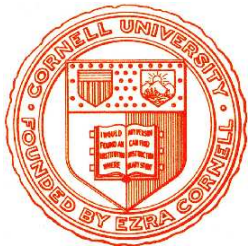
Automatisierte Logik und Programmierung

Einheit 9



Formale Inferenz in der Typentheorie

1. Verifikation und Evidenzkonstruktion
2. Universen und Gleichheitstypen
3. Formale Regeln und Beweise
4. Einbettung der Prädikatenlogik
5. Entwurfsentscheidungen im Rückblick

Ein Refinement Kalkül für die Typentheorie

Beweisen durch Verfeinerung typentheoretischer Urteile

- **Syntaktische Simulation semantischer Schlüsse**
  - Formaler Nachweis, daß bestimmte Urteile gefällt werden können
  - Inferenzregeln ersetzten semantische Argumente
- **Beweise operieren zielorientiert**
  - Oberstes Beweisziel ist die Behauptung, daß ein bestimmtes Urteil tatsächlich gefällt werden kann
  - Inferenzregeln verfeinern ein Ziel in Teilziele, welche den Voraussetzungen für das Fällen des semantischen Urteils entsprechen
  - Verfeinerungskalkül unterstützt Suche nach Beweisen
- **Beweisziele müssen hypothetische Urteile repräsentieren**
  - Syntaktisches Gegenstück ist Sequenz $x_1{:}T_1, .., x_n{:}T_n \vdash J$
  - Initialsequenz $\vdash J$ beschreibt Urteil ohne Annahmen

Unterstützung für alle Aspekte des Schliessens

- **Es gibt verschiedene Zielrichtungen formaler Beweise**
  - Typzugehörigkeitsprüfung: zeige $t \in T$ (Type-Checking, Verifikation)
  - Gleichheitsschließen: zeige $s = t \in T$ (Auswertung von Programmen)
  - Witness-Finding zeige Existenz eines Terms t mit $t \in T$
    (Synthese von Programmen, Evidenz für logische Aussagen, vgl. §3)
  - Schließen über Typ-Sein und Typ-Gleichheit
- **Beweiskalkül muß einheitlichen Formalismus anbieten**
  - Gleichheitsurteile subsumieren Typ-Sein und Typzugehörigkeit
  - Universen können Typ-Eigenschaft syntaktisch darstellen (Folie 3)
    aber Girard's Paradox muß vermieden werden
  - Alle vier Urteilsklassen sind durch Gleichheitsurteil $s = t \in T$ darstellbar
- **Gleichheit kann Witness-Finding nicht darstellen**
  - Kalkül soll Evidenz/Programme konstruieren und nicht nur überprüfen
  - Beweisziel $\vdash T$ (*Konstruiere Element des Typs* T) wäre die beste Form
  - Urteil $s = t \in T$ kann durch Gleichheitstyp repräsentiert werden (Folie 5)

Universen als Datentypen

Syntaktische Repräsentation des Konzepts “Typ-Sein”

- **Typ-Sein ist ein semantisches Konzept**
  - Beschreibung der Rolle eines Ausdrucks als Repräsentant eines Typs
  - Abhängigkeiten werden semantisch charakterisiert
    $x{:}S{\to}T$ *Typ*, falls S *Typ* und $T[s/x]$ *Typ* für alle $s \in S$
  - Voraussetzungen beziehen sich immer auf konkrete Typen
    Typ-wertige Funktionen sind nicht erforderlich (aber nützlich)
- **Beweiskalkül benötigt syntaktische Simulation**
  - T *Typ* könnte syntaktisch durch $T \in \mathbb{U}$ beschrieben werden
  - Ausdruck $\mathbb{U}$ ist Repräsentant des Universums aller Typen
- **Semantik muß Ausdruck $\mathbb{U}$ eine Bedeutung geben**
  - Formulierung $T \in \mathbb{U}$ verlangt, daß $\mathbb{U}$ selbst ein Typ ist (§8, Folie 10)
  - $\mathbb{U} \in \mathbb{U}$ darf nicht gelten, also ist $\mathbb{U}$ Element eines höheren Universums
  - Liefert Universenhierarchie $\mathbb{U} = \mathbb{U}_1 \in \mathbb{U}_2 \in \mathbb{U}_3 \in \ldots$
  - Klassifizierung des Typs typ-wertiger Funktionen $f{:}S{\to}\mathbb{U}_i$ verlangt kumulative Universenhierarchie: $\mathbb{U} = \mathbb{U}_1 \overset{\in}{\subset} \mathbb{U}_2 \overset{\in}{\subset} \mathbb{U}_3 \overset{\in}{\subset} \ldots$

Formulierung kumulativer Universen

Syntaktische Repräsentation der Typ-Eigenschaft

- **Ausdrücke**

  Kanonisch: $\mathbb{U}_j$ j natürliche Zahl oder Variable

  Nichtkanonisch: —

- **Reduktion von Ausdrücken**

  – entfällt, da keine spezifischen Terme für Universen existieren

- **Urteile für Typ- und Elementgleichheit**

$S = T$	falls	$S{=}T \in \mathbb{U}_j$
$\mathbb{U}_i = \mathbb{U}_j$	falls	i identisch mit j *(als natürliche Zahl bzw. Variable)*
$x_1{:}S_1{\to}T_1 = x_2{:}S_2{\to}T_2 \in \mathbb{U}_j$	falls	$S_1{=}S_2 \in \mathbb{U}_j$ und $T_1[s_1/x_1]{=}T_2[s_2/x_2] \in \mathbb{U}_j$ für alle Terme s_1, s_2 mit $s_1{=}s_2 \in S_1$.
$x_1{:}S_1{\times}T_1 = x_2{:}S_2{\times}T_2 \in \mathbb{U}_j$	falls	$S_1{=}S_2 \in \mathbb{U}_j$ und $T_1[s_1/x_1]{=}T_2[s_2/x_2] \in \mathbb{U}_j$ für alle Terme s_1, s_2 mit $s_1{=}s_2 \in S_1$.
⋮		— *analog für alle anderen Datentypen* —
$\{\} = \{\} \in \mathbb{U}_j$		— *für alle j !* —
$\mathbb{U}_i = \mathbb{U}_k \in \mathbb{U}_j$	falls	i identisch mit k und k kleiner als j *(als Zahl)*

Kumulativität wird dadurch erreicht, daß Basistypen zu jedem $\mathbb{U}_j$ gehören

Propositionale Gleichheit

Integration von Gleichheit in logische Aussagen

- **Fundamental für fast alle mathematischen Schlüsse**
  - Die meisten logischen Prädikate sind durch Gleichheiten zu beschreiben
- **Nicht dasselbe wie definitorische Gleichheit $A \equiv B$**
  - Definitorische Gleichheit ist ein syntaktisch-linguistisches Konzept
    Zwei Ausdrücke werden als identisch deklariert
  - Propositionale Gleichheit ist Gleichheit von Werten, z.B. `2+2 = 2`^{2}
- **Nicht dasselbe wie Gleichheitsurteile $s{=}t \in T$**
  - Gleichheitsurteile stellen die Gültigkeit einer Gleichheit fest
  - Propositionale Gleichheiten sind nur Behauptungen, die auch ungültig sein können, z.B. `2+2 = 3`
  - Propositionale Gleichheiten können mit Konnektiven verbunden werden

Formalisierung propositionaler Gleichheit

- **Essentiell für Argumente mit Gleichheitsannahmen**
  - Viele Typdefinitionen haben Voraussetzungen, in den hypothetische Urteile mit Gleichheitsanahmen verwendet werden

    $\lambda x_1.t_1{=}\lambda x_2.t_2 \in x{:}S{\to}T$ falls $t_1[s_1/x_1]{=}t_2[s_2/x_2]\in T[s_1/x]$ für alle $s_1{=}s_2\in S$
  - Logische Schlüsse verwenden zuweilen komplexe Gleichheitsannahmen

    "Aus $\forall x{:}\mathbb{N}.f(x){=}x \Rightarrow f(x{+}1){=}x{+}1$ und $f(0){=}0$ folgt $\forall n{:}\mathbb{N}.f(n){=}n$"
  - Hypothetische Urteile benötigen mehr als Typzugehörigkeitsannahmen
- **Propositionale Gleichheit als Datentyp**
  - Die Annahme $s{=}t\in T$ bedeutet in Wirklichkeit die Annahme, Evidenz für diese Gleichheit zu besitzen, also ein Element eines Gleichheitstyps
  - Die konkrete Evidenz ist eigentlich irrelevant
  - Ein Gleichheitstyp ermöglicht eine Formulierung komplexer Annahmen
- **Definiere Gleichheitstyp** $s{=}t$ `in` T
  - Syntaktische Darstellung ist bewußt ähnlich zum Gleichheitsurteil
  - Es gibt nur ein kanonisches Element Ax, das nicht von s, t, T abhängt
  - Es gibt keine nichtkanonischen Terme (Evidenz wird nicht analysiert)

Formulierung eines Gleichheitstyps

Schließen mit Werten von Ausdrücken

- **Ausdrücke***

  Kanonisch: $s{=}t$ `in` T s, t und T Terme

  Ax

  Nichtkanonisch: —

- **Reduktion von Ausdrücken**

  – entfällt, da keine nichtkanonischen Terme für Gleichheitstypen

- **Urteile für Typ- und Elementgleichheit**

  $s_1{=}t_1$ `in` T_1 = $s_2{=}t_2$ `in` T_2 falls $T_1{=}T_2$, $s_1{=}s_2{\in}T_1$, und $t_1{=}t_2{\in}T_1$

  Ax = Ax $\in$ $s{=}t$ `in` T falls $s{=}t{\in}T$

*: In Nuprl wird der Gleichheitstyp als $s = t \in T$ dargestellt ($\in$ statt `in`)
Wir verwenden diese Form, wenn keine Gefahr der Verwechselung mit Urteilen besteht

