Arbeitsgemeinschaft Deklarative Programmierung

Prof. Dr. M. Hanus
Lehr- und Forschungsgebiet Informatik II
Rheinisch-Westfälische Technische Hochschule Aachen
Ahornstraße 55, 52072 Aachen

Unifikation höherer Ordnung mittels expliziter Substitutionen

Vortrag von Hans-Georg Eßer
`esser@i2.informatik.rwth-aachen.de`
`http://www-users.informatik.rwth-aachen.de/~esser/`

22.01.1997

Grundlage: G. Dowek, Th. Hardin, C. Kirchner: *Higher Order Unification via Explicit Substitutions.* Technical Report RR-2709, INRIA Lorraine, November 1995

- Unifikationsprobleme höherer Ordnung im λ-Kalkül
- Reduktion auf E-Unifikationsprobleme erster Ordnung
  (für solche gibt es effiziente Lösungsverfahren)
- Kalkül, der zwei Klassen von Variablen unterscheidet:
  1. Variablen der $\beta\eta$-Konversion
  2. Unifikationsvariablen
- $\beta\eta$-Konversion wird internalisiert durch explizite Substitutionen;
  nach außen nur noch Unifikationsvariablen sichtbar
- Aus Lösungen des transformierten Problems können Lösungen des Ausgangsproblems gewonnen werden.

- Der λ-*Kalkül*:

$$a ::= x \mid (a\ a) \mid \lambda x.a \qquad (x \in \mathcal{V})$$

- Unifikationsproblem:
Zu zwei Termen t_1, t_2 suche *Substitution* θ mit $\theta(t_1) = \theta(t_2)$.

- Zwei Beispiele:
  1. $\lambda y.(X\ a) =^?_{\beta\eta} \lambda y.a$
  hat Lösungen
  $\{X/\lambda x.x\}$ und $\{X/\lambda x.a\}$
  2. $(f\ (X\ a)) =^?_{\beta\eta} (X\ (f\ a))$
  hat Lösungen
  $\{X/\lambda x.x\}$,
  $\{X/\lambda x.(f\ x)\}$,
  $\{X/\lambda x.(f\ (f\ x))\}$, usw.

- λ-Kalkül kein Kalkül erster Ordnung; Substitutionen erfordern Umbenennungen:
  - $\theta = \{x \mapsto a\}$, $\theta(\lambda x.x) = \lambda x.a$, aber
  $\theta(\lambda y.y) = \lambda y.y$
  nicht äquivalent (modulo α-Konv.)
  - $\theta = \{x \mapsto y\}$, $\theta(\lambda y.x) = \lambda y.y$, aber
  $\theta(\lambda z.x) = \lambda z.y$
  „Variable capture“

- Substitutionen erster Ordnung, d.h. ohne Variablen-Umbenennung (*Graftings*):

$$\{x \mapsto t\}$$

Substitutionen (höherer Ordnung):

$$\{x/t\}$$

- Im λ-Kalkül kein allgemeinster Unifikator, sondern nur ein *vollständiges System von Unifikatoren*
- Transformiere Unifikationsproblem in *gelöste Form*: gelöste Gleichungen und Flexibel-Flexibel-Paare;
korrekt und vollständig.

- Ansatz:
E-Unifikation: Unif. bzgl. Gleichheitstheorie
Unifikation höherer Ordnung
$\equiv$ E-Unif. bzgl. $\beta\eta$-Konversion
Problem: Methoden der E-Unif. verwenden Graftings; können hier nicht angewandt werden.
- Transformation des Unif.-Problems in einen Kalkül, in dem wir ein E-Unif.-Problem erster Ordnung lösen müssen
$\rightarrow$ *$\lambda\sigma$-Kalkül mit expliziten Substitutionen*

Erster Schritt: Unterscheidung verschiedener Arten von Variablen

1. *Gebundene Variablen*: von der Unifikation nicht betroffen – nur freie Variablen betrachten
2. *Konstanten*, die beim Unifikationsprozeß nicht substituiert werden dürfen
3. echte *Unifikationsvariablen*, über die das Unifikationsproblem definiert wird.

