Grober Aufbau eines Java-Programms

- Wir sollten an dieser Stelle vielleicht auch klären, wie ein Java-Programm grundsätzlich aufgebaut ist.
- Ein Modul (Klassenvereinbarung, z.B. die Klasse `Bewegung` in der Datei `Bewegung.java` von oben) ist zunächst nicht so einfach ausführbar.
- Damit eine Klasse ausführbar wird, muss sie eine Methode `main` enthalten, typischerweise mit der Signatur

```
public static void main(String[] args)
```

enthalten.

Ein einfaches imperatives Java-Programm

Allgemeines Schema einer ausführbaren Klasse:

```
public class KlassenName {
  public static void main(String[] args) {
    // Hier geht's los mit
    // Anweisungen (elementare Verarbeitungsschritte)
    // z.B. Methodenaufrufe
    // ...
  }
}
```

Die Textdatei, die den Java-Code enthält, heißt `KlassenName.java`, also genauso wie die enthaltene Klasse, mit der Endung `java`.

Beispiel: `HelloWorld.java`

Hier nochmal das Beispiel:

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }
}
```

In diesem Fall wird die Methode `System.out.println` aufgerufen. Als Argument bekommt diese Methode einen String, also eine Zeichenkette übergeben (kennen wir leider noch nicht!!!). Diesen String gibt die Methode über die Standard-Ausgabe (default: Kommandozeile) aus.

Beispiel: `HelloWorld.java`

- Innerhalb der `main`-Methode kann man natürlich auch Funktionen (Methoden) aufrufen, z.B.:

```
public class Bewegung {

  ... Klasse Bewegung wie bisher mit strecke, etc.

