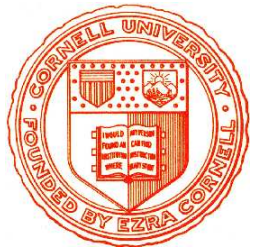
Automatisierte Logik und Programmierung

Einheit 10



Programmierung in der Typentheorie

1. Beweise als Programme
2. Formalisierung von Zahlen und Arithmetik
3. Konstruktion effizienter Algorithmen
4. Formalisierung von Listen
5. Induktion und Rekursion

Beweise als Programme

- **Formale Beweise können verifizieren und konstruieren**
  - Verifikation $\vdash s{=}t \in T$: "s und t erfüllen Spezifikation T und sind gleich"
  - Konstruktion $\vdash T\ \lfloor \mathsf{ext}\ t \rfloor$: "Es gibt einen Term t, der T erfüllt"
- **Beweise für ∀-∃-Theoreme beschreiben Programme**
  - Beweis für $\forall x{:}S.\, \exists y{:}T.\, spec(x, y)$ liefert Extrakt-Term p_{spec} der Form $\lambda x.\, (y_x, pf)$, so daß $spec(x, y_x)$ für alle $x \in S$ gilt
  - Programm, das y_x aus x berechnet kann aus Beweis extrahiert werden
- **Konstruktives Auswahlaxiom ist im Kalkül beweisbar** ↦ §9

  $\vdash\ \forall S{:}\mathbb{U}.\, \forall T{:}\mathbb{U}.\, \forall spec{:}\, S \times T \to \mathbb{P}.$
  $(\forall x{:}S.\, \exists y{:}T.\, spec(x, y)) \Rightarrow \exists f{:}S \to T.\, \forall x{:}S.\, spec(x, f(x))$

Programmentwicklung ist dasselbe wie Beweisführung

Programmentwicklung in der Typentheorie

- **Bisherige Theorie liefert nur einfache Programme**
  - λ-Terme: einfache sequentielle Funktionen
  - Paarbildung: einfachste Form einer Datenstruktur
  - Summentyp: einfache Fallunterscheidungen
  - Alle anderen Konzepte müssen durch Anwender simuliert werden
- **"Praktische" Programmierung braucht mehr**
  - Zahlen und arithmetische Operationen
  - Datencontainer mit flexibler Größe
  - Rekursive Daten- und Programmstrukturen

  ... unterstützt durch geeignete Inferenzmechanismen
- **Unterstütze Verifikation und Synthese**
  - Das Schreiben von Programmen ist nicht alles
  - Verifikation: nachträglicher Beweis von Programmeigenschaften
  - Synthese: Erzeuge Programme mit garantierten Eigenschaften

Formalisierung von Zahlen und Arithmetik

- **Simulation durch Church-Numerals wäre möglich**
  - +: Man kann alle berechenbaren Funktionen definieren (vgl. Einheit 5)
  - −: Aussagen über alle Zahlen sind schwer zu formalisieren
  - −: Beweise für sehr einfache Aussagen werden extrem aufwendig

  Ein Typ der Zahlen muß explizit in der Typentheorie definiert werden

- **Einfachster Ansatz: formale Peano-Arithmetik**
  - Wenige Grundkonstrukte: Typ $\mathbb{N}$, Element 0, Nachfolgerfunktion $\mathtt{s}$,
    Primitive Rekursion $h(t)$ where $h(0) = base \mid h(\mathtt{s}(i)) = g[i, h(i) \,/\, n,x]$
    Kurzdarstellung: $\mathsf{PR}[base\,;\, g[n\,,x]\,](t)$
  - Minimalkonzept mit einfacher Semantik und kleinem Regelsatz
  - Alle wichtigen arithmetischen Operationen sind leicht zu konstruieren

Formulierung der primitiv-rekursiven Arithmetik

Schließen über arithmetische Operationen

- **Ausdrücke**

  Kanonisch: $\mathbb{N}$

  0

  $\mathtt{s}(t)$ t Term

  Nichtkanonisch: $\mathsf{PR}[\mathit{base}\,;\, g[n\,,x]\,](t)$ t, base, g Terme, n, x Variablen

- **Reduktion von Ausdrücken**

  $\mathsf{PR}[\mathit{base}\,;\, g[n\,,x]\,](0) \xrightarrow{\beta} \mathit{base}$

  $\mathsf{PR}[\mathit{base}\,;\, g[n\,,x]\,](\mathtt{s}(t)) \xrightarrow{\beta} g[t, \mathsf{PR}[\mathit{base}\,;\, g[n\,,x]\,](t) \,/\, n, x]$

- **Urteile für Typ- und Elementgleichheit**

  $\mathbb{N} = \mathbb{N}$

  $0 = 0 \in \mathbb{N}$

  $\mathtt{s}(t_1) = \mathtt{s}(t_2) \in \mathbb{N}$ falls $t_1 = t_2 \in \mathbb{N}$

  $\mathbb{N} = \mathbb{N} \in \mathbb{U}_j$ — *für alle j !* —

Konservative Erweiterung der Peano Arithmetik

Definitionen für Standardoperationen auf $\mathbb{N}$

- **Oft Instanz einer primitiv-rekursiven Funktion**

$$
\begin{array}{lcl}
n{+}m & \equiv & h(m) \text{ where } h(0) = n \mid h(\mathtt{s}(i)) = \mathtt{s}(h(i)) \\
\mathtt{p}(n) & \equiv & h(n) \text{ where } h(0) = 0 \mid h(\mathtt{s}(i)) = i \\
n{-}m & \equiv & h(m) \text{ where } h(0) = n \mid h(\mathtt{s}(i)) = \mathtt{p}(h(i)) \\
|n\text{-}m| & \equiv & n{-}m + m{-}n \\
n{*}m & \equiv & h(m) \text{ where } h(0) = 0 \mid h(\mathtt{s}(i)) = n{+}(h(i)) \\
\text{if } n{=}m \text{ then } s \text{ else } t & \equiv & h(|m{-}n|) \text{ where } h(0) = s \mid h(\mathtt{s}(i)) = t \\
\text{if } n{<}m \text{ then } s \text{ else } t & \equiv & h(m{-}n) \text{ where } h(0) = t \mid h(\mathtt{s}(i)) = s \\
n \div m & \equiv & h(n) \text{ where } h(0) = 0 \\
 & & \mid h(\mathtt{s}(i)) = \text{ if } \mathtt{s}(h(i)){*}m{=}\mathtt{s}(i) \text{ then } \mathtt{s}(h(i)) \text{ else } h(i) \\
n \text{ rem } m & \equiv & n - (n \div m{*}m) \\
n{<}m & \equiv & \exists \mathtt{i}{:}\mathbb{N}.\ n{+}\mathtt{s(i)} = m \in \mathbb{N} \qquad \textit{(Prädikat)} \\
 & & \vdots
\end{array}
$$

- **Weitere Operationen mit Y-Kombinator definierbar**

Semantische Untersuchung von Eigenschaften

Zeige $(\lambda x.\langle(\lambda y.\mathbf{s}(y))\,x,\,0\rangle)\,0$
$= \text{match}\ \langle 0,\mathbf{s}(0)\rangle\ \text{with}\ \langle x,y\rangle \mapsto \langle y,x*y\rangle \in \mathbb{N}\times\mathbb{N}$

