Automatisierte Logik und Programmierung

Einheit 11

Fortgeschrittene Konzepte der Typentheorie

1. “Nichtkonstruktive” Datentypen
2. Rekursive Datentypen
3. Neuere Konstrukte

Mathematik benutzt nichtkonstruktive Konzepte

- **Mengentheoretische Operationen**
  - Teilmengenbildung $\{\, x{:}T \mid P[x] \,\}$
  - Vereinigung $S \cup T$ und Durchschnitt $S \cap T$ zweier Mengen
  - Vereinigung/Durchschnitt von Mengenfamilien $\bigcup_{x \in S} T[x]$, $\bigcap_{x \in S} T[x]$
  - Elemente dieser Typen sind Elemente der ursprünglichen Mengen
  - Informationen über Herkunft oder Selektionsprädikat sind nur implizit vorhanden (und nicht etwa in den Elementen gespeichert)
- **Äquivalenzklassen**
  - Äquivalenzrelation E induziert Klassen von Elementen eines Typs T
  - $[x]_E$ ist die Menge der zu x unter E äquivalenten Elemente
  - Menge der Klassen bildet einen neuen Typ $T//E$
  - Elemente von T sind Repräsentanten der Elemente von $T//E$
  - E induziert Gleichheit auf $T//E$: $[x]_E = [y]_E$ gilt g.d.w. $E(x, y)$

Mit bisherigen Typkonstrukten nicht simulierbar

Hilfreiche vordefinierte Typen

Konservative Definition gängiger Grundkonstrukte

- **Singulärer Typ:** $\texttt{Unit} \equiv 0 \in \mathbb{Z}$
  - Einziges Element ist Ax
  - Andere Bezeichnung: t ("der Wahrheitstyp" Gegenstück zu f)
- **Boolescher Datentyp:** $\mathbb{B} \equiv \texttt{Unit}+\texttt{Unit}$
  - $\mathsf{T} \equiv \mathsf{inl}(\mathsf{Ax})$, $\mathsf{F} \equiv \mathsf{inr}(\mathsf{Ax})$
  - if b then s else t $\equiv$ case b of $\mathsf{inl}(x) \mapsto s \mid \mathsf{inr}(y) \mapsto t$
  - Einbettung boolescher Werte in logische Prädikate durch Lifting:
    $\uparrow b \equiv \texttt{if } b \texttt{ then t else f}$
- **Rekursive Funktionen definiert durch Y-Kombinator**
  - $\mathsf{Y} \equiv \lambda \texttt{f. } (\lambda \texttt{x.f (x x)}) \; (\lambda \texttt{x.f (x x)})$
  - function $f(x) = t \equiv \mathsf{Y}(\lambda f . \lambda x . t)$ (geschlossener Term für f)
  - Nicht jede rekursiv definierte Funktion ist typisierbar
    Typ von function $f(x) = t$ muß im Kontext nachgewiesen werden

Teilmengenkonstrukt $\{x:S \mid P[x]\}$

- **Standardkonzept mit klarer intuitiver Bedeutung**
  - Ermöglicht natürliche Repräsentation von Teiltypen ($\mathbb{N}$, T $\texttt{List}^+$, ...) ohne ständige explizite Verwendung einschränkender Prädikate
  - z.B. $\forall \texttt{n}:\mathbb{N}.\ \exists \texttt{r}:\mathbb{N}.\ \texttt{r}^2 \leq \texttt{n} < (\texttt{r}+1)^2$ anstatt
    $\forall \texttt{n}:\mathbb{Z}.\ 0 \leq \texttt{n} \ \Rightarrow\ \exists \texttt{r}:\mathbb{Z}.\ 0 \leq \texttt{r} \wedge \texttt{r}^2 \leq \texttt{n} < (\texttt{r}+1)^2$
- **Konstruktive Interpretation von $\{x:S \mid P[x]\}$ schwierig**
  - Elemente sind Elemente aus $s \in S$, welche Eigenschaft $P[s]$ besitzen
  - Evidenz für $P[s]$ nur implizit vorhanden (kein Bestandteil des Elements)
- **Formale Ähnlichkeit zu $x:S \times P[x]$ nutzt wenig**
  - Elemente von $x:S \times P[x]$ sind Paare $\langle s, pf \rangle$ mit $s \in \{x:S \mid P[x]\}$
  - Evidenz $pf \in P[s]$ bleibt Bestandteil des Elements
  - Führt zu unnatürlichen Spezifikationen vieler Algorithmen
    Nullstellen finden $\{f{:}\mathbb{N}\to\mathbb{N} \mid \exists r{:}\mathbb{N}.\ f(r)=0\} \to \mathbb{N}$ (Suche nötig)
    würde zu $f{:}(\mathbb{N}\to\mathbb{N}) \times \exists r{:}\mathbb{N}.\ f(r)=0 \to \mathbb{N}$ (Projektion reicht)

$x:S \times P[x]$ ist zur Beschreibung von Teilmengen nicht geeignet

Der Teilmengenkonstruktor in der Typentheorie

- **Formalisierung der mengentheoretischen Aspekte**
  - Elemente von $\{x : S \mid P[x]\}$ sind Elemente von S
  - Es muß Evidenz für $P[s]$ geben, die aber nicht explizit bekannt ist
- **Sonderform des unabhängigen Konstruktors $\{S \mid P\}$**
  - Identisch mit S, wenn es Evidenz für P gibt, sonst leer
- **Häufigste Anwendung: Type-Squashing**
  - Definere $\downarrow P \equiv \{\mathsf{t} \mid P\}$