- Zwei Variablenmengen: $\mathcal{V}$ für Typ 1,2; $\mathcal{X}$ für Typ 3
- Schreibweisen: $x \in \mathcal{V}, X \in \mathcal{X}$

Zweiter Schritt:
de Bruijn-Notation für λ-Ausdrücke

- de Bruijn-Notation des λ-Kalküls: Variablen ersetzen durch *Bindungshöhe* (# zu überspringender λ-Binder)

  $a ::= \mathtt{n} \mid X \mid \lambda\, a \mid (a\, a) \quad (\mathtt{n} \in \mathbb{N}, X \in \mathcal{X})$

  (Variablen aus $\mathcal{V}$ durch $\mathtt{n} \in \mathbb{N}$ ersetzt)

- *Referential*: Liste von Variablennamen Übersetzung in de Bruijn-Notation bzgl. eines Referentials:

  1. $tr(x, \mathcal{R}) = \mathtt{j}$,
     wobei j erste Position von x in $\mathcal{R}$,
  2. $tr(X, \mathcal{R}) = X \qquad$ für $X \in \mathcal{X}$,
  3. $tr((a\ b), \mathcal{R}) = (tr(a, \mathcal{R})\ tr(b, \mathcal{R}))$,
  4. $tr(\lambda\ x.a, \mathcal{R}) = (\lambda\ tr(a, x.\mathcal{R}))$.

Unsere alten Beispiele:

1. $t_1 := \lambda y.(X\ a)$, $t_2 := \lambda y.a$
   Übersetze bzgl. Referential, das die freien $\mathcal{V}$-Variablen enthält: $\mathcal{R} = \{a\}$.
   $tr(t_1, \mathcal{R}) = tr(\lambda y.(X\ a), \{a\})$
   $= (\lambda\ tr((X\ a), \{y, a\})$
   $= (\lambda\ (tr(X, \{y, a\})\ tr(a, \{y, a\})))$
   $= (\lambda\ (X\ \mathsf{2}))$
   $tr(t_2, \mathcal{R}) = tr(\lambda y.a, \{a\})$
   $= (\lambda\ tr(a, \{y, a\}))$
   $= (\lambda\ \mathsf{2})$.
2. $t_3 := (f\ (X\ a))$, $t_4 := (X\ (f\ a))$
   Referential $\mathcal{R} := \{a, f\}$
   $tr(t_3, \mathcal{R}) = tr((f\ (X\ a))), \{a, f\})$
   $= (tr(f, \{a, f\})\ tr((X\ a), \{a, f\})$
   $= (\mathsf{2}\ (tr(X, \{a, f\})\ tr(a, \{a, f\})))$
   $= (\mathsf{2}\ (X\ 1))$
   $tr(t_4, \mathcal{R}) = (X\ (\mathsf{2}\ \mathsf{1}))$ analog.

β-Reduktion in beiden Kalkülen

- im λ-Kalkül:
  $$(\lambda x.a)b \to^\beta \{x/b\}a$$
- im λ-Kalkül in de Bruijn-Notation:
  $$((\lambda\, a)\ b) \to^\beta \{\mathtt{1}/b\}a,$$
  Substitution wie folgt definiert:

$$\begin{array}{ll}
\{\mathtt{n}/b\}(a_1\ a_2) & := (\{\mathtt{n}/b\}a_1\ \{\mathtt{n}/b\}a_2) \\
\{\mathtt{n}/b\}X & := X \\
\{\mathtt{n}/b\}\lambda a & := \lambda(\{\mathtt{n+1}/b^+\}a) \\
\{\mathtt{n}/b\}\mathtt{m} & := \mathtt{m-1},\ \text{falls } m > n \\
 & \ b,\quad \text{falls } m = n \\
 & \ \mathtt{m},\quad \text{falls } m < n
\end{array}$$

b^+: alle freien de Bruijn-Nummern in b um 1 erhöhen
($^+$ *Lifting-Operator*)

η-*Reduktion* in beiden Kalkülen

- im λ-Kalkül:

$$\lambda x.(a\ x) \rightarrow^{\eta} a, \text{ falls } x \notin FV(a)$$

- im λ-Kalkül in de Bruijn-Notation:

$$\lambda(a\ \mathtt{1}) \rightarrow^{\eta} b, \text{ falls } \exists b: \ a = b^{+}$$