Anwendungen der Propositionalen Gleichheit

- **Der leere Datentyp kann konservativ definiert werden**
  - $\{\} \equiv \texttt{X:}\mathbb{U}\texttt{→(X=X→X in }\mathbb{U}\texttt{)}$ kann keine Elemente haben
- **Seltsame Eigenschaften von $\{\}$ gelten auch für Simulation**

  Für jeden Typ T gilt $\lambda z.(\lambda x.\, x\, x)(\lambda x.\, x\, x) \in \{\}\to T$
  - Es gilt $t = \lambda z.\, (\lambda x.\, x\, x)\, (\lambda x.\, x\, x) \in \{\}\to T$ für jedes beliebige T, da $t \in \{\}\to T$ falls $\{\}\to T$ *Typ* und $(\lambda x.\, x\, x)\, (\lambda x.\, x\, x) \in T$ für alle $s \in \{\}$
  - Sei s Element von $\{\} = \texttt{X:}\mathbb{U}\texttt{→(X=X→X in }\mathbb{U}\texttt{)}$
  - Dann ist $s(T)$ Element von $T\texttt{=}T\to T\ \texttt{in}\ \mathbb{U}$ und reduziert zu Ax
  - $\mathsf{Ax} \in T\texttt{=}T\to T\ \texttt{in}\ \mathbb{U}$ bedeutet $T = T\to T \in \mathbb{U}$, also $T = T\to T$
  - Es ist $(\lambda x.\, x\, x)(\lambda x.\, x\, x) \in T$, wenn $\lambda x.\, x\, x$ zu T und zu $T\to T$ gehört
  - $\lambda x.\, x\, x \in T\to T = T$ gilt, falls $x\, x \in T$ für alle $x \in T$
  - $x\, x \in T$ gilt, falls $x \in T\to T$ und $x \in T$
  - Wegen $T = T\to T$ und der Annahme $x \in T$ ist beides der Fall

Darstellung von Urteilen im Beweiskalkül

- **Universen und Gleichheit ermöglichen einheitliche Form**
  - Aufgabe von Beweisen wird, Elemente eines Typs zu konstruieren
    Beweisziel ist Nachweis von $\vdash T$ "*Typ* T *besitzt Elemente*"
  - Verifikation von Gleichheiten und Typzugehörigkeit ist simulierbar
    Beweisziel wird Nachweis von z.B. $\vdash s{=}t \texttt{ in } T$
    Tatsächliche Aufgabe ist also, Evidenz für $s{=}t \in T$ zu finden
- **Konstruktion von Elementen durch Verfeinerung**
  - Beweisziel $\vdash T$ wird in Teilziele zerlegt
  - In atomaren Zielen kann ein Element von T direkt konstruiert werden
  - Ansonsten wird Element aus denen der Unterziele zusammengesetzt
  - Konkretes Element des Hauptziels kann erst beschrieben werden,
    wenn der Beweis vollständig vollendet wurde
  - Element kann (nachträglich) aus Beweis extrahiert werden
- **Darstellung von Beweiszielen (in Regeln) ist $\vdash T \lfloor \mathsf{ext}\ t \rfloor$**
  - Ziel ist einen noch unbekannten Term t mit $t \in T$ zu konstruieren

Unterstützte Formen logischer Schlüsse

Sechs Arten von Aufgaben können realisiert werden

- **Verifikation: Evidenz für das Fällen von Urteilen**
  - *Typ-Sein "T Typ"*: $\vdash\ T{=}T\ \texttt{in}\ \mathbb{U}_j\ \lfloor\text{ext}\ p\rfloor$

    (Für ein frei gewähltes j zeige Existenz eines Terms p mit $p \in T{=}T\ \texttt{in}\ \mathbb{U}_j$)
  - *Typgleichheit "$S{=}T$"*: $\vdash\ S{=}T\ \texttt{in}\ \mathbb{U}_j\ \lfloor\text{ext}\ p\rfloor$
  - *Typzugehörigkeit "$t \in T$"*: $\vdash\ t{=}t\ \texttt{in}\ T\ \lfloor\text{ext}\ p\rfloor$
  - *Elementgleichheit "$s{=}t \in T$"*: $\vdash\ s{=}t\ \texttt{in}\ T\ \lfloor\text{ext}\ p\rfloor$
- **Synthese: Konstruktion von Termen**
  - *Konstruktion von Datentypen*: $\vdash\ \mathbb{U}_j\ \lfloor\text{ext}\ T\rfloor$
  - *Konstruktion von Elementen eines Typs*: $\vdash\ T\ \lfloor\text{ext}\ t\rfloor$

    (Zeige Existenz eines Terms t mit $t \in T$)

(Neue) Darstellung von Sequenzen im Kalkül

- **Sequenz:** $H \vdash T$ $\lfloor$ext $t\rfloor$

  "*Unter der Annahme, daß x_i Variablen vom Typ T_i sind, ist T nicht leer (inhabited)*"

  – $H = x_1{:}T_1, .., x_n{:}T_n[x_1, .., x_{n-1}]$ Hypothesenliste, T, t Terme

  – $x_i : T_i$ Deklaration "x_i ist Variable vom Typ T_i" (x_i Variable, T_i Term)

  – Notation T $\lfloor$ext $t\rfloor$ drückt aus, daß t unbekannt (und unsichtbar) ist

  Geschlossene Sequenz:

  – Jede freie Variable x in T oder T_i ist zuvor durch $x : T_j$ deklariert

  Reine Sequenz:

  – Geschlossene Sequenz, in der jede Variable nur einmal deklariert ist

  Initialsequenz: Geschlossene Sequenz der Form $\vdash T$ ohne Hypothesen

- **Evidenzterm für** $H \vdash T$

  – Term t mit freien Variablen aus den x_i, so daß $t \in T$ gilt,
  wenn alle x_i mit Elementen $t_i \in T_i$ instantiiert werden

  Eine Sequenz ist gültig, wenn es einen Evidenzterm für sie gibt

Definition direkt umsetzbar in Implementierung

Inferenzregeln simulieren Definition der Urteile

- **Verfeinerung eines Verifikationsziels** (Gleichheitstyp in Nuprl Darstellung)

$$\frac{H \vdash \lambda x_1 . t_1 = \lambda x_2 . t_2 \in x{:}S \to T \qquad H \vdash S \in \mathbb{U}_j \qquad H, x'{:}S \vdash t_1[x'/x_1] = t_2[x'/x_2] \in T[x'/x]}{\texttt{lambdaEquality}\ j\ x'}$$

$\lambda x_1.t_1 = \lambda x_2.t_2 \in x{:}S \to T$, *falls* $x{:}S \to T$ *Typ und* $t_1[s_1/x_1] = t_2[s_2/x_2] \in T[s_1/x]$ *für alle* s_1, s_2 *mit* $s_1 = s_2 \in S$

  - Regel vereinfacht $x{:}S \to T$ *Typ* zu "S *Typ* und $T[s_1/x]$ *Typ* für alle $s_1 \in S$"
  - Letzteres ist bewiesen, wenn $t_1[s_1/x_1] = t_2[s_2/x_2] \in T[s_1/x]$ gezeigt ist
  - S *Typ* wird durch Wahl des Universums (nur!) hinreichend repräsentiert

- **Aufbau von Evidenz**

$$\frac{H \vdash x{:}S \to T \quad \lfloor \text{ext } \lambda x' . t \rfloor \qquad H, x'{:}S \vdash T[x'/x] \quad \lfloor \text{ext } t \rfloor \qquad H \vdash S \in \mathbb{U}_j \quad \lfloor \text{Ax} \rfloor}{\texttt{lambdaFormation}\ j\ x'}$$

  - Evidenz für Teilziele wird zu Evidenz für Hauptziel zusammengesetzt
  - Evidenz für Verifikation von $S \in \mathbb{U}_j$ hat keine konstruktive Bedeutung

  $\lfloor \text{ext Ax} \rfloor$ wird kurz als $\lfloor \text{Ax} \rfloor$ geschrieben (oder ganz weggelassen)

(Neue) Darstellung von Regeln

- **(Verfeinerungs-)Regel** (dec, val)
  - dec Dekomposition: Abbildung von Sequenzen in Liste von Sequenzen
    (Verfeinerung eines Beweisziels in Liste von Teilzielen)
  - val Validierung: Abbildung von Liste von Sequenzen und Termen in Terme
    (Konstruktion von Evidenz für Beweisziel aus Evidenzen für Teilziele)
  - Regeln werden als Regelschemata dargestellt mit Metavariablen
    als Platzhaltern für Formeln und Evidenzterme
  - Konkrete Regeln entstehen hieraus durch Instantiierung der Metavariablen
- **Korrektheit einer Regel** $r = (dec, val)$

  Sei $S_0 = H \vdash T$ eine reine Sequenz und seien $S_i = H_i \vdash T_i$ die Resultate der Anwendung von dec auf S_0. Dann gilt
  - Alle Teilzielsequenzen S_i sind rein
  - Wenn alle S_i gültig sind, dann ist auch S_0 gültig
  - Sind t_i Evidenzterme für S_i, dann ist $t = val(S_0, (S_i, t_i))$ Evidenzterm für S_0

Korrektheit der Regel `lambdaFormation`

$$\frac{\begin{array}{ll} H \vdash x{:}S{\to}T & [\text{ext } \lambda x'.\, t] \\ H, x'{:}S \vdash T[x'/x] & [\text{ext } t] \\ H \vdash S \in \mathbb{U}_j & [\mathsf{Ax}] \end{array}}{}\ \texttt{lambdaFormation}\ j\ x'$$

– Sei $S_0 = H \vdash x{:}S{\to}T$ eine reine Sequenz und $S_1 = H, x'{:}S \vdash T[x'/x]$ und $S_2 = H \vdash S \in \mathbb{U}_j$

– Alle freien Variablen in H und in $x{:}S{\to}T$ sind genau einmal deklariert. In T kommt x eventuell frei vor, in S nicht. x' ist neu und in $T[x'/x]$ genau einmal deklariert. Damit sind S_1 und S_2 rein.

