# Formale Systeme

Lernzusammenfassung zur Vorlesung am KIT
Lukas Bach - lbach@outlook.de - lukasbach.com

# 1 Aussagenlogik

## 1.1 Interpolanten

Für $A, B \in For0_\Sigma, \vDash A \to B$ ist ein $C \in For0_\Sigma$ eine Interpolante von $A \to B$, wenn

- $\vDash A \to C$ und $\vDash C \to B$
- und in C nur aussagenlogische Atome aus A und B vorkommen.

Für zwei $A, B \in For0\_\Sigma$ mit $\vDash A \to B$ existiert eine Interpolante (Craigsches Interpolationslemma).

## 1.2 Konj./Disj. Normalform

TODO, vgl. Skript Lemma 2.33

## 1.3 Shannon Formeln

$$val_I\big(sh(A_1, A_2, A_3)\big) = \begin{cases} val_I(A_2) & falls \quad val_I(A_1) = F \\ val_I(A_3) & falls \quad val_I(A_1) = W \end{cases}$$

$$sh(P_1, P_2, P_3) \leftrightarrow (\neg P_1 \wedge P_2) \vee (P_1 \wedge P_3)$$

## 1.4 Shannon Graphen

### 1.4.1 Reduktion von Shannon Graphen

Vgl. Skript Lemma 2.55. Für jedes Knotenpaar $(v, w)$, für das gilt: Die W-Nachfolger und die F-Nachfolger von v und w sind gleich, dann setze $v = w$. Das bedeutet auch, dass Knoten, die mit nur einer W-Kante und einer F-Kante zu einem einzigen Kindsknoten verbunden sind, durch dieses Kind ersetzt werden.

### 1.4.2 Formeln aus Shannon-Graph bilden

Um eine logisch äquivalente aussagenlogische Formel in DNF aus einem Shannon-Graphen zu bilden, wird aus jedem möglichen Pfad, der in einem 1-Knoten endet, die Konjunktion aller Entscheidungen auf dem Pfad gebildet und so die Konjunktionen aller Pfade per Disjunktion zu DNF zusammengeschlossen.

# 2 Erfüllbarkeitstests

## 2.1 Davis-Putnam-Logemann-Loveland (DPLL)

Prinzip: Solange Klauselmenge nicht leer ist wiederhole: suche Einerklauseln (Klauseln aus nur einem Literal), ersetze alle Literalvorkommen durch den Einerklauselwert (ggf. Klauseln löschen). Wenn keine Einerklausel mehr, wähle ein Atom, spalte Verfahren in zwei Pfade und setze das Atom in einem Pfad auf wahr, im anderen auf falsch (ggf. Klauseln löschen). Wenn die Klauselmenge leer ist, war sie erfüllbar. Stößt man in allen Pfaden auf Widersprüche, war die Klauselmenge nicht erfüllbar.

Formaler Algorithmus vgl. Skript Kapitel 3.1, insbesondere Beispiel 3.3.

## 2.2 Horn-Formeln

Eine Hornformel ist eine KNF-Formel, bei der jede Disjunktion (Klausel) höchstens ein positives Literal enthält. Die Klauseln lassen sich dann als Implikation schreiben.

### 2.2.1 Algorithmus zum Erfüllbarkeitstestem

(Markierungsalgorithmus.) Demarkiere zunächst alle Atome der Hornklauselmenge. Iteriere dann für alle Klauseln $A_1 \wedge A_2 \wedge \ldots \to B$, für die alle $A_i$ markiert sind, $B$ aber nicht. Markiere B und fahre fort.

Der Algorithmus terminiert, wenn keine passende Hornklausel gefunden werden kann (erfüllbar) oder wenn alle Atome markiert wurden. *Es können auch das 0- bzw das 1-Atom markiert werden.* Wird das 0-Atom markiert, so terminiert der Algorithmus mit „nicht erfüllbar“.

Im ersten Schritt wird immer ein „Fakt“ markiert, also eine Hornklausel der Form $1 \to B \equiv B$.

Klauselmengen ohne Fakten sind immer erfüllbar.

# 3 Prädikatenlogik 1. Ordnung

## 3.1 Anzahl Herbrand-Interpretationen

Frage: Wie viele Herbrand-Interpretationen gibt es über einer PL1-Signatur mit k Konstanten und n atomaren variablenfreien Formeln? $k^n$.