Dritter Schritt: Übergang zum $\lambda\sigma$-*Kalkül mit expliziten Substitutionen*

- Termersetzungssystem erster Ordnung
- explizite Behandlung der Substitutionen, die durch β-Reduktionen ausgelöst werden
- enthält λ-Kalkül in de Bruijn-Notation
- *Internalisierung* externer Operationen:
  λ-Kalkül: Anwendung einer Funktion f auf ein Argument x, $apply(f, x)$, internalisiert durch Ausdrücke $(f\ x)$.
  Dazu Ersetzungsregeln (β-Reduktion), so daß z.B. $(\lambda x.x\ 0) \rightarrow^{\beta} 0$.
  Also: Applikation *interner* Operator ( ) des λ-Kalküls.
  Aber: Substitution und Variablenumbenennung weiterhin *externe* Operationen.

- *simultane Substitution*:
  id identische Substitution
  $a_1.a_2...a_n.id$ ersetzt `1` durch a_1, ... ,`n` durch a_n und verringert alle anderen (freien) de Bruijn-Nummern um n.
- Lifting-Operation $^+$ wird internalisiert durch Operator $\uparrow$; Interpretation als unendliche simultane Substitution `2.3.4`...
- Kompositions-Operator $\circ$
  Schreibe de Bruijn-Nummer `n+1` als $\mathtt{1}[\uparrow \circ ... \circ \uparrow] = \mathtt{1}[\uparrow^n]$
- Jetzt Substitutionen Teil des *Terms* (!)
  $\mathcal{X}$: Ausdrucks-Metavariablen
  $\mathcal{Y}$: Substitutions-Metavariablen
- Terme des $\lambda\sigma$-Kalküls:

$$a ::= \mathtt{1} \mid X \mid (a\ a) \mid \lambda a \mid a[s]$$

$$s ::= Y \mid id \mid \uparrow \mid a.s \mid s \circ s$$

  wobei $X \in \mathcal{X}$ und $Y \in \mathcal{Y}$.

- Regeln $\boldsymbol{\lambda\sigma}$ des $\lambda\sigma$-Kalküls

Beta	$(\lambda a)b$	$\rightarrow a[b.id]$
App	$(a\ b)[s]$	$\rightarrow (a[s]\ b[s])$
VarCons	$\mathtt{1}[a.s]$	$\rightarrow a$
Id	$a[id]$	$\rightarrow a$
Abs	$(\lambda a)[s]$	$\rightarrow \lambda(a[\mathtt{1}.(s \circ \uparrow)])$
Clos	$(a[s])[t]$	$\rightarrow a[s \circ t]$
IdL	$id \circ s$	$\rightarrow s$
ShiftCons	$\uparrow \circ (a.s)$	$\rightarrow s$
AssEnv	$(s_1 \circ s_2) \circ s_3$	$\rightarrow s_1 \circ (s_2 \circ s_3)$
MapEnv	$(a.s) \circ t$	$\rightarrow a[t].(s \circ t)$
IdR	$s \circ id$	$\rightarrow s$
VarShift	$\mathtt{1}.\uparrow$	$\rightarrow id$
SCons	$\mathtt{1}[s].(\uparrow \circ s)$	$\rightarrow s$
Eta	$\lambda(a\ \mathtt{1})$	$\rightarrow b$ falls $a =_\sigma b[\uparrow]$

- definiert Gleichheitstheorie $=_{\lambda\sigma}$
- $\boldsymbol{\sigma} = \boldsymbol{\lambda\sigma} \setminus \{$**Beta, Eta**$\}$

- Beispiel für die Anwendung von $\boldsymbol{\lambda\sigma}$:
  Im λ-Kalkül: $t := ((\lambda x.(f\ x))\ a)$
  $\rightarrow^{\beta} \{x/a\}(f\ x) = (f\ a)$
  Wähle Referential $\mathcal{R} = \{a, f\}$.

