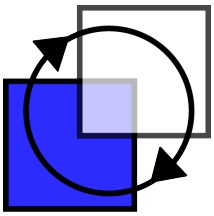
CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Prof. Bernd Finkbeiner, Ph.D.
Jana Hofmann, M.Sc.
Reactive Systems Group



# Programmierung 1 (WS 2020/21)
# Zusatzerklärung zum Thema
Typen polymorpher Prozeduren

## 1 Von der Prozedur zum Typ

Eine *typische Aufgabenstellung* lautet:

Geben Sie zu der folgenden polymorphen Prozedur ein Typschema an:

```
1 fun f a b c = a (c b)
```

### 1.1 Allgemeines Schema

Das Schema besteht aus den folgenden Schritten:

(a) Bereinigen Sie den Ausdruck, falls notwendig.

(b) Schreiben Sie untereinander alle vorkommenden Bezeichner auf.

(c) Klammern Sie Ihren Ausdruck vollständig.

(d) Jetzt kommt die rechte Seite. Bestimmen Sie nach und nach die Typen der Bezeichner.

*Tipp*: Gehen Sie stets von innen nach außen vor. Bestimmen Sie für einen Prozeduraufruf stets zuerst die Typen der Argumente, dann erst den Typ der Prozedur. Beginnen Sie mit Typen, die kein Prozedurtyp sein können.

Steht ein = zwischen zwei Bezeichnern, so müssen diese den gleichen Typ haben – und bei diesem Typ muss es sich um einen Typ mit Gleichheit handeln. Steht ein Bezeichner in der Bedingung des Konditionals, so hat dieser Bezeichner den Typ `bool`.

Tritt ein Bezeichner mehrfach auf, so müssen Sie diesen für jedes Auftreten getrennt betrachten und eventuell bisherige „Typvermutungen“ – auch in fremden Typangaben – verbessern.

(e) Bestimmen Sie den Ergebnistyp.

(f) Tragen Sie Ihre bisherigen Erkenntnisse zusammen. Das allgemeine Schema lautet:

$$\forall \textit{ Typvariablen} : \textit{Typ des ersten Arguments} \to \ldots \to \textit{Typ des letzten Arguments} \to \textit{Ergebnistyp}$$

### 1.2 Einfaches Beispiel

Geben Sie zu der folgenden polymorphen Prozedur ein Typschema an.

```
1 fun f a b c = a (c b)
```

(a) Der Ausdruck ist bereits bereinigt, hier ist also nichts zu tun.

(b) Aufschreiben aller Bezeichner.

```
1 a : ...
2 b : ...
3 c : ...
4 f : ...
```

(c) Die Klammerung bleibt.

(d) Wir gehen wie beschrieben von innen nach außen vor.

(i) `b` ist ganz innen in der Klammer und wird auf kein Argument angewendet – wir bestimmen den Typ von `b` als $\beta$.

(ii) `c` ist die nächstinnere Variable. `c` wird auf `b` angewendet und muss damit ein Prozedurtyp sein, der ein Argument des Typs von `b` nimmt ( $\beta$ ) und etwas von einem nicht näher bestimmten Typ (nennen wir ihn $\gamma$ ) zurückgibt. Damit gilt `c` : $\beta \to \gamma$.

(iii) `a` ist die letzte verbleibende Variable. `a` wird auf `(c b)` angewendet, also auf etwas vom Typ $\gamma$. Der Ergebnistyp kann nicht näher bestimmt werden (nennen wir ihn $\alpha$). Damit gilt `a` : $\gamma \to \alpha$.

(e) Der Ergebnistyp ist $\alpha$.

(f) Insgesamt hat `f` also den Typ $\forall \alpha\, \beta\, \gamma.\ (\gamma \to \alpha) \to \beta \to (\beta \to \gamma) \to \alpha$.

### 1.3 Fortgeschrittenes Beispiel

Geben Sie zu der folgenden polymorphen Prozedur ein Typschema an:

```
1 fun f a (b, c) d = a (b, c d a)
```

(a) Der Ausdruck ist bereits bereinigt, hier ist also nichts zu tun.

(b) Aufschreiben aller Bezeichner.

```
1 a: ...
2 b: ...
3 c: ...
4 d: ...
5 f: ...
```

(c) Da die Prozeduranwendung am stärksten klammert, lautet der Ausdruck vollständig geklammert:

```
1 fun f a (b, c) d = a (b, (c d) a)
```

(d) Betrachten wir die rechte Seite:

(i) `d` ist an der innersten Stelle und wird auf keinen anderen Wert angewendet. Wir bestimmen den Typ von `d` als $\delta$.

(ii) `b` wird ebenfalls auf kein Argument angewendet, es sind keine weiteren Informationen vorhanden. Wir bestimmen den Typ von `b` als $\beta$.

