Übersicht

- Was sind induktive Theoreme?
  - Syntax: Signatur, Terme, . . .
  - Semantik: Was bedeutet $s = t$ gilt?

Übersicht

- Was sind induktive Theoreme?
  - Syntax: Signatur, Terme, . . .
  - Semantik: Was bedeutet $s = t$ gilt?
- “Klassische” Ansätze
  - Explizite Induktion
  - Coversets
  - Konsistenzprüfung

Übersicht

- Was sind induktive Theoreme?
  - Syntax: Signatur, Terme, . . .
  - Semantik: Was bedeutet $s = t$ gilt?
- “Klassische” Ansätze
  - Explizite Induktion
  - Coversets
  - Konsistenzprüfung
- Der Ansatz von **QuodLibet**
  - zulässige Spezifikation
  - Induktion mit Gewichten
  - Inferenzregeln / Taktiken

Syntax

- Eine Signatur $\mathrm{sig} = (S, F, \tau)$ besteht aus einer Menge S von *Sorten* und einer Menge F von *Funktionssymbolen*. Die *Stelligkeitsfunktion* τ ordnet jedem Funktionssymbol $f \in F$ eine Stelligkeit $f : s_1, \ldots, s_n \rightarrow s$ zu.

Syntax

- Eine Signatur $\mathrm{sig} = (S, F, \tau)$ besteht aus einer Menge S von *Sorten* und einer Menge F von *Funktionssymbolen*. Die *Stelligkeitsfunktion* τ ordnet jedem Funktionssymbol $f \in F$ eine Stelligkeit $f : s_1, \ldots, s_n \to s$ zu.
- Terme $t \in \mathrm{Term}(F, V)$ werden rekursiv definiert.

Syntax

- Eine Signatur $\mathrm{sig} = (S, F, \tau)$ besteht aus einer Menge S von *Sorten* und einer Menge F von *Funktionssymbolen*. Die *Stelligkeitsfunktion* τ ordnet jedem Funktionssymbol $f \in F$ eine Stelligkeit $f : s_1, \ldots, s_n \to s$ zu.
- Terme $t \in \mathrm{Term}(F, V)$ werden rekursiv definiert.
- Eine Spezifikation $\mathrm{Spec} = (\mathrm{sig}, E)$ besteht aus einer Signatur sig und einer Menge E von (positiv) bedingten Gleichungen.

Semantik

- Eine sig-Algebra $\mathcal{A} = ((\mathcal{A}_s)_{s \in S}, (f^{\mathcal{A}})_{f \in F})$ ordnet jeder Sorte $s \in S$ eine nichtleere *Trägermenge* $\mathcal{A}_s$ und jedem Funktionssymbol f eine (sortentreue) Funktion $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$ zu.

Semantik

- Eine sig-Algebra $\mathcal{A} = ((\mathcal{A}_s)_{s \in S}, (f^{\mathcal{A}})_{f \in F})$ ordnet jeder Sorte $s \in S$ eine nichtleere *Trägermenge* $\mathcal{A}_s$ und jedem Funktionssymbol f eine (sortentreue) Funktion $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$ zu.
- $s = t$ gilt in $\mathcal{A}$, falls für jede *Variablenbelegung* φ gilt: $\varphi_{\mathcal{A}}(s) = \varphi_{\mathcal{A}}(t)$.

Semantik

- Eine sig-Algebra $\mathcal{A} = ((\mathcal{A}_s)_{s \in S}, (f^{\mathcal{A}})_{f \in F})$ ordnet jeder Sorte $s \in S$ eine nichtleere *Trägermenge* $\mathcal{A}_s$ und jedem Funktionssymbol f eine (sortentreue) Funktion $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$ zu.
- $s = t$ gilt in $\mathcal{A}$, falls für jede *Variablenbelegung* φ gilt: $\varphi_{\mathcal{A}}(s) = \varphi_{\mathcal{A}}(t)$.
- $s = t$ gilt in $\mathrm{Spec} = (\mathrm{sig}, E)$, wenn $s = t$ in jeder Algebra $\mathcal{A}$ gilt, in der E gilt.

Semantik

- Eine sig-Algebra $\mathcal{A} = ((\mathcal{A}_s)_{s \in S}, (f^{\mathcal{A}})_{f \in F})$ ordnet jeder Sorte $s \in S$ eine nichtleere *Trägermenge* $\mathcal{A}_s$ und jedem Funktionssymbol f eine (sortentreue) Funktion $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$ zu.
- $s = t$ gilt in $\mathcal{A}$, falls für jede *Variablenbelegung* φ gilt: $\varphi_{\mathcal{A}}(s) = \varphi_{\mathcal{A}}(t)$.
- $s = t$ gilt in $\mathrm{Spec} = (\mathrm{sig}, E)$, wenn $s = t$ im *initialen Modell* $\mathcal{I}(\mathrm{Spec})$ von Spec gilt.

Semantik

- Eine sig-Algebra $\mathcal{A} = ((\mathcal{A}_s)_{s \in S}, (f^{\mathcal{A}})_{f \in F})$ ordnet jeder Sorte $s \in S$ eine nichtleere *Trägermenge* $\mathcal{A}_s$ und jedem Funktionssymbol f eine (sortentreue) Funktion $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$ zu.
- $s = t$ gilt in $\mathcal{A}$, falls für jede *Variablenbelegung* φ gilt: $\varphi_{\mathcal{A}}(s) = \varphi_{\mathcal{A}}(t)$.
- $s = t$ gilt in $\mathrm{Spec} = (\mathrm{sig}, E)$, wenn $s = t$ im *initialen Modell* $\mathcal{I}(\mathrm{Spec})$ von Spec gilt.
- $\mathcal{I}(\mathrm{Spec}) = \mathrm{Term}(F)/ =_E$.

Semantik

- Eine sig-Algebra $\mathcal{A} = ((\mathcal{A}_s)_{s \in S}, (f^{\mathcal{A}})_{f \in F})$ ordnet jeder Sorte $s \in S$ eine nichtleere *Trägermenge* $\mathcal{A}_s$ und jedem Funktionssymbol f eine (sortentreue) Funktion $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$ zu.
- $s = t$ gilt in $\mathcal{A}$, falls für jede *Variablenbelegung* φ gilt: $\varphi_{\mathcal{A}}(s) = \varphi_{\mathcal{A}}(t)$.
- $s = t$ gilt in $\mathrm{Spec} = (\mathrm{sig}, E)$, wenn $s = t$ im *initialen Modell* $\mathcal{I}(\mathrm{Spec})$ von Spec gilt.
- $\mathcal{I}(\mathrm{Spec}) = \mathrm{Term}(F)/ =_E$.
- Junk? Confusion?

Semantik

- Eine sig-Algebra $\mathcal{A} = ((\mathcal{A}_s)_{s \in S}, (f^{\mathcal{A}})_{f \in F})$ ordnet jeder Sorte $s \in S$ eine nichtleere *Trägermenge* $\mathcal{A}_s$ und jedem Funktionssymbol f eine (sortentreue) Funktion $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \cdots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$ zu.
- $s = t$ gilt in $\mathcal{A}$, falls für jede *Variablenbelegung* φ gilt: $\varphi_{\mathcal{A}}(s) = \varphi_{\mathcal{A}}(t)$.
- $s = t$ gilt in $\mathrm{Spec} = (\mathrm{sig}, E)$, wenn $s = t$ im *initialen Modell* $\mathcal{I}(\mathrm{Spec})$ von Spec gilt.
- $\mathcal{I}(\mathrm{Spec}) = \mathrm{Term}(F)/ =_E$.
- Junk? Confusion? NEIN!

Abstrakte Datentypen und das initiale Modell (1)

Spec1:

0: Nat

s: Nat $\longrightarrow$ Nat

+: Nat $\times$ Nat $\longrightarrow$ Nat

+(x,0) = x

+(x,s(y)) = s(+(x,y))

Abstrakte Datentypen und das initiale Modell (1)

```
Spec1:
  0: Nat
  s: Nat → Nat
  +: Nat × Nat → Nat

  +(x,0) = x
  +(x,s(y)) = s(+(x,y))
```

- Betrachte die drei Modelle $\mathcal{A}_1 = (\mathbb{Z}, \ldots)$, $\mathcal{A}_2 = (\mathbb{Z}_5, \ldots)$ und $\mathcal{A}_3 = (\mathbb{N}, \ldots)$ der Spezifikation.

Abstrakte Datentypen und das initiale Modell (1)

```
Spec1:
  0: Nat
  s: Nat → Nat
  +: Nat × Nat → Nat

  +(x,0) = x
  +(x,s(y)) = s(+(x,y))
```

- Betrachte die drei Modelle $\mathcal{A}_1 = (\mathbb{Z}, \ldots)$, $\mathcal{A}_2 = (\mathbb{Z}_5, \ldots)$ und $\mathcal{A}_3 = (\mathbb{N}, \ldots)$ der Spezifikation.
- In $\mathcal{A}_1$ gibt es Junk-Elemente.