    Reduziert Struktur des Prädikats (Typs) P auf den "Wahrheitstyp" t
  - Wenn P nicht leer ist, enthält $\downarrow P$ nur das Element Ax

    Die Struktur der Elemente von P wird entfernt

    Alle Elemente von $\downarrow P$ sind gleich und besagen "P besitzt Evidenz"
  - Entfernt Überstrukturierung aus benutzerdefinierten Prädikaten

(siehe Folie 24)

Formulierung des Teilmengentyps

- **Ausdrücke**

  Kanonisch: $\{x : S \mid T\}$ S, T Terme, x Variable

  $\{S \mid T\}$ S, T Terme

  Nichtkanonisch: —

- **Reduktion von Ausdrücken**

  – entfällt, da keine kanonischen Elemente für den Teilmengenkonstruktor definiert

- **Urteile für Typ- und Elementgleichheit**

  $\{x_1 : S_1 \mid T_1\} = \{x_2 : S_2 \mid T_2\}$ falls $S_1 = S_2$ 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 $$p_2 \in \forall x : S_1 .\ T_2[x/x_2] \Rightarrow T_1[x/x_1]$$

  $\{x_1 : S_1 \mid T_1\} = \{x_2 : S_2 \mid T_2\} \in \mathbb{U}_j$ analog

  Prinzip der getrennten Definition von Typen kann nicht eingehalten werden

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

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

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

Wichtige benutzerdefinierte Teilmengentypen

- **Zahlenmengen und -intervalle:**

$$\mathbb{N} \equiv \{\mathtt{i}:\mathbb{Z} \mid 0 \leq \mathtt{i}\}$$

$$\mathbb{N}^{+} \equiv \{\mathtt{i}:\mathbb{Z} \mid 0 < \mathtt{i}\}$$

$$\{i\ldots\} \equiv \{\mathtt{j}:\mathbb{Z} \mid i \leq \mathtt{j}\}$$

$$\{\ldots i\} \equiv \{\mathtt{j}:\mathbb{Z} \mid \mathtt{j} \leq i\}$$

$$\{i..j\} \equiv \{\mathtt{k}:\mathbb{Z} \mid i \leq \mathtt{k} \leq j\}$$

$$\{i..j^{-}\} \equiv \{\mathtt{k}:\mathbb{Z} \mid i \leq \mathtt{k} < j\}$$

- **Listen**

$$T\,\mathsf{list}^{+} \equiv \{\mathtt{l}:T\,\mathsf{list} \mid \mathtt{l} \neq [\,] \in T\,\mathsf{list}\}$$

Behandlung von Teilmengen im 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
    $\mathbb{N}\to\mathbb{N}$, $\{f{:}\mathbb{N}\to\mathbb{N} \mid \exists r{:}\mathbb{N}.\, f(r)=0\}$ und $f{:}(\mathbb{N}\to\mathbb{N})\times\exists r{:}\mathbb{N}.\, f(r)=0$
  - Für $s\in\{x:T \mid P\}$ wissen wir, daß $P[s]$ gilt, aber wir
    wissen nicht, wie die Evidenz für $P[s]$ konkret aussieht
  - In Beweisen müssen wir das Wissen $P[s]$ verwenden können,
    ohne die Evidenz für $P[s]$ algorithmisch einzusetzen

  Evidenz für $P[s]$ darf nicht im Extraktterm vorkommen

- **Unterstütze versteckte Hypothesen**
  - Erzeugung durch Dekomposition der Annahme $z{:}\{x:S \mid T\}$

$$\Gamma, z{:}\{x:S \mid T\}, \Delta \vdash C \qquad \lfloor \mathsf{ext}\ (\lambda y.t)\ z \rfloor$$
$$\texttt{setElimination}\ i\ y\ v$$
$$\Gamma, z{:}\{x:S \mid T\}, y{:}S, [\![v]\!]{:}T[y/x], \Delta[y/z] \vdash C[y/z] \qquad \lfloor \mathsf{ext}\ t \rfloor$$

  - Freigabe der Hypothese nur in Teilzielen mit Extraktterm Ax

(Gleichheiten, Kleiner-Relation)

Regeln für Teilmengentypen

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

$\Gamma \vdash \mathbb{U}_j$ [ext $\{x : S \mid T\}$]
`setFormation` S x
$\Gamma \vdash S \in \mathbb{U}_j$ [Ax]
$\Gamma, x{:}S \vdash \mathbb{U}_j$ [ext T]

$\Gamma \vdash s = t \in \{x : S \mid T\}$ [Ax]
`set_memberEquality` j x'
$\Gamma \vdash s = t \in S$ [Ax]
$\Gamma \vdash T[s/x]$ [Ax]
$\Gamma, x'{:}S \vdash T[x'/x] \in \mathbb{U}_j$ [Ax]

$\Gamma \vdash \{x : S \mid T\}$ [ext s]
`set_memberFormation` j s x'
$\Gamma \vdash s \in S$ [Ax]
$\Gamma \vdash T[s/x]$ [Ax]
$\Gamma, x'{:}S \vdash T[x'/x] \in \mathbb{U}_j$ [Ax]

