

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**⁺, ...)

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

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 $x : S \times P[x]$**

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 $s \in \{x:S \mid P[x]\}$

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

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

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

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

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

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

Nichtkanonisch: —

TEILMENGEN, FORMAL

Syntax:

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

Nichtkanonisch: —

Auswertung: —

TEILMENGEN, FORMAL

Syntax:

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

Nichtkanonisch: —

Auswertung: —

Semantik:

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

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

$$\text{und } p_2 \in \forall x:S_1. T_2[x/x_2] \Rightarrow T_1[x/x_1]$$

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

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

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

TEILMENGEN, FORMAL

Syntax:

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

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

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

$$\text{und } p_2 \in \forall x:S_1. T_2[x/x_2] \Rightarrow T_1[x/x_1]$$

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

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

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

TEILMENGEN, FORMAL

Syntax:

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

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

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

$$\text{und } p_2 \in \forall x:S_1. T_2[x/x_2] \Rightarrow T_1[x/x_1]$$

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

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

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

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 $x:T \times P$

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 $\mathbb{Z} \rightarrow \mathbb{Z}$, $f:\mathbb{Z} \rightarrow \mathbb{Z} \times (\exists y:\mathbb{Z}. f(y)=0)$ und $\{f:\mathbb{Z} \rightarrow \mathbb{Z} \mid \exists y:\mathbb{Z}. f(y)=0\}$

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 $\mathbb{Z} \rightarrow \mathbb{Z}$, $f:\mathbb{Z} \rightarrow \mathbb{Z} \times (\exists y:\mathbb{Z}. f(y)=0)$ und $\{f:\mathbb{Z} \rightarrow \mathbb{Z} \mid \exists y:\mathbb{Z}. f(y)=0\}$
- 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

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 $\mathbb{Z} \rightarrow \mathbb{Z}$, $f:\mathbb{Z} \rightarrow \mathbb{Z} \times (\exists y:\mathbb{Z}. f(y)=0)$ und $\{f:\mathbb{Z} \rightarrow \mathbb{Z} \mid \exists y:\mathbb{Z}. f(y)=0\}$
- 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

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 $\mathbb{Z} \rightarrow \mathbb{Z}$, $f:\mathbb{Z} \rightarrow \mathbb{Z} \times (\exists y:\mathbb{Z}. f(y)=0)$ und $\{f:\mathbb{Z} \rightarrow \mathbb{Z} \mid \exists y:\mathbb{Z}. f(y)=0\}$
- 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

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 $\mathbb{Z} \rightarrow \mathbb{Z}$, $f:\mathbb{Z} \rightarrow \mathbb{Z} \times (\exists y:\mathbb{Z}. f(y)=0)$ und $\{f:\mathbb{Z} \rightarrow \mathbb{Z} \mid \exists y:\mathbb{Z}. f(y)=0\}$
- 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**

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 $\mathbb{Z} \rightarrow \mathbb{Z}$, $f:\mathbb{Z} \rightarrow \mathbb{Z} \times (\exists y:\mathbb{Z}. f(y)=0)$ und $\{f:\mathbb{Z} \rightarrow \mathbb{Z} \mid \exists y:\mathbb{Z}. f(y)=0\}$
- 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 $z:\{x:S \mid T\}$

$$\Gamma, z:\{x:S \mid T\}, \Delta \vdash C \text{ [ext } (\lambda y.t) z]$$

BY setElimination $i \ y \ v$

$$\Gamma, z:\{x:S \mid T\}, y:S, \llbracket v \rrbracket:T[y/x], \Delta[y/z] \vdash C[y/z] \text{ [ext } t]$$

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 $\mathbb{Z} \rightarrow \mathbb{Z}$, $f:\mathbb{Z} \rightarrow \mathbb{Z} \times (\exists y:\mathbb{Z}. f(y)=0)$ und $\{f:\mathbb{Z} \rightarrow \mathbb{Z} \mid \exists y:\mathbb{Z}. f(y)=0\}$
- 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 $z:\{x:S \mid T\}$

$$\Gamma, z:\{x:S \mid T\}, \Delta \vdash C \text{ [ext } (\lambda y.t) z]$$

BY setElimination $i \ y \ v$

$$\Gamma, z:\{x:S \mid T\}, y:S, \llbracket v \rrbracket:T[y/x], \Delta[y/z] \vdash C[y/z] \text{ [ext } t]$$