Abstrakte Datentypen und das initiale Modell (1)

```
Spec1:
  0: Nat
  s: Nat → Nat
  +: Nat × Nat → Nat

  +(x,0) = x
  +(x,s(y)) = s(+(x,y))
```

- Betrachte die drei Modelle $\mathcal{A}_1 = (\mathbb{Z}, \ldots)$, $\mathcal{A}_2 = (\mathbb{Z}_5, \ldots)$ und $\mathcal{A}_3 = (\mathbb{N}, \ldots)$ der Spezifikation.
- In $\mathcal{A}_1$ gibt es Junk-Elemente.
- In $\mathcal{A}_2$ gibt es Confusion.

Abstrakte Datentypen und das initiale Modell (1)

```
Spec1:
  0: Nat
  s: Nat → Nat
  +: Nat × Nat → Nat

  +(x,0) = x
  +(x,s(y)) = s(+(x,y))
```

- Betrachte die drei Modelle $\mathcal{A}_1 = (\mathbb{Z}, \ldots)$, $\mathcal{A}_2 = (\mathbb{Z}_5, \ldots)$ und $\mathcal{A}_3 = (\mathbb{N}, \ldots)$ der Spezifikation.
- In $\mathcal{A}_1$ gibt es Junk-Elemente.
- In $\mathcal{A}_2$ gibt es Confusion.
- $\mathcal{A}_3$ ist “das gemeinte” Modell.

Abstrakte Datentypen und das initiale Modell (2)

Spec2:

$\mathsf{emptyTree}: \mathsf{Tree}$

$\mathsf{tree}: \mathsf{Int} \times \mathsf{Tree} \times \mathsf{Tree} \rightarrow \mathsf{Tree}$

$\mathsf{size}: \mathsf{Tree} \rightarrow \mathsf{Int}$

$\mathsf{size}(\mathsf{emptyTree}) = 0$

$\mathsf{size}(\mathsf{tree}(i, t_1, t_2)) = \mathsf{size}(t_1) + \mathsf{size}(t_2) + 1$

Abstrakte Datentypen und das initiale Modell (2)

Spec2:

emptyTree: Tree

tree: Int $\times$ Tree $\times$ Tree $\rightarrow$ Tree

size: Tree $\rightarrow$ Int

size(emptyTree) = 0

size(tree(i,t_1,t_2)) = size(t_1) + size(t_2) + 1

- Gilt $\text{size}(t) = 0 \Rightarrow t = \text{emptyTree}$?

Abstrakte Datentypen und das initiale Modell (2)

Spec2:

emptyTree: Tree

tree: Int $\times$ Tree $\times$ Tree $\rightarrow$ Tree

size: Tree $\rightarrow$ Int

size(emptyTree) = 0

size(tree(i,t_1,t_2)) = size(t_1) + size(t_2) + 1

- Gilt $\mathrm{size}(t) = 0 \Rightarrow t = \mathrm{emptyTree}$?
- Nicht in jedem Modell von (sig, E),

Abstrakte Datentypen und das initiale Modell (2)

Spec2:

emptyTree: Tree

tree: Int $\times$ Tree $\times$ Tree $\rightarrow$ Tree

size: Tree $\rightarrow$ Int

size(emptyTree) = 0

size(tree(i,t_1,t_2)) = size(t_1) + size(t_2) + 1

- Gilt $\mathrm{size}(t) = 0 \Rightarrow t = \mathrm{emptyTree}$?
- Nicht in jedem Modell von (sig, E), aber in jedem *Standardmodell*.

Abstrakte Datentypen und das initiale Modell (2)

Spec2:

emptyTree: Tree

tree: Int $\times$ Tree $\times$ Tree $\rightarrow$ Tree

size: Tree $\rightarrow$ Int

size(emptyTree) = 0

size(tree(i,t_1,t_2)) = size(t_1) + size(t_2) + 1

- Gilt $\mathrm{size}(t) = 0 \Rightarrow t = \mathrm{emptyTree}$?
- Nicht in jedem Modell von (sig, E), aber in jedem *Standardmodell*.
- Gilt $\mathrm{tree}(x, t_1, t_2) = t_2$?

Abstrakte Datentypen und das initiale Modell (2)

Spec2:

emptyTree: Tree

tree: Int $\times$ Tree $\times$ Tree $\rightarrow$ Tree

size: Tree $\rightarrow$ Int

size(emptyTree) = 0

size(tree(i,t_1,t_2)) = size(t_1) + size(t_2) + 1

- Gilt $\mathrm{size}(t) = 0 \Rightarrow t = \mathrm{emptyTree}$?
- Nicht in jedem Modell von (sig, E), aber in jedem *Standardmodell*.
- Gilt $\mathrm{tree}(x, t_1, t_2) = t_2$?
- In manchen Modellen schon,

Abstrakte Datentypen und das initiale Modell (2)

Spec2:

emptyTree: Tree

tree: Int $\times$ Tree $\times$ Tree $\rightarrow$ Tree

size: Tree $\rightarrow$ Int

size(emptyTree) = 0

size(tree(i,t_1,t_2)) = size(t_1) + size(t_2) + 1

- Gilt $\mathrm{size}(t) = 0 \Rightarrow t = \mathrm{emptyTree}$?
- Nicht in jedem Modell von (sig, E), aber in jedem *Standardmodell*.
- Gilt $\mathrm{tree}(x, t_1, t_2) = t_2$?
- In manchen Modellen schon, aber nicht im initialen Modell.

Abstrakte Datentypen und das initiale Modell (2)

Spec2:

emptyTree: Tree

tree: Int $\times$ Tree $\times$ Tree $\rightarrow$ Tree

size: Tree $\rightarrow$ Int

size(emptyTree) = 0

size(tree(i,t_1,t_2)) = size(t_1) + size(t_2) + 1

- Gilt $\text{size}(t) = 0 \Rightarrow t = \text{emptyTree}$?
- Nicht in jedem Modell von (sig, E), aber in jedem *Standardmodell*.
- Gilt $\text{tree}(x, t_1, t_2) = t_2$?
- In manchen Modellen schon, aber nicht im initialen Modell.
- $\Rightarrow$ Die initiale Theorie ist interessant!!

Explizite Induktion

- Idee: $s = t \in ITh(\mathrm{Spec})$ gdw $\sigma(s) = \sigma(t) \in Th(\mathrm{Spec})$ für alle Grundsubstitutionen σ

Explizite Induktion

- Idee: $s = t \in ITh(\mathrm{Spec})$ gdw $\sigma(s) = \sigma(t) \in Th(\mathrm{Spec})$ für alle Grundsubstitutionen σ
- Aufteilung der Funktionssymbole in Konstruktoren und definierte Funktionssymbole.

Explizite Induktion

- Idee: $s = t \in ITh(\mathrm{Spec})$ gdw $\sigma(s) = \sigma(t) \in Th(\mathrm{Spec})$ für alle Grundsubstitutionen σ
- Aufteilung der Funktionssymbole in Konstruktoren und definierte Funktionssymbole.
- Beweis über den Aufbau von $\mathrm{Term}(C)$ (Aufbau von $\sigma(x_1), \ldots, \sigma(x_n)$)

Explizite Induktion

- Idee: $s = t \in ITh(\mathrm{Spec})$ gdw $\sigma(s) = \sigma(t) \in Th(\mathrm{Spec})$ für alle Grundsubstitutionen σ
- Aufteilung der Funktionssymbole in Konstruktoren und definierte Funktionssymbole.
- Beweis über den Aufbau von $\mathrm{Term}(C)$ (Aufbau von $\sigma(x_1), \ldots, \sigma(x_n)$)
- Formalisierter Ansatz: Coverset (Induktionsschema)

Bsp. Coverset: Ackermann-Péter-Robinson-Funktion

Spec3:

$a(0, y) = s(y)$

$a(s(x), 0) = a(x, s(0))$

$a(s(x), s(y)) = a(x, a(s(x), y))$

$leq(0, x) = true$

$leq(s(x), 0) = false$

$leq(s(x), s(y)) = leq(x, y)$

Bsp. Coverset: Ackermann-Péter-Robinson-Funktion

Spec3:

$\mathtt{a}(0,y) = \mathtt{s}(y)$

$\mathtt{a}(\mathtt{s}(x),0) = \mathtt{a}(x,\mathtt{s}(0))$

$\mathtt{a}(\mathtt{s}(x),\mathtt{s}(y)) = \mathtt{a}(x,\mathtt{a}(\mathtt{s}(x),y))$

$\mathtt{leq}(0,x) = \mathtt{true}$

$\mathtt{leq}(\mathtt{s}(x),0) = \mathtt{false}$

$\mathtt{leq}(\mathtt{s}(x),\mathtt{s}(y)) = \mathtt{leq}(x,y)$

- Um $\mathrm{leq}(+(x,y),\ \mathrm{a}(x,y))$ zu zeigen (implizit mit “= true”), reicht es

zu zeigen.

