verfasst von Eyad Alkassar

Seminar zum pi-Kalkül betreut von Andreas Rossberg

TYPEN UND OBJEKTE

im Pi-Kalkül

INHALT

WARUM TYPEN?

Motivation

- Vermeidung von Laufzeitfehlern
- besseres Verständnis von Verhalten von Programmen
- Dokumentation von Quellcode
- Optimierung von Compilern
- …

WARUM TYPEN?

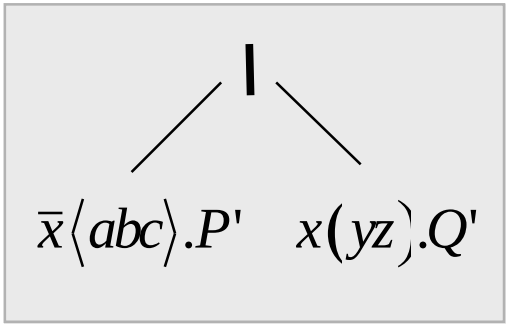
Motivation

- Vermeidung von Laufzeitfehlern
- besseres Verständnis von Verhalten von Programmen
- Dokumentation von Quellcode
- Optimierung von Compilern
- …

WARUM TYPEN?

Beispiele

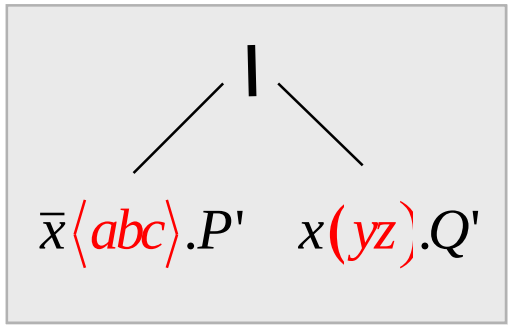
- **Vermeidung von Laufzeitfehler**
  - Was sind Laufzeitfehler im pi-Kalkül?
  - Bisher haben wir noch keine Übergänge gesehen, daher betrachten Reaktionen
  - Beispiel im polyadischen Kalkül



WARUM TYPEN?

Beispiele

- **Vermeidung von Laufzeitfehler**
  - Was sind Laufzeitfehler im pi-Kalkül?
  - Bisher haben wir noch keine Übergänge gesehen, daher betrachten Reaktionen
  - Beispiel im polyadischen Kalkül



$\triangleright$ *Fehler*

WARUM TYPEN?

Beispiele

- **Vermeidung von Laufzeitfehler**
  - Was sind Laufzeitfehler im pi-Kalkül?
  - Fügen Fehlerregel zu Reaktionsregeln hinzu:

$$\textit{FEHLER}: \frac{|\vec{y}| \neq |\vec{x}|}{(\overline{x}\langle \vec{y} \rangle.P' \ldots + N) \mid (x(\vec{z}).Q' \ldots + M) \longrightarrow \textit{fehler}}$$

$$\textit{PARFEHLER}: P \mid \textit{fehler} \longrightarrow \textit{fehler}$$

$$\textit{BINDFEHLER}: \textit{new } x \textit{ fehler} \longrightarrow \textit{fehler}$$

$$\textit{SUMFEHLER}: M + \textit{fehler} \longrightarrow \textit{fehler}$$

WARUM TYPEN?

Beispiele

- **Wir formulieren diese Eigenschaft ein wenig anders:**

  Eigenschaft PROGRESS: Wohlgetypte Terme des pi-Kalküls können in einem Schritt nicht zu *fehler* reagieren

Was bedeutet wohlgetypt?

WARUM TYPEN?

Beispiele

- **Besseres Verständnis von Programmverhalten**
  - *Wohlgetypt* als Eigenschaft von einem Ausdruck, der während der Ausführung erhalten bleibt
  - Und Einschränkung der möglichen Reaktionen

WARUM TYPEN?

Beispiele

- **Kontrollieren das Verhalten von Prozessen:**
  - Für ein einfaches Systems S und folgenden Eigenschaften

1. Es verwaltet x
2. Es ist x-vergesslich
3. Für jeden Subterm der Form z(y).Q ist Q y-vergesslich

  - gilt: wenn S→ S‘, dann hat S‘ Eigenschaften 1-3

WARUM TYPEN?

Beispiele

[Wright, Andrew K. und Matthias Felleisen 1994]

- **Wir formulieren die Eigenschaft anders:**

Eigenschaft PRESERVATION: Wohlgetypt ist stabil unter Reaktion (und struktureller Kongruenz).

+

Eigenschaft PROGRESS: Wohlgetypte Terme des pi-Kalküls können in einem Schritt nicht zu *fehler* reagieren

=

Eigenschaft SOUNDNESS: Wohlgetypte Terme des pi-Kalküls reagieren nie zu *fehler*.

Was bedeutet wohlgetypt?

EINFÜHRUNG VON TYPEN

Erste Version

- **Namen können als Daten ausgetauscht werden**
  - Kategorisierung dieser Namen in Typen

Menge aller Typen sei Σ

$\sigma_1, \sigma_2 \in \Sigma$

$x : \sigma_1$ $u : \sigma_1$ $y : \sigma_2$ $v : \sigma_2$

  - Namensvektoren werden Typvektoren zugewiesen:

$\sigma_i \in \Sigma$

$\vec{x} : \vec{\sigma}$, *wenn* $\forall x_i \; x_i : \sigma_i$

EINFÜHRUNG VON TYPEN

Erste Version

- **Namen auch Kanäle für andere Namen**
  - Namen welchen Typs darf ein Name eines bestimmten Typs übermitteln?