- Freigabe in Teilzielen mit Extraktterm **Ax** (Gleichheiten, Kleiner-Relation)

WICHTIGE BENUTZERDEFINIERTER MENGENTYPEN

● Zahlenmengen und -intervalle:

Theory `int_1`

$\mathbb{N}$	$\equiv \{i:\mathbb{Z} \mid 0 \leq i\}$	<code>nat</code>
$\mathbb{N}^+$	$\equiv \{i:\mathbb{Z} \mid 0 < i\}$	<code>nat_plus</code>
$\{i \dots\}$	$\equiv \{j:\mathbb{Z} \mid i \leq j\}$	<code>int_upper</code>
$\{\dots i\}$	$\equiv \{j:\mathbb{Z} \mid j \leq i\}$	<code>int_lower</code>
$\{i \dots j\}$	$\equiv \{k:\mathbb{Z} \mid i \leq k \leq j\}$	<code>int_iseg</code>
$\{i \dots j^-\}$	$\equiv \{k:\mathbb{Z} \mid i \leq k < j\}$	<code>int_seg</code>

WICHTIGE BENUTZERDEFINIERTER MENGENTYPEN

● Zahlenmengen und -intervalle:

Theory `int_1`

$\mathbb{N}$	$\equiv \{i:\mathbb{Z} \mid 0 \leq i\}$	<code>nat</code>
$\mathbb{N}^+$	$\equiv \{i:\mathbb{Z} \mid 0 < i\}$	<code>nat_plus</code>
$\{i \dots\}$	$\equiv \{j:\mathbb{Z} \mid i \leq j\}$	<code>int_upper</code>
$\{\dots i\}$	$\equiv \{j:\mathbb{Z} \mid j \leq i\}$	<code>int_lower</code>
$\{i \dots j\}$	$\equiv \{k:\mathbb{Z} \mid i \leq k \leq j\}$	<code>int_iseg</code>
$\{i \dots j^-\}$	$\equiv \{k:\mathbb{Z} \mid i \leq k < j\}$	<code>int_seg</code>

● Listen

$$T \text{ list}^+ \equiv \{l:T \text{ list} \mid 0 < ||l||\}$$

`listp`

- **Modifikation der Gleichheit auf Typen**

- **Modifikation der Gleichheit auf Typen**

- Rationale Zahlen: Paare ganzer Zahlen mit $\langle z_1, n_1 \rangle = \langle z_2, n_2 \rangle$,
falls $z_1 * n_2 = z_2 * n_1$

● 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

● 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: $\mathbb{Z} \bmod k$

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: $\mathbb{Z} \bmod k$

- **Faktorisierung modulo einer Äquivalenz**

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: $\mathbb{Z} \bmod k$

- **Faktorisierung modulo einer Äquivalenz**

- Elemente s, t werden aus Typ T ausgewählt
- Gleichheit von s und t wird über Äquivalenzrelation E neu definiert

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: $\mathbb{Z} \bmod k$

● 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

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: $\mathbb{Z} \bmod k$

● 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

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: $\mathbb{Z} \bmod k$

● 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



Quotiententypen wichtig für formale Mathematik

QUOTIENTENTYPEN, FORMAL

Syntax:

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

Nichtkanonisch: —

QUOTIENTENTYPEN, FORMAL

Syntax:

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

Nichtkanonisch: —

Auswertung: —

QUOTIENTENTYPEN, FORMAL

Syntax:

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

Nichtkanonisch: —

Auswertung: —

Semantik:

$$\begin{aligned} & x_1, y_1 : T_1 // E_1 \\ & = x_2, y_2 : T_2 // E_2 \end{aligned}$$

$\equiv$ $T_1 = T_2$ 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

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

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

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

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

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

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

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

QUOTIENTENTYPEN, FORMAL

Syntax:

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

Nichtkanonisch: —

Auswertung: —

Semantik:

$x_1, y_1 : T_1 // E_1$
 $= x_2, y_2 : T_2 // E_2$ $\equiv$ $T_1 = T_2$ 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

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

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

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

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

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

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

$s = t \in x, y : T // E$ $\equiv$ $x, y : T // E$ 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 BENUTZERDEFINIERTER QUOTIENTENTYPEN

