Nützliche Typen: natürliche Zahlen und algebraische Datentypen

- Natürliche Zahlen: nicht *konservativ*!
- Erfordert *zusätzliches* Axiom: *Unendlichkeit*.
- Neuer Typ indmit $Z :: ind, S : ind \Rightarrow S$ und

  $inj\ S \quad Sx \neq Z$

- Damit `nat` definierbar.
- Warum ind? Auch für *andere* Datentypen. . .
- Damit algebraische Datentypen (`datatype`) etc.
  - `datatype`: nur *kovariante* Rekursion

Einige nützliche Typen

- *Theorie* `Set`: getypte *Mengen*, $\alpha\,\texttt{set} \equiv \alpha \Rightarrow Bool$
- *Theorien* `Sum` und `Produkt`: *Summe* und *Produkte*
- *Numerische Typen*:
  - *Natürliche* Zahlen `nat`, *ganze* Zahlen `int`, *reele* Zahlen `real`, *komplexe* Zahlen `complex`.
  - *Arithmetische Operatoren* `+`, `-`, `*`; *Konstanten* ("12343423");
  - *Entscheidungsprozeduren*
- *Theorie* `Datatype`: α `option` ist entweder `Some x` oder `None`
- *Theorie* `List`: *Listen* α`list`
- *Theorie* `Map`: endliche, partielle *Abbildungen* α `~=>` β

Zusammenfassung

- Isabelle: *LCF-Architektur* mit *logischem Kern* und *Meta-Logik*
- Beweisen durch *Resolution*
  - *Lifting*; *Eliminations*- und *Destruktionsresolution*.
- *Typdefinitionen*: Einschränkung bestehender Typen
- Der Typ ind: *Unendlichkeitsaxiom*
- Nützliche vordefinierte Typen
  - *numerische* Typen, *Listen*, *Abbildungen*, *Mengen*. . .

Vorlesung vom 05.12.05: Modellierung funktionaler Programme

Wo sind wir?

- Teil I: Grundlagen formaler Logik
- Teil II: Modellierung
  - Datentypen
  - *Rekursive Programme*
  - Imperative Programme
- Teil III: Spezifikationsformalismen

Fahrplan

- Funktionale Programme:
  - *Rekursive* Funktionen + *algebraische* Datentypen
- *Rekursion* auch bei *imperative Programme*
- Modellierung von *Rekursion* durch *Fixpunkte*
- Beweis durch *Fixpunktinduktion*

Rekursion

- Problem: *rekursive* Funktionen

  $$fac \equiv \lambda\, n.\, if\ \ n = 0\ \ then\, 1\ else\, n * (fac\ (n-1))$$

  - Keine *konservative* Definition
- Lösung: Modellierung durch *Fixpunkte*
  - Gegeben $fix : (\alpha \Rightarrow \alpha) \Rightarrow \alpha$ mit $fix\ F = F(fix\ F)$, dann

    $$fac \equiv fix(\lambda\, F(\lambda\, n.\, if\ \ n = 0\ \ then\, 1\ else\, n * (F\ (n-1))))$$

    F heißt *Approximationsfunktional*
  - Typ von fix? Hier: $fix :: (Nat \Rightarrow Nat) \Rightarrow Nat \Rightarrow Nat)$
  - Wie *definieren* wir fix?

Fixpunkte

- Ungetypter Lambda-Kalkül: Fixpunktkombinator Y
- Getypter Lambda-Kalkül: *Familie* von Fixpunktkombinatoren $Y_\alpha : (\alpha \Rightarrow \alpha) \Rightarrow \alpha$
- In HOL: nicht jede Funktion hat einen Fixpunkt. (Gegenbeispiel: $\neg$)
- Lösung: *Konstruktion* von Fixpunkten
- Verschiedene Ansätze
  - Allgemein: *Ordnung*, *aufsteigende* Ketten
  - *Vollständige Halbordnungen* (LCF)
  - *Wohlfundierte Rekursion*

Vollständige Halbordnungen

*Def (*Halbordung*)*: reflexive, transitive, antisymmetrische Relation $A \times A$.

*Def (*Vollständige Halbordnung, cpo*)*: Halbordnung $(A, \sqsubseteq)$ mit kleinstem Element $\bot \in A$, so dass jede aufsteigende Kette F ein Supremum $sup\, F$ besitzt.

(Gerichtete volllständige Halbordnung, dcpo: ohne kleinstes Element)

Beispiele:

- Für Menge X ist $(X_\bot, \sqsubseteq)$ *flacher* cpo mit $X_\bot = X + \{\bot\}$, $x \sqsubseteq y$ gdw. $x = \bot$.
- Für cpo's A, B ist Funktionsraum $A \Rightarrow B$ ein cpo unter *punktweiser* Ordnung: $f \sqsubseteq g$ gdw. $\forall\, a \in A. fa \sqsubseteq ga$

Stetigkeit und Fixpunkte

Def: Funktion $f : A \Rightarrow B$ $(A, B$ cpos) heißt

- *monoton*, wenn $a \sqsubseteq a' \longrightarrow f(a) \sqsubseteq f(a')$.
- *stetig*, wenn f monoton und $f(sup\, F) = sup\, fF$

Fixpunktsatz: Jede Stetige Funktion $f : A \Rightarrow A$ hat *kleinsten Fixpunkt* $fix f \in A$.