– Sei t Evidenzterme für S_1 und $t' = \mathsf{Ax}$ Evidenzterm für S_2. Dann ist $t \in T[s/x]$ (= $T[x'/x][s/x']$) für alle $s \in S$.

Per Definition ist dann auch $\lambda x'.t \in x{:}S{\to}T$ und damit ist auch $\lambda x'.t = val(S_0, (S_1, t), (S_2, \mathsf{Ax}))$ Evidenzterm für S_0.

Es folgt, daß S_0 gültig ist, wenn S_1 und S_2 gültig sind

Darstellung von Beweisen nahezu unverändert

- **Beweise**
  - Jede Sequenz $S = H \vdash C$ ist unvollständiger Beweis mit Wurzel S
  - $(S, r, [\pi_1, \ldots, \pi_n])$ ist Beweis mit Wurzelsequenz S,
    wenn $r = (dec, val)$ eine korrekte Regel ist und π_i Beweise für alle Sequenzen S_i sind, die durch Anwendung von dec auf S entstehen
  - vollständiger Beweis: Beweis ohne unvollständige Teilbeweise
  - Theorem: vollständiger Beweis, dessen Wurzel eine Initialsequenz ist
- **Extraktterm $ext(\pi)$ eines vollständigen Beweises**
  - $ext(S, (dec, val), [\pi_1, \ldots, \pi_n]) = val([S, (S_1, ext(\pi_1)), .., (S_n, ext(\pi_n))])$,
    wobei S_i Wurzeln der π_i
    Für $\pi = (S, (dec, val), [])$ liefert dies den Spezialfall $ext(\pi) = val([S])$

(vgl. §3, Folie 21)

Systematik für Aufbau des Regelsystems

Jeder Datentyp benötigt vier Arten von Regeln

- **Typ-Formationsregeln:**
  - Wann sind zwei Typen gleich (*typ*`Equality`) $H \vdash S = T \in \mathbb{U}_j$ *
  - Wie ist ein Typ zusammengesetzt? (*typ*`Formation`) $H \vdash \mathbb{U}_j \ [\text{ext } T]$
- **Kanonische Regeln:**
  - Wann sind kanonische Elemente gleich? (*element*`Equality`) $H \vdash s = t \in T$ *
  - Wie sind Elemente zusammengesetzt? (*element*`Formation`) $H \vdash T \ [\text{ext } t]$
- **Nichtkanonische Regeln:**
  - Wann sind nichtkanonische Elemente gleich? (*nichtkanonisch*`Equality`) $H \vdash s = t \in T$ *
  - Wie kann man Elemente des Typs nutzen? (*typ*`Elimination`) $H, x{:}S, H' \vdash T \ [\text{ext } t]$
- **Berechnungsregeln:**
  - Reduktion von Redizes in Gleichheit (*nichtkanonisch*`Reduce`) $H \vdash \mathit{redex} = t \in T$ *
- **Spezielle Regeln**

*: Evidenz für Verifikationsziele $\vdash s = t \in T$ ist grundsätzlich ohne konstruktive Bedeutung (Ax)

Refinement Regeln für den Funktionenraum

- **Gleichheitsregel für Funktionenräume**
  - Funktionenräume sind gleich, wenn Argument- und Bildttypen gleich

$$\frac{H \vdash x_1{:}S_1 \to T_1 = x_2{:}S_2 \to T_2 \in \mathbb{U}_j \quad \lfloor \text{Ax} \rfloor \qquad H \vdash S_1 = S_2 \in \mathbb{U}_j \quad \lfloor \text{Ax} \rfloor \qquad H, x{:}S_1 \vdash T_1[x/x_1] = T_2[x/x_2] \in \mathbb{U}_j \quad \lfloor \text{Ax} \rfloor}{\texttt{functionEquality}\ x}$$

- **Formationsregel für Funktionenräume** (wird nicht mehr verwendet)
  - Funktionenraum wird als Evidenz für $\mathbb{U}_j$ konstruiert

$$\frac{H \vdash \mathbb{U}_j \quad \lfloor \text{ext}\ x : S \to T \rfloor \qquad H \vdash S \in \mathbb{U}_j \quad \lfloor \text{Ax} \rfloor \qquad H, x{:}S \vdash \mathbb{U}_j \quad \lfloor \text{ext}\ T \rfloor}{\texttt{functionFormation}\ x\ S}$$

- **Gleichheits-/Formationsregeln für kanonische Funktionen**

$$\frac{H \vdash \lambda x_1 . t_1 = \lambda x_2 . t_2 \in x : S \to T \qquad H \vdash S \in \mathbb{U}_j \qquad H, x'{:}S \vdash t_1[x'/x_1] = t_2[x'/x_2] \in T[x'/x]}{\texttt{lambdaEquality}\ j\ x'}$$

$$\frac{H \vdash x{:}S \to T \quad \lfloor \text{ext}\ \lambda x' . t \rfloor \qquad H, x'{:}S \vdash T[x'/x] \quad \lfloor \text{ext}\ t \rfloor \qquad H \vdash S \in \mathbb{U}_j \quad \lfloor \text{Ax} \rfloor}{\texttt{lambdaFormation}\ j\ x'}$$

Refinement Regeln für den Funktionenraum II

- **Gleichheitsregel für Funktionsapplikationen** (nichtkanonisch)
  - Applikationen sind gleich, wenn Funktionen und Argumente gleich

$$\frac{H \vdash f_1 t_1 = f_2 t_2 \in T[t_1/x] \quad \lfloor \mathsf{Ax} \rfloor \qquad H \vdash f_1 = f_2 \in x{:}S \to T \quad \lfloor \mathsf{Ax} \rfloor \qquad H \vdash t_1 = t_2 \in S \quad \lfloor \mathsf{Ax} \rfloor}{\texttt{applyEquality}\ x{:}S \to T}$$

- **Erzeugung nichtkanonischer Elemente**

$$\frac{H, f{:}x{:}S \to T, H' \vdash C \quad \lfloor \mathsf{ext}\ t[f\,s, \mathsf{Ax}/y, z] \rfloor \qquad H, f{:}x{:}S \to T, H' \vdash s \in S \quad \lfloor \mathsf{Ax} \rfloor \qquad H, f{:}x{:}S \to T, y{:}T[s/x], z{:}y{=}f\,s \in T[s/x], H' \vdash C \quad \lfloor \mathsf{ext}\ t \rfloor}{\texttt{functionElimination}\ i\ s\ y\ z}$$

  - Regel arbeitet auf Hypothesenliste anstatt der Konklusion
  - Regel zeigt, wie Element f des Funktionenraums verwendet wird
  - Evidenzterm enthält Applikation von f auf angegebenes $s \in S$

- **Reduktionsregel**
  - Redex auf linker Seite einer Gleichheit wird durch Kontraktum ersetzt

$$\frac{H \vdash (\lambda x.t)\,s = t_2 \in T \quad \lfloor \mathsf{Ax} \rfloor \qquad H \vdash t[s/x] = t_2 \in T \quad \lfloor \mathsf{Ax} \rfloor}{\texttt{applyReduce}}$$

Sonderregeln

- **Regel für Extensionalität von Funktionen**
  - Funktionen sind gleich, wenn sie auf allen Argumenten gleich sind
    Die Regel `lambdaEquality` sichert dies nur für λ-Abstraktionen

$$\frac{H \vdash f_1 = f_2 \in x{:}S \to T \quad \lfloor \text{ext } t \rfloor \qquad H, x'{:}S \vdash f_1 x' = f_2 x' \in T[x'/x] \quad \lfloor \text{ext } t \rfloor \qquad H \vdash S \in \mathbb{U}_j \quad \lfloor \text{Ax} \rfloor \qquad H \vdash f_1 \in x_1{:}S_1 \to T_1 \quad \lfloor \text{Ax} \rfloor \qquad H \vdash f_2 \in x_2{:}S_2 \to T_2 \quad \lfloor \text{Ax} \rfloor}{}$$

`functionExtensionality` $j\ x_1{:}S_1 \to T_1\ x_2{:}S_2 \to T_2\ x'$

  - f_1, f_2 können auf Obermengen von S und $T[x]$ definiert sein
  - Evidenz für punktweise Gleichheit (meist Ax) wird weitergereicht

- **Gleichheitsregel für deklarierte Variablen**

$$\frac{}{H, x{:}T, H' \vdash x = x \in T \quad \lfloor \text{Ax} \rfloor}$$

`hypothesisEquality` i

  - Reflexivität gilt nur für Elemente eines Typs (vgl. `declaration` aus §6)

- **Formation von Extrakttermen aus deklarierten Variablen**

$$\frac{}{H, x{:}T, H' \vdash T \quad \lfloor \text{ext } x \rfloor}$$

`hypothesis` i

  - Eine Variablendeklaration vom Typ T ist Evidenz für T (vgl. `axiom` aus §3)

Viele Regeln benötigen Steuerungsparameter

Angabe von Daten, die für Teilziel oder Extraktterm benötigt werden, aber im Beweisziel nicht auftauchen

- **Position i einer Hypothese**

$$\frac{}{H, x{:}T, H' \vdash T}\ \lfloor \text{ext } x \rfloor$$ `hypothesis` i

  – Benutzer muß angeben, auf welche Hypothese die Regel zugreifen soll

- **Name x einer neu zu erzeugenden Variablen**

$$\frac{H \vdash x_1{:}S_1 \to T_1 = x_2{:}S_2 \to T_2 \in \mathbb{U}_j \quad H \vdash S_1 = S_2 \in \mathbb{U}_j \quad H, x{:}S_1 \vdash T_1[x/x_1] = T_2[x/x_2] \in \mathbb{U}_j}{}\ \lfloor \text{Ax} \rfloor$$ `functionEquality` x