  public static void main(String[] args) {
    double strecke = Bewegung.strecke(1.0,2.0,3.0);
    System.out.println("Strecke: "+strecke);
  }
}
```

- Die `main`-Methode ruft `strecke(1.0,2.0,3.0)` auf, weist den Rückgabewert einer Variablen zu und gibt das Ergebnis aus.

Konventionen

In der Java-Programmierung gibt es einige Konventionen, deren Einhaltung das Lesen von Programmen erleichtert, z.B.:

- Klassennamen beginnen mit großen Buchstaben, z.B. `HelloWorld`.
- Methodennamen, Attributnamen und Variablennamen beginnen mit kleinen Buchstaben, z.B.
  - Methoden: `main`, `println`,
  - Klassenvariable: `out`,
  - Variable: `args` (weder Instanz- noch Klassenvariable, sondern Parametervariable).
- Bei zusammengesetzte Namen beginnt jeder innere Teilname mit einem großen Buchstaben, z.B. Klassennamen `HelloWorld`.

Kapitel 5: Prozeduraufrufe i

Kapitel 5: Prozeduraufrufe ii

Prozeduraufrufe

- Genauso wie bei Funktionen stellt sich auch für die Prozeduren die Frage: Was passiert, wenn eine Prozedur mit konkreten (Eingabe-Werten) ausgeführt wird, also die Anweisungen im Rumpf der Prozedur ausgeführt werden?
- In unserem Beispiel: Was passiert, wenn die Prozedur *fBerechnen*1 z.B. mit dem Wert 3.0 also *fBerechnen*1(3.0) *aufgerufen* wird?
- Wir klären diese Frage zunächst wieder informell, d.h. mit unserer Intuition, dass es sich bei einer Variable/Konstante um einen Zettel handelt.

Prozeduraufrufe

- Während der Ausführung des Programms werden Zettel beschrieben und verändert.
- Intuitiv werden dabei *Zustände* verändert: der Zustand in dem das Programm sich aktuell befindet.
- D.h. ein Programm besteht nun aus einer Menge an Zetteln, und der Zustand eines Programms ist gekennzeichnet durch den aktuellen Inhalt der Zettel.
- Zustandsänderungen sind durch Änderungen der Zettel gekennzeichnet.

Prozeduraufrufe

- Offenbar bewirkt eine Anweisung eine *Veränderung des Zustands* indem sich ein Programm (in Ausführung) gerade befindet.
- Eine Wertzuweisung an eine Variable z.B. verändert den Zustand so, dass die Variable einen neuen Wert bekommt (also ab jetzt neu substituiert wird).
- Stellt sich noch die Frage, wie so ein Zustand genau aussehen kann: intuitiv besteht ein Zustand wie bereits erwähnt aus der Menge an Zetteln, die anfangs vereinbart wurden[8].

[8] Wenn wir Deklarationen zu beliebigen Zeitpunkten im Rumpf zulassen würden, entsprechen alle Zettel, die bis zu dem entspr. Zeitpunkt deklariert wurden.

Definition ((Programm-) Zustand)

Ein (Programm-) Zustand (engl. State) ist eine Menge $\mathcal{S}$ von Paaren $z = (x, d)$ mit folgenden Eigenschaften:

- *x ist eine Variable ($x \in V$), d.h. x ist*
  - *ein Eingabeparameter (im Kopf der Prozedur),*
  - *eine (unter `CONST` vereinbarte) Konstante, oder*
  - *eine (unter `VAR` vereinbarte) Variable.*

  x heißt Name oder Bezeichner von z.
- *d ist ein Objekt der Sorte von x (oder undefiniert, notiert als ω) und heißt Inhalt von z.*
- *Die Menge $\mathcal{S}$ enthält keine zwei verschiedenen Paare (x, d_1) und (x, d_2) mit gleichem Namen x.*

Zustände

Bemerkungen:

- Ein Paar (x, d) formalisiert das Bild eines Zettels x auf dem der Wert d steht.
- (x, ω) kann als *leerer Zettel* verstanden werden.

Zustände

- Wir unterscheiden jetzt noch die Bezeichner, die Eingabeparameter der Prozedur sind (die kennen wir ja schon von Funktionen als $V(\sigma)$), und Bezeichner, die als Variablen oder Konstanten zu Beginn des Rumpfs mit **`VAR`** und **`CONST`** vereinbart werden:
  - $N(\mathcal{S})$ ist die Menge *aller* Namen in $\mathcal{S}$.
  - $V(\mathcal{S})$ sind die Eingabeparameter (wie bisher).
  - $N(\mathcal{S}) \setminus V(\mathcal{S})$ sind die deklarierten Bezeichner im Rumpf; diese heißen *lokale* Variablen/Konstanten.
