Formale Grundlagen der Informatik 3 –

11. Prädikatenlogik und Symbolische Auswertung

Christoph Walther
TU Darmstadt

1 Sorten, Signaturen, Terme

Definition 1 *(Sorten und Signaturen)*

(1) $\mathcal{S} \neq \emptyset$ sei eine endliche Menge, genannt die Menge der **Sortensymbole**.

(2) Für jedes $w \in \mathcal{S}^*$ und jedes $s \in \mathcal{S}$ sei Σ_{ws} eine endliche Menge von Funktionssymbolen mit $\Sigma_{ws} \cap \Sigma_{w's'} = \emptyset$ für $ws \neq w's'$.

(3) $\Sigma := (\Sigma_{ws})_{ws \in \mathcal{S}^*\mathcal{S}}$ heißt dann eine $\mathcal{S}$-**Signatur**.

Beispiel 1

(1) $\mathcal{S} := \{bool, nat, stack\}$ (2) $\Sigma_{bool} := \{true, false\}$

(3) $\Sigma_{nat} := \{O\}$, $\Sigma_{nat,nat} := \{succ\}$

(4) $\Sigma_{stack} := \{empty\}$, $\Sigma_{nat,stack,stack} := \{push\}$

Bedeutung:

(1) $bool, nat$ und $stack$ sind *Namen* von Mengen.

(2) Für die Sorte $bool$ gibt es 2 Funktionen, die *Konstanten* $true$ und $false$.

(3) O ist Konstante der Sorte nat, $succ$ ist Funktionssymbol mit $succ : nat \to nat$.

(4) $empty$ ist Konstante der Sorte $stack$, $push$ ist Funktionssymbol mit $push : nat \times stack \to stack$.

Definition 2 *(Terme über Σ und $\mathcal{V}$)*

(1) Für $s \in \mathcal{S}$ ist $\mathcal{V}_s$ unendliche Menge der **Variablensymbole** von s mit $\mathcal{V}_s \cap \mathcal{V}_{s'} = \emptyset$ für $s \neq s'$. Wir definieren $\mathcal{V} := (\mathcal{V}_s)_{s \in \mathcal{S}}$ und fordern $\mathcal{V} \cap \Sigma = \emptyset$.

(2) $\mathcal{T}(\Sigma, \mathcal{V})_s \subset (\mathcal{V} \cup \Sigma)^*$ ist die Menge aller **Terme** der Sorte s über $\mathcal{V}$ und Σ, definiert durch: Für jedes $t \in (\mathcal{V} \cup \Sigma)^*$ gilt $t \in \mathcal{T}(\Sigma, \mathcal{V})_s$ gdw.

(a) $t \in \mathcal{V}_s$ oder

(b) $t = f t_1 \ldots t_n$, $f \in \Sigma_{s_1,\ldots,s_n,s}$ und $t_1 \in \mathcal{T}(\Sigma, \mathcal{V})_{s_1}, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})_{s_n}$

(3) $\mathcal{T}(\Sigma, \mathcal{V}) := (\mathcal{T}(\Sigma, \mathcal{V})_s)_{s \in \mathcal{S}}$, die Menge aller Terme

(4) $\mathcal{T}(\Sigma)_s := \mathcal{T}(\Sigma, \emptyset)_s$, die Menge der **Grund**terme der Sorte s

(5) $\mathcal{T}(\Sigma) := (\mathcal{T}(\Sigma)_s)_{s \in \mathcal{S}}$.

Beispiel 2 *Seien $n, m \in \mathcal{V}_{nat}$ und $k \in \mathcal{V}_{stack}$:*

$true, false \in \mathcal{T}(\Sigma, \mathcal{V})_{bool}, O \in \mathcal{T}(\Sigma, \mathcal{V})_{nat}, empty \in \mathcal{T}(\Sigma, \mathcal{V})_{stack}$

$succ(succ(O)) \in \mathcal{T}(\Sigma, \mathcal{V})_{nat}$

$push(succ(succ(O)), push(O, push(succ(O), empty))) \in \mathcal{T}(\Sigma, \mathcal{V})_{stack}$

$n, m \in \mathcal{T}(\Sigma, \mathcal{V})_{nat}, k \in \mathcal{T}(\Sigma, \mathcal{V})_{stack}$

$push(n, push(O, push(succ(m), k))) \in \mathcal{T}(\Sigma, \mathcal{V})_{stack}$.

Stimmt eigentlich nicht: Es muß z.B.

$$\text{“}push\ succ\ succ\ O\ push\ O\ push\ succ\ O\ empty \in \mathcal{T}\left(\Sigma, \mathcal{V}\right)_{stack}\text{”}$$

heißen.

Also: In *Beispielen* verwenden wir Klammern und Kommata **!**

Definition 3 *(Sensible Signaturen)*

*Eine $\mathcal{S}$-Signatur Σ ist **sensibel** gdw. $\mathcal{T}\left(\Sigma\right)_s \neq \emptyset$ für alle $s \in \mathcal{S}$.*

Beispiel 3

(1) Die Signatur Σ aus Beispiel 1 ist sensibel.

(2) Für $\mathcal{S} = \{void\}$ ist die $\mathcal{S}$-Signatur Σ mit $\Sigma_{void} = \emptyset$ und $\Sigma_{void,void} = \{f\}$ nicht sensibel.

Ab jetzt: Wir betrachten nur noch sensible $\mathcal{S}$-Signaturen Σ **!**

- Es gilt $\mathcal{T}\left(\Sigma, \mathcal{V}\right) \subset \left(\mathcal{V} \cup \Sigma\right)^*$, d.h. $\mathcal{T}\left(\Sigma, \mathcal{V}\right)$ ist eine bestimmte Menge von Worten, d.h. $\mathcal{T}\left(\Sigma, \mathcal{V}\right)$ ist eine formale Sprache.
- Welche Bedeutung sollen die Worte aus $\mathcal{T}\left(\Sigma, \mathcal{V}\right)$ haben, d.h. was soll mit Elementen aus $\mathcal{T}\left(\Sigma, \mathcal{V}\right)$ beschrieben werden?