Bsp. Coverset: Ackermann-Péter-Robinson-Funktion

Spec3:

$$\mathtt{a}(0,y) = \mathtt{s}(y)$$
$$\mathtt{a}(\mathtt{s}(x),0) = \mathtt{a}(x,\mathtt{s}(0))$$
$$\mathtt{a}(\mathtt{s}(x),\mathtt{s}(y)) = \mathtt{a}(x,\mathtt{a}(\mathtt{s}(x),y))$$
$$\mathtt{leq}(0,x) = \mathtt{true}$$
$$\mathtt{leq}(\mathtt{s}(x),0) = \mathtt{false}$$
$$\mathtt{leq}(\mathtt{s}(x),\mathtt{s}(y)) = \mathtt{leq}(x,y)$$

- Um $\mathrm{leq}(+(x,y),\ \mathrm{a}(x,y))$ zu zeigen (implizit mit “= true”), reicht es
  - $\mathrm{leq}(+(0,y),\ \mathrm{a}(0,y))$

zu zeigen.

Bsp. Coverset: Ackermann-Péter-Robinson-Funktion

Spec3:

$$\begin{aligned}
&\mathrm{a}(0,y) = \mathrm{s}(y)\\
&\mathrm{a}(\mathrm{s}(x),0) = \mathrm{a}(x,\mathrm{s}(0))\\
&\mathrm{a}(\mathrm{s}(x),\mathrm{s}(y)) = \mathrm{a}(x,\mathrm{a}(\mathrm{s}(x),y))\\
&\mathrm{leq}(0,x) = \mathrm{true}\\
&\mathrm{leq}(\mathrm{s}(x),0) = \mathrm{false}\\
&\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(y)) = \mathrm{leq}(x,y)
\end{aligned}$$

- Um $\mathrm{leq}(+(x,y),\ \mathrm{a}(x,y))$ zu zeigen (implizit mit “= true”), reicht es
  - $\mathrm{leq}(+(0,y),\ \mathrm{a}(0,y))$
  - $\mathrm{leq}(+(x,\mathrm{s}(0)),\ \mathrm{a}(x,\mathrm{s}(0))) \Rightarrow \mathrm{leq}(+(\mathrm{s}(x),0),\ \mathrm{a}(\mathrm{s}(x),0))$

zu zeigen.

Bsp. Coverset: Ackermann-Péter-Robinson-Funktion

Spec3:

$\mathtt{a}(0,y) = \mathtt{s}(y)$

$\mathtt{a}(\mathtt{s}(x),0) = \mathtt{a}(x,\mathtt{s}(0))$

$\mathtt{a}(\mathtt{s}(x),\mathtt{s}(y)) = \mathtt{a}(x,\mathtt{a}(\mathtt{s}(x),y))$

$\mathtt{leq}(0,x) = \mathtt{true}$

$\mathtt{leq}(\mathtt{s}(x),0) = \mathtt{false}$

$\mathtt{leq}(\mathtt{s}(x),\mathtt{s}(y)) = \mathtt{leq}(x,y)$

- Um $\mathrm{leq}(+(x,y),\ \mathrm{a}(x,y))$ zu zeigen (implizit mit “= true”), reicht es
  - $\mathrm{leq}(+(0,y),\ \mathrm{a}(0,y))$
  - $\mathrm{leq}(+(x,\mathrm{s}(0)),\ \mathrm{a}(x,\mathrm{s}(0))) \Rightarrow \mathrm{leq}(+(\mathrm{s}(x),0),\ \mathrm{a}(\mathrm{s}(x),0))$
  - $\mathrm{leq}(+(\mathrm{s}(x),y),\ \mathrm{a}(\mathrm{s}(x),y)),$
    $\mathrm{leq}(+(x,\mathrm{a}(\mathrm{s}(x),y)),\ \mathrm{a}(x,\mathrm{a}(\mathrm{s}(x),y)))$
    $\Rightarrow \mathrm{leq}(+(\mathrm{s}(x),\mathrm{s}(y)),\ \mathrm{a}(\mathrm{s}(x),\mathrm{s}(y)))$

  zu zeigen.

Beweis durch Konsistenzprüfung

- Idee: $G \subseteq ITh(E)$ gdw $=_{E \cup G} \; = \; =_E$ auf Grundtermen

Beweis durch Konsistenzprüfung

- Idee: $G \subseteq ITh(E)$ gdw $=_{E \cup G} \; = \; =_E$ auf Grundtermen
- Reduktionstechniken (Vervollständigung, Simplifikation)

Beweis durch Konsistenzprüfung

- Idee: $G \subseteq ITh(E)$ gdw $=_{E \cup G} \; = \; =_E$ auf Grundtermen
- Reduktionstechniken (Vervollständigung, Simplifikation) bis (In-)Konsistenz der Ziele offensichtlich ist.

Beweis durch Konsistenzprüfung

- Idee: $G \subseteq ITh(E)$ gdw $=_{E \cup G} \; = \; =_E$ auf Grundtermen
- Reduktionstechniken (Vervollständigung, Simplifikation) bis (In-)Konsistenz der Ziele offensichtlich ist.
- Verkleinern von Gegenbeispielen

Beweis durch Konsistenzprüfung

- Idee: $G \subseteq ITh(E)$ gdw $=_{E \cup G} \; = \; =_E$ auf Grundtermen
- Reduktionstechniken (Vervollständigung, Simplifikation) bis (In-)Konsistenz der Ziele offensichtlich ist.
- Verkleinern von Gegenbeispielen
- Nachteile:
  - nur unbedingte Regelsysteme
  - nur terminierende Regelsysteme
  - Terminierung des Laufes?

Bsp. Konsistenzprüfung: $3 \geq z$

- Wir versuchen $\mathrm{leq}(z, \mathrm{s}(\mathrm{s}(\mathrm{s}(0)))) = \mathrm{true}$ zu zeigen.

Bsp. Konsistenzprüfung: $3 \geq z$

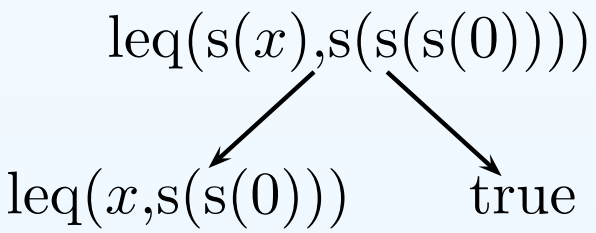
- Wir versuchen $\mathrm{leq}(z, \mathrm{s}(\mathrm{s}(\mathrm{s}(0)))) = \mathrm{true}$ zu zeigen.
- Gegenbeispiel: $\sigma_1 = \{\ z \leftarrow \mathrm{s}(\mathrm{s}(\mathrm{s}(\mathrm{s}(0))))\ \}$

Bsp. Konsistenzprüfung: $3 \geq z$

- Wir versuchen $\mathrm{leq}(z, \mathrm{s}(\mathrm{s}(\mathrm{s}(0)))) = \mathrm{true}$ zu zeigen.
- Gegenbeispiel: $\sigma_1 = \{\ z \leftarrow \mathrm{s}(\mathrm{s}(\mathrm{s}(\mathrm{s}(0))))\ \}$
- Überlappung mit $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(y)) = \mathrm{leq}(x, y)$ liefert

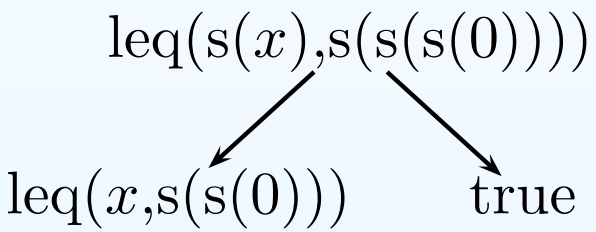
Bsp. Konsistenzprüfung: $3 \geq z$

- Wir versuchen $\mathrm{leq}(z,\, \mathrm{s}(\mathrm{s}(\mathrm{s}(0)))) = \mathrm{true}$ zu zeigen.
- Gegenbeispiel: $\sigma_1 = \{\ z \leftarrow \mathrm{s}(\mathrm{s}(\mathrm{s}(\mathrm{s}(0))))\ \}$
- Überlappung mit $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(y)) = \mathrm{leq}(x, y)$ liefert