(iii) Das `a` innerhalb der Klammer ist der nächstinnere Bezeichner, der keine Auswertung eines Argumentes benötigt. Wir bestimmen den Typ von `a` vorläufig als $\alpha$.

(iv) `c` nimmt etwas vom Typ $\delta$, dann etwas vom Typ $\alpha$. Damit gilt `c` : $\delta \to \alpha \to \gamma$.

(v) Wir betrachten das äußere Auftreten von `a`. Der Typ von `a` muss nun noch einmal angepasst werden, da `a` eine Prozedur ist, die das Argument `(b, (c d) a)` nimmt. `a` nimmt also ein Tupel des Typs $\beta$ `*` $\gamma$ und liefert dann etwas eines unbestimmten Typs – nennen wir ihn $\varepsilon$. Wir korrigieren den bisherigen Typ von `a` zu `a` : $\beta$ `*` $\gamma \to \varepsilon$. Damit aber müssen wir den Typ von `a` auch in `c` anpassen: `c` : $\delta \to (\beta$ `*` $\gamma \to \varepsilon) \to \gamma$.

(vi) Wir haben keine weiteren zu betrachtenden Abhängigkeiten und sind damit fertig.

(e) Der Ergebnistyp ist $\varepsilon$.

(f) Insgesamt lautet der Typ von `f`:
$\forall \beta\, \gamma\, \delta\, \varepsilon : (\beta$ `*` $\gamma \to \varepsilon) \to (\beta$ `*` $(\delta \to (\beta$ `*` $\gamma \to \varepsilon) \to \gamma)) \to \delta \to \varepsilon$

### 1.4 Jetzt sind Sie dran!

Bestimmen Sie das Typschema der folgenden Prozeduren:

(a) `fun f a b = a b b`

(b) `fun f a b c = a b c`

(c) `fun f a b c = a (b c)`

(d) `fun f a (b, c) = (a b, a c, a b c)`

(e) `fun f (a, b, c) d = if c then (a + 4) = b else d (true)`

(f) `fun f a (b, c) d = a (b (c d) c a)`

(g) `fun f a (b, c) d (e, f) = a (f e) ((a b) (c d))`

Überprüfen Sie Ihre Lösungen mithilfe von SOSML und fragen Sie bei Problemen eine Tutorin in der Office Hour.

## 2 Vom Typ zur Prozedur

Eine *typische Aufgabenstellung* lautet:

Deklarieren Sie eine polymorphe Prozedur, die das angegebenen Typschemata besitzt:

$$\forall\alpha\,\beta\,\gamma : (\ \alpha\ \to\ \alpha\ \to\ \beta\ )\ \to\ \alpha\ \to\ \beta$$

### 2.1 Allgemeines Schema

Das Schema besteht aus den folgenden Schritten:

(a) Entfernen Sie unnötige Klammern bei Pfeilen (Erinnerung: $\to$ klammert rechts!). So soll beispielsweise $\alpha \to (\ \beta\ \to\ \gamma\ )$ umgeformt werden zu $\alpha\ \to\ \beta\ \to\ \gamma$.

(b) Beginnen Sie je nach Aufgabenstellung mit einer Prozedurdeklaration: `fun` $(\ \alpha\ ,\ \beta\ ,\ \ldots)$ `f ...` oder Abstraktionsdeklaration. Sind explizit alle Typangaben gefordert, so müssen Sie auch alle Typvariablen vor dem Namen der Prozedur einführen ($(\ \alpha\ ,\ \beta\ ,\ \ldots)$).

(c) Sammeln Sie alles bis vor den letzten Pfeil. Das heißt: Führen Sie für jedes weitere Argument einen neuen Bezeichner ein und typen Sie die Argumente je nach Aufgabenstellung explizit (oder verzichten auf Typangaben).

**Hinweis:** Bei Tupeltypen als Argument empfiehlt sich ein Einführen von mehreren Bezeichnern (kartesisches Argumentmuster), da dadurch nachher Projektionen vermieden werden.

**Achtung:** Eine Prozedur erhält immer nur einen einzigen Bezeichner.

(d) Jetzt beginnt die eigentliche Arbeit: Mit Hilfe der eingesammelten Typen ist es nun Ihre Aufgabe, etwas von dem Typen zu konstruieren, der rechts vom letzten Pfeil steht.

Unterscheiden Sie dazu zwischen zusammengesetzten Tupel-Typen und nicht weiter zerlegbaren Typen. Wenden Sie für zusammengesetzte Typen das folgende Schema mehrfach an und verwenden Sie kartesische Muster, um diese zusammenzusetzen.