$\Gamma, z{:}\{x : S \mid T\}, \Delta \vdash C$ [ext $(\lambda y.t)$ z]
`setElimination` i y v
$\Gamma, z{:}\{x : S \mid T\}, y{:}S, [\![v]\!]{:}T[y/x], \Delta[y/z] \vdash C[y/z]$ [ext t]

Logische Interpretation des Teilmengentyps

- **$\{x : S \mid P[x]\}$ verhält sich wie ein Existenzquantor**
  - $\{x : S \mid P[x]\}$ ist genau dann nicht leer, wenn $\exists x{:}S.P[x]$ gültig ist
  - Elemente $s \in \{x : S \mid P[x]\}$ sind Zeugen für Gültigkeit von $\exists x{:}S.P[x]$
    Evidenz für $P[s]$ existiert, wird aber unterdrückt
  - $\{x : S \mid P[x]\}$ beschreibt nur den algorithmischen Aspekt des Quantors
  - $\forall$-$\exists$-Theorem auf Basis des Teilmengentyps liefern bessere Extraktterme
- **$\{P \mid Q\}$ verhält sich ähnlich wie die Konjunktion $P \wedge Q$**
  - $\{P \mid Q\}$ hat Elemente, wenn P und Q gültig sind
  - Elemente $p \in \{P \mid Q\}$ sind Elemente von P (partielle Evidenz)
    Evidenz für Q existiert, wird aber unterdrückt
  - $\{P \mid Q\}$ beschreibt eine nichtkommutative Konjunktion
  - $\{P \mid \mathsf{t}\}$ ist isomorph zu P
  - $\{\mathsf{t} \mid Q\}$ hat genau dann Elemente, wenn dies für Q gilt
    aber alle Elemente von $\{\mathsf{t} \mid Q\}$ sind gleich (“Squashing”)

Quadratwurzelalgorithmus mit Teilmengentypen

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

- **Beweis hat die gleiche Struktur wie zuvor**
  - set_memberR ist Taktik für set_memberFormation
  - Versteckte Hypothese $r^2 \leq i \div 4 < (r+1)^2$ wird für die Beweisziele $r^2 \leq i$ und $i < (r+1)^2$ freigegeben,
  - Extrakt-Term enthält keine Beweiskomponenten
  - Effizienter Algorithmus für $\lfloor \sqrt{n} \rfloor$ wird ohne Projektion generiert

Durchschnittskonstruktor $\cap x:S.T[x]$

- **Durchschnitt einer Familie von Typen $T[x]$ mit $x \in S$**
  - Verallgemeinerung des einfachen Durchschnitts $S \cap T$ zweier Typen
  - Elemente sind Objekte, die für alle $x \in S$ zum Typ $T[x]$ gehören
  - Keine Abhängigkeit der Elemente von der Wahl eines konkreten $x \in S$
- **Strukturell ähnlich zu $x{:}S \rightarrow T[x]$** (aber andere Semantik)
  - Elemente von $x{:}S \rightarrow T[x]$ konstruieren für alle $x \in S$ ein $y \in T[x]$
  - $x \in S$ als Argument der Funktionen aus $x{:}S \rightarrow T[x]$ angegeben werden
  - Starke Abhängigkeit von $y \in T[x]$ von der Wahl des konkreten $x \in S$
- **Viele Anwendungen**
  - Record-Strukturen in Programmiersprachen
  - Definition eines Typs aller Terme der Typentheorie
  - Repräsentation eines echten Polymorphismus (ohne Abhängigkeiten)
  - Polymorphe Logik
  - “Guarded” Types

Formulierung des Durchschnittstyps

- **Ausdrücke**

  Kanonisch: $\cap x{:}S.T$ x Variable, S und T Terme

  Nichtkanonisch: —

- **Reduktion von Ausdrücken**

  – entfällt, da keine kanonischen Elemente für den Teilmengenkonstruktor definiert

- **Urteile für Typ- und Elementgleichheit**

  $\cap x_1{:}S_1.T_1 = \cap x_2{:}S_2.T_2$ falls $S_1{=}S_2$ und $T_1[s_1/x_1]{=}T_2[s_2/x_2]$
  für alle Terme s_1, s_2 mit $s_1{=}s_2 \in S_1$.

  $\cap x_1{:}S_1.T_1 = \cap x_2{:}S_2.T_2 \in \mathbb{U}_j$ analog

  $t_1 = t_2 \in \cap x{:}S.T$ falls $\cap x{:}S.T$ *Typ* und
  $t_1{=}t_2 \in T[s/x]$ für alle Terme $s \in S$

Regeln für Durchschnittsbildung

$\Gamma \vdash \cap x_1{:}S_1.T_1 = \cap x_2{:}S_2.T_2 \in \mathbb{U}_j$ [Ax]
`isectEquality` x
$\Gamma \vdash S_1 = S_2 \in \mathbb{U}_j$ [Ax]
$\Gamma, x{:}S_1 \vdash T_1[x/x_1] = T_2[x/x_2] \in \mathbb{U}_j$ [Ax]

$\Gamma \vdash \mathbb{U}_j$ [ext $\cap x : S . T$]
`isectFormation` x S
$\Gamma \vdash S \in \mathbb{U}_j$ [Ax]
$\Gamma, x{:}S \vdash \mathbb{U}_j$ [ext T]