2 Σ-Algebren

Definition 4 *(Σ-Algebra)*

Sei $\mathcal{S}$ Menge von Sortensymbolen und sei Σ eine $\mathcal{S}$-Signatur. Dann ist eine Σ-Algebra ein Paar $A = (\mathcal{A}, \alpha)$ mit

(1) $\mathcal{A} = (\mathcal{A}_s)_{s \in \mathcal{S}}$ ($\mathcal{A}$ ist eine Familie von **Trägermengen** $\mathcal{A}_s$ für jede Sorte $s \in \mathcal{S}$)

(2) $\mathcal{A}_s \neq \emptyset$ für jedes $s \in \mathcal{S}$ (keine Trägermenge ist leer)

(3) $\alpha = (\alpha_f)_{f \in \Sigma}$ (α ist Familie von **Deutungsfunktionen** α_f für jedes $f \in \Sigma$)

(4) $\alpha_f : \mathcal{A}_{s_1} \times \ldots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$ für jedes $f \in \Sigma_{s_1,\ldots,s_n,s}$.

Bedeutung:

- Jedem Sortensymbol s wird eine nicht-leere Trägermenge $\mathcal{A}_s$ zugeordnet
- Jedem Funktions*symbol* f wird mittels der Deutungsfunktion α eine **totale** *Funktion* α_f auf diesen Trägermengen zugeordnet
- Die Deutungsfunktion respektiert die Signatur **!**

Beispiel 4

$\mathcal{S} := \{bool, nat\}, \Sigma_{bool} := \{true, false\}, \Sigma_{nat} := \{O\}, \Sigma_{nat,nat} := \{succ\}.$

(1) $\mathcal{A}_{bool} = \{\top, \bot\}$

(2) $\mathcal{A}_{nat} = \mathbb{N}$

(3) $\alpha_{true} := \top$

(4) $\alpha_{false} := \bot$

(5) $\alpha_{O} := 0$

(6) $\alpha_{succ}(n) := n + 1$

Bedeutung:

$true$ und $false$ werden als *verschiedene* Konstanten gedeutet. Das Konstantensymbol O wird als 0 und das Funktionssymbol $succ$ als Nachfolgerfunktion in den natürlichen Zahlen gedeutet.

Beispiel 5

$\mathcal{S} := \{bool, nat\}\,, \Sigma_{bool} := \{true, false\}\,, \Sigma_{nat} := \{O\}\,, \Sigma_{nat,nat} := \{succ\}\,.$

(1) $\mathcal{A}_{bool} = \mathbb{N}$

(2) $\mathcal{A}_{nat} = \mathbb{R}$

(3) $\alpha_{true} := 7$

(4) $\alpha_{false} := 7$

(5) $\alpha_O := \sqrt[3]{2}$

(6) $\alpha_{succ}(n) := 1/(n^2 + 1)$

Ist das denn sinnvoll ???

Unwichtig – “sinnvoll” ist nirgends gefordert!

$(\mathcal{A}, \alpha)$ ist eine Σ-Algebra, nur das ist hier gefragt.

Beispiel 6

$\mathcal{S} := \{bool, nat\}, \Sigma_{bool} := \{true, false\}, \Sigma_{nat} := \{O\}, \Sigma_{nat,nat} := \{succ\}.$

(1) $\mathcal{A}_{bool} = \{\top, \bot\}$

(2) $\mathcal{A}_{nat} = \mathbb{N}$

(3) $\alpha_{true} := \top$

(4) $\alpha_{false} := \bot$

(5) $\alpha_{O} := \bot$

(6) $\alpha_{succ}(n) := \top$

$(\mathcal{A}, \alpha)$ ist *keine* Σ-Algebra !

Es gilt $O \in \Sigma_{nat}$ sowie $succ \in \Sigma_{nat,nat}$ und damit ist

$\alpha_O :\rightarrow \{\top, \bot\}$ und $\alpha_{succ} : \mathbb{N} \rightarrow \{\top, \bot\}$

verboten (Signatur wird nicht respektiert!)

3 Deutung von Termen durch Σ-Algebren

Mit Σ-Algebren werden Grundterme gedeutet, d.h. die Worte aus $\mathcal{T}(\Sigma)$ bekommen eine *Bedeutung*:

Definition 5 *(Deutung von Grundtermen)*

*Sei $\mathcal{S}$ Menge von Sortensymbolen, Σ eine $\mathcal{S}$-Signatur, $A = (\mathcal{A}, \alpha)$ eine Σ-Algebra. Dann ist für jedes $t \in \mathcal{T}(\Sigma)_s$ die **Deutung** $A(t) \in \mathcal{A}_s$ definiert durch:*

(1) $A(t) := \alpha_f$, falls $t = f \in \Sigma_s$ für eine Sorte $s \in \mathcal{S}$

(2) $A(t) := \alpha_f(A(t_1), \ldots, A(t_n))$, falls $t = ft_1 \ldots t_n$ und $f \in \Sigma_{s_1,\ldots,s_n,s}$ für gewisse Sorten $s_1, \ldots, s_n, s \in \mathcal{S}$.

Bedeutung:

- Jedes Konstantensymbol f wird durch ein Element α_f der Trägermenge gedeutet
- Für jeden (nicht-Konstanten) Term $ft_1 \ldots t_n$ werden (rekursiv) zunächst die Deutungen $A(t_1), \ldots, A(t_n)$ der Argumente $t_1, \ldots, t_n$ bestimmt. Man erhält eine Folge von Elementen $a_1, \ldots, a_n$ der Trägermenge(n). Auf diese Elemente wird dann die dem Funktionssymbol f mittels der Deutungsfunktion zugeordnete Funktion α_f angewendet, also $A(ft_1 \ldots t_n) := \alpha_f(a_1, \ldots, a_n)$.

Bemerkung 1 Das kennen wir eigentlich schon – z.B. Ausrechnen von $\mathsf{3} * \mathsf{5} + \mathsf{7}$:

- $\mathsf{3}, *, \mathsf{5}, +$ und $\mathsf{7}$ sind *Symbole*. Bedeutung:
  - $\mathsf{3} := 3, \mathsf{5} := 5, \mathsf{7} := 7$, d.h. diesen Konstanten*symbolen* werden konkrete natürlich Zahlen zugeordnet.
  - $* := \mathit{Multiplikation}$ *in* $\mathbb{N}, + := \mathit{Addition}$ *in* $\mathbb{N}$, d.h. diesen Funktions*symbolen* werden konkrete *Funktionen* auf den natürlichen Zahlen zugeordnet.
- Wie wird die Bedeutung festgelegt?
  - Mündliche Überlieferung – Schule: Das kleine *Einmaleins*
  - Durch Implementierung in einem Rechner (Taschenrechner, PC, ...)
  - *Formal*: Durch eine Σ-Algebra !

