_{144}, \overline{x_{141}}\}, \{\overline{x_{141}}, \overline{x_{144}}\} \rightsquigarrow \{\overline{x_{141}}\}$

Analog durch Vertauschung der Rollen der Zellen F_{13} und F_{14} mit ebenfalls 9 Resolventen:

27. ... $\rightsquigarrow \{x_{134}, x_{133}, x_{132}\}$
30. ... $\rightsquigarrow \{\overline{x_{131}}\}$
31. $\{x_{232}\}, \{\overline{x_{231}}, \overline{x_{232}}\} \rightsquigarrow \{\overline{x_{231}}\}$

Schließlich:

32. $\{x_{131}, x_{231}, x_{141}, x_{241}\}, \{\overline{x_{131}}\} \rightsquigarrow \{x_{231}, x_{141}, x_{241}\}$
33. $\{x_{231}, x_{141}, x_{241}\}, \{\overline{x_{231}}\} \rightsquigarrow \{x_{141}, x_{241}\}$
34. $\{x_{141}, x_{241}\}, \{\overline{x_{141}}\} \rightsquigarrow \{x_{241}\}$

Abschließende Anmerkungen zu diesem Beispiel:

- (Aussagen-)Logik kann eine Basis für 'künstliche Intelligenz' sein ($\rightsquigarrow$ *Wirtschaftsinformatik II*)
- Die Zahl der Variablen ist so groß, dass manuelle Betrachtungen oder Wertetabellen unbrauchbar sind.
- Die Gleichförmigkeit der Klauseln legt nahe, Prädikatenlogik zu verwenden (z.B. mit i,j,k als Variablen und $x_{i,j,k}$ als dreistelligem Prädikat $x(i,j,k)$)
- Das schnelle Finden von Lösungen aussagenlogischer Formeln ist eine Kernfrage der Informatik! (Stichwort: `satsolver`)

4.3 Hornlogik

- Die Frage der Erfüllbarkeit einer beliebigen KNF ist nur schwer zu beantworten, da alle bekannten Algorithmen i.d.R. exponentielle Laufzeit haben.
- Durch Einschränkung der erlaubten Teilformeln: Hornlogik, mit leichter Lösbarkeit des Erfüllbarkeitstests. ($\rightsquigarrow$ Alfred Horn, 1918-2001, amerikanischer Mathematiker)

Definition 4.3.1.
- *Eine Formel* $F = (\bigwedge_{i=1}^{n}(\bigvee_{j=1}^{m_i} L_{ij}))$ *in KNF heißt Hornformel, wenn jede der Disjunktionen* $\bigvee_{j=1}^{m_i} L_{ij}$ *maximal ein positives Literal enthält.*
- *Eine Klausel* $\{L_1, \ldots, L_r\}$ *heißt Hornklausel, wenn höchstens eines der Literale* $L_1, \ldots, L_r$ *positiv ist.*

Beispiele:

- Hornformel: $A \wedge (\neg A \vee B) \wedge (\neg B \vee \neg C \vee D) \wedge (\neg C \vee \neg D)$
- keine Hornformel: $(A \vee B) \wedge (B \vee \neg C \vee D)$

- Hornklauseln: $\{A\}, \{A, \neg B\}, \{B, \neg C, \neg D\}, \{\neg C, \neg D\}$
- Eine Disjunktion mit positiven und negativen Literalen kann als Implikation interpretiert werden:

$$(\neg A \vee \neg B \vee C \vee D) \equiv (A \wedge B) \rightarrow (C \vee D)$$

- Eine Hornformel ist also meist eine Konjunktion von einfachen Implikationen:

$$(A \rightarrow B) \wedge ((B \wedge C) \rightarrow D)$$

  Die rechten Seiten bestehen dabei aus einem (positiven) Literal, die linken Seiten sind Konjunktionen positiver Literale.

  Ausnahmen: Disjunktionen ohne positive Literale sowie einzelne positive Literale haben nicht die Form einer Implikation.

- Eine Implikation der Form $A \rightarrow (B \wedge C)$ lässt sich umformen zu einer Konjunktion zweier Implikationen:

$$A \rightarrow (B \wedge C) \equiv (A \rightarrow B) \wedge (A \rightarrow C)$$

  Also: Bei Implikationen von Konjunktionen reicht jeweils ein Literal auf der rechten Seite der Implikationen ($\rightsquigarrow$ KNF!)

Definition 4.3.2.
- *Eine Hornklausel ohne negative Literale (d.h. bestehend nur aus einem einzelnen positiven Literal) heißt Tatsachenklausel.*
- *Eine Hornklausel mit einem positiven und mindestens einem negativen Literal heißt Prozedurklausel oder Regel.*
- *Tatsachenklauseln und Regeln werden auch als Programmklauseln oder definite Klauseln bezeichnet.*
- *Eine Hornklausel ohne positives, aber mit mindestens einem negativen Literal heißt Zielklausel oder Frageklausel.*
- *Eine Menge von Hornklauseln heißt Logikprogramm oder Hornklauselprogramm.*

Beispiel:

- Tatsachenklauseln $\{A\}, \{B\}$
- Prozedurklauseln $\{\neg A, \neg B, C\}, \{\neg C, \neg B, D\}$
- Zielklausel $\neg D$
- Interpretation als Anfrage:

  Folgt D aus $A \wedge B \wedge (A \wedge B \rightarrow C) \wedge (C \wedge B \rightarrow D)$?

- Eine Zielklausel entspricht einer Disjunktion $(\neg L_1 \vee \ldots \vee \neg L_r)$ bzw. der Negation $\neg Z$ einer Konjunktion $Z = (L_1 \wedge \ldots \wedge L_r)$.
- Ein Logikprogramm F ohne Zielklauseln (d.h. nur Programmklauseln) entspricht also einer Sammlung von positiven Literalen (T, Tatsachenklauseln) und von Implikationen von Konjunktionen mit ausschließlich positiven Literalen (P, Prozedurklauseln), z.B.

$$\begin{aligned} F \equiv T \wedge P \equiv \quad & A \\ \wedge \quad & B \\ \wedge \quad & (A \wedge B \rightarrow C) \\ \wedge \quad & (C \wedge B \rightarrow D) \end{aligned}$$

- Wegen $\boldsymbol{F \wedge \neg Z \equiv \neg(\neg F \vee Z) \equiv \neg(F \rightarrow Z)}$ ist also $\boldsymbol{F \wedge \neg Z}$ genau dann **un**erfüllbar, wenn $\boldsymbol{F \rightarrow Z}$ Tautologie ist.
- Das Logikprogramm $\boldsymbol{T \wedge P \wedge \neg Z}$ ist also genau dann **un**erfüllbar, wenn die Anfrage $\boldsymbol{Z}$ eine Folgerung aus den Tatsachen $\boldsymbol{T}$ und den Regeln $\boldsymbol{P}$ ist.

Formulierung in der Programmiersprache `Prolog` ('Programmation en Logique'):

- Tatsachen und Regeln werden in einer Datei gespeichert, etwa `bsp.pro`:

```
1 a.
2 b.
3 c:- a,b.
4 d:- c,b.
```

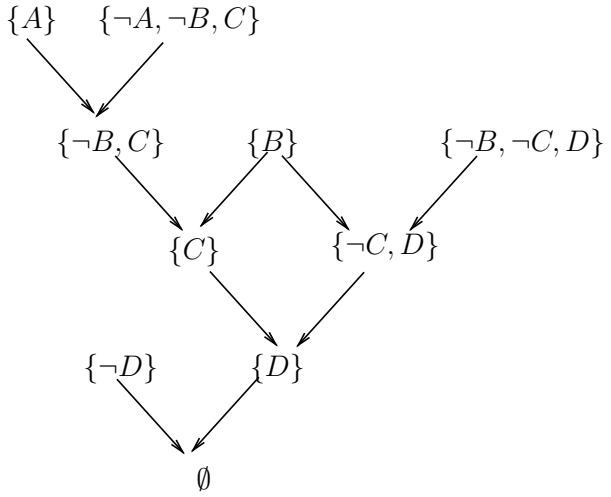
- Das Prolog-Programm wird gestartet: `swipl -s bsp.pro`.
- Dann wird die Anfrage gestellt (in positiver Form):

```
1 ?- c,d.
2 true.
```

(Ende mit `^-D`)

Beispiel zur Resolution bei Hornklauseln:

- Logikprogramm: $\boldsymbol{\{A\}, \{B\}, \{\neg A, \neg B, C\}, \{\neg B, \neg C, D\}, \{\neg D\}}$
- Mögliche Resolutionen:



- Damit folgt das Ziel $\boldsymbol{D}$ aus den Tatsachen $\boldsymbol{\{A\}, \{B\}}$ und den Regeln $\boldsymbol{\{\neg A, \neg B, C\}, \{\neg B, \neg C, D\}}$

Gesucht also: schneller Algorithmus zum Test auf (Un-)Erfüllbarkeit.
Grundidee dazu:

- Versuche eine Belegung $\boldsymbol{\alpha}$ zu konstruieren, die alle Klauseln erfüllt.

