# Automatisierte Logik und Programmierung

## Einheit 10

## Fortgeschrittene Konzepte der CTT

1. Teilmengen- und Quotiententypen

2. Rekursive Datentypen

3. Durchschnitt, Vereinigung, starke Abhängigkeit

# TEILMENGENKONSTRUKT $\{x : S \mid P[x]\}$

- **Steigerung der praktischen Ausdruckskraft**
  - Mengentheories Standardkonzept mit klarer intuitiver Bedeutung
  - Liefert natürliche Repräsentation von Untertypen ($\mathbb{N}$, $T$ `List`$^+$, ...)
- **Konstruktive Interpretation schwierig**
  - Elemente sind die Elemente aus $S$, welche die Eigenschaft $P$ besitzen
  - Beweisterm für Eigenschaft $P$ darf kein Bestandteil des Elements sein
- **Formale Ähnlichkeit zu $x : S \times P[x]$**
  - Elemente sind Paare $\langle s, pf\rangle$ mit $s \in \{x : S \mid P[x]\}$
  - Beweiskomponente $pf \in P[s]$ bleibt Bestandteil des Elements
  - Entfernung des Beweisterms aus generierten Algorithmen mühsam
  - Evidenz für $P[s]$ ist bei Teilmengen nur implizit vorhanden
  - $x : S \times P[x]$ ist ungeeignet als Beschreibung von Teilmengen

**Teilmengentyp muß explizit repräsentiert werden**

# Teilmengen, formal

## Syntax:

Kanonisch: $\{x:S \mid T[x]\}$ $\quad$ **set**$\{\}(S;\,x.T[x])$

$\{S \mid T\}$ $\quad$ **set**$\{\}(S;\,.T)$

Nichtkanonisch: —

## Auswertung: —

## Semantik:

*Verletzt Prinzip der getrennten Definition von Typen*

$\{x_1:S_1 \mid T_1\} = \{x_2:S_2 \mid T_2\}$ $\equiv$ $S_1{=}S_2$ und es gibt Terme $p_1$, $p_2$ und eine Variable $x$, die weder in $T_1$ noch in $T_2$ vorkommt, so daß

$$p_1 \in \forall x:S_1.\, T_1[x/x_1] \Rightarrow T_2[x/x_2]$$

und $\quad p_2 \in \forall x:S_1.\, T_2[x/x_2] \Rightarrow T_1[x/x_1]$

$T = \{S_2 \mid T_2\}$ $\equiv$ $T = \{x_2:S_2 \mid T_2\}$ für ein beliebiges $x_2 \in \mathcal{V}$

$\{S_1 \mid T_1\} = T$ $\equiv$ $\{x_1:S_1 \mid T_1\}{=}T$ für ein beliebiges $x_1 \in \mathcal{V}$

$s = t \in \{x:S \mid T\}$ $\equiv$ $\{x:S \mid T\}$ Typ und $s = t \in S$ und es gibt einen Term $p$ mit der Eigenschaft $p \in T[s/x]$

Details im Appendix A.3.12 des Nuprl Manuals

## Teilmengen: Einfluss auf das Inferenzsystem

- **Verwaltung von implizitem Wissen erforderlich**
  - $\{x\!:\!T \mid P\}$ hat mehr Information als $T$ und weniger als $x\!:\!T\times P$
    - Nullstellenbestimmung ist verschieden aufwendig für Elemente von $\mathbb{Z}\to\mathbb{Z}$, $\texttt{f}:\mathbb{Z}\to\mathbb{Z}\times(\exists\texttt{y}:\mathbb{Z}.\,\texttt{f(y)}=\texttt{0})$ und $\{\texttt{f}:\mathbb{Z}\to\mathbb{Z} \mid \exists\texttt{y}:\mathbb{Z}.\,\texttt{f(y)=0}\}$
  - Für $s\in\{x\!:\!T \mid P\}$ wissen wir, daß $P[s]$ gilt, aber wir wissen nicht, wie ein Beweisterm für $P[s]$ konkret aussieht
  - Das Wissen $P[s]$ muß in Beweisen verwendbar sein, aber die Evidenz für $P[s]$ kann nicht algorithmisch verwendet werden
  - Ein Beweisterm für $P[s]$ darf nicht im Extraktterm vorkommen
- **Unterstütze versteckte Hypothesen**
  - Erzeugung durch Dekomposition der Annahme $z\!:\{x\!:\!S \mid T\}$

$$\Gamma,\ z\!:\{x\!:\!S \mid T\},\ \Delta \vdash C \ \lfloor \mathsf{ext}\ (\lambda y.t)\ z \rfloor$$