● Rationale Zahlen

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

WICHTIGE BENUTZERDEFINIERTER QUOTIENTENTYPEN

● Rationale Zahlen

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

● Restklassenräume

$x_1 = x_2 \bmod k$	$\equiv k \text{ divides } x_1 - x_2$	eqmod
$\mathbb{Z} \bmod k$	$\equiv x, y : \mathbb{Z} // x = y \bmod k$	int_mod

QUOTIENTENTYPEN: EINFLUSS AUF DAS INFERENZSYSTEM

- Gleichheit $E[s, t / x, y]$ ist implizites Wissen

QUOTIENTENTYPEN: EINFLUSS AUF DAS INFERENZSYSTEM

- Gleichheit $E[s, t / x, y]$ ist implizites Wissen
 - Wir wissen $E[s, t / x, y]$ wenn $s = t \in x, y : T // E$

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

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

$$\Gamma, v : s = t \in x, y : T // E, \Delta \vdash C \text{ [ext } u]$$

BY quotient_equalityElimination $i \ j \ v'$

$$\Gamma, v : s = t \in x, y : T // E, \llbracket v' \rrbracket : E[s, t / x, y], \Delta \vdash C \text{ [ext } u]$$

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

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

$$\Gamma, v : s = t \in x, y : T // E, \Delta \vdash C \text{ [ext } u]$$

BY quotient_equalityElimination i j v'

$$\Gamma, v : s = t \in x, y : T // E, \llbracket v' \rrbracket : E[s, t / x, y], \Delta \vdash C \text{ [ext } u]$$

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

- Freigabe versteckter Hypothesen wie zuvor

- **Überstrukturierung** auf $x, y : T // E$ möglich
 - $x_1 <_q x_2$ muß wohlgeformt sein

- **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 $x_2 = x'_2 \in \mathbb{Q}$?

- 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 $z_2 * n_1 = z'_2 * n'_1 \in \mathbb{Z}$

- **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 $x_1 = \langle 2, 1 \rangle$, $x'_1 = \langle 4, 2 \rangle$, $x_2 = x'_2 = \langle 3, 1 \rangle$

● 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

- **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**

- $\downarrow P \equiv \{0 \in \mathbb{Z} \mid P\}$

- **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

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

Elegante Beweise erfordern viele Spezialtaktiken

REKURSIVE DEFINITION

- **Größere Freiheit in der Formulierung**
 - Zahlen unterstützen induktive Beweise und primitive Rekursion

REKURSIVE DEFINITION

- **Größere Freiheit in der Formulierung**

- Zahlen unterstützen induktive Beweise und primitive Rekursion
- Unnatürliche Simulation rekursiver Datentypen (Bäume, Graphen,..)

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

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

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

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

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

Repräsentation rekursiv definierter Strukturen

- Rekursive Typdefinition mit Gleichung $X = T[X]$

Repräsentation rekursiv definierter Strukturen

- Rekursive Typdefinition mit Gleichung $X = T[X]$
 - z.B. `rectype bintree =  $\mathbb{Z} + \mathbb{Z} \times \text{bintree} \times \text{bintree}$`

Repräsentation rekursiv definierter Strukturen

- **Rekursive Typdefinition mit Gleichung $X = T[X]$**
 - z.B. `rectype bintree =  $\mathbb{Z} + \mathbb{Z} \times \text{bintree} \times \text{bintree}$`
 - Kanonische Elemente definiert durch Aufrollen der Gleichung

Repräsentation rekursiv definierter Strukturen

● Rekursive Typdefinition mit Gleichung $X = T[X]$

- z.B. `rectype bintree =  $\mathbb{Z}$  +  $\mathbb{Z} \times \text{bintree} \times \text{bintree}$`
- Kanonische Elemente definiert durch Aufrollen der Gleichung
- Verarbeitung durch induktiven Operator `let*  $f(x) = t$  in  $f(e)$`

liefert terminierende freie rekursive Funktionsdefinitionen

```
let* sum(b-tree) =  
  case b-tree of inl(leaf)  $\mapsto$  leaf  
    | inr(triple)  $\mapsto$  let  $\langle \text{num}, \text{pair} \rangle = \text{triple}$   
                        in let  $\langle \text{left}, \text{right} \rangle = \text{pair}$   
                        in num+sum(left)+sum(right)  
in sum(t)
```

