Rechnergestützte Beweissysteme

Beweiskalküle für die Prädikatenlogik

Kalküle der Prädikatenlogik

1. Kalküle und Beweise
   - Korrektheit, Vollständigkeit
2. Der Sequenzenkalkül der PL-1
   - Skolemisieren und Instanziieren
   - Entsprechung in PVS
   - Korrektheit, Vollständigkeit
3. Die Behandlung der Gleichheit
   - Gleichheitsaxiome
   - Ersetzung
4. Resolventenmethode für PL-1
   - Pränexe Normalform
   - Unifikation
   - Prolog

Kalküle der Prädikatenlogik

1. Kalküle und Beweise
   - Korrektheit, Vollständigkeit
2. Der Sequenzenkalkül der PL-1
   - Skolemisieren und Instanziieren
   - Entsprechung in PVS
   - Korrektheit, Vollständigkeit
3. Die Behandlung der Gleichheit
   - Gleichheitsaxiome
   - Ersetzung
4. Resolventenmethode für PL-1
   - Pränexe Normalform
   - Unifikation
   - Prolog

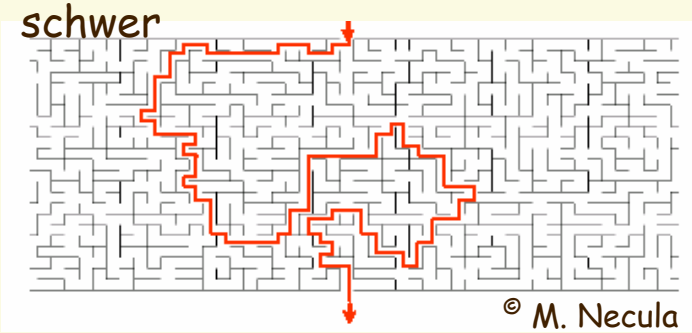
Beweiskalküle

- Problem des Beweisens ist:
  - Stelle einen Beweiskalkül bereit, mit dem man die Relation \models berechnen/entscheiden kann.
- Ein Beweiskalkül ist eine Menge von Schlussregeln, die festlegen
  - wie man aus einer endlichen Menge $P_1,\ldots,P_n$ von logischen Formeln (den Prämissen)
  - eine neue Formel Q (die Konklusion) herleiten darf

$$\frac{P_1, \ldots, P_n}{Q}$$

  - Es dürfen Nebenbedingungen (Bed.) gefordert werden, aber
  - es muss entscheidbar sein, ob eine Regel richtig angewendet wurde

Einen Beweis zu finden ist schwer



Die Korrektheit eines Beweis zu überprüfen ist Routine

Was ist ein Beweis

- Gegeben ein Kalkül, bestehend aus
  - einer Menge von Schlussregeln
  - einer Menge Ax von Axiomen und
  - einer Aussage p.
  -
- Ein Beweis von p aus den Axiomen Ax besteht aus
  - einer Folge $p_1,\ldots,p_n$ von Aussagen mit
    - $p_n = p$
    - Für jedes $i \leq n$ gilt
      - $p_i \in Ax$ oder
      - p_i ist mit den Schlussregeln des Kalküls aus $\{p_1,\ldots,p_{i-1}\}$ herleitbar.
- Wir schreiben
  - $(\Sigma, Ax) \vdash p$
  - wenn Ein Beweis von p aus den Axiomen Ax existiert.

Korrektheit, Vollständigkeit

- Ein Kalkül heißt
  - **korrekt**, falls jede herleitbare Aussage wahr ist, d.h.
    - $Ax \vdash p$ impliziert $Ax \models p$
  - **vollständig**, falls jede wahre Aussage herleitbar ist, d.h.
    - $Ax \models p$ impliziert $Ax \vdash p$
- Für die Prädikatenlogik 1. Stufe gibt es korrekte und vollständige Kalküle

Kalküle für die Prädikatenlogik

- Sequenzenkalkül
  - es gibt davon mehrere Varianten
    - single conclusion sequential calculus
    - multiple conclusion sequential calculus
  - PVS verwendet einen „mcsc"
- Natürliches Schließen
  - Sowohl klassisch
    - mit Tertium non datur: $p \vee \neg p$, bzw. $\neg \neg p = p$
  - als auch intuitionistisch
- Resolventenmethode
  - automatische Beweise
  - oft hoffnungslos blind und resourcenfressend
  - man muss Glück haben oder an den richtigen Schrauben drehen

Kalküle der Prädikatenlogik

1. Kalküle und Beweise
   - Korrektheit, Vollständigkeit
2. Der Sequenzenkalkül der PL-1
   - Skolemisieren und Instanziieren
   - Entsprechung in PVS
   - Korrektheit, Vollständigkeit
3. Die Behandlung der Gleichheit
   - Gleichheitsaxiome
   - Ersetzung
4. Resolventenmethode für PL-1
   - Pränexe Normalform
   - Unifikation
   - Prolog

Prädikatenlogischer Sequenzenkalkül

- Eine *prädikatenlogische Sequenz* ist, ein Ausdruck der Form

$$H_1, H_2, \ldots, H_n \vdash G_1, G_2, \ldots, G_k$$

- wobei die H_i und die G_j prädikatenlogische Aussagen sind.

- Die bisherigen Regeln des Sequenzenkalküls behalten wir bei. Wir benötigen noch Regeln für die Quantoren.

Beweis einer Existenzaussage

- Um eine existenzielle Aussage $\exists x{:}\tau.\ E$ zu beweisen, reicht es, für x ein konkretes Element $t \in A_\tau$ zu finden, das die Aussage erfüllt:

($\exists$-R)

$$\frac{\Gamma \vdash \Delta,\ p[x/t]}{\Gamma \vdash \Delta,\ \exists x{:}\tau.p}$$

t muss ein Term vom Typ τ ohne Variablen sein.

- In einem Rückwärtsbeweis nennt man diesen Schritt:

Instanziieren

Korrektheit der Regel

$$\frac{\Gamma \vdash \Delta, p[x/t]}{\Gamma \vdash \Delta, \exists x{:}\tau.p}$$

- Wir zeigen:
  - Wenn die Prämisse war ist, dann auch die Konklusion.
  - Sei I eine Interpretation in der die Prämisse wahr ist, dann gilt entweder
    - eine der Aussagen in Γ ist falsch, oder
    - eine der Aussagen in Δ ist wahr, oder
    - p[x/t] ist wahr.
  - In den ersten beiden Fällen ist auch die Konklusion wahr
  - im letzten Fall gilt mit der Substition $\sigma := \{x \mapsto t\}$, dass $[\![p]\!]_\sigma = \text{True}$, also ist $\exists x{:}\tau.p$ wahr.

Verwendung einer Existenzaussage

- Sei c ein „frisches" Symbol vom Typ τ, d.h. man weiß nicht mehr, als dass c ein Element aus A_τ repräsentiert. Was man dann aus E[x/c] schließen kann, folgt auch schon aus $\exists x{:}\tau.E$

$$(\exists\text{-L})\quad \frac{\Gamma, E[x/c] \vdash \Delta}{\Gamma, \exists x{:}\tau.E \vdash \Delta}$$

c ist ein „frischer" Name für ein Element vom Typ τ.

- In einem Rückwärtsbeweis heißt dieser Schritt:

Skolemisieren

Beweis einer All-Aussage

- Wenn man E[x/c] beweisen kann, ohne irgendetwas über c vorauszusetzen – ausser, dass es Typ τ hat - dann hat man $\forall x{:}\tau.E$ gezeigt:

($\forall$-R)
$$\frac{\Gamma \vdash \Delta, E[x/c]}{\Gamma \vdash \Delta, \forall x{:}\tau.E}$$
c eine „frische" Konstante vom Typ τ.

- c heißt Skolem-Konstante. Die Verendung dieser Regel in einem Rückwärtsbeweis nennt man

Skolemisieren

Verwendung einer All-Aussage

- Wenn Δ aus $E[x/t]$ folgt, dann folgt es auch aus $\forall x{:}\tau.E$, denn man darf jederzeit t für x einsetzen.

($\forall$-L)

$$\frac{\Gamma,\ E[x/t] \vdash \Delta}{\Gamma,\ \forall x{:}\tau.E \vdash \Delta}$$

t ein beliebiger konstanter Term vom Typ τ.

- In einem Rückwärtsbeweis heißt dieser Schritt:

Instanziieren

Skolemisieren in PVS

[n]

```
... , q
   ⊢
forall(x:T):E, ...
```

`skolem n "a"` →

```
... , q
   ⊢
E[x/a], ....
```

[-n]

```
... , exists (x:T):E
   ⊢
r, ...
```

`skolem -n "a"` →

```
... , E[x/a]
   ⊢
r, ....
```

Statt „a“ kann irgendein neuer Name stehen.