  – Automatische Namenserzeugung wäre mehr als nur Pattern Matching

- **Level j des Universums eines (Teil-)typs**

$$\frac{H \vdash \lambda x_1 . t_1 = \lambda x_2 . t_2 \in x : S \to T \quad H \vdash S \in \mathbb{U}_j \quad H, x'{:}S \vdash t_1[x'/x_1] = t_2[x'/x_2] \in T[x'/x]}{}\ \lfloor \text{Ax} \rfloor$$ `lambdaEquality` j x'

  – Notwendig für Teilziele, die Typ-Sein eines Ausdrucks prüfen müssen

Steuerungsparameter für Inferenzregeln (II)

- **Term s, der als spezifische Instanz gebraucht wird**

$$\frac{H, f{:}x{:}S{\to}T, H' \vdash C \quad \lfloor \text{ext } t[fs, \mathsf{Ax}/y, z] \rfloor \qquad H, f{:}x{:}S{\to}T, H' \vdash s \in S \quad \lfloor \mathsf{Ax} \rfloor \qquad H, f{:}x{:}S{\to}T, y{:}T[s/x], z{:}y{=}f\ s \in T[s/x], H' \vdash C \quad \lfloor \text{ext } t \rfloor}{\texttt{functionElimination}\ i\ \boldsymbol{s}\ y\ z}$$

  – Term kann Funktionsargument sein oder Variable instantiieren

- **Typ T eines Teilterms des Beweiszieles**

$$\frac{H \vdash f_1 t_1 = f_2 t_2 \in T[t_1/x] \quad \lfloor \mathsf{Ax} \rfloor \qquad H \vdash f_1 = f_2 \in x{:}S{\to}T \quad \lfloor \mathsf{Ax} \rfloor \qquad H \vdash t_1 = t_2 \in S \quad \lfloor \mathsf{Ax} \rfloor}{\texttt{applyEquality}\ \boldsymbol{x{:}S{\to}T}}$$

  – Teilterme, die in Unterzielen isoliert auftreten, benötigen einen Typ

- **Abhängigkeit eines Terms C von einer Variablen x**

$$\frac{H \vdash C[s/x] \quad \lfloor \text{ext } u \rfloor \qquad H \vdash s = t \in T \quad \lfloor \mathsf{Ax} \rfloor \qquad H \vdash C[t/x] \quad \lfloor \text{ext } u \rfloor \qquad H, x{:}T \vdash C \in \mathbb{U}_j \quad \lfloor \mathsf{Ax} \rfloor}{\texttt{substitution}\ j\ s{=}t{\in}T\ \boldsymbol{x}\ \boldsymbol{C}}$$

  – Ermöglicht Verallgemeinerung einer instantiierten Form des Terms C

Anwendung der Regeln für den Funktionenraum

- **Konstruktion eines Terms in $x{:}S{\to}(Tx{\to}Tx)$**
  - Deklarationen $S{:}\mathbb{U}, T{:}S{\to}\mathbb{U}$ sind vorgegeben

$S{:}\mathbb{U}, T{:}S{\to}\mathbb{U} \vdash x{:}S{\to}(Tx{\to}Tx)$	⌊ext $\lambda x.\,(\lambda p.p)$⌋	`lambdaFormation` 1 x
1. $S{:}\mathbb{U}, T{:}S{\to}\mathbb{U}, x{:}S \vdash Tx{\to}Tx$	⌊ext $\lambda p.p$⌋	`lambdaFormation` 1
1.1. $S{:}\mathbb{U}, T{:}S{\to}\mathbb{U}, x{:}S, p{:}Tx \vdash Tx$	⌊ext p⌋	`hypothesis` 4
1.2. $S{:}\mathbb{U}, T{:}S{\to}\mathbb{U}, x{:}S \vdash Tx \in \mathbb{U}$	⌊Ax⌋	`applyEquality` $S{\to}\mathbb{U}$
1.2.1. $S{:}\mathbb{U}, T{:}S{\to}\mathbb{U}, x{:}S \vdash T \in S{\to}\mathbb{U}$	⌊Ax⌋	`hypothesisEquality` 2
1.2.2. $S{:}\mathbb{U}, T{:}S{\to}\mathbb{U}, x{:}S \vdash x \in S$	⌊Ax⌋	`hypothesisEquality` 3
2. $S{:}\mathbb{U}, T{:}S{\to}\mathbb{U}, x{:}S \vdash S \in \mathbb{U}$	⌊Ax⌋	`hypothesisEquality` 1

  - Extraktterm wird nach Vollendung des Beweises generiert
  - Gleichheitsziele tragen nicht zum Extraktterm bei

Formaler Beweis einer Typzugehörigkeit

$T{:}\mathbb{U} \vdash (\lambda f.\lambda x.\, f\,x)(\lambda z.z) \in T \to T$	`applyEquality` $(T \to T) \to (T \to T)$
1. $T{:}\mathbb{U} \vdash \lambda f.\lambda x.\, f\,x \in (T \to T) \to (T \to T)$	`lambdaEquality` 1 f
1.1. $T{:}\mathbb{U}, f{:}T \to T \vdash \lambda x.\, f\,x \in T \to T$	`lambdaEquality` 1 x
1.1.1. $T{:}\mathbb{U}, f{:}T \to T, x{:}T \vdash f\,x \in T$	`applyEquality` $T \to T$
1.1.1.1. $T{:}\mathbb{U}, f{:}T \to T, x{:}T \vdash f \in T \to T$	`hypothesisEquality` 2
1.1.1.2. $T{:}\mathbb{U}, f{:}T \to T, x{:}T \vdash x \in T$	`hypothesisEquality` 3
1.1.2. $T{:}\mathbb{U}, f{:}T \to T \vdash T \in \mathbb{U}$	`hypothesisEquality` 1
1.2. $T{:}\mathbb{U} \vdash T \to T \in \mathbb{U}$	`functionEquality` x
1.2.1. $T{:}\mathbb{U} \vdash T \in \mathbb{U}$	`hypothesisEquality` 1
1.2.2. $T{:}\mathbb{U}, x{:}T \vdash T \in \mathbb{U}$	`hypothesisEquality` 1
2. $T{:}\mathbb{U} \vdash \lambda z.z \in T \to T$	`lambdaEquality` 1 z
2.1. $T{:}\mathbb{U}, z{:}T \vdash z \in T$	`hypothesisEquality` 2
2.2. $T{:}\mathbb{U} \vdash T \in \mathbb{U}$	`hypothesisEquality` 1

Formale Analyse von $\lambda x.x\,x$

- **Es gilt $\lambda x.x\,x \notin S{\to}T$ für alle Typen S, T**
  - Semantisches Argument in Einheit 8 hat gezeigt, daß das Urteil $\lambda\mathtt{x}.\mathtt{x}\,\mathtt{x} \in S{\to}T$ nicht gefällt werden kann
- **Beweisansätze bleiben unvollständig**

$S{:}\mathbb{U}, T{:}\mathbb{U} \vdash \lambda x.\, x\ x \in S{\to}T$	`lambdaEquality` 1 x
1. $S{:}\mathbb{U}, T{:}\mathbb{U}, x{:}S \vdash x\ x \in T$	`applyEquality` $S{\to}T$
1.1. $S{:}\mathbb{U}, T{:}\mathbb{U}, x{:}S \vdash x \in S{\to}T$	`????`
1.2. $S{:}\mathbb{U}, T{:}\mathbb{U}, x{:}S \vdash x \in S$	`hypothesisEquality` 3

  - Beweis kann nicht abgeschlossen werden
  - Auch alternative Beweisversuche führen nicht zum Erfolg
- **$\lambda x.x\,x \notin S{\to}T$ ist formal nicht beweisbar**
  - Man könnte nur zeigen, daß aus der Annahme $\lambda x.\, x\ x \in S{\to}T$ jede beliebige Aussage folgen würde

Vollständiger Regelsatz für Funktionenraum

$H \vdash x_1{:}S_1 \to T_1 = x_2{:}S_2 \to T_2 \in \mathbb{U}_j$ [Ax]
`functionEquality` x
$H \vdash S_1 = S_2 \in \mathbb{U}_j$ [Ax]
$H, x{:}S_1 \vdash T_1[x/x_1] = T_2[x/x_2] \in \mathbb{U}_j$ [Ax]

$H \vdash \mathbb{U}_j$ [ext $x : S \to T$]
`functionFormation` x S
$H \vdash S \in \mathbb{U}_j$ [Ax]
$H, x{:}S \vdash \mathbb{U}_j$ [ext T]

$H \vdash \lambda x_1.t_1 = \lambda x_2.t_2 \in x{:}S \to T$ [Ax]
`lambdaEquality` j x'
$H, x'{:}S \vdash t_1[x'/x_1] = t_2[x'/x_2] \in T[x'/x]$ [Ax]
$H \vdash S \in \mathbb{U}_j$ [Ax]

$H \vdash x{:}S \to T$ [ext $\lambda x'.t$]
`lambdaFormation` j x'
$H, x'{:}S \vdash T[x'/x]$ [ext t]
$H \vdash S \in \mathbb{U}_j$ [Ax]

$H \vdash f_1 t_1 = f_2 t_2 \in T[t_1/x]$ [Ax]
`applyEquality` $x : S \to T$
$H \vdash f_1 = f_2 \in x{:}S \to T$ [Ax]
$H \vdash t_1 = t_2 \in S$ [Ax]

$H, f : x{:}S \to T, H' \vdash C$ [ext $t[f\,s, \mathsf{Ax}/y, z]$]
`functionElimination` i s y z
$H, f : x{:}S \to T, H' \vdash s \in S$ [Ax]
$H, f : x{:}S \to T, y{:}T[s/x], z{:}\, y = f\, s \in T[s/x], H' \vdash C$ [ext t]

$H \vdash (\lambda x.t)\, s = t_2 \in T$ [Ax]
`applyReduce`
$H \vdash t[s/x] = t_2 \in T$ [Ax]

$H \vdash f_1 = f_2 \in x{:}S \to T$ [ext t]
`functionExtensionality` j $x_1 : S_1 \to T_1$ $x_2 : S_2 \to T_2$ x'
$H, x'{:}S \vdash f_1 x' = f_2 x' \in T[x'/x]$ [ext t]
$H \vdash S \in \mathbb{U}_j$ [Ax]
$H \vdash f_1 \in x_1{:}S_1 \to T_1$ [Ax]
$H \vdash f_2 \in x_2{:}S_2 \to T_2$ [Ax]

Warum haben Regeln Wohlgeformtheitsziele ?