Repräsentation rekursiv definierter Strukturen

- **Rekursive Typdefinition mit Gleichung $X = T[X]$**

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

```
let* sum(b-tree) =  
  case b-tree of inl(leaf)  $\mapsto$  leaf  
                | inr(triple)  $\mapsto$  let  $\langle \text{num}, \text{pair} \rangle = \text{triple}$   
                                   in let  $\langle \text{left}, \text{right} \rangle = \text{pair}$   
                                   in num+sum(left)+sum(right)  
in sum(t)
```

- **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

Syntax:

Kanonisch: $\text{rectype } X = T_X$

$\text{rec}\{\}(X.T_X)$

Nichtkanonisch: $\text{let}^* f(x) = t \text{ in } f(e)$

$\text{rec_ind}\{\}(e; f, x.t)$

INDUKTIVE TYPEN, FORMAL

Syntax:

Kanonisch: $\text{rectype } X = T_X$

$\text{rec}\{\}(X.T_X)$

Nichtkanonisch: $\text{let}^* f(x) = t \text{ in } f(e)$

$\text{rec_ind}\{\}(e; f, x.t)$

Auswertung:

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

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

INDUKTIVE TYPEN, FORMAL

Syntax:

Kanonisch: $\text{rectype } X = T_X$

$\text{rec}\{\}(X.T_X)$

Nichtkanonisch: $\text{let}^* f(x) = t \text{ in } f(e)$

$\text{rec_ind}\{\}(e; f, x.t)$

Auswertung:

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

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

Semantik:

$\text{rectype } X_1 = T_{X_1}$

$= \text{rectype } X_2 = T_{X_2} \quad \equiv \quad T_{X_1}[X/X_1] = T_{X_2}[X/X_2] \text{ f\"ur alle Typen } X$

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

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

INDUKTIVE TYPEN, FORMAL

Syntax:

Kanonisch: $\text{rectype } X = T_X$

$\text{rec}\{\}(X.T_X)$

Nichtkanonisch: $\text{let}^* f(x) = t \text{ in } f(e)$

$\text{rec_ind}\{\}(e; f, x.t)$

Auswertung:

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

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

Semantik:

$\text{rectype } X_1 = T_{X_1}$

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

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

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

Inferenzregeln und Taktiken im Appendix A.3.11 des Nuprl Manuals

RAHMENBEDINGUNGEN FÜR $\text{rectype } X = T_X$

Induktive Definitionen müssen **wohlfundiert** sein

RAHMENBEDINGUNGEN FÜR $\text{rectype } X = T_X$

Induktive Definitionen müssen wohlfundiert sein

- Semantik ist kleinster Fixpunkt von $T[X]$

RAHMENBEDINGUNGEN FÜR $\text{rectype } X = T_X$

Induktive Definitionen müssen wohlfundiert sein

- Semantik ist kleinster Fixpunkt von $T[X]$
 - Existenz des Fixpunkts muß gesichert sein

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

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` $X = X \rightarrow \mathbb{Z}$ müssen ausgeschlossen werden

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

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 $\text{rectype } X = X \rightarrow \mathbb{Z}$ müssen ausgeschlossen werden
 - $\text{rectype } X = X \rightarrow \mathbb{Z}$ hat $\lambda x. x \ x$ als kanonisches Element
 - $\lambda x. x \ x$ wäre sogar Extrakt-Term von $\vdash \text{rectype } X = X \rightarrow \mathbb{Z}$
- Syntaktische Einschränkungen erforderlich

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 $\text{rectype } X = X \rightarrow \mathbb{Z}$ müssen ausgeschlossen werden
 - $\text{rectype } X = X \rightarrow \mathbb{Z}$ hat $\lambda x. x \ x$ als kanonisches Element
 - $\lambda x. x \ x$ wäre sogar Extrakt-Term von $\vdash \text{rectype } X = X \rightarrow \mathbb{Z}$
- **Syntaktische Einschränkungen erforderlich**
 - Allgemeine Wohlfundiertheit rekursiver Typen ist unentscheidbar
 - Entspricht dem Halteproblem rekursiver Programme

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 $\text{rectype } X = X \rightarrow \mathbb{Z}$ müssen ausgeschlossen werden
 - $\text{rectype } X = X \rightarrow \mathbb{Z}$ hat $\lambda x. x \ x$ als kanonisches Element
 - $\lambda x. x \ x$ wäre sogar Extrakt-Term von $\vdash \text{rectype } X = X \rightarrow \mathbb{Z}$
- **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 $f(x) = t[f, x]$