$\Gamma \vdash t_1 = t_2 \in \cap x{:}S.T$ [Ax]
`isect_memberEquality` j x'
$\Gamma, x'{:}S \vdash t_1 = t_2 \in T[x'/x]$ [Ax]
$\Gamma \vdash S \in \mathbb{U}_j$ [Ax]

$\Gamma \vdash \cap x{:}S.T$ [ext t]
`isect_memberFormation` j x'
$\Gamma, x'{:}S \vdash T[x'/x]$ [ext t]
$\Gamma \vdash S \in \mathbb{U}_j$ [Ax]

$\Gamma \vdash f_1 = f_2 \in T[t/x]$ [Ax]
`isect_member_caseEquality` $\cap x : S . T$ t
$\Gamma \vdash f_1 = f_2 \in \cap x{:}S.T$ [Ax]
$\Gamma \vdash t \in S$ [Ax]

$\Gamma, f{:}\cap x{:}S.T, \Delta \vdash C$ [ext $t[f, \mathsf{Ax}/y, z]$]
`isectElimination` i s y z
$\Gamma, f{:}\cap x{:}S.T, \Delta \vdash s \in S$ [Ax]
$\Gamma, f{:}\cap x{:}S.T, y{:}T[s/x], z{:}\, y{=}f \in T[s/x], \Delta \vdash C$ [ext t]

Anwendungen des Durchschnittstyps

- **Einfacher (binärer) Durchschnitt $S \cap T$**
  - Durchschnitt einer Mengenfamilie mit booleschem Index

    $S \cap T \equiv \cap \mathsf{b}{:}\mathbb{B}.\ \mathsf{if}\ \mathsf{b}\ \mathsf{then}\ S\ \mathsf{else}\ T$

  Achtung: $S \cap T$ ist nicht identisch mit $x{:}S \cap T$ mit T unabhängig von x

- **$\mathtt{Top}$: Typ aller Terme der Typentheorie**
  - Leerer Durchschnitt hinterläßt keine Bedingungen an Elemente

    $\mathtt{Top} \equiv \cap \mathtt{x}{:}\mathsf{Void}.\mathsf{Void}$

  - Wegen $t \in \mathtt{Top}$, falls $t \in \mathsf{Void}$ für alle $x \in \mathsf{Void}$ gehört jeder Term t zu $\mathtt{Top}$

- **Record Strukturen $\{l_1{:}T_1; ..; l_n{:}T_n\}$**
  - Durchschnitt einer Familie von Funktionen auf Labels l_i (Folie 26)
  - $\{l_1{:}T_1; ..; l_n{:}T_n\} \equiv \cap \mathsf{l}{:}\mathsf{Atom}.\ \mathsf{if}\ \mathtt{l}{=}l_1\ \mathsf{then}\ T_1\ \mathsf{else}$

    $\vdots \qquad \vdots$

    $\mathsf{if}\ \mathtt{l}{=}l_n\ \mathsf{then}\ T_n\ \mathsf{else}\ \mathtt{Top}$

- **Guarded types $\cap x{:}S.T$, wobei T unabhängig von x**
  - Isomorph zu $\mathtt{Top}$, wenn S leer ist, und sonst isomorph zu T
  - Nützlich für Wohlgeformtheitsbeweise zu Mengenaussagen, die in pathologischen Fällen nicht einmal Typen wären (z.B. $x \in S \Rightarrow x \in T$)

Logische Interpretation des Durchschnittstyps

- **$\cap x{:}S.P[x]$ verhält sich wie ein Allquantor**
  - $\cap x{:}S.P[x]$ ist genau dann nicht leer, wenn $\forall x{:}S.P[x]$ gültig ist
  - Elemente $p \in \cap x{:}S.P[x]$ sind uniforme Evidenz für alle $P[s]$
  - Anders als bei $\forall x{:}S.P[x]$ wird Evidenz für $P[s]$ nicht aus s konstruiert
    algorithmischer Aspekt des Allquantors entfällt
  - Definition $\forall[x{:}S].P[x] \equiv \cap x{:}S.P[x]$ liefert polymorphe Logik
    Viele Quantoren in Theoremen können polymorph gewählt werden
    z.B. $\forall[\mathtt{i}{:}\mathbb{Z}].\ \forall[\mathtt{j}{:}\mathbb{Z}].\ \mathtt{i+j=j+i} \in \mathbb{Z}$
- **Polymorphe Typdeklarationen sind möglich**
  - z.B. besagt $f \in \cap T{:}\mathbb{U}.\ T \to T$, daß f auf beliebigen Typen operiert
  - Typparameter T muß in Deklaration von f nicht verwendet werden
- **$P \cap Q$ verhält sich wie Konjunktion**
  - $P \cap Q$ ist genau dann nicht leer, wenn $P \wedge Q$ gültig ist
  - Elemente $p \in P \cap Q$ sind uniforme Evidenz für alle P und Q
  - Keine Paarbildung erforderlich

Vereinigungskonstruktor $\cup x:S.T[x]$

- **Vereinigungs einer Familie von Typen $T[x]$ mit $x \in S$**
  - Verallgemeinerung der einfachen Vereinigung $S \cap T$ zweier Typen
  - Elemente sind Objekte y , die für ein $x \in S$ zum Typ $T[x]$ gehören
  - Keine Angabe eines konkreten $x \in S$ erforderlich
- **Strukturell ähnlich zu $x{:}S \times T[x]$** (aber andere Semantik)
  - Elemente von $x{:}S \times T[x]$ sind Paare (x, y) mit $x \in S$ und ein $y \in T[x]$
  - Starke Abhängigkeit von $y \in T[x]$ von der Wahl des konkreten $x \in S$
  - $x \in S$ muß als Teil des Elements mit angegeben werden
- **Nichttriviale Formalisierung**
  - Bei nicht disjunkten Typen muß Urteil der Elementgleichheit im Überlappingsbereich auf allen Typen übereinstimmen