- Deutung von Termen durch Σ-Algebren, d.h. die Worte aus $\mathcal{T}(\Sigma, \mathcal{V})$ bekommen eine *Bedeutung*.
- Was machen wir mit den Variablensymbolen ?

Definition 6 *(A-Variablenbelegung)*

Sei $A = (\mathcal{A}, \alpha)$ eine Σ-Algebra (bzgl. Sortenmenge $\mathcal{S}$). Dann heißt eine totale Abbildung

$$\mathfrak{a} : \mathcal{V} \to_{\mathcal{S}} \mathcal{A}$$

*eine A-**Variablenbelegung**.*

Für $y \in \mathcal{V}_s$ und $a \in \mathcal{A}_s$ ist $\mathfrak{a}[y/a] : \mathcal{V} \to_{\mathcal{S}} \mathcal{A}$ definiert durch: $\mathfrak{a}[y/a](x) := a$, falls $x = y$, und $\mathfrak{a}[y/a](x) := \mathfrak{a}(x)$ andernfalls.

*Jede A-**Variablenbelegung** $\mathfrak{a} : \mathcal{V} \to_{\mathcal{S}} \mathcal{A}$ wird zu einer Abbildung*

$$\widehat{\mathfrak{a}} : \mathcal{T}(\Sigma, \mathcal{V}) \to_{\mathcal{S}} \mathcal{A}$$

erweitert durch:

(1) $\widehat{\mathfrak{a}}(t) := \mathfrak{a}(x)$, falls $t = x \in \mathcal{V}$

(2) $\widehat{\mathfrak{a}}(t) := \alpha_f$, falls $t = f \in \Sigma_s$ für eine Sorte $s \in \mathcal{S}$

(3) $\widehat{\mathfrak{a}}(t) := \alpha_f(\widehat{\mathfrak{a}}(t_1), \ldots, \widehat{\mathfrak{a}}(t_n))$, falls $t = f t_1 \ldots t_n$ und $f \in \Sigma_{s_1,\ldots,s_n,s}$ für gewisse Sorten $s_1, \ldots, s_n, s \in \mathcal{S}$.

Bemerkung 2

- Es gilt $\widehat{\mathfrak{a}}(t) = A(t)$ für jeden Grundterm $t \in \mathcal{T}(\Sigma)$.
- Mit $\widehat{\mathfrak{a}}$ werden Terme $t \in \mathcal{T}(\Sigma, \mathcal{V})$ genauso gedeutet wie durch die Σ-Algebra A. Falls jedoch Variablensymbole x in t vorkommen, so werden diese mittels der Variablenbelegung $\mathfrak{a}$ gedeutet, also als $\mathfrak{a}(x)$.
- *Ab jetzt:* Keine Unterscheidung zwischen $\widehat{\mathfrak{a}}$ und $\mathfrak{a}$, d.h. wir schreiben immer $\mathfrak{a}$.
- Kleine Hilfe für die Übungen: Schreiben Sie $\overline{a}$ anstatt $\mathfrak{a}$, falls Ihnen Frakturbuchstaben nicht so leicht von der Hand gehen.

Bemerkung 3 Das kennen wir auch schon – z.B. Ausrechnen von $3 * \mathtt{x} + \mathtt{y}$:

- Eine A-Variablenbelegung modelliert den Speicher einer Rechners.
- Gelangt der Rechner bei Auswertung eines Ausdrucks (= Term) an eine Variable x, so wird aus dem Speicher die dort abgelegte Zuordnung (formal: $\mathfrak{a}(x)$) zur Berechnung verwendet.
- "Speicherbelegung" $\mathfrak{a}[y/a]$ entsteht aus "Speicherbelegung" $\mathfrak{a}$, indem der Inhalt der Variablen y mit a "überschrieben" wird.

4 Deutung von Formeln durch Σ-Algebren

Definition 7 *(Formeln über $\mathcal{V}$ und Σ)*

Gegeben $\mathcal{S}$-Signatur Σ sowie $\mathcal{V}$.

*Die Menge $\mathcal{AT}(\Sigma, \mathcal{V}) \subset (\mathcal{V} \cup \Sigma \cup \{\equiv\})^*$ der **atomaren Formeln** über $\mathcal{V}$ und Σ ist definiert durch:*

(1) $t_1 \equiv t_2 \in \mathcal{AT}(\Sigma, \mathcal{V})$, falls $t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{V})_s$ für ein $s \in \mathcal{S}$.

*Die Menge $\mathcal{F}(\Sigma, \mathcal{V}) \subset (\mathcal{V} \cup \Sigma \cup \{\equiv, \neg, \wedge, \forall\})^*$ der **Formeln** über $\mathcal{V}$ und Σ ist definiert durch:*

(1) $\phi \in \mathcal{F}(\Sigma, \mathcal{V})$, falls $\phi \in \mathcal{AT}(\Sigma, \mathcal{V})$

(2) $\neg\phi \in \mathcal{F}(\Sigma, \mathcal{V})$, falls $\phi \in \mathcal{F}(\Sigma, \mathcal{V})$

(3) $\phi_1 \wedge \phi_2 \in \mathcal{F}(\Sigma, \mathcal{V})$, falls $\phi_1, \phi_2 \in \mathcal{F}(\Sigma, \mathcal{V})$

(4) $\forall x{:}s\ \phi \in \mathcal{F}(\Sigma, \mathcal{V})$, falls $s \in \mathcal{S}$, $x \in \mathcal{V}_s$ und $\phi \in \mathcal{F}(\Sigma, \mathcal{V})$.

Bemerkung 4

- Anstatt Prädikatensymbole P verwenden wir Funktionssymbole $p \in \Sigma_{w,bool}$ und schreiben $p(\ldots) \equiv true$ anstatt $P(\ldots)$.
- *Grund:* => **Kapitel 9** (Folien 6 und 7)

Schreibweisen

- In Beispielen Klammern und Kommata wie bei Termen.
- Kurz "$\forall x\ \phi$" anstatt "$\forall x{:}s\ \phi$", wenn s beliebig ist.
- Wir verwenden in Beispielen auch die üblichen Junktoren $\vee, \rightarrow, \leftrightarrow$ und den Existenzquantor $\exists$. Ausdrücke der erweiterten Sprache werden formal – wie üblich – als Abkürzung für Ausdrücke der Kernsprache aufgefaßt. Z.B. steht "$\forall x\ \phi \rightarrow \exists y\ \psi$" für "$\neg(\forall x\ \phi \wedge \forall y\ \neg\psi)$".
- In Beispielen lassen wir "$\equiv\ true$" weg, also z.B. "$even(x) \rightarrow \neg odd(x)$" anstatt "$even(x) \equiv true \rightarrow \neg odd(x) \equiv true$".