Instanziieren in PVS

[n]

```
... , q
  ⊢
Exists(x:T):E, ...
```

`inst n "t1"` →

```
... , q
  ⊢
E[x/t1], ....
```

[-n]

```
... , Forall(x:T):E
  ⊢
r, ...
```

`inst -n "t1"` →

```
... , E[x/t1]
  ⊢
r, ....
```

Statt „t_1“ kann irgendein Term vom Typ T stehen.

Skolemisieren

- Um einen Allquantor $\forall x.P(x)$ zu beweisen, zeigt man P(c) für eine *„feste aber beliebige Konstante* c".

  `(skolem n ("c1" ... "cn"))`

  `n` : Nummer der Formel,
  `"c1" ... "cn"` : Namen für die Konstanten

```
 zirkulaer :
  |-------
{1}   FORALL (x, y, z: t):x <= y AND y <= z AND z <= x IMPLIES x = y
AND y = z

Rule? (skolem 1 ("a" "b" "c"))
For the top quantifier in 1, we introduce Skolem constants: (a b c),
this simplifies to:

zirkulaer :
  |-------
{1}   a <= b AND b <= c AND c <= a IMPLIES a = b AND b = c
```

Instanziieren

- Einen All-Quantor in der Prämisse kann man mit beliebigen Werten des passenden Typs instanziieren:

  (inst n "t_1" ... "t_n")

- Man beachte hier die fehlenden Klammern um die Terme t_1 ... t_n !!

```
zirkulaer :
{-1}  FORALL (x, y, z: t): x <= y AND y <= z IMPLIES x <= z
[-2]  a <= b
[-3]  b <= c
[-4]  c <= a
  |-------
[1]   a = b AND b = c

Rule? (inst -1 "b" "c" "a")
Instantiating top quantifier in -1 with terms: b,c,a, this simplifies to:

zirkulaer :
{-1}  b <= c AND c <= a IMPLIES b <= a
[-2]  a <= b
[-3]  b <= c
[-4]  c <= a
  |-------
[1]   a = b AND b = c
```

Ein Beispiel – Der Beweisbaum

- Wir zeigen jetzt die Allgemeingültigkeit von
  $(\exists y{:}\tau.\forall x{:}\sigma.P(x,y)) \rightarrow \forall x{:}\sigma.\exists y{:}\tau.\ P(x,y)$

$$P(a,b) \vdash P(a,b) \qquad \text{Axiom}$$

$$\forall x{:}\sigma.P(x,b) \vdash P(a,b) \qquad \forall\text{-L}$$

$$\forall x{:}\sigma.P(x,b) \vdash \exists y{:}\tau.\ P(a,y) \qquad \exists\text{-R}$$

$$\exists y{:}\tau.\forall x{:}\sigma.P(x,y) \vdash \exists y{:}\tau.\ P(a,y) \qquad \exists\text{-L}$$

$$\exists y{:}\tau.\forall x{:}\sigma.P(x,y) \vdash \forall x{:}\sigma.\exists y{:}\tau.\ P(x,y) \qquad \forall\text{-R}$$

$$\vdash (\exists y{:}\tau.\forall x{:}\sigma.P(x,y)) \rightarrow \forall x{:}\sigma.\exists y{:}\tau.\ P(x,y)$$

Spezifikation in PVS

```
PredLog   : THEORY
 BEGIN

   s,t : TYPE
   P:[s,t -> boolean]

   test:THEOREM
      (EXISTS (y:t):
         FORALL (x:s):
                  P(x,y) )
     IMPLIES
        FORALL (x:s):
         EXISTS (y:t):
                  P(x,y)
  END PredLog
```

s, t sind Typen
P ist zweistellige Relation zwischen s und t

Quantoren haben geringste Bindungsstärke, sie erstrecken sich also so weit nach rechts wie möglich.
Daher ist die Klammer notwendig

Der Beweis in PVS

```
test :

  |-------
{1} (EXISTS (y: t): FORALL (x: s): P(x, y))
IMPLIES
    (FORALL (x: s): EXISTS (y: t): P(x, y))

Rule? (flatten)
Applying disjunctive simplification to
flatten sequent,this simplifies to:
test :

{-1} (EXISTS (y: t): FORALL (x: s): P(x, y))
  |-------
{1}   FORALL (x: s): EXISTS (y: t): P(x, y)

Rule? (skolem 1 "a")
For the top quantifier in 1, we introduce
Skolem constants: a, this simplifies to:
test :

[-1] (EXISTS (y: t): FORALL (x: s): P(x, y))
  |-------
{1}   EXISTS (y: t): P(a, y)

Rule? (skolem -1 "b")
```

```
For the top quantifier in -1, we
introduce Skolem constants: b, this
simplifies to:
test :

{-1}  FORALL (x: s): P(x, b)
  |-------
[1]   EXISTS (y: t): P(a, y)

Rule? (inst -1 "a")
Instantiating the top quantifier in -1
with the terms: a, this simplifies to:
test :

{-1}  P(a, b)
  |-------
[1]   EXISTS (y: t): P(a, y)

Rule? (inst 1 "b")
Instantiating the top quantifier in 1
with the terms:  b,

Q.E.D.
Run time  = 0.27 secs.
Real time = 102.55 secs.
```

Einige Quantoren-Formeln

- Die folgenden Formeln sind allgemeingültig und leicht auch in PVS zu beweisen:
  - $\forall x{:}T.P(x) \rightarrow \exists x{:}T.P(x)$, falls $T \neq \emptyset$
  - $\exists y{:}S.\ \forall x{:}T.\ P(x,y) \rightarrow \forall x{:}T.\exists y{:}S.P(x,y)$
  - $\exists x{:}T.P(x) \vee Q(x) \leftrightarrow \exists x{:}T.P(x) \vee \exists x{:}T.Q(x)$
  - $\forall x{:}T.P(x) \wedge Q(x) \leftrightarrow \forall x{:}T.P(x) \wedge \forall x{:}T.Q(x)$
  - Wenn $f{:}T \rightarrow S$ dann
    $(\forall x{:}T.R(x,f(x))) \rightarrow \forall x{:}T.\exists y{:}S.R(x,y)$

Signaturvergrößerung

- Seien Σ_1 und Σ_2 Signaturen mit $\Sigma_1 \subseteq \Sigma_2$, d.h. Σ_2 hat evtl. zusätzliche Operationszeichen. Offensichtlich gilt
  - $\mathcal{L}(\Sigma_1) \subseteq \mathcal{L}(\Sigma_2)$, d.h.
  - jede Σ_1-Formel ist auch eine Σ_2-Formel
  - Insbesondere:
    - Jedes Σ_2-Modell ist auch ein Σ_1-Modell
  - Andererseits kann man aus jedem nichtleeren Σ_1-Modell $\mathcal{M}_1$ ein Σ_2-Modell $\mathcal{M}_2$ gewinnen:
    - zu jedem Funktionszeichen f aus Σ_2-Σ_1 definiert man die zugehörige Operation beliebig
- Speziell für $\Sigma^c := \Sigma \cup \{c\}$
  - Jede Σ-Aussage ist auch eine Σ^c-Aussage
  - Eine Σ-Aussage $\forall x.\phi$ ist wahr in einem Σ-Modell $\mathcal{M} \Leftrightarrow \phi[x/c]$ ist wahr in jedem Σ^c-Model, das man aus $\mathcal{M}$ enthält, indem man die Konstante c interpretiert

Umkehrbarkeit von

$$\frac{\Gamma \vdash \Delta, E[x/c]}{\Gamma \vdash \Delta, \forall x{:}\tau.E}$$

- Wir zeigen:
  - Wenn die Konklusion wahr ist, dann auch die Prämisse.
  - Sei I eine Interpretation in der die Konklusion wahr ist,
  - dann gilt entweder
    - eine der Aussagen in Γ ist falsch, oder
    - eine der Aussagen in Δ ist wahr, oder
    - $\forall x{:}\tau.E$ ist wahr.
  - In den ersten beiden Fällen ist auch die Prämisse wahr
  - im letzten Fall ist $\forall x{:}\tau.E$ auch wahr in der Interpretation mit einer zusätzlichen Konstanten c ohne Axiome
    - Insbesondere ist in dieser Interpretation auch $E[x/c]$ wahr
- Für die Regel

$$\frac{\Gamma, E[x/t] \vdash \Delta}{\Gamma, \forall x{:}\tau.E \vdash \Delta}$$

  - gelingt eine entsprechende Aussage nicht !!!