Derzeit nicht in Nuprl eingebettet

Logische Interpretation des Vereinigungstyps

- **$\cup x{:}S.P[x]$ verhält sich wie ein Existensquantor**
  - $\cup x{:}S.P[x]$ ist genau dann nicht leer, wenn $\exists x{:}S.P[x]$ gültig ist
  - Elemente $p \in \cup x{:}S.P[x]$ sind Evidenz für ein $P[s]$
  - Anders als bei $\exists x{:}S.P[x]$ fehlt der Zeuge $s \in S$ für den $P[s]$ gilt
    algorithmischer Aspekt des Existenzquantors entfällt
  - $\exists[x{:}S].P[x] \equiv \cup x{:}S.P[x]$ liefert semi-klassischen Existenzquantor
- **$P \cup Q$ verhält sich wie Disjunktion**
  - $P \cup Q$ ist genau dann nicht leer, wenn $P \vee Q$ gültig ist
  - Elemente $p \in P \cup Q$ sind Evidenz für P oder für Q
  - Keine Kennzeichnung, welche der beiden Aussagen bewiesen wurde
  - $P[\vee]Q \equiv P \cup Q$ liefert semi-klassische Disjunktion

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: (rationale) Cauchyfolgen mit gleichem Grenzwert
  - Restklassenräume: $\mathbb{Z}$ `mod` k
- **Entspricht 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

- **Ausdrücke**

  Kanonisch: $x, y : T//E$ E, T Terme, x, y Variablen

  Nichtkanonisch: —

- **Reduktion von Ausdrücken**

  – entfällt, da keine kanonischen Elemente für Quotiententypen definiert

- **Urteile für Typ- und Elementgleichheit**

$x_1, y_1 : T_1//E_1$
$= x_2, y_2 : T_2//E_2$ falls $T_1 = T_2$ 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 $t \in \forall x{:}T_1 . \forall y{:}T_1 . \forall z{:}T_1 .$

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

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

Behandlung von Quotienten im Inferenzsystem

- **Gleichheit $E[s,t\,/\,x,y]$ ist implizites Wissen**
  - Wir wissen $E[s,t/x,y]$ wenn $s = t \in x,y : T//E$
  - In Beweisen müssen wir das Wissen $E[s,t/x,y]$ verwenden können, ohne die Evidenz für $E[s,t/x,y]$ algorithmisch einzusetzen

  Evidenz für $E[s,t/x,y]$ darf nicht im Extraktterm vorkommen

- **Versteckte Hypothesen erforderlich**
  - Dekomposition einer Quotientengleichheit muß versteckte Hypothesen generieren

$$\Gamma, v{:}\, s = t \in x,y : T//E, \Delta \vdash C \qquad [\text{ext } u]$$
$$\texttt{quotient\_equalityElimination}\ i\ j\ v'$$
$$\Gamma, v{:}\, s = t \in x,y : T//E, [\![v']\!]{:}E[s,t/x,y], \Delta \vdash C \qquad [\text{ext } u]$$
$$\Gamma, v{:}\, s = t \in x,y : T//E, \Delta \vdash E[s,t/x,y] \in \mathbb{U}_j \qquad [\text{Ax}]$$

  - Freigabe versteckter Hypothesen wie zuvor

Regeln für Quotiententypen

$\Gamma \vdash \mathbb{U}_j$ [ext $x, y : T//E$]

quotientFormation $T\ E\ x\ y\ z\ v\ v'$

$\Gamma \vdash T \in \mathbb{U}_j$ [Ax]
$\Gamma, x{:}T, y{:}T \vdash E \in \mathbb{U}_j$ [Ax]
$\Gamma, x{:}T, \vdash E[x, x/x, y]$ [Ax]
$\Gamma, x{:}T, y{:}T, v{:}\, E[x, y/x, y] \vdash E[y, x/x, y]$ [Ax]
$\Gamma, x{:}T, y{:}T, z{:}T, v{:}\, E[x, y/x, y], v'{:}\, E[y, z/x, y] \vdash E[x, z/x, y]$ [Ax]

$\Gamma \vdash x_1, y_1 : T_1//E_1 = x_2, y_2 : T_2//E_2 \in \mathbb{U}_j$ [Ax]

quotientWeakEquality $x\ y\ z\ v\ v'$

$\Gamma \vdash T_1 = T_2 \in \mathbb{U}_j$ [Ax]
$\Gamma, x{:}T_1, y{:}T_1 \vdash E_1[x, y/x_1, y_1] = E_2[x, y/x_2, y_2] \in \mathbb{U}_j$ [Ax]
$\Gamma, x{:}T_1 \vdash E_1[x, x/x_1, y_1]$ [Ax]
$\Gamma, x{:}T_1, y{:}T_1, v{:}\, E_1[x, y/x_1, y_1] \vdash E_1[y, x/x_1, y_1]$ [Ax]
$\Gamma, x{:}T_1, y{:}T_1, z{:}T_1, v{:}\, E_1[x, y/x_1, y_1], v'{:}\, E_1[y, z/x_1, y_1] \vdash E_1[x, z/x_1, y_1]$ [Ax]