Hinweis

- Im allgemeinen kann es in einer prädikatenlogischen Sprache neben einem Gleichheitsprädikatensymbol (hier: $\equiv$) noch weitere Prädikatensymbole geben.
- *Hier Sonderfall:* Es gibt nur ein 2-stelliges Prädikatensymbol, nämlich $\equiv$ (=> **Kapitel 9**, Folien 6 und 7).

Was bedeuten Formeln?

- Deutung von Formeln durch Σ-Algebren, d.h. die Worte aus $\mathcal{F}(\Sigma, \mathcal{V})$ bekommen eine *Bedeutung*.
- Was bedeutet $\equiv$?
- Wie deuten wir die Junktoren $\neg$ und $\wedge$?
- Wie deuten wir den Allquantor $\forall$?

Definition 8 *(Deutung von Formeln)*

Sei $A = (\mathcal{A}, \alpha)$ *eine* Σ*-Algebra (bzgl. Sortenmenge* $\mathcal{S}$*) und* $\mathfrak{a} : \mathcal{V} \rightarrow_{\mathcal{S}} \mathcal{A}$ *eine* A*-Variablenbelegung. Dann ist das Paar* $I = (A, \mathfrak{a})$ *eine* Σ***-Interpretation***. *Die* Σ*-Interpretation* I ***erfüllt eine Formel*** $\phi \in \mathcal{F}(\Sigma, \mathcal{V})$ *(kurz:* $I \models_{Alg(\Sigma)} \phi$*) gdw.*

(1) $\phi = t_1 \equiv t_2$ und $\mathfrak{a}(t_1) = \mathfrak{a}(t_2)$

(2) $\phi = \neg\phi'$ und $I \not\models_{Alg(\Sigma)} \phi'$

(3) $\phi = \phi_1 \wedge \phi_2$ und sowohl $I \models_{Alg(\Sigma)} \phi_1$ als auch $I \models_{Alg(\Sigma)} \phi_2$

(4) $\phi = \forall x{:}s\ \phi'$ und $I[x/a] \models_{Alg(\Sigma)} \phi'$ für jedes $a \in \mathcal{A}_s$
(wobei $I[x/a] := (A, \mathfrak{a}[x/a])$).

- $\phi \in \mathcal{F}(\Sigma, \mathcal{V})$ ist **erfüllbar** gdw. $I \models_{Alg(\Sigma)} \phi$ für **eine** Σ-Interpretation I.
- Eine Σ-Interpretation I **erfüllt eine Formelmenge** $\Phi \subset \mathcal{F}(\Sigma, \mathcal{V})$ (kurz: $I \models_{Alg(\Sigma)} \Phi$) gdw. $I \models_{Alg(\Sigma)} \phi$ für alle $\phi \in \Phi$.
- $\Phi \subset \mathcal{F}(\Sigma, \mathcal{V})$ ist **erfüllbar** gdw. $I \models_{Alg(\Sigma)} \Phi$ für eine Σ-Interpretation I.
- $\phi \in \mathcal{F}(\Sigma, \mathcal{V})$ ist **allgemeingültig** gdw. $I \models_{Alg(\Sigma)} \phi$ für **jede** Σ-Interpretation I.

Bemerkung 5 Also:

- $\equiv$ deuten wir als *Identität*.
- $\neg$ und $\wedge$ deuten wir (wie üblich) als "*nicht*" und "*und*".
- $\forall$ deuten wir (wie üblich) als "*für alle*".

Schreibweise: Für eine Σ-Algebra A steht $A[x/a]$ für jede Σ-Interpretation $(A, \mathfrak{a}[x/a])$ mit beliebiger A-Variablenbelegung $\mathfrak{a}$.

Definition 9 *(Semantische Folgerung)*

- Eine Formel $\phi \in \mathcal{F}(\Sigma, \mathcal{V})$ **folgt** aus einer Formelmenge $\Phi \subset \mathcal{F}(\Sigma, \mathcal{V})$ (kurz: $\Phi \models_{\mathcal{F}(\Sigma,\mathcal{V})} \phi$) gdw. $I \models_{Alg(\Sigma)} \phi$ für **alle** Σ-Interpretationen I mit $I \models_{Alg(\Sigma)} \Phi$. Wir schreiben $\models_{\mathcal{F}(\Sigma,\mathcal{V})} \phi$, falls $\Phi = \emptyset$.
- $\Phi^{\models_{\mathcal{F}(\Sigma,\mathcal{V})}}$, definiert als $\{\phi \in \mathcal{F}(\Sigma, \mathcal{V}) \mid \Phi \models_{\mathcal{F}(\Sigma,\mathcal{V})} \phi\}$, ist die Menge aller **Folgerungen** von $\Phi \subset \mathcal{F}(\Sigma, \mathcal{V})$.

Satz 10 $\phi \in \mathcal{F}(\Sigma, \mathcal{V})$ *ist allgemeingültig gdw.* $\models_{\mathcal{F}(\Sigma,\mathcal{V})} \phi$*, d.h.* $\emptyset^{\models_{\mathcal{F}(\Sigma,\mathcal{V})}}$ *ist die Menge aller allgemeingültigen Formeln.*

Satz 11 $\models_{\mathcal{F}(\Sigma,\mathcal{V})}$ *ist* ***monoton****, d.h. für alle* $\Phi, \Psi \subset \mathcal{F}(\Sigma, \mathcal{V})$ *gilt*

$$\Phi \subset \Psi \Rightarrow \Phi^{\models_{\mathcal{F}(\Sigma,\mathcal{V})}} \subset \Psi^{\models_{\mathcal{F}(\Sigma,\mathcal{V})}}$$

Konvention:

Im Folgenden kurz "$\models$" anstatt "$\models_{Alg(\Sigma)}$" und "$\models_{\mathcal{F}(\Sigma,\mathcal{V})}$". D.h. "overloading" von "$\models$". Aber aus dem Kontext ist immer ersichtlich, ob "$\models_{Alg(\Sigma)}$" oder "$\models_{\mathcal{F}(\Sigma,\mathcal{V})}$" gemeint ist.