Bsp. Konsistenzprüfung: $3 \geq z$

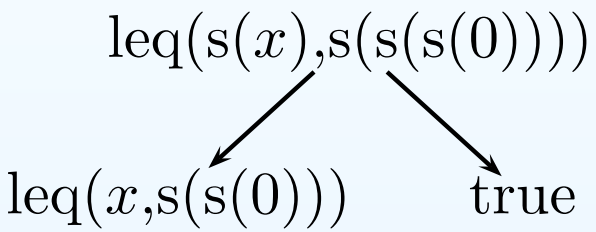
- Wir versuchen $\mathrm{leq}(z,\, \mathrm{s}(\mathrm{s}(\mathrm{s}(0)))) = \mathrm{true}$ zu zeigen.
- Gegenbeispiel: $\sigma_1 = \{\ z \leftarrow \mathrm{s}(\mathrm{s}(\mathrm{s}(\mathrm{s}(0))))\ \}$
- Überlappung mit $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(y)) = \mathrm{leq}(x, y)$ liefert



- neues Ziel: $\mathrm{leq}(x,\, \mathrm{s}(\mathrm{s}(0))) = \mathrm{true}$

Bsp. Konsistenzprüfung: $3 \geq z$

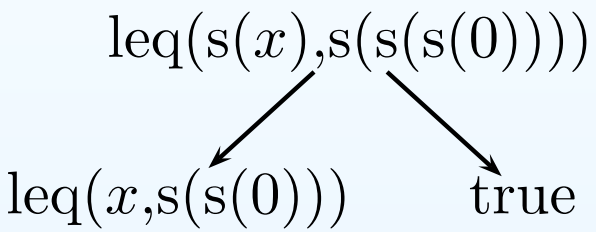
- Wir versuchen $\mathrm{leq}(z,\, \mathrm{s}(\mathrm{s}(\mathrm{s}(0)))) = \mathrm{true}$ zu zeigen.
- Gegenbeispiel: $\sigma_1 = \{\ z \leftarrow \mathrm{s}(\mathrm{s}(\mathrm{s}(\mathrm{s}(0))))\ \}$
- Überlappung mit $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(y)) = \mathrm{leq}(x, y)$ liefert



- neues Ziel: $\mathrm{leq}(x,\, \mathrm{s}(\mathrm{s}(0))) = \mathrm{true}$
- Gegenbeispiel: $\sigma_2 = \{\ x \leftarrow \mathrm{s}(\mathrm{s}(\mathrm{s}(0)))\ \}$

Bsp. Konsistenzprüfung: $3 \geq z$

- Wir versuchen $\mathrm{leq}(z,\, \mathrm{s}(\mathrm{s}(\mathrm{s}(0)))) = \mathrm{true}$ zu zeigen.
- Gegenbeispiel: $\sigma_1 = \{\ z \leftarrow \mathrm{s}(\mathrm{s}(\mathrm{s}(\mathrm{s}(0))))\ \}$
- Überlappung mit $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(y)) = \mathrm{leq}(x, y)$ liefert



- neues Ziel: $\mathrm{leq}(x,\, \mathrm{s}(\mathrm{s}(0))) = \mathrm{true}$
- Gegenbeispiel: $\sigma_2 = \{\ x \leftarrow \mathrm{s}(\mathrm{s}(\mathrm{s}(0)))\ \}$
- Durch die Inferenzregel wurde ein kleineres Gegenbeispiel eingeführt. Dies wird fortgesetzt, bis $\mathrm{true} = \mathrm{false}$ abgeleitet wird. Dies ist “offensichtlich inkonsistent”.

Auftretende Probleme

- Modularität:

Auftretende Probleme

- Modularität:
  - Man betrachte ein ausgezeichnetet Modell, also gilt eine Gleichheit, oder sie gilt nicht.

Auftretende Probleme

- Modularität:
  - Man betrachte ein ausgezeichnetet Modell, also gilt eine Gleichheit, oder sie gilt nicht.
  - Durch Hinzunahme von harmlosen Gleichungen kann das Modell kollabieren.

Auftretende Probleme

- Modularität:
  - Man betrachte ein ausgezeichnetet Modell, also gilt eine Gleichheit, oder sie gilt nicht.
  - Durch Hinzunahme von harmlosen Gleichungen kann das Modell kollabieren.
- negativ bedingte Gleichungen in der Spezifikation?

Auftretende Probleme

- Modularität:
  - Man betrachte ein ausgezeichnetet Modell, also gilt eine Gleichheit, oder sie gilt nicht.
  - Durch Hinzunahme von harmlosen Gleichungen kann das Modell kollabieren.
- negativ bedingte Gleichungen in der Spezifikation?
  - Probleme bei der Definition von $=_E$

Auftretende Probleme

- Modularität:
  - Man betrachte ein ausgezeichnetet Modell, also gilt eine Gleichheit, oder sie gilt nicht.
  - Durch Hinzunahme von harmlosen Gleichungen kann das Modell kollabieren.
- negativ bedingte Gleichungen in der Spezifikation?
  - Probleme bei der Definition von $=_E$
  - Probleme bei der Operationalisierung

Der Ansatz von **QuodLibet**

- Erweiterung von Syntax / Semantik

Der Ansatz von **QuodLibet**

- Erweiterung von Syntax / Semantik
- zulässige Spezifikationen

Der Ansatz von **QuodLibet**

- Erweiterung von Syntax / Semantik
- zulässige Spezifikationen
- Induktion mit Gewichten

Der Ansatz von **QuodLibet**

- Erweiterung von Syntax / Semantik
- zulässige Spezifikationen
- Induktion mit Gewichten
- Verkleinern von Gegenbeispielen

Der Ansatz von **QuodLibet**

- Erweiterung von Syntax / Semantik
- zulässige Spezifikationen
- Induktion mit Gewichten
- Verkleinern von Gegenbeispielen
- Inferenzregeln

Der Ansatz von **QuodLibet**

- Erweiterung von Syntax / Semantik
- zulässige Spezifikationen
- Induktion mit Gewichten
- Verkleinern von Gegenbeispielen
- Inferenzregeln
- Interaktives Beweisen / Taktiken

Der Ansatz von **QuodLibet**

- Erweiterung von Syntax / Semantik
- zulässige Spezifikationen
- Induktion mit Gewichten
- Verkleinern von Gegenbeispielen
- Inferenzregeln
- Interaktives Beweisen / Taktiken

Siehe Systemvorführung ;o)

Erweiterung von Syntax / Semantik (1)

- Betrachte nicht *ein* Modell, aber auch nicht alle.

Erweiterung von Syntax / Semantik (1)

- Betrachte nicht *ein* Modell, aber auch nicht alle.
- Junk und Confusion?

Erweiterung von Syntax / Semantik (1)

- Betrachte nicht *ein* Modell, aber auch nicht alle.
- Junk und Confusion?
  - Benutze Konstruktorvariablen, um über den termerzeugten Anteil zu reden

Erweiterung von Syntax / Semantik (1)

- Betrachte nicht *ein* Modell, aber auch nicht alle.
- Junk und Confusion?
  - Benutze Konstruktorvariablen, um über den termerzeugten Anteil zu reden
  - Nur freie Konstruktoren zugelassen

Erweiterung von Syntax / Semantik (1)

- Betrachte nicht *ein* Modell, aber auch nicht alle.
- Junk und Confusion?
  - Benutze Konstruktorvariablen, um über den termerzeugten Anteil zu reden
  - Nur freie Konstruktoren zugelassen
  - Nur orthogonale Regelsysteme zugelassen

Erweiterung von Syntax / Semantik (1)

- Betrachte nicht *ein* Modell, aber auch nicht alle.
- Junk und Confusion?
  - Benutze Konstruktorvariablen, um über den termerzeugten Anteil zu reden
  - Nur freie Konstruktoren zugelassen
  - Nur orthogonale Regelsysteme zugelassen
- Die Ungleichheit wird bewacht und “konstruktiv” ausgewertet: $\mathcal{A}$ erfüllt $s \neq t$, gdw $s^{\mathcal{A}}, t^{\mathcal{A}} \in \mathcal{A}^C$ und $s^{\mathcal{A}} \neq t^{\mathcal{A}}$

Erweiterung von Syntax / Semantik (1)

- Betrachte nicht *ein* Modell, aber auch nicht alle.
- Junk und Confusion?
  - Benutze Konstruktorvariablen, um über den termerzeugten Anteil zu reden
  - Nur freie Konstruktoren zugelassen
  - Nur orthogonale Regelsysteme zugelassen
- Die Ungleichheit wird bewacht und “konstruktiv” ausgewertet: $\mathcal{A}$ erfüllt $s \neq t$, gdw $s^{\mathcal{A}}, t^{\mathcal{A}} \in \mathcal{A}^C$ und $s^{\mathcal{A}} \neq t^{\mathcal{A}}$
- Erweiterung durch Standardprädikat def
  $\mathrm{def}(t)$ gilt unter $(\mathcal{A}, \varphi)$ gdw $\varphi(t) \in \mathcal{A}^C$.