$$new\ x : \sigma_1\ (x\langle uv\rangle | ...)$$

EINFÜHRUNG VON TYPEN

Erste Version

- Definition von **Typdisziplin**

- Definition **wohlgetypt**

EINFÜHRUNG VON TYPEN

Erste Version

- Definition von **Typdisziplin**

  Eine Typdisziplin zu einer Typmenge Σ

  ist eine Funktion $ob : \Sigma \xrightarrow{part} \Sigma^*$

- Definition **wohlgetypt**

EINFÜHRUNG VON TYPEN

Erste Version

- Definition von **Typdisziplin**

  Eine Typdisziplin zu einer Typmenge Σ

  ist eine Funktion $ob : \Sigma \xrightarrow{part} \Sigma^*$

- Definition **wohlgetypt**

  Ein Prozess heißt wohlgetypt unter ob,

  wenn für alle $x(\vec{y}).P$ oder $x\langle\vec{y}\rangle.P$ gilt :

  wenn $x : \sigma$ dann $\vec{y} : ob(\sigma)$

EINFÜHRUNG VON TYPEN

Erste Version

- Definition von **Typdisziplin**

  Eine Typdisziplin zu einer Typmenge Σ

  ist eine Funktion $ob : \Sigma \xrightarrow{part} \Sigma^*$

- Definition **wohlgetypt**

  Ein Prozess heißt wohlgetypt unter ob,

  wenn für alle $x(\vec{y}).P$ oder $x\langle\vec{y}\rangle.P$ gilt :

  wenn $x : \sigma$ dann $\vec{y} : ob(\sigma)$

EINFÜHRUNG VON TYPEN

BEISPIEL FÜR DEFINITIONEN

- **Beispiel: Typen von Bools**

$$\mathrm{True}(b) \doteq b(tf).\overline{t}$$

$$\mathrm{False}(b) \doteq b(tf).\overline{f}$$

EINFÜHRUNG VON TYPEN

BEISPIEL FÜR DEFINITIONEN

Beispiel: Typen von Bools

$$\mathrm{True}(b) \doteq b(tf).\overline{t}$$

$$\mathrm{False}(b) \doteq b(tf).\overline{f}$$

Typen und ihre Disziplin

b: BOOL

t: TRUE

f: FALSE

EINFÜHRUNG VON TYPEN

BEISPIEL FÜR DEFINITIONEN

Beispiel: Typen von Bools

$$\mathrm{True}(b) \doteq b(tf).\overline{t}$$

$$\mathrm{False}(b) \doteq b(tf).\overline{f}$$

Typen und ihre Disziplin

b: BOOL

t: TRUE

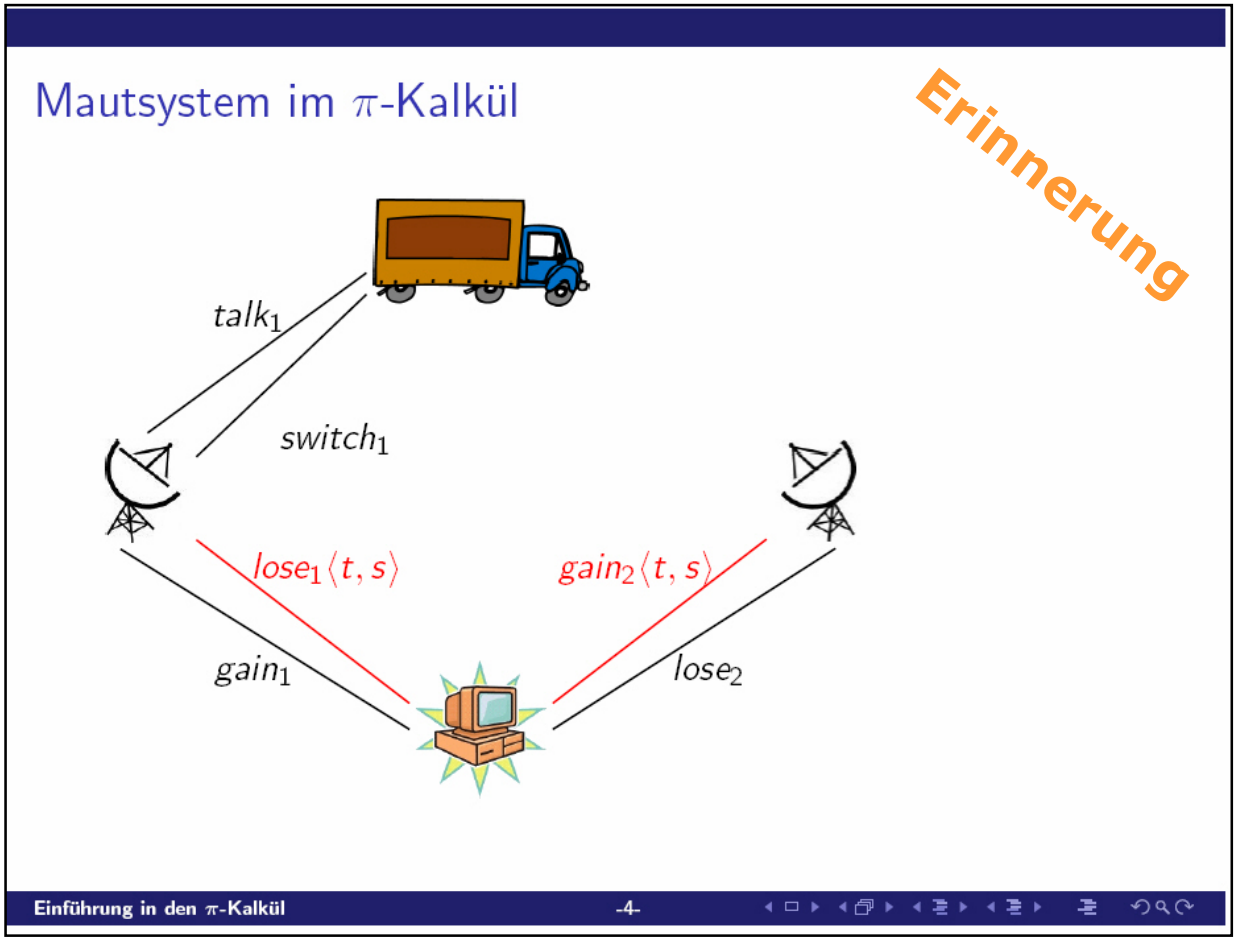
f: FALSE

$$\begin{cases} \mathrm{BOOL} \mapsto \mathrm{TRUE}, \mathrm{FALSE} \\ \mathrm{TRUE} \mapsto \varepsilon \\ \mathrm{FALSE} \mapsto \varepsilon \end{cases}$$