Welche Formeln werden durch eine Σ-Algebra erfüllt ?

Definition 12 *(Geschlossene Formeln, $\mathcal{F}_g(\Sigma, \mathcal{V})$)*

*Für $\phi \in \mathcal{F}(\Sigma, \mathcal{V})$ sei $\mathcal{V}(\phi) \subset \mathcal{V}$ die Menge aller Variablensymbole in der Formel ϕ. $\mathcal{V}_f(\phi) \subset \mathcal{V}(\phi)$ ist die Menge aller **freien** Variablen von ϕ, definiert durch:*

(1) $\mathcal{V}_f(\phi) := \mathcal{V}(\phi)$, falls $\phi \in \mathcal{AT}(\Sigma, \mathcal{V})$

(2) $\mathcal{V}_f(\neg\phi) := \mathcal{V}_f(\phi)$

(3) $\mathcal{V}_f(\phi_1 \wedge \phi_2) := \mathcal{V}_f(\phi_1) \cup \mathcal{V}_f(\phi_2)$

(4) $\mathcal{V}_f(\forall x{:}s\ \phi) := \mathcal{V}_f(\phi) \setminus \{x\}$

*$\phi \in \mathcal{F}(\Sigma, \mathcal{V})$ ist **geschlossen** (kurz: $\phi \in \mathcal{F}_g(\Sigma, \mathcal{V})$) gdw. $\mathcal{V}_f(\phi) = \emptyset$.*

Definition 13 *(Theorie einer Σ-Algebra)*

- Eine Σ-Algebra $A = (\mathcal{A}, \alpha)$ erfüllt eine **geschlossene** Formel $\phi \in \mathcal{F}_g(\Sigma, \mathcal{V})$ (kurz: $A \models \phi$) gdw. $I \models \phi$ für **jede** Σ-Interpretation $I = (A, \mathfrak{a})$ gilt.
- Die **Theorie** $Th(A)$ einer Σ-Algebra A ist definiert durch

$$Th(A) := \{\phi \in \mathcal{F}_g(\Sigma, \mathcal{V}) \mid A \models \phi\} \ .$$

Bemerkung 6

- Solange wir nur geschlossene Formeln betrachten, spielt die Variablenbelegung einer Interpretation keine Rolle. Daher $A \models \phi$ anstatt $(A, \mathfrak{a}) \models \phi$.
- Die *Theorie* $Th(A)$ einer Σ-Algebra A enthält alle *wahren* Aussagen über die Funktionen und Elemente (der Trägermengen) der Algebra, die wir mittels geschlossener Formeln aus $\mathcal{F}_g(\Sigma, \mathcal{V})$ aufschreiben können.
- *Später:* Σ-Algebra $\mathcal{M}_P$ wird definiert durch die Funktionen, die durch die Prozeduren eines Programms P berechnet werden. Theorie $Th(\mathcal{M}_P)$ enthält dann alle wahren Aussagen über P. *Verifikation* einer Programmaussage ϕ bedeutet dann: Feststellen, ob $\phi \in Th(\mathcal{M}_P)$ gilt.

Satz 14 *Seien A, B Σ-Algebren und sei $\phi \in \mathcal{F}_g(\Sigma, \mathcal{V})$. Dann gilt:*

(1) $\emptyset^{\models} \subset Th(A)$, d.h. *jede* Theorie enthält *alle allgemeingültigen* Formeln,

(2) $Th(A) \neq \emptyset$, d.h. *keine* Theorie ist *leer*,

(3) $\phi \notin Th(A)$ oder $\neg\phi \notin Th(A)$, d.h. $Th(A)$ ist *konsistent* (*widerspruchsfrei*),

(4) $\phi \in Th(A)$ oder $\neg\phi \in Th(A)$, d.h. $Th(A)$ ist *vollständig*,

(5) $Th(A) \cup \{\phi\}$ ist erfüllbar gdw. $\phi \in Th(A)$,

(6) $Th(A) = Th(B)$ gdw. $Th(A) \subseteq Th(B)$.

Beispiel 7 *Sei* $\mathcal{S} := \{bool, nat\}$, $\Sigma_{bool} := \{true, false\}$, $\Sigma_{nat} := \{O\}$, $\Sigma_{nat,nat} := \{succ\}$, $\Sigma_{nat,nat,nat} := \{plus\}$, $\Sigma_{nat,nat,bool} := \{gt\}$ *und* $PLUS_> = (\mathcal{A}, \alpha)$ *definiert durch*

(1) $\mathcal{A}_{nat} = \mathbb{N}$, $\mathcal{A}_{bool} = \{\top, \bot\}$,

(2) $\alpha_{true} := \top$, $\alpha_{false} := \bot$, $\alpha_O := 0$, $\alpha_{succ}(n) := n + 1$,

(3) $\alpha_{plus}(n, m) := n + m$,

(4) $\alpha_{gt}(n, m) := \begin{cases} \top & \text{, falls } n > m \\ \bot & \text{, falls } n \leq m \end{cases}$,

Dann enthält $Th(PLUS)$ *alle wahren Aussagen über* 0*, Nachfolgerfunktion und Addition, die mittels Formeln aus* $\mathcal{F}_g(\Sigma, \mathcal{V})$ ***ausgedrückt werden können****. Z.B.*

- $[\forall x{:}nat.\ \neg succ(x) \equiv O] \in Th(PLUS_>)$,
- $[\forall x{:}nat.\ gt(succ(x), O) \equiv true] \in Th(PLUS_>)$,
- $[\exists x{:}nat.\ \neg plus(x, x) \equiv x] \in Th(PLUS_>)$,
- $[\forall x, y{:}nat.\ plus(x, y) \equiv plus(y, x)] \in Th(PLUS_>)$,
- $[\forall x, y, z{:}nat.\ plus(x, plus(y, z)) \equiv plus(plus(x, y), z)] \in Th(PLUS_>)$,
- $[\forall x, y, z{:}nat.\ plus(x, y) \equiv plus(x, z) \rightarrow y \equiv z] \in Th(PLUS_>)$,
- $\ldots$

Aber:

Beispielsweise kann

$$\left|Range_{gt}(n)\right| < \infty \textit{ für alle } n \in \mathbb{N}$$

mit $Range_{gt}(n) := \{m \in \mathbb{N} \mid \alpha_{gt}(n, m) = \top\}$ *nicht mittels Formeln aus* $\mathcal{F}_g\,(\Sigma, \mathcal{V})$ *ausgedrückt werden* ***!***

Bezug zu und Erinnerung an *FGdI 2*:

- Signaturen werden in FGdI 2 durch S anstatt durch Σ bezeichnet.
- Eine S-*Algebra* über einer Signatur S für Funktionssymbole (und Prädikatssymbole bzw. Relationssymbole) wird in FGdI 2 als S-*Struktur* bezeichnet.
- In FGdI 2 wird $\mathtt{f}^A$ anstatt $\alpha_{\mathtt{f}}$ für die Deutungsfunktion eines Funktionssymbols $\mathtt{f}$ geschrieben:
  - *Schreibweise* hier: $A(\mathtt{f}(\mathtt{t}_1, \ldots, \mathtt{t}_n)) := \alpha_{\mathtt{f}}(A(\mathtt{t}_1), \ldots, A(\mathtt{t}_n))$
  - *Schreibweise* FGdI 2: $(\mathtt{f}(\mathtt{t}_1, \ldots, \mathtt{t}_n))^A := \mathtt{f}^A(\mathtt{t}_1^A, \ldots, \mathtt{t}_n^A)$
- A-*Variablenbelegungen* $\mathfrak{a} : \mathcal{V} \to \mathcal{A}$ werden in FGdI 2 kurz *Belegungen* genannt und i.A. durch "β" bezeichnet:
  - *Schreibweise* hier: $I(\mathtt{x}) := \mathfrak{a}(\mathtt{x})$ und $I(\mathtt{f}(\mathtt{t}_1, \ldots, \mathtt{t}_n)) := \alpha_{\mathtt{f}}(I(\mathtt{t}_1), \ldots, I(\mathtt{t}_n))$
  - *Schreibweise* FGdI 2: $\mathtt{x}^I := \mathfrak{a}(\mathtt{x})$ und $(\mathtt{f}(\mathtt{t}_1, \ldots, \mathtt{t}_n))^I := \mathtt{f}^I(\mathtt{t}_1^I, \ldots, \mathtt{t}_n^I)$

5 Eigenschaften der Symbolischen Auswertung

Satz 15 *(Korrektheit der Symbolischen Auswertung)*

Seien

- AX_P die Menge aller *Axiome* der Datentyp- und Prozedurdefinitionen eines $\mathcal{L}$-Programms P,
- $Lem_{verified}$ die Menge aller *bewiesenen* Lemmata von P,
- $AX_{Lem_{verified}}$ die Menge der Axiome der Lemmata aus $Lem_{verified}$,
- $\mathcal{E}_P := \{\forall x, y{:}\tau.\ eq_\tau(x,y) \equiv true \leftrightarrow x \equiv y | \text{Typ } \tau \neq bool \text{ ist in } P \text{ definiert}\}$
  $\cup \{\forall x{:}bool.\ x \equiv true \vee x \equiv false\}$,
- $seq = \langle H, IH \Vdash b \rangle$ eine *HPL*-Sequenz in P, und
- b ein boolscher Term in P, so daß b' aus b durch Anwendung endlich vieler Schritte der symbolischen Auswertung entsteht (kurz: $b \vdash_{AX_P, Lem_{verified}, seq} b'$).
  Dann gilt

$$AX_P \cup AX_{Lem_{verified}} \cup \mathcal{E}_P$$
$$\models$$
$$\forall \ldots \left[\bigwedge_{h \in H} h \equiv \texttt{true} \wedge \bigwedge_{\forall \ldots ih \in IH} \forall \ldots (ih \equiv \texttt{true}) \rightarrow b \equiv b' \right]. \quad \blacksquare$$

Bedeutet:

- In *allen Modellen* von $AX_P \cup AX_{Lem_{verified}} \cup \mathcal{E}_P$ werden b und b' bzgl. H und IH *identisch gedeutet*
- *Damit:* Symbolische Auswertung führt Gleichheitsumformungen durch.
- **Sonderfall** $b \vdash_{AX_P, Lem_{verified}, seq} true$:
  Dann gilt

$$AX_P \cup AX_{Lem_{verified}} \cup \mathcal{E}_P$$
$$\models$$
$$\forall \ldots \left[\bigwedge_{h \in H} h \equiv \texttt{true} \wedge \bigwedge_{\forall \ldots ih \in IH} \forall \ldots (ih \equiv \texttt{true}) \rightarrow b \equiv true \right]$$

  d.h. b ist in *jedem* Modell von $AX_P \cup AX_{Lem_{verified}} \cup \mathcal{E}_P$ bzgl. H und IH *wahr*!

Aber: Symbolische Auswertung ist *unvollständig*, d.h. es gibt Programme P und boolsche Terme b mit $AX_P \cup AX_{Lem_{verified}} \cup \mathcal{E}_P \models \forall \ldots [\ldots \longrightarrow b \equiv true]$ aber $b \not\vdash_{AX_P, Lem_{verified}, seq} true$.

Beispiel 8 *(Unvollständigkeit der Symbolischen Auswertung)*

- Definiere Prozedur "$<$" (entsprechend Prozedur "$>$")
- `lemma` $<$ `is transitive` $<=$ $\forall$ `x, y, z : nat if`$\{$`x` $<$ `y, if`$\{$`y` $<$ `z, x` $<$ `z, true`$\}$`, true`$\}$ wird mittels *Verify* automatisch bewiesen (Induktionsbeweis)
- `lemma x` $<$ `succ(x)` $<=$ $\forall$ `x : nat x` $<$ `succ(x)` wird mittels *Verify* automatisch bewiesen (Induktionsbeweis)
- `lemma x` $<$ `succ(succ(x))` $<=$ $\forall$ `x : nat x` $<$ `succ(succ(x))` wird mittels *Verify* automatisch bewiesen
  - **Aber:** Auch hier *Induktionsbeweis* obwohl *nicht erforderlich* – `lemma x` $<$ `succ(succ(x))` kann allein mittels `lemma` $<$ `is transitive` und zweimaliger Anwendung von `lemma x` $<$ `succ(x)` ohne Induktion bewiesen werden :

(1)	`x` $<$ `succ(x)`	`lemma x` $<$ `succ(x)`
(2)	`succ(x)` $<$ `succ(succ(x))`	Instanz von `lemma x` $<$ `succ(x)`
(3)	`x` $<$ `succ(succ(x))`	mit (1), (2) & `lemma` $<$ `is transitive`

Warum Unvollständigkeit ?