Analysiere Bedingungen der semantischen Urteile

- Beide Elementterme sind nichtkanonisch und müssen reduziert werden
  - $(\lambda x.\langle(\lambda y.\mathbf{s}(y))\,x,\,0\rangle)\,0 \xrightarrow{l} \langle(\lambda y.\mathbf{s}(y))\,0,\,0\rangle$
  - $\text{match}\ \langle 0,\mathbf{s}(0)\rangle\ \text{with}\ \langle x,y\rangle \mapsto \langle y,x*y\rangle \xrightarrow{l} \langle\mathbf{s}(0),\,0*\mathbf{s}(0)\rangle$
- Es gilt $\langle(\lambda y.\mathbf{s}(y))\,0,\,0\rangle = \langle\mathbf{s}(0),\,0*\mathbf{s}(0)\rangle \in \mathbb{N}\times\mathbb{N}$
  falls $(\lambda y.\mathbf{s}(y))\,0 = \mathbf{s}(0) \in \mathbb{N}$ und $0 = 0*\mathbf{s}(0) \in \mathbb{N}$ und $\mathbb{N}\times\mathbb{N}$ Typ
- Beide Gleichungen enthalten nichtkanonische Elemente, die reduziert werden müssen: $(\lambda y.\mathbf{s}(y))\,0 \xrightarrow{l} \mathbf{s}(0)$ und $0*\mathbf{s}(0) \xrightarrow{l} 0$
  Damit gilt $\langle(\lambda y.\mathbf{s}(y))\,0,\,0\rangle = \langle\mathbf{s}(0),\,0*\mathbf{s}(0)\rangle \in \mathbb{N}\times\mathbb{N}$
  falls $\mathbf{s}(0) = \mathbf{s}(0) \in \mathbb{N}$ und $0 = 0 \in \mathbb{N}$ und $\mathbb{N}\times\mathbb{N}$ *Typ*
- Das erste Urteil darf gefällt werden, falls $0 = 0 \in \mathbb{N}$ ist ✓
  Das zweite Urteil ist bereits ein Basisurteil ✓
  Das dritte Urteil darf gefällt werden, falls $\mathbb{N}$ *Typ* ✓

Regeln für Natürliche Zahlen

$H \vdash \mathbb{N} \in \mathbb{U}_j$ [Ax]
`natEquality`

$H \vdash \mathbb{U}_j$ [ext $\mathbb{N}$]
`natFormation`

$H \vdash 0 = 0 \in \mathbb{N}$ [Ax]
`zeroEquality`

$H \vdash \mathbb{N}$ [ext 0]
`zeroFormation`

$H \vdash \mathtt{s}(t_1) = \mathtt{s}(t_2) \in \mathbb{N}$ [Ax]
`succEquality`
$H \vdash t_1 = t_2 \in \mathbb{N}$

$H \vdash \mathbb{N}$ [ext $\mathtt{s}(t)$]
`succFormation`
$H \vdash \mathbb{N}$ [ext t]

$H \vdash \mathsf{PR}[f_1; g_1[n_1, x_1]](t_1) = \mathsf{PR}[f_2; g_2[n_2, x_2]](t_2) \in T[t_1/z]$ [Ax]
`prEquality` z T n h_n
$H \vdash t_1 = t_2 \in \mathbb{N}$ [Ax]
$H \vdash f_1 = f_2 \in T[0/z]$ [Ax]
$H, n{:}\mathbb{N}, h_n{:}T[x/z] \vdash g_1(\mathtt{s}(n), h_n) = g_2(\mathtt{s}(n), h_n) \in T[\mathtt{s}(n)/z]$ [Ax]

$H, z{:}\mathbb{N}, \Delta \vdash C$ [ext $\mathsf{PR}[f; g[n, x]](z)$]
`natElimination` i n h_n
$H, z{:}\mathbb{N}, \Delta \vdash C[0/z]$ [ext f]
$H, z{:}\mathbb{N}, \Delta, n{:}\mathbb{N}, h_n{:}C[n/z] \vdash C[\mathtt{s}(n)/z]$ [ext $g(n, h_n)$]

$H \vdash \mathsf{PR}[f; g[n, x]](0) = t_2 \in T$ [Ax]
`prReduceBase`
$H \vdash f = t_2 \in T$ [Ax]

$H \vdash \mathsf{PR}[f; g[n, x]](\mathtt{s}(t)) = t_2 \in T$ [Ax]
`prReduceUp`
$H \vdash g(t, \mathsf{PR}[f; g[n, x]](t)) = t_2 \in T$ [Ax]

Nachweis der Kommutativität der Addition

$\vdash \forall n{:}\mathbb{N}.\forall m{:}\mathbb{N}.\ n+m = m+n \in \mathbb{N}$	allR THEN allR
1. $n{:}\mathbb{N}, m{:}\mathbb{N} \vdash n+m = m+n \in \mathbb{N}$	natElimination 2
1.1. $n{:}\mathbb{N}, m{:}\mathbb{N} \vdash n+0 = 0+n \in \mathbb{N}$	prReduceBase
1.1.1. $n{:}\mathbb{N}, m{:}\mathbb{N} \vdash n = 0+n \in \mathbb{N}$	natElimination 1
1.1.1.1. $n{:}\mathbb{N}, m{:}\mathbb{N} \vdash 0 = 0+0 \in \mathbb{N}$	symmetry THEN prReduceBase
1.1.1.1.1. $n{:}\mathbb{N}, m{:}\mathbb{N} \vdash 0 = 0 \in \mathbb{N}$	zeroEquality
1.1.1.2. $n{:}\mathbb{N}, m{:}\mathbb{N}, k{:}\mathbb{N}, f_k{:}k=0+k \in \mathbb{N} \vdash \mathbf{s}(k) = 0+\mathbf{s}(k) \in \mathbb{N}$	symmetry THEN prReduceUp THEN symmetry
1.1.1.2.1. $n{:}\mathbb{N}, m{:}\mathbb{N}, k{:}\mathbb{N}, f_k{:}k=0+k \in \mathbb{N} \vdash \mathbf{s}(k) = \mathbf{s}(0+k) \in \mathbb{N}$	succEquality THEN hypothesis 4
1.2. $n{:}\mathbb{N}, m{:}\mathbb{N}, k{:}\mathbb{N}, f_k{:}n+k=k+n \in \mathbb{N} \vdash n+\mathbf{s}(k) = \mathbf{s}(k)+n \in \mathbb{N}$	
	$\vdots$

- **Beweise in der Peano-Arithmetik sind mühsam**
  - Die meisten Beweise benötigen aufwendige Induktionen
  - Grundeigenschaften erscheinen in Hunderten von Variationen

  Peano-Arithmetik ist für praktisches formales Schließen ungeeignet

Eine praktischere Formalisierung von Zahlen

- **Peano-Arithmetik ist nur theoretisch hinreichend**
  - +: Nur wenige Grundkonstrukte, überschaubarer Regelsatz
  - +: Alle berechenbaren Funktionen sind konservativ definierbar
  - −: Zahlen in Nachfolgerdarstellung 0, $\mathsf{s}(0)$, $\mathsf{s}(\mathsf{s}(0))$, ... sind unnatürlich
  - −: Beweise einfacher Eigenschaften sind immer noch zu mühsam
    Es fehlt auch ein Urteil und die entsprechende Regel für $0 \neq \mathsf{s}(x)$
  - −: Formale Arithmetik ist auf diesem Niveau nicht praktikabel