- Also: Tatsachenklauseln müssen auf jeden Fall erfüllt werden. ($\rightsquigarrow$ Werte $\alpha(A) = 1$ für Tatsachen $\{A\}$)
- Sind alle Prämissen einer Implikation erfüllt, so muss ihre Konklusion auch erfüllt sein. ($\rightsquigarrow \alpha$ wird um $\alpha(A) = 1$ der Konklusion A erweitert).
- Alle übrigbleibenden atomaren Formeln werden auf 0 gesetzt.
- Damit alle Tatsachen und Regeln sicherlich erfüllt.
- Enthält am Ende die Zielklausel ein Atom, das nicht auf 1 gesetzt ist, so ist sie erfüllt (negatives Literal!)...

Algorithmus 4.3.3 (Markierungsalgorithmus für Logikprogramme)**.**

1. *Gegeben sei ein Logikprogramm mit einer einzelnen Zielklausel.*
2. *Markiere jedes Literal aus Tatsachenklauseln.*
3. *Markiere die Tatsachenklauseln als erfüllt.*
4. *Wiederhole, solange es neu markierte Literale gibt:*
   - *Für alle neu markierten Literale: Markiere jedes Auftreten des Literals in den Klauseln.*
   - *Danach für alle Programmklauseln: Sind alle negativen Literale in der Klausel markiert, so markiere auch das positive Literal (falls es noch nicht markiert ist) und die Klausel selbst.*
5. *Markiere schließlich die Zielklausel, falls alle ihre Literale markiert sind.*
6. *Bleibt die Zielklausel unmarkiert: Ausgabe von 'Ziel ist keine Folgerung'.*
7. *Wurde die Zielklausel markiert: Ausgabe von 'Ziel ist Folgerung'.*

Ablauf am Beispiel $\{A\}, \{B\}, \{\neg A, \neg B, C\}, \{\neg B, \neg C, D\}, \{\neg D\}$:

- Atomare Formeln sind A, B, C, D.
- Markiere A und B.
- Markiere die Klauseln $\{A\}$ und $\{B\}$.
- Alle negativen Literale in $\{\neg A, \neg B, C\}$ markiert: Markiere C und die Klausel.
- Alle negativen Literale in $\{\neg B, \neg C, D\}$ markiert: Markiere D und die Klausel.
- Da D bei Zielklausel $\{\neg D\}$ markiert: *Ziel ist Folgerung*!

Eigenschaften des Markierungsalgorithmus:

- n sei die Anzahl verschiedener atomarer Formeln im Logikprogramm F.
- Die Wiederholung in Zeile (4) des Algorithmus braucht immer mindestens ein neu markiertes Literal, also endet der Algorithmus nach maximal n Durchläufen.
- Definiere eine Belegung α durch
$$\alpha(A) = \begin{cases} 1, & \text{falls } A \text{ markiert,} \\ 0, & \text{sonst.} \end{cases}$$
- Dann gilt einerseits: Ergibt der Algorithmus die Ausgabe 'keine Folgerung', so erfüllt α alle Klauseln, ist also ein Modell von F.

- Andererseits gilt auch: Ist α' ein Modell von F, so gilt für das vom Algorithmus bestimmte α:

$$\alpha(A) = 1 \Rightarrow \alpha'(A) = 1$$

  Da α' Modell von F ist, muss für mindestens ein Atom A der Zielklausel mit $\alpha'(A) = 0$ gelten. Also auch $\alpha(A) = 0$, d.h. der Algorithmus ergibt die Ausgabe 'keine Folgerung'.

- Ziel: Algorithmus, der möglichst noch effizienter ist als der Markierungsalgorithmus.
- Dazu: Genauere Betrachtung der möglichen Resolutionsschritte.

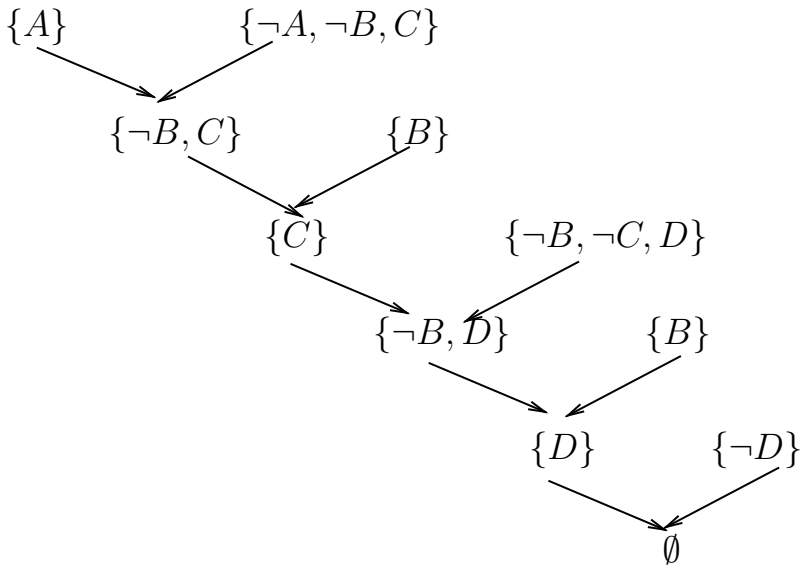
Definition 4.3.4. *F sei eine Formel in KNF. Eine Folge von Klauseln $K_0, K_1, \ldots, K_n$ heißt lineare Resolution in F, wenn gilt:*

- *K_0 ist aus $\mathcal{K}(F)$.*
- *Für $i = 1, \ldots, n$ gibt es Seitenklauseln B_i, so dass K_i durch Resolution aus K_{i-1} und B_i entsteht. Dabei ist stets B_i aus $\mathcal{K}(F) \cup \{K_0, \ldots, K_{i-1}\}$ zu wählen.*

- Bei einer linearen Resolution wird also die Klausel K_0 durch Resolution mit bekannten Klauseln schrittweise in die Klausel K_n umgewandelt.
- Bei der normalen Resolution muss die Resolution für alle Paare von Klauseln versucht werden, bei der linearen Resolution ist hingegen eine der zwei Klauseln immer vorgegeben.

