universität innsbruck
Institut für Informatik

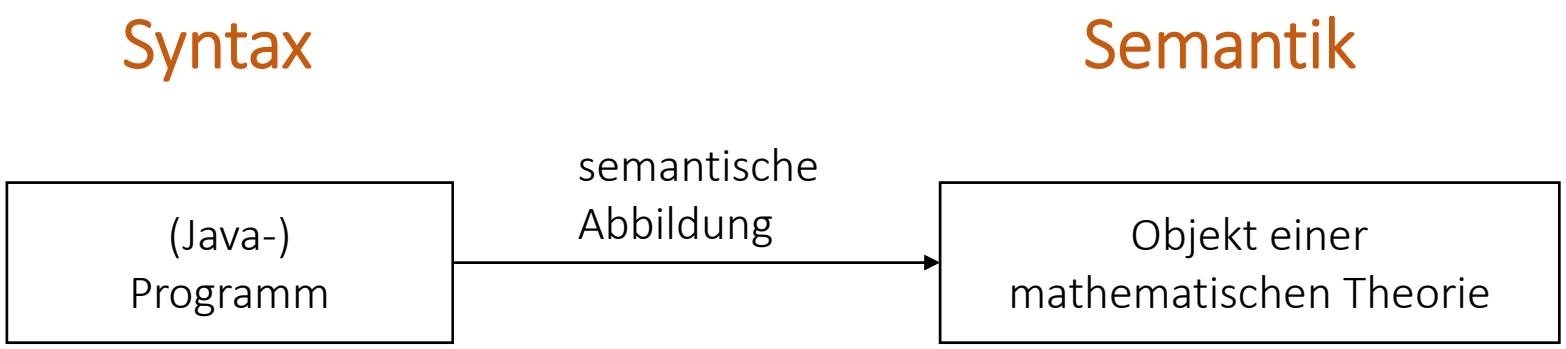
Software Engineering

5. Formale Spezifikation
Teil 1: Sequentielle Systeme

Ruth Breu

Übersicht

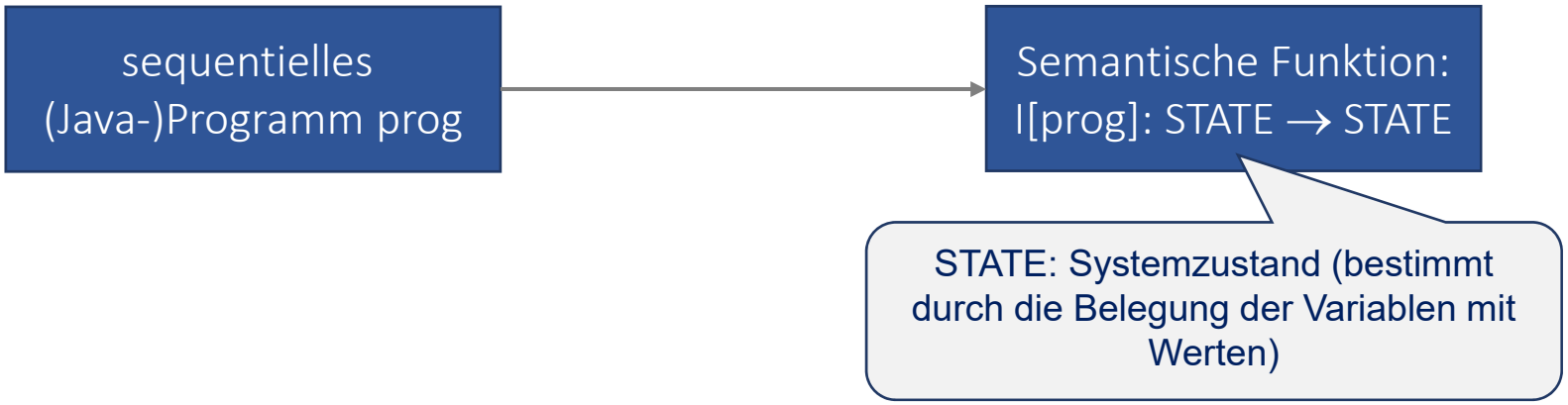
5.1 Grundbegriffe



Gründe der Definition einer Semantik

- Tiefgehendes Verständnis der Sprachkonstrukte gewinnen
- Beweis von allgemeinen Eigenschaften von Programmen
  Beispiel: Was bedeutet Nicht-Terminierung,
  Nicht-Determinismus?
- Gemeinsame Basis für Programmierung und Spezifikation

Semantik sequentieller Systeme



Die semantische Funktion beschreibt die Zustandsänderung, die die Ausführung eines Programms hervorruft

