
Vorlesung „Methoden des Software Engineering“

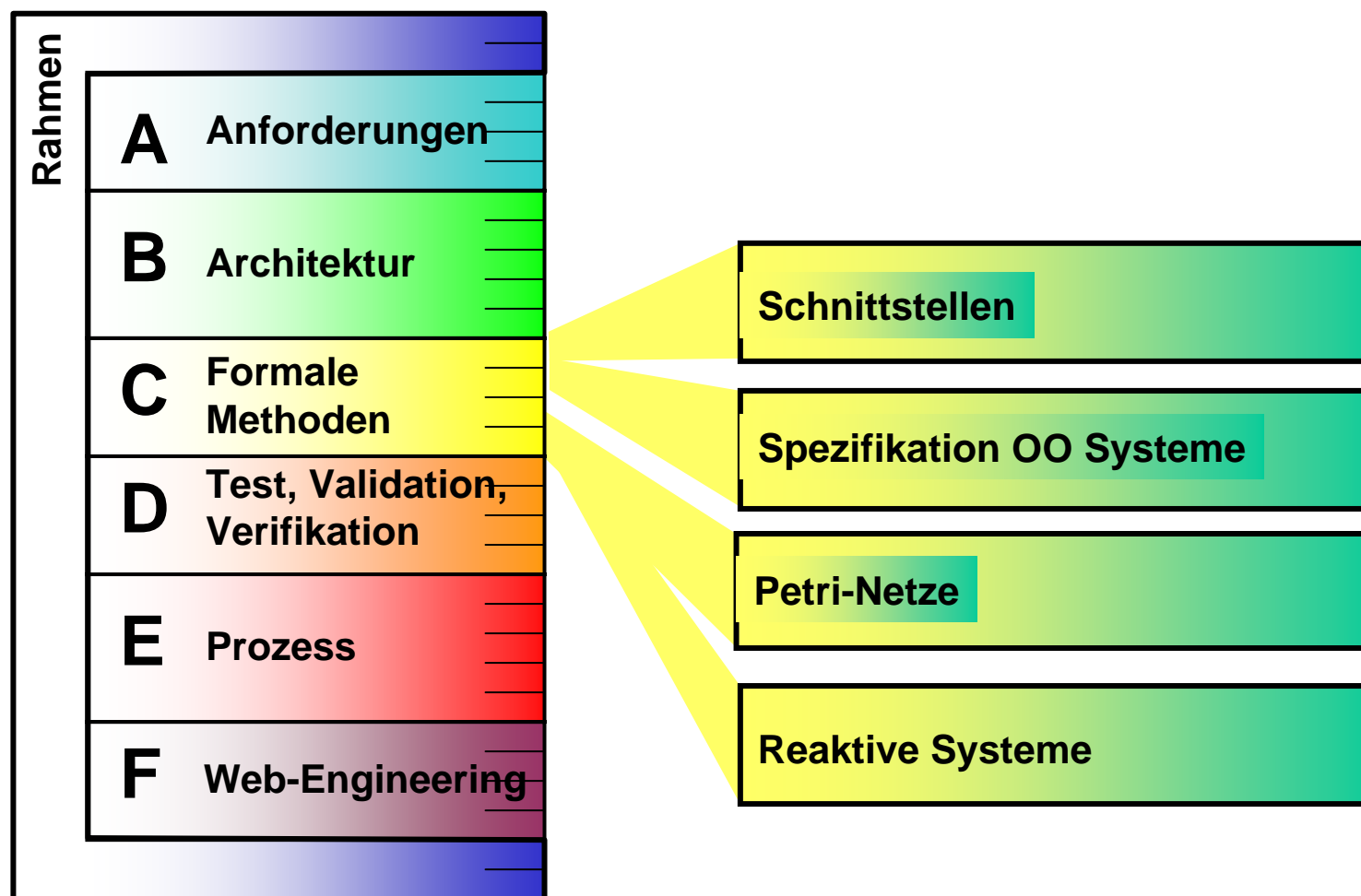
Block C „Formale Methoden“

Schnittstellen und deren Interpretation

Martin Wirsing

Einheit C.1, 28.11.2006

Gliederung Block C



Ziele

- Die Rolle formaler Methoden in der Software-Entwicklung verstehen lernen
- (Die Syntax von) Schnittstellen durch Signaturen beschreiben lernen
- Die Interpretation von Schnittstellen durch
Algebren und Transitionssysteme
kennen lernen

