Tutorium 07: Prolog-Basics

Paul Brinkmeier

02. Dezember 2019

Tutorium Programmierparadigmen am KIT

Heutiges Programm

Programm

- ÜBs 5 und 6
- Typisierter λ-Kalkül
- Einführung in Prolog
- Aufgaben zu Prolog

Fragen von letzter Woche

- Zwischenschritte beim SKI-Kalkül müssen *nicht* angegeben werden

Fragen von letzter Woche

- Zwischenschritte beim SKI-Kalkül müssen *nicht* angegeben werden
- Nicht mehr relevante Aufgaben $\rightsquigarrow$ kommt auf Vorlesung an
  - Alles, was in der Vorlesung angesprochen wird, ist relevant

Fragen von letzter Woche

- Zwischenschritte beim SKI-Kalkül müssen *nicht* angegeben werden
- Nicht mehr relevante Aufgaben $\rightsquigarrow$ kommt auf Vorlesung an
    - Alles, was in der Vorlesung angesprochen wird, ist relevant
    - Bis auf Z-Folien

Fragen von letzter Woche

- Zwischenschritte beim SKI-Kalkül müssen *nicht* angegeben werden
- Nicht mehr relevante Aufgaben $\rightsquigarrow$ kommt auf Vorlesung an
  - Alles, was in der Vorlesung angesprochen wird, ist relevant
  - Bis auf Z-Folien
- Freie Variablen sind relevant für α-Äquivalenz
  - Kommt aber so in Klausuraufgaben nicht vor

Fragen von letzter Woche

- Zwischenschritte beim SKI-Kalkül müssen *nicht* angegeben werden
- Nicht mehr relevante Aufgaben $\rightsquigarrow$ kommt auf Vorlesung an
  - Alles, was in der Vorlesung angesprochen wird, ist relevant
  - Bis auf Z-Folien
- Freie Variablen sind relevant für α-Äquivalenz
  - Kommt aber so in Klausuraufgaben nicht vor
- „Standardbibliothek“ für λ-Kalkül
  - Alles von den ÜBs
  - Sonstiges nur mit Quellenangabe
    - Bspw. ProPa Folie X, ProPa Klausur SS18 Seite Y

Übungsblatt 5

1.5 — β-Reduktion

Gegeben war:

$$(\lambda a.a)\ (\lambda b.b)\ ((\lambda c.c)\ ((\lambda d.d)\ (\lambda e.e)\ (\lambda f.f)))\ \lambda g.g\ ((\lambda h.h)\ (\lambda i.i))$$

- Vorgehensweise: Redexe markieren, Termliste durchsuchen
- Häufigster Fehler: $((\lambda h.h)\ (\lambda i.i))$ kann man in $\lambda g.g$ *nicht* reinziehen (Funktionsaufrufe sind linksassoziativ)

4 — Church-Paare

$$\begin{aligned} \mathit{pair} &= \lambda a.\lambda b.\lambda f.f\ a\ b \\ \mathit{pair}\ c_{42}\ c_{100} &\overset{2}{\Longrightarrow} \lambda f.f\ c_{42}\ c_{100} \end{aligned}$$

- $\leadsto$ *Destructuring*/*Pattern Matching*/Fallunterscheidung durch Aufruf einer Funktion mit den Elementen des Tupels
- Wie bei den Listen von letzter Woche

$$\begin{aligned} \mathit{fst} &= \lambda p.p\ (\lambda a.\lambda b.a) \\ \mathit{snd} &= \lambda p.p\ (\lambda a.\lambda b.b) \end{aligned}$$

- *fst*/*snd* ruft Tupel mit Funktion auf, die nur ihr erstes/zweites Argument zurückgibt

4 — Church-Paare

Geben Sie *next* an, sodass $next\ (n, m) = (m, m + 1)$.

Geben Sie *next* an, sodass *next* $(n, m) = (m, m + 1)$.

$$next = \lambda p.(p\ (\lambda n.\lambda m.pair\ m\ (succ\ m)))$$

In Haskell: `next (n, m) = (m, succ m)`

4 — Church-Paare

Geben Sie *next* an, sodass $next\ (n, m) = (m, m + 1)$.

$$next = \lambda p.(p\ (\lambda n.\lambda m.pair\ m\ (succ\ m)))$$

In Haskell: `next (n, m) = (m, succ m)`

$$next = \lambda p.pair\ (snd\ p)\ (succ\ (snd\ p))$$

In Haskell: `next p = (snd p, succ (snd p))`

$$\begin{aligned} \mathit{pred} &= \lambda n.\mathit{fst}\ (n\ \mathit{next}\ (\mathit{pair}\ c_0\ c_0)) \\ \mathit{sub} &= \lambda m.\lambda n.n\ \mathit{pred}\ m \end{aligned}$$