BY `setElimination` $i$ $y$ $v$

$$\Gamma,\ z\!:\{x\!:\!S \mid T\},\ y\!:\!S,\ [\![v]\!]\!:\!T[y/x],\ \Delta[y/z] \vdash C[y/z] \ \lfloor \mathsf{ext}\ t \rfloor$$

  - Freigabe in Teilzielen mit Extraktterm $\mathsf{Ax}$ (Gleichheiten, Kleiner-Relation)

# Wichtige benutzerdefinierte Mengentypen

- **Zahlenmengen und -intervalle:** Theory `int_1`

| | | |
|---|---|---|
| $\mathbb{N}$ | $\equiv \{\mathtt{i}:\mathbb{Z} \mid 0 \leq \mathtt{i}\}$ | **nat** |
| $\mathbb{N}^+$ | $\equiv \{\mathtt{i}:\mathbb{Z} \mid 0 < \mathtt{i}\}$ | **nat_plus** |
| $\{i\ldots\}$ | $\equiv \{\mathtt{j}:\mathbb{Z} \mid i \leq \mathtt{j}\}$ | **int_upper** |
| $\{\ldots i\}$ | $\equiv \{\mathtt{j}:\mathbb{Z} \mid \mathtt{j} \leq i\}$ | **int_lower** |
| $\{i..j\}$ | $\equiv \{\mathtt{k}:\mathbb{Z} \mid i \leq \mathtt{k} \leq j\}$ | **int_iseg** |
| $\{i..j^-\}$ | $\equiv \{\mathtt{k}:\mathbb{Z} \mid i \leq \mathtt{k} < j\}$ | **int_seg** |

- **Listen**

| | | |
|---|---|---|
| $T\ \mathtt{list}^+$ | $\equiv \{\mathtt{l}:T\ \mathtt{list} \mid 0 < \vert\vert\mathtt{l}\vert\vert\}$ | **listp** |

# Quotiententypen

- **Modifikation der Gleichheit auf Typen**
  - Rationale Zahlen: Paare ganzer Zahlen mit $\langle z_1, n_1\rangle = \langle z_2, n_2\rangle$,
    falls $z_1 * n_2 = z_2 * n_1$
  - Reelle Zahlen: konvergierende rationale Folgen mit gleichem Grenzwert
  - Restklassenräume: $\mathbb{Z}$ `mod` $k$
- **Faktorisierung modulo einer Äquivalenz**
  - Elemente $s$,$t$ werden aus Typ $T$ ausgewählt
  - Gleichheit von $s$ und $t$ wird über Äquivalenzrelation $E$ neu definiert
  - Benutzerdefinierte Gleichheit wird in das Typsystem eingebettet
  - Substitutions- und Gleichheitsregeln werden direkt anwendbar

$$\Downarrow$$

**Quotiententypen wichtig für formale Mathematik**

# Quotiententypen, formal

## Syntax:

Kanonisch: $x,y:T//E$ **quotient**{}($T$; $x$,$y$.$E$)

Nichtkanonisch: —

## Auswertung: —

## Semantik:

$x_1,y_1:T_1//E_1 = x_2,y_2:T_2//E_2 \quad\equiv\quad$ $T_1 = T_2$ und es gibt (verschiedene) Variablen $x, y, z$, die weder in $E_1$ noch in $E_2$ vorkommen, und Terme $p_1$, $p_2$, $r$,$s$ und $t$ mit der Eigenschaft

$p_1 \in \forall x{:}T_1.\forall y{:}T_1.\ E_1[x,y/x_1,y_1] \Rightarrow E_2[x,y/x_2,y_2]$

und $p_2 \in \forall x{:}T_1.\forall y{:}T_1.\ E_2[x,y/x_2,y_2] \Rightarrow E_1[x,y/x_1,y_1]$

und $r \in \forall x{:}T_1.E_1[x,x/x_1,y_1]$

und $s \in \forall x{:}T_1.\forall y{:}T_1.\ E_1[x,y/x_1,y_1] \Rightarrow E_1[y,x/x_1,y_1]$

und $t \in \forall x{:}T_1.\forall y{:}T_1.\forall z{:}T_1.$

$E_1[x,y/x_1,y_1] \Rightarrow E_1[y,z/x_1,y_1] \Rightarrow E_1[x,z/x_1,y_1]$

$s = t \in x,y:T//E \quad\equiv\quad$ $x,y:T//E$ Typ und $s \in T$ und $t \in T$ und es gibt einen Term $p$ mit der Eigenschaft $p \in E[s,t/x,y]$