- Ein Zustand $\mathcal{S}$ definiert eine Substitution für alle Namen $x \in N(\mathcal{S})$ in $\mathcal{S}$ deren Wert $d \neq \omega$ ist: das Paar $(x, d) \in \mathcal{S}$ definiert offenbar die Substitution $\sigma = [x/d]$.

Zustände

- Dies kommt Ihnen bekannt vor:
  - Bei Funktionen haben wir gesehen, dass der Aufruf eines Algorithmus eine Substitution σ für die Eingabeparameter des Algorithmus $V(\sigma)$ spezifiziert (analog gilt dies nun für $V(\mathcal{S})$, das wir bisher mit $V(\sigma)$ bezeichnet hatten).
  - Zustände beinhaltet offenbar nun aber nicht mehr nur die Eingabeparameter sondern auch die im Rumpf zusätzlich vereinbarten Konstanten und Variablen, daher ist $N(\mathcal{S})$ auch wirklich eine Obermenge von $V(\mathcal{S})$.
- Dieser Unterschied zwischen Funktion und Prozedur drückt sich auch in unserer unterschiedlichen Schreibweise aus: σ bzw. $\mathcal{S}$ und V bzw. N.

Zustände

- Die Unterscheidung in V und N ist leider nötig:
- Offensichtlich ist für einen Prozeduraufruf $(e, d) \in \mathcal{S}$ mit $d \neq \omega$ für alle *Eingabeparameter* $e \in V(\mathcal{S})$, d.h. alle Variablen, die Eingabeparameter bezeichnen, sind definiert/nicht leer (der Aufruf der Prozedur mit konkreten Werten erwirkt eine entsprechende Substitution).
- Dies gilt aber nicht notwendigerweise für alle lokalen *Variablen und Konstanten* $N(\mathcal{S}) \setminus V(\mathcal{S})$, denn Variablen und Konstanten müssen in unserer Schreibweise nicht sofort initialisiert werden (das passiert teilweise erst viel später), können also zunächst „leer“ bzw. undefiniert sein.

Zustände

- Damit wird der Wert eines Ausdrucks u nun abhängig vom Zustand $\mathcal{S}$, in dem er betrachtet wird.
- Wir schreiben daher nun $W_{\mathcal{S}}$ statt W_σ (was OK ist, da $\mathcal{S}$ ja auch eine Menge von Substitutionen definiert).
- Die formale Definition von $W_{\mathcal{S}}$ ist allerdings leicht: es genügt eine Erweiterung der rekursiven Definition des Wertes eines Ausdrucks u für den Fall, dass u eine im Rumpf vereinbarte lokale Variable oder Konstante aus $N(\mathcal{S}) \setminus V(\mathcal{S})$ ist.

- Die anderen Fälle, dass u ein Eingabeparameter, ein Literal, die Anwendung eines Basisoperators oder ein Aufruf einer Funktion (mit Rückgabewert ohne Seiteneffekte) ist, bleiben unverändert:
  In diesen Fällen ersetzen wir einfach $W_\sigma(u)$ durch $W_\mathcal{S}(u)$.
- Für den Fall $u = x$ mit $x \in N(\mathcal{S}) \setminus V(\mathcal{S})$ (lokale Variable/Konstante) definieren wir nun zusätzlich:
  - ist $(x, d) \in \mathcal{S}$ und $d \neq \omega$, so ist $W_\mathcal{S}(u) = d$
  - ist $(x, \omega) \in \mathcal{S}$ oder $x \notin N(\mathcal{S})$, so ist $W_\mathcal{S}(u) = \omega$
- Wir schreiben $W(u)$ statt $W_\mathcal{S}(u)$, wenn sich $\mathcal{S}$ aus dem Kontext ergibt.

Zustandsübergänge

- Was passiert mit einem Zustand $\mathcal{S}$, wenn Anweisungen ausgeführt werden, die Variablen aus $N(\mathcal{S})$ verändern bzw. neue vereinbaren?
- Intuitiv:
  - Bei der Vereinbarung einer Variablen (analog Konstanten) am Anfang des Prozedurrumpfs[9] wird ein neuer Zettel hinzugefügt, der Zettel ist zunächst leer.
  - Bei der Initialisierung einer Variablen (analog Konstanten) wird der entsprechende Zettel (erstmalig) verändert (bildlich: der leere Zettel wird durch einen neuen, nun nicht-leeren Zettel gleichen Namens ersetzt).
  - Bei einer weiteren Wertzuweisung an eine Variable wird der entsprechende Zettel (wieder) verändert.

[9] wieder leicht erweiterbar auf beliebige Stellen im Rumpf.

Definition (Zustandsänderung)

Eine Anweisung a bewirkt eine Zustandsänderung von Zustand $\mathcal{S}$ in einen Nachfolgezustand $\hat{\mathcal{S}}$ wie folgt:

- *Deklaration: Hat a die Form* ***var*** $x \in S$ *bzw.* ***const*** $x \in S$ *mit* $x \notin N(\mathcal{S})$*, so ist* $\hat{\mathcal{S}} = \mathcal{S} \cup \{(x, \omega)\}$