Beispiel zur linearen Resolution:

- Logikprogramm: $\{A\}, \{B\}, \{\neg A, \neg B, C\}, \{\neg B, \neg C, D\}, \{\neg D\}$
- Lineare Resolution:



- Im Beispiel ist die Seitenklausel sogar stets aus der Menge $\mathcal{K}(F)$

Definition 4.3.5. • *F sei eine Formel in KNF.*

- *Eine lineare Resolution $K_0, K_1, \ldots, K_n$ in F heißt Input-Resolution, wenn die Seitenklauseln stets aus $\mathcal{K}(F)$ sind.*

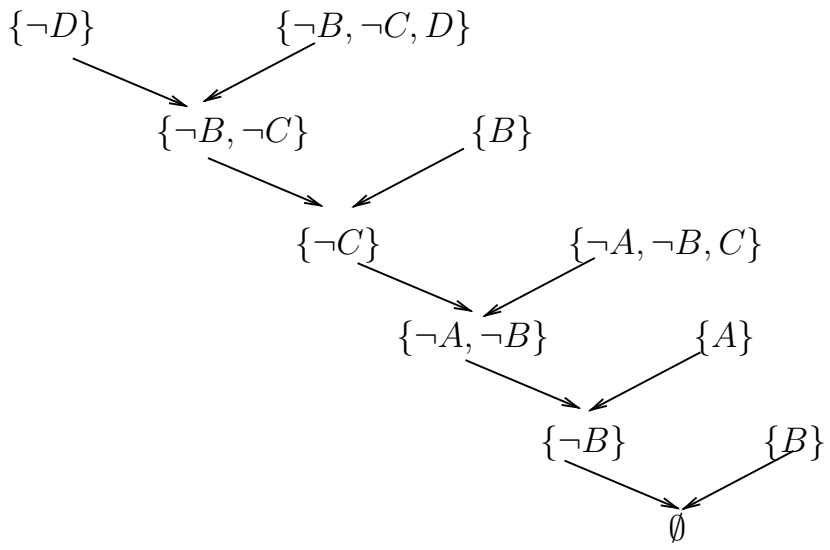
Achtung: Nicht bei jeder unerfüllbaren KNF F kann $\emptyset$ durch Input-Resolution erzeugt werden!

Definition 4.3.6.
- *F sei eine Hornformel.*
- *Eine SLD-Resolution ist eine Input-Resolution $K_0, K_1, \ldots, K_n$, bei der K_0 eine Zielklausel ist (und als Seitenklauseln B_i daher nur Programmklauseln aus $\mathcal{K}(F)$ verwendet werden).*

- SLD steht für '**l**inear resolution with unrestricted **s**election function for **d**efinite clauses'.
- Welches Literal und welche Seitenklausel bei einer SLD verwendet werden, wird durch eine Auswahlfunktion bestimmt (s.u.).

Beispiel zur SLD-Resolution:

- Logikprogramm: $\{A\}, \{B\}, \{\neg A, \neg B, C\}, \{\neg B, \neg C, D\}, \{\neg D\}$
- SLD-Resolution:



- Alle Zwischenklauseln sind (neue) Frageklauseln!

Satz 4.3.7. *(a) Lineare Resolution ist für KNF vollständig, d.h. für jede unerfüllbare KNF F gibt es eine lineare Resolution $K_0, K_1, \ldots, K_n$ in F mit $K_n = \emptyset$.*

(b) SLD-Resolution ist für Hornformeln vollständig, d.h. für jede unerfüllbare Hornformel F gibt es eine SLD-Resolution $K_0, K_1, \ldots, K_n$ in F mit $K_n = \emptyset$.

Beweis zu (a): siehe [Kreuzer: Logik für Informatiker]

Beweis zu (b) über den Markierungsalgorithmus:

- Sei F unerfüllbare Hornformel, d.h. der Markierungsalgorithmus markiert alle Literale einer Zielklausel Z.
- $K''_1, \ldots, K''_{m''}$ seien die vom Algorithmus markierten Klauseln (in Reihenfolge der Markierung und inkl. der Zielklausel).

- Streiche aus der Liste zunächst alle anderen Zielklauseln.
- Streiche aus der Liste dann alle Programmklauseln K_i'', deren positives Literal A bereits in einem K_j'' mit $j < i$ auftaucht (als A oder $\neg A$).

  Die verbleibende Liste sei $K_1', ..., K_{m'}'$ (mit $m' \leq m''$).

  - In der verbleibenden Liste kommt jedes Literal höchstens einmal in positiver Form vor.
  - Auf der Klauselmenge $\{K_1', ..., K_{m'}'\}$ kann der Markierungsalgorithmus immer noch erfolgreich durchgeführt werden, mit der Reihenfolge $K_1', K_2', ..., K_{m'}'$.
  - Damit enthält jedes K_i' nur negative Literale, die schon vorher in $K_1', ..., K_{i-1}'$ positiv vorkommen.

- Wiederhole so oft wie möglich: Streiche aus der Liste weiter alle Programmklauseln K_i', für die es kein $j > i$ gibt, so dass das positive Literal A von K_i' in K_j' vorkommt (als $\neg A$).

  Die verbleibende Liste sei $K_1, ..., K_m$ (mit $m \leq m'$).

  - Jedes in einer Klausel positiv verwendete Literal A taucht also in der Liste später noch einmal auf, dann aber nur negativ als $\neg A$. (*)
  - Die letzte Klausel K_m enthält also kein positives Literal.
  - K_m ist damit die Zielklausel.
  - Auf der Klauselmenge $\{K_1, ..., K_m\}$ kann der Markierungsalgorithmus immer noch erfolgreich durchgeführt werden, mit der Reihenfolge $K_1, K_2, ..., K_m$.
  - Damit enthält jedes K_i nur negative Literale, die schon vorher in $K_1, ..., K_{i-1}$ positiv vorkommen. (**)
  - Kommt ein Literal A in einer Klausel K_i positiv vor, dann kommt $\neg A$ in K_i selbst nicht vor. (***)

Behauptung:

(1) Wir können Zielklauseln $Z_1, \ldots, Z_m$ mit $Z_m = K_m$ konstruieren, so dass $Z_m, Z_{m-1}, \ldots, Z_1$ eine SLD-Resolution ist, und

(2) dabei wird beim Übergang von Z_{i+1} nach Z_i bei $i<m$ stets K_i als Seitenklausel verwendet,

(3) zudem enthält Z_i nur (negative) Literale, die in $K_1, ..., K_i$ positiv vorkommen.

Beweis dieser Behauptung mit Induktion über $i \in \{1, ..., m\}$, dabei:

- Achtung: Wir starten die Induktion mit $i = m$ (statt mit $i = 1$) und führen den Induktionsschritt in der Richtung von i zu $i-1$ durch, solange $i > 1$ gilt.
- Alternative Leseweise: Führe 'normale' Induktion mit einem Parameter k durch und ersetze überall i durch $m - k$.

Induktionsanfang:

- Für $i = m$ setzen wir $Z_i := K_m$, damit ist (1) und (3) erfüllt.

  Für (2) ist hier nichts zu zeigen.

Induktionsschritt:

- Für ein i mit $m \geq i > 1$ sei Z_i unter Erfüllung von (1), (2) und (3) bereits konstruiert. Wir betrachten also jetzt Z_i und K_{i-1} und müssen Z_{i-1} konstruieren.

- Das positive Literal $\boldsymbol{A}$ der Klausel $\boldsymbol{K_{i-1}}$ wird in den Klauseln $\boldsymbol{K_i, ..., K_m}$ verwendet, wegen (*) nur als $\boldsymbol{\neg A}$.
- Also wird es bei keinem der Resolutionschritte $\boldsymbol{K_m, K_{m-1}, ..., K_i}$ entfernt, es ist also in $\boldsymbol{Z_i}$ negativ enthalten.
- Wir definieren $\boldsymbol{Z_{i-1}}$ als die Resolvente aus $\boldsymbol{Z_i}$ und $\boldsymbol{K_{i-1}}$, d.h.

$$\boldsymbol{Z_{i-1} := (Z_i \setminus \{\neg A\}) \cup (K_{i-1} \setminus \{A\})}$$

- (1) und (2) für $\boldsymbol{i-1}$ sind damit erfüllt.
- Außerdem kommt $\boldsymbol{A}$ in $\boldsymbol{Z_{i-1}}$ wegen (***) nicht vor.
  Mit (**) für $\boldsymbol{i}$ ist (3) also bei $\boldsymbol{i-1}$ erfüllt.

Es fehlt nur noch der Nachweis, dass wir zu $\emptyset$ resolvieren können:

- $\boldsymbol{K_1}$ als erste markierte Klausel muss eine Tatsachenklausel aus einem einzelnen positiven Literal $\boldsymbol{A}$ sein.
- Nach (3) ist dann $\boldsymbol{Z_2 = \{\neg A\}}$. Die Resolvente von $\boldsymbol{Z_2}$ und $\boldsymbol{K_1}$ ist damit die leere Menge.
- Insgesamt haben wir damit also eine SLD-Resolution

$$\boldsymbol{K_m = Z_m, ..., Z_1 = \emptyset}$$

  konstruiert, wobei $\boldsymbol{K_m}$ eine Zielklausel der ursprünglichen Hornformel ist.