Ohne diese Ziele wären seltsame Beweise möglich

- **Verzicht auf $\vdash S \in \mathbb{U}_j$ in `lambdaEquality`**

$$\begin{array}{l} H \vdash \lambda x_1 . t_1 = \lambda x_2 . t_2 \in x : S \to T \\ \quad H, x' : S \vdash t_1[x'/x_1] = t_2[x'/x_2] \in T[x'/x] \end{array} \quad \texttt{lambdaEquality } x'$$

  – Modifizierte Regel würde Beweis für unsinnige Aussage ermöglichen

  $\vdash \lambda x.\, \mathsf{T} \in x : (\lambda y.y) \to \mathbb{B}$ `lambdaEquality` x

  1. $x : (\lambda y.y) \vdash \mathsf{T} \in \mathbb{B}$ *wäre beweisbar mit Regeln für* $\mathbb{B}$

- **Verzicht auf $\vdash C \in \mathbb{U}_j$ in `substitution`**

$$\begin{array}{ll} H \vdash C[s/x] & \lfloor \text{ext } u \rfloor \\ \quad H \vdash s = t \in T & \lfloor \text{Ax} \rfloor \\ \quad H \vdash C[t/x] & \lfloor \text{ext } u \rfloor \end{array} \quad \texttt{substitution } s{=}t \in T \ \boldsymbol{x}\ \boldsymbol{C}$$

  – Modifizierte Regel würde Beweis mit unsinnigen Argumenten ermöglichen

  $\vdash$ `if 1+1=2 then` $\mathbb{B}$ `else` λ`y.y`

  1. $\vdash$ `1+1=2` $\in \mathbb{N}$ `substitution` $1{+}1{=}2 \in \mathbb{N}$ x `if` x`=2 then` $\mathbb{B}$ `else` λ`y.y`
  2. $\vdash$ `if 2=2 then` $\mathbb{B}$ `else` λ`y.y`

  – Unterziele sind ok, aber `if` x`=2 then` $\mathbb{B}$ `else` λ`y.y` ist für $x \neq 2$ kein Typ

Regeln für Produktraum

Regeln ergeben sich direkt aus Semantikdefinition in §8

$H \vdash x_1{:}S_1 \times T_1 = x_2{:}S_2 \times T_2 \in \mathbb{U}_j$ [Ax]
productEquality x'
$H \vdash S_1 = S_2 \in \mathbb{U}_j$ [Ax]
$H, x'{:}S_1 \vdash T_1[x'/x_1] = T_2[x'/x_2] \in \mathbb{U}_j$ [Ax]

$H \vdash \mathbb{U}_j$ [ext $x : S \times T$]
productFormation x S
$H \vdash S \in \mathbb{U}_j$ [Ax]
$H, x{:}S \vdash \mathbb{U}_j$ [ext T]

$H \vdash \langle s_1, t_1 \rangle = \langle s_2, t_2 \rangle \in x{:}S \times T$ [Ax]
pairEquality j x'
$H \vdash s_1 = s_2 \in S$ [Ax]
$H \vdash t_1 = t_2 \in T[s_1/x]$ [Ax]
$H, x'{:}S \vdash T[x'/x] \in \mathbb{U}_j$ [Ax]

$H \vdash x{:}S \times T$ [ext $\langle s, t \rangle$]
pairFormation j s x'
$H \vdash s \in S$ [Ax]
$H \vdash T[s/x]$ [ext t]
$H, x'{:}S \vdash T[x'/x] \in \mathbb{U}_j$ [Ax]

$H \vdash \text{match } e_1 \text{ with } \langle x_1, y_1 \rangle \mapsto t_1 = \text{match } e_2 \text{ with } \langle x_2, y_2 \rangle \mapsto t_2 \in C[e_1/z]$ [Ax]
spreadEquality z C $x : S \times T$ s t y
$H \vdash e_1 = e_2 \in x{:}S \times T$ [Ax]
$H, s{:}S, t{:}T[s/x], y{:}\, e_1 = \langle s, t \rangle \in x{:}S \times T \vdash t_1[s, t/x_1, y_1] = t_2[s, t/x_2, y_2] \in C[\langle s, t \rangle / z]$ [Ax]

$H \vdash \text{match } \langle s, t \rangle \text{ with } \langle x, y \rangle \mapsto u = t_2 \in T$ [Ax]
spreadReduce
$H \vdash u[s, t/x, y] = t_2 \in T$ [Ax]

$H, z{:}\, x{:}S \times T, H' \vdash C$ [ext match z with $\langle s, t \rangle \mapsto u$]
productElimination i s t
$H, z{:}\, x{:}S \times T, s{:}S, t{:}T[s/x]\ H'[\langle s, t \rangle / z] \vdash C[\langle s, t \rangle / z]$ [ext u]

Regeln für unabhängige Räume sind einfacher

$H \vdash x_1{:}S_1 \to T_1 = x_2{:}S_2 \to T_2 \in \mathbb{U}_j$ [Ax]
`functionEquality` x
$H \vdash S_1 = S_2 \in \mathbb{U}_j$ [Ax]
$H, x{:}S_1 \vdash T_1[x/x_1] = T_2[x/x_2] \in \mathbb{U}_j$ [Ax]

$H \vdash S_1 \to T_1 = S_2 \to T_2 \in \mathbb{U}_j$ [Ax]
`independent_functionEquality`
$H \vdash S_1 = S_2 \in \mathbb{U}_j$ [Ax]
$H \vdash T_1 = T_2 \in \mathbb{U}_j$ [Ax]

$H, f{:}x{:}S \to T, H' \vdash C$ [ext $t[f\,s, \mathsf{Ax}/y, z]$]
`functionElimination` i s y z
$H, f{:}x{:}S \to T, H' \vdash s \in S$ [Ax]
$H, f{:}x{:}S \to T, y{:}T[s/x], z{:}y{=}f\,s \in T[s/x], H' \vdash C$ [ext t]

$H, f{:}S \to T, H' \vdash C$ [ext $t[f\,s, /y]$]
`independent_functionElimination` i y
$H, f{:}S \to T, H' \vdash S$ [ext s]
$H, f{:}S \to T, y{:}T, H' \vdash C$ [ext t]

$H \vdash x_1{:}S_1 \times T_1 = x_2{:}S_2 \times T_2 \in \mathbb{U}_j$ [Ax]
`productEquality` x'
$H \vdash S_1 = S_2 \in \mathbb{U}_j$ [Ax]
$H, x'{:}S_1 \vdash T_1[x'/x_1] = T_2[x'/x_2] \in \mathbb{U}_j$ [Ax]

$H \vdash S_1 \times T_1 = S_2 \times T_2 \in \mathbb{U}_j$ [Ax]
`independent_productEquality`
$H \vdash S_1 = S_2 \in \mathbb{U}_j$ [Ax]
$H \vdash T_1 = T_2 \in \mathbb{U}_j$ [Ax]

$H \vdash x{:}S \times T$ [ext $\langle s, t \rangle$]
`pairFormation` j s x'
$H \vdash s \in S$ [Ax]
$H \vdash T[s/x]$ [ext t]
$H, x'{:}S \vdash T[x'/x] \in \mathbb{U}_j$ [Ax]

$H \vdash S \times T$ [ext $\langle s, t \rangle$]
`independent_pairFormation`
$H \vdash S$ [ext s]
$H \vdash T$ [ext t]

$H \vdash \langle s_1, t_1 \rangle = \langle s_2, t_2 \rangle \in x{:}S \times T$ [Ax]
`pairEquality` j x'
$H \vdash s_1 = s_2 \in S$ [Ax]
$H \vdash t_1 = t_2 \in T[s_1/x]$ [Ax]
$H, x'{:}S \vdash T[x'/x] \in \mathbb{U}_j$ [Ax]

$H \vdash \langle s_1, t_1 \rangle = \langle s_1, t_1 \rangle \in S \times T$ [Ax]
`independent_pairEquality`
$H \vdash s_1 = s_2 \in S$ [Ax]
$H \vdash t_1 = t_2 \in T$ [Ax]

Formaler Beweis einer Typzugehörigkeit II

$S{:}\mathbb{U}, T{:}\mathbb{U} \vdash \lambda p.\,\mathsf{match}\ p\ \mathsf{with}\ \langle x, y\rangle \mapsto x \in (S \times T) \to S$

`lambdaEquality 1 p`

1. $S{:}\mathbb{U}, T{:}\mathbb{U}, p{:}S \times T \vdash \mathsf{match}\ p\ \mathsf{with}\ \langle x, y\rangle \mapsto x \in S$

`productElimination 3 s t`

1.1. $S{:}\mathbb{U}, T{:}\mathbb{U}, p{:}S \times T, s{:}S, t{:}T \vdash \mathsf{match}\ \langle s, t\rangle\ \mathsf{with}\ \langle x, y\rangle \mapsto x \in S$