Erweiterung von Syntax / Semantik (1)

- Betrachte nicht *ein* Modell, aber auch nicht alle.
- Junk und Confusion?
  - Benutze Konstruktorvariablen, um über den termerzeugten Anteil zu reden
  - Nur freie Konstruktoren zugelassen
  - Nur orthogonale Regelsysteme zugelassen
- Die Ungleichheit wird bewacht und “konstruktiv” ausgewertet: $\mathcal{A}$ erfüllt $s \neq t$, gdw $s^{\mathcal{A}}, t^{\mathcal{A}} \in \mathcal{A}^C$ und $s^{\mathcal{A}} \neq t^{\mathcal{A}}$
- Erweiterung durch Standardprädikat def $\mathrm{def}(t)$ gilt unter $(\mathcal{A}, \varphi)$ gdw $\varphi(t) \in \mathcal{A}^C$.
- weiter Literale der Art $t_1 < t_2$

Erweiterung von Syntax / Semantik (2)

- $\mathcal{A}$ ist *Datenmodell* von Spec, wenn $\mathcal{A}$ ein Modell von Spec ist und $t_1^{\mathcal{A}} = t_2^{\mathcal{A}} \Rightarrow t_1 \equiv t_2$ für alle $t_1, t_2 \in \mathrm{Term}(C)$ gilt.

Erweiterung von Syntax / Semantik (2)

- $\mathcal{A}$ ist *Datenmodell* von Spec, wenn $\mathcal{A}$ ein Modell von Spec ist und $t_1^{\mathcal{A}} = t_2^{\mathcal{A}} \Rightarrow t_1 \equiv t_2$ für alle $t_1, t_2 \in \mathrm{Term}(C)$ gilt.
- $C \equiv C_1 \vee \cdots \vee C_n$ gilt in Spec, wenn C in allen Datenmodellen von Spec gilt.

Erweiterung von Syntax / Semantik (2)

- $\mathcal{A}$ ist *Datenmodell* von Spec, wenn $\mathcal{A}$ ein Modell von Spec ist und $t_1^{\mathcal{A}} = t_2^{\mathcal{A}} \Rightarrow t_1 \equiv t_2$ für alle $t_1, t_2 \in \mathrm{Term}(C)$ gilt.
- $C \equiv C_1 \vee \cdots \vee C_n$ gilt in Spec, wenn C in allen Datenmodellen von Spec gilt.
- Was hat das mit der induktiven Theorie zu tun?

Erweiterung von Syntax / Semantik (2)

- $\mathcal{A}$ ist *Datenmodell* von Spec, wenn $\mathcal{A}$ ein Modell von Spec ist und $t_1^{\mathcal{A}} = t_2^{\mathcal{A}} \Rightarrow t_1 \equiv t_2$ für alle $t_1, t_2 \in \mathrm{Term}(C)$ gilt.
- $C \equiv C_1 \vee \cdots \vee C_n$ gilt in Spec, wenn C in allen Datenmodellen von Spec gilt.
- Was hat das mit der induktiven Theorie zu tun? Viel!

Inferenzregeln (1)

- Eine Inferenzregel von **QuodLibet** hat die Form:

$$\frac{\langle \Gamma; \omega \rangle}{\langle \Gamma_1, \omega_1 \rangle, \ldots, \langle \Gamma_n; \omega_n \rangle}, \quad \text{mit} \quad \langle \Pi_1, \hat{\omega}_1 \rangle^{U_1}, \ldots, \langle \Pi_k, \hat{\omega}_k \rangle^{U_k}$$

falls “Anwendungsbedingungen”

Inferenzregeln (1)

- Eine Inferenzregel von **QuodLibet** hat die Form:

$$\frac{\langle \Gamma; \omega \rangle}{\langle \Gamma_1, \omega_1 \rangle, \ldots, \langle \Gamma_n; \omega_n \rangle}, \quad \text{mit} \quad \langle \Pi_1, \hat{\omega}_1 \rangle^{U_1}, \ldots, \langle \Pi_k, \hat{\omega}_k \rangle^{U_k}$$

falls “Anwendungsbedingungen”

- Intention: Um Γ zu zeigen, reicht es $\Gamma_1, \ldots, \Gamma_n$ zu zeigen.

Inferenzregeln (1)

- Eine Inferenzregel von **QuodLibet** hat die Form:

$$\frac{\langle \Gamma; \omega \rangle}{\langle \Gamma_1, \omega_1 \rangle, \ldots, \langle \Gamma_n; \omega_n \rangle}, \quad \text{mit} \quad \langle \Pi_1, \hat{\omega}_1 \rangle^{U_1}, \ldots, \langle \Pi_k, \hat{\omega}_k \rangle^{U_k}$$

falls “Anwendungsbedingungen”

- Intention: Um Γ zu zeigen, reicht es $\Gamma_1, \ldots, \Gamma_n$ zu zeigen.
- Beispiele:

Inferenzregeln (1)

- Eine Inferenzregel von **QuodLibet** hat die Form:

$$\frac{\langle \Gamma; \omega \rangle}{\langle \Gamma_1, \omega_1 \rangle, \ldots, \langle \Gamma_n; \omega_n \rangle}, \quad \text{mit} \quad \langle \Pi_1, \hat{\omega}_1 \rangle^{U_1}, \ldots, \langle \Pi_k, \hat{\omega}_k \rangle^{U_k}$$

falls “Anwendungsbedingungen”

- Intention: Um Γ zu zeigen, reicht es $\Gamma_1, \ldots, \Gamma_n$ zu zeigen.
- Beispiele:
  - einfache Fallunterscheidung: $\dfrac{\langle \Gamma; \omega \rangle}{\langle \overline{\lambda}, \Gamma; \omega \rangle, \langle \lambda, \Gamma; \omega \rangle}$

Inferenzregeln (1)

- Eine Inferenzregel von **QuodLibet** hat die Form:

$$\frac{\langle \Gamma; \omega \rangle}{\langle \Gamma_1, \omega_1 \rangle, \ldots, \langle \Gamma_n; \omega_n \rangle}, \quad \text{mit} \quad \langle \Pi_1, \hat{\omega}_1 \rangle^{U_1}, \ldots, \langle \Pi_k, \hat{\omega}_k \rangle^{U_k}$$

falls “Anwendungsbedingungen”

- Intention: Um Γ zu zeigen, reicht es $\Gamma_1, \ldots, \Gamma_n$ zu zeigen.
- Beispiele:
  - einfache Fallunterscheidung: $\dfrac{\langle \Gamma; \omega \rangle}{\langle \overline{\lambda}, \Gamma; \omega \rangle, \langle \lambda, \Gamma; \omega \rangle}$
  - Fallunterscheidung:

$$\frac{\langle \Gamma; \omega \rangle}{\langle \overline{\lambda_1}, \Gamma; \omega \rangle, \langle \overline{\lambda_2}, \lambda_1 \Gamma; \omega \rangle, \ldots, \langle \overline{\lambda_n}, \lambda_{n-1}, \ldots, \lambda_1, \Gamma; \omega \rangle, \langle \lambda_n, \ldots, \lambda_1, \Gamma; \omega \rangle}$$

Inferenzregeln (1)

- Eine Inferenzregel von **QuodLibet** hat die Form:

$$\frac{\langle \Gamma; \omega \rangle}{\langle \Gamma_1, \omega_1 \rangle, \ldots, \langle \Gamma_n; \omega_n \rangle}, \quad \text{mit} \quad \langle \Pi_1, \hat{\omega}_1 \rangle^{U_1}, \ldots, \langle \Pi_k, \hat{\omega}_k \rangle^{U_k}$$

falls “Anwendungsbedingungen”

- Intention: Um Γ zu zeigen, reicht es $\Gamma_1, \ldots, \Gamma_n$ zu zeigen.
- Beispiele:
  - einfache Fallunterscheidung: $\dfrac{\langle \Gamma; \omega \rangle}{\langle \overline{\lambda}, \Gamma; \omega \rangle, \langle \lambda, \Gamma; \omega \rangle}$
  - Fallunterscheidung:

$$\frac{\langle \Gamma; \omega \rangle}{\langle \overline{\lambda_1}, \Gamma; \omega \rangle, \langle \overline{\lambda_2}, \lambda_1 \Gamma; \omega \rangle, \ldots, \langle \overline{\lambda_n}, \lambda_{n-1}, \ldots, \lambda_1, \Gamma; \omega \rangle, \langle \lambda_n, \ldots, \lambda_1, \Gamma; \omega \rangle}$$

  - Subsumption: $\dfrac{\langle \Gamma; \omega \rangle}{}$

    mit $\langle \Pi; \hat{\omega} \rangle^{\mathcal{L}}$, falls $\sigma(\Pi) \cup \mathrm{DefCond}(\sigma, \Pi) \subseteq \Gamma$.