Logik berechenbarer Funktionen

- Alle Basisdatentypen (Nat, $Bool$ etc.) sind *flache* cpo's.
- Basisfunktionen ($if\ then\ else, +, -$ etc.) sind *geliftet* (und damit *stetig*).
- Funktionsräume $A \Rightarrow B$ werden *punktweise* geordnet.
- Approximationsfunktional F ist *stetig*, wenn es aus stetigen Funktionen besteht.
- Laut Fixpunktsatz hat jedes *stetige Approximationsfunktional* einen kleinsten Fixpunkt.

LCF in HOL

- Kodierung von LCF in HOL: HOLCF
  - Neue *Objektlogik*
  - Konservative Entwicklung
  - *Typklasse* cpo, *stetige* Funktionen, *Fixpunkte*
- Problem:
  - keine *automatische* β-Reduktion
  - *Stetigkeitsbeweise* nötig, neue *Datentypen*
  - Schlechtere Beweisunterstützung (als in HOL)
- *Alternativer* Ansatz in *HOL*: *wohlfundierte* Rekursion

Wohlfundierte Ordnung und Induktion

- *Def:* Ordnung $\sqsubseteq\, \subseteq A \times A$ *wohlfundiert*: keine unendlich absteigenden Ketten $x_0, x_2, x_3, \ldots$ mit $x_{n+1} \sqsubseteq x_n$.
- *Def (*Wohlfundierte Induktion*)*: Wenn $\forall\, x.(\forall\, y. y \sqsubseteq x \wedge P(y)) \longrightarrow P(x)$, dann $\forall\, x. P(x)$.
  - Induktionsannahme gilt für *alle* kleineren Werte
  - Keine *explizite* Induktionsverankerung
- Wohlfundiertheit in HOL:

  $wf :: (\alpha \times \alpha)\ set \Rightarrow Bool$
  $wf\ R \equiv (\forall\, x.(\forall\, y. y \sqsubseteq x \longrightarrow P(y)) \longrightarrow P(x)) \longrightarrow \forall\, x. P(x)$

Wohlfundierte Rekursion

- Für $f : \alpha \Rightarrow \beta$
- Wohlfundierte Ordnung auf α *Parameter* der Definition
- Voraussetzung: *rekursiver Aufruf* mit *kleineren* Argumenten.

  D.h. für Approximationsfunktional $H : (\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ muß gelten $Hfa = Hf'a$ gdw. $\forall\, x \sqsubseteq a. fx = f'x$

Wohlfundierte Rekursion

- Gegeben $H : (\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ und $\sqsubseteq$.
- Ziel: Definition des *Fixpunktes* als *Relation* $\alpha \times \beta\ Set$
- *Wohlfundierte Rekursionsordnung*:
  Definiere Relation $R_{\sqsubseteq,H}$ als *kleinste Relation* so dass $\forall\, z.(z \sqsubseteq x) \longrightarrow (z, gz) \in R_{\sqsubseteq,H}$, dann $(x, Hgx) \in R_{\sqsubseteq,H}$.
  - *Induktive* Definition
- Approximationsfunktional H bei x 'abschneiden' (`cut`):

$$f\ |_x\ (y) \stackrel{def}{=} \begin{cases} f(x) & y \sqsubseteq x \\ undefined & ows \end{cases}$$

Wohlfundierte Rekursion

- Gegeben *Approximationsfunktional* $H : (\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$ und $\sqsubseteq$ auf α.
- Rekursionsordnung $R_{F,\sqsubseteq}$ wird induktiv als Relation auf $\alpha \times \beta$ definiert mit $F \stackrel{def}{=} \lambda\, fx. H(f\ |_x)x$.
- *Wohlfundierte Rekursion* gegeben durch $R_{F,\sqsubseteq}$:

  $wfrec_{\sqsubseteq}\, H = \iota y.(x, y) \in R_{\lambda\, fx. H(f|_x)x, \sqsubseteq}$
- Wohldefiniert, wenn jeder Aufruf Argumente *verkleinert*

Wohlfundierte Rekursion in Isabelle

- *Wohlfundierte Ordnung* als *Argument* der Funktionsdefinition
- *Wohldefiniertheit* als *Beweisverpflichtung*
- Vordefinierte Taktiken und Ordnungen: `less_than`, `measure`, . . .
- 1. Beispiel: *Fakultät*

```
consts
  fac :: "nat => nat"
recdef fac "less_than"
 "fac n = (if n= 0 then 1 else n* fac (n - 1))"
```

- 2. Beispiel: *Mergesort* . . .


```
consts merge :: "('a::linorder)list * 'a list => 'a list"
recdef merge "measure (%(xs,ys). size xs + size ys)"
  "merge(x#xs, y#ys) =
     (if x < y then x # merge(xs, y#ys)
               else y # merge(x#xs, ys))"
  "merge(xs,[]) = xs"
  "merge([],ys) = ys"

consts msort :: "('a::linorder) list => 'a list"
recdef msort "measure size"
    "msort [] = []"
    "msort [x] = [x]"
    "msort xs = merge(msort(take (size xs div 2) xs),
      msort(drop (size xs div 2) xs))"
```