EINFÜHRUNG VON TYPEN

Beispiel für Definitionen

- **Beispiel: Mautsystem**



EINFÜHRUNG VON TYPEN

Beispiel für Definitionen

- **Beispiel: Mautsystem**

$$\mathrm{Truck}(\mathrm{talk}, \mathrm{switch}) \doteq \overline{\mathrm{talk}}.\mathrm{Truck}\langle \mathrm{talk}, \mathrm{switch}\rangle + \mathrm{switch}(t, s).\mathrm{Truck}\langle t, s\rangle$$

$$\mathrm{Trans}_1(\mathrm{talk}, \mathrm{switch}) \doteq \mathrm{talk}.\mathrm{Trans}_1\langle \mathrm{talk}, \mathrm{switch}\rangle + \mathrm{lose}_1(t,s).\overline{\mathrm{switch}}\langle t,s\rangle.\mathrm{gain}_1(t,s).\mathrm{Trans}_1\langle t,s\rangle$$

EINFÜHRUNG VON TYPEN

Beispiel für Definitionen

- **Beispiel: Mautsystem**

$$\text{Truck}(\text{talk}, \text{switch}) \doteq \overline{\text{talk}}.\text{Truck}\langle \text{talk}, \text{switch}\rangle + \text{switch}(t, s).\text{Truck}\langle t, s\rangle$$

$$\text{Trans}_1(\text{talk}, \text{switch}) \doteq \text{talk}.\text{Trans}_1\langle \text{talk}, \text{switch}\rangle + \text{lose}_1(t,s).\overline{\text{switch}}\langle t,s\rangle.\text{gain}_1(t,s).\text{Trans}_1\langle t,s\rangle$$

$$\Sigma = \{\text{TALK}, \text{SWITCH}, \text{GAIN}, \text{LOSE}\}$$

EINFÜHRUNG VON TYPEN

Beispiel für Definitionen

- **Beispiel: Mautsystem**

$$\text{Truck}(\text{talk}, \text{switch}) \doteq \overline{\text{talk}}.\text{Truck}\langle \text{talk}, \text{switch}\rangle + \text{switch}(t, s).\text{Truck}\langle t, s\rangle$$

$$\text{Trans}_1(\text{talk}, \text{switch}) \doteq \text{talk}.\text{Trans}_1\langle \text{talk}, \text{switch}\rangle + \text{lose}_1(t,s).\overline{\text{switch}}\langle t,s\rangle.\text{gain}_1(t,s).\text{Trans}_1\langle t,s\rangle$$

$$\Sigma = \{\text{TALK}, \text{SWITCH}, \text{GAIN}, \text{LOSE}\}$$

$$talk_i : \text{TALK},\ switch_i : \text{SWITCH},\ gain_i : \text{GAIN},\ lose_i : \text{LOSE}$$

EINFÜHRUNG VON TYPEN

Beispiel für Definitionen

- **Beispiel: Mautsystem**

$$\mathrm{Truck}(\mathrm{talk}, \mathrm{switch}) \doteq \overline{\mathrm{talk}}.\mathrm{Truck}\left\langle \mathrm{talk}, \mathrm{switch} \right\rangle + \mathrm{switch}(t, s).\mathrm{Truck}\left\langle t, s \right\rangle$$

$$\mathrm{Trans}_1(\mathrm{talk}, \mathrm{switch}) \doteq \mathrm{talk}.\mathrm{Trans}_1\left\langle \mathrm{talk}, \mathrm{switch} \right\rangle + \mathrm{lose}_1(t,s).\overline{\mathrm{switch}}\left\langle t,s \right\rangle.\mathrm{gain}_1(t,s).\mathrm{Trans}_1\left\langle t,s \right\rangle$$

$$\{\mathrm{TALK} \mapsto \varepsilon$$

EINFÜHRUNG VON TYPEN

Beispiel für Definitionen

- **Beispiel: Mautsystem**

$$\mathrm{Truck}(\mathrm{talk}, \mathrm{switch}) \doteq \overline{\mathrm{talk}}.\mathrm{Truck}\langle \mathrm{talk}, \mathrm{switch}\rangle + \mathrm{switch}(t, s).\mathrm{Truck}\langle t, s\rangle$$

$$\mathrm{Trans}_1(\mathrm{talk}, \mathrm{switch}) \doteq \mathrm{talk}.\mathrm{Trans}_1\langle \mathrm{talk}, \mathrm{switch}\rangle + \mathrm{lose}_1(t,s).\overline{\mathrm{switch}}\langle t,s\rangle.\mathrm{gain}_1(t,s).\mathrm{Trans}_1\langle t,s\rangle$$

$$\begin{cases} \mathrm{TALK} \mapsto \varepsilon \\ \mathrm{SWITCH} \mapsto \mathrm{TALK},\ \mathrm{SWITCH} \end{cases}$$

EINFÜHRUNG VON TYPEN

Beispiel für Definitionen

- **Beispiel: Mautsystem**

$$\mathrm{Truck}(\mathrm{talk}, \mathrm{switch}) \doteq \overline{\mathrm{talk}}.\mathrm{Truck}\langle \mathrm{talk}, \mathrm{switch}\rangle + \mathrm{switch}(t, s).\mathrm{Truck}\langle t, s\rangle$$

$$\mathrm{Trans}_1(\mathrm{talk}, \mathrm{switch}) \doteq \mathrm{talk}.\mathrm{Trans}_1\langle \mathrm{talk}, \mathrm{switch}\rangle + \mathrm{lose}_1(t,s).\overline{\mathrm{switch}}\langle t,s\rangle.\mathrm{gain}_1(t,s).\mathrm{Trans}_1\langle t,s\rangle$$

$$\begin{cases} \mathrm{TALK} \mapsto \varepsilon \\ \mathrm{SWITCH} \mapsto \mathrm{TALK},\ \mathrm{SWITCH} \\ \mathrm{GAIN} \mapsto \mathrm{TALK},\ \mathrm{SWITCH} \end{cases}$$

EINFÜHRUNG VON TYPEN

Beispiel für Definitionen

- **Beispiel: Mautsystem**

$$\text{Truck}(\text{talk}, \text{switch}) \doteq \overline{\text{talk}}.\text{Truck}\langle \text{talk}, \text{switch}\rangle + \text{switch}(t, s).\text{Truck}\langle t, s\rangle$$

$$\text{Trans}_1(\text{talk}, \text{switch}) \doteq \text{talk}.\text{Trans}_1\langle \text{talk}, \text{switch}\rangle + \text{lose}_1(t,s).\overline{\text{switch}}\langle t,s\rangle.\text{gain}_1(t,s).\text{Trans}_1\langle t,s\rangle$$

$$\begin{cases} \text{TALK} \mapsto \varepsilon \\ \text{SWITCH} \mapsto \text{TALK},\ \text{SWITCH} \\ \text{GAIN} \mapsto \text{TALK},\ \text{SWITCH} \\ \text{LOSE} \mapsto \text{TALK},\ \text{SWITCH} \end{cases}$$

EINFÜHRUNG VON TYPEN

Was bringt‘s?