Inferenzregeln (2)

- Induktionsschritt: $\dfrac{\langle \Gamma; \omega \rangle}{\langle \sigma(\hat{\omega}) < \omega, \Gamma; \omega \rangle}$

  mit $\langle \Pi; \hat{\omega} \rangle^{\mathcal{I}}$, falls $\sigma(\Pi) \cup \mathrm{DefCond}(\sigma, \Pi) \subseteq \Gamma$.

Inferenzregeln (2)

- Induktionsschritt: $\dfrac{\langle \Gamma; \omega \rangle}{\langle \sigma(\hat{\omega}) < \omega, \Gamma; \omega \rangle}$

  mit $\langle \Pi; \hat{\omega} \rangle^{\mathcal{I}}$, falls $\sigma(\Pi) \cup \mathrm{DefCond}(\sigma, \Pi) \subseteq \Gamma$.

- Constant Rewriting: $\dfrac{\langle \Gamma_1, \lambda[p \leftarrow t_1], \Gamma_2; \omega \rangle}{\langle \Gamma_1, \lambda[p \leftarrow t_2], \Gamma_2; \omega \rangle}$,

  falls $t_1 \neq t_2 \in \Gamma_1 \cup \Gamma_2$ (oder $t_2 \neq t_1$)

Inferenzregeln (2)

- Induktionsschritt: $\dfrac{\langle \Gamma; \omega \rangle}{\langle \sigma(\hat{\omega}) < \omega, \Gamma; \omega \rangle}$

  mit $\langle \Pi; \hat{\omega} \rangle^{\mathcal{I}}$, falls $\sigma(\Pi) \cup \mathrm{DefCond}(\sigma, \Pi) \subseteq \Gamma$.

- Constant Rewriting: $\dfrac{\langle \Gamma_1, \lambda[p \leftarrow t_1], \Gamma_2; \omega \rangle}{\langle \Gamma_1, \lambda[p \leftarrow t_2], \Gamma_2; \omega \rangle}$,

  falls $t_1 \neq t_2 \in \Gamma_1 \cup \Gamma_2$ (oder $t_2 \neq t_1$)

- Fallunterscheidung nach dem Aufbau von $\mathrm{Term}(C)$:

  $$\frac{\langle \Gamma; \omega \rangle}{\langle \sigma_1(\Gamma); \sigma_1(\omega) \rangle, \ldots, \langle \sigma_n(\Gamma); \sigma_n(\omega) \rangle},$$

  falls $\sigma_1, \ldots, \sigma_n$ eine Überdeckung zu Γ ist

Inferenzregeln (2)

- Induktionsschritt: $\dfrac{\langle \Gamma; \omega \rangle}{\langle \sigma(\hat{\omega}) < \omega, \Gamma; \omega \rangle}$

  mit $\langle \Pi; \hat{\omega} \rangle^{\mathcal{I}}$, falls $\sigma(\Pi) \cup \mathrm{DefCond}(\sigma, \Pi) \subseteq \Gamma$.

- Constant Rewriting: $\dfrac{\langle \Gamma_1, \lambda[p \leftarrow t_1], \Gamma_2; \omega \rangle}{\langle \Gamma_1, \lambda[p \leftarrow t_2], \Gamma_2; \omega \rangle}$,

  falls $t_1 \neq t_2 \in \Gamma_1 \cup \Gamma_2$ (oder $t_2 \neq t_1$)

- Fallunterscheidung nach dem Aufbau von $\mathrm{Term}(C)$:

  $$\frac{\langle \Gamma; \omega \rangle}{\langle \sigma_1(\Gamma); \sigma_1(\omega) \rangle, \ldots, \langle \sigma_n(\Gamma); \sigma_n(\omega) \rangle},$$

  falls $\sigma_1, \ldots, \sigma_n$ eine Überdeckung zu Γ ist

- Weitere Inferenzregeln, z.B. zum Beweisen von $\mathrm{def}(t)$ oder $\omega < \hat{\omega}$

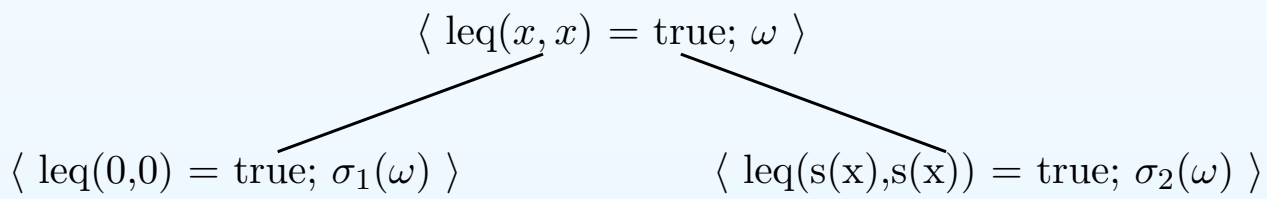
Beispiel: leq(x, x)

- Wir wollen $\mathrm{leq}(x, x) = \mathrm{true}$ beweisen.

$$\langle\ \mathrm{leq}(x, x) = \mathrm{true};\ \omega\ \rangle$$

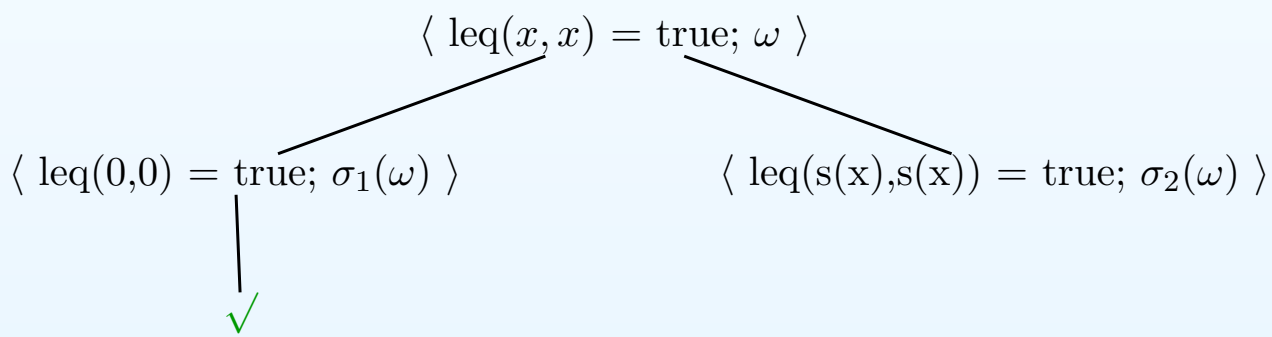
Beispiel: leq(x, x)

- Wir wollen $\mathrm{leq}(x, x) = \mathrm{true}$ beweisen.
- Fallunterscheidung nach $\{\sigma_1 = \{x \leftarrow 0\}, \sigma_2 = \{x \leftarrow s(x)\}\}$



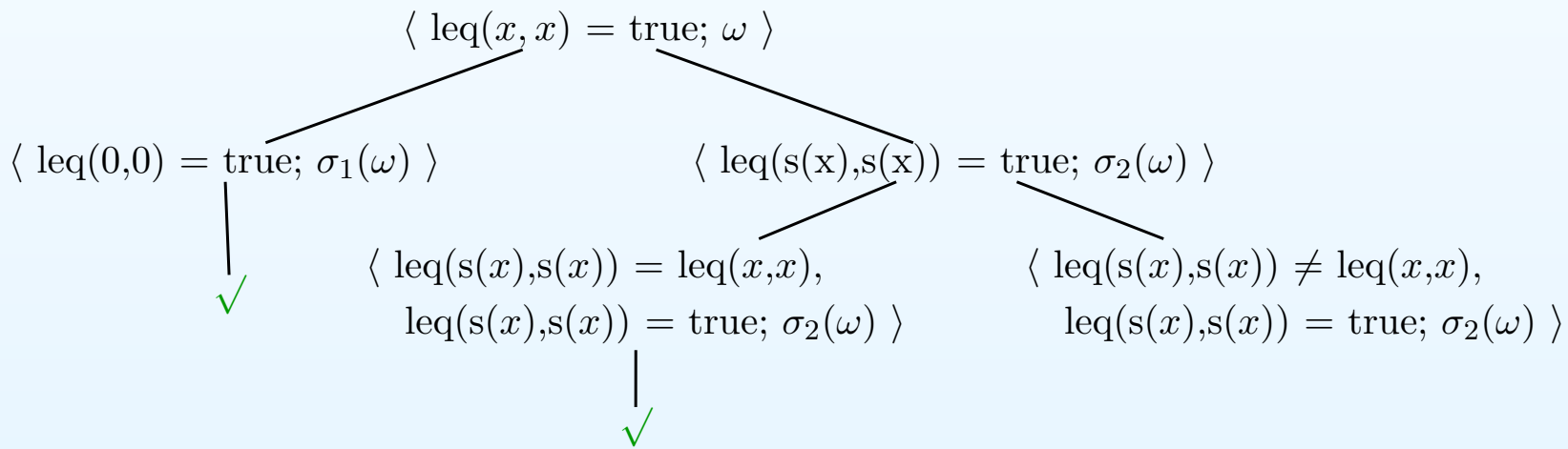
Beispiel: leq(x, x)