  Formalisierung von Zahlen sollte auf Dezimalarithmetik aufgebaut werden
- **Der Datentyp der Zahlen in Nuprl**

  Verzicht auf rein schematische Theorie zugunsten der Praktikabilität
  - Grundlage sind ganze Zahlen $\mathbb{Z}$ einschließlich der negativen Zahlen
  - Kanonische Elemente sind Zahlen in Dezimaldarstellung (kein Limit)
  - Primitive Rekursion (Induktive Definition) auch auf negativen Zahlen
  - Fast alle Operationen von Folie 5 sind vordefiniert
    `+`, `-`, `*`, `÷`, `rem`, arithmetische Vergleiche, Prädikat "<"
  - Reduktion basiert auf bekannter Arithmetik anstatt Term-Rewriting
  - Arithmetische Entscheidungsprozedur als zusätzliche Inferenzregel

Der Datentyp der ganzen Zahlen

- **Ausdrücke**

Kanonisch:	$\mathbb{Z}$, n, $-n$	n natürliche Zahl (Dezimaldarstellung)
	$s<t$, Ax	s, t Terme
Nichtkanonisch:	$-t$	t Term
	$s+t$, $s-t$, $s*t$, $s \div t$, s rem t	s, t Terme
	if $u=v$ then s else t	u, v, s, t Terme
	if $u<v$ then s else t	u, v, s, t Terme
	$\mathsf{ind}[base\,;\ f[i\,,x]\,;\ g[j\,,y]\,](u)$	$u, f, g, base$ Terme, i, j, x, y Variablen

Auch: $h(u)$ where $h(0) = base \mid h(z{<}0) = f[z, h(z{+}1)/i, x] \mid h(z{>}0) = g[z, h(z{-}1)/j, y]$

- **Reduktion von Ausdrücken** (nächste Folie)

- **Urteile für Typ- und Elementgleichheit**

$\mathbb{Z} = \mathbb{Z}$		
$s = t \in \mathbb{Z}$	g.d.w.	es eine ganze Zahl i gibt mit $s \xrightarrow{*} i$ und $t \xrightarrow{*} i$
$s_1{<}t_1 = s_2{<}t_2$	falls	$s_1{=}s_2 \in \mathbb{Z}$ und $t_1{=}t_2 \in \mathbb{Z}$
$\mathsf{Ax} = \mathsf{Ax} \in s{<}t$	falls	es Zahlen i und j gibt mit $s \xrightarrow{*} i$, $t \xrightarrow{*} j$ und i ist kleiner als j
$\mathbb{Z} = \mathbb{Z} \in \mathbb{U}_j$		— *für alle j !* —
$s_1{<}t_1 = s_2{<}t_2 \in \mathbb{U}_j$	falls	$s_1{=}s_2 \in \mathbb{Z}$ und $t_1{=}t_2 \in \mathbb{Z}$ — *für alle j !* —

Reduktion arithmetischer Operationen

$\mathsf{ind}[base\,;\ f[i\,,x]\,;\ g[j\,,y]\,](\mathsf{0}) \xrightarrow{\beta} base$

$\mathsf{ind}[base\,;\ f[i\,,x]\,;\ g[j\,,y]\,](n)^{*} \xrightarrow{\beta} f[n, \mathsf{ind}[base\,;\ f[i\,,x]\,;\ g[j\,,y]\,](n{-}1)\,/\,i, x]$

$\mathsf{ind}[base\,;\ f[i\,,x]\,;\ g[j\,,y]\,](\mathsf{-}n)^{*} \xrightarrow{\beta} g[\mathsf{-}n, \mathsf{ind}[base\,;\ f[i\,,x]\,;\ g[j\,,y]\,](\mathsf{-}n\mathsf{+}1)\,/\,j, y]$

$\mathsf{-}i \xrightarrow{\beta}$ *Negation von i (als Zahl)*

$i\mathsf{+}j \xrightarrow{\beta}$ *Summe von i und j*

$i\mathsf{-}j \xrightarrow{\beta}$ *Differenz von i und j*

$i\mathsf{*}j \xrightarrow{\beta}$ *Produkt von i und j*

$i \div j \xrightarrow{\beta}$ 0, *falls j=0; sonst Integer-Division von i und j*

i rem $j \xrightarrow{\beta}$ 0, *falls j=0; sonst Divisionsrest von i und j*

if i=j then s else $t \xrightarrow{\beta}$ s, *falls $i = j$; ansonsten t*

if i<j then s else $t \xrightarrow{\beta}$ s, *falls $i < j$; ansonsten t*

*: $n{>}0$ natürliche Zahl

Regeln für ganze Zahlen

$H \vdash \mathbb{Z} \in \mathbb{U}_j$ [Ax]
intEquality

$H \vdash \mathbb{U}_j$ [ext $\mathbb{Z}$]
intFormation

$\Gamma \vdash s_1<t_1 = s_2<t_2 \in \mathbb{U}_j$ [Ax]
less_thanEquality
$\Gamma \vdash s_1 = s_2 \in \mathbb{Z}$ [Ax]
$\Gamma \vdash t_1 = t_2 \in \mathbb{Z}$ [Ax]

$\Gamma \vdash \mathbb{U}_j$ [ext $s<t$]
less_thanFormation
$\Gamma \vdash \mathbb{Z}$ [ext s]
$\Gamma \vdash \mathbb{Z}$ [ext t]

$H \vdash n = n \in \mathbb{Z}$ [Ax]
natural_numberEquality

$H \vdash \mathbb{Z}$ [ext n]
natural_numberFormation n

$\Gamma \vdash \mathsf{Ax} \in s<t$ [Ax]
less_thanMember
$\Gamma \vdash s < t$ [Ax]

Eine Konstruktion von Elementen aus $s<t$ ist nur mit der arith *Prozedur (Folie 15) möglich*

$H \vdash -s_1 = -s_2 \in \mathbb{Z}$ [Ax]
minusEquality
$H \vdash s_1 = s_2 \in \mathbb{Z}$ [Ax]

$H \vdash s_1+t_1 = s_2+t_2 \in \mathbb{Z}$ [Ax]
addEquality
$H \vdash s_1 = s_2 \in \mathbb{Z}$ [Ax]
$H \vdash t_1 = t_2 \in \mathbb{Z}$ [Ax]

$H \vdash \mathbb{Z}$ [ext $s+t$]
addFormation
$H \vdash \mathbb{Z}$ [ext s]
$H \vdash \mathbb{Z}$ [ext t]

$H \vdash s_1-t_1 = s_2-t_2 \in \mathbb{Z}$ [Ax]
subtractEquality
$H \vdash s_1 = s_2 \in \mathbb{Z}$ [Ax]
$H \vdash t_1 = t_2 \in \mathbb{Z}$ [Ax]

$H \vdash \mathbb{Z}$ [ext $s-t$]
subtractFormation
$H \vdash \mathbb{Z}$ [ext s]
$H \vdash \mathbb{Z}$ [ext t]

Regeln für ganze Zahlen II

$H \vdash s_1 * t_1 = s_2 * t_2 \in \mathbb{Z}$ [Ax]
multiplyEquality
$H \vdash s_1 = s_2 \in \mathbb{Z}$ [Ax]
$H \vdash t_1 = t_2 \in \mathbb{Z}$ [Ax]

$H \vdash \mathbb{Z}$ [ext $s*t$]
multiplyFormation
$H \vdash \mathbb{Z}$ [ext s]
$H \vdash \mathbb{Z}$ [ext t]