Formale Methoden

Verwendung von mathematischen Notationen zur
Beschreibung von
Anforderungs- und Entwurfsspezifikationen
zusammen mit
Validierungs- und Verifikationstechniken

Geschichtlicher Überblick

ab ca. 1960: **formale Sprachen und Automaten**; Petri-Netze

ab ca. 1970: **Semantik von Programmiersprachen** (Scott, Strachey):

ab ca. 1969: **Beweise von Programmen, Zusagekalküle**

(Dijkstra, Hoare, Floyd)

ab ca. 1970: **Formale Programmentwicklung** (CIP, Burstall/ Darlington, Hoare)

ab ca. 1980: **Temporale Logik** (Kröger, Manna, Pnueli)

zur Beschreibung des dynamischen Verhaltens von Systemen.

ab ca. 1990: **Integration diagrammatischer Notationen**

mit formalen Techniken,

ab ca. 1995: große Erfolge der Temporallogik durch Einsatz von **Modelchecking**

Signaturen zur Schnittstellenbeschreibung

Zur Angabe der Schnittstelle eines Systems benötigt man die Angabe der

- Namen der nach außen sichtbaren Datentypen („Sorten“, „Typen“)
- Namen und Typ der nach außen sichtbaren Funktionen („Funktionssymbole“)

Definition

Eine (**algebraische**) *Signatur* Σ ist ein Paar (S, F) mit

S eine Menge von Sorten, d.h. Namen für Mengen.

F eine $S^* \times S$ –sortierte Familie von Funktionssymbolen,

wobei $\langle s_1, \dots, s_n \rangle \in S^*$ den Definitionsbereich,

$s \in S$ den Wertebereich der Funktionen aus

$F_{\langle \langle s_1, \dots, s_n \rangle, s \rangle}$ bezeichnet.

Beispiel: Boolesche Werte

sig BOOL0 =

 sort Bool;

 ops true : Bool;

 false : Bool;

 not_ : Bool \rightarrow Bool;

 and, _or_ , _implies_ : Bool \times Bool \rightarrow Bool;

end

Beispiel: Keller

```
sig STACK0 =  
  sorts   Stack , Nat  
  ops     empty : Stack  
          push  : Stack × Nat → Stack  
          top   : Stack → Nat  
          pop   : Stack → Stack
```

Bemerkung: In der funktionalen Sichtweise werden Zustände explizit durch Terme dargestellt.

Beispiel:

Der Zustand z eines Kellers wird explizit dargestellt mit Hilfe von `push`:

Der Term $\text{push}(\text{push}(\text{empty}, 1), 2)$

repräsentiert einen zwei-elementigen Keller.

Zustandsbasierte Signaturen

Bei einem zustandsbasierten (prozeduralen) System ist der Zustand implizit und erscheint nicht mehr als Argument in der Funktionalität der Operationen. Als Ergebnistyp wird ggf. der triviale Typ Void verwendet.

Beispiel Keller:

Die Operationen erhalten folgende Typen:

```
ops    empty : Void
       push  : Nat → Void
       top   : Nat
       pop   : Void
```

Objektorientierte Signaturen

Bei objektorientierten Signaturen kommt der Objektbezeichner als erstes Argument hinzu. (Void bezeichne den trivialen Typ.)