Beispiel

prog ≡ int x, y, z;
z = x;
x = y;
y = z;

Zustand (Beispiel): σ_0 = [x → 3, y → 5, z → 2]

σ_0[x] = 3 *(„die Belegung der Variable x in Zustand σ ist 3“)*

Semantische Funktion:

I[prog]: STATE → STATE

I[prog] (σ) = σ [x → σ[y], y → σ[x], z → σ[x]]

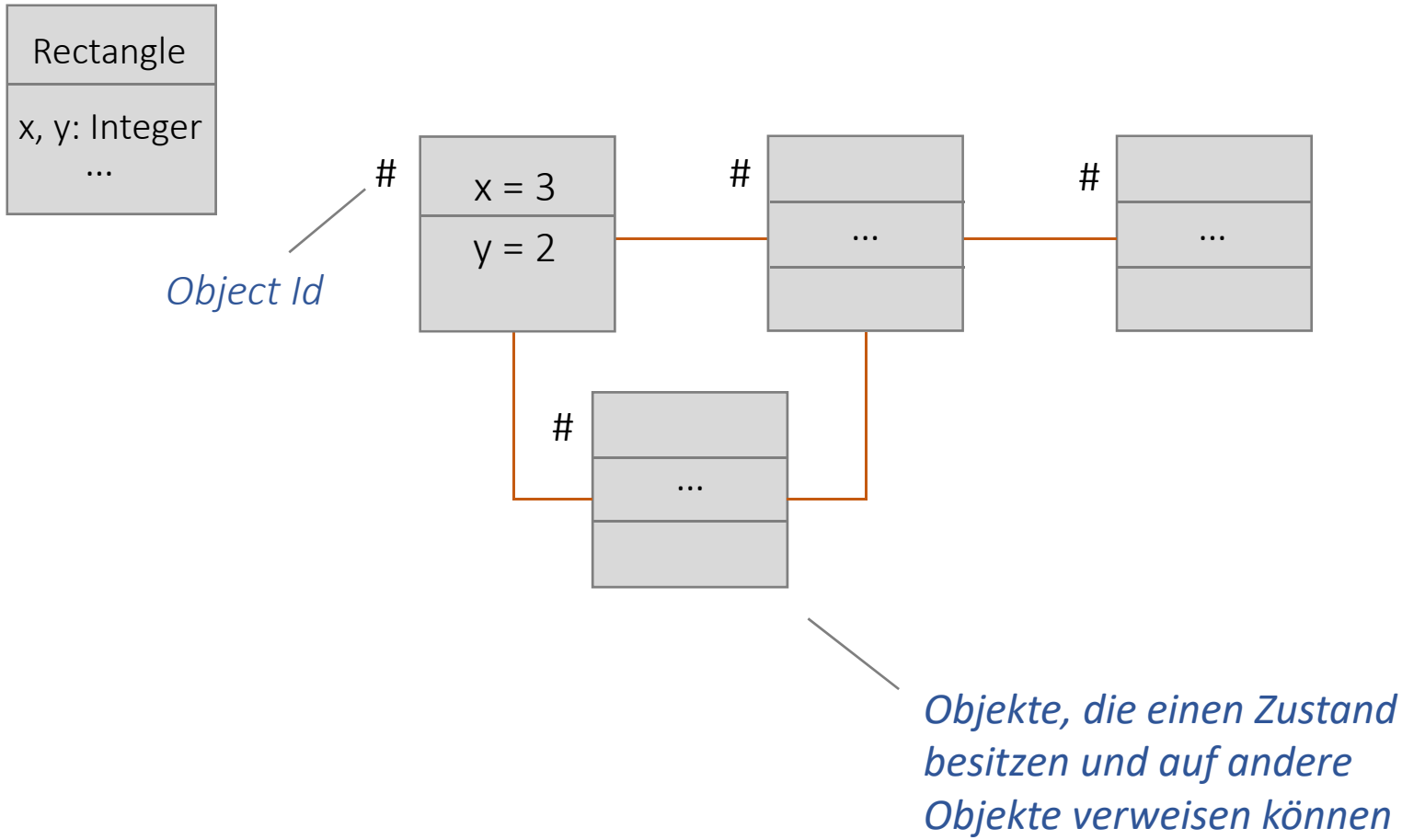
„der Wert der Variable x ist im neuen Zustand der Wert der Variable y im alten Zustand“

Beispiel:

I[prog] ([x → 3, y → 5, z → 2]) =

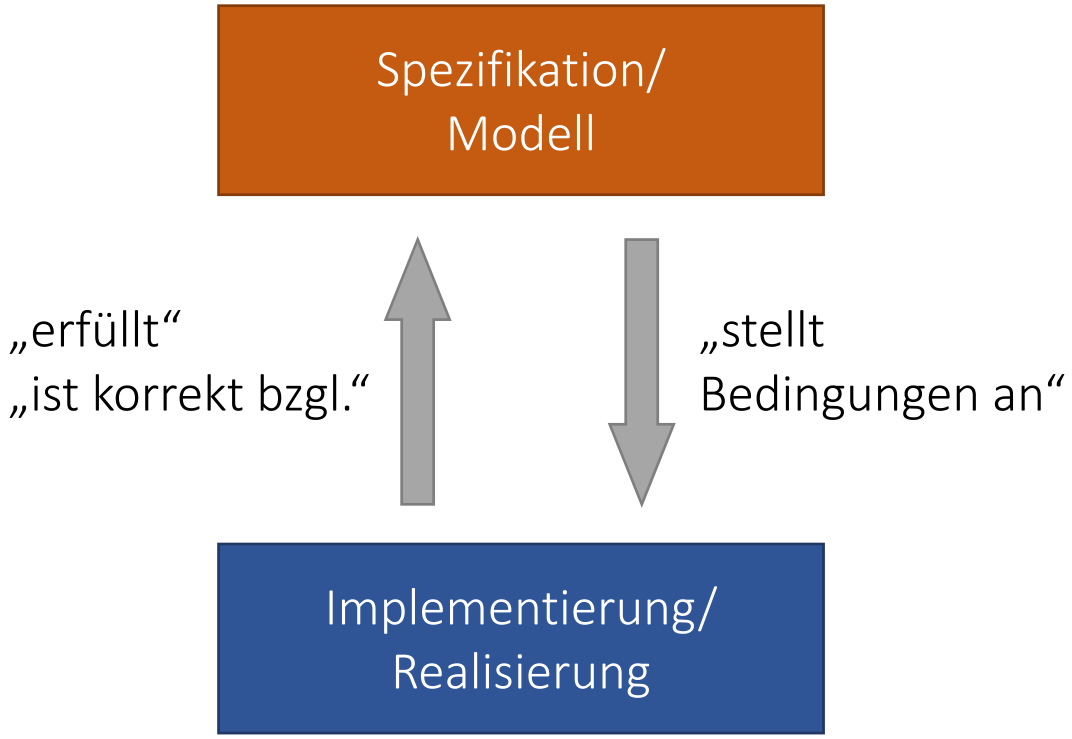
= [x → 5, y → 3, z → 3]

Zustand eines objektorientierten Programms

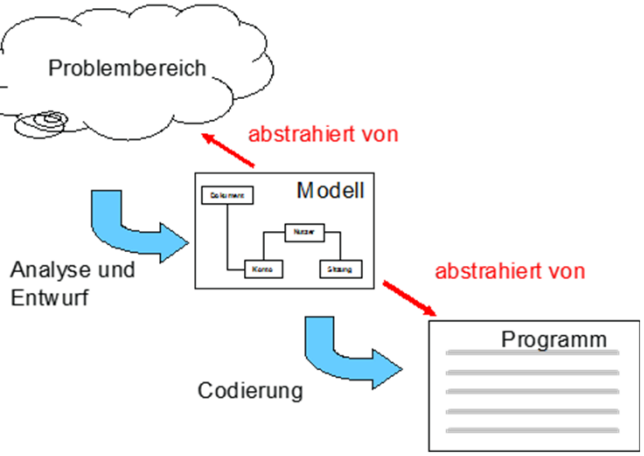


Spezifikation

- Eine Spezifikation beschreibt die Eigenschaften eines Systems, sie ist ein **Modell** des Systems



Modellbegriff



- Pragmatisches Modell
  - Ziel: Kommunikation vereinfachen, Anforderungen dokumentieren
- Konstruktives Modell („Model Engineering“)
  - Ziel: Ausführbarkeit, Generierung von Code
- Formales Modell („Formale Spezifikation“)
  - Ziel: Formaler bzw. automatisierter Beweis von Systemeigenschaften