- $\Phi \models \varphi$ ist *rekursiv aufzählbar* (= *semi-entscheidbar*), d.h. man kann einen automatischen Beweiser ATP bauen, der (immer) "$\Phi \models \varphi$" *feststellt*, *falls* dies tatsächlich *gilt* (=> "ATP ist vollständig")
- $\Phi \not\models \varphi$ ist *nicht rekursiv aufzählbar*, d.h. man kann *keinen* automatischen Beweiser ATP bauen, der (immer) "$\Phi \not\models \varphi$" *feststellt*, falls $\Phi \models \varphi$ *nicht gilt*
- *Vorraussetzung*: "ATP ist korrekt", d.h. es gilt $\Phi \models \varphi$, falls ATP "$\Phi \models \varphi$" feststellt
- **Konsequenz**: Für $\Phi \not\models \varphi$ kann ein Beweisversuch von $\Phi \models \varphi$ mittels ATP zu einem *unendlichen Lauf* von ATP führen.
- **Anders gesagt**: Eine *allgemeingültige* Aussage kann *immer maschinell bewiesen* werden, es kann i.A. jedoch *nicht maschinell festgestellt* werden, ob eine Aussage *nicht allgemeingültig* ist.
- **Damit:**
  Implementiert man eine *vollständige* Symbolische Auswertung, so *terminiert* diese i.A. *nicht* für b falls $AX_P \cup AX_{Lem_{verified}} \cup \mathcal{E}_P \not\models \forall \ldots [\ldots \rightarrow b \equiv true]$ **!**

- **Konsequenz:**
  Wegen der *Unvollständigkeit* der *symbolischen Auswertung* sind die *HPL*-Regeln

  6. *Case Analysis*
  7. *Use Lemma*
  8. *Unfold Procedure*
  9. *Apply Equation*
  10. *Purge*
  11. *Induction*
  12. *Insert Induction Hypotheses*
  13. *Insert Hypotheses*
  14. *Move Hypotheses*
  15. *Delete Hypotheses*

  (=> **Kapitel 3**) erforderlich, um einen Beweis gegebenenfalls durch Benutzer-interaktion weiterzuführen.

Warum muß man sich überhaupt um $\not\models$ kümmern ?

(1) I.a. sind Programme fehlerhaft und Behauptungen über Programme falsch:
- Ein System muß auch für fehlerhafte Programme und falsche Behauptungen sinnvolle Ergebnisse liefern, die es einem Benutzer ermöglichen Fehler zu erkennen, um diese dann zu korrigieren

(2) In Kapitel 12 definieren wir, wann ein Lemma *wahr* ist. Hier nur kurze vereinfachte Vorschau (Details => **Kapitel 12**):
- Der Wahrheitsbegriff für Lemmata basiert auf sogenannten *Standardmodellen* $\mathcal{M}_P$ (= *erzeugte* Algebren) von AX_P (es gilt also $\mathcal{M}_P \models AX_P$)
- Ein Lemma lem ist *wahr* gdw. lem im *Standardmodell* $\mathcal{M}_P$ *gilt* (d.h. $\mathcal{M}_P \models lem$ bzw. $lem \in Th(\mathcal{M}_P)$)
- **Also:** Wir sprechen über die Gültigkeit von Aussagen in einem *bestimmten Modell* von AX_P und *nicht* in *allen Modellen* (= Allgemeingültigkeit) von AX_P
- **Offensichtlich:** $AX_P \models lem \curvearrowright \mathcal{M}_P \models lem$
  - *Damit:* Herleitungen der Symbolischen Auswertung sind auch im Standardmodell korrekt
- **Aber:** $\mathcal{M}_P \models lem \not\curvearrowright AX_P \models lem$ **!** D.h. Es gibt Programme P und *wahre* Behauptungen lem über P, so daß lem *nicht* aus AX_P *folgt*.

Beispiel 9 *(Assoziativität von* `plus` *ist nicht allgemeingültig)*

- *Programm*:

$$P = \langle\ \texttt{structure bool <= ...}, \texttt{structure nat <= ...},\ \texttt{function plus(x, y : nat) : nat <= ...}\ \rangle$$

- *Axiome* (vgl. **Kapitel 9**):

$$\begin{aligned}
AX_P = \{\ & \forall x, y{:}\texttt{bool}\ \texttt{if}_{bool}\{\texttt{true}, x, y\} \equiv x \\
& \forall x, y{:}\texttt{bool}\ \texttt{if}_{bool}\{\texttt{false}, x, y\}, y \\
& \texttt{eq}_{\texttt{nat}}(\texttt{0},\texttt{0}) \equiv \texttt{true} \\
& \forall x{:}\texttt{nat}\ \texttt{pred}(\texttt{succ}(x)) \equiv x \\
& \forall x, y{:}\texttt{nat}\ \texttt{eq}_{nat}(\texttt{succ}(x), \texttt{succ}(y)) \equiv \texttt{eq}_{nat}(x, y) \\
& \forall x{:}\texttt{nat}\ \texttt{eq}_{nat}(\texttt{0}, \texttt{succ}(x)) \equiv \texttt{false} \\
& \forall x{:}\texttt{nat}\ \texttt{eq}_{nat}(\texttt{succ}(x), \texttt{0}) \equiv \texttt{false} \\
& \forall x, y{:}\texttt{nat}\ \texttt{if}_{nat}\{\texttt{true}, x, y\} \equiv x \\
& \forall x, y{:}\texttt{nat}\ \texttt{if}_{nat}\{\texttt{false}, x, y\} \equiv y \\
& \forall x{:}\texttt{nat}\ \texttt{?0}(x) \equiv \texttt{eq}_{nat}(x, \texttt{0}) \\
& \forall x{:}\texttt{nat}\ \texttt{?succ}(x) \equiv \texttt{eq}_{nat}(x, \texttt{succ}(\texttt{pred}(x))) \\
& \forall x, y{:}\texttt{nat}\ \texttt{plus}(x, y) \equiv \texttt{if}_{\texttt{nat}}\{\texttt{eq}_{\texttt{nat}}(x, \texttt{0}), \texttt{0}, \\
& \qquad\qquad \texttt{succ}(\texttt{plus}(\texttt{pred}(x), y))\}\ \}
\end{aligned}$$