`spreadReduce`

1.1.1. $S{:}\mathbb{U}, T{:}\mathbb{U}, p{:}S \times T, s{:}S, t{:}T \vdash s \in S$

`hypothesisEquality 3`

Alternativer Beweis mit `spreadEquality`

$S{:}\mathbb{U}, T{:}\mathbb{U} \vdash \lambda p.\,\mathsf{match}\ p\ \mathsf{with}\ \langle x, y\rangle \mapsto x \in (S \times T) \to S$

`lambdaEquality 1 p`

1. $S{:}\mathbb{U}, T{:}\mathbb{U}, p{:}S \times T \vdash \mathsf{match}\ p\ \mathsf{with}\ \langle x, y\rangle \mapsto x \in S$

`spreadEquality z S S×T s t z`

1.1. $S{:}\mathbb{U}, T{:}\mathbb{U}, p{:}S \times T \vdash p \in S \times T$ `hypothesisEquality 3`

1.2. $S{:}\mathbb{U}, T{:}\mathbb{U}, p{:}S \times T, s{:}S, t{:}T, p{=}\langle s, t\rangle \in S \times T \vdash s \in S$

`hypothesisEquality 4`

Vollständiger Regelsatz für Summentyp

$H \vdash S_1{+}T_1 = S_2{+}T_2 \in \mathbb{U}_j$ [Ax]
unionEquality
$H \vdash S_1 = S_2 \in \mathbb{U}_j$ [Ax]
$H \vdash T_1 = T_2 \in \mathbb{U}_j$ [Ax]

$H \vdash \mathbb{U}_j$ [ext $S{+}T$]
unionFormation
$H \vdash \mathbb{U}_j$ [ext S]
$H \vdash \mathbb{U}_j$ [ext T]

$H \vdash \mathsf{inl}(s_1) = \mathsf{inl}(s_2) \in S{+}T$ [Ax]
inlEquality j
$H \vdash s_1 = s_2 \in S$ [Ax]
$H \vdash T \in \mathbb{U}_j$ [Ax]

$H \vdash S{+}T$ [ext $\mathsf{inl}(s)$]
inlFormation j
$H \vdash S$ [ext s]
$H \vdash T \in \mathbb{U}_j$ [Ax]

$H \vdash \mathsf{inr}(t_1) = \mathsf{inr}(t_2) \in S{+}T$ [Ax]
inrEquality j
$H \vdash t_1 = t_2 \in T$ [Ax]
$H \vdash S \in \mathbb{U}_j$ [Ax]

$H \vdash S{+}T$ [ext $\mathsf{inr}(t)$]
inrFormation j
$H \vdash T$ [ext t]
$H \vdash S \in \mathbb{U}_j$ [Ax]

$H \vdash \mathsf{case}\ e_1\ \mathsf{of}\ \mathsf{inl}(x_1) \mapsto u_1 \mid \mathsf{inr}(y_1) \mapsto v_1 = \mathsf{case}\ e_2\ \mathsf{of}\ \mathsf{inl}(x_2) \mapsto u_2 \mid \mathsf{inr}(y_2) \mapsto v_2 \in C[e_1/z]$ [Ax]
decideEquality $z\ C\ S{+}T\ s\ t\ y$
$H \vdash e_1 = e_2 \in S{+}T$ [Ax]
$H, s{:}S, y{:}\, e_1{=}\mathsf{inl}(s) \in S{+}T \vdash u_1[s/x_1] = u_2[s/x_2] \in C[\mathsf{inl}(s)/z]$ [Ax]
$H, t{:}T, y{:}\, e_1{=}\mathsf{inr}(t) \in S{+}T \vdash v_1[t/y_1] = v_2[t/y_2] \in C[\mathsf{inr}(t)/z]$ [Ax]

$H, z{:}S{+}T, H' \vdash C$ [ext $\mathsf{case}\ z\ \mathsf{of}\ \mathsf{inl}(x) \mapsto u \mid \mathsf{inr}(y) \mapsto v$]
unionElimination $i\ x\ y$
$H, z{:}S{+}T, x{:}S, H'[\mathsf{inl}(x)/z] \vdash C[\mathsf{inl}(x)/z]$ [ext u]
$H, z{:}S{+}T, y{:}T, H'[\mathsf{inr}(y)/z] \vdash C[\mathsf{inr}(y)/z]$ [ext v]

$H \vdash \mathsf{case}\ \mathsf{inl}(s)\ \mathsf{of}\ \mathsf{inl}(x) \mapsto u \mid \mathsf{inr}(y) \mapsto v = t_2 \in T$ [Ax]
decideReduceLeft
$H \vdash u[s/x] = t_2 \in T$ [Ax]

$H \vdash \mathsf{case}\ \mathsf{inr}(t)\ \mathsf{of}\ \mathsf{inl}(x) \mapsto u \mid \mathsf{inr}(y) \mapsto v = t_2 \in T$ [Ax]
decideReduceRight
$H \vdash v[t/y] = t_2 \in T$ [Ax]

Formaler Beweis einer Typzugehörigkeit III

$S{:}\mathbb{U}, T{:}\mathbb{U} \vdash \lambda z.\ \mathsf{case}\ z\ \mathsf{of}\ \mathsf{inl}(x) \mapsto x \mid \mathsf{inr}(y) \mapsto (y, y)$
$\quad \in\ z{:}S{+}T \rightarrow \mathsf{case}\ z\ \mathsf{of}\ \mathsf{inl}(x) \mapsto S \mid \mathsf{inr}(y) \mapsto T \times T$

`lambdaEquality 1 z THEN unionElimination 3 s t`

1. $S{:}\mathbb{U}, T{:}\mathbb{U}, z{:}S{+}T, s{:}S \vdash \mathsf{case}\ \mathsf{inl}(s)\ \mathsf{of}\ \mathsf{inl}(x) \mapsto x \mid \mathsf{inr}(y) \mapsto (y, y)$
$\quad \in\ \mathsf{case}\ \mathsf{inl}(s)\ \mathsf{of}\ \mathsf{inl}(x) \mapsto S \mid \mathsf{inr}(y) \mapsto T \times T$

`decideReduceLeft`

1.1. $S{:}\mathbb{U}, T{:}\mathbb{U}, z{:}S{+}T, s{:}S \vdash \mathsf{s}$
$\quad \in\ \mathsf{case}\ \mathsf{inl}(s)\ \mathsf{of}\ \mathsf{inl}(x) \mapsto S \mid \mathsf{inr}(y) \mapsto T \times T$

`subst*` $\mathsf{case}\ \mathsf{inl}(s)\ \mathsf{of}\ \mathsf{inl}(x) \mapsto S \mid \mathsf{inr}(y) \mapsto T \times T = S \in \mathbb{U}$
`THENL [ decideReduceLeft THEN hypothesisEquality 1`
`; hypothesisEquality 4]`

2. $S{:}\mathbb{U}, T{:}\mathbb{U}, z{:}S{+}T, y : T \vdash \mathsf{case}\ \mathsf{inr}(t)\ \mathsf{of}\ \mathsf{inl}(x) \mapsto x \mid \mathsf{inr}(y) \mapsto (y, y)$
$\quad \in\ \mathsf{case}\ \mathsf{inr}(t)\ \mathsf{of}\ \mathsf{inl}(x) \mapsto S \mid \mathsf{inr}(y) \mapsto T \times T$

`decideReduceRight`
`THEN subst*` $\mathsf{case}\ \mathsf{inr}(t)\ \mathsf{of}\ \mathsf{inl}(x) \mapsto S \mid \mathsf{inr}(y) \mapsto T \times T = T \times T \in \mathbb{U}$
`THENL [ decideReduceRight THEN ....; hypothesisEquality 4]`

*`subst` ist eine Taktik auf Basis der Regel `substitution`, die Parameter automatisch bestimmt

Regelsatz für {}, Universen, Gleichheit

$\Gamma \vdash \{\} = \{\} \in \mathbb{U}_j$ [Ax]
voidEquality

$\Gamma \vdash \mathsf{any}(s) = \mathsf{any}(t) \in T$ [Ax]
anyEquality
$\Gamma \vdash s = t \in \{\}$ [Ax]

$H \vdash \mathbb{U}_j = \mathbb{U}_j \in \mathbb{U}_k$ [Ax]
universeEquality $(j < k)$

$H \vdash T \in \mathbb{U}_k$ [Ax]
cumulativity j $(j < k)$
$H \vdash T \in \mathbb{U}_j$ [Ax]

$H \vdash s_1 = t_1 \in T_1 = s_2 = t_2 \in T_2 \in \mathbb{U}_j$ [Ax]
equalityEquality
$H \vdash T_1 = T_2 \in \mathbb{U}_j$ [Ax]
$H \vdash s_1 = s_2 \in T_1$ [Ax]
$H \vdash t_1 = t_2 \in T_1$ [Ax]

$H \vdash \mathsf{Ax} = \mathsf{Ax} \in s = t \in T$ [Ax]
axiomEquality
$H \vdash s = t \in T$ [Ax]

$H, z\colon s = t \in T, H' \vdash C$ [ext u]
equalityElimination i
$H, z\colon s = t \in T, H'[\mathsf{Ax}/z] \vdash C[\mathsf{Ax}/z]$ [ext u]

$H, x\!:\!T, H' \vdash x = x \in T$ [Ax]
hypothesisEquality i

$\Gamma \vdash \mathbb{U}_j$ [ext {}]
voidFormation

$\Gamma, z\colon \{\}, H' \vdash C$ [ext $\mathsf{any}(z)$]
voidElimination i

$H \vdash \mathbb{U}_k$ [ext $\mathbb{U}_j$]
universeFormation j $(j < k)$

$H \vdash \mathbb{U}_j$ [ext $s = t \in T$]
equalityFormation T
$H \vdash T \in \mathbb{U}_j$ [Ax]
$H \vdash T$ [ext s]
$H \vdash T$ [ext t]