- **Kontrollieren das Verhalten von Prozessen:**
  - Für ein einfaches Systems S und folgenden Eigenschaften

1. Es verwaltet x
2. Es ist x-vergesslich
3. Für jeden Subterm der Form z(y).Q ist Q y-vergesslich

  - gilt: wenn S→ S‘, dann hat S‘ Eigenschaften 1-3

EINFÜHRUNG VON TYPEN

Was bringt‘s?

- **Kontrollieren das Verhalten von Prozessen:**

  • Für ein **wohlgetyptes** einfaches Systems S mit **x:A** und folgenden Eigenschaften

  1. Es verwaltet x
  2. Es ist x-vergesslich
  3. Für jeden Subterm der Form z(y).Q ist Q y-vergesslich **falls y:A gilt.**

  • gilt, wenn S → S‘, dann hat S‘ Eigenschaften 1-3

EINFÜHRUNG VON TYPEN

Was bringt‘s?

- **Eigenschaft SOUNDNESS**: Wohlgetypte Terme des pi-Kalküls können nicht zu *fehler* reagieren. (ohne Beweis)

$$\bar{x}\langle abc\rangle.P' \mid x(yz).Q'$$

EINFÜHRUNG VON TYPEN

Was bringt‘s?

- **Eigenschaft SOUNDNESS**: Wohlgetypte Terme des pi-Kalküls können nicht zu *fehler* reagieren. (ohne Beweis)

$$\bar{x}\langle abc\rangle.P' \mid x(yz).Q'$$

$$ob(x) = \sigma_1\sigma_2\sigma_3$$

$$ob(x) = \sigma_1\sigma_2$$

$$\Rightarrow \text{nicht wohlgetypt}$$

EINFÜHRUNG VON TYPEN

Was bringt‘s?

- besseres Verständnis des pi-Kalküls
- wie wird der getypte pi-Kalkül zu CCS?

EINFÜHRUNG VON TYPEN

Was bringt‘s?

- **pi-Kalkül kontrollieren**
- **Nur noch Daten übertragen (CCS mit Value-Übertragung)**

  - Unterteilen Σ in Σ' und Δ :
    - Δ Menge aller Namen die als Link auftauchen dürfen
    - Σ' Menge elementarer Typen
  - $ob : \Delta \rightarrow (\Sigma')*$

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Betrachten noch mal die Bool Typen**

b: BOOL	$\left\{\begin{array}{l} \text{BOOL} \mapsto \text{TRUE,FALSE} \\ \text{TRUE} \mapsto \varepsilon \\ \text{FALSE} \mapsto \varepsilon \end{array}\right.$
t: TRUE	
f: FALSE	

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Betrachten noch mal die Bool Typen**

b: BOOL	$\left\{ \begin{array}{l} \text{BOOL} \mapsto \text{TRUE,FALSE} \\ \text{TRUE} \mapsto \varepsilon \\ \text{FALSE} \mapsto \varepsilon \end{array} \right.$
t: TRUE	
f: FALSE	

- Problem: TRUE und FALSE sind prinzipiell gleich, wollen sie zu einem Sort zusammen-fassen

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Betrachten noch mal die Bool Typen**

b: BOOL	$\left\{\begin{array}{l} \text{BOOL} \mapsto \text{TRUE},\text{FALSE} \\ \text{TRUE} \mapsto \varepsilon \\ \text{FALSE} \mapsto \varepsilon \end{array}\right.$
t: TRUE	
f: FALSE	

- Problem: TRUE und FALSE sind prinzipiell gleich, wollen sie zu einem Sort zusammen-fassen

t: CHAN ε	$\{\text{CHAN } \varepsilon \mapsto \varepsilon$
f: CHAN ε	

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Erinnern uns noch mal an Listen**

$$\mathrm{Nil}(l) \doteq l(nc).\overline{n}$$

$$\mathrm{Node}(l) \doteq (vl').l(nc).\overline{c}\langle vl' \rangle$$

...

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Erinnern uns noch mal an Listen**

$$\mathrm{Nil}(l) \doteq l(nc).\overline{n}$$

$$\mathrm{Node}(l) \doteq (vl').l(nc).\overline{c}\langle vl' \rangle$$

$$\ldots$$

- Würden gerne einen Typkonstruktor für Listen definieren

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Erinnern uns noch mal an Listen**

$$\mathrm{Nil}(l) \doteq l(nc).\overline{n}$$

$$\mathrm{Node}(l) \doteq (vl').l(nc).\overline{c}\langle vl' \rangle$$

$$\ldots$$

- Würden gerne einen Typkonstruktor für Listen definieren

$$ob : \begin{cases} \textsc{List}(\sigma) \mapsto \textsc{Chan}(\varepsilon),\ \textsc{Chan}(\sigma, \textsc{List}(\sigma)) \\ \ldots \end{cases}$$

ERWEITERUNGEN

TYPKONSTRUKTOREN

- Formalisieren diese Idee mit Hilfe Typkonstruktoren

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Formalisieren diese Idee mit Hilfe Typkonstruktoren**

  - Eine **Typsprache** $\sum$ hat Elemente der Form

$$C(\sigma_1,....,\sigma_n)$$

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Formalisieren diese Idee mit Hilfe Typkonstruktoren**

  - Eine **Typsprache** $\sum$ hat Elemente der Form

    $$C(\sigma_1,....,\sigma_n)$$

  - C aus der Menge Ω der **Typkonstruktoren** ist mit **Stelligkeit n**

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Formalisieren diese Idee mit Hilfe Typkonstruktoren**