Also:

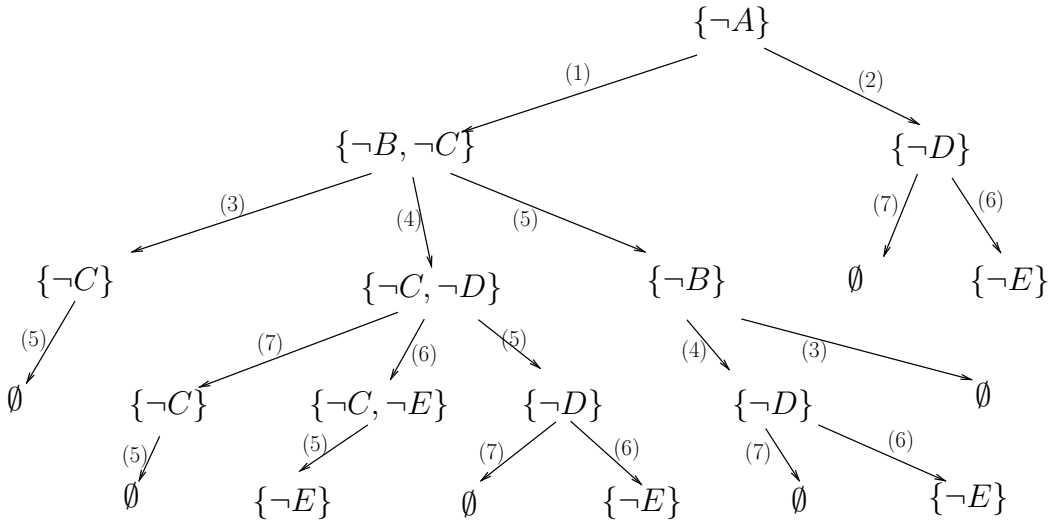
> Für jede unerfüllbare Hornformel $\boldsymbol{F}$ gibt es eine SLD-Resolution in $\boldsymbol{F}$ von einer Zielklausel zu $\emptyset$.

Offen ist dabei noch, welche Seitenklauseln in jedem Schritt angewendet werden sollen, wozu es mehrere Strategien gibt.

Als Beispiel betrachte folgende Klauseln eines `Prolog`-Programmes (d.h. atomare Formeln haben Kleinbuchstaben...):

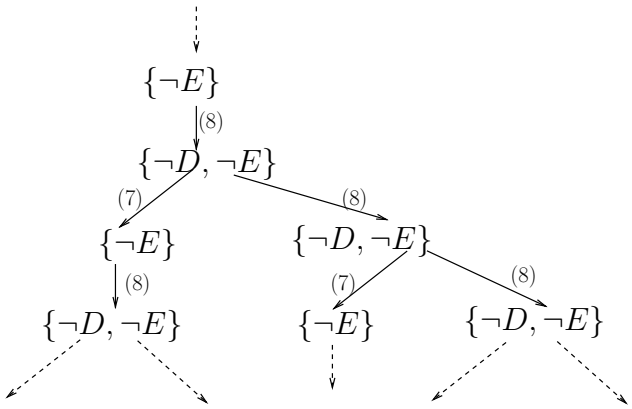
(1) `a:- b,c.` (2) `a:- d.` (3) `b.` (4) `b:- d.`
(5) `c.` (6) `d:- e.` (7) `d.`

Bei der Anfrage '`?- a.`', d.h. der Zielklausel $\neg\boldsymbol{A}$, sind nun folgende Resolutionen möglich (dabei wird bei Resolutionen stets nur die verwendete Seitenklausel angegeben):



Anmerkungen dazu:

- Erfolgreiche SLD-Resolutionen enden mit $\emptyset$.
- Manche der SLD-Resolutionen sind aber erfolglos (im Beispiel: da keine Resolution mit $\boldsymbol{E}$ möglich ist).
- Ergänze die Klauselmenge um '(8) `e:- d,e.`', dann:

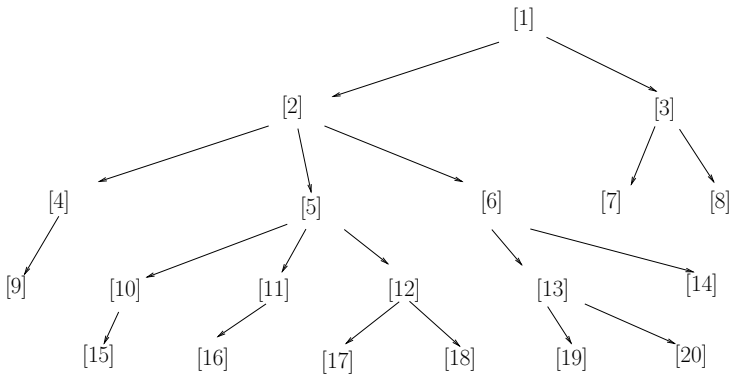


  d.h. nicht jeder Versuch einer SLD-Resolution endet, es können evtl. Klauseln beliebig oft wiederholt auftreten!
- Frage: Warum konnten wir im Satz über die Vollständigkeit der SLD-Resolution trotzdem $\emptyset$ erreichen?

Wichtige Struktur in der Informatik: Bäume(siehe 'Diskrete Strukturen', 'Algorithmen und Datenstrukturen', u.v.m.)

- Ein Baum besteht aus einer Menge von Knoten, die miteinander mit Pfeilen verbunden sein können.
- Von jedem Knoten aus kann es ausgehende Pfeile geben, die zu weiteren Knoten führen.
- Es gibt einen Start-Knoten (die Wurzel), in den kein Pfeil führt.
- Jeder Knoten außer der Wurzel wird von genau einem Pfeil erreicht.
- Ein Knoten mit ausgehenden Pfeilen wird innerer Knoten genannt.
- Ein Knoten ohne ausgehende Pfeile wird Blatt genannt.
- Der Knoten am Beginn eines Pfeiles ist der Vaterknoten, der Knoten an der Pfeilspitze heißt Kindknoten.
- Zudem verwendet man Begriffe wie Geschwisterknoten, Vorfahren, Nachkommen, ...
- Die $\boldsymbol{n}$-te Schicht im Baum sind die Knoten mit gleichem Abstand $\boldsymbol{n}$ zur Wurzel.

Beim obigen Beispiel der SLD-Resolutionen ergibt sich der Baum:



- Ein `Prolog`-Programm definiert also zunächst nur Möglichkeiten für Übergänge zwischen Knoten.
- Aus einer Zielklausel ergibt sich daraus implizit ein Baum mit durch Resolution erreichbaren Klauseln.
- In diesem Baum dürfen Klauseln mehrfach auftreten.
- Gefragt ist nun, ob eines der Blätter die leere Klausel ist.
- Der Baum ist allerdings i.d.R. viel zu groß, um ihn komplett aufzuschreiben...