Übersicht Formale Spezifikationstechniken

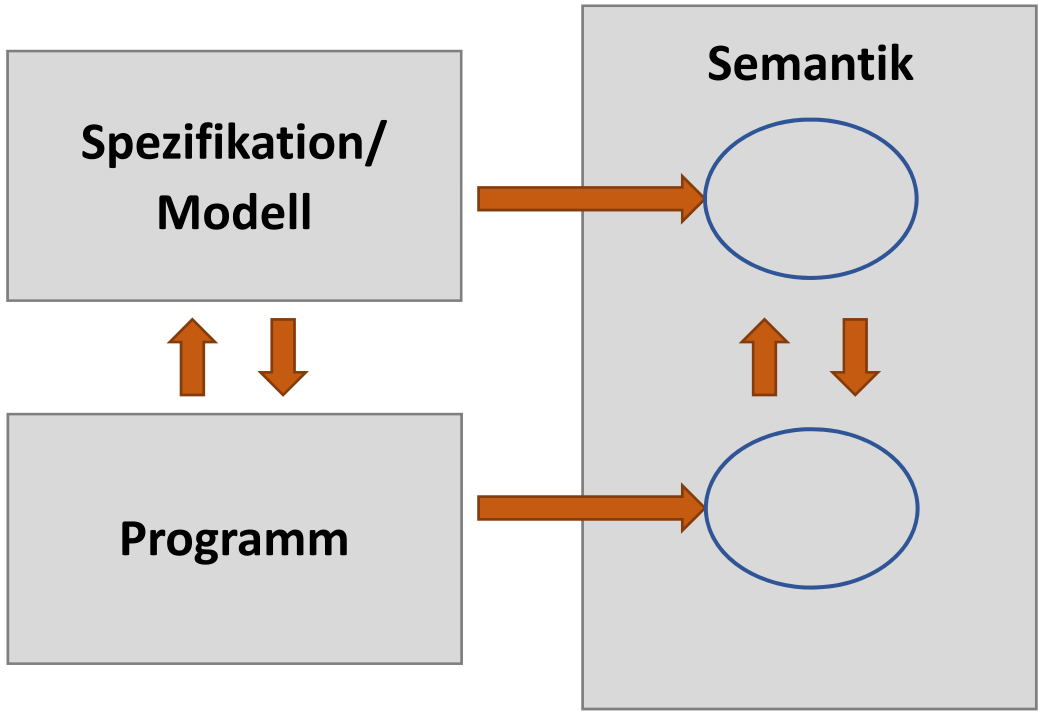
Spezifikationstechniken für sequentielle Systeme (Beispiele)

- Hoare-Logik – Spezifikation von Zuständen (-> 5.2.1)
- Z – Spezifikation von Vor-/Nachbedingungen, Invarianten
- OCL – ausführbare Spezifikationen (-> 5.2.2)

Spezifikationstechniken für verteilte Systeme (Beispiele)

- Petrinetze – graphikorientiert (-> 5.3.3)
- Statecharts – graphikorientiert (-> Kapitel 6)
- Temporale Logik
- Prozessalgebren – logikorientiert (-> 5.3.4)

Zusammenhang Spezifikation/Semantik/ Programm



Einsatz formaler Techniken

- Formale Spezifikationstechniken werden in der Praxis meist in sicherheitskritischen Bereichen angewandt
- Sicherheit = Safety oder Security
  - Safety = Schutz vor gefährlichen Fehlern technischer Systeme
  - Security = Schutz vor zielgerichteten Angriffen von innen und außen
- Beispiele:
  - Korrektheitsbeweise für Stellwerksteuerungen
  - Korrektheitsbeweise für Security Protokolle, z.B. zur Authentifizierung in verteilten Systemen

5.2 Spezifikation Sequentieller Systeme

5.2.1 Hoare-Kalkül

5.2.2 OCL – Object Constraint Language

5.2.1 Hoare-Kalkül

- Was berechnet das folgende Programm?

```
int i = 7; int j = 4;
int p = 1;
int k = 0;
while (k < j) {
        p *= i;
        k++;
}
```

- Das Programm berechnet in `p` die `j`-fache Potenz von `i`
- Genauer: Am Ende des Programms gilt:

$$p = i^j$$

Wie kann man das überprüfen?

- Testen = Stichproben nehmen
- Assert-Statement zur Laufzeit (nächste Folie) = Prüfungen zur Laufzeit
- Mathematischer Beweis?

Zusicherungen (Assertions) in Java - Beispiel

```
int i = 7; int j = 4;
int p = 1;
int k = 0;
while (k < j) {
        p *= i;
        k++;
}
assert k == j && p == Math.pow(i,k);
```

Assertions in Java

- Mit Zusicherungen kann man zur Laufzeit überprüfen, ob an einer Stelle des Programms eine Eigenschaft (d.h. ein Boolescher Ausdruck) gültig ist.
- Die Zusicherung (Überprüfung) einer Bedingung erfolgt mit

  ```
  assert <boolean expression>;
  ```

  Bedeutung:
  - Hat der Ausdruck den Wert `true`, wird das Programm fortgesetzt
  - Hat der Ausdruck den Wert `false`, wird ein Fehler vom Typ `AssertionError` geworfen
- Optional kann eine Fehlermeldung angegeben werden

  ```
  assert <boolean expression>: <String expression>;
  ```

Beweis von Zusicherungen

- Die Gültigkeit von Zusicherungen kann man mit dem Hoare-Kalkül beweisen.
- Informell nennt man ein Programm
  korrekt bzgl. seiner Vor- und Nachbedingung,
  wenn die Nachbedingung nach jeder Ausführung des Programms gilt, unter der Voraussetzung, dass vor Ausführung des Programms die Vorbedingung gegolten hat.