$\Gamma \vdash x_1, y_1 : T_1//E_1 = x_2, y_2 : T_2//E_2 \in \mathbb{U}_j$ [Ax]

quotientEquality

$\Gamma \vdash x_1, y_1 : T_1//E_1 \in \mathbb{U}_j$ [Ax]
$\Gamma \vdash x_2, y_2 : T_2//E_2 \in \mathbb{U}_j$ [Ax]
$\Gamma \vdash T_1 = T_2 \in \mathbb{U}_j$ [Ax]
$\Gamma, v{:}\, T_1{=}T_2{\in}\mathbb{U}_j, x{:}T_1, y{:}T_1 \vdash E_1[x, y/x_1, y_1] \Rightarrow E_2[x, y/x_2, y_2]$ [Ax]
$\Gamma, v{:}\, T_1{=}T_2{\in}\mathbb{U}_j, x{:}T_1, y{:}T_1 \vdash E_2[x, y/x_2, y_2] \Rightarrow E_1[x, y/x_1, y_1]$ [Ax]

$\Gamma \vdash s = t \in x, y : T//E$ [Ax]

quotient_memberWeakEquality j

$\Gamma \vdash x, y : T//E \in \mathbb{U}_j$ [Ax]
$\Gamma \vdash s = t \in T$ [Ax]

$\Gamma \vdash x, y : T//E$ [ext t]

quotient_memberFormation j

$\Gamma \vdash x, y : T//E \in \mathbb{U}_j$ [Ax]
$\Gamma \vdash T$ [ext t]

Regeln für Quotiententypen II

$\Gamma \vdash s = t \in x, y : T//E$ [Ax]
`quotient_memberEquality` j
$\Gamma \vdash x, y : T//E \in \mathbb{U}_j$ [Ax]
$\Gamma \vdash s \in T$ [Ax]
$\Gamma \vdash t \in T$ [Ax]
$\Gamma \vdash E[s, t/x, y]$ [Ax]

$\Gamma, v{:}\, s{=}t \in x, y : T//E, \Delta \vdash C$ [ext u]
`quotient_equalityElimination` i j v'
$\Gamma, v{:}\, s{=}t \in x, y : T//E, [\![v']\!]{:}E[s, t/x, y], \Delta \vdash C$ [ext u]
$\Gamma, v{:}\, s = t \in x, y : T//E, \Delta \vdash E[s, t/x, y] \in \mathbb{U}_j$ [Ax]

$\Gamma, z{:}\, x, y : T//E, \Delta \vdash s = t \in S$ [Ax]
`quotientElimination` i j x' y' v
$\Gamma, z{:}\, x, y : T//E, \Delta, x'{:}T, y'{:}T \vdash E[x', y'/x, y] \in \mathbb{U}_j$ [Ax]
$\Gamma, z{:}\, x, y : T//E, \Delta \vdash S \in \mathbb{U}_j$ [Ax]
$\Gamma, z{:}\, x, y : T//E, \Delta, x'{:}T, y'{:}T, v{:}\, E[x', y'/x, y] \vdash s[x'/z] = t[y'/z] \in S[x'/z]$ [Ax]

$\Gamma, z{:}\, x, y : T//E, \Delta \vdash s = t \in S$ [Ax]
`quotientElimination_2` i j x' y' v
$\Gamma, z{:}\, x, y : T//E, \Delta, x'{:}T, y'{:}T \vdash E[x', y'/x, y] \in \mathbb{U}_j$ [Ax]
$\Gamma, z{:}\, x, y : T//E, \Delta \vdash S \in \mathbb{U}_j$ [Ax]
$\Gamma, z{:}\, x, y : T//E, x'{:}T, y'{:}T, v{:}\, E[x', y'/x, y], \Delta[x'/z] \vdash s[x'/z] = t[y'/z] \in S[x'/z]$ [Ax]

Wichtige benutzerdefinierte Quotiententypen

- **Rationale Zahlen**

$x_1 =_q x_2 \equiv$ match x_1 with $\langle z_1, n_1\rangle \mapsto$ match x_2 with $\langle z_2, n_2\rangle \mapsto z_1 * n_2 = z_2 * n_1$

$\mathbb{Q} \equiv \mathtt{x,y} : \mathbb{Z}\times\mathbb{N}^+ /\!/ \mathtt{x} =_q \mathtt{y}$

$x_1 + x_2 \equiv$ match x_1 with $\langle z_1, n_1\rangle \mapsto$ match x_2 with $\langle z_2, n_2\rangle \mapsto \langle z_1 * n_2 + z_2 * n_1, n_1 * n_2\rangle$

$x_1 - x_2 \equiv$ match x_1 with $\langle z_1, n_1\rangle \mapsto$ match x_2 with $\langle z_2, n_2\rangle \mapsto \langle z_1 * n_2 - z_2 * n_1, n_1 * n_2\rangle$

$x_1 * x_2 \equiv$ match x_1 with $\langle z_1, n_1\rangle \mapsto$ match x_2 with $\langle z_2, n_2\rangle \mapsto \langle z_1 * z_2, n_1 * n_2\rangle$