Inferenzregeln und Taktiken im Appendix A.3.14 des Nuprl Manuals

# Wichtige benutzerdefinierte Quotiententypen

- **Rationale Zahlen**

| | | | |
|---|---|---|---|
| $x_1 \texttt{=}_q x_2$ | $\equiv$ | let $\langle z_1, n_1\rangle = x_1$ in let $\langle z_2, n_2\rangle = x_2$ in $z_1 \texttt{*} n_2 \texttt{=} z_2 \texttt{*} n_1$ | **rat_equal** |
| $\mathbb{Q}$ | $\equiv$ | $\texttt{x,y} : \mathbb{Z}\times\mathbb{N}^+ // \texttt{x=}_q\texttt{y}$ | **rat** |
| $x_1 \texttt{+} x_2$ | $\equiv$ | let $\langle z_1, n_1\rangle = x_1$ in let $\langle z_2, n_2\rangle = x_2$ in $\langle z_1 \texttt{*} n_2 \texttt{+} z_2 \texttt{*} n_1, n_1 \texttt{*} n_2\rangle$ | **rat_add** |
| $x_1 \texttt{-} x_2$ | $\equiv$ | let $\langle z_1, n_1\rangle = x_1$ in let $\langle z_2, n_2\rangle = x_2$ in $\langle z_1 \texttt{*} n_2 \texttt{-} z_2 \texttt{*} n_1, n_1 \texttt{*} n_2\rangle$ | **rat_sub** |
| $x_1 \texttt{*} x_2$ | $\equiv$ | let $\langle z_1, n_1\rangle = x_1$ in let $\langle z_2, n_2\rangle = x_2$ in $\langle z_1 \texttt{*} z_2, n_1 \texttt{*} n_2\rangle$ | **rat_mul** |
| $x_1 \texttt{<}_q x_2$ | $\equiv$ | let $\langle z_1, n_1\rangle = x_1$ in let $\langle z_2, n_2\rangle = x_2$ in $z_1 \texttt{*} n_2 \texttt{<} z_2 \texttt{*} n_1$ | **rat_lt** |

- **Restklassenräume**

| | | | |
|---|---|---|---|
| $x_1 \texttt{=} x_2 \ \texttt{mod}\ k$ | $\equiv$ | $k\ \texttt{divides}\ x_1 \texttt{-} x_2$ | **eqmod** |
| $\mathbb{Z}\ \texttt{mod}\ k$ | $\equiv$ | $\texttt{x,y} : \mathbb{Z} // \texttt{x=y mod}\ k$ | **int_mod** |

# QUOTIENTENTYPEN: EINFLUSS AUF DAS INFERENZSYSTEM

- **Gleichheit $E[s,t\,/\,x,y]$ ist implizites Wissen**
  - Wir wissen $E[s,t/x,y]$ wenn $s = t \in x,y : T//E$
  - Beweisterm für $E[s,t/x,y]$ darf nicht algorithmisch verwendet werden
  - Dekomposition obiger Gleichheit muß versteckte Hypothesen erzeugen

$$\Gamma,\, v{:}\, s = t \in x,y : T//E\,,\, \Delta \vdash C \;\lfloor\mathsf{ext}\ u\rfloor$$

$$\begin{array}{l}\texttt{BY quotient\_equalityElimination}\ i\ j\ v' \\ \Gamma,\, v{:}\, s = t \in x,y : T//E\,,\, [\![v']\!]{:}E[s,t/x,y],\, \Delta \vdash C \;\lfloor\mathsf{ext}\ u\rfloor \\ \Gamma,\, v{:}\, s = t \in x,y : T//E\,,\, \Delta \vdash E[s,t/x,y] \in \mathbb{U}_j \;\lfloor\mathsf{Ax}\rfloor\end{array}$$