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch** $f(x) = t[f, x]$

- z.B. $\lambda f. \text{let}^* \min_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \min_f(y+1) \text{ in } \min_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch** $f(x) = t[f, x]$
 - z.B. $\lambda f. \text{let}^* \min_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \min_f(y+1) \text{ in } \min_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$
 - Semantik: Kleinster Fixpunkt von $t[f, x]$

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch $f(x) = t[f, x]$**
 - z.B. $\lambda f. \text{let}^* \min_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \min_f(y+1) \text{ in } \min_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$
 - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu $\text{let}^* f(x) = t$ in $f(e)$ aber**

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch $f(x) = t[f, x]$**
 - z.B. $\lambda f. \text{let}^* \text{min}_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \text{min}_f(y+1) \text{ in } \text{min}_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$
 - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu $\text{let}^* f(x) = t \text{ in } f(e)$ aber**
 - Keine Koppelung an bekannte rekursiver Struktur erforderlich
 - Kein Extraktterm einer Eliminationsregel

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch $f(x) = t[f, x]$**
 - z.B. $\lambda f. \text{let}^* \min_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \min_f(y+1) \text{ in } \min_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$
 - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu $\text{let}^* f(x) = t \text{ in } f(e)$ aber**
 - Keine Koppelung an bekannte rekursiver Struktur erforderlich
 - Kein Extraktterm einer Eliminationsregel
 - Flexiblerer Einsatz in Programmierung (“reale” Programme)

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch $f(x) = t[f, x]$**
 - z.B. $\lambda f. \text{let}^* \text{min}_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \text{min}_f(y+1) \text{ in } \text{min}_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$
 - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu $\text{let}^* f(x) = t \text{ in } f(e)$ 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

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch $f(x) = t[f, x]$**
 - z.B. $\lambda f. \text{let}^* \text{min}_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \text{min}_f(y+1) \text{ in } \text{min}_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$
 - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu $\text{let}^* f(x) = t \text{ in } f(e)$ 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**

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch $f(x) = t[f, x]$**
 - z.B. $\lambda f. \text{let}^* \min_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \min_f(y+1) \text{ in } \min_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$
 - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu $\text{let}^* f(x) = t \text{ in } f(e)$ 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

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch $f(x) = t[f, x]$**
 - z.B. $\lambda f. \text{let}^* \min_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \min_f(y+1) \text{ in } \min_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$
 - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu $\text{let}^* f(x) = t \text{ in } f(e)$ 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

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch $f(x) = t[f, x]$**
 - z.B. $\lambda f. \text{let}^* \min_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \min_f(y+1) \text{ in } \min_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$
 - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu $\text{let}^* f(x) = t \text{ in } f(e)$ 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 $\text{letrec } f(x) = t$ ist Abkürzung für $Y(\lambda f. \lambda x. t)$

REKURSIVE FUNKTIONEN

- **Definition von Funktionen durch $f(x) = t[f, x]$**
 - z.B. $\lambda f. \text{let}^* \min_f(y) = \text{if } f(y)=0 \text{ then } y \text{ else } \min_f(y+1) \text{ in } \min_f(0)$
 $\lambda x. \text{let}^* \text{sqr}(y) = \text{if } x < (y+1)^2 \text{ then } y \text{ else } \text{sqr}(y+1) \text{ in } \text{sqr}(0)$
 - Semantik: Kleinster Fixpunkt von $t[f, x]$
- **Analog zu $\text{let}^* f(x) = t \text{ in } f(e)$ 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