- *Initialisierung: Hat a die Form* $x \leftarrow u;$ *mit* $x \in N(\mathcal{S})$ *und ist u ein Ausdruck mit* $W(u) \neq \omega$*, so ist* $\hat{\mathcal{S}} = \mathcal{S} \setminus \{(x, \omega)\} \cup \{(x, W(u))\}$
- *Wertzuweisung: Hat a die Form* $x \leftarrow u;$ *mit* $(x, d) \in \mathcal{S}$ *(d.h.* $x \in N(\mathcal{S})$ *hat den Wert d) und ist u ein Ausdruck mit* $W(u) \neq \omega$*, so ist* $\hat{\mathcal{S}} = \mathcal{S} \setminus \{(x, d)\} \cup \{(x, W(u))\}$

Zustandsübergänge

- Wir schreiben $\mathcal{S} \xrightarrow{a} \hat{\mathcal{S}}$, um auszudrücken, dass wir $\hat{\mathcal{S}}$ aus $\mathcal{S}$ durch die Anwendung von a erhalten haben.
- Um uns möglicherweise Schreibarbeit zu ersparen, können wir auch Zustandsübergänge von mehreren sequentiellen Anweisungen zusammenfassen:
- Ist A eine Folge von Anweisungen (Deklaration/Initialisierung,Wertzuweisung) $a_1; \ldots a_k;$ und es gilt $\mathcal{S} = \mathcal{S}_0 \xrightarrow{a_1} \ldots \xrightarrow{a_k} \mathcal{S}_k = \hat{\mathcal{S}}$, so heißt $\hat{\mathcal{S}}$ Nachfolgezustand von $\mathcal{S}$ bzgl. A und wir schreiben $\mathcal{S} \xrightarrow{A} \hat{\mathcal{S}}$.

Zustandsübergänge

- Wir vereinbaren zudem, dass der *Anfangszustand* beim Aufruf einer Prozedur) $\mathcal{S}_0$ die Eingabevariablen $V(\mathcal{S})$ der Prozedur (so wie bei Funktionen, bei denen es ja nur diesen Zustand gibt), die ausgeführt wird, enthält. Diese Eingabeparameter sind mit den entsprechenden konkreten Eingabe-Werten des Aufrufs belegt.
- Beispiel: Beim Aufruf der Prozedur *fBerechnen*1(x) mit 3.0, also *fBerechnen*1(3.0), ist der initiale Zustand $\mathcal{S}_0 = \{(x, 3.0)\}$, der wie bei Funktionen die Substitution des Eingabeparameters x durch den konkreten Wert 3.0 enthält.

Zustandsübergänge

- Beispiel: hier nochmal der Algorithmus

  PROCEDURE *fBerechnen*1($x : \mathbb{R}$) $\rightarrow \mathbb{R}$
  OUTPUT Berechnung der Funktion f
  PRE $x \neq 1$
  BODY
  VAR $y_1, y_2, y_3 : \mathbb{R}$;
  $y_1 \leftarrow x + 1$;
  $y_2 \leftarrow y_1 + 1/y_1$;
  $y_3 \leftarrow y_2 \cdot y_2$;
  RETURN y_3;

  von oben.

- Wie wird der Algorithmus für einen konkreten Wert (z.B. 3.0) ausgeführt, d.h. was passiert beim Aufruf von *fBerechnen*(3.0)?

Ausführung des Aufrufs *fBerechnen*(3.0):

Anweisung	x	y_1	y_2	y_3	Zustand
Aufruf *fBerechnen*1(3.0)	3.0	■	■	■	$\{(x, 3.0)\}$
VAR $y_1, y_2, y_3 : \mathbb{R}$;	3.0	ω	ω	ω	$\{(x, 3.0), (y_1, \omega), (y_2, \omega), (y_3, \omega)\}$
$y_1 \leftarrow x + 1$;	3.0	$W(x + 1)$	ω	ω	$\{(x, 3.0), (y_1, W(x + 1)), (y_2, \omega), (y_3, \omega)\}$
(bzw.)	3.0	4.0	ω	ω	$\{(x, 3.0), (y_1, 4.0), (y_2, \omega), (y_3, \omega)\}$
...					

Legende: ■ = Zettel existiert noch nicht.

Fortsetzung:

Anweisung	x	y_1	y_2	y_3	Zustand
...					
$y_2 \leftarrow y_1 + 1/y_1;$	3.0	4.0	$W(y_1 + 1/y_1)$	ω	$\{(x, 3.0), (y_1, 4.0), (y_2, W(y_1 + 1/y_1)), (y_3, \omega)\}$
(bzw.)	3.0	4.0	4.25	ω	$\{(x, 3.0), (y_1, 4.0), (y_2, 4.25), (y_3, \omega)\}$
$y_3 \leftarrow y_2 \cdot y_2;$	3.0	4.0	4.25	$W(y_2 \cdot y_2)$	$\{(x, 3.0), (y_1, 4.0), (y_2, W(y_1 + 1/y_1)), (y_3, W(y_2 \cdot y_2))\}$
(bzw.)	3.0	4.0	4.25	18.0625	$\{(x, 3.0), (y_1, 4.0), (y_2, 4.25), (y_3, 18.0625)\}$

Prozeduraufrufe in Prozeduren

- Die Erweiterung des Modulbegriffs um Prozeduren ermöglicht uns nun natürlich auch, diese Prozeduren in anderen Algorithmen zu verwenden.
- Das Modul `BEWEGUNGI` auf der nächsten Folie ist eine imperative Variante des Moduls *Bewegung* aus dem vorherigen Kapitel .
- Die Prozedur *streckeI* wird in der Prozedur *arbeitI* *aufrufen*.
- Solch ein Aufruf ist syntaktisch ein Ausdruck, kann also überall stehen, wo ein Ausdruck stehen darf (die formale Erweiterung der induktiven Definition von Ausdrücken ist trivial, und sei den Fleißigen unter Ihnen überlassen).

Prozeduraufrufe in Prozeduren

LMU