  - Eine **Typsprache** $\sum$ hat Elemente der Form

    $$C(\sigma_1,....,\sigma_n)$$

  - C aus der Menge Ω der **Typkonstruktoren** ist mit **Stelligkeit n**

  - Disziplin für Typkonstruktoren wird über Typvariablen s_i definiert:

    $$ob : C(s_1,...,s_n) \mapsto \rho_1,...,\rho_n$$

    $$ob(C(\sigma_1,....,\sigma_n)) = \rho_1,...,\rho_n[\vec{s} := \vec{\sigma}]$$

ERWEITERUNGEN

TYPKONSTRUKTOREN

- Der wichtigste Typkonstruktor ist der Channel:

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Der wichtigste Typkonstruktor ist der Channel:**

$$\mathrm{CHAN}_n\,(s_1,\ldots,s_n) \mapsto s_1,\ldots,s_n$$

ERWEITERUNGEN

TYPKONSTRUKTOREN

- **Der wichtigste Typkonstruktor ist der Channel:**

$$\text{CHAN}_n\,(s_1,\ldots,s_n) \mapsto s_1,\ldots,s_n$$

- Anwendung von CHAN bei Listen

$$ob: \begin{cases} \text{LIST}(\sigma) \mapsto \text{CHAN}(\varepsilon),\ \text{CHAN}(\sigma, \text{LIST}(\sigma)) \\ \ldots \end{cases}$$

ANWENDUNGSBSP

OBJEKTE IM PI-KALKÜL

- Beispiel für die Mächtigkeit des pi-Kalküls

ANWENDUNGSBSP

OBJEKTE IM PI-KALKÜL

- Beispiel für die Mächtigkeit des pi-Kalküls
- Eine nicht-triviale Anwendung unserer Typen

ANWENDUNGSBSP

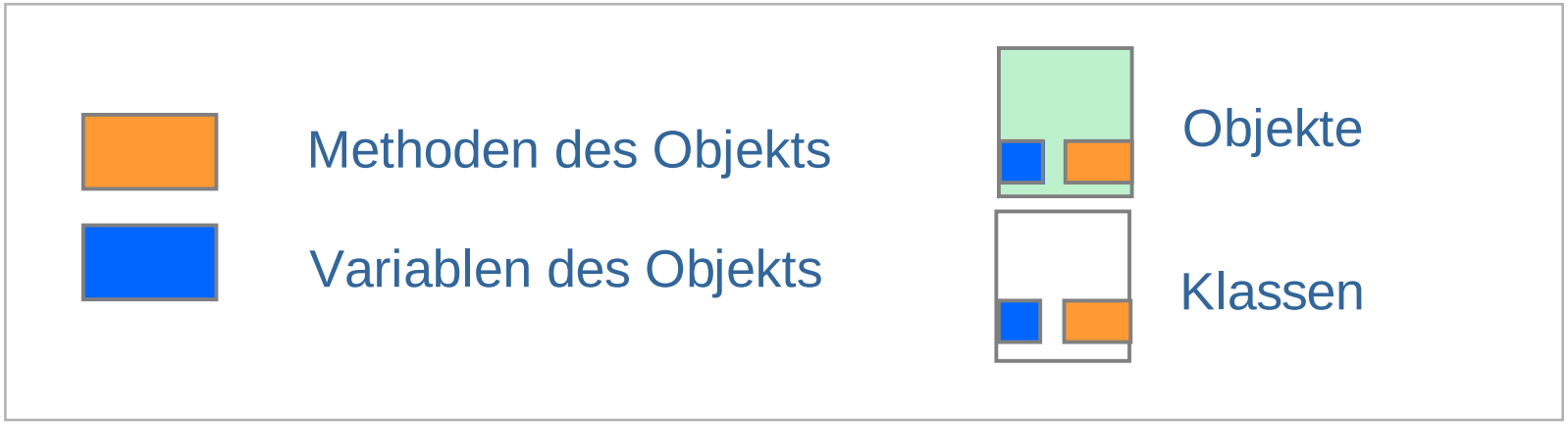
OBJEKTE IM PI-KALKÜL

- Beispiel für die Mächtigkeit des pi-Kalküls
- Eine nicht-triviale Anwendung unserer Typen
- Folgende Symbole werden benutzt

ANWENDUNGSBSP

OBJEKTE IM PI-KALKÜL

- Beispiel für die Mächtigkeit des pi-Kalküls
- Eine nicht-triviale Anwendung unserer Typen
- Folgende Symbole werden benutzt