Beispiel Keller:

sig STACKOO =

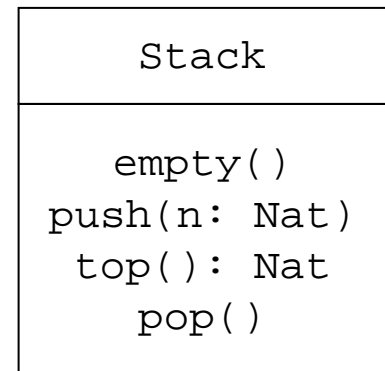
sorts Stack , Nat

ops empty : Stack \rightarrow Void

push : Stack \times Nat \rightarrow Void

top : Stack \rightarrow Nat

pop : Stack \rightarrow Void



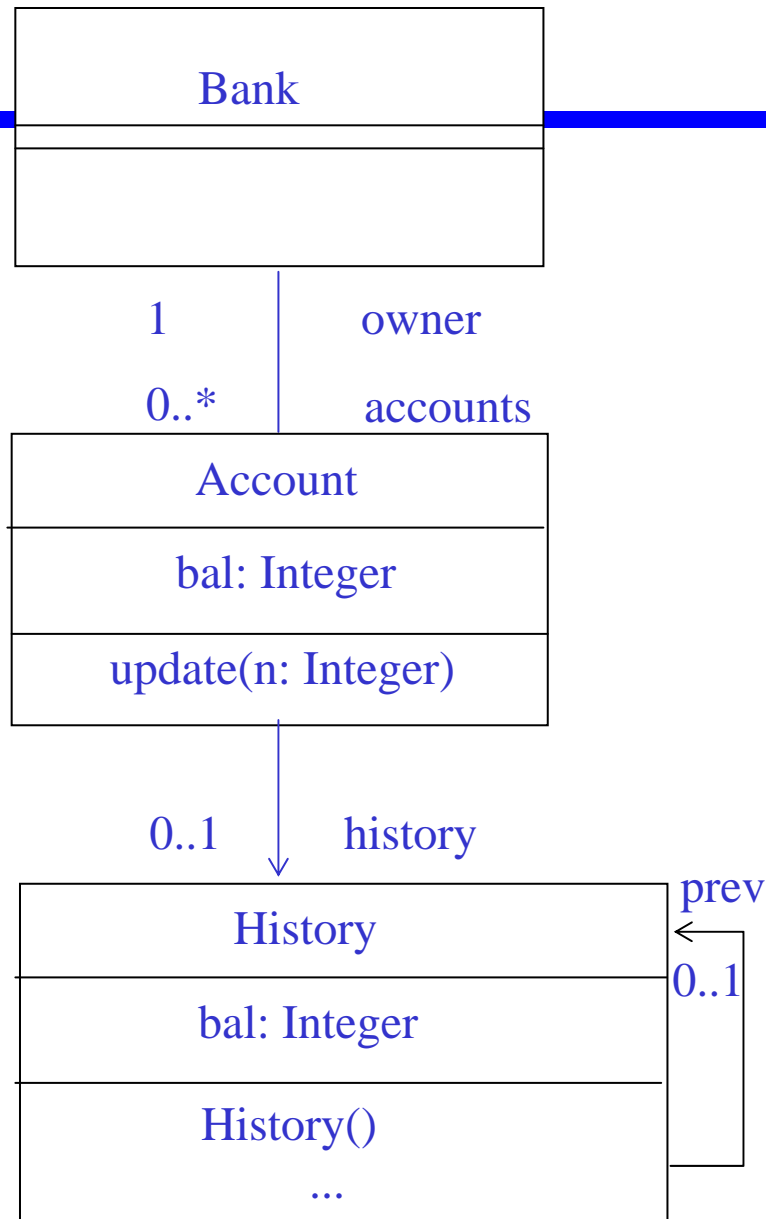
Klassendiagramm

- Ein (einfaches) **Klassendiagramm** in UML besteht aus Klassen und Assoziationen.

Für die Einbeziehung von Vererbung siehe FOOSE.

- Die **Signatur** eines Klassendiagramms besteht aus Basistypen, allen Klassen des Diagramms und Typen für endliche Kollektionen (Mengen, Sequenzen, Multimengen) sowie aus Operationen für die Attribute, Assoziationsrollen, Methoden und Konstruktoren.