Mehrfachinstanziierung

- Bei einer Instanziierung verschwindet entweder ein Allquantor im Antezedent oder ein Existenzquantor im Succedent. Es ist aber manchmal nötig, diese Quantoren mehrfach zu instanziieren.
- Beispiel:
- $\forall x{:}T.\ \forall y{:}T.\ P(x,y) \rightarrow P(y,x),\ P(a,P(b,c)) \vdash P(P(c,b),a)$
- Bei einem Rückwärtsbeweis mit ($\forall$-L) bzw. mit dem PVS-Befehl **`inst`**, verschwindet der Allquantor nach der ersten Instanziierung.
- Lösung: Vor der Instanziierung die allquantifizierte Formel kopieren - mit Regel (con-L) bzw. PVS-Befehl (copy).
- Besser: Kopieren gleich in die Regel einbauen. Auf diese Weise wird die ($\forall$-L)-Regel invertierbar:

Invertierbare Instanziierungsregeln

- Invertierbare Varianten der Instanziierungsregeln:

($\forall$-L') $\dfrac{\Gamma, \forall x{:}\tau.E, E[x/t] \vdash \Delta}{\Gamma, \forall x{:}\tau.E \vdash \Delta}$ t ein beliebiger konstanter Term vom Typ τ.

($\exists$-R') $\dfrac{\Gamma \vdash \Delta, \exists x{:}\tau.p, p[x/t]}{\Gamma \vdash \Delta, \exists x{:}\tau.p}$ t ein beliebiger konstanter Term vom Typ τ.

- Kombination der Quantoren-Regeln mit Kontraktionen (con-R) bzw. (con-L) $\Rightarrow$ korrekt.
- Invertierbar - mit Hilfe von (weak-L), bzw. (weak-R)
- Im Rückwärtsbeweis „vergibt man sich nichts" durch eine schlechte Instanziierung

Termmodell $\mathbb{T}_\Sigma$

- Σ eine Signatur
  - Typnamen $\tau_1,\ldots,\tau_k$.
  - Operationsnamen: $f_i:\tau_{i1},\ldots,\tau_{im} \to \tau_i$
  - Relationsnamen: $R_j:\tau_{j1},\ldots,\tau_{jn}$

- Termmodell $\mathbb{T}_\Sigma$: Für jeden Typ τ :

  $\mathbb{T}_\tau$ = Menge aller variablen-freien Terme vom Typ τ.

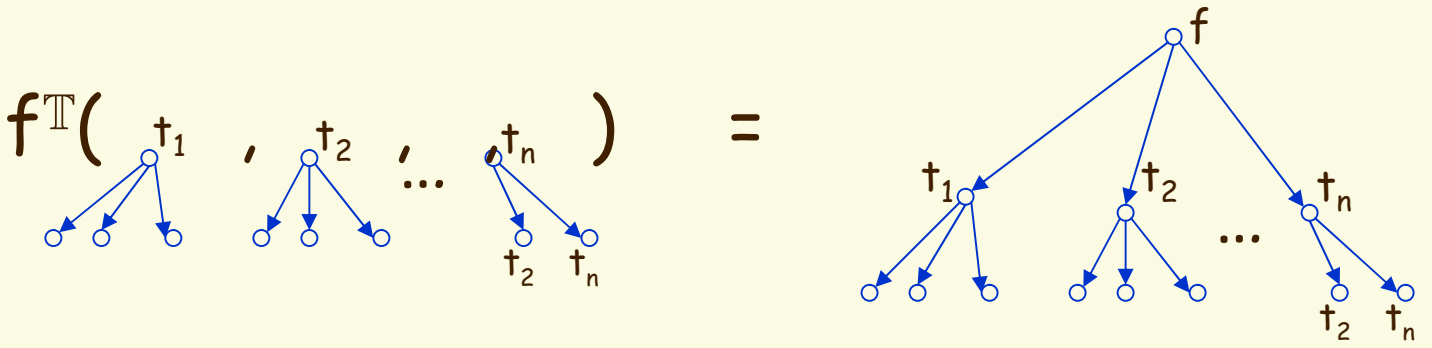
  (Bemerkung: $\mathbb{T}_\tau$ ist aufzählbar)

  Für jedes Operationssymbol $f: \tau_1,\ldots,\tau_n \to \tau$ definiere

$$f : \mathbb{T}_{\tau 1} \times \ldots \times \mathbb{T}_{\tau n} \to \mathbb{T}_\tau$$

  durch

$$f^{\mathbb{T}}(t_1,\ldots,t_n) := f(t_1,\ldots,t_n)$$



Herbrandstruktur

- Σ eine Signatur
  - Typnamen $\tau_1,\ldots,\tau_k$.
  - Operationsnamen: $f_i:\tau_{i1},\ldots,\tau_{im} \rightarrow \tau_i$
  - Relationsnamen: $R_j:\tau_{j1},\ldots,\tau_{jn}$
- Wähle wie im Termmodell
  - Grundmengen $\mathbb{T}_\tau$: Variablenfreie Terme vom Typ τ
  - Operationen syntaktisch: $f^{\mathbb{T}}(t_1,\ldots,t_n) := f(t_1,\ldots,t_n)$
  - $=\tau$ syntaktische Termgleichheit auf $\mathbb{T}_\tau$
  - sonstige Relationen nach Belieben: $R^{\mathbb{T}} \subseteq \mathbb{T}_{\tau 1} \times \ldots \times \mathbb{T}_{\tau n}$
- Jede solche Struktur heißt Herbrandstruktur $\mathcal{H}$

Herbrandstruktur – Beispiel 1

- Beispiel:
  - Sortensymbole: { G }
  - Operationssymbole :
    - $o : G \times G \rightarrow G$,
    - $^{-1} : G \rightarrow G$,
    - $e : G$
  - Relationssymbol
    - $= :: G \times G$

- Termmodell
  - $\mathbb{T}_G = \{ e, e^{-1}, eoe, (eoe)oe, eo(eoe), eoe^{-1}, (eoe)^{-1}, \ldots \}$
  - $=_G = \{ (t,t) \mid t \in \mathbb{T}_G \}$

  - Operationen (z.B.)
    - $o^{\mathbb{T}}(e, eoe^{-1}) = eo(eoe^{-1})$
    - $^{-1\mathbb{T}}(e^{-1}) = (e^{-1})^{-1}$

Herbrandstruktur – Beispiel 2

- Beispiel:
  - Sortenname: N
  - Operationssymbole
    - $0 : N$
    - $s : N \to N$
    - $+ : N \times N \to N$
  - Relationssymbol
    - $\leq : N \times N$

- Termmodell (Grundmenge)
  - $\mathbb{T}_N = \{0, s(0), s(s(0)), 0+0, s(0)+s(s(0)), \ldots\}$

  Operationen (z.B.)

  - $s^{\mathbb{T}}(0) = s(0)$,
  - $+^{\mathbb{T}}(s(0), s(s(0))) = s(0)+s(s(0))$

- Relation alles erlaubt, z.B.:
  - $s(0) \leq^{\mathbb{T}} 0$
  - $0+0 \leq^{\mathbb{T}} s(s(0))$

Herbrandstruktur – Beispiel 3

- Σ = (N,S; (null, push, 0, s, +), $\leq$)
  - (N; 0,s,+; $\leq$) wie vorhin
  - null: S
  - push : $S \times N \rightarrow S$
- Termmodell (Grundmengen)
  - $\mathbb{T}_N$ = {0, s(0), s(s(0)), 0+0, s(0)+s(s(0)), ... } wie bisher
  - $\mathbb{T}_S$ = { null, push(0,null), push(s(0),push(0,null)), ... }
- Operationen (z.B.)
  - push$^{\mathbb{T}}$ (s(0), push(0,null)) = push(s(0), push(0,null)), etc.
- Relationen beliebig, z.B.
  - $\leq^{\mathbb{T}}$:= { (s(0), 0), (s(s(0)), 0+s(0)) }
    - etwas verrückt, aber legal

Semantische Implikation

Sei $\Gamma \cup \{\phi\}$ eine Menge von Aussagen der PL1.

Wir sagen

$$\Gamma \models \phi$$

falls in jedem Modell von Γ auch ϕ richtig ist.

Beispiel: Σ sei Signatur (R; +, -,·, 0, 1; =) vom Typ (2,2,2,0,0)

Γ := Axiome der Ringtheorie $\cup$ { $\forall$ x:R. x· x · x = x}.

ϕ := $\forall$x:R. $\forall$y:R. x · y = y · x.

Es gilt $\Gamma \models \phi$.

Wieso ? Betrachte alle Ringe mit $x^3=x$. Prüfe nach, dass sie kommutativ sind.

Wie soll ich das anstellen ?

Logische Implikation

Sei $\Gamma \cup \{\phi\}$ eine Menge von Aussagen der PL1.

Wir sagen

$$\Gamma \vdash \phi$$

falls mit Hilfe der Aussagen in Γ die Aussage ϕ herleitbar ist.

Das bedeutet: Mit einem logischen Kalkül können wir ϕ beweisen, wobei wir jederzeit auf Aussagen aus Γ zurückgreifen können.

Beispiel: $\Sigma=(G; \cdot, e; =)$ vom Typ (2, 0)
Γ := { $\forall$ x,y,z:G. x·(y·z) = (x·y)·z, $\forall$ x:G.$\exists$ y:G. x·y = e, $\forall$x:G. x·e = x}
ϕ := $\forall$ x:G. $\exists$ y:G. y· x = e.
Es gilt $\Gamma \vdash \phi$, denn wir können dies leicht mit PVS zeigen.