```
MODULE BEWEGUNGI
  OPS
    PROCEDURE streckel(m : ℝ, t : ℝ, k : ℝ) → ℝ
      PRE   m > 0, t ≥ 0
      BODY
        VAR   b : ℝ;
        b ← k · t²;
        b ← b/(2 · m);
        RETURN   b;
    PROCEDURE endgeschwindigkeit(m : ℝ, t : ℝ, k : ℝ) → ℝ
        ...
    PROCEDURE arbeitl(m : ℝ, k : ℝ, t : ℝ) → ℝ
      PRE   m > 0, t ≥ 0
      BODY
        VAR   s : ℝ;
        s ← strecke(m, t, k);
        RETURN   k · s;
```

Prozeduren ohne Rückgabe

- Bei einer Funktion ist der Bildbereich eine wichtige Information: $f : D \to B$.
- Bei einer Prozedur, die keine Funktion ist, wählt man als Bildbereich oft die leere Menge: $p : D \to \emptyset$.
- Dies signalisiert, dass die Seiteneffekte der Prozedur zur eigentlichen Umsetzung eines Algorithmus gehören, dagegen aber kein (bestimmtes) Element aus dem Bildbereich einer Abbildung als Ergebnis des Algorithmus angesehen werden kann.
- Bei der `PROCEDURE`-Vereinbarung fehlt dann einfach der Bildbereich.

Beispiel

PROCEDURE *vertausche*($x : \mathbb{N}, y : \mathbb{N}$)
  OUTPUT Vertausche die Werte von x und y
  BODY
    VAR $a : \mathbb{N}$;
    $a \leftarrow x$;
    $x \leftarrow y$;
    $y \leftarrow a$;

In dieser Prozedur fehlt die finale **RETURN**-Anweisung, da nichts zurück gegeben wird.

Prozeduren ohne Rückgabe

- Wir lassen den Aufruf einer solchen Prozedur ohne Wertzuweisung an eine Variable (was genau genommen nur ein Ausdruck ist) als Anweisung zu.
- Beispiel:

PROCEDURE *groesserOderKleiner*$(x : \mathbb{N}, y : \mathbb{N}) \to \mathbb{B}$
  BODY
    vertausche(x, y);
    RETURN $x < y$;

- Diese Anweisung (hier *vertausche*(x, y);) heißt *Ausdrucksanweisung*.
- Ganz allgemein ist ein Ausdruck gefolgt von einem Strichpunkt solch eine Ausdrucksanweisung.

Prozedurenaufrufe in Prozeduren

- Und jetzt stellt sich natürlich die Frage: vertauscht *vertausche*(x, y) in *groesserOderKleiner* tatsächlich die Werte von x und y?
- Oder konkret: was für ein Ergebis liefert z.B. *groesserOderKleiner*$(1, 2)$?
- Diese Antwort ist leider nicht in allen Programmiersprachen gleich ...
- Die Anwort hängt davon ab, wie Variablen an eine Prozedur *übergeben* werden.

Prozedurenaufrufe in Prozeduren

- Also: was passiert, wenn eine Prozedur mit Signatur
  $p_a(e_1 : T_1, \ldots, e_n : T_n) \quad [\rightarrow T] \quad$ (d.h. $\rightarrow T$ ist optional)
  innerhalb des Rumpfes einer anderen Prozedur p_u mit Variablen $x_1 \in T_1, \ldots, x_n \in T_n$, die in p_u bekannt sind, als Argument aufgerufen wird, d.h. im Rumpf von p_u steht die Anweisung

  $$p_a(x_1, \ldots, x_n);$$

  Wir erwähnt: wir sagen dazu, die Variablen $x_1, \ldots, x_n$ werden an p_a *übergeben*. (Im Beispiel oben: p_a ist *vertausche* und p_u ist *groesserOderKleiner*)
- Variablen können grundsätzlich auf zwei Arten übergeben werden:

Parameterübergabe bei Prozeduraufrufen

- Möglichkeit 1: *Call-by-value*
  - Für jede (formale) Eingabe-Variable $e_1, \ldots, e_n$ von p_a wird im Methoden-Block eine neue Variable angelegt.
  - Diese Extra-Variablen erhalten die *Werte* der übergebenen Variablen $x_1, \ldots, x_n$ aus p_u (im Zustand zum Zeitpunkt des Aufrufs von p_a in p_u).
  - Die übergebenen Variablen $x_1, \ldots, x_n$ sind im Rumpf von p_a **nicht** *sichtbar*, d.h. nicht in der Menge $N(\mathcal{S}_0^{p_a})$ (wobei $\mathcal{S}_0^{p_a}$ der Anfangszustand der Prozedur p_a ist) enthalten (später können natürlich Variablen/Konstanten gleichen Namens innerhalb von p_a deklariert werden, dann sind diese aber neue Zettel und haben nix mit den anderen zu tun).
  - Auf diese Weise bleibt der Wert der ursprüngliche Variablen von Anweisungen innerhalb der Methode p_a unberührt.

Parameterübergabe: Call-by-value

- Java wertet Parameter call-by-value aus und wir spezifizieren das in unserer Semantik ebenso.
- Dies macht insbesondere deshalb Sinn, weil in Java (und bei uns) nicht nur Variablen sondern auch ganze Ausdrücke an Methoden übergeben werden dürfen (so hatten wir übrigens auch die Syntax von Ausdrücken definiert).
- Wir erweitern dazu die Formalisierung der Zustandsübergänge, die eine Anweisung *a* bewirkt, um folgende Fälle:

Parameterübergabe: Call-by-value

Fall 1: Prozedur ohne Rückgabe

- Beim Aufruf einer Prozedur mit der Signatur $p(e_1 : T_1, \ldots, e_n : T_n)$ mit konkreten Eingaben (Ausdrücken) $w_1, \ldots, w_n$

  (d.h. Anweisung a hat die Form $p(w_1, \ldots, w_n);$)

  im Zustand $\mathcal{S}$ wird zunächst der Zustand

  $$\mathcal{S}_{init_p} = \{(e_1, W_{\mathcal{S}}(w_1)), \ldots, (e_n, W_{\mathcal{S}}(w_n))\}$$

  erreicht, d.h., $\mathcal{S}_{init_p}$ enthält nur die Eingabe-Variablen mit den Werten der übergebenen Ausdrücke bzgl. $\mathcal{S}$.

Parameterübergabe: Call-by-value

Fall 1 (Fortsetzung):

- Der Rumpf r von p überführt $\mathcal{S}_{init_p}$ in $\mathcal{S}_{end_p}$, d.h. $\mathcal{S}_{init_p} \xrightarrow{r} \mathcal{S}_{end_p}$.
- Da p keinen Rückgabewert hat, ist der Nachfolgezustand von $\hat{\mathcal{S}}$ bzgl. a gleich $\mathcal{S}$, d.h. $\mathcal{S} \xrightarrow{a} \mathcal{S}$

  (die Seiteneffekte haben also keinen Einfluss auf $\hat{\mathcal{S}}$ bzw. $\mathcal{S}$ — WHAT????).

Fall 2: Prozedur mit Rückgabe

- Beim Aufruf einer Prozedur der Signatur $p(e_1 : T_1, \ldots, e_n : T_n) \to T$ mit konkreten Eingaben (Ausdrücke) $w_1, \ldots, w_n$

  (d.h. Anweisung a hat die Form $x = p(w_1, \ldots, w_n);$)

  im Zustand $\mathcal{S} = \{\ldots(x, w), \ldots\}$ (wobei $w = \omega$ sein kann) wird zunächst der Zustand

  $$\mathcal{S}_{init_p} = \{(e_1, W_{\mathcal{S}}(w_1)), \ldots, (e_n, W_{\mathcal{S}}(w_n))\}$$

  erreicht, d.h., $\mathcal{S}_{init_p}$ enthält nur die Eingabe-Variablen mit den Werten der übergebenen Ausdrücke bzgl. $\mathcal{S}$ (wie oben).

Fall 2 (Fortsetzung):

- Der Rumpf r von p überführt $\mathcal{S}_{init_p}$ in $\mathcal{S}_{end_p}$, d.h. $\mathcal{S}_{init_p} \xrightarrow{r} \mathcal{S}_{end_p}$.
- Die finale **`RETURN`**-Anweisung in p enthält den Ausdruck t vom Typ T mit Wert $W_{\mathcal{S}_{end_p}}(t)$ im Zustand $\mathcal{S}_{end_p}$.
- Der Nachfolgezustand ist damit gegeben durch $\hat{\mathcal{S}} = \mathcal{S} \setminus \{(x, w)\} \cup \{(x, W_{\mathcal{S}_{end_p}}(t))\}$ und es gilt wieder $\mathcal{S} \xrightarrow{a} \hat{\mathcal{S}}$

  (also letztlich ein ähnlicher Effekt wie Funktionen, die Seiteneffekte passieren nur “innerhalb” von p).

Beispiel

Hier nochmal die beiden Prozeduren von vorher:

```
    PROCEDURE vertausche(x : ℕ, y : ℕ)
      BODY
1       VAR  a : ℕ;
2       a ← x;
3       x ← y;
4       y ← a;
    PROCEDURE gOk(x : ℕ, y : ℕ) → 𝔹
      BODY
5       vertausche(x, y);
6       RETURN  x < y;
```

Was für ein Ergebis liefert $gOk(1, 2)$?

Parameterübergabe: Call-by-value

```
    PROCEDURE vertausche(x : ℕ, y : ℕ)
      BODY
1       var  a : ℕ;
2       a ← x;
3       x ← y;
4       y ← a;
```

```
    PROCEDURE gOk(x : ℕ, y : ℕ) → 𝔹
      BODY
5       vertausche(x, y);
6       return  x < y;
```

Aufruf von $gOk(1, 2)$:

- Wir unterscheiden die formalen Eingabeparameter von *gOk* und *vertausche*: x_a/y_a sind die von *gOk*, der äußeren Prozedur, x_i/y_i die von *vertausche*, der inneren Prozedur
- Der Zustand vor Zeile 5 ist $\mathcal{S}_0 = \{(x_a, 1), (y_a, 2)\}$; dies ist der Anfangszustand von *gOk* für die konkreten Eingabewerte 1 und 2.
- Beim Aufruf von $vertausche(x_a, y_a)$ in Zeile 5 werden neue Zettel mit Namen x_i und y_i für die Eingabeparameter von *vertausche* angelegt, die nur im Rumpf von *vertausche* gelten, und mit den Werten von den übergebenen Ausdrücken x_a und y_a belegt sind.

Parameterübergabe: Call-by-value