$H \vdash C[s/x]$ [ext u]
substitution j $s{=}t \in T$ x C
$H \vdash s{=}t \in T$ [Ax]
$H \vdash C[t/x]$ [ext u]
$H, x\!:\!T \vdash C \in \mathbb{U}_j$ [Ax]

$H \vdash s = t \in T$ [Ax]
equality *(Entscheidungsprozedur)*

Effekte von {} sind im Kalkül beweisbar

- **$\{\}\to T$ ist Typ, auch wenn T kein Typ ist**
  - Ausdruck ist für T=`λx.λy.y` oder T=`0=1∈2` typisierbar

$\vdash \{\}\to(\lambda x.\lambda y.y)\in\mathbb{U}$	`functionEquality` x
`1.` $\vdash \{\}\in\mathbb{U}$	`voidEquality`
`2.` $x{:}\{\} \vdash \lambda x.\lambda y.y\in\mathbb{U}$	`voidElimination 1`

- **Der Term $t = \lambda z.\,(\lambda x.\,x\,x)\,(\lambda x.\,x\,x)$ wird typisierbar**

$T{:}\mathbb{U} \vdash \lambda z.\,(\lambda x.\,x\,x)\,(\lambda x.\,x\,x)\in\{\}\to T$	`lambdaEquality 1` z
`1.` $T{:}\mathbb{U}, z{:}\{\} \vdash (\lambda x.\,x\,x)\,(\lambda x.\,x\,x)\in T$	`voidElimination 2`
`2.` $T{:}\mathbb{U} \vdash \{\}\to T\in\mathbb{U}$	`functionEquality` x
`2.1.` $T{:}\mathbb{U} \vdash \{\}\in\mathbb{U}$	`voidEquality`
`2.2.` $T{:}\mathbb{U}, x{:}\{\} \vdash T\in\mathbb{U}$	`hypothesisEquality 1`

- **Starke Normalisierbarkeit gilt nur für Extraktterme**
  - Regeln erzeugen keine Terme mit Selbstreferenz

Weitere Regeln

$H, x{:}T, H' \vdash T$ ⌊ext x⌋
`hypothesis` i

$H, x{:}T, H' \vdash C$ ⌊ext t⌋
`thin` i
$H, H' \vdash C$ ⌊ext t⌋

$H, H' \vdash C$ ⌊ext $(\lambda x\,.\,t)\ s$⌋
`cut` i T x
$H, H' \vdash T$ ⌊ext s⌋
$H, x{:}T, H' \vdash C$ ⌊ext t⌋

$H \vdash T$ ⌊ext t⌋
`introduction` t
$H \vdash t \in T$ ⌊Ax⌋

$H, z{:}T, H' \vdash C$ ⌊ext t⌋
`hyp_replacement` i S j
$H, z{:}S, H' \vdash C$ ⌊ext t⌋
$H, z{:}T, H' \vdash T = S \in \mathbb{U}_j$ ⌊Ax⌋

$H \vdash C$ ⌊ext t⌋
`lemma` "*theorem-name*"

$H \vdash t \in T$ ⌊Ax⌋
`extract` "*theorem-name*"

$H \vdash C$ ⌊ext $t[\sigma]$⌋
`instantiate` Γ' C' σ
$H' \vdash C'$ ⌊ext t⌋

$H \vdash C$ ⌊ext $t[y/x]$⌋
`rename` y x
$\Gamma[x/y] \vdash C[x/y]$ ⌊ext t⌋

$H \vdash C$ ⌊ext t⌋
`unfold` *def-name*
$H \vdash C_{\downarrow}$ ⌊ext t⌋

$H \vdash C_{\downarrow}$ ⌊ext t⌋
`fold` *def-name*
$H \vdash C$ ⌊ext t⌋

$H, z{:}T$,H' $\vdash C$ ⌊ext t⌋
`unfoldHyp` i *def-name*
$H, z{:}T_{\downarrow}$,H' $\vdash C$ ⌊ext t⌋

$H, z{:}T_{\downarrow}$,H' $\vdash C$ ⌊ext t⌋
`foldHyp` i *def-name*
$H, z{:}T$,H' $\vdash C$ ⌊ext t⌋

Wie soll man sich das alles merken?
Wir haben jetzt bereits 59 Regeln

- **Viele Regeln folgen der Struktur von Termen**
  - Formations- und Eliminationsregeln zerlegen Typen
  - Gleichheitsregeln zerlegen kanonische/nichtkanonische
  - Reduktionsregeln führen Ein-Schritt Berechnungen aus
- **Fasse gleichartige Regeln unter einem Namen zusammen**
  - `D` i : Dekomposition eines Terms in Hypothese i (Konklusion $\hat{=}$ 0)
  - `EqD` i : Dekomposition einer Gleichheit in Hypothese i
  - `ReduceEquands` i : Evaluation eines Redex in Hypothese i

  Damit sind die meisten Regeln abgedeckt (Details im Nuprl Manual S. 154ff)
- **Bestimme Parameter automatisch, wenn möglich**
  - Neue Variablennamen sind leicht zu erzeugen
  - Universenlevel / Typ von Termen ist oft durch Typechecking zu finden
  - Ansonsten müssen Parameter explizit angegeben werden

Implementierung als Taktiken, die Regeln aufrufen $\mapsto$ ALuP II

Einbettung der Logik in den Kalkül

- **Logik kann in Typentheorie eingebettet werden**
  - Prinzip "*Propositionen als Datentypen*" macht Aussagen zu Typen
    - Gleichheit ist das grundlegende Prädikat
    - Logische Konnektive entsprechen Typkonstruktoren (vgl. §3, Folie 24)
  - *Curry-Howard Isomorphie* stellt Bezug zwischen Inferenzregeln her
    - Regeln für $\wedge$, $\vee$, $\Rightarrow$, $\neg$, $\forall$, $\exists$ entsprechen denen für $\rightarrow$, $\times$, $+$, $\{\}$
- **Logik wird konservative Erweiterung der Typentheorie**
  - Konnektive sind durch definitorische Gleichheiten definierbar
  - Inferenzregeln sind als Taktiken "programmierbar"
  - Logik kann als Bibliothekskonzept eingeführt werden

    Eine echte Erweiterung des Kalküls ist nicht erforderlich

Definitorische Gleichungen für logische Konzepte

and	$P \wedge Q$	$\equiv$	$P \times Q$
or	$P \vee Q$	$\equiv$	$P + Q$
implies	$P \Rightarrow Q$	$\equiv$	$P \rightarrow Q$
not	$\neg P$	$\equiv$	$P \rightarrow \{\}$
false	f	$\equiv$	$\{\}$
exists	$\exists x : T \,.\, P[x]$	$\equiv$	$x : T \times P[x]$
all	$\forall x : T \,.\, P[x]$	$\equiv$	$x : T \rightarrow P[x]$
prop	$\mathbb{P}_i$	$\equiv$	$\mathbb{U}_i$

Curry-Howard Isomorphie: $\Rightarrow, \neg, \forall$ vs. $\rightarrow, \{\}$

$H \vdash A \Rightarrow B$ ⌊ext $\lambda a.b$⌋
`impliesR`
$H, a{:}A \vdash B$ ⌊ext b⌋

$H \vdash (\forall x)B$ ⌊ext $\lambda x'.\, b$⌋
`allR`
$H, x'{:}\mathbb{U} \vdash B[x'/x]$ ⌊ext b⌋

$H \vdash \neg A$ ⌊ext $\lambda a.b$⌋
`notR`
$H, a{:}A \vdash \mathsf{f}$ ⌊ext b⌋

$H, f{:}A \Rightarrow B, H' \vdash C$ ⌊ext $c[f\ a/b]$⌋
`impliesL` i
$H, f{:}A \Rightarrow B, H' \vdash A$ ⌊ext a⌋
$H, b{:}B, H' \vdash C$ ⌊ext c⌋

$H, f{:}(\forall x)B, H' \vdash C$ ⌊ext $c[f(t)/b]$⌋
`allL` i t
$H, f{:}(\forall x)B, b{:}B[t/x], H' \vdash C$ ⌊ext c⌋

$H, f{:}\neg A, H' \vdash C$ ⌊ext $\mathsf{any}(f(a))$⌋
`notL`
$H, f{:}\neg A, H' \vdash A$ ⌊ext a⌋

$H \vdash x{:}S \rightarrow T$ ⌊ext $\lambda x'.\, t$⌋
`lambdaFormation` j x'
$H, x'{:}S \vdash T[x'/x]$ ⌊ext t⌋
$H \vdash S \in \mathbb{U}_j$ ⌊Ax⌋

$H, f{:}x{:}S \rightarrow T, H' \vdash C$ ⌊ext $t[f\,s, \mathsf{Ax}/y, z]$⌋
`functionElimination` i s y z
$H, f{:}x{:}S \rightarrow T, H' \vdash s \in S$ ⌊Ax⌋
$H, f{:}x{:}S \rightarrow T, y{:}T[s/x], z{:}y{=}f\ s \in T[s/x], H' \vdash C$ ⌊ext t⌋

$\Gamma, z{:}\{\}, H' \vdash C$ ⌊ext $\mathsf{any}(z)$⌋
`voidElimination` i

Curry-Howard Isomorphie: $\wedge$, $\vee$, $\exists$ vs. $\times$, $+$

$H \vdash A \wedge B$ ⌊ext (a, b)⌋
andR
$H \vdash A$ ⌊ext a⌋
$H \vdash B$ ⌊ext b⌋

$H \vdash (\exists x) B$ ⌊ext (t, b)⌋
exR t
$H \vdash B[t/x]$ ⌊ext b⌋

$H, x{:}A \wedge B, H' \vdash C$ ⌊ext $c[x.1, x.2/a, b]$⌋
andL i
$H, a{:}A, b{:}B, H' \vdash C$ ⌊ext c⌋