Haben Sie einen Typ nicht explizit gegeben, so schauen Sie nach, mit Hilfe welcher Prozeduren Sie diesen Typ erhalten können (In welchen Prozedurtypen steht dieser Typ ganz rechts?). Versuchen Sie wiederum, die benötigten Argumenttypen zusammenzusetzen.

### 2.2 Einfaches Beispiel

Deklarieren Sie eine polymorphe Prozedur, die das angegebene Typschema besitzt. Geben Sie alle Typen explizit an.

$$\forall\alpha\,\beta\,\gamma : (\ \gamma\ \to\ \alpha\ )\ \to\ \beta\ \to\ ((\ \beta\ \to\ \gamma\ )\ \to\ \alpha\ )$$

(a) Wir ändern die Klammerung zu $(\ \gamma\ \to\ \alpha\ )\ \to\ \beta\ \to\ (\ \beta\ \to\ \gamma\ )\ \to\ \alpha$

(b) Die Aufgabenstellung verlangt explizite Typangaben. Wir beginnen also mit `fun` $(\ \alpha\ ,\ \beta\ ,\ \gamma\ )$ `f ...`

(c) Nach dem obigen Schema erhalten wir:

```
1 fun (α, β, γ) f (a : γ → α) (b : β) (c : β → γ) = ...
```

(d) Unser Ziel ist es, etwas vom Typ $\alpha$ zu erhalten. Wir haben vom Typ $\alpha$ nichts Explizites gegeben, deshalb müssen wir dieses selbst mit einer Prozeduranwendung konstruieren.

`a` ist die einzige Prozedur, die uns etwas vom Typ $\alpha$ zurückgibt. Sie möchte etwas vom Typ $\gamma$ übergeben bekommen.

```
1 fun (α, β, γ) f (a : γ → α) (b : β) (c : β → γ) = a (* etwas vom Typ γ *)
```

Vom Typ $\gamma$ haben wir ebenfalls nichts Explizites, können aber die Prozedur `c` benutzen, um etwas vom Typ $\gamma$ zu erhalten. Diese möchte dafür etwas vom Typ $\beta$ übergeben bekommen.

```
fun (α, β, γ) f (a : γ → α) (b : β) (c : β → γ) = a (c (* etwas vom Typ β *))
```

`b` ist vom Typ $\beta$ – wir können dies also `c` als Argument übergeben!

```
fun (α, β, γ) f (a : γ → α) (b : β) (c : β → γ) = a (c b)
```

Damit sind wir fertig.

### 2.3 Fortgeschrittenes Beispiel

Deklarieren Sie eine polymorphe Prozedur, die das angegebene Typschema besitzt. Verzichten Sie auf explizite Typangaben.

$$\forall\alpha\,\beta\,\gamma : (\alpha * \alpha \to \beta) \to (\beta \to \alpha \to \gamma) \to \alpha \to (\gamma * \beta)$$

(a) Die Klammerung bleibt bestehen.

(b) Die Aufgabenstellung verlangt von uns, auf explizite Typen zu verzichten. Wir beginnen mit `fun f ...`

(c) Nach dem obigen Schema erhalten wir:

```
fun f a b c = ...
```

(d) Unser Ziel ist es, etwas vom Typ $\gamma * \beta$ zu erhalten – wir brauchen also ein Tupel:

```
fun f a b c = ((* etwas vom Typ γ *), (* etwas vom Typ β *))
```

Wir beginnen, etwas vom Typ $\gamma$ zu bauen. Da wir nichts Explizites von diesem Typen gegeben haben, müssen wir ihn selbst mit einer Prozeduranwendung konstruieren.

`b` ist die einzige Prozedur, die uns etwas vom Typ $\gamma$ zurückgibt. Sie möchte etwas vom Typ $\beta$ und etwas vom Typ $\alpha$ übergeben bekommen.

```
fun f a b c = (b (* etwas vom Typ β *) (* etwas vom Typ α *), (* etwas vom Typ β *))
```

Etwas vom Typ $\alpha$ haben wir explizit mit `c` und etwas vom Typ $\beta$ erhalten wir mit `a`. `a` möchte ein Tupel vom Typ $\alpha * \alpha$ übergeben bekommen.

```
fun f a b c = (b (a ((* etwas vom Typ α *), (* etwas vom Typ α *))) c, (* etwas vom Typ β *))
```

Dieses Tupel vom Typ $\alpha * \alpha$ können wir uns mit `c` konstruieren.

```
fun f a b c = (b (a (c, c)) c, (* etwas vom Typ β *))
```

Für die zweite Komponente möchten wir etwas vom Typ $\beta$ haben. Im ersten Teil haben wir schon festgestellt, dass `a (c, c)` vom passenden Typ ist und ergänzen zu:

```
fun f a b c = (b (a (c, c)) c, a (c, c))
```

Damit sind wir fertig.