$x_1 <_q x_2 \equiv$ match x_1 with $\langle z_1, n_1\rangle \mapsto$ match x_2 with $\langle z_2, n_2\rangle \mapsto z_1 * n_2 < z_2 * n_1$

- **Restklassenräume**

$x_1 = x_2 \ \mathtt{mod}\ k \equiv k\ \mathtt{divides}\ x_1 - x_2$

$\mathbb{Z}\ \mathtt{mod}\ k \equiv \mathtt{x,y} : \mathbb{Z} /\!/ \mathtt{x=y\ mod}\ k$

Operationen auf $\mathbb{Z}$ mod k können direkt von $\mathbb{Z}$ übernommen werden

Behandlung überstrukturierter Prädikate

- **Manche Prädikate auf $\mathbb{Q}$ enthalten zu viel Struktur**
  - $x_1 <_q x_2$ muß im Quotiententyp (!) 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 $x_1 = \langle 2, 1 \rangle$, $x_1' = \langle 4, 2 \rangle$, $x_2 = x_2' = \langle 3, 1 \rangle$
  - Unabhängigkeit vom Repräsentanten nicht mehr gegeben
  - Definition von $<_q$ enthält zu viel Struktur, wo nur "Wahrheit" nötig ist
- **Type-Squashing erforderlich** (neue Definition von $<_q$)

$$\boldsymbol{x_1 <_q x_2} \equiv \downarrow \mathsf{match}\ x_1\ \mathsf{with}\ \langle z_1, n_1 \rangle \mapsto \mathsf{match}\ x_2\ \mathsf{with}\ \langle z_2, n_2 \rangle \mapsto z_1 * n_2 < z_2 * n_1$$

Definition Reeller Zahlen mit Quotiententypen

$$
\begin{array}{lll}
\boldsymbol{x_1 \leq_q x_2} & \equiv & x_1 < x_2 \;\; \vee \;\; x_1 = x_2 \in \mathbb{Q} \\
\boldsymbol{z/n} & \equiv & \langle z\,,n\rangle \\
\boldsymbol{|x|} & \equiv & \text{match } x \text{ with } \langle z\,,n\rangle \mapsto \text{if } z<0 \text{ then } \langle -z\,,n\rangle \text{ else } \langle z\,,n\rangle \\
\mathbb{R}_{pre} & \equiv & \{\mathtt{f}:\mathbb{N}^+\to\mathbb{Q} \mid \forall \mathtt{m},\mathtt{n}:\mathbb{N}^+.\;\; |\mathtt{f}(\mathtt{n})-\mathtt{f}(\mathtt{m})| \;\leq\; 1/\mathtt{m}+1/\mathtt{n}\} \\
\boldsymbol{x_1 =_r x_2} & \equiv & \forall \mathtt{n}:\mathbb{N}^+.\;\; |x_1(\mathtt{n}) - x_2(\mathtt{n})| \;\leq\; 2/\mathtt{n} \\
\mathbb{R} & \equiv & \mathtt{x},\mathtt{y} : \mathbb{R}_{pre}//\mathtt{x}=_r\mathtt{y} \\
\boldsymbol{x_1+x_2} & \equiv & \lambda \mathtt{n}.\; x_1(\mathtt{n})+x_2(\mathtt{n}) \\
\boldsymbol{x_1-x_2} & \equiv & \lambda \mathtt{n}.\; x_1(\mathtt{n})-x_2(\mathtt{n}) \\
\boldsymbol{|x|} & \equiv & \lambda \mathtt{n}.\; |x(\mathtt{n})|
\end{array}
$$

Elegante Beweise erfordern viele Spezialtaktiken

Atomare Bezeichner

- **Verwaltung eindeutiger Namen**
  - Token: Textketten, die (anders als Strings) keine Struktur haben
  - Einzig mögliche Analyse ist Test auf Gleichheit von Token
  - Wichtig für Programmiersprachen und Sicherheitsprotokolle

---

- **Ausdrücke**

Kanonisch:	Atom	
	"*token*"	*token* Text-Token
Nichtkanonisch:	if a=b then s else t	a, b, s, t Terme

- **Reduktion von Ausdrücken**

if a=b then s else t $\xrightarrow{\beta}$ s, falls $a = b$, sonst t

- **Urteile für Typ- und Elementgleichheit**

Atom = Atom

"*token*" = "*token*" $\in$ Atom — *für alle token !* —

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

Regeln für Atomare Bezeichner

$\Gamma \vdash \mathsf{Atom} = \mathsf{Atom} \in \mathbb{U}_j$ [Ax]
`atomEquality`

$\Gamma \vdash \mathbb{U}_j$ [ext Atom]
`atomFormation`

$\Gamma \vdash$ "*token*" = "*token*" $\in$ Atom [Ax]
`tokenEquality`

$\Gamma \vdash$ Atom [ext "*token*"]
`tokenFormation` "*token*"

$\Gamma \vdash \mathsf{if}\ u_1{=}v_1\ \mathsf{then}\ s_1\ \mathsf{else}\ t_1 = \mathsf{if}\ u_2{=}v_2\ \mathsf{then}\ s_2\ \mathsf{else}\ t_2 \in T$ [Ax]
`atom_eqEquality` v
$\Gamma \vdash u_1 = u_2 \in \mathsf{Atom}$ [Ax]
$\Gamma \vdash v_1 = v_2 \in \mathsf{Atom}$ [Ax]
$\Gamma, v\colon u_1{=}v_1 \in \mathsf{Atom} \vdash s_1 = s_2 \in T$ [Ax]
$\Gamma, v\colon \neg(u_1{=}v_1 \in \mathsf{Atom}) \vdash t_1 = t_2 \in T$ [Ax]