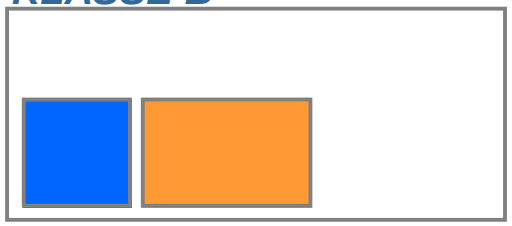
ANWENDUNGSBSP

Methoden des Objekts
Variablen des Objekts
Objekte
Klassen

BOPL - PROGRAMME

KLASSE A

KLASSE B



ANWENDUNGSBSP

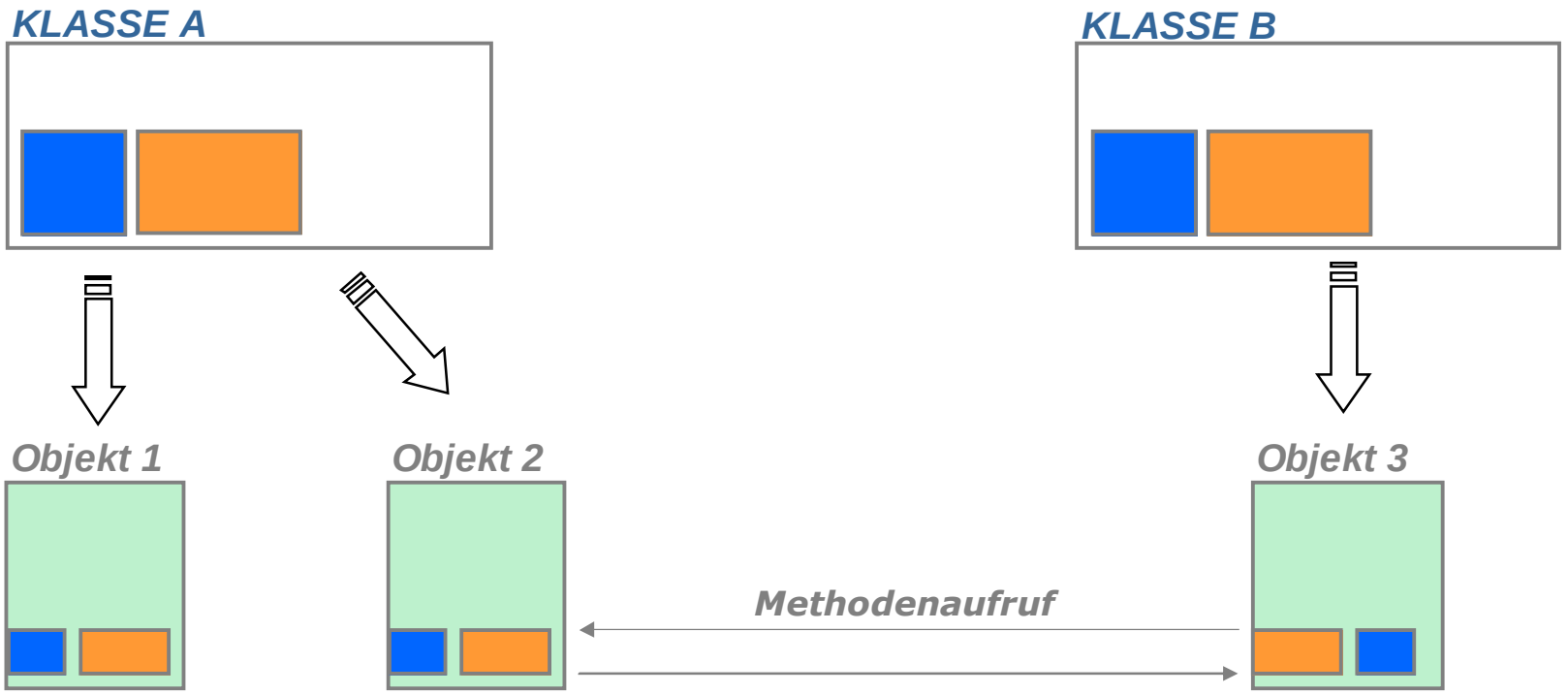
Methoden des Objekts
Variablen des Objekts
Objekte
Klassen

BOPL - PROGRAMME

ANWENDUNGSBSP

Methoden des Objekts
Variablen des Objekts
Objekte
Klassen

BOPL - PROGRAMME



ANWENDUNGSBSP

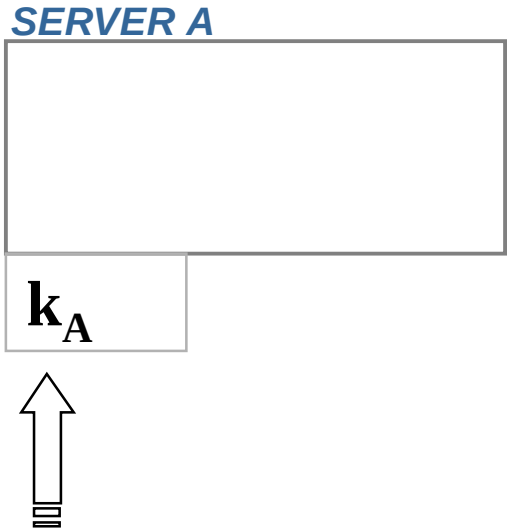
AUFBAU VON POPEL PROGRAMMEN

- **Klassendeklaration A: *class-dec*$_A$**

```
Class A

  VAR V1: A, V2: B

  METHODE M1(X1: B): A = S1

  METHODE M2(X1: A, X2: A): B = S2
```