$H \vdash s_1 \div t_1 = s_2 \div t_2 \in \mathbb{Z}$ [Ax]
divideEquality
$H \vdash s_1 = s_2 \in \mathbb{Z}$ [Ax]
$H \vdash t_1 = t_2 \in \mathbb{Z}$ [Ax]
$H \vdash t_1 \neq 0$ [Ax]

$H \vdash \mathbb{Z}$ [ext $s \div t$]
divideFormation
$H \vdash \mathbb{Z}$ [ext s]
$H \vdash \mathbb{Z}$ [ext t]

$H \vdash s_1 \text{ rem } t_1 = s_2 \text{ rem } t_2 \in \mathbb{Z}$ [Ax]
remainderEquality
$H \vdash s_1 = s_2 \in \mathbb{Z}$ [Ax]
$H \vdash t_1 = t_2 \in \mathbb{Z}$ [Ax]
$H \vdash t_1 \neq 0$ [Ax]

$H \vdash \mathbb{Z}$ [ext $s \text{ rem } t$]
remainderFormation
$H \vdash \mathbb{Z}$ [ext s]
$H \vdash \mathbb{Z}$ [ext t]

$H \vdash 0 \leq s \text{ rem } t \wedge s \text{ rem } t < t$ [Ax]
remainderBounds1
$H \vdash 0 \leq s$ [Ax]
$H \vdash 0 < t$ [Ax]

$H \vdash 0 \leq s \text{ rem } t \wedge s \text{ rem } t < -t$ [Ax]
remainderBounds2
$H \vdash 0 \leq s$ [Ax]
$H \vdash t < 0$ [Ax]

$H \vdash s \text{ rem } t \leq 0 \wedge s \text{ rem } t > t$ [Ax]
remainderBounds3
$H \vdash s \leq 0$ [Ax]
$H \vdash t < 0$ [Ax]

$H \vdash s \text{ rem } t \leq 0 \wedge s \text{ rem } t > -t$ [Ax]
remainderBounds4
$H \vdash s \leq 0$ [Ax]
$H \vdash 0 < t$ [Ax]

$H \vdash s = (s \div t) * t + (s \text{ rem } t)$ [Ax]
divideRemainderSum
$H \vdash s = s \in \mathbb{Z}$ [Ax]
$H \vdash t \neq 0$ [Ax]

Regeln für ganze Zahlen III

$H \vdash \mathsf{ind}(u_1; x_1, f_{x1}.s_1; base_1; y_1, f_{y1}.t_1) = \mathsf{ind}(u_2; x_2, f_{x2}.s_2; base_2; y_2, f_{y2}.t_2) \in T[u_1/z]$ [Ax]

`indEquality` z T x f_x v

$H \vdash u_1 = u_2 \in \mathbb{Z}$ [Ax]

$H, x{:}\mathbb{Z}, v{:}\, x{<}0, f_x{:}T[(x{+}1)/z] \vdash s_1[x, f_x/x_1, f_{x1}] = s_2[x, f_x/x_2, f_{x2}] \in T[x/z]$ [Ax]

$H \vdash base_1 = base_2 \in T[0/z]$ [Ax]

$H, x{:}\mathbb{Z}, v{:}\, 0{<}x, f_x{:}T[(x{-}1)/z] \vdash t_1[x, f_x/y_1, f_{y1}] = t_2[x, f_x/y_2, f_{y2}] \in T[x/z]$ [Ax]

$H, z{:}\mathbb{Z}, \Delta \vdash C$ [ext $\mathsf{ind}(z;\ x, f_x.s[\mathsf{Ax}/v];\ base;\ x, f_x.t[\mathsf{Ax}/v])$]

`intElimination` i x f_x v

$H, z{:}\mathbb{Z}, \Delta, x{:}\mathbb{Z}, v{:}\, x{<}0, f_x{:}C[(x{+}1)/z] \vdash C[x/z]$ [ext s]

$H, z{:}\mathbb{Z}, \Delta \vdash C[0/z]$ [ext $base$]

$H, z{:}\mathbb{Z}, \Delta, x{:}\mathbb{Z}, v{:}\, 0{<}x, f_x{:}C[(x{-}1)/z] \vdash C[x/z]$ [ext t]

$H \vdash \mathsf{ind}(i; x, f_x.s; base; y, f_y.t) = t_2 \in T$ [Ax]

`indReduceBase`

$H \vdash base = t_2 \in T$ [Ax]

$H \vdash i = 0 \in \mathbb{Z}$ [Ax]

$H \vdash \mathsf{ind}(i; x, f_x.s; base; y, f_y.t) = t_2 \in T$ [Ax]

`indReduceDown`

$H \vdash t[i, \mathsf{ind}(i{+}1; x, f_x.s; base; y, f_y.t)/x, f_x] = t_2 \in T$ [Ax]

$H \vdash i < 0$ [Ax]

$H \vdash \mathsf{ind}(i; x, f_x.s; base; y, f_y.t) = t_2 \in T$ [Ax]

`indReduceUp`

$H \vdash t[i, \mathsf{ind}(i{-}1; x, f_x.s; base; y, f_y.t)/y, f_y] = t_2 \in T$ [Ax]

$H \vdash 0 < i$ [Ax]

Regeln für ganze Zahlen IV

$H \vdash$ if u_1=v_1 then s_1 else t_1 = if u_2=v_2 then s_2 else $t_2 \in T$ [Ax]

int_eqEquality

$H \vdash u_1 = u_2 \in \mathbb{Z}$ [Ax]

$H \vdash v_1 = v_2 \in \mathbb{Z}$ [Ax]

$H, v\colon u_1$=$v_1 \vdash s_1 = s_2 \in T$ [Ax]

$H, v\colon u_1 \neq v_1 \vdash t_1 = t_2 \in T$ [Ax]

$H \vdash$ if u=v then s else $t = t_2 \in T$ [Ax]

int_eqReduceTrue

$H \vdash s = t_2 \in T$ [Ax]

$H \vdash u = v \in \mathbb{Z}$ [Ax]

$H \vdash$ if u=v then s else $t = t_2 \in T$ [Ax]

int_eqReduceFalse

$H \vdash t = t_2 \in T$ [Ax]

$H \vdash u \neq v$ [Ax]

$H \vdash$ if u_1<v_1 then s else t = if u_2<v_2 then s_2 else $t_2 \in T$ [Ax]

lessEquality

$H \vdash u_1 = u_2 \in \mathbb{Z}$ [Ax]

$H \vdash v_1 = v_2 \in \mathbb{Z}$ [Ax]

$H, v\colon u_1$<$v_1 \vdash s_1 = s_2 \in T$ [Ax]

$H, v\colon u_1 \geq v_1 \vdash t_1 = t_2 \in T$ [Ax]

$H \vdash$ if u<v then s else $t = t_2 \in T$ [Ax]

lessReduceTrue

$H \vdash s = t_2 \in T$ [Ax]

$H \vdash u < v$ [Ax]

$H \vdash$ if u<v then s else $t = t_2 \in T$ [Ax]

lessReduceFalse

$H \vdash t = t_2 \in T$ [Ax]

$H \vdash u \geq v$ [Ax]

$H \vdash C$ [ext t_j]

arith j

$H \vdash s_j \in \mathbb{Z}$ [Ax]

Entscheidungsprozedur für elementare Arithmetik (Alup II). Nichtarithmetische Ausdrücke in C erzeugen Wohlgeformtheitsziele. Extraktterm entspricht logischer Struktur von C