Atomare variablenfreie Formeln: Für zwei zweistellige Prädikate a, b und zwei Konstanten c, d gibt es zB 8 atomare variablenfreie Formeln: a(c,c), a(c,d), a(d,c), a(d,d) b(c,c) b(c,d)…

## 3.2 Unifikation nach Robinson

Solange Termmenge nicht nur einen Term enthält, wiederhole: Finde einen Teilterm t eines Terms, der sich an der Stelle des Teiltermvorkommens von einem anderen Term der Termmenge unterscheidet. Finde außerdem eine Variable x …Wenn dieser Teilterm nicht existiert, breche ab mit „nicht unifizierbar“. Ansonsten,

TODO Differenz von Mengen (Wann nicht unifizierbar)

TODO vielleicht lieber einfach ein Beispiel?

Occur-Check: Ist die Variable, durch die ersetzt werden soll, in dem zu ersetzenden Term vorhanden? Falls ja: Nicht unifizierbar. Notation: $x \in Var\big(f(x)\big)$ für zu ersetzende Variable x und Zielterm f(x).

Nicht unifizierbar da Konstanten-Kollision: „Nicht unifizierbar, die Menge enthält mehr als einen Term und die Differenz der Menge enthält keine Variable."

## 3.3 Pränex-Normalform

$A \in For\_\Sigma$ hat Pränex-Normalform, wenn A die Gestalt hat $Q_1 x_1 \dots Q_n x_n B$ mit Quantoren $Q_i \in \{\exists, \forall\}$ und B keine Quantoren enthält. B heißt Matrix von A.

Pränex-Normalform kann erreicht werden, indem die Quantoren nach vorne in der Formel geschoben werden. Bis auf folgende Sonderfälle können die Quantoren ohne Veränderung verschoben werden. Ggf. müssen überdeckte Variablen umbenannt werden.

$$\forall x A \to B \quad \equiv \quad \exists x (A \to B)$$
$$\exists x A \to B \quad \equiv \quad \forall x (A \to B)$$

## 3.4 Skolem-Normalform

Eine Formel ist in Skolem-Normalform, wenn sie

- geschlossen ist
- die Gestalt $\forall x_1 .. \forall x_n B$ mit quantoremfreien B hat
- und die Matrix B in KNF ist.

Vorgehensweise: Pränex-Normalform bilden, dann beseitigen der Existenzquantoren:

$$\begin{array}{cll} & \forall x_1 \dots \forall x_n \exists y B & hat\ ein\ Modell\ \text{über}\ \Sigma \\ \Leftrightarrow & \forall x_1 \dots \forall x_n \{y/g(x_1, \dots, x_n)\}(B) & hat\ ein\ Modell\ \text{über}\ \Sigma_g \end{array}$$

Dabei sind $x_i$ paarweise verschieden und nicht in B gebunden. Existieren keine Allquantoren, so wird statt dem n-stelligen Funktionssymbol g durch eine Konstante g (bzw ein 0-stelliges Funktionssymbol) ausgetauscht.

# 4 Beweistheorie

## 4.1 Resolutionskalkül

Vorgehensweise: Um $\{B_1, \dots, B_n\} \vDash A$ zu beweisen, bilde Formelmenge $\{B_1, \dots, B_n, \neg A\}$.

- Bringe Formeln zuerst in Pränex und dann in Skolem-Normalform (Formel-Matrizen in KNF)
- Lasse ∀-Präfixe weg und schreibe verbleibende KNF als Menge von Klauseln. Bilde Vereinigung der Klauselmengen.
- Versuche daraus mittels Variantenbildung von Klauseln und Resolution die leere Klausel herzuleiten.
  - Leite dafür aus zwei Klauselmengen $C_1 \cup K_1$ und $C_2 \cup K_2$ aus Klauseln $C_1, C_2, K_1 \neq \square, K_2 \neq \square$ mit je disjunkten Variablen eine neue Klauselmenge $\mu(C_1 \cup C_2)$ ab mit $\mu$ als allgemeinster Unifikator von $K_1 \cup \sim K_2$.
  - Dazu müssen in jedem Schritt zwei Klauselmengen Klauselmengen $C_1 \cup K_1$ und $C_2 \cup K_2$ gefunden werden, für die $K_1 \cup \sim K_2$ unifizierbar sind.
  - Als Schritt kann auch eine Variante einer bestehenden Klauselmenge erzeugt werden, in der Variablen umbelegt werden.