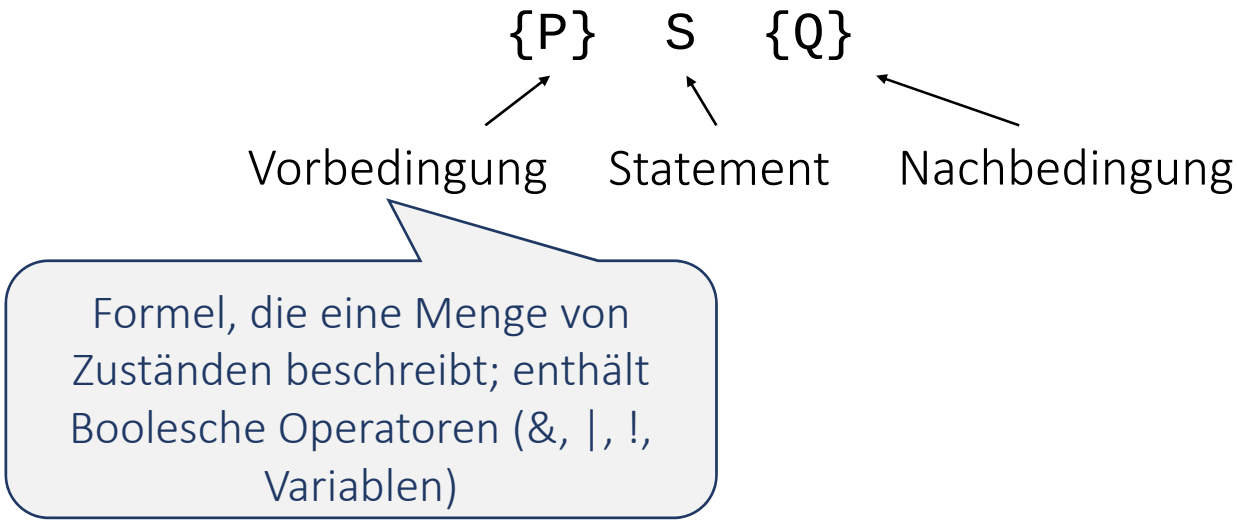
C.A.R. Hoare, *1934
Erfinder von Quicksort,
Hoare-Logik (1969),
Strukturierte Programmierung

Statements im Hoare-Kalkül

- Der Hoare-Kalkül verwendet eine imperative Kernsprache mit folgender BNF:

```
<statement> ::= <assign> | <sequ> | <cond> | <while>
<assign> ::= <var> = <exp>
<sequ> ::= <statement> ; <statement>
<cond> ::= if <bexp> then <statement> else <statement> fi
<while> ::= while <bexp> do <statement> od
```

Hoare-Formel



- Bedeutung von Hoare-Formeln bestimmt durch die Begriffe
  - totale Korrektheit und
  - partielle Korrektheit

Partielle Korrektheit

$$\{P\} \quad S \quad \{Q\}$$

ist gültig, wenn S partiell korrekt bzgl. Vorbedingung P und Nachbedingung Q ist, d.h. wenn folgendes gilt:

Wenn P im Anfangszustand von S gilt und wenn S terminiert, dann gilt Q nach Ausführung von S.

Totale Korrektheit

$$\{P\} \quad S \quad \{Q\}$$

ist gültig, wenn S total korrekt bzgl. Vorbedingung P und Nachbedingung Q ist, d.h. wenn folgendes gilt:

Wenn P im Anfangszustand von S gilt, dann terminiert S und gilt Q nach Ausführung von S.

- Totale Korrektheit = Partielle Korrektheit + Terminierung
- Für eine Anweisung S ohne Iteration stimmen totale und partielle Korrektheit überein.

Beispiele

- Totale und partielle Korrektheit

```
{true} if (y>0) then x=y else x=-y fi {x == |y|}
{x>=0} if (y>0) then x=y else x=-y fi {x>=0}
{x>1} x=x+1; y=x {x>2 & y>2}
{x>=0} while (x!=0) do x=x-1 od {x==0}
```

- Partielle Korrektheit, aber nicht totale Korrektheit

```
{true} while (x!=0) do x=x-1 od {x==0}
```

Nicht-Terminierung

- Die Hoare-Formel

```
{x>0} while (x>0) do x=x+1 od {false}
```

  ist partiell korrekt, aber nicht total korrekt.