Synthese eines Quadratwurzelprogramms

- **Algorithmus ist durch ∀-∃-Theorem spezifizierbar**
  - Formaler Beweis für $\forall n{:}\mathbb{N}.\, \exists r{:}\mathbb{N}.\, r^2{\leq}n \,\wedge\, n{<}(r{+}1)^2$ liefert Extrakt-Term p_{sqrt} der Form $\lambda n\,.\,\langle r\,,\langle pf_1\,,pf_2\rangle\rangle$ mit $r{=}\lfloor\sqrt{n}\rfloor$
  - Quadratwurzelprogramm kann aus Beweis extrahiert werden
    $\texttt{sqrt} \;\equiv\; \lambda\texttt{n}.\;\texttt{match}\; p_{sqrt}\texttt{(n)}\;\texttt{with}\;\langle\texttt{r}\,,\texttt{pf}\rangle \mapsto \texttt{r}$
- **Standardbeweis verwendet Induktion auf n**
  - Basis: $\vdash \exists r{:}\mathbb{N}.\, r^2{\leq}0 \,\wedge\, 0{<}(r{+}1)^2$
  - Schritt: $\exists r{:}\mathbb{N}.\, r^2{\leq}n \,\wedge\, n{<}(r{+}1)^2 \;\vdash\; \exists r{:}\mathbb{N}.\, r^2{\leq}n{+}1 \,\wedge\, n{+}1{<}(r{+}1)^2$
  - Lösung ist für beide Fälle leicht zu konstruieren
- **Formaler Beweis benötigt definitorische Erweiterungen**
  - Definitionen von $i{\leq}j \;\equiv\; i{<}j{+}1$ und $i^2 \;\equiv\; i{*}i$
  - Natürliche Zahlen als Teiltyp $\{x : \mathbb{Z} \mid 0{\leq}x\}$ von $\mathbb{Z}$ (Einheit 11)
- **Induktionsregel für $\mathbb{N}$ basiert auf Bibliothekslemma**
  $\forall\texttt{P}{:}\mathbb{N}{\to}\mathbb{P}.\;\; (\texttt{P(0)} \wedge (\forall\texttt{i}{:}\mathbb{N}^+.\,\texttt{P(i-1)} \Rightarrow \texttt{P(i)})) \;\Rightarrow\; \forall\texttt{i}{:}\mathbb{N}.\,\texttt{P(i)}$
  - Beweis mit Regel `intElimination`, Anwendung als Taktik `NatInd`

Beweis der Existenz von Quadratwurzeln

- **Formaler Beweis mit Standardinduktion**

$\vdash \forall n{:}\mathbb{N}.\ \exists r{:}\mathbb{N}.\ r^2 \le n < (r+1)^2$	allR
1. $n{:}\mathbb{N} \vdash \exists r{:}\mathbb{N}.\ r^2 \le n < (r+1)^2$	NatInd 1
1.1. $\vdash \exists r{:}\mathbb{N}.\ r^2 \le 0 < (r+1)^2$	existsR ⌈0⌉ THEN Auto ✓
1.2. $i{:}\mathbb{N}^+,\ r{:}\mathbb{N},\ r^2 \le i-1 < (r+1)^2 \vdash \exists r{:}\mathbb{N}.\ r^2 \le i < (r+1)^2$	Decide ⌈$(r+1)^2 \le i$⌉ THEN Auto
1.2.1. $i{:}\mathbb{N}^+,\ r{:}\mathbb{N},\ r^2 \le i-1 < (r+1)^2,\ (r+1)^2 \le i \vdash \exists r{:}\mathbb{N}.\ r^2 \le i < (r+1)^2$	existsR ⌈r+1⌉ THEN Auto ✓
1.2.2. $i{:}\mathbb{N}^+,\ r{:}\mathbb{N},\ r^2 \le i-1 < (r+1)^2,\ \neg((r+1)^2 \le i) \vdash \exists r{:}\mathbb{N}.\ r^2 \le i < (r+1)^2$	existsR ⌈r⌉ THEN Auto ✓

  – Taktik `Decide` erzeugt Fallunterscheidung für entscheidbare Prädikate

  – Taktik `Auto` enthält Entscheidungsprozedur `arith`

- **Extrahierter Algorithmus (nach Projektion)**

```
function sqrt n = if n=0 then 0
                  else let r = sqrt (n-1)  in
                  if (r+1)^2≤n  then r+1 else r
```

Algorithmus ist korrekt aber ineffizient

- **Laufzeit für $\lfloor\sqrt{n}\rfloor$ ist linear in Größe der Eingabe n**
  - Bereits für 10-stellige Zahlen unakzeptabel langsam
  - Man kann effizientere Algorithmen programmieren

  Elegante Beweise liefern nicht notwendig gute Programme
- **Suche Beweisstruktur mit effizienterem Extrakt**
  - Nutze Binärdarstellung der Zahlen und bestimme Lösung bitweise
  - Entspricht induktivem Beweis, der jeweils das nächste Bit konstruiert also $\lfloor\sqrt{n}\rfloor$ aus $\lfloor\sqrt{n}\rfloor \div 2$ bzw. aus $\lfloor\sqrt{n \div 4}\rfloor$ berechnet
- **Induktion benötigt Lemma für "4-adische Induktion"**

  $\forall P{:}\mathbb{N}\to\mathbb{P}.\ (P(0) \wedge (\forall i{:}\mathbb{N}^+.\, P(i \div 4) \Rightarrow P(i))) \Rightarrow \forall i{:}\mathbb{N}.\, P(i)$
  - Beweis mit Regel `intElimination` liefert ineffizienten Extraktterm
  - Besserer Beweis gibt effizientere Evidenz explizit zu Beginn an Regel `introduction` liefert Extrakt-Term mit Y-Kombinator
  - Taktik `NatInd4` wandelt Lemma in Inferenzregel um

Erzeugung eines effizienteren Algorithmus

- **Formaler Beweis mit Binärinduktion**

$\vdash \forall n{:}\mathbb{N}.\ \exists r{:}\mathbb{N}.\ r^2 \le n < (r+1)^2$	allR
1. $n{:}\mathbb{N} \vdash \exists r{:}\mathbb{N}.\ r^2 \le n < (r+1)^2$	NatInd4 1
1.1. $\vdash \exists r{:}\mathbb{N}.\ r^2 \le 0 < (r+1)^2$	existsR ⌈0⌉ THEN Auto ✓
1.2. $i{:}\mathbb{N}, r{:}\mathbb{N}, r^2 \le i \div 4 < (r+1)^2 \vdash \exists r{:}\mathbb{N}.\ r^2 \le i < (r+1)^2$	Decide ⌈$(2*r+1)^2 \le i$⌉ THEN Auto
1.2.1. $i{:}\mathbb{N}, r{:}\mathbb{N}, r^2 \le i \div 4 < (r+1)^2, ((2*r)+1)^2 \le i \vdash \exists r{:}\mathbb{N}.\ r^2 \le i < (r+1)^2$	existsR ⌈2*r+1⌉ THEN Auto ✓
1.2.2. $i{:}\mathbb{N}, r{:}\mathbb{N}, r^2 \le i \div 4 < (r+1)^2, \neg(((2*r)+1)^2 \le i) \vdash \exists r{:}\mathbb{N}.\ r^2 \le i < (r+1)^2$	existsR ⌈2*r⌉ THEN Auto ✓