Strategien zum Durchsuchen eines Baumes:

- BFS (Breadth-first-search, Breitensuche)
  - Beginne bei der Wurzel des Baumes ($\mathbf{0}$-te Schicht).
  - Danach besuche alle Kinder der Wurzel ($\mathbf{1}$-te Schicht).
  - Danach besuche deren Kinder ($\mathbf{2}$-te Schicht), usw.
  - Im Beispiel entsteht also die Suchreihenfolge

$$[\mathbf{1}], [\mathbf{2}], [\mathbf{3}], \ldots, [\mathbf{19}], [\mathbf{20}]$$

- DFS (Depth-first-search, Tiefensuche)
  - Beginne bei der Wurzel des Baumes.
  - Nach dem Besuch eines Knotens besuche zunächst alle seine Nachkommen.
  - Erst danach besuche nacheinander die jüngeren Geschwister (und deren Nachkommen).
  - Im Beispiel entsteht also die Suchreihenfolge

$$[\mathbf{1}], [\mathbf{2}], [\mathbf{4}], [\mathbf{9}], [\mathbf{5}], [\mathbf{10}], [\mathbf{15}], [\mathbf{11}], \ldots, [\mathbf{14}], [\mathbf{3}], [\mathbf{7}], [\mathbf{8}]$$

Bei einem BFS-Algorithmus müssen i.W. zwei Schichten des Baumes gespeichert werden, z.B. wie folgt:

Algorithmus 4.3.8 (BFS-Grundstruktur)**.**

- $\boldsymbol{X} := \mathbf{0}$*-te Schicht des Baumes;*
- ***while*** *(Suche nicht beendet)*
- $\quad \boldsymbol{Y} :=$ *leere Menge;*
- $\quad$ ***forall*** $\boldsymbol{k} \in \boldsymbol{X}$ ***do:***
- $\quad\quad$ *bearbeite Knoten* $\boldsymbol{k}$*;*
- $\quad\quad$ *füge Kinder von* $\boldsymbol{k}$ *zu* $\boldsymbol{Y}$ *hinzu;*
- $\quad \boldsymbol{X} := \boldsymbol{Y}$*;*

Hat z.B. ein Baum $\boldsymbol{n}$ Schichten und jeder innere Knoten maximal $\mathbf{2}$ Kinder, so kann die $\boldsymbol{n}$-te Schicht bis zu $\mathbf{2}^{\boldsymbol{n}}$ Knoten enthalten!

Bei einem DFS-Algorithmus muss 'nur' ein Pfad von der Wurzel zum aktuellen Knoten gespeichert werden.

Implementierung meist als rekursive Methode `DFS_suche(`$\boldsymbol{k}$`)`:

Algorithmus 4.3.9 (DFS-Grundstruktur)**.**

- *DFS_suche(**k**):*
- *bearbeite Knoten **k**;*
- ***forall** (**j** Kind von **k**) **do:***
- *DFS_suche(**j**);*

mit Start `DFS_suche(Wurzel).`

- Hat ein Baum also n Schichten, so ergibt sich eine Rekursionstiefe von höchstens n.
- Der Pfad wird implizit im 'Rekursionsstack' gespeichert.

Da die BFS-Strategie oft zuviel Speicherplatz benötigt, benutzt `Prolog` normalerweise die DFS-Strategie!

Wegen der Möglichkeit unendlicher langer Pfade bei SLD-Resolutionen gilt jedoch:

Satz 4.3.10.
- *Durchsucht man den SLD-Baum in BFS-Strategie, so werden leere Klauseln auf jeden Fall gefunden, wenn sie vorhanden sind.*
- *Durchsucht man den SLD-Baum in DFS-Strategie, so führt die Suche evtl. nicht zu einem Resultat, sondern kann unendlich lange laufen.*

Beispiel:

- Logikprogramm mit nur 3 Klauseln `a:- a.`, `a:- b.`, `b.`

$\rightsquigarrow$ Speicherfehler wegen zu hoher Rekursionstiefe!

- Reihenfolge `a:- b.`, `a:- a.`, `b.` funktioniert jedoch...

5 Prädikatenlogik

5.1 Syntax der Prädikatenlogik

Bei der Prädikatenlogik erweitert man die Aussagenlogik mit dem Ziel, komplexere Zusammenhänge möglichst einfach formulieren zu können, z.B.

- die Eigenschaften der natürlichen Zahlen (Peano-Axiome), etwa

  'Verschiedene natürliche Zahlen n und m besitzen verschiedene Nachfolger.'

- oder die Stetigkeit einer reellen Funktion $f : \mathbb{R} \to \mathbb{R}$ in $x \in \mathbb{R}$

  'Für alle $\varepsilon > 0$ gibt es $\delta > 0$, so dass für alle y gilt: Ist $|x-y| < \delta$, dann auch $|f(x)-f(y)| < \varepsilon$'

Für die Informatik interessanter sind z.B.:

- aussagenlogische Zusammenhänge unter Reduktion der Zahl der Variablen und Klauseln formulieren (vgl. Sudoku-Beispiel):

  'Alle Vorlesungstermine sind überschneidungsfrei.'

- die Korrektheit von Algorithmen (vgl.Beispiel 2.6.4)

  'Für alle Eingaben liefert der Markierungsalgorithmus die korrekte Ausgabe.'

Man braucht also:

- Variablen, wie 'x', 'y', ... über einem Universum U,
- Funktionen, wie 'f', Betrag '$|\ |$', Differenz '$-$' mit evtl. unterschiedlicher Zahl von Parametern ('Stelligkeit'),
- Terme, zur Bildung komplexerer Ausdrücke '$|f(x) - f(y)|$',
- Prädikate, wie '$<$', mit denen aus Werten des Universums Wahrheitswerte gebildet werden können,
- Junktoren, wie '$\vee$', '$\wedge$' zur Verknüpfung von Wahrheitswerten,
- Quantoren, wie 'für alle' oder 'es gibt'.

Im folgenden: Definition der Syntax, dabei hier Strenge der Formulierung gegenüber der Definition der Aussagenlogik etwas gelockert...

Definition 5.1.1 (Syntax der Prädikatenlogik). *(1) Eine Variable ist ein Symbol der Form x_i mit $i \in \mathbb{N}_0$.*

(2) Ein Funktionssymbol hat die Form $f_i^{(k)}$ mit $i, k \in \mathbb{N}_0$.

Bei einem Funktionssymbol $f_i^{(k)}$ gibt der obere Index k an, wieviele Parameter verwendet werden sollen:

- $f_i^{(1)}$: einstellige Funktion (z.B. $\cos$, $\sqrt{\ }$)
- $f_i^{(2)}$: zweistellige Funktion (z.B. Operationen $+$, $-$, $*$, $/$)
- $f_i^{(0)}$: nullstellige Funktion, d.h. ohne Parameter. Solche Funktionen werden auch Konstanten genannt.

- Als Variablennamen werden auch u, v, w, x, y, z u.ä. verwendet.
- Als Funktionssymbole werden auch f, g, h u.ä. verwendet.
- Die Stelligkeit eines Funktionssymboles ergibt sich meist implizit aus seiner Anwendung und wird dann nicht explizit angegeben.

(3) Ein Prädikatsymbol hat die Form $P_i^{(k)}$ *mit* $i, k \in \mathbb{N}_0$.

Auch bei Prädikatsymbolen gibt der obere Index k die Stelligkeit an:

- $P_i^{(1)}$: einstelliges Prädikat (z.B. 'x ist positiv')
- $P_i^{(2)}$: zweistelliges Prädikat (z.B. Vergleiche $\leq$, $\geq$, ...)
- $P_i^{(0)}$: nullstelliges Prädikat, d.h. Konstanten zur Angabe von Wahrheitswerten
- Wieder ergibt sich die Stelligkeit meist implizit.
- Als Prädikatssymbole werden auch P, Q, R u.ä. verwendet.

(4) Ein Term entsteht durch endlich viele Anwendungen aus:

(4a) Jede Variable ist ein Term.

(4b) Ist f *eine* k*-stelliges Funktionssymbol und sind* t_1, ..., t_k *bereits Terme, so ist auch* $f(t_1, ..., t_k)$ *ein Term.*

- Bei zweistelligen Funktionen f, g, h und Variablen x, y ist z.B. $f(g(x, y), h(y, f(y, x)))$ ein Term.
- Bei Konstanten lässt man in der Regel das Klammerpaar um die Parameter weg.
- Als Namen von Konstanten werden auch a, b, c u.ä. verwendet.