ANWENDUNGSBSP

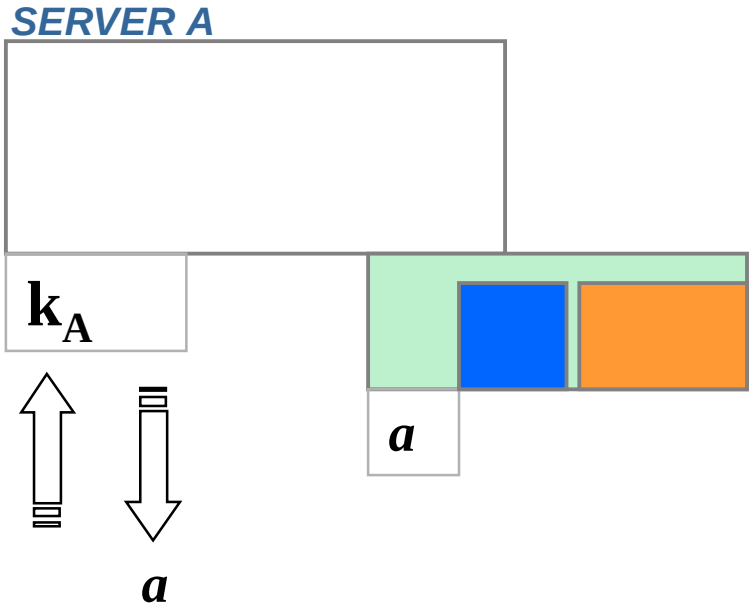
OBJEKTE IM PI-KALKÜL



- Klasse als Server, Dienst fordert neues Objekt über Adress k_A

ANWENDUNGSBSP

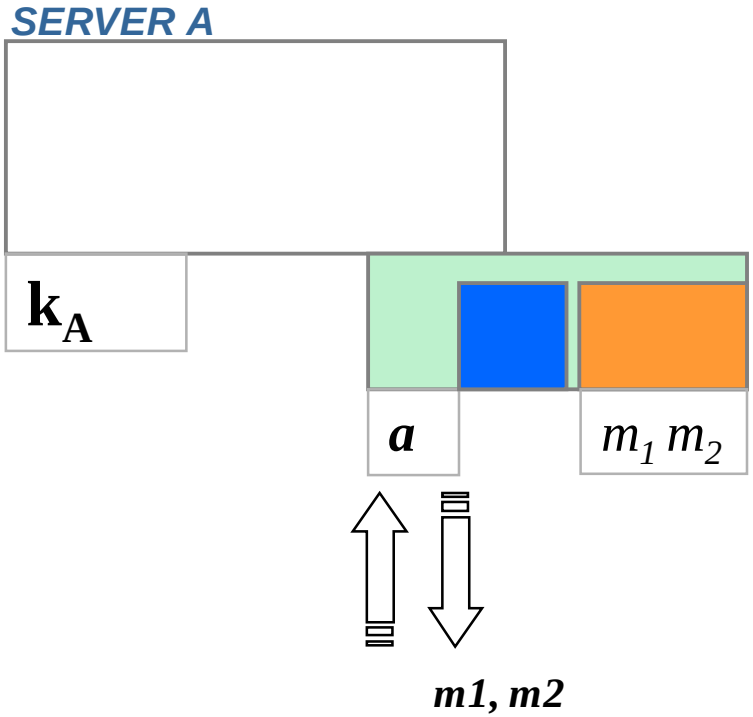
OBJEKTE IM PI-KALKÜL



- Server legt neues Objekt an und sendet als Antwort Adresse a des Objekts

ANWENDUNGSBSP

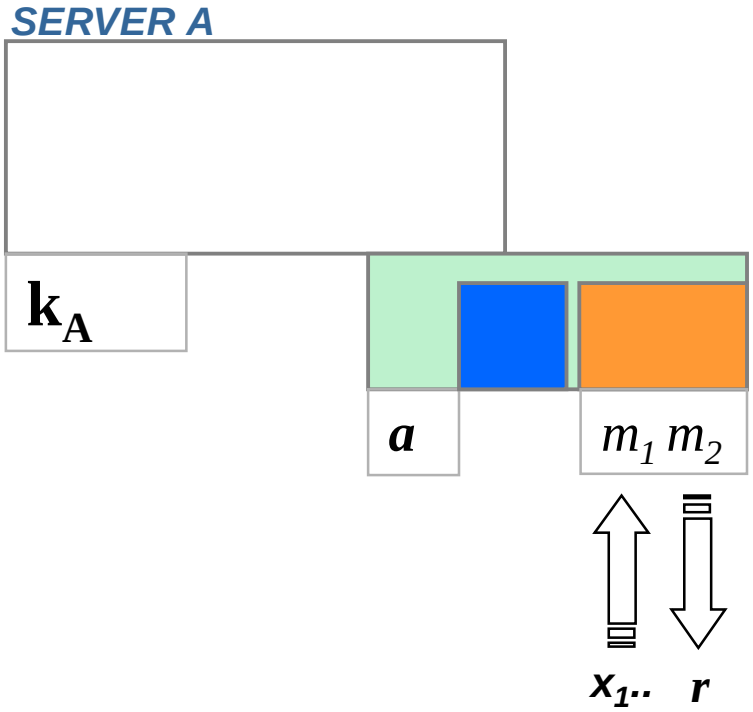
OBJEKTE IM PI-KALKÜL



- Benutzer fordert von Objekt A Methoden an

ANWENDUNGSBSP

OBJEKTE IM PI-KALKÜL



- Über die Adressen der Methoden ruft Benutzer diese auf
- und erhält Adresse des Rückgabe wertes

ANWENDUNGSBSP

ÜBERTRAGUNG VON KLASSEN

Klassendeklaration A

```
Class A
  VAR V1: A, V2: B

  METHODE M1(X1: B): A

  METHODE M2(X1: A, X2: A): B
```

pi-Kalkül Namen

k_A : Eindeutiger Name für Klasse A

ANWENDUNGSBSP

ÜBERTRAGUNG VON KLASSEN

Klassendeklaration A

Class A
VAR V1: A, V2: B

METHODE M1(X1: B): A

METHODE M2(X1: A, X2: A): B

pi-Kalkül Namen

k_A : Eindeutiger Name für Klasse A

v_1 ... : Variable der Klasse A

ANWENDUNGSBSP

ÜBERTRAGUNG VON KLASSEN

Klassendeklaration A

Class A
VAR V1: A, V2: B

METHODE M1(X1: B): A

METHODE M2(X1: A, X2: A): B

pi-Kalkül Namen

k_A : Eindeutiger Name für Klasse A
v_1 ... : Variable der Klasse A

x_1 ... : Methodenparameter in A
m_1 : Name der Methode M1
m_2 : Name der Methode M2
r : Name des Rückgabewertes einer Methode

ANWENDUNGSBSP

ÜBERTRAGUNG VON KLASSEN

Klassendeklaration A

```
Class A
  VAR V1: A, V2: B

  METHODE M1(X1: B): A

  METHODE M2(X1: A, X2: A): B
```

pi-Kalkül Namen

k_A : Eindeutiger Name für Klasse A

v_1 ... : Variable der Klasse A

x_1 ... : Methodenparameter in A

m_1 : Name der Methode M1

m_2 : Name der Methode M2

r : Name des Rückgabewertes einer Methode

a : Name eines beliebigen Objekts der Klasse A

ANWENDUNGSBSP

ÜBERTRAGUNG VON KLASSEN

- **Welche Typen haben unsere Namen?**

$a : A$	$A \mapsto ?$

ANWENDUNGSBSP

ÜBERTRAGUNG VON KLASSEN

- **Welche Typen haben unsere Namen?**

$a : A$ $m_1 : \text{METHOD}(B, A)$ $m_2 : \text{METHOD}(A, A, B)$	$A \mapsto ?$ $\text{METHOD}(s_1, .., s_n, s') \mapsto s_1, .., s_n, \text{CHAN } s'$

ANWENDUNGSBSP

ÜBERTRAGUNG VON KLASSEN

- **Welche Typen haben unsere Namen?**

$a : A$ $m_1 : \text{METHOD}(B, A)$ $m_2 : \text{METHOD}(A, A, B)$	$A \mapsto \text{METHOD}(B, A),\ \text{METHOD}(A, A, B)$ $\text{METHOD}(s_1, .., s_n, s') \mapsto s_1, .., s_n, \text{CHAN } s'$