13PK1Resolution-print.pdf Folie 6

## 4.2 Aussagenlogische Tableukalkül

### 4.2.1 Tableuregeln

| **Abkömmlinge vom Typ $\alpha$:** | | |
|---|---|---|
| **Vorzeichenformel** | $V_1$ | $V_2$ |
| $\mathbf{0}\neg B$ | $1B$ | $1B$ |
| $\mathbf{1}\neg B$ | $0B$ | $0B$ |
| $\mathbf{1}(B \wedge C)$ | $1B$ | $1C$ |
| $\mathbf{0}(B \vee C)$ | $0B$ | $0C$ |
| $\mathbf{0}(B \to C)$ | $1B$ | $0C$ |
| Vorzeichenformel ist wahr wenn beide Abkömmlinge wahr sind. (UND) | | |

| **Abkömmlinge vom Typ $\beta$:** | | |
|---|---|---|
| **Vorzeichenformel** | $V_1$ | $V_2$ |
| | | |
| | | |
| $0(B \wedge C)$ | $0B$ | $0C$ |
| $1(B \vee C)$ | $1B$ | $1C$ |
| $1(B \to C)$ | $0B$ | $1C$ |
| Vorzeichenformel ist wahr wenn eins der Abkömmlinge wahr ist. (ODER) | | |

Ein Pfad im Tableukalkül enthält

- eine vertikale Erweiterung im Falle vom $\alpha$-Typ, bei dem jeder Knoten wahr sein muss,
- oder eine horizontale aufteilende Erweiterung im Falle vom $\beta$ Typ, bei dem einer der Teilpfade wahr sein muss.

TODO vielleicht besser als Tableuregeln formalisieren.

### 4.2.2 Generierung von Tableuregeln

TODO, vgl Skript Definition 5.42

## 4.3 Prädikatenlogischer Tableukalkül

### 4.3.1 Tableuregeln

| **Abkömmlinge vom Typ $\alpha$:** | | |
|---|---|---|
| **Vorzeichenformel** | $V_1$ | $V_2$ |
| $1\neg A$ | $0A$ | – |
| $0\neg B$ | $1A$ | – |
| $1A \wedge B$ | $1A$ | $1B$ |
| $0A \vee B$ | $0A$ | $0B$ |
| $0A \to B$ | $1A$ | $0B$ |
| Vorzeichenformel ist wahr wenn beide Abkömmlinge wahr sind. (UND) | | |

| **Abkömmlinge vom Typ $\beta$:** | | |
|---|---|---|
| **Vorzeichenformel** | $V_1$ | $V_2$ |
| | | |
| | | |
| $0A \wedge B$ | $0A$ | $0B$ |
| $1A \vee B)$ | $1A$ | $1B$ |
| $1(A \to B)$ | $0A$ | $1B$ |
| Vorzeichenformel ist wahr wenn eins der Abkömmlinge wahr ist. (ODER) | | |

| **Abkömmlinge vom Typ $\gamma$:** | | |
|---|---|---|
| **Vorzeichenformel** | $V_1$ | |
| $1\forall x A(x)$ | $1A(x)$ | |
| $0\exists x A(x)$ | $0A(x)$ | |
| Vorzeichenformel ist wahr wenn Abkömmling für alle Variablenbelegung wahr ist. (FORALL) | | |

| **Abkömmlinge vom Typ $\delta$:** | | |
|---|---|---|
| **Vorzeichenformel** | $V_1$ | |
| $1\exists x A(x)$ | $1A(x)$ | |
| $0\forall x A(x)$ | $0A(x)$ | |
| Vorzeichenformel ist wahr wenn Abkömmling für eine Variablenbelegung wahr ist. (EXISTS) | | |

$\gamma$-Varialen werden durch $X_i$ Variablen ersetzt, $\delta$-Variablen durch $sk_i(x_1, x_2, \dots)$ mit $x_i$ als alle freien Variablen innerhalb des ursprünglichen Quantors.

### 4.3.2 Schließende Substitution

EIne Substitution $\sigma$ schließt einen Pfad $\pi$, wenn es Formeln B und C gibt mit $\sigma(B) = \sigma(C)$ und für die $\sigma$ kollisionsfrei ist

sowie 1B und 0C auf $\pi$ liegen. $\sigma$ schließt außerdem $\pi$, wenn auf $\pi$ eine der Formeln 01 oder 10 liegen.

Um ein Tableau abzuschließen, müssen alle Pfade abgeschlossen werden. Die finale schließende Subsitution ist die Menge aller schließenden Substitutionen aller Pfade des Tableaus.