Beispiel: Bank



Signatur (Ausschnitt)

sorts Bank. Account, History, Integer,
Set(Account)

ops $_.\text{owner}: \text{Account} \rightarrow \text{Bank}$
 $_.\text{accounts}: \text{Bank} \rightarrow \text{Set}(\text{Account})$
 $_.\text{bal}: \text{Account} \rightarrow \text{Integer}$
 $\text{update}: \text{Account} \times \text{Integer} \rightarrow \text{Void}$
 $_.\text{history}: \text{Account} \rightarrow \text{History}$
 $_.\text{bal}: \text{History} \rightarrow \text{Integer}$
 $_.\text{prev}: \text{History} \rightarrow \text{History}$
 $\text{History}: \rightarrow \text{History}$
 ...

Klassensignatur eines Klassendiagramms Δ

Sorten sind alle Basistypen (Void, Integer, Boolean, ...),

alle in Δ auftretenden Klassen, und

Set(C), Seq(C), Bag(C) für alle Klassen C aus Δ

Operationen

$_.a: C \rightarrow D$ für jedes Attribut von C und

jede von C nach D gerichtete Assoziation mit Multiplizität 1 oder 0..1

$_.a: C \rightarrow \text{Set}(D)$ für jede von C nach D gerichtete Assoziation

mit Multiplizität *

$m: C \times T_1 \times \dots \times T_n \rightarrow \text{Void}$

für jede Methode $m(x_1:T_1, \dots, x_n: T_n)$ der Klasse C

$m: C \times T_1 \times \dots \times T_n \rightarrow T$

für jede Methode $m(x_1:T_1, \dots, x_n: T_n): T$ der Klasse C

$C: T_1 \times \dots \times T_n \rightarrow C$

für jeden Konstruktor $C(x_1:T_1, \dots, x_n: T_n)$ der Klasse C

Schnittstelleninterpretation durch Strukturen und Transitionssysteme

Eine Σ – Algebra besitzt zu

- jeder Sorte eine Trägermenge,
- jedem Funktionssymbol eine Funktion.

Definition

Eine Σ -*Algebra* A besteht aus

- einer Familie $(A_s)_{s \in S}$ von **nichtleeren Trägermengen**, und
- **(totalen) Funktionen** $f^A: A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s \cup \{\perp\}$, für alle $f \in F_{\langle\langle s_1, \dots, s_n \rangle\rangle, s}$

Bemerkung

Enthält die Signatur Σ auch Prädikatensymbole, so spricht man von Σ -**Strukturen**.

Beispiel: Keller funktional

Interpretation von Kellern als Sequenzen natürlicher Zahlen:

Trägermengen: $\mathbf{S}_{\text{Stack}} = \mathbf{N}^*$ $\mathbf{S}_{\text{Nat}} = \mathbf{N}$

Funktionen: $\text{empty}^{\text{S}} = \langle \rangle$

$$\text{push}^{\text{S}}(\langle k_1, \dots, k_i \rangle, n) = \langle k_1, \dots, k_i, n \rangle,$$

$$\text{top}^{\text{S}}(\langle k_1, \dots, k_i \rangle) = \begin{cases} k_i, & \text{falls } i > 0 \\ \perp, & \text{falls } i = 0 \end{cases}$$

$$\text{pop}^{\text{S}}(\langle k_1, \dots, k_i \rangle) = \begin{cases} \langle k_1, \dots, k_{i-1} \rangle, & \text{falls } i > 0 \\ \perp, & \text{falls } i = 0 \end{cases}$$

Zustand

Zur Interpretation prozeduraler und objektorientierter Signaturen benötigt man den Begriff des Zustands.

Definition

- 1) Ein **Zustand eines prozeduralen (zustandsbasierten) Systems** ist gegeben durch die Interpretation einer (festen) Menge von Variablen.
- 2) Ein **Zustand eines objektorientierten Systems** ist gegeben durch ein Objektdiagramm.