- *Signatur*:

$$\begin{aligned}\Sigma_P = \{\ & \texttt{true}, \texttt{false} :\rightarrow \texttt{bool} \\ & \texttt{if}_{bool} : \texttt{bool} \times \texttt{bool} \times \texttt{bool} \rightarrow \texttt{bool} \\ & \texttt{0} :\rightarrow \texttt{nat} \\ & \texttt{succ}, \texttt{pred} : \texttt{nat} \rightarrow \texttt{nat} \\ & \texttt{eq}_{\texttt{nat}} : \texttt{nat} \times \texttt{nat} \rightarrow \texttt{bool} \\ & \texttt{if}_{nat} : \texttt{bool} \times \texttt{nat} \times \texttt{nat} \rightarrow \texttt{nat} \\ & \texttt{?0}, \texttt{?succ} : \texttt{nat} \rightarrow \texttt{bool} \\ & \texttt{plus} : \texttt{nat} \times \texttt{nat} \rightarrow \texttt{nat}\ \ \}\end{aligned}$$

- Σ_P-Algebra $A = (\mathcal{A}, \alpha)$ mit

  - $\mathcal{A}_{\texttt{bool}} = \{\top, \bot\}$
  - $\mathcal{A}_{\texttt{nat}} = \mathbb{N} \cup \left\{z + \frac{1}{2} | z \in \mathbb{Z}\right\}$, also
    $\mathcal{A}_{\texttt{nat}} = \{\ldots, -3.5, -2.5, -1.5, -0.5, 0, +0.5, +1, +1.5, +2, +2.5, \ldots\}$
  - $\alpha_{\texttt{true}} := \top$ sowie $\alpha_{\texttt{false}} := \bot$
  - $\alpha_{\texttt{if}_s}(\top, a, b) := a$ für $s \in \{\texttt{bool}, \texttt{nat}\}$ und alle $a, b \in \mathcal{A}_s$
  - $\alpha_{\texttt{if}_s}(\bot, a, b) := b$ für $s \in \{\texttt{bool}, \texttt{nat}\}$ und alle $a, b \in \mathcal{A}_s$
  - $\alpha_{\texttt{0}} := 0$ sowie $\alpha_{\texttt{succ}}(a) := a + 1$ für alle $a \in \mathcal{A}_{\texttt{nat}}$
  - $\alpha_{\texttt{pred}}(0) := 0$ sowie $\alpha_{\texttt{pred}}(a) := a - 1$ für alle $a \in \mathcal{A}_{\texttt{nat}} \setminus \{0\}$

- $\alpha_{\mathtt{eq}_{\mathtt{nat}}}(a,a) := \top$ sowie $\alpha_{\mathtt{eq}_{\mathtt{nat}}}(a,b) := \bot$ für alle $a,b \in \mathcal{A}_{\mathtt{nat}}$ mit $a \neq b$
- $\alpha_{?\mathtt{0}}(0) := \top$ sowie $\alpha_{?\mathtt{0}}(a) := \bot$ für alle $a \in \mathcal{A}_{\mathtt{nat}} \setminus \{0\}$
- $\alpha_{?\mathtt{succ}}(0) := \bot$ sowie $\alpha_{?\mathtt{succ}}(a) := \top$ für alle $a \in \mathcal{A}_{\mathtt{nat}} \setminus \{0\}$
- $\alpha_{\mathtt{plus}}(a,b) := (a-b) + \frac{1}{2}$ für alle $a,b \in \mathcal{A}_{\mathtt{nat}} \setminus \mathbb{N}$
- $\alpha_{\mathtt{plus}}(a,b) := a + b$ für alle $a,b \in \mathcal{A}_{\mathtt{nat}}$ mit $a \in \mathbb{N}$ oder $b \in \mathbb{N}$

- *Es gilt* $A \models AX_P$ (=> Übung)
- *Aber* (**!**): Für $\varphi := \forall x,y,z\,\mathtt{:nat}$
  $\mathtt{eq}_{nat}\mathtt{(plus(}x\mathtt{,plus(}y\mathtt{,}z\mathtt{))\mathtt{,}plus(plus(}x\mathtt{,}y\mathtt{)\mathtt{,}}z\mathtt{)} \equiv \mathtt{true}$
  gilt $A \not\models \varphi$ denn

$$\begin{aligned} & A\,[x/9.5, y/4.5, z/3.5]\,(\mathtt{plus}(x, \mathtt{plus}(y,z))) \\ = \;& \alpha_{\mathtt{plus}}(9.5, \alpha_{\mathtt{plus}}(4.5, 3.5)) \\ = \;& \alpha_{\mathtt{plus}}(9.5, 1.5) \\ = \;& 8.5 \\ \neq \;& 2.5 \\ = \;& \alpha_{\mathtt{plus}}(5.5, 3.5) \\ = \;& \alpha_{\mathtt{plus}}(\alpha_{\mathtt{plus}}(9.5, 4, 5), 3.5) \\ = \;& A\,[x/9.5, y/4.5, z/3, 5]\,(\mathtt{plus}(\mathtt{plus}(x,y),z)) \end{aligned}$$

- und damit gilt $AX_P \not\models \varphi$ **!** ■

Damit: Eine *vollständige* symbolische Auswertung muß angesetzt auf lem *nicht* notwendigerweise *terminieren* (***obwohl*** lem ***wahr ist***)

Konsequenz:

Die Terminierung der Symbolischen Auswertung muß erzwungen werden !

- **Naiver Ansatz**: Die *Symbolische Auswertung* eines boolschen Terms b wird nach n Schritten mit Ergebnis b' *gestoppt*.
- **Aber**: Ergebnis b' im allgemeinen für weitere Arbeit *unbrauchbar* **!**
- **Nützlicher Ansatz**: *Heuristik*, die ausgehend von Programm P und Beweisziel b entscheidet, wann Auswertungsversuch mit *brauchbarem* (!) Ergebnis b' *abgebrochen* werden soll.
- **Aber**: Die *Heuristik* **muß** in gewissen Fällen **irren**, denn andernfalls wäre $\Phi \not\models \varphi$ ja doch *rekursiv aufzählbar*.

Fazit: Um *nicht-terminierende* symbolische Auswertung zu *verhindern*,

muß symbolische Auswertung *unvollständig* sein (s. Beispiel 8) **!**