TODO C-Regel (Abschlussregel)

#### 4.3.3 Baumnotation

(TODO, siehe Seite 192)

# 5 Gleichheitslogik

## 5.1 Reduktionssystem

### 5.1.1 Definition Reduktionssystem

Ein Reduktionssystem $(D, \succ)$ besteht aus einer nichtleeren Menge D und einer binären Relation $\succ$ auf D.

Weiter ist $\rightarrow$ die reflexive transitive Hülle von $\succ$.

### 5.1.2 Eigenschaften von Reduktionssystemen

Ein Reduktionssystem $(D, \succ)$ heißt…

- *konfluent*, wenn $\forall s, s_1, s_2 \in D: (s \rightarrow s_1, s \rightarrow s_2) \Rightarrow \exists t \in D: s_1 \rightarrow t, s_2 \rightarrow t$.
- *lokal konfluent*, wenn $\forall s, s_1, s_2 \in D: (s \succ s_1, s \succ s_2) \Rightarrow \exists t \in D: s_1 \rightarrow t, s_2 \rightarrow t$
- *noethersch*, wohlfundiert oder terminierend, wenn es keine unendliche Folge $s_0 \succ s_1 \succ \cdots \succ s_i \succ \cdots$ gibt.
- *kanonisch*, wenn es konfluent und noethersch ist.

Ein Element $s \in D$ heißt *irreduziebel* oder Normalform, wenn $\neg\exists t \in D: s \succ t$.

Ein Element $s \in D$ heißt *Normalform für s* in $(D, \succ)$, wenn $s_0$ irreduzibel ist und $s \rightarrow s_0$ gilt.

# 6 Modale Aussagenlogik

## 6.1 Modaloperatoren

Eine Aussage ◇A („A ist *möglich*“) ist im Zustand s wahr, wenn von s ein Zustand t *in einem Schritt* erreichbar ist, in dem A wahr ist.

Eine Aussage □A („A ist *notwendig*“) ist im Zustand s wahr, wenn in allen von s *in einem Schritt* erreichbaren Zustände t die Aussage A wahr ist.

## 6.2 Kripke-Struktur

Kripe-Struktur K=(S,R,I) aus

- $S$: nichtleere Menge von Zuständen
- $R \subseteq S \times S$: Relation
- $I: (\Sigma \times S) \rightarrow \{W, F\}$: Interpretationsfunktion

bzw. Kripke-Rahmen (S,R).

Eine Formel wird immer im Kontext eines konkreten Zustandes ausgewertet, daher der Wahrheitswert der Formel ist von diesem abhängig.

$$
\begin{aligned}
val_s(\mathbf{0}) &= \mathbf{F} \\
val_s(\mathbf{1}) &= \mathbf{W} \\
val_s(P) &= I(P, s) \text{ für Atome } P \\
val_s(\neg A) &= \begin{cases} \mathbf{F} & \text{falls } val_s(A) = \mathbf{W} \\ \mathbf{W} & \text{falls } val_s(A) = \mathbf{F} \end{cases} \\
val_s(A \circ B) &= \begin{cases} \text{Wie in der Aussagenlogik} \\ \text{(wobei } \circ \text{ eine aussagenlogische Verknüpfung ist)} \end{cases} \\
val_s(\Box A) &= \begin{cases} W & \text{falls für alle } s' \in S \text{ mit } sRs' \text{ gilt } val_{s'}(A) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases} \\
val_s(\Diamond A) &= \begin{cases} W & \text{falls ein } s' \in S \text{ existiert mit } sRs' \\ & \text{und } val_{s'}(A) = \mathbf{W} \\ \mathbf{F} & \text{sonst} \end{cases}
\end{aligned}
$$

1. Jede aussagenlogisch Tautologie ist eine allgemeingültige modale Formel.
2. $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$
3. $(\Box A \wedge \Box B) \leftrightarrow \Box(A \wedge B)$
4. $(\Box A \vee \Box B) \rightarrow \Box(A \vee B)$
5. $\Box A \leftrightarrow (\neg\Diamond\neg A)$
6. $\Diamond A \leftrightarrow (\neg\Box\neg A)$
7. $(\Diamond A \vee \Diamond B) \leftrightarrow \Diamond(A \vee B)$
8. $\Diamond(A \wedge B) \rightarrow (\Diamond A \wedge \Diamond B)$
9. $(\Box(A \rightarrow B) \wedge \Box(B \rightarrow A)) \leftrightarrow \Box(A \leftrightarrow B)$
10. $\Box(A \rightarrow B) \rightarrow (\Diamond A \rightarrow \Diamond B)$

**Lemma 8.9 (Relative Allgemeingültigkeit modaler Formeln)**
In den folgenden Formeln ist $A$ eine aussagenlogische Variable.