Zustand (Prozedural)

Gegeben sei eine Menge von Sorten S und zugehörige Trägermengen A_s für alle $s \in S$.

1) Eine **Zustandssignatur** ist eine

Familie von (System-) Variablen $(X_s)_{s \in S}$

mit X_s abzählbar für alle $s \in S$.

2) Ein **Zustand** ist eine Belegung

$\beta: X_s \rightarrow A_s$ für alle $s \in S$.

Beispiel Keller:

$S = \{ \text{Stack}, \text{Nat} \}$ $A_{\text{Stack}} = \mathbb{N}^*$, $A_{\text{Nat}} = \mathbb{N}$

Systemvariable: $X_{\text{Stack}} = \{ \text{stack} \}$, $X_{\text{Nat}} = \{ \text{result} \}$

Ein Zustand ist eine Belegung $\beta: X_{\text{Stack}} \rightarrow A_{\text{Stack}}$, $\beta: X_{\text{Nat}} \rightarrow A_{\text{Nat}}$

Z.B. $\beta_1(\text{stack}) = \langle 17, 12, 3 \rangle$ oder nur $\beta_2(\text{stack}) = \langle 7, 10, 3, 11 \rangle$

$\beta_1(\text{result}) = 3$ wenn result unwichtig

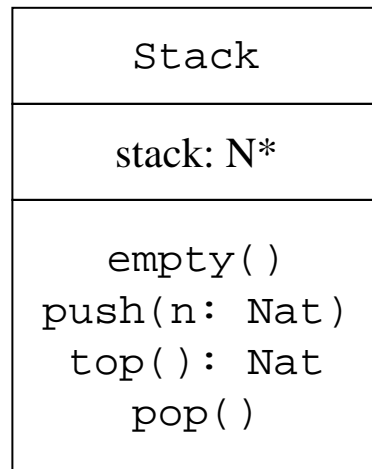
Zustand (Objektorientiert)

Ein (objektorientierter) **Systemzustand** besteht aus

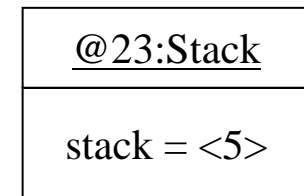
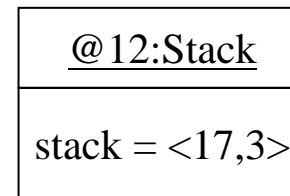
1. einer Menge von aktuell existierenden Objekten (für jede Klasse des Systems) und
2. einer Belegung aller Instanzvariablen, d.h. einer Funktion, die jeder Instanzvariable $o.a$ (für jedes Objekt o und Attribut a) einen Wert zuweist.
3. (eventuell zusätzlich) einer Belegung der lokalen/freien Variablen des aktuell betrachteten Ausdrucks oder Programms.

Beispiel Keller mit zwei Stack-Objekten

Klassendiagramm:



Objektdiagramm



Formale Beschreibung des Objektdiagramms:

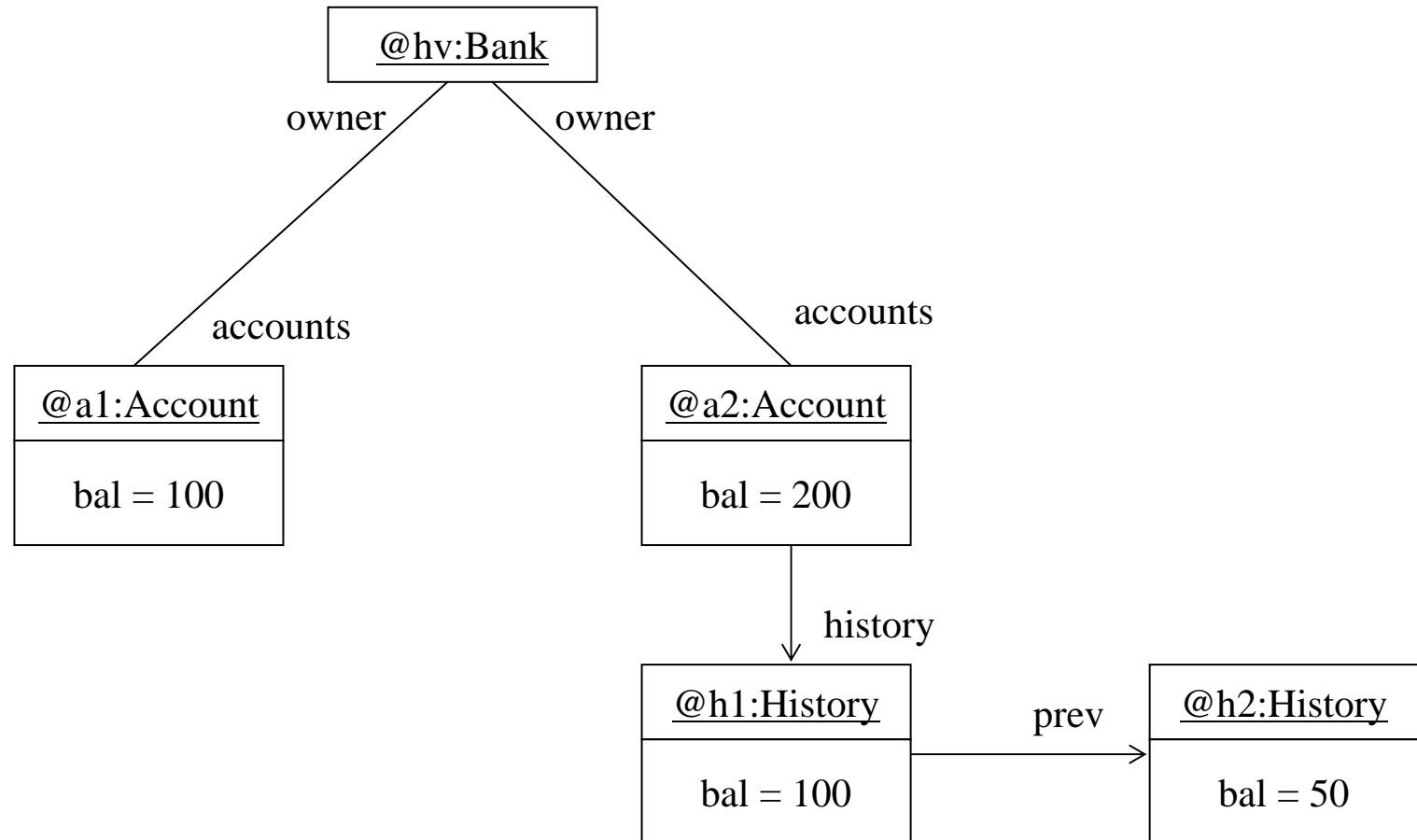
Existierende Objekte

@12, @23

$\sigma(@12.stack) = \langle 17, 3 \rangle$

$\sigma(@23.stack) = \langle 5 \rangle$

Beispiel: Zustand des Banksystems



Transitionssystem für Keller (prozedural)

Ein markiertes **Transitionssystem** gibt die Beziehung zwischen dem Zustand vor und dem Zustand nach Ausführung einer Operation an;
jede **Transition** wird mit dem passenden Methodenaufruf markiert.

Markierungen: $\text{pop}()$, $\text{top}()$, $\text{push}(n)$ für $n \in \mathbb{N}$

Transitionen:

$\text{stack} = s$	$\xrightarrow{\text{push}(n)}$	$\text{stack} = s^\circ \langle n \rangle$ für alle $n \in \mathbb{N}$
$\text{stack} = s^\circ \langle n \rangle$	$\xrightarrow{\text{pop}()}$	$\text{stack} = s$
$\text{stack} = s^\circ \langle n \rangle$	$\xrightarrow{\text{top}()}$	$\text{stack} = s^\circ \langle n \rangle$, $\text{result} = n$

Rückgabewert

wobei wir kurz $\text{stack} = s$ für den Zustand $\beta + [\beta(\text{stack}) = s]$ schreiben

und

$$\beta + [x = v](y) = \begin{cases} v & \text{falls } x=y \\ \beta(y) & \text{sonst} \end{cases}$$

Transitionssystem

Gegeben sei eine Signatur $\Sigma = (S, F)$.