  - Beweisstruktur identisch zu der des einfachen Beweises
  - Effizientere Induktionsstruktur führt zu besserer Komplexität

- **Extrahierter Algorithmus hat logarithmische Laufzeit**

```
function sqrt n = if n=0 then 0
                  else let r = sqrt (n÷4)  in
                  if (2*r+1)^2≤n  then 2*r+1 else 2*r
```

Listen

- **Grundform des Datencontainers**
  - Sammlung beliebig vieler Elemente eines Datentyps T
  - Elemente werden der Reihe nach in eine Liste eingefügt
  - Induktive Beschreibung startet mit leerer Liste und fügt Elemente jeweils vorne an eine bestehende Liste an
  - Simulierbar durch endliche Funktionen möglich, aber aufwendig

    $$T\,\mathsf{list} \quad \equiv \quad n:\mathbb{N} \times (i:\mathbb{Z} \times (1 \leq i \wedge i \leq n)) \to T$$

- **Formale Struktur ähnlich zur Peano-Arithmetik**
  - Neben der Größe (Zahl) werden Information zu Elementen angegeben
  - Grundkonstrukte: Typ $T\,\mathsf{list}$, Basiselement $[\,]$, Konstruktor $a::l$, Induktion (primitive Rekursion) $\mathsf{ListInd}[base\,;\ f[x\,,l\,,z]\,](t)$
    Langversion: $h(t)\ \mathsf{where}\ h([\,]) = base \mid h(a::L) = f[a, L, h(L)\,/\,x,l,z]$
  - Theoretisch und praktisch ausreichend:
    Alle wichtigen Listenoperationen sind hinreichend effizient darstellbar

Formalisierung eines Listentyps

- **Ausdrücke**

Kanonisch:	$T\,\mathsf{list}$	T Term
	$[\,]$	
	$a::l$	a, l Terme
Nichtkanonisch:	$\mathsf{ListInd}[base;\ f[x,l,z]](t)$	$t, base, g$ Terme, x, l, z Variablen

- **Reduktion von Ausdrücken**

$$\mathsf{ListInd}[base;\ f[x,l,z]]([\,]) \xrightarrow{\beta} base$$

$$\mathsf{ListInd}[base;\ f[x,l,z]](a::L) \xrightarrow{\beta} f[a, L, \mathsf{ListInd}[base;\ f[x,l,z]](L)/x, l, z]$$

- **Urteile für Typ- und Elementgleichheit**

$T_1\,\mathsf{list} = T_2\,\mathsf{list}$	falls $T_1 = T_2$
$[\,] = [\,] \in T\,\mathsf{list}$	falls T Typ
$a_1::l_1 = a_2::l_2 \in T\,\mathsf{list}$	falls T Typ und $a_1 = a_2 \in T$ und $l_1 = l_2 \in T\,\mathsf{list}$
$T_1\,\mathsf{list} = T_2\,\mathsf{list} \in \mathbb{U}_j$	falls $T_1 = T_2 \in \mathbb{U}_j$

Regeln für Listen

$\Gamma \vdash T_1\,\mathsf{list} = T_2\,\mathsf{list} \in \mathbb{U}_j$ [Ax]
`listEquality`
$\Gamma \vdash T_1 = T_2 \in \mathbb{U}_j$ [Ax]

$\Gamma \vdash \mathbb{U}_j$ [ext $T\,\mathsf{list}$]
`listFormation`
$\Gamma \vdash \mathbb{U}_j$ [ext T]

$\Gamma \vdash [] = [] \in T\,\mathsf{list}$ [Ax]
`nilEquality` j
$\Gamma \vdash T \in \mathbb{U}_j$ [Ax]

$\Gamma \vdash T\,\mathsf{list}$ [ext $[]$]
`nilFormation` j
$\Gamma \vdash T \in \mathbb{U}_j$ [Ax]

$\Gamma \vdash t_1 :: l_1 = t_2 :: l_2 \in T\,\mathsf{list}$ [Ax]
`consEquality`
$\Gamma \vdash t_1 = t_2 \in T$ [Ax]
$\Gamma \vdash l_1 = l_2 \in T\,\mathsf{list}$ [Ax]

$\Gamma \vdash T\,\mathsf{list}$ [ext $t :: l$]
`consFormation`
$\Gamma \vdash T$ [ext t]
$\Gamma \vdash T\,\mathsf{list}$ [ext l]

$\Gamma \vdash \mathsf{ListInd}[base_1; t_1[x_1, l_1, z_1]](s_1) = \mathsf{ListInd}[base_2; t_2[x_2, l_2, z_2]](s_2) \in T[s_1/z]$ [Ax]
`list_indEquality` z T $S\,\mathsf{list}$ x l z
$\Gamma \vdash s_1 = s_2 \in S\,\mathsf{list}$ [Ax]
$\Gamma \vdash base_1 = base_2 \in T[[]/z]$ [Ax]
$\Gamma, x{:}S, l{:}S\,\mathsf{list}, z{:}T[l/z] \vdash t_1[x, l, z/x_1, l_1, z_1] = t_2[x, l, z/x_2, l_2, z_2] \in T[x :: l/z]$ [Ax]

$\Gamma, z{:}T\,\mathsf{list}, \Delta \vdash C$ [ext $\mathsf{ListInd}[base; t[x, l, z]](z)$]
`listElimination` i z x l
$\Gamma, z{:}T\,\mathsf{list}, \Delta \vdash C[[]/z]$ [ext $base$]
$\Gamma, z{:}T\,\mathsf{list}, \Delta, x{:}T, l{:}T\,\mathsf{list}, z{:}C[l/z] \vdash C[x :: l/z]$ [ext t]

$\Gamma \vdash \mathsf{ListInd}[base; t[x, l, z]]([]) = t_2 \in T$ [Ax]
`list_indReduceBase`
$\Gamma \vdash base = t_2 \in T$ [Ax]

$\Gamma \vdash \mathsf{ListInd}[base; t[x, l, z]](s :: u) = t_2 \in T$ [Ax]
`list_indReduceUp`
$\Gamma \vdash t[s, u, \mathsf{ListInd}[base; t[x, l, z]](u)/x, l, z] = t_2 \in T$ [Ax]