Korrekt - Vollständig

- Ein Beweiskalkül heißt korrekt, wenn man nichts falsches herleiten kann, genauer, falls

  aus $\Gamma \vdash \phi$ folgt $\Gamma \models \phi$

- Ein Beweiskalkül heißt vollständig, wenn man alles, was wahr ist, herleiten kann, genauer, falls

  aus $\Gamma \models \phi$ folgt $\Gamma \vdash \phi$

- PVS (bzw. der Sequenzenkalkül) ist korrekt
  - alle Regeln sind korrekt

- aber ist PVS auch vollständig ?

Vollständigkeit

$$\frac{\Gamma \vdash \Delta, \exists x{:}\tau.p, \; p[x/t]}{\Gamma \vdash \Delta, \exists x{:}\tau.p}$$

- Satz: Die bisherigen Regeln des Sequenzenkalküls sind *vollständig:*
  - aus $\Gamma \models \phi$ folgt $\Gamma \vdash_{PVS} \phi$.
- Beweisidee: Versuche den aussagenlogischen Beweis zu imitieren.
  - Die Hypothesen und Goals werden als Mengen aufgefasst, daher kann man auf die Strukturregeln verzichten
  - Die Weakening-Regeln sind überflüssig
  - Kann man ein Axiom anwenden, so endet der entsprechende Ast.
- Eliminiere, wie im aussagenlogischen Fall systematisch alle logischen Operatoren
- $\exists$-links und $\forall$-rechts werden durch Skolemisieren eliminiert
- Nur wenn keine der obigen Regeln anwendbar sind, benutze die Instanziierungsregeln $\exists$-rechts bzw. $\forall$-links und zwar in der invertierbaren Form (s.o.)

Vollständigkeitsbeweis

- Instanziiere systematisch alle variablenfreien Terme $t \in \mathbb{T}_\tau$
- Entweder schließt sich so jeder Ast des Baumes
  - ist jedes Blatt ein Axiom, so ist φ allgemeingültig
  - ist ein Blatt kein Axiom so liefert es ein Gegenbeispiel, da alle Regeln invertierbar sind.
- Oder es gibt einen unendlich langen Pfad π
  - in diesem Falle werden wir ein Herbrandmodell basteln, in dem φ nicht gültig ist.

Sei π ein unendlich langer Ast im Beweisbaum von φ

Idee: Konstruiere eine Herbrand-Struktur $\mathcal{H}$ in der $\neg\varphi$ gilt:
Setze

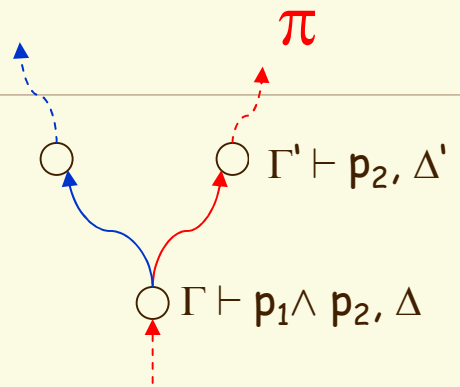
$$R_i^{\mathcal{H}}(c_1,\ldots,c_n) \quad :\Leftrightarrow \quad \pi \text{ enthält eine Sequenz der Form } \Gamma, R_i(c_1,\ldots,c_n) \vdash \Delta.$$

Im folgenden Lemma zeigen wir, dass für jede Sequenz $\Gamma, p \vdash q, \Delta$ aus π gilt:

$$\mathcal{H} \models p \text{ und } \mathcal{H} \models \neg q.$$

Weil π mit der Sequenz $\vdash \varphi$ beginnt, schließen wir: $\mathcal{H} \models \neg\varphi$.

Lemma



- Für jede Formel p gilt:
  - A): Wenn $\Gamma, p \vdash \Delta$ aus π, dann: $\mathcal{H} \models p$
  - B): Wenn $\Gamma \vdash p, \Delta$ aus π, dann: $\mathcal{H} \models \neg p$.
- Beweis (induktion über Aufbau von p):
  - $p = R_j(t_1, \ldots, t_n)$:
    - A) Wenn $\Gamma, p \vdash \Delta$ in π, dann gilt $\mathcal{H} \models p$ nach Konstruktion
    - B) Wenn $\Gamma \vdash p, \Delta$ in π, dann kann nicht $\mathcal{H} \models p$, gelten, denn dafür müsste p in einem Sequenten aus π links auftauchen. Da atomare Aussagen weder entstehen noch verschwinden können, hätte man auch einen Sequenten $\Gamma, p \vdash p, \Delta$ in π und abgebrochen.
  - $p = p_1 \wedge p_2$
    - A) Aus $\Gamma, p_1 \wedge p_2 \vdash \Delta$ in π, folgt $\Gamma', p_1, p_2 \vdash \Delta'$ in π. Nach Ind Hyp. $\mathcal{H} \models p_1$ und $\mathcal{H} \models p_2$ nach Ind.Hyp., also $\mathcal{H} \models p_1 \wedge p_2$
    - B) Aus $\Gamma \vdash p_1 \wedge p_2, \Delta$ in π, folgt $\Gamma' \vdash p_1, \Delta'$ in π oder $\Gamma' \vdash p_2, \Delta'$ in π, also $\mathcal{H} \models \neg p_1$ oder $\mathcal{H} \models \neg p_2$ nach Ind Hyp., also $\mathcal{H} \models \neg (p_1 \wedge p_2)$
  - $p = p_1 \rightarrow p_2$, $p = \neg p_1$ (analog)

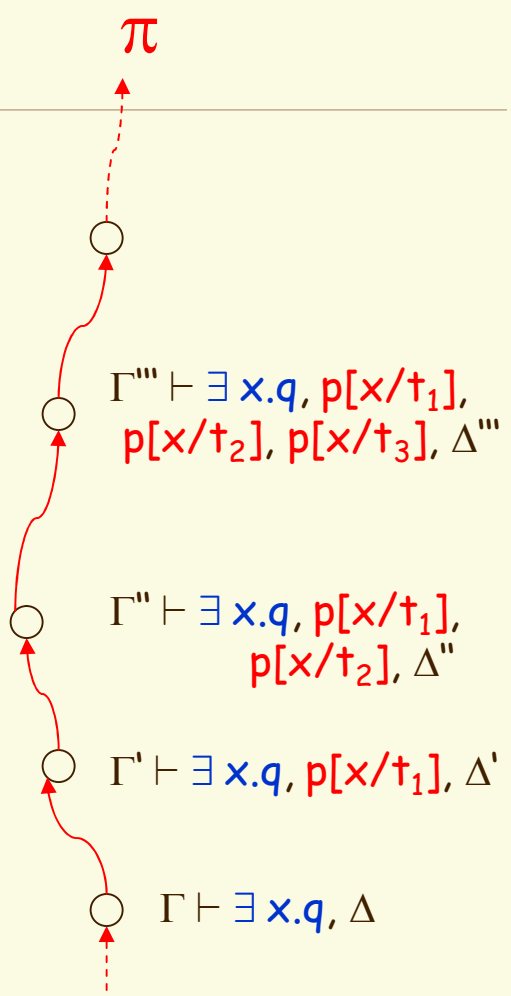
Lemma (Forts.)

p = $\exists x{:}\tau.q$

A) Ist $\Gamma, \exists x{:}\tau.q \vdash \Delta$ aus π, so erscheint irgendwann später auch $\Gamma', q[x/c] \vdash \Delta'$ in π. Nach Ind.Vor. gilt $\mathcal{H} \models q[x/c]$, also $\mathcal{H} \models \exists x{:}\tau.q$.

B) Ist $\Gamma \vdash \exists x{:}\tau.q, \Delta$ aus π, so erscheint für jeden konstanten Term $t{:}\tau$, also für jedes Element aus $\mathbb{T}_\tau$ irgendwann auch ein Sequent $\Gamma' \vdash q[x/t], \Delta'$ in p. Nach Ind.Vor. gilt $\mathcal{H} \models \neg q[x/t]$, für jedes Element von $\mathbb{T}_\tau$, also $\mathcal{H} \models \neg\exists x{:}\tau.q$.

- P= $\forall x{:}\tau.q$ analog.



Kompaktheitssatz

- Satz: Eine Menge von Aussagen der Prädikatenlogik 1. Stufe hat ein Modell genau dann wenn jede endliche Teilmenge ein Modell hat.