(5) Eine prädikatenlogische Formel entsteht durch endlich viele Anwendungen aus:

(5a) Ist P *eine* k*-stelliges Prädikatsymbol und sind* t_1, ..., t_k *bereits Terme, so ist* $P(t_1, ..., t_k)$ *eine Formel.*

(5b) Sind F *und* G *Formeln, so sind auch* $\neg F$, $(F \vee G)$ *und* $(F \wedge G)$ *Formeln.*

(5c) Ist x *eine Variable und* F *eine Formel, so sind auch* $\forall x\ F$ *und* $\exists x\ F$ *Formeln.*

- Wieder als Abkürzung: '$(F \to G)$' für '$(\neg F \vee G)$'
- $\forall$: Allquantor, 'für alle'
- $\exists$: Existenzquantor, 'es existiert'

Als Alphabet für prädikatenlogische Formeln reicht also:

$$\{x, f, P, 0, 1, (,), \neg, \vee, \wedge, \forall, \exists\}$$

Beispiel: Stetigkeit einer reellen Funktion $f : \mathbb{R} \to \mathbb{R}$ in $x \in \mathbb{R}$

'Für alle $\varepsilon > 0$ gibt es $\delta > 0$, so dass für alle y gilt: Ist $|x - y| < \delta$, dann auch $|f(x) - f(y)| < \varepsilon$'

Formalisiert mit Hilfe von:

- Universum $\mathbb{R}$, Variablennamen x, y, δ, ε
- Funktionssymbole: f, g, h, o
- g einstellig; $g(x)$ steht für 'Absolutbetrag von x'
- h zweistellig; $h(x, y)$ steht für '$x - y$'
- o nullstellig; steht für die Konstante 0
- Prädikat P zweistellig; $P(x, y)$ steht für '$x > y$'

Dann können wir als Formel verwenden:

$$\forall\varepsilon\ (P(\varepsilon, o) \rightarrow \exists\delta\ (P(\delta, o) \wedge$$
$$\forall y\ (\ P(\delta, g(h(x, y))) \rightarrow P(\varepsilon, g(h(f(x), f(y))))\ )))$$

5.2 Semantik der Prädikatenlogik

- Bei aussagenlogischen Formeln war eine Belegung α eine Zuweisung von Wahrheitswerten an atomare Formeln.
- In der Prädikatenlogik müssen wir nun allen Komponenten (Variablen, Funktionssymbole, Prädikatsymbole) passende Werte zuweisen!

Definition 5.2.1 (Semantik der Prädikatenlogik)**.** *F sei eine prädikatenlogische Formel.*

(1) Eine zu F passende Struktur α ist ein Tupel $\alpha = (U, \phi, \psi, \xi)$, wobei:

- *(1a) U ist eine nichtleere Menge (Grundmenge oder Universum von α).*
- *(1b) ϕ ordnet jedem in F vorkommenden Funktionssymbol f (mit Stelligkeit k) eine Abbildung $\phi(f) : U^k \rightarrow U$ zu.*
- *(1c) ψ ordnet jedem in F vorkommenden Prädikatsymbol P (mit Stelligkeit k) eine Teilmenge $\psi(P) \subseteq U^k$ zu.*
- *(1d) ξ ordnet jeder in F vorkommenden Variablen x ein Element $\xi(x)$ aus U zu.*

Anmerkung: ξ darf an Variablen zuweisen, die gar nicht auftreten!

Eine passende Struktur zur Stetigkeitsformel (grün=Syntax!)

$$\forall\varepsilon : (P(\varepsilon, o) \rightarrow \exists\delta : (P(\delta, o) \wedge$$
$$\forall y\ (\ P(\delta, g(h(x, y))) \rightarrow P(\varepsilon, g(h(f(x), f(y))))\ )))$$

wäre

$$\alpha_1 = (U_1, \phi_1, \psi_1, \xi_1)$$

mit

- $U_1 = \mathbb{R}$
- ϕ_1: $\phi_1(g) =$ Absolutbetrag, $\phi_1(h) =$ Differenz,
  $\phi_1(f) = \cos$, $\phi_1(o) = 0$

- ψ_1: $\psi_1(P) = \{(z_1, z_2) \in \mathbb{R}^2 \mid z_1 > z_2\}$
- ξ_1: $\xi_1(x) = 1, \xi_1(y) = 2, \xi_1(\varepsilon) = 3, \xi_1(\delta) = 4$

(passend zur Bedeutung: 'cos ist in 1 stetig')

Anmerkung: Die Werte von ξ für die Variablen y, ε und δ werden gar nicht benutzt (da die Variablen nur gebunden verwendet werden).

Ebenfalls passend wäre aber auch

$$\alpha_2 = (U_2, \phi_2, \psi_2, \xi_2)$$

mit

- $U_2 = \mathbb{N}_0$
- ϕ_2: $\phi_2(g) = \phi_2(f) = id$, Identitätsfunktion mit $id(n) = n$,
  $\phi_2(h) =$ Maximum zweier Zahlen, $\phi_2(o) = 1$
- ψ_2: $\psi_2(P) = \{(n, n) \mid n \in \mathbb{N}_0\}$
- ξ_2: $\xi_2(x) = 1, \xi_2(y) = 2, \xi_2(\varepsilon) = 3, \xi_2(\delta) = 4$

Ausgehend von einer Struktur können wir nun eine Formel 'interpretieren':

(2) Der Wert $\alpha(t)$ eines Terms t in einer passenden Struktur α wird rekursiv definiert über:

(2a) Ist t eine Variable, so sei

$$\alpha(t) := \xi(t)$$

(2b) Ist t von der Form $f(t_1, ..., t_k)$ mit einem k-stelligen Funktionssymbol f und k Termen $t_1, ..., t_k$, so sei

$$\alpha(t) := \phi(f)(\alpha(t_1), ..., \alpha(t_k))$$

Beispiel-Terme: $g(h(x, y))$, $g(h(f(x), f(y)))$

- Bei $\alpha_1 = (\mathbb{R}, \phi_1, \psi_1, \xi_1)$, d.h.
  $\phi(g)$ Betrag, $\phi_1(h)$ Differenz, $\phi_1(f) = \cos$, $\xi_1(x) = 1$, $\xi_1(y) = 2$

$$\begin{aligned} \alpha_1(\, g(h(x, y))\,) &= |\alpha_1(\, h(x, y)\,)| = |\alpha_1(x) - \alpha_1(y)| \\ &= |\xi_1(x) - \xi_1(y)| = |1 - 2| = 1 \\ \alpha_1(\, g(h(f(x), f(y)))\,) &= ... = |cos(1) - cos(2)| = 0.9564... \end{aligned}$$

- Bei $\alpha_2 = (\mathbb{N}_0, \phi_2, \psi_2, \xi_2)$, d.h.
  $\phi_2(g) = \phi_2(f) = id$, $\phi_2(h)$ Maximum, $\xi_2(x) = 1$, $\xi_2(y) = 2$

$$\begin{aligned} \alpha_2(\, g(h(x, y))\,) &= id(\alpha_2(\, h(x, y)\,)) = \alpha_2(\, h(x, y)\,) \\ &= max\{\alpha_2(x), \alpha_2(y)\} = 2 \\ \alpha_2(\, g(h(f(x), f(y)))\,) &= id(max\{id(\xi_2(x)), id(\xi_2(y))\}) = 2 \end{aligned}$$

(3) Der Wahrheitswert $\alpha(F)$ einer prädikatenlogischen Formel F in einer passenden Struktur α wird rekursiv definiert über:

(3a) Ist F von der Form $P(t_1, ..., t_k)$ mit einem k-stelligen Prädikatssymbol P und k Termen $t_1, ..., t_k$, so sei

$$\alpha(F) := \begin{cases} 1, \text{ falls } (\alpha(t_1), ..., \alpha(t_k)) \in \psi(P) \\ 0, \text{ sonst.} \end{cases}$$

(3b) Ist F von der Form $\neg G$, $(G \vee H)$ oder $(G \wedge H)$, so wird $\alpha(F)$ wie in der Aussagenlogik definiert:

$$\begin{aligned} \alpha(\neg G) &:= \begin{cases} 1, \text{ falls } \alpha(G) = 0 \\ 0, \text{ sonst.} \end{cases} \\ \alpha(\,(G \wedge H)\,) &:= \begin{cases} 1, \text{ falls } \alpha(G) = \alpha(H) = 1 \\ 0, \text{ sonst.} \end{cases} \\ \alpha(\,(G \vee H)\,) &:= \begin{cases} 0, \text{ falls } \alpha(G) = \alpha(H) = 0 \\ 1, \text{ sonst.} \end{cases} \end{aligned}$$