- **Repräsentation unendlicher Datenstrukturen**

- **Repräsentation unendlicher Datenstrukturen**

- Rekursive Definition durch die Gleichung $X = T[X]$

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

- `stream` = $\text{Atom} \times \text{stream}$

- **Repräsentation unendlicher Datenstrukturen**

- Rekursive Definition durch die Gleichung $X = T[X]$

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

- `stream` = $\text{Atom} \times \text{stream}$

- **Formal ähnlich zum induktiven Datentyp**

- Kanonische Elemente definiert durch Aufrollen der Gleichung

- **Repräsentation unendlicher Datenstrukturen**

- Rekursive Definition durch die Gleichung $X = T[X]$

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

- $\text{stream} = \text{Atom} \times \text{stream}$

- **Formal ähnlich zum induktiven Datentyp**

- Kanonische Elemente definiert durch Aufrollen der Gleichung

- Andere Semantik für $\text{inftype } X = T_X$: größter Fixpunkt von $T[X]$,

- **Repräsentation unendlicher Datenstrukturen**

- Rekursive Definition durch die Gleichung $X = T[X]$

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

- `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

- **Repräsentation unendlicher Datenstrukturen**

- Rekursive Definition durch die Gleichung $X = T[X]$

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

- $\text{stream} = \text{Atom} \times \text{stream}$

- **Formal ähnlich zum induktiven Datentyp**

- Kanonische Elemente definiert durch Aufrollen der Gleichung

- Andere Semantik für $\text{inftype } X = T_X$: größter Fixpunkt von $T[X]$,

- Kein Basisfall für Induktionsanfang erforderlich

- Verarbeitung durch induktiven Operator $\text{let}^\infty f(x) = t \text{ in } f(e)$

- **Repräsentation unendlicher Datenstrukturen**

- Rekursive Definition durch die Gleichung $X = T[X]$

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

- `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 **let**[∞] $f(x) = t$ in $f(e)$

- **Parametrisierte simultane Rekursion möglich**

- **Repräsentation unendlicher Datenstrukturen**

- Rekursive Definition durch die Gleichung $X = T[X]$

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

- $\text{stream} = \text{Atom} \times \text{stream}$

- **Formal ähnlich zum induktiven Datentyp**

- Kanonische Elemente definiert durch Aufrollen der Gleichung

- Andere Semantik für $\text{inftype } X = T_X$: größter Fixpunkt von $T[X]$,

- Kein Basisfall für Induktionsanfang erforderlich

- Verarbeitung durch induktiven Operator $\text{let}^\infty f(x) = t \text{ in } f(e)$

- **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

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)

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

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
 - $\text{Top} \equiv \cap x:\text{Void}.\text{Void}$

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
 - $\text{Top} \equiv \cap x:\text{Void}.\text{Void}$
- 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

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
 - $\text{Top} \equiv \cap x:\text{Void}.\text{Void}$
- 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!)

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
 - $\text{Top} \equiv \cap x:\text{Void}.\text{Void}$
- 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

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 $s \sim t$**

- 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

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 $s \sim t$**

- 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

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 $s \sim t$**

- 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

● Direkte Berechnung

- Reduktion an beliebiger Stelle in Sequenz
- Rückwärtsreduktion (vom Kontraktum zum Redex) möglich

- **Direkte Berechnung**

- Reduktion an beliebiger Stelle in Sequenz
- Rückwärtsreduktion (vom Kontraktum zum Redex) möglich

- **Falten und Auffalten von Definitionen**

- Einsetzen der Definition für eine benutzerdefinierte Abstraktion
- Zugriff über den Namen des Definitionsobjektes in der Library

- **Direkte Berechnung**

- Reduktion an beliebiger Stelle in Sequenz
- Rückwärtsreduktion (vom Kontraktum zum Redex) möglich

- **Falten und Auffalten von Definitionen**

- Einsetzen der Definition für eine benutzerdefinierte 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

- **Direkte Berechnung**

- Reduktion an beliebiger Stelle in Sequenz
- Rückwärtsreduktion (vom Kontraktum zum Redex) möglich

- **Falten und Auffalten von Definitionen**

- Einsetzen der Definition für eine benutzerdefinierte 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

- **Direkte Berechnung**

- Reduktion an beliebiger Stelle in Sequenz
- Rückwärtsreduktion (vom Kontraktum zum Redex) möglich

- **Falten und Auffalten von Definitionen**

- Einsetzen der Definition für eine benutzerdefinierte 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

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

TYPENTHEORIE IM RÜCKBLICK

Inferenzkalkül für Mathematik & Programmierung

TYPENTHEORIE IM RÜCKBLICK

Inferenzkalkül für Mathematik & Programmierung

- **Extrem ausdrucksstarkes Inferenzsystem**
 - Vereinheitlicht und erweitert Logik, λ -Kalkül und einfache Typentheorie

TYPENTHEORIE IM RÜCKBLICK

Inferenzkalkül für Mathematik & Programmierung

- **Extrem ausdrucksstarkes Inferenzsystem**

- Vereinheitlicht und erweitert Logik, λ -Kalkül und einfache Typentheorie
- Formalisierung “natürlicher” Gesetze der zentralen Konzepte