### 2.4 Jetzt sind Sie dran!

Deklarieren Sie polymorphe Prozeduren, die die angegebenen Typschemata besitzen. Verzichten Sie auf explizite Typangaben.

(a) $\forall\alpha\,\beta.\ (\alpha \to \alpha \to \beta) \to \alpha \to \beta$

(b) $\forall\alpha\,\beta\,\gamma\,\delta.\ \alpha * \beta * \gamma \to (\gamma * \alpha \to \delta) \to \delta$

(c) $\forall\alpha\,\beta\,\gamma.\ (\alpha * \alpha \to \beta) \to (\beta \to \alpha \to \gamma) \to \alpha \to \gamma$

(d) $\forall\alpha\,\beta\,\gamma\,\delta.\ (\alpha \to \beta \to \gamma \to \delta) \to \alpha \to (\alpha \to \beta) \to (\beta \to \gamma) \to \delta$

Überprüfen Sie Ihre Lösungen mithilfe von SOSML und fragen Sie bei Problemen eine Tutorin in der Office Hour.

### 2.5 Typen mit Gleichheit und konkrete Typen

Wenn Sie laut Aufgabenstellung auf explizite Typangaben verzichten müssen, und Typen mit Gleichheit (beispielsweise `'α`) oder konkrete Typen (beispielsweise `int`) haben, müssen Sie in die Trickkiste greifen: Beispielsweise können Sie erzwingen, dass eine Variable `a` den Typ `int` hat, in dem Sie `a + 1` berechnen. Dass eine Variable `b` einen Typ mit Gleichheit hat, können Sie erzwingen, in dem Sie `b = b` berechnen. Wenn Sie das Ergebnis des Vergleichs (vom Typ `bool`) nicht benötigen, können Sie den Vergleich in einem `let`-Ausdruck „verstecken“.

#### 2.5.1 Beispiel

Versuchen wir einmal, eine Prozedur des Typs des folgenden Typschemas ohne explizite Typangaben zu erzeugen:

$$\forall\text{'}\alpha\,\beta : \texttt{('}\alpha\ \texttt{*}\ \texttt{'}\alpha\ \to\ \beta\texttt{)}\ \to\ \texttt{(}\beta\ \to\ \texttt{'}\alpha\ \to\ \texttt{int)}\ \to\ \texttt{'}\alpha\ \to\ \texttt{(int *}\ \beta\texttt{)}$$

Dieser Typ entspricht dem Typ des vorherigen Beispiels, nur muss jetzt $\alpha$ ein Typ mit Geichheit sein, und statt $\gamma$ ist der Typ `int` verlangt.

Wie können wir unsere alte Lösung – `fun f a b c = (b (a (c, c)) c, a (c, c))` – modifizieren?

- Aus $\gamma$ soll `int` werden. Dazu addieren wir an einer Stelle, an der wir den Typ $\gamma$ vor uns haben, 1:

```
1 fun f a b c = ((b (a (c, c)) c) + 1, a (c, c))
```

- Wir müssen erzwingen, dass $\alpha$ ein Typ mit Gleichheit ist. Mit `c` haben wir eine Variable dieses Typs, und so können wir einfach `c = c` berechnen. Da wir das Ergebnis (des Typs `bool`) nicht brauchen, verwenden wir einen `let`-Ausdruck:

```
1 fun f a b c = let
2                   val tmp = c = c
3               in
4                   ((b (a (c, c)) c) + 1, a (c, c))
5               end
```

Damit sind wir fertig.

#### 2.5.2 Jetzt sind Sie dran!

Deklarieren Sie polymorphe Prozeduren, die die angegebenen Typschemata besitzen. Verzichten Sie auf explizite Typangaben.

(a) $\forall\alpha\,\beta\,\gamma : \alpha\ \texttt{*}\ \texttt{(}\alpha\ \texttt{*}\ \beta\ \to\ \texttt{'}\gamma\texttt{)}\ \to\ \beta\ \texttt{*}\ \texttt{(}\beta\ \to\ \texttt{'}\gamma\texttt{)}\ \to\ \texttt{bool}$

(b) $\forall\alpha\,\beta\,\gamma : \texttt{(int}\ \to\ \texttt{'}\alpha\texttt{)}\ \to\ \texttt{int}\ \to\ \texttt{((int}\ \to\ \texttt{'}\alpha\texttt{)}\ \to\ \texttt{int}\ \to\ \beta\texttt{)}\ \to\ \texttt{'}\alpha\ \texttt{*}\ \beta\ \texttt{*}\ \texttt{int}$

Überprüfen Sie Ihre Lösungen mithilfe von SOSML und fragen Sie bei Problemen eine Tutorin in der Office Hour.