  $tr(t, \{a, f\})$
  $= (tr((\lambda x.(f\ x)), \{a, f\}\ tr(a, \{a, f\})$
  $= ((\lambda\ tr((f\ x), \{x, a, f\}))\ \mathsf{1})$
  $= ((\lambda\ (tr(f, \{x, a, f\})\ (tr(x, \{x, a, f\})))\ \mathsf{1})$
  $= ((\lambda\ (\mathsf{3}\ \mathsf{1}))\ \mathsf{1})$
  $\rightarrow_{\mathbf{Beta}}\quad (\mathsf{3}\ \mathsf{1})[\mathsf{1}.id]$
  $\rightarrow_{\mathbf{App}}\quad (\mathsf{3}[\mathsf{1}.id]\ \mathsf{1}[\mathsf{1}.id])$
  $\rightarrow_{\mathbf{VarCons}}\quad (\mathsf{3}[\mathsf{1}.id]\ \mathsf{1})$
  $=\quad ((\mathsf{1}[\uparrow \circ \uparrow])[\mathsf{1}.id]\ \mathsf{1})$
  $\rightarrow_{\mathbf{Clos}}\quad (\mathsf{1}[\uparrow \circ \uparrow \circ \mathsf{1}.id]\ \mathsf{1})$
  $\rightarrow_{\mathbf{ShiftCons}}\quad (\mathsf{1}[\uparrow \circ id]\ \mathsf{1})$
  $\rightarrow_{\mathbf{IdR}}\quad (\mathsf{1}[\uparrow]\ \mathsf{1})$
  $=\quad (\mathsf{2}\ \mathsf{1}) = tr((f\ a), \{a, f\})$

  D.h.: $t \rightarrow^{\beta} t_{\beta}$ und $tr(t, \mathcal{R}) \rightarrow_{\lambda\sigma} tr(t_{\beta}, \mathcal{R})$.

- Typen
  - Einführung von Typen im λ- und λσ-Kalkül
  - Betrachte nur Unifikationsprobleme mit korrekt getypten Ausdrücken
  - Wesentlich für Funktion der Algorithmen
  - Hier zur Vereinfachung keine Betrachtung des Typkonzepts

- Übersetzung vom λ-Kalkül (in de Bruijn-Notation) in den $\lambda\sigma$-Kalkül: *Pre-Cooking*
- Ziel: Transformation des Unif.-Problems in einen Kalkül erster Ordnung (Graftings)
  $\mathcal{X}$-Substitutionen im λ-Kalkül entsprechen $\mathcal{X}$-Graftings im $\lambda\sigma$-Kalkül.
- $\Lambda_{DB}(\mathcal{X}) \to \mathcal{T}_{\lambda\sigma}(\mathcal{X}) : a \mapsto a_F$ mit $a_F := F(a, 0)$, definiert durch
  1. $F((\lambda a), n) := (\lambda\ F(a, n+1))$,
  2. $F((a\ b), n) := (F(a, n)\ F(b, n))$,
  3. $F(\mathtt{k}, n) := \mathtt{1}[\uparrow^{k-1}]$,
  4. $F(X, n) := X[\uparrow^n]$.
- Pre-Cooking ist typ-erhaltend.

- Zusammenhang Substitutionen (λ) / Graftings ($\lambda\sigma$):

$$(a\{X_1/b_1, ..., X_p/b_p\})_F$$
$$= \{X_1 \mapsto (b_1)_F, ..., X_p \mapsto (b_p)_F\}a_F.$$

- Wichtig: Übergang von $\beta\eta$-Äquivalenz zur $\lambda\sigma$-Äquivalenz:

$$a =_{\beta\eta} b \iff a_F =_{\lambda\sigma} b_F$$

- D.h. insbesondere:
  Unif.-Problem (höh. Ordnung) $a =^?_{\beta\eta} b$ mit Unifikator θ lösbar $\Rightarrow$
  Unif.-Problem (1. Ordnung) $a_F =^?_{\lambda\sigma} b_F$ mit Unifikator (Grafting) θ_F lösbar.

  Frage: Rückrichtung? ($\to$ später)

Vierter Schritt: Lösen des Unifikationsproblems im $\lambda\sigma$-Kalkül.