Inferenzkalkül für Mathematik & Programmierung

- **Extrem ausdrucksstarkes Inferenzsystem**

- Vereinheitlicht und erweitert Logik, λ -Kalkül und einfache Typentheorie
- Formalisierung “natürlicher” Gesetze der zentralen Konzepte
- Direkte Darstellung anstatt Simulation

Inferenzkalkül für Mathematik & Programmierung

● Extrem 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
 - Mathematischen Grundkonzepten
 - Grundkonstrukten der Programmierung, einschließlich Rekursion
 - Prädikatenlogik (höherer Stufe)

Inferenzkalkül für Mathematik & Programmierung

● Extrem 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
 - Mathematischen Grundkonzepten
 - Grundkonstrukten der Programmierung, einschließlich Rekursion
 - Prädikatenlogik (höherer Stufe)

● Praktische Probleme

- Beweise erfordern viel Schreibarbeit

Inferenzkalkül für Mathematik & Programmierung

● Extrem 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
 - Mathematischen Grundkonzepten
 - Grundkonstrukten der Programmierung, einschließlich Rekursion
 - Prädikatenlogik (höherer Stufe)

● Praktische Probleme

- Beweise erfordern viel Schreibarbeit → *Interaktive Beweissysteme*

Inferenzkalkül für Mathematik & Programmierung

● Extrem 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
 - Mathematischen Grundkonzepten
 - Grundkonstrukten der Programmierung, einschließlich Rekursion
 - Prädikatenlogik (höherer Stufe)

● Praktische Probleme

- Beweise erfordern viel Schreibarbeit → *Interaktive Beweissysteme*
- Beweise sind unübersichtlich (viele Regelanwendungen)

Inferenzkalkül für Mathematik & Programmierung

● Extrem 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
 - Mathematischen Grundkonzepten
 - Grundkonstrukten der Programmierung, einschließlich Rekursion
 - Prädikatenlogik (höherer Stufe)

● Praktische Probleme

- Beweise erfordern viel Schreibarbeit → *Interaktive Beweissysteme*
- Beweise sind unübersichtlich (viele Regelanwendungen)
- Beweise sind schwer zu finden (viele Regeln und Parameter)

Inferenzkalkül für Mathematik & Programmierung

● Extrem 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
 - Mathematischen Grundkonzepten
 - Grundkonstrukten der Programmierung, einschließlich Rekursion
 - Prädikatenlogik (höherer Stufe)

● Praktische Probleme

- Beweise erfordern viel Schreibarbeit → *Interaktive Beweissysteme*
- Beweise sind unübersichtlich (viele Regelanwendungen)
- Beweise sind schwer zu finden (viele Regeln und Parameter)
→ *Automatisierung der Beweisführung*

AUSBLICK: THEMEN DES ZWEITEN TEILS (SOMMER 2006)

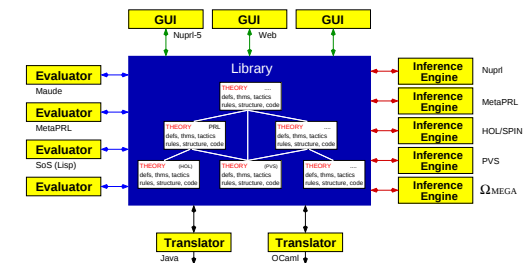
Konstruiere semiautomatische Beweissysteme

AUSBLICK: THEMEN DES ZWEITEN TEILS (SOMMER 2006)

Konstruiere semiautomatische Beweissysteme

● Aufbau von Beweissystemen

- Implementierung interaktiver Beweisassistenten
- Das NuPRL Logical Programming Environment



AUSBLICK: THEMEN DES ZWEITEN TEILS (SOMMER 2006)

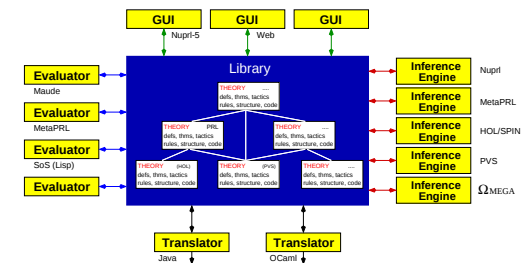
Konstruiere semiautomatische Beweissysteme

● Aufbau von Beweissystemen

- Implementierung interaktiver Beweisassistenten
- Das NuPRL Logical Programming Environment

● Beweisautomatisierung

- Taktisches Beweisen
- Entscheidungsprozeduren
- Integration externer Systeme



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

⋮