ANWENDUNGSBSP

ÜBERTRAGUNG VON KLASSEN

- **Welche Typen haben unsere Namen?**

$a : A$	$A \mapsto \mathrm{METHOD}(B, A),\ \mathrm{METHOD}(A, A, B)$
$m_1 : \mathrm{METHOD}(B, A)$	$\mathrm{METHOD}(s_1, .., s_n, s') \mapsto s_1, .., s_n, \mathrm{CHAN}\ s'$
$m_2 : \mathrm{METHOD}(A, A, B)$	
$k_A : \mathrm{CLASS}\ A$	$\mathrm{CLASS}\ s \mapsto s$

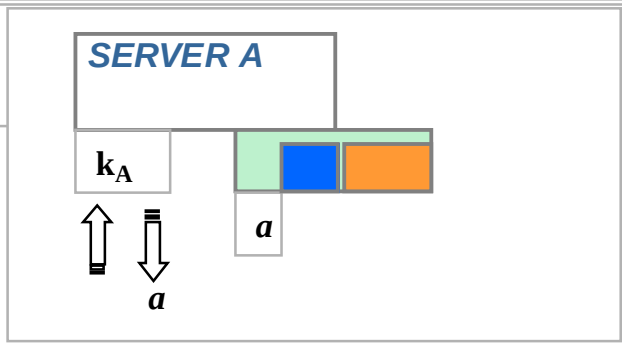
ANWENDUNGSBSP

ÜBERTRAGUNG VON KLASSEN

- **Welche Typen haben unsere Namen?**

$a : A$	$A \mapsto METHOD(B, A),\ METHOD(A, A, B)$
$m_1 : METHOD(B, A)$	$METHOD\ (s_1, .., s_n, s') \mapsto s_1, .., s_n, CHAN\ s'$
$m_2 : METHOD(A, A, B)$	
$k_A : CLASS\ A$	$CLASS\ \ s \mapsto s$
$v_1 \ldots : REF\ A$	
$x_1 \ldots : A$	
$r : CHAN\ A$	

ANWENDUNGSBSP

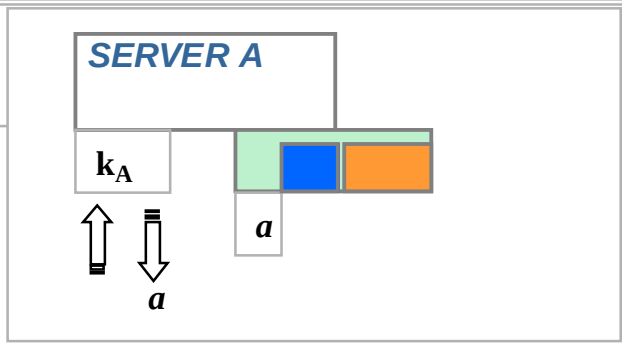


- **Modellierung der Klassendefinition**

$$[\![\text{class-dec}_A]\!] \stackrel{\text{Def}}{=} \; !\text{new a } \bar{k}_A\langle a\rangle.\text{Object}_A\langle a\rangle$$

  - Zugriff auf neues Objekt **a** über Namen $\mathbf{k_a}$

ANWENDUNGSBSP



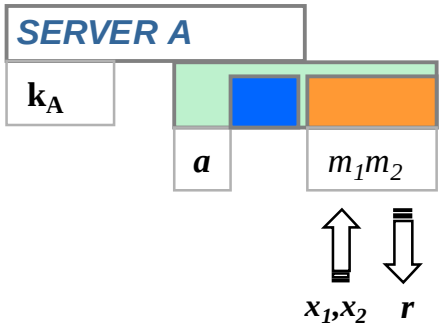
- **Modellierung eines Objekts a**

$$[\![\text{class-dec}_A]\!] \overset{\text{Def}}{=} \; !\text{new a } \bar{k}_A \langle a \rangle . \text{Object}_A \langle a \rangle$$

$$\text{Object}_A (a) \overset{\text{Def}}{=} \text{new } v_1\text{:REF A, } v_2\text{: REF B}$$
$$(\text{Nullref} \langle v1 \rangle | \text{Nullref} \langle v2 \rangle | \; !\text{Methoden}_A \langle a \rangle)$$

- jedes Objekt hat seine eigenen Methoden und Variablen
- Variablen werden zunächst mit Null initialisiert

ANWENDUNGSBSP



Modellierung der Methoden

$$\text{Object}_A(a) \overset{\text{Def}}{=} \text{new } v_1\text{:REF A}, v_2\text{: REF B}$$
$$(\text{Nullref}\langle v1\rangle | \text{Nullref}\langle v2\rangle | !\text{Methoden}_A\langle a\rangle)$$

$$\text{Methoden}_A(a) \overset{\text{Def}}{=} \text{new } m_1\text{: METHOD(B, A)}, m_2\text{: METHOD(A, A, B)}$$
$$a\langle m_1m_2\rangle.\begin{pmatrix} m_1(x_1\text{: B, r: CHAN A}).[\![S_1]\!] \\ + m_2(x_1\text{: A, } x_2\text{: A, r: CHAN B}).[\![S_2]\!] \end{pmatrix}$$

- Der Name **a** wird wie als Selbstreferenz (wie **self)** verwendet

ANWENDUNGSBSP

OBJEKTE IN PI

- **Was haben wir modelliert?**
  - getypte Methoden
  - Enkapsulierung von Daten
  - Parallele Berechnung
- **Was haben wir nicht modelliert?**
  - Subtyping
  - Vererbung, …

ZUSAMMENFASSUNG

- Typen helfen uns Verhalten von Prozessen einzuschränken und besser zu verstehen
- Mit Hilfe von Typen lassen sich Objekte recht leicht im pi-Kalkül modellieren

- Noch viele Erweiterungen des Typsystems möglich
- Aussagen wie Deadlock mit Typen untersuchen
- Den pi-Kalkül kann man (fast) wie eine höhere Programmiersprache benutzen.

LITERATUR

COMMUNICATING AND MOBILE SYSTEMS: THE PI-CALCULUS Robin Milner, Cambridge University, Cambridge, Great Britain 1999

THE PI-CALCULUS : A THEORY OF MOBILE PROCESSES Davide Sangiorgi, INRIA Sophia Antipolis and David Walker University of Oxford, Cambridge University Press 2001

DANKE