- getypter $\lambda\sigma$-Kalkül: bedingtes Termersetzungssystem
  Problem $a_F =^?_{\lambda\sigma} b_F$: bedingtes E-Unifikationsproblem erster Ordnung.
  $\rightarrow$ verschiedene Lösungsverfahren, z.B. (bedingtes) Narrowing.
- Unifikationsalgorithmus **Unif**; nutzt die besonderen Eigenschaften von $\boldsymbol{\lambda\sigma}$

- Erste Idee zur Lösung unseres ursprünglichen Problems (Unif.-Problem höherer Ordnung im λ-Kalkül):
  1. Übersetze $a =^?_{\beta\eta} b$ in den $\lambda\sigma$-Kalkül (mit Pre-Cooking-Funktion F),
  2. Wende **Unif** auf $a_F =^?_{\lambda\sigma} b_F$ an (erhalte gelöste Form P)
  3. Rückübersetze P in eine gelöste Form $R = F^{-1}(P)$ im λ-Kalkül.
     Lösungen von R = Lös. von $a =^?_{\beta\eta} b$
- Nicht so einfach möglich:
  P muß nicht im Bild der Pre-Cooking-Abbildung liegen (d.h.: nicht jeder $\lambda\sigma$-Term Bild eines λ-Terms unter F).

- **Back**: bestimmt zu gelöster Form P des Unif.-Problems $a_F =^?_{\lambda\sigma} b_F$ ein gelöstes System Q, das in Bild(F) liegt und dieselben Lösungen wie P hat.
- Lösungsschema:

$$\begin{tikzcd} a =^?_{\beta\eta} b \arrow[d, "F"'] & & R \\ a_F =^?_{\lambda\sigma} b_F \arrow[r, "\mathbf{Unif}"'] & P \arrow[r, "\mathbf{Back}"'] & Q \arrow[u, "F^{-1}"'] \end{tikzcd}$$

- Es folgen noch die Regeln für **Unif** und **Back**.

Dec-λ
$P \wedge \lambda a =^?_{\lambda\sigma} \lambda b$
$\to P \wedge a =^?_{\lambda\sigma} b$
(Dekomposition λ-Abstraktion)

Dec-App
$P \wedge (\mathtt{n}\ a_1 \ldots a_p) =^?_{\lambda\sigma} (\mathtt{n}\ b_1 \ldots b_p)$
$\to P \wedge \bigwedge_{i=1..p} a_1 =^?_{\lambda\sigma} b_i$
(Dekomposition Applikation)

Dec-Fail
$P \wedge (\mathtt{n}\ a_1 \ldots a_p) =^?_{\lambda\sigma} (\mathtt{m}\ b_1 \ldots b_q)$
$\to$ **false**, falls $\mathtt{n} \neq \mathtt{m}$
(Dekomposition Appl. nicht möglich)

Exp-λ
P
$\to \exists Y, P \wedge X =^?_{\lambda\sigma} \lambda Y,$
falls $X \in TVar(P)$, $Y \notin TVar(P)$
und X nicht gelöst
(Anwendung des Grafting $\{X \mapsto \lambda Y\}$)

Exp-App
$P \wedge X[a_1 \ldots a_p . \uparrow^n] =^?_{\lambda\sigma} (\mathtt{m}\ b_1 \ldots b_q)$
$\to P \wedge X[a_1 \ldots a_p . \uparrow^n] =^?_{\lambda\sigma} (\mathtt{m}\ b_1 \ldots b_q)$
$\wedge \bigvee_{r \in R_p \cup R_i} \exists H_1, \ldots, H_k, X =^?_{\lambda\sigma} (\mathtt{r}\ H_1 \ldots H_k),$
falls X von atomarem Typ und nicht gelöst,
$H_1, \ldots, H_k$ Variablen $\notin TVar(P)$;
$R_p = \{r \mid 1 \leq r \leq p, (\mathtt{r}\ H_1 \ldots H_k)$ hat passenden Typ $\}$,
$R_i = \{m - n + p\}$, falls $m \geq n + 1$, sonst $\emptyset$

Die Regeln für **Unif**

Replace $P \wedge X =^?_{\lambda\sigma} a$
$\rightarrow \{X \mapsto a\}(P) \wedge X =^?_{\lambda\sigma} a$,
falls $X \in TVar(P), X \notin TVar(a)$ und
$a \in \mathcal{X} \Rightarrow a \in TVar(P)$
(Lösungen für X im Rest-System einsetzen)

Normalize $P \wedge a =^?_{\lambda\sigma} b$
$\rightarrow P \wedge a' =^?_{\lambda\sigma} b'$,
falls a oder b nicht in langer Normalform,
wobei x' die lange Normalform von x ist,
falls x keine gelöste Variable; sonst x.