  - Freigabe versteckter Hypothesen wie zuvor

# TYPE-SQUASHING

- **Überstrukturierung auf $x\,,y : T//E$ möglich**
  - $x_1 <_q x_2$ muß wohlgeformt sein
  - Gilt $x_1 <_q x_2 = x_1' <_q x_2'$, wenn $x_1 = x_1' \in \mathbb{Q}$ und $x_2 = x_2' \in \mathbb{Q}$?
    - $z_1 * n_2 < z_2 * n_1 = z_1' * n_2' < z_2' * n_1'$ verlangt $z_1 * n_2 = z_1' * n_2' \in \mathbb{Z}$ und $z_2 * n_1 = z_2' * n_1' \in \mathbb{Z}$
    - Nicht gültig für $x_1 = \langle 2\,,1\rangle$, $x_1' = \langle 4\,,2\rangle$, $x_2 = x_2' = \langle 3\,,1\rangle$
  - Definition von $<_q$ enthält zu viel Struktur, wo nur "Wahrheit" nötig ist
  - Unabhängigkeit vom Repräsentanten nicht mehr gegeben

- **Type-Squashing für benutzerdefinierte Prädikate**
  - $\downarrow\!P \;\equiv\; \{0 \in \mathbb{Z} \mid P\}$
  - Reduziert Struktur des Prädikats (Typs) $P$ auf "Wahrheitstyp"
  - Entfernt Überstrukturierung aus Definition von Prädikaten

# Definition Reeller Zahlen mit Quotiententypen

| | | | |
|---|---|---|---|
| $x_1 <_q x_2$ | $\equiv$ | $\downarrow$let $\langle z_1, n_1\rangle = x_1$ in let $\langle z_2, n_2\rangle = x_2$ in $z_1 * n_2 < z_2 * n_1$ | **rat_lt** |
| $x_1 \leq_q x_2$ | $\equiv$ | $x_1 < x_2 \;\vee\; x_1 = x_2 \in \mathbb{Q}$ | **rat_le** |
| $z/n$ | $\equiv$ | $\langle z, n\rangle$ | **rat_frac** |
| $\lvert x \rvert$ | $\equiv$ | let $\langle z, n\rangle = x$ in if $z<0$ then $\langle -z, n\rangle$ else $\langle z, n\rangle$ | **rat_abs** |
| $\mathbb{R}_{pre}$ | $\equiv$ | $\{\mathtt{f}:\mathbb{N}^+ \to \mathbb{Q} \mid \forall \mathtt{m},\mathtt{n}:\mathbb{N}^+.\ \lvert \mathtt{f(n)} - \mathtt{f(m)} \rvert \leq \mathtt{1/m+1/n}\}$ | **real_pre** |
| $x_1 =_r x_2$ | $\equiv$ | $\forall \mathtt{n}:\mathbb{N}^+.\ \lvert x_1\mathtt{(n)} - x_2\mathtt{(n)} \rvert \leq \mathtt{2/n}$ | **real_equal** |
| $\mathbb{R}$ | $\equiv$ | $\mathtt{x,y} : \mathbb{R}_{pre} // \mathtt{x} =_r \mathtt{y}$ | **real** |
| $x_1 + x_2$ | $\equiv$ | $\lambda\mathtt{n}.\ x_1\mathtt{(n)} + x_2\mathtt{(n)}$ | **real_add** |
| $x_1 - x_2$ | $\equiv$ | $\lambda\mathtt{n}.\ x_1\mathtt{(n)} - x_2\mathtt{(n)}$ | **real_sub** |
| $\lvert x \rvert$ | $\equiv$ | $\lambda\mathtt{n}.\ \lvert x\mathtt{(n)} \rvert$ | **real_abs** |

**Elegante Beweise erfordern viele Spezialtaktiken**

# 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
  $\mapsto$ 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

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

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

# Induktive Typen, formal

## Syntax:

Kanonisch: $\textsf{rectype}\ X = T_X$ $\qquad$ $\textbf{rec}\{\}(X\,.\,T_X)$

Nichtkanonisch: $\textsf{let}^*\ f(x) = t\ \ \textsf{in}\ \ f(e)$ $\qquad$ $\textbf{rec\_ind}\{\}(e;\ f,x\,.\,t)$

## Auswertung:

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

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

## Semantik:

$\textsf{rectype}\ X_1 = T_{X1}$

$= \textsf{rectype}\ X_2 = T_{X2}$ $\quad\equiv\quad$ $T_{X1}[X/X_1] = T_{X2}[X/X_2]$ für alle Typen $X$

$s = t \in \textsf{rectype}\ X = T_X$ $\quad\equiv\quad$ $\textsf{rectype}\ X = T_X$ Typ