- Beweis: Sei Γ eine Menge von Aussagen. Die Behauiptung folgt aus den folgenden Äquivalenzumformungen :

  Γ hat kein Modell

  $\Leftrightarrow \Gamma \models \bot$

  $\Leftrightarrow \Gamma \vdash \bot$

  $\Leftrightarrow$ Es gibt eine endliche Teilmenge $\Gamma_0 \subseteq \Gamma$ mit $\Gamma_0 \vdash \bot$

  $\Leftrightarrow$ Es gibt eine endliche Teilmenge $\Gamma_0 \subseteq \Gamma$ mit $\Gamma_0 \models \bot$

  $\Leftrightarrow$ Eine endliche Teilmenge von Γ hat kein Modell

Kalküle der Prädikatenlogik

1. Kalküle und Beweise
   - Korrektheit, Vollständigkeit
2. Der Sequenzenkalkül der PL-1
   - Skolemisieren und Instanziieren
   - Entsprechung in PVS
   - Korrektheit, Vollständigkeit
3. Die Behandlung der Gleichheit
   - Gleichheitsaxiome
   - Ersetzung
4. Resolventenmethode für PL-1
   - Pränexe Normalform
   - Unifikation
   - Prolog

Die Behandlung der Gleichheit

- Für jeden Typ τ ist meist eine Gleichheitsrelation $=_\tau$ definiert. Den Index τ lässt man weg.

- Axiome der Gleichheit
  - $x = x$
  - $x = y \rightarrow y = x$
  - $x = y \wedge y = z \rightarrow x = z$
  - Für jedes n-stellige Funktionszeichen f und jedes Prädikat R gilt:
    - $s_1 = t_1 \wedge \ldots \wedge s_n = t_n \rightarrow f(s_1,\ldots,s_n) = f(t_1, \ldots, t_n)$
    - $s_1 = t_1 \wedge \ldots \wedge s_n = t_n \wedge R(s_1,\ldots,s_n) \rightarrow R(t_1,\ldots,t_n)$

- Folgerung: Für jeden Term r und jede Formel φ gilt:
  - $s=t \rightarrow r[x/s] = r[x/t]$ (*Beweis*: *Ind. über Aufbau von* r)
  - $s=t \wedge \varphi[x/s] \rightarrow \varphi[x/t]$ (*Beweis*: *Ind. über Aufbau von* φ)

Gleichheit im Sequenzenkalkül

- Im Sequenzenkalkül kommt man mit den folgenden Regeln aus:

Axiom :

$$\frac{}{\Gamma \vdash \Delta, t = t} \quad \text{(reflexiv)}$$

Regel :

$$\frac{\Gamma, s = t \vdash \Delta, p[x/t]}{\Gamma, s = t \vdash \Delta, p[x/s]} \quad \text{(replace ... +)}$$

Übung: Leiten Sie mit diesen Regeln die Axiome der vorigen Folie her.

Gleichheit in PVS

- Das Axiom für Reflexivität wird nicht benötigt. PVS führt es von selber aus – ähnlich wie die „¬" – Regeln.
- Die Ersetzungsregel wird durch das Kommando „replace" aufgerufen:


```
                ...
[-k]  s = t
        ...
 |------
        ...,
[n]   r[x/s]
        ...
```

```
(replace -k n)
```

Optionale Parameter
`:dir RL` bzw `:dir LR`

```
                ...
[-k]  s = t
        ...
 |------
        ...,
[n]   r[x/t]
        ...
```

Gleichheiten einsetzen

- Eine Gleichung **x=t** in der Hypothese erlaubt es, gezielt oder überall „**x**" durch „**t**" zu ersetzen. Der Befehl lautet

  (replace n wo)

```
zirkulaer.1.1.2 :
[-1]   a = c
{-2}   c <= c
{-3}   c <= b
[-4]   b <= c
   |-------
{1}    c = b
Rule? (replace -1 (-2 1) :dir RL)

Replacing using formula -1,this
simplifies to:
zirkulaer.1.1.2 :
[-1]   a = c
{-2}   a <= a
[-3]   c <= b
[-4]   b <= c
   |-------
{1}    a = b
```

n : Nummer der Formel **x=t**
wo : Nummer(n) der Formel(n) in denen ersetzt werden soll

Optionale Argumente können in LISP selektiv angegeben werden:

Hier hat das optionale Argument **:dir** den Default-Wert LR
Als Werte sind hier zugelassen: RL oder LR.

Abgeleitete Gleichheitsregeln

Weitere Gleichheitsregeln kann man herleiten, z.B.:

Regel : $\dfrac{\Gamma, s = t \vdash \Delta, p[x/s]}{\Gamma, s = t \vdash \Delta, p[x/t]}$ (replace ... :dir RL +)

Regel : $\dfrac{\Gamma, s = t, p[x/t] \vdash \Delta}{\Gamma, s = t, p[x/s] \vdash \Delta}$ (replace ... -)

Regel : $\dfrac{\Gamma, s = t, p[x/s] \vdash \Delta}{\Gamma, s = t, p[x/t] \vdash \Delta}$ (replace ... :dir RL -)

Herleitung einer Gleichheitsregel

- Aus der Substitution rechts folgt Substitution links
  - Trick: Negation um Ausdruck nach rechts zu bringen.

$$\dfrac{\dfrac{}{p[x/s] \vdash p[x/s], \Delta}}{\dfrac{\Gamma, s = t, p[x/s] \vdash p[x/s], \Delta}{\Gamma, s = t, p[x/s] \vdash \neg\neg p[x/s], \Delta}} \qquad \dfrac{\dfrac{\dfrac{\Gamma, s = t,\ p[x/t] \vdash \Delta}{\Gamma, s = t \vdash \neg p[x/t], \Delta}}{\Gamma, s = t \vdash \neg p[x/s], \Delta}}{\dfrac{\Gamma, s = t, p[x/s] \vdash \neg p[x/s], \Delta}{\Gamma, s = t, p[x/s], \neg\neg p[x/s] \vdash \Delta}}$$

$$\dfrac{\Gamma, s = t, p[x/s] \vdash \neg\neg p[x/s], \Delta \qquad \Gamma, s = t, p[x/s], \neg\neg p[x/s] \vdash \Delta}{\Gamma, s = t, p[x/s] \vdash \Delta}$$

Rewrite

- Wenn ein AXIOM, LEMMA oder THEOREM eine Gleichung „ $\forall x_1,..,x_n.\ s = t$“ ist, sind meist drei Befehle nötig, diese anzuwenden
  - `(lemma` „*meinAxiom*“`)` % Axiom in Antezedenten
  - `(inst ... )` % Instanziieren
  - `(replace ...)` % Ersetzen
- Der Befehl
  - `(rewrite` „*meinAxiom*“`)`

  versucht diese drei Schritte selber zu erledigen. Es ist aber nicht garantiert, dass die *gewünschte* Substitution gefunden wird.

Beispiel für rewrite

Aufgabe:

```
Gleichheit  : THEORY
BEGIN
Gruppoid: TYPE
* : [Gruppoid,Gruppoid -> Gruppoid]

assoziativ : AXIOM
          FORALL (x,y,z:Gruppoid):
          x*(y*z) = (x*y)*z

fassoc : THEOREM
          FORALL (x,y,z,u:Gruppoid):
          x*((y*z)*u) = ((x*y)*z)*u

END Gleichheit
```

Standardbeweis (verkürzt):

```
 fassoc :
|-------
{1}   FORALL (x, y, z, u: Gruppoid):
  x * ((y * z) * u) = ((x * y) * z) * u
    (skolem!)
    (lemma "assoziativ")
    (inst-cp -1 "x!1" "(y!1*z!1)" "u!1")
    (replace -2 1)
    (inst -1 "x!1" "y!1" "z!1")
    (replace -1 1)
|-------
{1}   TRUE
```

Beweis mit „rewrite“:

```
(skolem!)
(rewrite "assoziativ")
(rewrite "assoziativ")
```

Kalküle der Prädikatenlogik

1. Kalküle und Beweise
   - Korrektheit, Vollständigkeit
2. Der Sequenzenkalkül der PL-1
   - Skolemisieren und Instanziieren
   - Entsprechung in PVS
   - Korrektheit, Vollständigkeit
3. Die Behandlung der Gleichheit
   - Gleichheitsaxiome
   - Ersetzung
4. Resolventenmethode für PL-1
   - Pränexe Normalform
   - Unifikation
   - Prolog

Resolventenmethode für PL1

- Der *Gentzenkalkül* ist vollständig – aber interaktiv/mühsam
  - Schwierigkeit: *Instantiierungen*
- *Resolventenmethode* für die Aussagenlogik war automatisch und effizient
- Kann man die Resolventenmethode auf die Prädikatenlogik ausdehnen ?
- Kann man geeignete Instanziierungen automatisch finden ?