Beispiel-Prädikate: $P(\varepsilon, o)$, $P(\delta, g(h(f(x), f(y))))$

- Bei $\alpha_1 = (\mathbb{R}, \phi_1, \psi_1, \xi_1)$:
  $\phi_1(o)$ Konstante $0 \in \mathbb{R}$, $\psi_1(P) = \{(z_1, z_2) \in \mathbb{R}^2 \mid z_1 > z_2\}$
  $\xi_1(x) = 1, \xi_1(y) = 2, \xi_1(\varepsilon) = 3, \xi_1(\delta) = 4$

$$\begin{aligned} \alpha_1(\,P(\varepsilon, o)\,) &= \text{“Ist } 3 > 0 \text{ ?”} = 1 \\ \alpha_1(\,P(\delta, g(h(f(x), f(y))))\,) &= \text{“Ist } 4 > 0.9564... \text{ ?”} = 1 \end{aligned}$$

- Bei $\alpha_2 = (\mathbb{N}_0, \phi_2, \psi_2, \xi_2)$:
  $\phi_2(o) = 1, \psi_2(P) = \{(n, n) \mid n \in \mathbb{N}_0\}$
  $\xi_2(x) = 1, \xi_2(y) = 2, \xi_2(\varepsilon) = 3, \xi_2(\delta) = 4$

$$\begin{aligned} \alpha_2(\,P(\varepsilon, o)\,) &= \text{“Ist } 3 = 1 \text{ ?”} = 0 \\ \alpha_2(\,P(\delta, g(h(f(x), f(y))))\,) &= \text{“Ist } 4 = 2 \text{ ?”} = 0 \end{aligned}$$

- Für eine Variablenbelegung ξ, eine Variable x und einen Wert $u \in U$ sei $\xi|_{x \mapsto u}$ die wie folgt modifizierte Belegung:

$$\xi|_{x \mapsto u}(y) = \begin{cases} u & \text{falls } y = x \\ \xi(y) & \text{falls } y \neq x \end{cases}$$

- Für eine Struktur $\alpha = (U, \phi, \psi, \xi)$, eine Variable x und einen Wert $u \in U$ sei $\alpha|_{x \mapsto u}$ die wie folgt modifizierte Struktur

$$\alpha|_{x \mapsto u} := (U, \psi, \phi, \xi|_{x \mapsto u})$$

(3c) Ist F von der Form $\forall x\, G$ oder $\exists x\, G$, so wird $\alpha(F)$ wie folgt definiert:

$$\begin{aligned} \alpha(\,\forall x\, G\,) &:= \begin{cases} 1, \text{ falls für alle } u \in U \text{ gilt } \alpha|_{x \mapsto u}(G) = 1 \\ 0, \text{ sonst.} \end{cases} \\ \alpha(\,\exists x\, G\,) &:= \begin{cases} 1, \text{ falls es ein } u \in U \text{ gibt mit } \alpha|_{x \mapsto u}(G) = 1 \\ 0, \text{ sonst.} \end{cases} \end{aligned}$$

Beispiel-Strukturen α_1 und α_2 bei der Formel F:

$$\forall \varepsilon\ (P(\varepsilon, o) \rightarrow \exists \delta\ (P(\delta, o) \wedge$$
$$\forall y\ (\ P(\delta, g(h(x, y))) \rightarrow P(\varepsilon, g(h(f(x), f(y))))\ )))$$

- Auswertung bei $\alpha_1 = (\mathbb{R}, \phi_1, \psi_1, \xi_1)$:

  "Für alle ε gilt: Ist $\varepsilon > 0$, dann gibt es ein δ mit: Es ist $\delta > 0$ und für alle $y \in \mathbb{R}$ folgt aus $\delta > |1 - y|$ auch $\varepsilon > |cos(1) - cos(y)|$"

  Dies entspicht der Stetigkeit von cos in 1, also $\alpha_1(F) = 1$

- Auswertung bei $\alpha_2 = (\mathbb{N}_0, \phi_2, \psi_2, \xi_2)$:

  "Für alle ε gilt: Ist $\varepsilon = 1$, dann gibt es ein δ mit: Es ist $\delta = 1$ und für alle $y \in \mathbb{N}_0$ folgt aus $\delta = max\{1, y\}$ auch $\varepsilon = max\{1, y\}$"

  Hier gilt also ebenfalls $\alpha_2(F) = 1$.

- Hier betrachtet: Prädikatenlogik erster Stufe d.h. Quantifizierung nur über Elemente von U.
- Prädikatenlogik zweiter Stufe: Quantifizierung auch über Funktionen oder Prädikate erlaubt.
- Oft auch möglich: komplexeres Universum U, z.B. $U = \mathbb{R} \cup C(\mathbb{R})$ (aus $\mathbb{R}$ und allen stetigen Funktionen $C(\mathbb{R})$.
  - Unterscheidung zwischen $\mathbb{R}$ und $C(\mathbb{R})$ z.B. über Prädikat isR
  - Dann "für alle $x \in \mathbb{R}$ gilt $P(x)$" formalisiert über
    $$\forall x\ (isR(x) \rightarrow P(x))$$
  - Auswertung einer Funktion f, d.h. Bestimmung des Wertes von $f(x) = y$, ebenfalls über Prädikat, z.B. $isvalof(f, x, y)$
  - Dann "für alle $x \in \mathbb{R}$ gilt $P(f(x))$" formalisiert über
    $$\forall x\ (isR(x) \rightarrow \exists y\ (isvalof(f, x, y) \wedge P(y)))$$

Definition 5.2.2 (Prädikatenlogische Modelle)**.** *F sei eine prädikatenlogische Formel und α eine dazu passende Struktur.*

(1) Eine zu F passende Struktur α heißt Modell für F, wenn $\alpha(F) = 1$ gilt. (Schreibweise: $\alpha \models F$)

(3) F heißt erfüllbar, wenn es ein Modell für F gibt.

(3) F heißt unerfüllbar, wenn es kein Modell für F gibt.

(2) F heißt allgemeingültig oder eine Tautologie, wenn jede passende Struktur α bereits ein Modell für F ist.

Zwei prädikatenlogische Formeln F heißen äquivalent, wenn für alle Strukturen α, die sowohl zu F als auch zu G passen, gilt $\alpha(F) = \alpha(G)$. (Schreibweise: $F \equiv G$)

- Sowohl α_1 als auch α_2 sind Modelle für die "Stetigkeits-Formel".

- Daher: Meist komplexere Formeln verwendet, die dann weniger Modelle zulassen... (i.W. mit Konjunktionen verknüpft)

Die Aussagenlogik ist Teil der Prädikatenlogik:

- Universum $\boldsymbol{U}$ kann beliebig gewählt werden
- Betrachte Formeln $\boldsymbol{F}$, die keine Variablen, Funktionssymbole und Quantoren enthalten ...
- ... sondern nur nullstellige Prädikatsymbole (als "atomare Formeln").
- $\boldsymbol{U^0}$ hat genau ein Element, das 'leere Tupel' $()$:

$$\boldsymbol{U^0} = \{\ ()\ \}$$

- Damit gibt es nur zwei Teilmengen von $\boldsymbol{U^0}$ (d.h. Prädikate): $\boldsymbol{U^0}$ (für 'wahr') und $\emptyset$ (für 'falsch').
- Punkt (3a) der Semantikdefinition liefert dann für nullstellige Prädikate $\boldsymbol{P}$ und eine passende Struktur $\boldsymbol{\alpha}$:

$$\boldsymbol{\alpha}(\boldsymbol{P}) := \begin{cases} \mathbf{1}, \text{ falls } () \in \boldsymbol{\psi}(\boldsymbol{P}) \\ \mathbf{0}, \text{ sonst.} \end{cases} = \begin{cases} \mathbf{1}, \text{ falls } \boldsymbol{\psi}(\boldsymbol{P}) = \boldsymbol{U^0} \\ \mathbf{0}, \text{ sonst.} \end{cases}$$

  d.h. die Belegung der 'atomaren Formeln' geschieht über die $\boldsymbol{\psi}$-Komponente von Strukturen.