Synthese einer Listenoperation

Bestimme das maximale Element einer Liste von Zahlen

- **Konstruktion durch einfachen Induktionsbeweis**
  - Spezifikationstheorem: $\forall L{:}\mathbb{Z}\,\mathsf{list}^{+}.\exists m{:}\mathbb{Z}.\ m \in L \wedge \forall x \in L.x \leq m$
  - Basis: $\vdash \forall a{:}\mathbb{Z}.\exists m{:}\mathbb{Z}.\ m \in [a] \wedge \forall x \in [a].x \leq m$
  - Schritt: $\exists m{:}\mathbb{Z}.\ m \in L \wedge \forall x \in L.x \leq m$
    $\vdash \forall a{:}\mathbb{Z}.\exists m{:}\mathbb{Z}.\ m \in a::L \wedge \forall x \in a::L.x \leq m$
  - Lösung in beiden Fällen leicht zu konstruieren
- **Formaler Beweis benötigt definitorische Erweiterungen**
  - $T\,\mathsf{list}^{+} \equiv \{l{:}T\,\mathsf{list} \mid l \neq [] \in T\,\mathsf{list}\}$
  - $[a] \equiv a::[]$
  - $x \in l \equiv h(l)$ where $h([]) = \mathsf{f} \mid h(a::L) = x{=}a \in \mathbb{Z} \vee x \in L$
  - $\forall x \in l.P(x) \equiv h(l)$ where $h([]) = \mathsf{t} \mid h(a::L) = P(a) \wedge \forall x \in L.P(x)$
- **Induktionsregel für $\mathbb{Z}\,\mathsf{list}^{+}$ basiert auf Bibliothekslemma**
  - $\forall \mathtt{P}:\mathbb{Z}\,\mathsf{list}^{+}\to\mathbb{P}.\ (\forall \mathtt{a}:\mathbb{Z}.\mathtt{P}([\mathtt{a}])$
    $\wedge\ (\forall \mathtt{l}:\mathbb{Z}\,\mathsf{list}^{+}.\forall \mathtt{a}:\mathbb{Z}.\ \mathtt{P(l)} \Rightarrow \mathtt{P(a::l)})) \Rightarrow \forall \mathtt{L}:\mathbb{Z}\,\mathsf{list}^{+}.\ \mathtt{P(L)}$
  - Beweis mit Regel `listElimination`, Anwendung als Taktik `List1Ind`

Synthese des Maximum-Algorithmus

- **Formaler Beweis des Spezifikationstheorems**

$\vdash \forall L{:}\mathbb{Z}\,\mathsf{list}^+.\exists m{:}\mathbb{Z}.\ m\in L \wedge \forall x\in L.x\leq m$	allR
1. $L{:}\mathbb{Z}\,\mathsf{list}^+ \vdash \exists m{:}\mathbb{Z}.\ m\in L \wedge \forall x\in L.x\leq m$	List1Ind 1 THEN allR
1.1. $L{:}\mathbb{Z}\,\mathsf{list}^+, a{:}\mathbb{Z} \vdash \exists m{:}\mathbb{Z}.\ m\in [a] \wedge \forall x\in [a].x\leq m$	existsR ⌈a⌉ THEN Auto ✓
1.2. $L{:}\mathbb{Z}\,\mathsf{list}^+, m{:}\mathbb{Z}, m\in L, \forall x\in L.x\leq m, a{:}\mathbb{Z} \vdash \exists m{:}\mathbb{Z}.\ m\in a::L \wedge \forall x\in a::L.x\leq m$	Decide ⌈m≤a⌉ THEN Auto
1.2.1. $L{:}\mathbb{Z}\,\mathsf{list}^+, m{:}\mathbb{Z}, m\in L, \forall x\in L.x\leq m, a{:}\mathbb{Z}, m\leq a \vdash \exists m{:}\mathbb{Z}.\ m\in a::L \wedge \forall x\in a::L.x\leq m$	existsR ⌈a⌉ THEN Auto ✓
1.2.2. $L{:}\mathbb{Z}\,\mathsf{list}^+, m{:}\mathbb{Z}, m\in L, \forall x\in L.x\leq m, a{:}\mathbb{Z}, \neg(m\leq a) \vdash \exists m{:}\mathbb{Z}.\ m\in a::L \wedge \forall x\in a::L.x\leq m$	existsR ⌈m⌉ THEN Auto ✓

  - Taktik `Auto` verwendet mehrere Bibliothekslemmata
    - $\forall P{:}\mathbb{Z}\,\mathsf{list}^+\to\mathbb{P}.\forall a{:}\mathbb{Z}.\ (\forall x\in [a].P(x)) \Leftrightarrow P(a)$
    - $\forall P{:}\mathbb{Z}\,\mathsf{list}^+\to\mathbb{P}.\forall l{:}\mathbb{Z}\,\mathsf{list}^+.\forall a{:}\mathbb{Z}.\ (\forall x\in a::L.P(x)) \Leftrightarrow (P(a)\wedge\forall x\in l.P(x))$
    - $\forall x,a{:}\mathbb{Z}.\ x\in [a] \Leftrightarrow x{=}a\in\mathbb{Z}$
    - $\forall l{:}\mathbb{Z}\,\mathsf{list}^+.\forall x,a{:}\mathbb{Z}.\ x\in a::L \Leftrightarrow x{=}a\in\mathbb{Z}\vee x\in l$

- **Extrahierter Algorithmus (nach Projektion)**

```
function max L = case L of   [a]   ↦ a
                           | a::l  ↦ let m = max l in
                                     if m≤a then a else m
```

Rekursive Definition

- **Größere Freiheit in der Formulierung**
  - Zahlen unterstützen induktive Beweise und primitive Rekursion
  - Unnatürliche Simulation rekursiver Datentypen (Bäume, Graphen,..)
  - Y-Kombinator unterstützt freie, schwer zu kontrollierende Rekursion

  Direkte Einbettung rekursiver Definition für bekannte Konstrukte
- **Induktive Typkonstruktoren**
  - Wohlfundierte, rekursiv definierte Datentypen und ihre Elemente
- **Partiell Rekursive Funktionen**
  - Totale rekursive Funktionen auf eingeschränktem Definitionsbereich
  - (Fast exakter) Definitionsbereich aus Algorithmus ableitbar
- **Lässige Typkonstruktoren**
  - Schließen über unendliche Objekte

Repräsentation rekursiv definierter Strukturen

- **Rekursive Typdefinition mit Gleichung $X = T[X]$**
  - z.B. `rectype bintree` = $\mathbb{Z} + \mathbb{Z} \times$ `bintree` $\times$ `bintree`
  - Kanonische Elemente definiert durch Aufrollen der Gleichung
  - Verarbeitung durch induktiven Operator $\mathsf{let}^* f(x) = t \ \mathsf{in} \ f(e)$ liefert terminierende freie rekursive Funktionsdefinitionen

```
sum(t) where
sum(b-tree) =
    case b-tree of inl(leaf) ↦ leaf
                 | inr(triple) ↦ match triple with ⟨num,pair⟩
                                 ↦ match pair with ⟨left,right⟩
                                   ↦ num+sum(left)+sum(right)
```

- **Parametrisierte simultane Rekursion möglich**
  - rectype $X_1(x_1)$ = T_{X_1} and .. and $X_n(x_n)$ = T_{X_n} select $X_i(a_i)$
  - Allgemeinste Form einer rekursiven Typdefinition

Induktive Typen, formal

- **Ausdrücke**

  Kanonisch: $\mathsf{rectype}\ X = T$ — T Terme, X Variable

  Nichtkanonisch: $\mathsf{let}^*\ f(x) = t\ \mathsf{in}\ f(e)$ — e, t Terme, f, x Variablen

- **Reduktion von Ausdrücken**

  $$\mathsf{let}^*\ f(x) = t\ \mathsf{in}\ f(e) \xrightarrow{\beta} t[\lambda y.\mathsf{let}^*\ f(x) = t\ \mathsf{in}\ f(y),\ e\ /\ f, x]$$

  Terminierung von $\mathsf{let}^*\ f(x) = t\ \mathsf{in}\ f(e)$ *verlangt* $e \in \mathsf{rectype}\ X = T[X]$