- Allgemein: die Gültigkeit von `{P} S {false}` drückt Nichtterminierung aus, d.h.

  `{P} S {false}` partiell korrekt $\Rightarrow$
  S terminiert **nicht** für alle Anfangszustände, die P erfüllen.

Hoare-Kalkül

- Der Hoare-Kalkül dient zum (konstruktiven) Beweisen von partieller und totaler Korrektheit
- Idee: Leite rückwärts schreitend ausgehend von der gewünschten Nachbedingung die Vorbedingung ab
- Der Kalkül besteht aus einem Axiom und Ableitungsregeln
  - Axiom = ohne Voraussetzungen anzuwendende Regel (hier: für alle Zuweisungen anzuwendende Regel)
  - Ableitungsregel

$$\frac{\{b \;\&\; P\}\ S1\ \{Q\},\ \{!b \;\&\; P\}\ S2\ \{Q\}}{\{P\}\ \texttt{if}\ b\ \texttt{then}\ S1\ \texttt{else}\ S2\ \texttt{fi}\ \{Q\}}$$

Die Formel unten ist gültig,
wenn die beiden Formeln oben bewiesen wurden

Hoare-Regel Zuweisung

Zuweisungsaxiom

```
{P[exp/x]} x = exp {P}
```

Ersetze x in P durch exp

Beispiel

```
{max-5 == 35} max = max – 5 {max==35}
```

Hoare-Regel Abschwächung

Abschwächungsregel

$$\frac{P1 \Rightarrow P, \{P\}\ S\ \{Q\},\ Q \Rightarrow Q1}{\{P1\}\ S\ \{Q1\}}$$

Beispiel

Implikation ist gültig

Anwendung des Zuweisungsaxioms

Implikation ist gültig

$$\frac{n==3 \Rightarrow 2n>=6,\ \{2n>=6\}\ n=2*n\ \{n>=6\},\ n>=6 \Rightarrow n>0}{\{n==3\}\ n=2*n\ \{n>0\}}$$

Obere Zeile gilt,
deshalb gilt auch untere Zeile

Kurze Schreibweise

$$\frac{\{2n>=6\}\ n=2*n\ \{n>=6\}}{\{n==3\}\ n=2*n\ \{n>0\}}$$

Weitere Hoare-Regeln

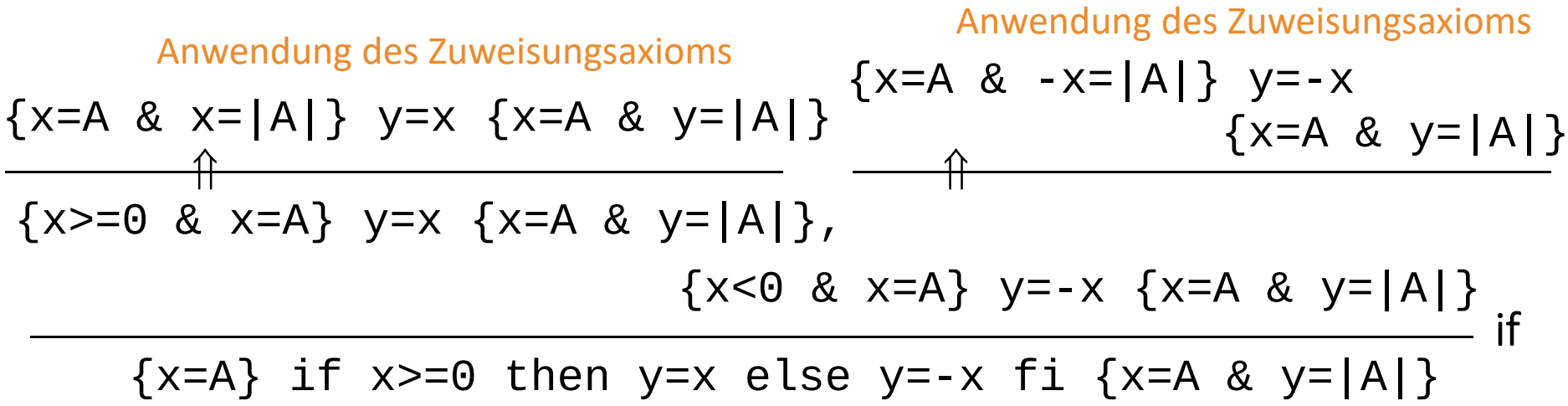
Fallunterscheidung

$$\frac{\{b \;\&\; P\}\ S1\ \{Q\},\ \{!b \;\&\; P\}\ S2\ \{Q\}}{\{P\}\ \texttt{if}\ b\ \texttt{then}\ S1\ \texttt{else}\ S2\ \texttt{fi}\ \{Q\}}$$