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

Inferenzregeln und Taktiken im Appendix A.3.11 des Nuprl Manuals

# RAHMENBEDINGUNGEN FÜR 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 rectype $\mathtt{X} = \mathtt{X} \rightarrow \mathbb{Z}$ müssen ausgeschlossen werden
    - rectype $\mathtt{X} = \mathtt{X} \rightarrow \mathbb{Z}$ hat $\lambda\mathtt{x.\ x\ x}$ als kanonisches Element
    - $\lambda\mathtt{x.\ x\ x}$ wäre sogar Extrakt-Term von $\vdash$ rectype $\mathtt{X} = \mathtt{X} \rightarrow \mathbb{Z}$
- **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

# REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch** $f(x) = t[f,x]$
  - z.B. $\lambda$`f`. let* `min`$_f$`(y)` = if `f(y)=0 then y else min`$_f$ `(y+1) in min`$_f$`(0)`
    $\lambda$`x`. let* `sqr(y)` = if `x<(y+1)`$^2$ `then y else sqr (y+1) in 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 CTT 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**
  - Formale Notation letrec $f(x) = t$ ist Abkürzung für $\mathbf{Y}(\lambda f.\lambda x.t)$
  - Terminierungsbeweis durch Benutzer erforderlich

# Lässige Typen

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

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

    `stream = Atom` $\times$ `stream`

- **Formal ähnlich zum induktiven Datentyp**
  - Kanonische Elemente definiert durch Aufrollen der Gleichung
  - Andere Semantik für 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 CTT**

# Neuere Typkonstrukte der CTT (I)

- **Durchschnitt $\cap x\!:\!S.T[x]$** — Appendix A.3.13 des Nuprl Manuals
  - Verallgemeinerung des einfachen Durchschnitts $S\cap T$
    - Durchschnitt einer Familie von Datentypen
    - Elemente müssen für alle $x\in S$ zum Typ $T[x]$ gehören
  - Strukturell ähnlich zu $x{:}S\rightarrow T[x]$ (aber andere Semantik)
  - Gut für Formalisierung von Record-Strukturen in Programmiersprachen
  - Ermöglicht Definition eines Typs aller Terme
    - `Top` $\equiv$ $\cap$x:Void.Void
  - Ermöglicht Definition von guarded types $\cap x\!:\!S.T$ ($T$ nicht abhängig von $x$)
    - Codiert Aussage: "$T$ ist ein Typ, wenn $S$ Elemente hat"
    - Nützlich für Wohlgeformtheitsbeweise zu mengentheoretischen Aussagen
- **Abhängiger Durchschnitt $x\!:\!S\cap T[x]$** — Bisher nur in MetaPRL
  - Element $s$ muß zu $S$ und gleichzeitig zu $T[s]$ gehören (Selbstreferenz!)
  - Gut für Formalisierung von Abstrakten Datentypen,
    Record-Strukturen mit internen Abhängigkeiten, Objekten

# Neuere Typkonstrukte der CTT (II)

- **Vereinigung $\cup x{:}S.T[x]$** Bisher nur in MetaPRL
  - Vereinigung einer Familie von Datentypen
  - Elemente müssen zu einem $T[x]$ mit $x \in S$ gehören
  - Elementgleichheit schwierig bei Überlappungen der Typen
- **Squiggle-Equality $s\tilde{}t$**
  - Einfacherer, syntaktischer Gleichheitstyp, ohne Abhängigkeit vom Typ
    - $s\tilde{}t$ gilt, wenn $s$ und $t$ zum gleichen Term reduzierbar sind oder in $\mathbb{Z}$ oder Atom semantisch gleich sind
  - Substitutionregel gilt auch für Terme die squiggle-gleich sind
- **Stark abhängige Funktionen $\{f \mid x{:}S \to T[f,x]\}$**
  - Selbstreferenz: Bildbereich hängt ab von Eingabe und Funktion $f$ selbst
  - Mächtiger als abhängiger Durchschnitt, aber Beweise werden aufwendig
- **Aktuell in Entwicklung**
  - Logic of Events: Schließen über Kommunikation und verteilte Prozesse
  - Reflektion: Schließen über Beweisverfahren und das Meta-Level der CTT

# Inferenzregeln zum praktischen Arbeiten

- **Direkte Berechnung**
  – Reduktion an beliebiger Stelle in Sequenz
  – Rückwärtsreduktion (vom Kontraktum zum Redex) möglich
- **Falten und Auffalten von Definitionen**
  – Einsetzen der Definition für eine benutzerdefiniere Abstraktion
  – Zugriff über den Namen des Definitionsobjektes in der Library
- **Anwendung von Lemmata**
  – Einsetzen der Konklusion oder des Extraktterms
  – Zugriff über den Namen des Lemmas in der Library
- **Entscheidungsprozeduren**
  – Bisher nur für Arithmetik und Gleichheit
- **Umbenennung und Ausdünnen**
  – Übersichtlichkeit in der Beweisführung