Prädikatenlogische Klauseln

- Ein Literal ist ein atomare oder negiert atomare Formel
  $R(t_1,\ldots,t_n)$ bzw. $\neg R(t_1,\ldots,t_n)$.
  - Beispiele: sorted(ins(3,ins(2,y))),
    istPerm(ins(3,ins(2,y)),ins(2,ins(3,y))
    P(f(x),y)

- Eine Klausel ist eine Disjunktion von Literalen
  - Beispiele: $\neg P(f(x,w),w) \vee \neg Q(s(w)) \vee R(x,f(g(w),w))$
    $istPerm(x,y) \vee P(x,g(y))$

- Eine Klausel ist eine offene Formel – und jede offene Formel lässt sich in Klauselform bringen:

Formeln als Aussagen

- Eine **Formel**, in der freie Variablen vorkommen, kann man als – schlampig geschriebene - **Aussage** deuten, indem man alle freien Variablen allquantifiziert:

- Für eine **Formel** $\Psi(x_1,\ldots,x_n)$, in der die Variablen $x_1{:}\tau_1 \ldots x_n{:}\tau_n$ frei vorkommen, sei

  $$\forall x_1{:}\tau_1 \ \ldots \ \forall x_n{:}\tau_v \ \Psi(x_1,\ldots,x_n)$$

  die zugehörige **Aussage**.

- Beispiele: „$R(x,y) \wedge R(y,z) \rightarrow R(x,z)$" wird interpretiert als Aussage

  „$\forall x{:}\tau.\ \forall y{:}\tau.\ \forall z{:}\tau.(\ R(x,y) \wedge R(y,z) \rightarrow R(x,z)\ )$"

  „$T(x,y) \leftrightarrow \exists z{:}\tau.\ R(x,z) \wedge S(z,y)$" als „$\forall x{:}\tau_1.\ \forall y{:}\tau_2.(\ T(x,y) \leftrightarrow \exists z{:}\tau.\ R(x,z) \wedge S(z,y)\ )$"

Pränexe Normalform

- Eine Formel ist in pränexer Normalform, falls alle Quantoren aussen stehen – genauer:
  - Jede Konjunktion von Klauseln ist in pränexer Normalform
  - Ist Q in pränexer Normalform, dann auch

    $\forall\, x{:}\tau.\ Q$

    $\exists\, x{:}\tau.\ Q$

Beispiele

$\forall x{:}\tau.\ \forall y{:}\tau.(\ T(x,y) \leftrightarrow \exists z{:}\tau.\ R(x,z) \wedge S(z,y)\ )$ nicht in pränexer Normalform

$\forall x{:}\tau.\ \forall y{:}\tau.\ \exists z{:}\tau\ (\ T(x,y) \leftrightarrow R(x,z) \wedge S(z,y)\ )$ ist in pränexer Normalform.

Quantorenregeln

Die folgenden Quantorenumformungen führen zu äquivalenten Formeln:

(1) $\forall x{:}\tau.\ \varphi \wedge \forall x{:}\tau.\ \psi \quad \Leftrightarrow \quad \forall x{:}\tau.\ \varphi \wedge \psi$

(2) $\exists x{:}\tau.\ \varphi \vee \exists x{:}\tau.\ \psi \quad \Leftrightarrow \quad \exists x{:}\tau.\ \varphi \vee \psi$

(3) $\neg\, \forall x{:}\tau.\ \psi \quad \Leftrightarrow \quad \exists x{:}\tau.\ \neg\psi$

(4) $\neg\, \exists x{:}\tau.\ \psi \quad \Leftrightarrow \quad \forall x{:}\tau.\ \neg\psi$

Falls x nicht frei in φ :

(5) $\varphi \vee \forall x{:}\tau.\ \psi \quad \Leftrightarrow \quad \forall x{:}\tau.\ \varphi \vee \psi$

(6) $\varphi \vee \exists x{:}\tau.\ \psi \quad \Leftrightarrow \quad \exists x{:}\tau.\ \varphi \vee \psi$

(7) $\varphi \wedge \forall x{:}\tau.\ \psi \quad \Leftrightarrow \quad \forall x{:}\tau.\ \varphi \wedge \psi$

(8) $\varphi \wedge \exists x{:}\tau.\ \psi \quad \Leftrightarrow \quad \exists x{:}\tau.\ \varphi \wedge \psi$

Falls y nicht frei in ψ

(9) $\forall x{:}\tau.\ \psi \quad \Leftrightarrow \quad \forall y{:}\tau.\ \psi[x/y]$

(10) $\exists x{:}\tau.\ \psi \quad \Leftrightarrow \quad \exists y{:}\tau.\ \psi[x/y]$

Pränexe Normalform

- Jede Formel lässt sich in eine äquivalente Formel in pränexer Normalform umwandeln.
- Beweis: *Übung*.
- Beispiel: $\neg\exists x{:}\tau.\ \psi \vee \forall x{:}\sigma.\ \varphi \Leftrightarrow \forall x{:}\tau.\neg\psi \vee \forall x{:}\sigma.\varphi$

  wähle eine Variable $z{:}\sigma$, die weder in φ noch in ψ frei vorkommt:

  $\Leftrightarrow \forall x{:}\tau.\neg\psi \vee \forall z{:}\sigma.\ \varphi[x/z]$

  $\Leftrightarrow \forall x{:}\tau.\forall z{:}\sigma.\ \neg\psi \vee \varphi[x/z]$

Allquantoren und Existenzquantoren

- In einer Formel in pränexer Normalform können sich Allquantoren und Existenzquantoren abwechseln:

$$\forall x_0{:}\Re.\forall \varepsilon{:}\Re.\ \exists \delta{:}\Re.\ \forall x{:}\Re.\ |x-x_0| < \delta \ \Rightarrow\ |f(x)-f(x_0)| < \varepsilon$$

  - Offensichtlich kann δ von x_0 und von ε abhängen, nicht von x.

- Wenn die obige Formel richtig ist, so gibt es eine Funktion $\delta\colon \Re\times\Re \to\Re$, so dass die folgende Formel richtig ist:

$$\forall x_0{:}\Re.\forall \varepsilon{:}\Re.\ \forall x{:}\Re.|x-x_0| < \delta(x_0,\varepsilon) \ \Rightarrow\ |f(x)-f(x_0)| < \varepsilon$$

- Die beiden Formeln sind nicht äquivalent, es gilt aber:

  - Wenn eine der Formeln erfüllbar ist, dann auch die andere.
  - Man sagt: sie sind: erfüllungsäquivalent.

Skolemisierung

Thoralf Skolem

- Zu jeder Formel φ in pränexer Normalform lässt sich eine offene Formel ψ finden, so dass gilt:

  φ hat ein Modell gdw. ψ hat ein Modell

  d.h.

  φ ist erfüllbar gdw. ψ ist erfüllbar.

Beweis:

$\forall x_1{:}\tau_1.\ \forall x_2{:}\tau_2.\ \ldots\ \forall x_n{:}\tau_v.\ \exists\ y{:}\sigma.\ \Theta$

ist genau dann erfüllbar, wenn

$\forall x_1{:}\tau_1.\ \forall x_2{:}\tau_2.\ \ldots\ \forall x_n{:}\tau_v.\ \Theta\ [y\ /\ f(x_1, x_2, \ldots\ x_n)]$

erfüllbar ist.

Vorbereitung einer Formel für Resolventenmethode:

- Sei eine ψ Formel, die bewiesen werden soll.
  Idee: Wir wollen zeigen, dass $\neg\psi$ unerfüllbar ist.

1. Wandle $\neg\psi$ um in eine äquivalente pränexe Normalform
2. $$Q_1 x_1{:}\tau_1 . Q_2 x_2{:}\tau_2 . \ldots Q_n x_n{:}\tau_n . \varphi .$$
   wobei $Q_i \in \{\forall, \exists\}$
3. Ersetze jeden Existenzquantor durch eine neue Skolemfunktion.
   Die neue Formel ist erfüllungsäquivalent und hat die Form
   $$\forall y_1{:}\sigma_1 . \forall y_2{:}\sigma_2 . \ldots \forall y_k{:}\sigma_n . \Phi$$
   wobei Φ eine offene Formel ist.
4. Wandle Φ in eine Menge C von Klauseln um.

- Es gilt:

  ψ allgemeingültig $\Leftrightarrow$ C nicht erfüllbar.