Sequentielle Komposition

$$\frac{\{P\}\ S1\ \{R\},\ \{R\}\ S2\ \{Q\}}{\{P\}\ S1;\ S2\ \{Q\}}$$

Beispiel



„Beweisbaum“,
Beweis wird von unten nach oben entwickelt
Blätter des Baums: Anwendung des Zuweisungsaxioms

Hoare-Regeln Iteration

Partielle Korrektheit

I = „Invariante“

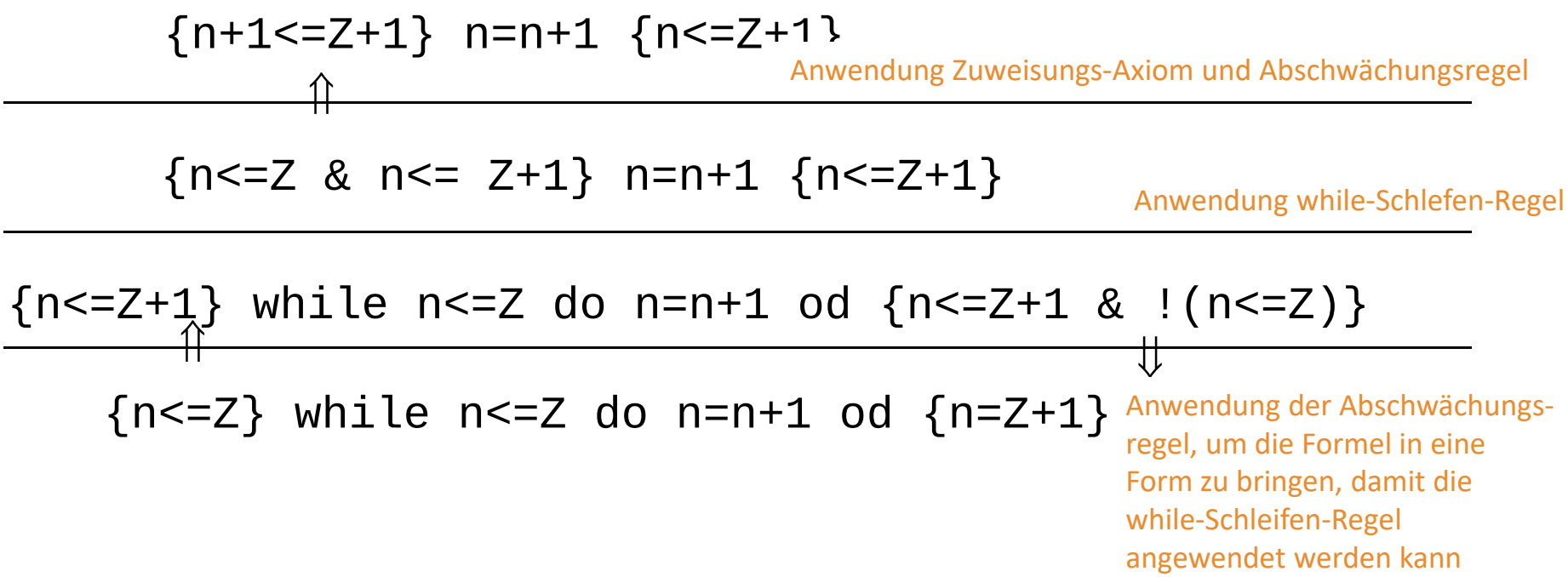
$$\frac{\{b \;\&\; I\}\ S\ \{I\}}{\{I\}\ \texttt{while}\ b\ \texttt{do}\ S\ \texttt{od}\ \{!b \;\&\; I\}}$$

Totale Korrektheit

$$\frac{\begin{array}{l}\{b \;\&\; I\}\ S\ \{I\} \\ \{b \;\&\; I \;\&\; t{=}{=}z\}\ S\ \{t<z\} \\ I \Rightarrow t >= 0\end{array}}{\{I\}\ \texttt{while}\ b\ \texttt{do}\ S\ \texttt{od}\ \{!b \;\&\; I\}}$$

`t` – ein Integer-Ausdruck für die Terminierung der while-Schleife
`z` – eine logische Variable, die nicht in `I`, `b`, `S` oder `t` vorkommt, also durch `S` nicht verändert wird