- **Urteile für Typ- und Elementgleichheit**

  $\mathsf{rectype}\ X_1 = T_1$

  $= \mathsf{rectype}\ X_2 = T_2$ falls $T_1[X/X_1] = T_2[X/X_2]$ für alle Typen X

  $s = t \in \mathsf{rectype}\ X = T_X$ falls $\mathsf{rectype}\ X = T_X$ Typ

  und $s = t \in T_X[\mathsf{rectype}\ X = T_X\ /\ X]$

Rahmenbedingungen für $\mathsf{rectype}\ X = T_X$

Induktive Definitionen müssen wohlfundiert sein

- **Semantik ist kleinster Fixpunkt von $T[X]$**
  - Existenz des Fixpunkts muß gesichert sein
    $T[X]$ muß Basisfall für Induktionsanfang enthalten
    Rekursiver Aufruf von X muß "natürliche" Elemente ermöglichen
  - Typen wie $\mathsf{rectype}\ \mathtt{X} = \mathtt{X} \to \mathbb{Z}$ müssen ausgeschlossen werden
    $\mathsf{rectype}\ \mathtt{X} = \mathtt{X} \to \mathbb{Z}$ hat $\lambda \mathtt{x}.\ \mathtt{x}\ \ \mathtt{x}$ als kanonisches Element
    $\lambda \mathtt{x}.\ \mathtt{x}\ \ \mathtt{x}$ wäre sogar Extrakt-Term von $\vdash\ \mathsf{rectype}\ \mathtt{X} = \mathtt{X} \to \mathbb{Z}$
- **Syntaktische Einschränkungen erforderlich**
  - Allgemeine Wohlfundiertheit rekursiver Typen ist unentscheidbar
    Entspricht dem Halteproblem rekursiver Programme
  - Kriterium: $T[X]$ darf X nur positiv enthalten
    Innerhalb von Funktionenräumen darf X nur "rechts" vorkommen

Regeln für induktive Typen

$H \vdash \mathsf{rectype}\ X_1 = T_{X1} = \mathsf{rectype}\ X_2 = T_{X2} \in \mathbb{U}_j$ [Ax]
`recEquality` X
$H, X{:}\mathbb{U}_j \vdash T_{X1}[X/X_1] = T_{X2}[X/X_2] \in \mathbb{U}_j$ [Ax]

$H \vdash s = t \in \mathsf{rectype}\ X = T_X$ [Ax]
`rec_memberEquality` j
$H \vdash s = t \in T_X[\mathsf{rectype}\ X = T_X/X]$ [Ax]
$H \vdash \mathsf{rectype}\ X = T_X \in \mathbb{U}_j$ [Ax]

$H \vdash \mathsf{rectype}\ X = T_X$ [ext t]
`rec_memberFormation` j
$H \vdash T_X[\mathsf{rectype}\ X = T_X/X]$ [ext t]
$H \vdash \mathsf{rectype}\ X = T_X \in \mathbb{U}_j$ [Ax]

$H \vdash \mathsf{let}^*\ f_1(x_1) = t_1\ \mathsf{in}\ f_1(e_1) = \mathsf{let}^*\ f_2(x_2) = t_2\ \mathsf{in}\ f_2(e_2) \in T[e_1/z]$ [Ax]
`rec_indEquality` $z\ T\ \mathsf{rectype}\ X = T_X\ j\ P\ f\ x$
$H \vdash e_1 = e_2 \in \mathsf{rectype}\ X = T_X$ [Ax]
$H \vdash \mathsf{rectype}\ X = T_X \in \mathbb{U}_j$ [Ax]
$H, P{:}(\mathsf{rectype}\ X = T_X) \to \mathbb{P}_j, f{:}(x{:}\{x : \mathsf{rectype}\ X = T_X \mid P(x)\} \to T[y/z]),$
$x{:}T_X[\{x : \mathsf{rectype}\ X = T_X \mid P(x)\}/X] \vdash t_1[f, x/f_1, x_1] = t_2[f, x/f_2, x_2] \in T[x/z]$ [Ax]

$H, z{:}\mathsf{rectype}\ X = T_X, H' \vdash C$ [ext $\mathsf{let}^*\ f(x) = t[\lambda y.\Lambda/P]\ \mathsf{in}\ f(z)$]
`recElimination` $i\ j\ P\ y\ f\ x$
$H, z{:}\mathsf{rectype}\ X = T_X, H' \vdash \mathsf{rectype}\ X = T_X \in \mathbb{U}_j$ [Ax]
$H, z{:}\mathsf{rectype}\ X = T_X, H', P{:}(\mathsf{rectype}\ X = T_X) \to \mathbb{P}_j, f{:}(y{:}\{x : \mathsf{rectype}\ X = T_X \mid P(x)\} \to C[y/z]),$
$x{:}T_X[\{x : \mathsf{rectype}\ X = T_X \mid P(x)\}/X] \vdash C[x/z]$ [ext t]

$H, z{:}\mathsf{rectype}\ X = T_X, H' \vdash C$ [ext $t[z/x]$]
`recUnrollElimination` $i\ x\ v$
$H, z{:}\mathsf{rectype}\ X = T_X, H', x{:}T_X[\mathsf{rectype}\ X = T_X/X], v{:}\, z{=}x \in T_X[\mathsf{rectype}\ X = T_X/X] \vdash C[x/z]$ [ext t]

Rekursive Funktionen

- **Definition von Funktionen durch $f(x) = t[f, x]$**
  - z.B. λf . let* $\mathtt{min}_f(\mathtt{y}) = \mathsf{if}\ \mathtt{f(y)=0}\ \mathsf{then}\ \mathtt{y}\ \mathsf{else}\ \mathtt{min}_f\ \mathtt{(y+1)}\ \mathsf{in}\ \mathtt{min}_f\mathtt{(0)}$
    λx . let* $\mathtt{sqr(y)} = \mathsf{if}\ \mathtt{x<(y+1)}^2\ \mathsf{then}\ \mathtt{y}\ \mathsf{else}\ \mathtt{sqr\ (y+1)}\ \mathsf{in}\ \mathtt{sqr(0)}$
  - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu let* $f(x) = t$ in $f(e)$ aber**
  - Keine Koppelung an bekannte rekursiver Struktur erforderlich
  - Kein Extraktterm einer Eliminationsregel
  - Flexiblerer Einsatz in Programmierung (“reale” Programme)
  - Benötigt Bestimmung der induktiven Struktur des Definitionsbereichs
- **Explizite Verankerung in Typentheorie bis Nuprl-3**
  - Datentyp der partiell-rekursiven Funktionen
  - Automatische Bestimmung einer Approximation des Definitionsbereichs
  - Logisch komplex und beschränkt auf Funktionen erster Stufe
- **Heute ersetzt durch Y-Kombinator**
  - Behandlung totaler Funktionen auf nachgewiesenem Definitionsbereich
  - Terminierungsbeweis durch Benutzer erforderlich

Lässige Typen

- **Repräsentation unendlicher Datenstrukturen**
  - Rekursive Definition durch die Gleichung $X = T[X]$

    z.B. $\texttt{inftree} = \mathbb{Z} \times \texttt{inftree} \times \texttt{inftree}$

    $\texttt{stream} = \mathsf{Atom} \times \texttt{stream}$
- **Formal ähnlich zum induktiven Datentyp**
  - Kanonische Elemente definiert durch Aufrollen der Gleichung
  - Andere Semantik für $\mathsf{inftype}\ X = T_X$: größter Fixpunkt von $T[X]$,
  - Kein Basisfall für Induktionsanfang erforderlich
  - Verarbeitung durch induktiven Operator $\mathsf{let}^{\infty}\ f(x) = t\ \mathsf{in}\ f(e)$
- **Parametrisierte simultane Rekursion möglich**

Kein fester Bestandteil der Nuprl Typentheorie