Satz 5.2.3. *Für prädikatenlogische Formeln gelten die aussagenlogischen Äquivalenzen aus den Beispielen 1.1.23 und 1.1.24.*

Außerdem gelten z.B. die folgenden Äquivalenzen für Formeln $\boldsymbol{F}, \boldsymbol{G}, \boldsymbol{H}$, wenn $\boldsymbol{H}$ die Variable $\boldsymbol{x}$ nicht enthält:

1. $\neg(\forall \boldsymbol{x}\ \boldsymbol{F}) \equiv \exists \boldsymbol{x}\ \neg \boldsymbol{F}$
2. $\neg(\exists \boldsymbol{x}\ \boldsymbol{F}) \equiv \forall \boldsymbol{x}\ \neg \boldsymbol{F}$
3. $(\forall \boldsymbol{x}\ \boldsymbol{F}) \wedge (\forall \boldsymbol{x}\ \boldsymbol{G}) \equiv \forall \boldsymbol{x}\ (\boldsymbol{F} \wedge \boldsymbol{G})$
4. $(\exists \boldsymbol{x}\ \boldsymbol{F}) \vee (\exists \boldsymbol{x}\ \boldsymbol{G}) \equiv \exists \boldsymbol{x}\ (\boldsymbol{F} \vee \boldsymbol{G})$
5. $\forall \boldsymbol{x}\ \forall \boldsymbol{y}\ \boldsymbol{F} \equiv \forall \boldsymbol{y}\ \forall \boldsymbol{x}\ \boldsymbol{F}$
6. $\exists \boldsymbol{x}\ \exists \boldsymbol{y}\ \boldsymbol{F} \equiv \exists \boldsymbol{y}\ \exists \boldsymbol{x}\ \boldsymbol{F}$
7. $(\forall \boldsymbol{x}\ \boldsymbol{F}) \wedge \boldsymbol{H} \equiv \forall \boldsymbol{x}\ (\boldsymbol{F} \wedge \boldsymbol{H})$
8. $(\forall \boldsymbol{x}\ \boldsymbol{F}) \vee \boldsymbol{H} \equiv \forall \boldsymbol{x}\ (\boldsymbol{F} \vee \boldsymbol{H})$
9. $(\exists \boldsymbol{x}\ \boldsymbol{F}) \wedge \boldsymbol{H} \equiv \exists \boldsymbol{x}\ (\boldsymbol{F} \wedge \boldsymbol{H})$
10. $(\exists \boldsymbol{x}\ \boldsymbol{F}) \vee \boldsymbol{H} \equiv \exists \boldsymbol{x}\ (\boldsymbol{F} \vee \boldsymbol{H})$

Beweis: als Übung...

Aussagenlogische Gesetze konnten durch "Ausprobieren" überprüft werden. (Wahrheitswertetafel!)

Bei prädikatenlogischen Formeln sind Modelle oft unendlich, d.h. ein vollständiges Ausprobieren geht nicht!

Beispiel-Formel $\boldsymbol{F}$ mit erzwungenermaßen unendlichen Modellen:

$$\begin{aligned} &\forall \boldsymbol{x}\ \forall \boldsymbol{y}\ \forall \boldsymbol{z}\ ((\boldsymbol{P}(\boldsymbol{x}, \boldsymbol{y}) \wedge \boldsymbol{P}(\boldsymbol{y}, \boldsymbol{z})) \rightarrow \boldsymbol{P}(\boldsymbol{x}, \boldsymbol{z})) \\ \wedge\ &\forall \boldsymbol{x}\ \neg \boldsymbol{P}(\boldsymbol{x}, \boldsymbol{x}) \quad \wedge \quad \forall \boldsymbol{x}\ \boldsymbol{P}(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x})) \end{aligned}$$

- Passendes unendliches Modell:

$$U = \mathbb{N}_0, \quad \psi(P) = \{(a,b) \in {\mathbb{N}_0}^2 \mid a < b\}, \quad \phi(f) : n \mapsto n+1$$

- Andererseits: Sei α Modell von F, wähle beliebiges $u \in U$, dann:
  Definiere induktiv u_i durch $u_0 = u$ und $u_{i+1} = \phi(f)(u_i)$.
  Mit Induktion gilt für alle $i, j \in \mathbb{N}_0$, dass $(u_i, u_{i+j+1}) \in \psi(P)$.
  Wegen Teilformel $\forall x : \neg P(x,x)$ folgt damit jedoch, dass für $i \neq k$ auch $u_i \neq u_k$ gelten muss.
  Damit ist U unendlich!

Beispiele für Anwendungen der Prädikatenlogik in der Informatik:

Beispiel 5.2.4 (Prolog)**.** • *Bisher nur bekannt: Hornlogik in Prolog*

- *Prolog nutzt jedoch auch prädikatenlogische Schreibweisen:*

```
vater(adam,tobias).
vater(adam,ulrike).
vater(tobias,frank).
mutter(ulrike,claudia).
grossvater(X,Y) :-  vater(X,Z),  vater(Z,Y).
grossvater(X,Y) :-  vater(X,Z),  mutter(Z,Y).
```

  Anfragen z.B.:

```
?- grossvater(adam,claudia).
true.

?- grossvater(X,frank).
X=adam
```

Beispiel 5.2.5 (Spezifikationssprachen)**.** • *Prinzip:* `design-by-contract`*, Bertrand Meyer*

- *Spezifikation der Semantik eines Algorithmus durch Angabe von Vor- und Nachbedingungen (Dijkstra, Floyd, Hoare...)*
- *z.B. Java Modeling Language (JML)*
- *hier Beispiel aus* `ACSL`*:*
  `ACSL` = `ANSI/ISO C Specification Language`,
  Ziel: formal exakte Definition der Programmiersprache `C`
- *im Folgenden: Spezifikation einer Methode zur binären Suche:*
  (Beispiel 2.19 aus Manual zu Version 1.5 von `ACSL`*)*

```
/*@
 requires n >= 0 && \valid ( t +(0..n-1) );
 assigns \nothing ;
 ensures -1 <= \result <= n-1;
 behavior success :
   ensures \result >= 0 ==> t [\result] == v;
 behavior failure :
   assumes t_is_sorted :
     \forall integer k1 , integer k2 ;
       0 <= k1 <= k2 <= n-1 == > t[k1] <= t[k2];
   ensures \result == -1 == >
     \forall integer k ; 0 <= k < n == > t[k] != v;
*/
int bsearch ( double t[] , int n , double v );
```

Beispiel 5.2.6 (Beweisassistenten). • *computerunterstütztes Beweisen logischer Zusammenhänge:*

  - *Verifikation, dass ein gegebener Beweis richtig ist,*
  - *also dass nur erlaubte Beweisschritte verwendet werden*
  - *und dass im Beweis keine Lücken sind.*

- *Wegen zu hoher Komplexität (Laufzeit / benötigter Speicherplatz) i.d.R. unmöglich, dass Computer selbst Beweise finden...*
- *Beispiele: COQ, HOL, PVS, Mizar, ...*
- *Beispiel aus dem Manual vom COQ:*
  - *COQ ... is designed to ... verify that programs are correct with respect to their specification...*
  - *COQ also provides an interactive proof assistant to build proofs using specific programs called tactics.*

Beispiel 5.2.7 (Gödelscher erster Unvollständigkeitssatz). • *Ziel des 'Hilbertprogrammes' (David Hilbert, 1921):*

  - *Formalisierung der gesamten Mathematik durch ein Axiomensystem in Prädikatenlogik erster Stufe,*
  - *Nachweis der Widerspruchsfreiheit der Axiome.*

- *Zerschlagen durch (Kurt Gödel, 1931):*
  - *Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig (erster Unvollständigkeitssatz).*

  dabei bedeuted "hinreichend mächtig":

  - *Modelle enthalten mindestens die natürlichen Zahlen,*
  - *Addition und Multiplikation natürlicher Zahlen werden beschrieben,*
  - *zudem müssen elementare Eigenschaften von Zahlen beweisbar sein, wie 'Null ist kleinste natürliche Zahl' oder 'für alle (x, y) gibt es z mit $z = x \cdot y$'*

- *Damit später nachgewiesen: Es gibt prinzipielle Grenzen für die Fähigkeiten von Computern! ($\rightsquigarrow$ "Halteproblem", Vorlesung über "Berechenbarkeit")*