```
PROCEDURE vertausche(x : ℕ, y : ℕ)
  BODY
1     var  a : ℕ;
2     a ← x;
3     x ← y;
4     y ← a;
```

```
PROCEDURE gOk(x : ℕ, y : ℕ) → 𝔹
  BODY
5     vertausche(x, y);
6     return  x < y;
```

Aufruf von $gOk(1, 2)$ (Fortsetzung):

- Der Anfangszustand (Nachfolgezustand von $\mathcal{S}_0$) von *vertausche* ist also $\mathcal{S}_1 = \{(x_i, W_{\mathcal{S}_0}(x_a), (y_i, W_{\mathcal{S}_0}(y_a))\}$, d.h. $\mathcal{S}_1 = \{(x_i, 1, (y_i, 2)\}$ (vor Zeile 1).
- Am Ende (nach Zeile 4) von *vertausche* ist offenbar der Zustand $\mathcal{S}_2 = \{(x_i, 2), (y_i, 1), (a, 1)\}$ erreicht (a ist die lokale Variable in *vertausche*.
- Nach Beendigung des Rumpfes von *vertausche* sind nun x_i und y_i aus *vertausche* in *gOk* nicht mehr sichtbar. Der Nachfolgezustand von $\mathcal{S}_2$ (nach Zeile 5) ist also $\mathcal{S}_3 = \{(x_a, 1), (y_a, 2)\}$.
- Rückgabe ist als $W(x_a < y_a)$, d.h. *TRUE*.

Parameterübergabe bei Prozeduraufrufen

- Hier noch Möglichkeit 2: *Call-by-reference*
  - Für jeden Eingabe-Parameter $e_1, \ldots, e_n$ wird im Methoden-Block *keine* neue Variable angelegt.
  - $e_1, \ldots, e_n$ erhalten stattdessen eine *Referenz* (*Verweis*) auf die übergebenen Variablen $x_1, \ldots, x_n$.
  - Wird innerhalb der Methode der Wert einer der Eingabe-Variablen $e_1, \ldots, e_n$ verändert, so wird also in Wirklichkeit eine der Variablen $x_1, \ldots, x_n$ aus p_u verändert, obwohl diese im Rumpf von p_a nicht sichtbar sind.
  - Das hat dann offenbar auch Auswirkungen außerhalb der Methode!
- Achtung: call-by-reference ist daher eine potentielle Quelle unbeabsichtigter Seiteneffekte!!!

Parameterübergabe: Call-by-reference