$\Gamma \vdash \mathsf{if}\ u{=}v\ \mathsf{then}\ s\ \mathsf{else}\ t = t_2 \in T$ [Ax]
`atom_eqReduceTrue`
$\Gamma \vdash s = t_2 \in T$ [Ax]
$\Gamma \vdash u = v \in \mathsf{Atom}$ [Ax]

$\Gamma \vdash \mathsf{if}\ u{=}v\ \mathsf{then}\ s\ \mathsf{else}\ t = t_2 \in T$ [Ax]
`atom_eqReduceFalse`
$\Gamma \vdash t = t_2 \in T$ [Ax]
$\Gamma \vdash \neg(u = v \in \mathsf{Atom})$ [Ax]

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

Induktive Datentypen

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 $f(e)$ where $f(x) = t$ 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: $f(e)\ \mathsf{where}\ f(x) = t$ — e, t Terme, f, x Variablen

- **Reduktion von Ausdrücken**

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

  Terminierung von $f(e)\ \mathsf{where}\ f(x) = t$ *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 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 $\mathtt{X} = \mathtt{X} \to \mathbb{Z}$ müssen ausgeschlossen werden
    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 $\vdash$ 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

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

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

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

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

$\Gamma, z{:}\text{rectype } X = T_X, \Delta \vdash C$ [ext $f(z)$ where $f(x) = t[\lambda y.\mathsf{f}/P]$]
`recElimination` i j P y f x
$\Gamma, z{:}\text{rectype } X = T_X, \Delta \vdash \text{rectype } X = T_X \in \mathbb{U}_j$ [Ax]
$\Gamma, z{:}\text{rectype } X = T_X, \Delta, P{:}(\text{rectype } X = T_X) \to \mathbb{P}_j, f{:}(y{:}\{x : \text{rectype } X = T_X \mid P(x)\} \to C[y/z]),$
$x{:}T_X[\{x : \text{rectype } X = T_X \mid P(x)\}/X] \vdash C[x/z]$ [ext t]

$\Gamma, z{:}\text{rectype } X = T_X, \Delta \vdash C$ [ext $t[z/x]$]
`recUnrollElimination` i x v
$\Gamma, z{:}\text{rectype } X = T_X, \Delta, x{:}T_X[\text{rectype } X = T_X/X], v{:}z{=}x \in T_X[\text{rectype } X = T_X/X] \vdash C[x/z]$ [ext t]

Rekursive Funktionen

- **Definition von Funktionen durch $f(x) = t[f, x]$**
  - z.B. λf . $\min_f(0)$ where $\min_f(y) =$ if f(y)=0 then y else $\min_f$ (y+1)
    λx . sqr(0) where sqr(y) = if x<$(y+1)^2$ then y else sqr (y+1)
  - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu $f(e)$ where $f(x) = t$ 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. `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 $f(e)$ where$^\infty$ $f(x) = t$

- **Parametrisierte simultane Rekursion möglich**

Kein fester Bestandteil der Nuprl Typentheorie

Neuere Typkonstrukte

- **Abhängiger Durchschnitt $x : S \cap T[x]$** Bisher nur in MetaPRL
  – Element s muß zu S und gleichzeitig zu $T[s]$ gehören (Selbstreferenz!)
- **Squiggle-Equality $s \sim t$**
  – Einfacherer, syntaktischer Gleichheitstyp, ohne Abhängigkeit vom Typ
  $s \sim t$ 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 $\{f \mid x : S \rightarrow T[f, x]\}$**
  – 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

Typentheorie im Rückblick

Inferenzkalkül für Mathematik & Programmierung

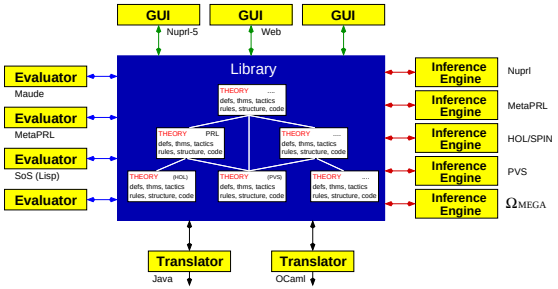
- **Ausdrucksstarkes Inferenzsystem**
  - Vereinheitlicht und erweitert Logik, λ-Kalkül und einfache Typentheorie
  - Formalisierung "natürlicher" Gesetze der zentralen Konzepte
  - Direkte Darstellung anstatt Simulation
  - Umfangreiche Theorie bestehend aus
    - Prädikatenlogik (höherer Stufe)
  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

    $\rightarrow$ *Automatisierung der Beweisführung*

Ausblick: Themen des zweiten Teils (Sommer 2014)

Konstruiere semiautomatische Beweissysteme

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



- **Automatisierung des formalen Schließens**
  - Taktisches Beweisen
  - Entscheidungsprozeduren
  - Integration externer Systeme in den Inferenzmechanismus

- **Anwendungen & Demonstrationen**
  - Formalisierung mathematischen Wissens
  - Synthese effizienter Algorithmen aus formalen Spezifikationen

⋮