Resolventenmethode

- Um φ zu beweisen zeige die Widersprüchlichkeit einer Menge C von Klauseln. (Man benötigt, dass es für jeden Typ τ mindestens eine Konstante c_τ gibt.)
- Benutze die Regeln (κ, κ_1, κ_2 Klauseln, p Literal, t Term) :

$$\frac{\kappa_1 \cup \{ p \}, \quad \kappa_2 \cup \{\neg p\}}{\kappa_1 \cup \kappa_2}$$

Resolution

$$\frac{\kappa}{\kappa[x/t]}$$

Instanziierung

Resolventenmethode - Vorbereitung

Beweisaufgabe: $(\exists x{:}\sigma.\forall y{:}\tau.P(x,y)) \Rightarrow \forall y{:}\tau.\exists x{:}\sigma.P(x,y)$

- Erster Schritt: Negation
  - $(\exists x{:}\sigma.\forall y{:}\tau.P(x,y)) \wedge \neg\forall y{:}\tau.\exists x{:}\sigma.P(x,y)$
- Umwandeln in pränexe Normalform
  - $\exists x{:}\sigma.\forall y{:}\tau.\ \exists u{:}\tau.\forall z{:}\sigma.\ (P(x,y) \wedge \neg P(z,u))$
- Skolemisieren
  - $P(c,y) \wedge \neg P(z,f(y))$

  Kontextbedingung:
  $c \in \sigma$, $d \in \tau$, $z{:}\sigma$, $y{:}\tau$, $f{:}\tau \rightarrow \tau$
- Menge von Klauseln
  - $\{\{ P(c,y)\} , \{ \neg P(z,f(y)) \} \}$

  Kontextbedingung:
  $c \in \sigma$, $d \in \tau$, $z{:}\sigma$, $y{:}\tau$, $f{:}\tau \rightarrow \tau$

Resolventenmethode - Abschluss

- Menge von Klauseln
  - {{ P(c,y)} , {¬P(z,f(y))} }

Kontextbedingung:
$c \in \sigma$, $d \in \tau$, $z:\sigma$, $y:\tau$, $f:\tau \rightarrow \tau$

- Instanziiere zweite Klausel: z := c, y := d
  - ¬P(c,f(d))
- Instanziiere erste Klausel: y:= f(d)
  - P(c,f(d))
  -
- Bilde Resolvente
  - $\dfrac{P(c,f(d)) , \neg P(c,f(d))}{[]}$
- Leere Klausel erreicht. Q.E.D.

Verständnisfrage: Warum darf man y in der einen Klausel durch d und in der zweiten Klausel durch etwas anderes – f(d) ersetzen ?

Anfang eines Resolutionsbeweises

- Zeige:
  $\forall x.A(0,x,x) \wedge \forall x,y,z.\ (\ A(x,y,z) \rightarrow A(s(x),y,s(z))\ ) \quad \rightarrow \quad \exists\, x.\ A(s(s(0)),s(0),x))$
- Negation:
  $\forall x.A(0,x,x)$
  $\wedge\ \forall x,y,z.A(x,y,z) \rightarrow A(s(x),y,s(z))$
  $\wedge\ \ \forall x.\ \neg\ A(s(s(0)),s(0),x)$
- Menge von Klauseln:
  $\{\ A(0,x,x),\ \ \neg\ A(x,y,z) \vee A(s(x),y,s(z))\ ,\ \ \neg\ A(\ s(s(0)),\ s(0),\ x)\ \}$
- Entspricht Prolog-Programm:

mit Aufruf:

```
A(0,X,X)          ←
A(s(X),Y,s(Z))  ← A(X,Y,Z).
```

```
← A(s(s(0)),s(0),X).
```

Wiederholte Instanziierungen

κ_1: `A(0,x,x),`

κ_2: $\neg$ `A(x,y,z)` $\vee$ `A(s(x),y,s(z))` ,

κ_3: $\neg$ `A(s(s(0)),s(0),x)`

Instanziiere κ_3 mit x:= s(s(s(0)) :

κ_4: $\neg$ `A(s(s(0)),s(0),s(s(s(0))))`

Instanziiere κ_2 mit x:= s(0), y:=s(0), z:= s(s(0)) :

κ_5: $\neg$ `A(s(0),s(0),s(s(0))`$\vee$`A(s(s(0)),s(0),s(s(s(0))))`

Resiolviere κ_4 mit κ_5 :

κ_6: $\neg$ `A(s(0), s(0), s(s(0)))`

Instanziiere κ_2 erneut, diesmal mit x:= 0, y:= s(0), z:= s(0) :

κ_7: $\neg$ `A(0,s(0),s(0))` $\vee$ `A(s(0),s(0),s(s(0)))`

Resolvente mit κ_6 ergibt

κ_8: $\neg$ `A(0,s(0),s(0))`

Instanziiere κ_1 mit x:= s(0)

κ_n: `A(0,s(0),s(0))`

Resolvente von κ_8 mit κ_9 ergibt leere Klausel.

```
A(0,X,X)          ←
A(s(X),Y,s(Z))  ← A(X,Y,Z).
```

```
← A(s(s(0)),s(0),X).
```

Wie findet man eine Instanziierung

- Gegeben: Zwei Klauseln $\kappa_1 \cup \{p\}$, $\kappa_2 \cup \{\neg q\}$
- Gesucht: Instanziierungen *, $^+$ so daß $p^* = q^+$. Dann kann man resolvieren und erhält als neue Klausel:

$$\kappa_1{}^* \cup \kappa_2{}^+$$

- Beispiel: $\kappa_1 = \{ p\} = \{P(c,y)\}$ und $\kappa_2 = \{\neg q\} = \{\neg P(z,f(y))\}$

Wir machen die Variablen disjunkt (wieso dürfen wir das?) und erhalten

$\kappa_1 = P(c,x_1)$ und $\kappa_2 = \neg P(x_2,f(x_3))$

Ein Vergleich legt die Substitution

$x_2=c$, $x_1=f(x_3)$ nahe.

Wir erhalten als „allgemeinste Substitution"

$\kappa_1 = \{ P(c,f(x_3)) \}$ $\kappa_2 = \{\neg P(c,f(x_3)) \}$

Substitutionen

Eine typkorrekte Substitution ρ ordnet einer Menge

$$M=\{x_1:\tau_1, x_2:\tau_2, \ldots, x_n:\tau_n\}$$

von Variablen eine Menge von Termen $t_1:\tau_1, \ldots, t_n:\tau_n$ zu. Man schreibt die Substitution rechts an den Term, also $x_1\sigma = t_1, \ldots, x_n\sigma = t_n$.

M heißt Definitionsbereich von σ (Schreibweise: M = dom σ).

Man kann σ auf **alle** Variablen fortsetzen, indem man für Variablen $x \notin M$ setzt:

$$x\,\sigma = x.$$

Durch die Definition

$$f(t_1, \ldots, t_n)\,\rho \;:=\; f(t_1\rho, \ldots, t_n\rho)$$

lässt sich jede Substitution ρ eindeutig auf Terme fortsetzen.

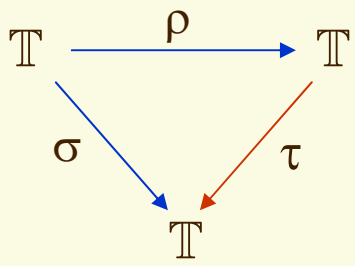
Lemma: Für jede Substitution ρ wie oben und für jeden Term t gilt:

$$t\rho = t[x_1/t_1, \ldots, x_n/t_n]$$

Beweis: Induktion über Aufbau von t.

Allgemeiner als ...

- Seien ρ, σ Substitutionen. ρ heißt *allgemeiner* als σ, falls es eine Substitution τ gibt mit $\sigma = \tau \circ \rho$.



- Substitutionen ρ und σ heißen *äquivalent*, falls ρ allgemeiner als σ und σ allgemeiner als ρ ist.

- Eine Substitution τ heißt *Umbenennung*, falls τ injektiv ist und Variablen in Variablen abbildet, also $x_i\tau = x_k$ gilt.

- Lemma: ρ und σ sind genau dann äquivalent, falls es eine Umbenennung τ gibt mit $\rho = \tau \circ \sigma$.

Unifikatoren

- Sei $S=\{(s_1,t_1)\ldots , (s_n, t_n)\}$ eine Menge von Termpaaren (paarweise gleichen Typs). Eine Substitution ρ heißt Unifikator für S falls
  $s_1 \rho = t_1\rho, \ldots, s_n \rho = t_n\rho$.

- ρ heißt allgemeinster Unifikator für S, falls ρ allgemeiner als jeder andere Unifikator von S ist.

- Satz: Gibt es einen Unifikator von S, so gibt es auch einen allgemeinsten Unifikator von S.

- Den allgemeinsten Unifikator eines Paares (s,t) von Termen nennt man auch mgu(s,t) *„most general unifier"*.

Beispiel einer Unifikation

Statt des Paars (s,t) schreibt man meist s=t:

1. { f(x,g(0,y),z) = f(h(u,v),g(u,k(z)),k(x)) }
2. { x=h(u,v), g(0,y)=g(u,k(z)), z=k(x) }
3. { x=h(u,v), 0=u, y=k(z), z=k(x) }
4. { x=h(u,v), u=0, y=k(z), z=k(h(u,v)) }
5. { x=h(0,v), u=0, y=k(z), z=k(h(0,v)) }
6. { x=h(0,v), u=0, y=k(k(h(0,v))), z=k(h(0,v)) }

Der allgemeinste Unifikator von

f(x,g(0,y),z) und f(h(u,v),g(u,k(z)),k(x))

ist ρ mit

$x\rho$ = h(0,v), $u\rho$ = 0, $y\rho$ = k(k(h(0,v))), $z\rho$ = k(h(0,v)).