- *next* wird n-mal auf $(0, 0)$ angewendet, erhöht immer das zweite Argument und speichert eine Kopie davon $\rightsquigarrow$ im letzten Schritt ist das Tupel $(n-1, n)$, *fst* liefert also $n-1$
- *sub* nimmt einfach n-mal den Vorgänger von m

Übungsblatt 6

Typsystem für λ-Terme

Für Variablen t: $$\frac{\Gamma(t) = \sigma \qquad \sigma \succeq \tau}{\Gamma \vdash t : \tau}\textsc{Var}$$

Für Aufrufe $f\ a$: $$\frac{\Gamma \vdash f : \phi \to \alpha \qquad \Gamma \vdash a : \phi}{\Gamma \vdash f\ a : \alpha}\textsc{App}$$

Für Funktionsterme $\lambda p.b$: $$\frac{\Gamma, p : \pi \vdash b : \rho}{\Gamma \vdash \lambda p.b : \pi \to \rho}\textsc{Abs}$$

- Γ ist ein Typkontext
- D.h. *Liste* von Variable/Typ-Tupeln
- Bspw. $x : \texttt{char}, y : \texttt{int}$
- **Vorsicht**: Γ ist keine Menge, denn Γ hat eine Reihenfolge!

Var

- „Der Typkontext Γ enthält einen Typ σ für t“
- „σ kann mit τ instanziiert werden“

$$\frac{\boxed{\Gamma(t) = \sigma} \quad \boxed{\sigma \succeq \tau}}{\boxed{\Gamma \vdash t : \tau}}\ \textsc{Var}$$

- dann gilt:
- „Variable t hat im Kontext Γ den Typ τ“
- $\sigma \succeq \tau \rightsquigarrow$ „σ hat τs Struktur und ist (mind.) allgemeiner“
  - $\text{int} \to \text{int} \succeq \text{int} \to \text{int}$
  - $\forall\alpha.\alpha \to \alpha \succeq \text{int} \to \text{int}$
  - $\alpha \to \alpha \not\succeq \text{int} \to \text{int}$
  - $\text{int} \to \text{int} \not\succeq \forall\alpha.\alpha \to \alpha$

App

- „f ist im Kontext Γ eine Funktion, die ϕs auf αs abbildet“
- „a ist im Kontext Γ ein Term des Typs ϕ“

$$\frac{\boxed{\Gamma \vdash f : \phi \to \alpha} \qquad \boxed{\Gamma \vdash a : \phi}}{\boxed{\Gamma \vdash f\ a : \alpha}}\ \textsc{App}$$

- dann gilt:
- „a eingesetzt in f ergibt einen Term des Typs α“

Abs

- „Damit *b* als Funktion von *p* typisierbar ist…“
- „… müssen wir den Typ von *p* in den Kontext einfügen“

$$\frac{\boxed{\Gamma, p : \pi} \; \boxed{\vdash b : \rho}}{\boxed{\Gamma \vdash \lambda p.b : \pi \to \rho}}\ \textsc{Abs}$$

- dann gilt:
- „$\lambda p.b$ ist eine Funktion, die πs auf ρs abbildet“

3 — λ-Terme und ihre Typen

Geben sie für jeden Typ der unten stehenden Tabelle *all* die Terme aus A - F an, die diesen Typ haben können.

- Zeigt, dass D den vierten Typ haben kann.
- $D = \lambda x.\lambda y.y\ (x\ y)$
- Typ: $((\alpha \to \beta) \to \alpha) \to (\alpha \to \beta) \to \beta$

3 — λ-Terme und ihre Typen

Geben sie für jeden Typ der unten stehenden Tabelle *all* die Terme aus *A* - *F* an, die diesen Typ haben können.

- Zeigt, dass *D* den vierten Typ haben kann.
- $D = \lambda x.\lambda y.y\ (x\ y)$
- Typ: $((\alpha \to \beta) \to \alpha) \to (\alpha \to \beta) \to \beta$
- Ansatz (unten links: leerer Kontext):

$$\frac{...\vdash ...}{\vdash \lambda x.\lambda y.y\ (x\ y) : ((\alpha \to \beta) \to \alpha) \to (\alpha \to \beta) \to \beta}\textsc{Abs}$$

4 — Typ-Prüfung

- Sehr gute Übungsaufgabe zum Thema
- Abgegeben von ca. 3 Leuten :(
- Wenn ihr mir die Aufgabe aufs nächste Blatt (oder per Mail schreibt), korrigiere ich sie euch noch zur Übung

Einführung in Prolog

Aufgaben zu Prolog