1. In der Klasse aller reflexiven Kripke-Strukturen ist allgemeingültig:
   $\Box A \rightarrow A$
2. In der Klasse aller transitiven Kripke-Strukturen ist allgemeingültig:
   $\Box A \rightarrow \Box\Box A$
3. In der Klasse aller symmetrischen Kripke-Strukturen ist allgemeingültig:
   $A \rightarrow \Box\Diamond A$
4. In der Klasse aller dichten Kripke-Strukturen ist allgemeingültig:
   $\Box\Box A \rightarrow \Box A$
   Dabei heißt eine Kripke-Struktur $(S, R, I)$ dicht, wenn für alle $t_1, t_2 \in S$ mit $R(t_1, t_2)$ eine Welt $t_3 \in S$ existiert mit $R(t_1, t_3)$ und $R(t_3, t_2)$.
5. In der Klasse aller partiell funktionalen Kripke-Strukturen ist allgemeingültig:
   $\Diamond A \rightarrow \Box A$
   Dabei heißt eine Kripke-Struktur $(S, R, I)$ partiell funktional, wenn für alle $s, t_1, t_2 \in S$ mit $R(s, t_1)$ und $R(s, t_2)$ folgt $t_1 = t_2$.
6. In der Klasse aller endlosen Kripke-Strukturen ist allgemeingültig:
   $\Box A \rightarrow \Diamond A$
   Dabei heißt eine Kripke-Struktur $(S, R, I)$ endlos, wenn für jedes $s \in S$ ein $t$ existiert mit $R(s, t)$.

## 6.3 Lineare Temporale Logik

Eine LTL entspricht einer Kripke-Struktur (S,R,I) mit R als strikte Quasi-Ordnung <. Zustände $s \in S$ werden als Zeitpunkte bezeichnet.

Für zwei Zeitpunkte gilt $s_1 \prec s_3 \Leftrightarrow \neg\exists s_2: s_1 < s_2 < s_3$.

### 6.3.1 Omega-Struktur

Eine Omega-Struktur $(\mathbb{N}, <, \xi)$ ist eine LTL-Struktur mit $s = \mathbb{N}, R =<$ und $\xi: \mathbb{N} \to 2^{\Sigma}, p \in \xi(n) \Leftrightarrow$ in der Omega-Struktur ist p zum Zeitpunkt n wahr. Außerdem $\xi_n(m) = \xi(n+m)$.

### 6.3.2 Semantik

- $\xi \vDash p: p \in \xi(0)$
- $\xi \vDash \Box A: \forall n \in \mathbb{N}: \xi_n \vDash A$
- $\xi \vDash \diamondsuit A: \exists n \in \mathbb{N}: \xi_n \vDash A$
- $\xi \vDash \mathrm{A\ U\ B}: \exists n \in \mathbb{N}: (\xi_n \vDash B \wedge \forall m: 0 \leq m < n: \xi_m \vDash A)$ („until")
- $\xi \vDash X\ A: \xi_1 \vDash A$
- $\xi \vDash A\ U_w\ B$: $\left(\forall n \in \mathbb{N}: \xi_n \vDash (A \wedge \neg B)\right) \vee A\ U\ B$ („schwacher Until")
- $\xi \vDash A\ V\ B: ((\xi \vDash B) \wedge (\forall n \in \mathbb{N}: (\xi_n \vDash \neg B) \Rightarrow \exists m \in \mathbb{N}: (0 \leq m < n \wedge \xi_m \vDash A))$ („Release")

Achtung, Unterschied zu Modaler Aussagenlogik: In modaler Aussagenlogik bedeutet $(S, R, I) \vDash p$ dass p in allen Zuständen gilt. In LTL bedeutet $\xi \vDash p$ dass p nur am Anfang notwendigerweise gilt.