- Wir wollen $\mathrm{leq}(x, x) = \mathrm{true}$ beweisen.
- Fallunterscheidung nach $\{\sigma_1 = \{x \leftarrow 0\}, \sigma_2 = \{x \leftarrow s(x)\}\}$
- Subsumption mit $\mathrm{leq}(0,x) = \mathrm{true}$ und $\sigma = \{x \leftarrow 0\}$



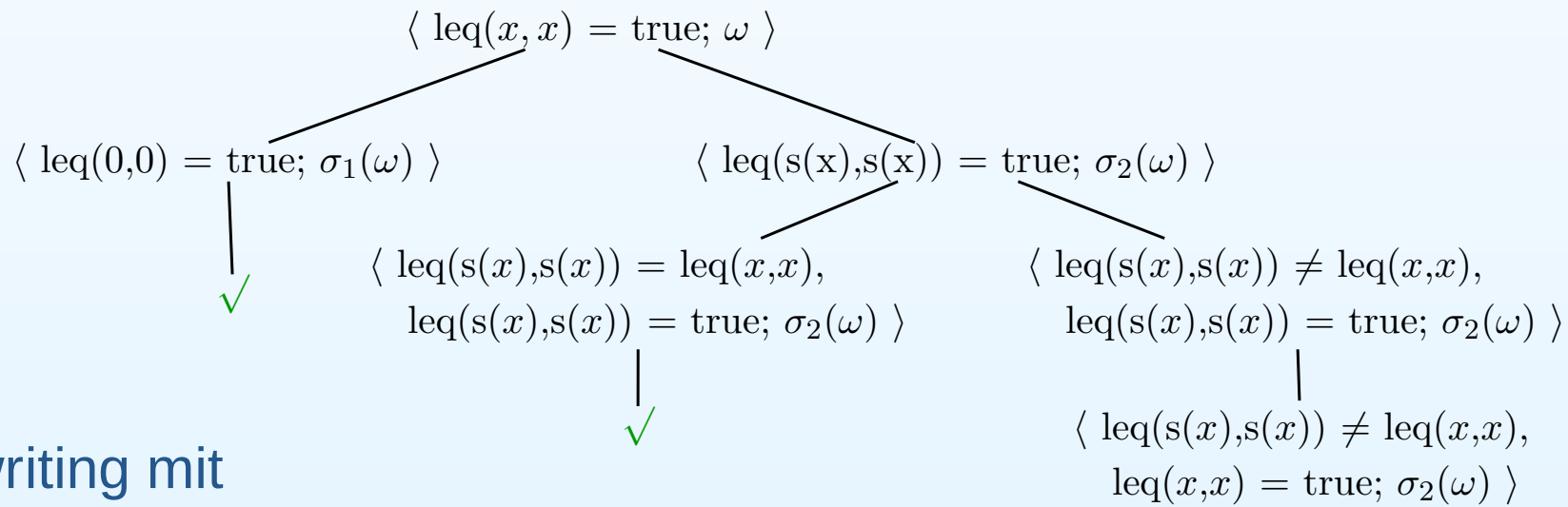
Beispiel: leq(x, x)

- Wir wollen $\mathrm{leq}(x, x) = \mathrm{true}$ beweisen.
- Fallunterscheidung nach $\{\sigma_1 = \{x \leftarrow 0\}, \sigma_2 = \{x \leftarrow s(x)\}\}$
- Subsumption mit $\mathrm{leq}(0,x) = \mathrm{true}$ und $\sigma = \{x \leftarrow 0\}$
- Fallunterscheidung nach $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(x)) = \mathrm{leq}(x,x)$



Beispiel: leq(x, x)

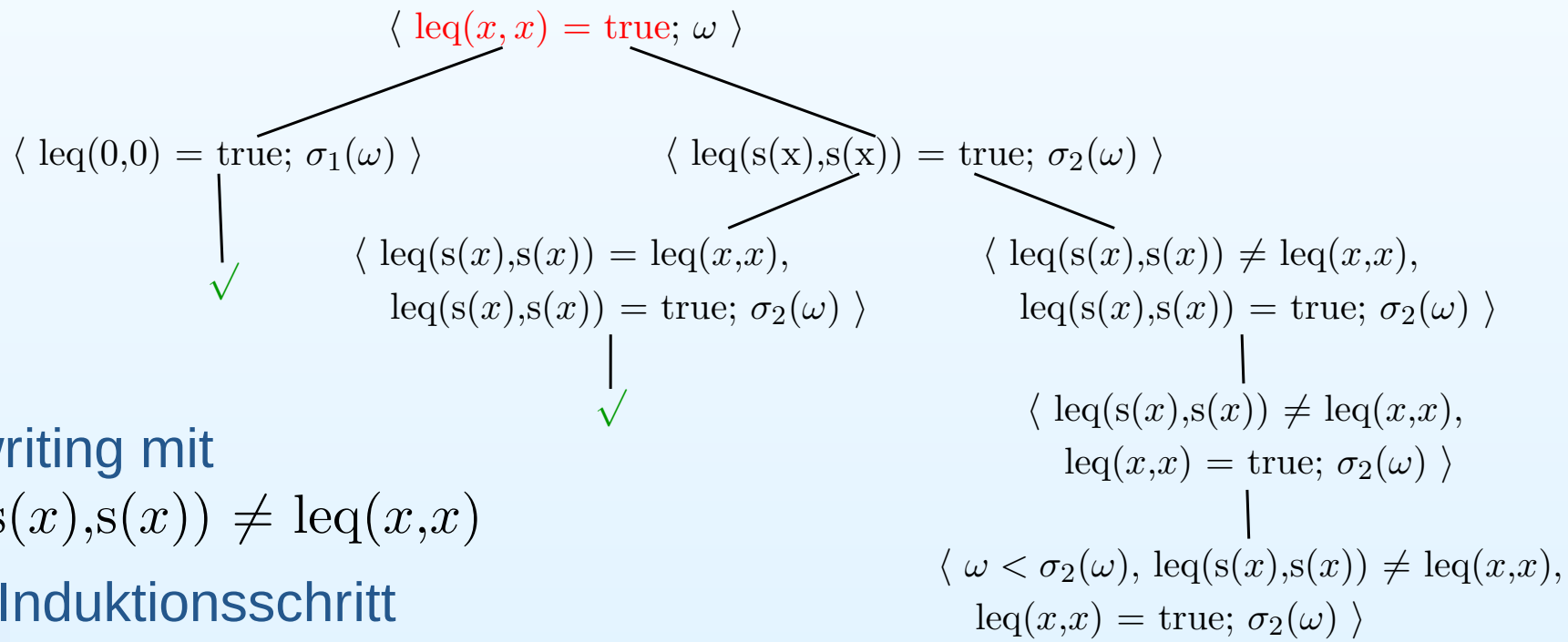
- Wir wollen $\mathrm{leq}(x, x) = \mathrm{true}$ beweisen.
- Fallunterscheidung nach $\{\sigma_1 = \{x \leftarrow 0\}, \sigma_2 = \{x \leftarrow s(x)\}\}$
- Subsumption mit $\mathrm{leq}(0,x) = \mathrm{true}$ und $\sigma = \{x \leftarrow 0\}$
- Fallunterscheidung nach $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(x)) = \mathrm{leq}(x,x)$



- Rewriting mit
  $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(x)) \neq \mathrm{leq}(x,x)$

Beispiel: leq(x, x)