Allgemeinster Unifikator - mgu

- Wenn ein Unifikator für s und t existiert, dann auch mgu(s,t).
- Ein Unifikator für s und t existiert nicht, falls bei der Konstruktion des Gleichungssystems
  - eine Gleichung der Art $x = t(\ldots,x,\ldots)$
  - eine Gleichung zwischen zwei Termen
    $f(t_1,\ldots,t_n) = g(s_1,\ldots,s_k)$ mit $f \neq g$ oder $n \neq k$

  entsteht
- Beispiele:
  - f(g(x,y),h(x)) und f(z,h(z)) sind nicht unifizierbar.
    Es entsteht die Gleichung $z = g(z,y)$
  - f(g(x,y),h(x)) und f(z,g(y,z)) sind nicht unifizierbar.
    Es entsteht die Gleichung $h(x) = g(y,z)$

Unifikationsalgorithmus

Seien $S=\{(s_1,t_1), \ldots , (s_n,t_n)\}$ eine Menge von Termpaaren. Wir suchen einen allgemeinsten Unifikator der (s_i,t_i).

REPEAT

Entferne jedes Paar der Form (x,x)

Falls t keine Variable ist, ersetze (t,x) durch (x,t)

Occur Check: Ist (x,t) in S und kommt x in t vor, goto **Abbruch**.

Ist (x,t) aus S, ersetze alle anderen Paare (s_i,t_i) durch $(s_i[x/t],t_i[x/t])$

Ist $(f(s_1,\ldots,s_m),g(t_1,\ldots,t_k))$ aus S, und $f\neq g$ oder $m\neq k$, goto **Abbruch**.

Ersetze $(f(s_1,\ldots,s_k),f(t_1,\ldots,t_k))$ durch $(s_1,t_1), \ldots, (s_k,t_k)$

UNTIL $S=\{(x_1,r_1), \ldots (x_m,r_m)\}$ und kein x_i kommt in einem r_j vor.

Lemma: Das Programm ist korrekt, hält immer, und σ mit $x_1\sigma = r_1, \ldots x_m\sigma = r_m$, $y\sigma = y$ für jede andere Variable y ist allgemeinster Unifikator.

Beweis: (1.) Ein Programmschritt führe S in S' über. Dann gilt für jede Substitution σ: σ unifiziert S gdw. σ unifiziert S'. (2.) Ist τ ein Unifikator von $S=\{(x_1,r_1), \ldots (x_n,r_n)\}$, dann gilt $x_1\tau = r_1\tau, \ldots , x_n\tau = r_n\tau$, und $y\tau = y$ für jede andere Variable y. Es folgt $x_1\tau = x_1\sigma\tau, \ldots , x_n\tau = x_n\sigma\tau$ und $y\tau = y\sigma\tau$ für jede andere Variable y, also $\tau = \sigma\tau$.

Das Programm hält, weil der Wert

(U(S),L(S),R(S)) lexikographisch immer kleiner wird. Dabei ist U(S) die Anzahl der ungelösten Variablen, $L(S) = \Sigma_i |s_i|$, $R(S) = |\{(s_i,t_j) \mid t_i \text{ ist Variable}\}|$

Abbruch: Unifikation unmöglich.

Resolventenmethode mit Unifikation

- Gegeben: Zwei Klauseln $\kappa_1 \cup \{p\}$, $\kappa_2 \cup \{\neg q\}$
- Die Variablen in $\kappa_1 \cup \{p\}$ und $\kappa_2 \cup \{\neg q\}$ dürfen umbenannt werden
- Bilde $\rho = mgu(p, q)$ und resolviere $\rho(\kappa_1)$ mit $\rho(\kappa_2)$

$$\frac{\kappa_1 \cup \{p\}, \quad \kappa_2 \cup \{\neg q\}}{(\kappa_1 \cup \kappa_2)\rho}$$

wobei $\rho = mgu(p,q)$

Notwendigkeit des Occur-Check

Offensichtlich gilt nicht:

$\forall x.\exists y.P(x,y) \rightarrow \exists x.P(x,x)$

Wenn man bei der Unifikation den occur-check weglassen würde, könnte man letzteres beweisen:

- Negation liefert $\forall x.\exists y.P(x,y) \wedge \forall x.\neg P(x,x)$
- Skolemisierung liefert die Klauseln
- $P(x,f(x))$, $\neg P(u,u)$
- Unifikation ohne occur-check liefert „*Unifikator*“ σ mit $x\sigma = u$, $u\sigma = f(u)\sigma$. Und damit wird (scheinbar) die leere Klausel gewonnen.

- Fast alle Prolog-Implementierungen lassen den occur-Check weg, da er aufwendig ist. In der Praxis ist das erträglich –
  - Aber: es kann Probleme geben.

Prolog Programm

- Ein *Prolog Programm* ist eine Menge von Implikationen

```
add(0,x,x) ←                          % 0+x=x
add(s(x),y,s(z)) ← add(x,y,z)         % x+y=z => s(x)+y=s(z)
```

- Der *Aufruf* eines Prolog-Programms geschieht mit einer Zielklausel, z.B.:

```
← add(s(0), s(s(0)), x)               % s(s(0))+s(0) = x
← add(s(0),y, s(s(0)))                % y = s(s(0))-s(0)
```

- Charakteristisch für ein Prolog-Programm:
  - Alle Programmklauseln enthalten genau ein positives Literal
  - Die Goal-Klausel enthält kein positives Literal
- Klauseln mit höchstens einem positiven Literal heißen auch „Horn-Klauseln"
- *Prolog* ist eine ernstzunehmende Programmiersprache. (http://www.pdc.dk)

Ausführung des Prolog-Aufrufs

- π_1: a(0,x,x), % Programmklausel
  π_2: $\neg$ a(x,y,z) $\vee$ a(s(x),y,s(z)) , % Programmklausel
  γ_3: $\neg$ a(s(s(0)), s(0), u) % Goalklausel

- Resolviere γ_3 mit π_2 : mgu(a(s(x_1),y_1,s(z_1)), a(s(s(0)),s(0),u) ist
  Substitution x_1:=s(0), y_1:=s(0), u:= s(z_1)
  Resolvente ist: κ_4= $\neg$ a(s(0),s(0),z_1)

- Resolviere κ_4 mit π_2: mgu(a(s(0),s(0),z_1), a(s(x_2),y_2,s(z_2))) ist die
  Substitution x_2:=0, y_2:=s(0), z_1=s(z_2)
  Resolvente ist: κ_5= $\neg$ a(0,s(0),z_2)

- Resolviere κ_5 mit π_1: mgu(a(0,s(0),z_2), a(0,x_3, x_3)) ist die
  Substitution x_3:=s(0), z_2=s(0)
  Resolvente ist die leere Klausel

- Ergebnis: u = s(z_1) = s(s(z_2)) = s(s(s(0))).

Ist Resolution vollständig ?

- Die folgende Regel *allein* ist nicht vollständig :

Resolution $\dfrac{\kappa_1 \cup \{p\},\ \ \kappa_2 \cup \{\neg q\}}{(\kappa_1 \cup \kappa_2)\rho}$ $\rho = \text{mgu}(t_1, t_2)$

- Beispiel: $\kappa_1 = P(x) \vee P(y)$, $\kappa_2 = \neg P(u) \vee \neg P(v)$
- Wir benötigen eine weitere Regel:

Faktorisierung: $\dfrac{\kappa_1 \cup \{p,q\}}{(\kappa_1 \cup \{p\})\rho}$ $\rho = \text{mgu}(p,q)$

- Zusammen mit der Faktorisierung ist die Resolventenmethode vollständig.

Prädikatenlogische Beispiele - 1

Relationen: `hoert, student_von, kennt`

Operation : `dozent`

- `hoert(otto,inf1,ss02).`
- `hoert(chris,inf2,ss02).`
- `hoert(eva,inf1,ss02).`

- `student_von(eva, sommer).`
- `student_von(eva, hesse).`
- `student_von(x,dozent(u,v)) ← hoert(x,u,v).`

- `kennt(x,y) ← student_von(x,u), student_von(y,u)`

- `dozent(inf1,ss02) = hesse`
- `dozent(inf2,ss02) = sommer`

Folgt daraus :

- `kennt(otto,eva) ?`
- `kennt(eva, chris) ?`

Prädikatenlogische Beispiele - 2

Die Konjunktion der folgenden Aussagen spezifiziert den ggT in Nat:

- $\forall x,y \in \text{Nat}.\ x = y \rightarrow ggT(x,y) = x$
- $\forall x,y \in \text{Nat}.\ x > y \rightarrow ggT(x,y) = ggT(x-y,y)$
- $\forall x,y \in \text{Nat}.\ x < y \rightarrow ggT(x,y) = ggT(x,y-x)$
- Aus diesen Formeln und den Axiomen für <, = und – folgt u.a.

  $ggT(36,45) = 9.$

Derartige logische Beschreibungen stellen **Prolog-Programme** dar:

```
ggT(x,x,x) :- !.
ggt(x,y,z) :- x > y, ggT(x-y,y,z).
ggT(x,y,z) :- x < y, ggT(x,y-x,z).
```

Man startet das Programm mit der Eingabe

```
? ggT(36,45,U).
```