1) Ein Transitionssystem

$$\Gamma = (Z, T)$$

ist gegeben durch

- * eine Menge Z von Zuständen und
- * eine Transitionsrelation $T \subseteq Z \times Z$.

2) Ein markiertes Transitionssystem

$$\Gamma = (Z, A, T)$$

ist gegeben durch

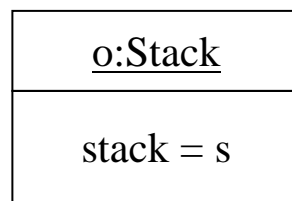
- * eine Menge Z von Zuständen
- * eine Menge A von Aktionen und
- * eine Transitionsrelation $T \subseteq Z \times A \times Z$.

Transitionssystem für Keller (objektorientiert)

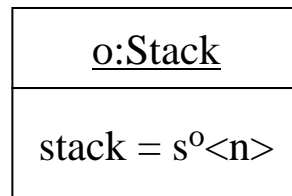
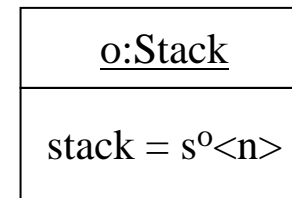
Markierungen: $o.pop()$, $o.top()$, $o.push(n)$

für Stackobjekte o und $n \in \mathbb{N}$

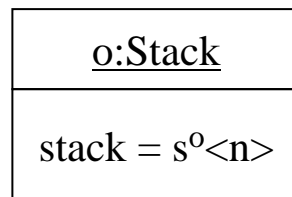
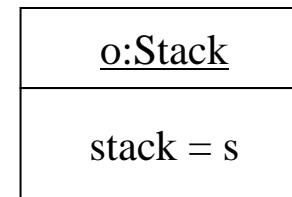
Transitionen:



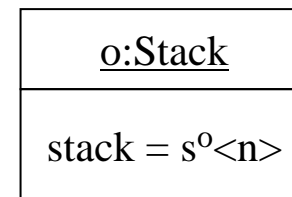
$o.push(n)$ →



$o.pop()$ →



$o.top()$ →



, result = n

Transitionssystem für Keller (objektorientiert)

Bemerkung:

- Die Regeln der vorhergehenden Seite sind Graphersetzungsgesetze.
- Die Transitionsrelationen gelten für alle Objektdiagramme, die die linke Seite einer Regel enthalten.
- Die rechte Seite des Gesamtobjektdiagramms entsteht durch Ersetzung der linken Seite der Regel durch die rechte Seite der Regel.

Transitionssystem für Keller (objektorientiert)

Genauer formuliert man die Transitionsrelation mathematisch:

Für alle Zustände σ gelte:

$$\sigma + [\text{o.stack} = s] \xrightarrow{\text{o.push}(n)} \sigma + [\text{o.stack} = s^\circ \langle n \rangle]$$

$$\sigma + [\text{o.stack} = s^\circ \langle n \rangle] \xrightarrow{\text{o.pop}()} \sigma + [\text{o.stack} = s]$$

$$\sigma + [\text{o.stack} = s^\circ \langle n \rangle] \xrightarrow{\text{o.top}()} \sigma + [\text{o.stack} = s^\circ \langle n \rangle], \text{ result} = n$$

Dabei bedeutet $\sigma + [\text{o.a} = v]$, dass in σ an der Stelle o.a geändert wurde und der Wert von o.a gleich v ist:

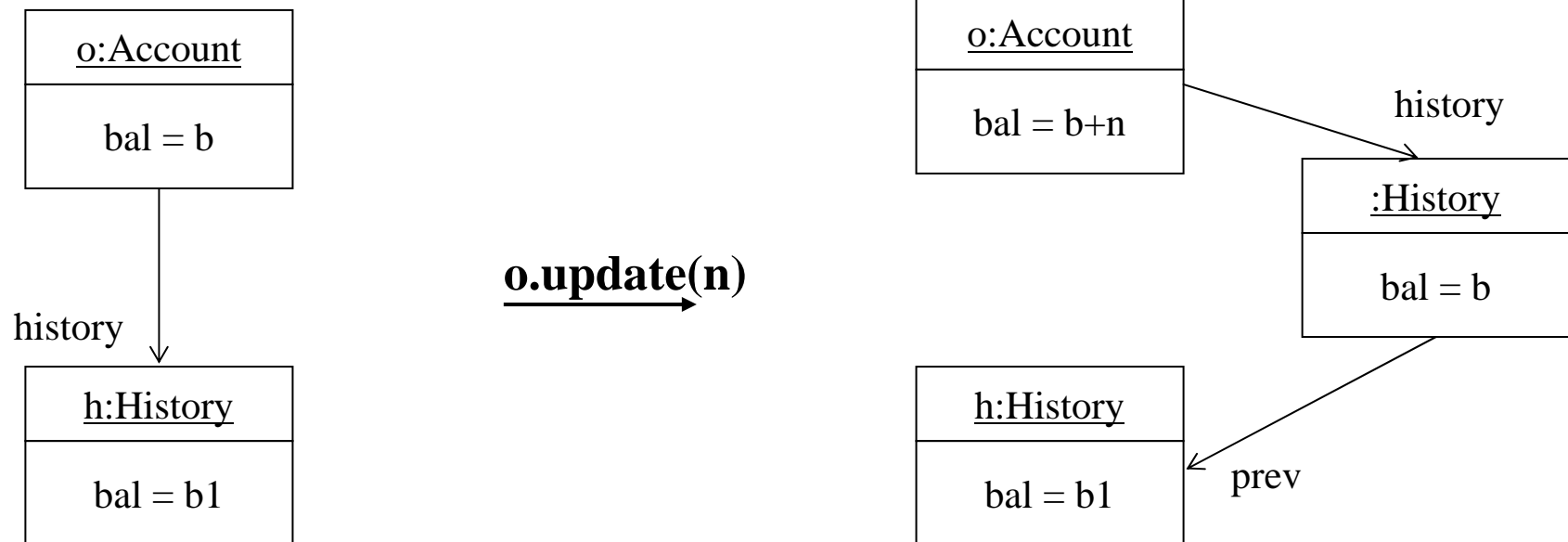
$$\sigma + [\text{o.a} = v](\text{p.b}) = \begin{cases} v & \text{falls } \text{o} = \text{p} \text{ und } \text{a} = \text{b} \\ \sigma(\text{p.b}) & \text{sonst} \end{cases}$$

Transitionssystem für Bank (objektorientiert)

Markierungen: $o.update(n)$, für Accountobjekte o , $n \in \mathbb{N}$

Transitionen:

Sei $b+n \geq 0$. Dann gelte:



Zusammenfassung

- Eine Signatur dient zur syntaktischen Beschreibung von Schnittstellen.
- Die funktionale Interpretation von Schnittstellen wird durch Algebren und Strukturen beschrieben.
- Die Interpretation prozeduraler und objektorientierter Schnittstellen wird durch Transitionssysteme beschrieben.

Literatur

- Wirsing: Grundlagen der Systementwicklung, Kap. 2.2 und 3.6
- Hennicker: Formale objektorientierte Software-Entwicklung,
Kap 2.1, 2.6 und 4.1
- Ehrig, Mahr, Cornelius, Große-Rohde, Zeitz: Mathematisch-
strukturelle Grundlagen der Informatik, Springer, 1999,
Kap. 7, Signaturen und Algebren