```
    PROCEDURE vertausche(x : ℕ, y : ℕ)
        BODY
1           var  a : ℕ;
2           a ← x;
3           x ← y;
4           y ← a;
```

```
    PROCEDURE gOk(x : ℕ, y : ℕ) → 𝔹
        BODY
5           vertausche(x, y);
6           return  x < y;
```

Aufruf von $gOk(1, 2)$ (mit call-by-reference):

- Beim Aufruf von *vertausche*(x, y) mit $(x, 1)$ und $(x, 2)$ in Zeile 5 werden für die Eingabeparameter von *vertausche* Referenzen auf die ursprünglichen Zettel angelegt.
- Mit unserer vorherigen Unterscheidung referenziert x_i nur x_a (und y_i entspr. x_i), d.h. x_i und x_a bezeichnen letztlich den selben Zettel (Speicherzelle).
- Im Rumpf von *vertausche* werden also jetzt tatsächlich die Werte der Zettel x und y vertauscht.
- Am Ende (Zeile 6) wird dann natürlich *FALSE* zurück gegeben.

Parameterübergabe: Call-by-value/-reference

- Tatsächlich ist call-by-reference in einigen Programmiersprachen möglich, z.B. in C durch die Verwendung sog. *Pointer* (vergessen Sie es schnell wieder, das ist häßliches Programmieren — hä??? Warum gibt es diese Pointer bzw. call-by-reference?).
- Übrigens wird uns call-by-reference noch verfolgen:
- Java stellt neben den atomaren Datentypen auch Referenz-Typen (z.B. Arrays) bzw. Objekt-Typen (z.B. benutzereigene Datentypen durch Klassen) zur Verfügung, deren Variablen anders behandelt werden und dabei zu einem call-by-reference Effekt führen.
- Dazu aber später mehr.

Parameterübergabe: Call-by-value/-reference

- Aber Moment mal: Mit call-by-value haben Prozeduren vielleicht Seiteneffekte, aber diese Seiteneffekte sind außerhalb der Prozeduren nicht bemerkbar.
- D.h., wenn eine Prozedur keine Rückgabe liefert, dann ist sie für uns momentan eigentlich nutzlos???
- Abgesehen von externen Effekten wie Ausgabe, etc. stimmt das aktuell (ändert sich aber mit Referenztypen).
- Wenn zwei Prozeduren (Algorithmen) gemeinsame Daten verändern sollen (was übrigens schlechter Stil ist), brauchen wir aktuell noch etwas anderes, nämlich sog. globale Variablen/Konstanten (siehe später).

- Wie bereits erwähnt ergänzen sich funktionale und imperative Konzepte wie folgendes Beispiel zeigt:

MODULE BEWEGUNGFI

OPS

FUNCTION *strecke*$(m : \mathbb{R}, t : \mathbb{R}, k : \mathbb{R}) \to \mathbb{R}$
PRE $m > 0, t \geq 0$
BODY $k \cdot t^2/(2 \cdot m)$

FUNCTION *endgeschwindigkeit*$(m : \mathbb{R}, t : \mathbb{R}, k : \mathbb{R}) \to \mathbb{R}$
PRE $m > 0, t \geq 0$
BODY $(k/m) \cdot t$

PROCEDURE *arbeit*$(m : \mathbb{R}, k : \mathbb{R}, t : \mathbb{R}) \to \mathbb{R}$
PRE $m > 0, t \geq 0$
BODY
VAR $s : \mathbb{R}$;
$s \leftarrow$ *strecke*(m, t, k);
RETURN $k \cdot s$;