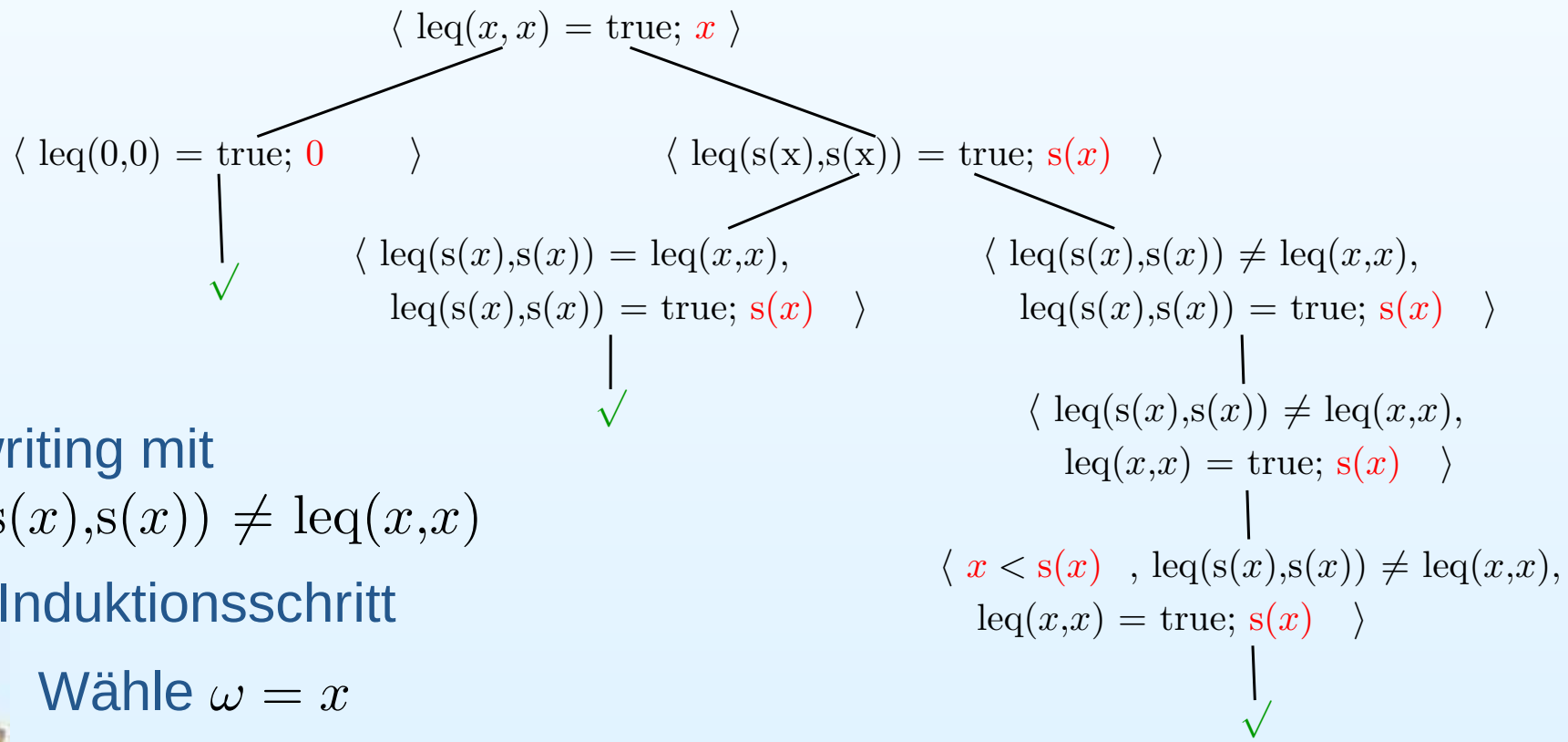
- Wir wollen $\mathrm{leq}(x, x) = \mathrm{true}$ beweisen.
- Fallunterscheidung nach $\{\sigma_1 = \{x \leftarrow 0\}, \sigma_2 = \{x \leftarrow s(x)\}\}$
- Subsumption mit $\mathrm{leq}(0,x) = \mathrm{true}$ und $\sigma = \{x \leftarrow 0\}$
- Fallunterscheidung nach $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(x)) = \mathrm{leq}(x,x)$



- Rewriting mit $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(x)) \neq \mathrm{leq}(x,x)$
- Induktionsschritt

Beispiel: leq(x, x)

- Wir wollen $\mathrm{leq}(x, x) = \mathrm{true}$ beweisen.
- Fallunterscheidung nach $\{\sigma_1 = \{x \leftarrow 0\}, \sigma_2 = \{x \leftarrow s(x)\}\}$
- Subsumption mit $\mathrm{leq}(0,x) = \mathrm{true}$ und $\sigma = \{x \leftarrow 0\}$
- Fallunterscheidung nach $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(x)) = \mathrm{leq}(x,x)$



- Rewriting mit $\mathrm{leq}(\mathrm{s}(x),\mathrm{s}(x)) \neq \mathrm{leq}(x,x)$
- Induktionsschritt
- Wähle $\omega = x$

Beweisbäume

- Ein Beweisbaum ist ein Und-Oder-Baum

Beweisbäume

- Ein Beweisbaum ist ein Und-Oder-Baum
- Inferenzknoten sind Und-Knoten. Alle Teilziele müssen gezeigt werden.

Beweisbäume

- Ein Beweisbaum ist ein Und-Oder-Baum
- Inferenzknoten sind Und-Knoten. Alle Teilziele müssen gezeigt werden.
- Zielknoten sind Oder-Knoten. Es reicht einen Beweisversuch abzuschließen.

Beweisbäume

- Ein Beweisbaum ist ein Und-Oder-Baum
- Inferenzknoten sind Und-Knoten. Alle Teilziele müssen gezeigt werden.
- Zielknoten sind Oder-Knoten. Es reicht einen Beweisversuch abzuschließen.
- Gewichte werden möglichst spät instantiiert.
  Die Gewichte sind Tupel von Termen. Zum Vergleich von Grundtermen wird die Länge des (eindeutigen) E-gleichen Konstruktorgrundterms benutzt. Dieses liften wir auf Terme mit Variablen.

Beweisbäume

- Ein Beweisbaum ist ein Und-Oder-Baum
- Inferenzknoten sind Und-Knoten. Alle Teilziele müssen gezeigt werden.
- Zielknoten sind Oder-Knoten. Es reicht einen Beweisversuch abzuschließen.
- Gewichte werden möglichst spät instantiiert.
  Die Gewichte sind Tupel von Termen. Zum Vergleich von Grundtermen wird die Länge des (eindeutigen) E-gleichen Konstruktorgrundterms benutzt. Dieses liften wir auf Terme mit Variablen.
- Beweise sind schwer automatisierbar $\Rightarrow$ interaktives Beweisen.

Beweisbäume

- Ein Beweisbaum ist ein Und-Oder-Baum
- Inferenzknoten sind Und-Knoten. Alle Teilziele müssen gezeigt werden.
- Zielknoten sind Oder-Knoten. Es reicht einen Beweisversuch abzuschließen.
- Gewichte werden möglichst spät instantiiert.
  Die Gewichte sind Tupel von Termen. Zum Vergleich von Grundtermen wird die Länge des (eindeutigen) E-gleichen Konstruktorgrundterms benutzt. Dieses liften wir auf Terme mit Variablen.
- Beweise sind schwer automatisierbar $\Rightarrow$ interaktives Beweisen.
- Versuch einen hohen Automatisierungsgrad zu erreichen.

Einschränkungen / Leistung

- Freie Konstruktoren
  - Definition einer Kongruenzrelation

Einschränkungen / Leistung

- Freie Konstruktoren
  - Definition einer Kongruenzrelation
- Orthogonale Regelsysteme
  - In der Praxis meist keine Einschränkung

Einschränkungen / Leistung

- Freie Konstruktoren
  - Definition einer Kongruenzrelation
- Orthogonale Regelsysteme
  - In der Praxis meist keine Einschränkung
- Keine Existenzquantoren
  - Konstruktiv simulieren (Skolemfunktion)

Einschränkungen / Leistung

- Freie Konstruktoren
  - Definition einer Kongruenzrelation
- Orthogonale Regelsysteme
  - In der Praxis meist keine Einschränkung
- Keine Existenzquantoren
  - Konstruktiv simulieren (Skolemfunktion)

- Buchführung

Einschränkungen / Leistung

- Freie Konstruktoren
  - Definition einer Kongruenzrelation
- Orthogonale Regelsysteme
  - In der Praxis meist keine Einschränkung
- Keine Existenzquantoren
  - Konstruktiv simulieren (Skolemfunktion)

- Buchführung
- Einfach zu bedienen

Einschränkungen / Leistung

- Freie Konstruktoren
  - Definition einer Kongruenzrelation
- Orthogonale Regelsysteme
  - In der Praxis meist keine Einschränkung
- Keine Existenzquantoren
  - Konstruktiv simulieren (Skolemfunktion)

- Buchführung
- Einfach zu bedienen
- Nicht-terminierende Regelsysteme!

Einschränkungen / Leistung

- Freie Konstruktoren
  - Definition einer Kongruenzrelation
- Orthogonale Regelsysteme
  - In der Praxis meist keine Einschränkung
- Keine Existenzquantoren
  - Konstruktiv simulieren (Skolemfunktion)

- Buchführung
- Einfach zu bedienen
- Nicht-terminierende Regelsysteme!
- Erweiterbarkeit

Einschränkungen / Leistung

- Freie Konstruktoren
  - Definition einer Kongruenzrelation
- Orthogonale Regelsysteme
  - In der Praxis meist keine Einschränkung
- Keine Existenzquantoren
  - Konstruktiv simulieren (Skolemfunktion)

- Buchführung
- Einfach zu bedienen
- Nicht-terminierende Regelsysteme!
- Erweiterbarkeit
- Hoher Automatisierungsgrad