1. $A\ \mathbf{U}\ B \leftrightarrow A\ \mathbf{U}_w\ B \wedge \diamondsuit B$
2. $A\ \mathbf{U}_w\ B \leftrightarrow A\ \mathbf{U}\ B \vee \Box(A \wedge \neg B)$
3. $A\ \mathbf{V}\ B \leftrightarrow \neg(\neg A\ \mathbf{U}\ \neg B)$
4. $A\ \mathbf{U}\ B \leftrightarrow \neg(\neg A\ \mathbf{V}\ \neg B)$
5. $A\ \mathbf{U}\ B \leftrightarrow (B \vee (A \wedge X\ (A\ \mathbf{U}\ B)))$
6. $A\ \mathbf{V}\ B \leftrightarrow (B \wedge A) \vee (B \wedge X\ (A\ \mathbf{V}\ B))$
7. $\neg(A\ \mathbf{V}\ B) \leftrightarrow \neg A\ \mathbf{U}\ \neg B$
8. $\neg(A\ \mathbf{U}\ B) \leftrightarrow \neg A\ \mathbf{V}\ \neg B$

## 6.4 Modellprüfung

### 6.4.1 Büchi-Automatenkonstruktion aus LTL-Formel

Aus einer LTL Formel B wird ein Büchi-Automat $\mathcal{A}_B$ erzeugt.

- Vokabular: $V = 2^P$ für P als Menge der aussagenlogischen Atome aus B.
- Zustände: $S = \{\, s \subseteq TeilFormeln(B) \mid 0 \notin s,$
  $(C_1 \wedge C_2) \in s \Rightarrow C_1 \in s\ und\ C_2 \in s,$
  $(C_1 \vee C_2) \in s \Rightarrow C_1 \in s\ oder\ C_2 \in s\ \}$
- Anfangszustände: $s_0 = \{s \in S | B \in s\}$
- Übergangsfunktion: Für $s, t \in S$ und $a \in 2^P$ gilt $t \in \delta(s, a)$ wenn alle Bedingungen gelten:
  - $\forall p \in P: p \in s \Rightarrow p \in a$
  - $\forall p \in P: \neg p \in s \Rightarrow p \notin a$
  - $XA \in s \Rightarrow A \in t$
  - $(A\ U\ B) \in s \Rightarrow B \in s \vee (A \in s \wedge (A\ U\ B) \in t)$
  - $(A\ V\ B) \in s \Rightarrow (B \in s \wedge A \in s) \vee (B \in s \vee (A\ V\ B) \in t)$
- Endzustandsmengen:
  - Sei $UntilFormeln_i \subseteq TeilFormeln(B) = \{A_i\ U\ B_i\}$
  - Endzustand i: $F_i = \{s \in S | UntilFormeln_i \notin s\ \ oder\ \ UntilFormeln_i \in s\ \ und\ B_i \in s\}$
  - Alle Endzustandsmengen: $\{F_i | 1 \leq i \leq k\}$.

# 7 JML

## 7.1 Prädikatenlogische Notationen

==, &&, ||, !, ==>, <==>, (\forall C x; e1; e2), (\exists C x; e1; e2)

(C ist der Datentyp von x in forall/exists, zB int)

## 7.2 Methodenkontrakte

```
/*@ public normal_behaviour
  @   requires ...;
  @   assignable a, b, c;
  @   ensures ...;
  @*/
```

Requires definiert Vorbedinungen, ensures definiert Nachbedinungen. In der Nachbedinung können Variablen \result und \old(Varname) verwendet werden.

Assignable schränkt die Variablen ein, die geändert werden dürfen. Mit assignable \nothing kann die Methode als pur erklärt werden.

## 7.3 Schleifeninvarianten

```
/*@ loop_invariant (invariant...);
  @ assignable a, b, c;
  @ decreases abc;
  @*/
```

invariant beschreibt die Schleifeninvariante, die vor der Schleife, nach der Schleife und bei jedem Schleifendurchlauf gelten muss.

decreases beschreibt einen Ausdruck, der bei jedem Schleifendurchlauf $\geq 0$ ist und in jedem Schleifendurchlauf echt kleiner wird. Damit soll die Terminierung der Schleife überprüft werden.

## 7.4 Klasseninvarianten

```
/*@ invariant (invariant...);
  @*/
```

## 7.5 Verallgemeinerte Quantoren

```
(\sum     T x; R; t)
(\product T x; R; t)
(\max     T x; R; t)
(\min     T x; R; t)
```

Ein Quantor kann genauso wie \forall und \exists eingesetzt werden, gibt aber einen konkreten Wert zurück und wird in einem Kontrakt als Variablenwert eingesetzt.