(*lange Normalform*: η-lange Normalform; wenn a vom Typ $A_1 \rightarrow \ldots \rightarrow A_n \rightarrow B$ ist (Curry-Form von $A_1 \times \ldots \times A_n \rightarrow B$), dann beginnt die η-lange Form mit n Lambdas)

Die Regeln für **Back**

Anti-Exp-λ P
$\rightarrow \exists Y\ (P \wedge X =^?_{\lambda\sigma} (Y[\uparrow]\ 1))$,
falls $X \in Var(P), Y \in \mathcal{X}, Y \notin Var(P)$

Anti-Dec-λ $P \wedge a =^?_{\lambda\sigma} b$
$\rightarrow P \wedge \lambda a =^?_{\lambda\sigma} \lambda b$

Normalize wie in **Unif**

Beispiel

$\lambda y.(X\ a) =^?_{\beta\eta} \lambda y.a$ (λ-Kalkül)
$\rightarrow_{tr} (\lambda\ (X\ 2)) =^?_{\beta\eta} (\lambda\ 2)$ (de Bruijn)
$\rightarrow_{\text{Pre Cooking}} (\lambda\ (X[\uparrow]\ 2)) =^?_{\lambda\sigma} (\lambda\ 2)$ ($\lambda\sigma$-Kalkül)
$\rightarrow_{\textbf{Dec-}\boldsymbol{\lambda}} (X[\uparrow]\ 2) =^?_{\lambda\sigma} 2$
$\rightarrow_{\textbf{Exp-}\boldsymbol{\lambda}}$
$\exists Y\ (((X[\uparrow]\ 2) =^?_{\lambda\sigma} 2) \wedge (X =^?_{\lambda\sigma} \lambda Y))$
$\rightarrow_{\textbf{Replace}}$
$\exists Y\ ((((\lambda Y)[\uparrow]\ 2) =^?_{\lambda\sigma} 2) \wedge (X =^?_{\lambda\sigma} \lambda Y))$
$\rightarrow_{\textbf{Normalize}}$
$\exists Y\ ((Y[2.\uparrow] =^?_{\lambda\sigma} 2) \wedge (X =^?_{\lambda\sigma} \lambda Y))$
$\rightarrow_{\textbf{Exp-app}}$
$\exists Y\ ((Y[2.\uparrow] =^?_{\lambda\sigma} 2) \wedge (X =^?_{\lambda\sigma} \lambda Y)) \wedge (Y = 1))$
$\vee\ \exists Y\ ((Y[2.\uparrow] =^?_{\lambda\sigma} 2) \wedge (X =^?_{\lambda\sigma} \lambda Y)) \wedge (Y = 2))$
$\rightarrow_{\textbf{Replace}}$
$((1[2.\uparrow] =^?_{\lambda\sigma} 2) \wedge (X =^?_{\lambda\sigma} \lambda 1))$
$\vee((2[2.\uparrow] =^?_{\lambda\sigma} 2) \wedge (X =^?_{\lambda\sigma} \lambda 2))$
$\rightarrow_{\textbf{Normalize}}$
$(X =^?_{\lambda\sigma} \lambda 1) \vee (X =^?_{\lambda\sigma} \lambda 2)$.

beide Probleme in gelöster Form
$\rightarrow$ Lösungen $\lambda x.x$ und $\lambda x.a$

- Unifikationsproblem höherer Ordnung zurückgeführt auf E-Unifikationsproblem erster Ordnung
- Explizite Substitutionen internalisieren $\beta\eta$-Reduktion
  $\rightarrow$ Konzentration auf Unifikationsvariablen
- Einfachere Implementierung eines Unif.-Algorithmus, da komplizierte Behandlung des β-Reduktionsprozeß wegfällt
- Anwendung: Higher Order Pattern (entscheidbare Teilklasse des λ-Kalküls)