$H, z{:}(\exists x) B, H' \vdash C$ ⌊ext $c[z.1, z.2/x', b]$⌋
exL i
$H, x'{:}\mathbb{U}, b{:}B[x'/x], H' \vdash C$ ⌊ext c⌋

$H \vdash A \vee B$ ⌊ext inl(a)⌋
orR1
$H \vdash A$ ⌊ext a⌋

$H \vdash A \vee B$ ⌊ext inr(b)⌋
orR2
$H \vdash B$ ⌊ext b⌋

$H, x{:}A \vee B, H' \vdash C$
⌊ext case x of inl$(a) \to c_1$ | inr$(b) \to c_2$⌋
orL i
$H, a{:}A, H' \vdash C$ ⌊ext c_1⌋
$H, b{:}B, H' \vdash C$ ⌊ext c_2⌋

$H \vdash x{:}S \times T$ ⌊ext $\langle s, t \rangle$⌋
pairFormation j s x'
$H \vdash s \in S$ ⌊Ax⌋
$H \vdash T[s/x]$ ⌊ext t⌋
$H, x'{:}S \vdash T[x'/x] \in \mathbb{U}_j$ ⌊Ax⌋

$H, z{:}\, x{:}S \times T, H' \vdash C$ ⌊ext match z with $\langle s, t \rangle \mapsto u$⌋
productElimination i s t
$H, z{:}\, x{:}S \times T, s{:}S, t{:}T[s/x]\ H'[\langle s, t \rangle / z] \vdash C[\langle s, t \rangle / z]$ ⌊ext u⌋

$H \vdash S + T$ ⌊ext inl(s)⌋
inlFormation j
$H \vdash S$ ⌊ext s⌋
$H \vdash T \in \mathbb{U}_j$ ⌊Ax⌋

$H \vdash S + T$ ⌊ext inr(t)⌋
inrFormation j
$H \vdash T$ ⌊ext t⌋
$H \vdash S \in \mathbb{U}_j$ ⌊Ax⌋

$H, z{:}S + T, H' \vdash C$ ⌊ext case z of inl$(x) \mapsto u$ | inr$(y) \mapsto v$⌋
unionElimination i x y
$H, z{:}S + T, x{:}S, H'[\text{inl}(x)/z] \vdash C[\text{inl}(x)/z]$ ⌊ext u⌋
$H, z{:}S + T, y{:}T, H'[\text{inr}(y)/z] \vdash C[\text{inr}(y)/z]$ ⌊ext v⌋

Logische Regeln als konservative Erweiterung

- **Logische Regeln entsprechen Beweisfragmenten**

$H \vdash A \Rightarrow B$ [ext $\lambda a.b$]
`impliesR`
$H, a{:}A \vdash B$ [ext b]

$H \vdash A \Rightarrow B$ [ext $\lambda a.b$]
`unfold` *implies*
1. $H \vdash A \to B$ [ext $\lambda a.b$]
`lambdaFormation 1` a
1.1. $H, a{:}A \vdash B$ [ext b]
1.2. $H \vdash A \in \mathbb{U}$ [Ax]
`fold` *prop*
1.2.1. $H \vdash A \in \mathbb{P}$ [Ax]
`Auto`

  - Wohlgeformtheit der Aussage A wird implizit vorausgesetzt
  - In der Typentheorie muß $A \in \mathbb{P}$ explizit bewiesen werden
  - Taktik `Auto` kann Wohlgeformtheit für "einfache" Aussagen beweisen

- **Implementierung von `impliesR` als Taktik**
  - `impliesR` ≡ `unfold` *implies*
    `THEN` `lambdaFormation 1` a
    `THENL [Id; fold` *prop* `THEN Auto]`
  - Taktik `Id` läßt aktuelles Ziel unverändert, a ist neuer Name (↦ Alup II)
  - Verwendung von fold *prop* kann entfallen, da `Auto` auch $A \in \mathbb{U}$ löst

Implementierung der Regel impliesL

- **Beweisfragment für `impliesL` ist komplex**

$H, f{:}A \Rightarrow B, H' \vdash C$ ⌊ext $c[f\ a/b]$⌋
  `impliesL` i
  $H, f{:}A \Rightarrow B, H' \vdash A$ ⌊ext a⌋
  $H, b{:}B, H' \vdash C$ ⌊ext c⌋

$H, f{:}A \Rightarrow B, H' \vdash C$ ⌊ext $(\lambda x.c[f\ x/b])\ a$⌋
  `cut` i A x
  1. $H, f{:}A \Rightarrow B, H' \vdash A$ ⌊ext a⌋
  2. $H, f{:}A \to B, x{:}A, H' \vdash C$ ⌊ext $c[f\ s/b]$⌋
     `unfoldHyp` i *implies*
  2.1. $H, f{:}A \to B, x{:}A, H' \vdash C$ ⌊ext $c[f\ x/b]$⌋
       `functionElimination` i x b z
  2.1.1. $H, f{:}A \to B, x{:}A, H' \vdash x \in A$ ⌊Ax⌋
         `hypothesisEquality` $i{+}1$
  2.1.2. $H, f{:}A \to B, x{:}A, b{:}B,$
         $z : b{=}f\ x \in B, H' \vdash C$ ⌊ext c⌋
         `thin` i `THEN thin` i `THEN thin` $i{+}1$
  2.1.2.1. $H, b{:}B, H' \vdash C$ ⌊ext c⌋

  - `functionElimination` benötigt explizit einen Term $a \in A$
  - Regel `cut` ermöglicht, diesen separat konstruieren zu lassen
  - Regel `thin` eliminiert Hypothesen, die nicht gebraucht werden

- **Implementierung von `impliesL` als Taktik**

```
impliesL i ≡      cut i A x
            THENL [Id; unfoldHyp i implies
                       THEN functionElimination i
                       THENL [Id; hypothesisEquality i+1 THEN thin i
                                  THEN thin i THEN thin i+1]]
```

  - Term A muß aus Kontext bestimmt werden, x ist neuer Name (↦ Alup II)

Das Auswahlaxiom ist formal beweisbar

$$(\forall x{:}S.\, \exists y{:}T.\, P(x,y)) \Rightarrow \exists f{:}S\to T.\forall x{:}S.\, P(x,f(x))$$

```
⊢ ∀S,T:𝕌j.∀P: S×T→ℙj. (∀x:S. ∃y:T. P(x,y)) ⇒ ∃f:S→T.∀x:S. P(x,f x)
                                                    Repeat allR
1. S:𝕌j,T:𝕌j,P: S×T→ℙj ⊢ (∀x:S. ∃y:T. P(x,y)) ⇒ ∃f:S→T.∀x:S. P(x,f x)
                                                    impliesR
1.1..., h:(∀x:S. ∃y:T. P(x,y)) ⊢ ∃f:S→T.∀x:S. P(x,f x)
                                                    exR λx. (h x).1
1.1.1..., h:(∀x:S. ∃y:T. P(x,y)) ⊢ ∀x:S. P(x, (λx. (h x).1) x)
                                                    allR THEN reduce
1.1.1.1..., h:(∀x:S. ∃y:T. P(x,y)), x:S ⊢ P(x, (h x).1))
                                                    allL 4 x
1.1.1.1.1..., h:(∀x:S. ∃y:T. P(x,y)), x:S, p:(∃y:T. P(x,y)) ⊢ P(x, p.1))
                                                    exL 5
1.1.1.1.1.1..., h:(∀x:S. ∃y:T. P(x,y)), x:S, y:T, z:P(x,y)) ⊢ P(x, (y,z).1))
                                                    reduce THEN hypothesisEquality 7
```

- **Aussage ist konstruktiv**
  - Die Funktion f kann aus einer Evidenz der $\forall\exists$-Aussage extrahiert werden
  - Grundlage für Synthese von Programmen aus Beweisen

Metalogische Aspekte

- **Wichtige Sätze der Refinement Logik sind leicht zu zeigen**
  - *Ist $H \vdash C$ in RL beweisbar, dann gibt es einen Beweis ohne Schnittregel*

    Nachweis des Schnitteliminationssatzes durch Normalisierung der Evidenz
  - *RL ist konsistent*

    Andernfalls gäbe es einen schnittfreien Beweis für die Sequenz $\vdash \mathsf{f}$
- **Typisierbarkeit**
  - Typisierbare Terme können nichttypisierbare Teilterme enthalten
  - Typzugehörigkeit und Typsein ist i.a. nicht entscheidbar
    - Objekt- und Typausdrücke können nichtterminierende Terme sein
    - Halteproblem wird Teil des Typisierbarkeitsproblems
  - Wohlgeformtheitsziele im Beweiskalkül sind unvermeidbar
- **Normalisierbarkeits- und Konfluenzaussagen**
  - Manche typisierbare Terme sind weder stark noch schwach normalisierbar
  - Wegen Festlegung auf Lazy Evaluation ist Konfluenz irrelevant
  - Extrahierbare Terme sind stark normalisierbar und konfluent

Entwurfsentscheidungen im Rückblick

- **Beweisführung ist analytisch mit Verfeinerungsregeln**
  – Unterstützt die (maschinelle) Suche nach Beweisen
- **Urteile werden durch Konklusionen $T \lfloor \mathsf{ext}\ t \rfloor$ dargestellt**
  – Ermöglicht Konstruktion von Evidenzen für Typen und Aussagen
- **Gleichheit ist ein Datentyp und extensional**
  – Unterstützt Typzugehörigkeits- und Gleichheitsbeweise
- **Typ-Sein wird durch eine Universenhierarchie repräsentiert**
  – Jedes Universum für sich ist eine unvollständige Repräsentation
- **Logische Aussagen werden als Datentypen repräsentiert**
  – Curry-Howard Isomorphie zwischen Konnektiven und Typkonstruktoren