Siehe Appendix A.3.15 / A.3.16 des Nuprl Manuals

# ÜBERSICHT: STANDARDTYPEN DER CTT

| | | |
|---|---|---|
| Function Space | $S \to T$, $x{:}S \to T$ | $\lambda x.t$, $f\,t$ |
| Product Space | $S \times T$, $x{:}S \times T$ | $\langle s,t\rangle$, let $\langle x,y\rangle = e$ in $u$ |
| Disjoint Union | $S+T$ | inl$(s)$, inr$(t)$, case $e$ of inl$(x) \mapsto u$ \| inr$(y) \mapsto v$ |
| Universes | $\mathbb{U}_j$ | — *types of level $j$* — |
| Equality | $s = t \in T$ | Ax |
| Empty Type | Void | any$(x)$, — *no members* — |
| Atoms | Atom | "*token*", if $a$=$b$ then $s$ else $t$ |
| Numbers | $\mathbb{Z}$ | 0,1,-1,2,-2,... $s$+$t$, $s$-$t$, $s$*$t$, $s$÷$t$, $s$ rem $t$,<br>if $a$=$b$ then $s$ else $t$, if $i$<$j$ then $s$ else $t$<br>ind$(u;\ x,f_x.s;\ base;\ y,f_y.t)$ |
| | $i$<$j$ | Ax |
| Lists | $S$ list | [], $t$::$list$, list_ind$(L;\ base;\ x,l,f_l.t)$ |
| Inductive Types | rectype $X = T[X]$ | let$^*$ $f(x) = t$ in $f(e)$, — *members defined by $T[X]$* — |
| Subset | $\{x{:}S \mid P[x]\}$, | — *some members of $S$* — |
| Intersection | $\cap x{:}S.T[x]$,<br>$x{:}S \cap T[x]$ | — *members that occur in all $T[x]$* —<br>— *members $x$ that occur $S$ and $T[x]$* — |
| Union | $\cup x{:}S.T[x]$ | — *members that occur in some $T[x]$, tricky equality*— |
| Quotient | $x,y : S//E[x,y]$ | — *members of $S$, new equality* — |
| Very Dep. Functions | $\{f \mid x{:}S \to T[f,x]\}$ | |
| Squiggle Equality | $s$˜$t$ | — *a "simpler" equality* |

# Typentheorie im Rückblick

## Inferenzkalkül für Mathematik & Programmierung

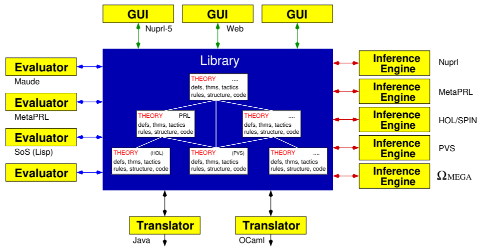
- **Extrem ausdrucksstarkes Inferenzsystem**
  - Vereinheitlicht und erweitert Logik, $\lambda$-Kalkül und einfache Typentheorie
  - Formalisierung "natürlicher" Gesetze der zentralen Konzepte
  - Direkte Darstellung anstatt Simulation
  - Umfangreiche Theorie bestehend aus
    - Mathematischen Grundkonzepten
    - Grundkonstrukten der Programmierung, einschließlich Rekursion
    - Prädikatenlogik (höherer Stufe)
- **Praktische Probleme**
  - Beweise erfordern viel Schreibarbeit $\longrightarrow$ *Interaktive Beweissysteme*
  - Beweise sind unübersichtlich (viele Regelanwendungen)
  - Beweise sind schwer zu finden (viele Regeln und Parameter)

$\longrightarrow$ *Automatisierung der Beweisführung*

# Ausblick: Themen des zweiten Teils (Sommer 2006)

## Konstruiere semiautomatische Beweissysteme

- **Aufbau von Beweissystemen**
  - Implementierung interaktiver Beweisassistenten
  - Das NuPRL Logical Programming Environment




- **Beweisautomatisierung**
  - Taktisches Beweisen
  - Entscheidungsprozeduren
  - Integration externer Systeme

- **Anwendungen & Demonstrationen**
  - Entwicklung formaler Theorien
  - Programmsynthese

⋮