Beispiel – Partielle Korrektheit



Wir führen den Beweis mit folgender Invariante I durch: {n<=Z+1}

Beispiel – Totale Korrektheit

$$\frac{\{n+1<=Z+1\}\ n=n+1\ \{n<=Z+1\}}{\{n<=Z\ \&\ n<=\ Z+1\}\ n=n+1\ \{n<=Z+1\}}$$

(*) Terminierung

$$\frac{\{n<=Z+1\}\ \texttt{while}\ n<=Z\ \texttt{do}\ n=n+1\ \texttt{od}\ \{n<=Z+1\ \&\ !(n<=Z)\}}{\{n<=Z\}\ \texttt{while}\ n<=Z\ \texttt{do}\ n=n+1\ \texttt{od}\ \{n=Z+1\}}$$

(*) Terminierung: Sei t = Z+1-n

Teil {b & I & t==z} S {t<z} aus der Regel

$$\frac{\{Z+1-(n+1)<z\ \}\ n=n+1\ \{Z+1-n<z\}}{\{n<=Z\ \&\ n<=Z+1\ \&\ Z+1-n==z\ \}\ n=n+1\ \{Z+1-n<z\}}$$

Teil I ⇒ t >=0 aus der Regel

$$n<=Z+1 \Rightarrow Z+1-n >=0$$ Implikation ist gültig

Beweisskizzen

- Kompakte Darstellung für Beweise mit dem Hoare-Kalkül
- Eine Beweisskizze für partielle/totale Korrektheit ist ein mit Zusicherungen ergänztes Programm
- Annotationen:

```
{P} while b do {b & P} S {P} od {!b & P}

{P} if b then {b&P}  S1 {Q}
         else {!b&P} S2 {Q} fi
{Q}
```

{P} ⇒ {P1} S {Q1} ⇒ {Q} wobei P ⇒ P1 und Q1 ⇒ Q

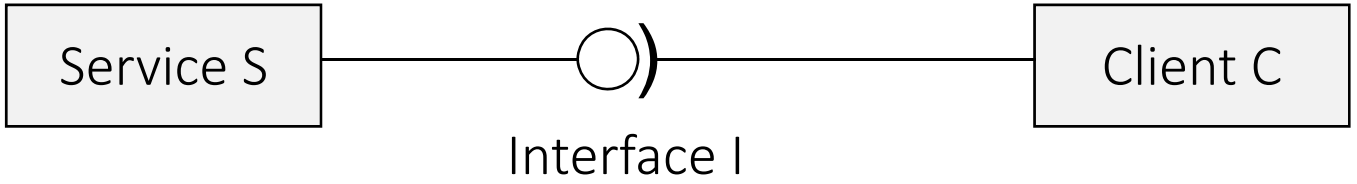
Beispiel

```
{x==A}
if x>=0
then
  {x>=0 & x==A} ⇒
  {x==A & x==|A|}
y=x
  {x==A & y==|A|}

else
  {x<0 & x==A} ⇒
  {x==A & -x==|A|}
y=-x
  {x==A & y==|A|}

fi
{x==A & y ==|A|}
```

Anwendung: Der Vertragsgedanke



- Die Spezifikation des Interfaces I stellt einen Vertrag zwischen C und S dar
  - C kann darauf vertrauen, dass die spezifizierten Eigenschaften eingehalten werden
  - S kann eine beliebige Implementierung bereitstellen, die die Spezifikation erfüllt

Spezifikation von Interfaces

- Die Methoden des Interfaces werden mit Vor- und Nachbedingungen verknüpft

  pre m **post**

  Falls vor Aufruf von m die Vorbedingung erfüllt ist, gilt nach Ausführung von m die Nachbedingung

  - Für das Einhalten der Vorbedingung ist der Aufrufer verantwortlich (z.B. durch Einhalten einer bestimmten Reihenfolge der Methoden)
  - Für das Einhalten der Nachbedingung ist der Aufgerufene durch korrekte Implementierung verantwortlich
- Robuste Komponenten stoßen eine Fehlerbehandlung an, wenn Vor- und Nachbedingungen nicht eingehalten werden!

Beschreibungsformen für Vor- und Nachbedingungen

- Formal
  - durch logische Formeln
  - Durch Boolesche Ausdrücke in der Programmiersprache (vgl. assert-Statements in Java)
- Informell durch Text
- In Klassendiagrammen durch OCL-Ausdrücke (vgl. Kapitel 5.2.2)

5.2.2 OCL - Object Constraint Language

- Kontext: Klassendiagramme und die Objektstrukturen, die durch sie beschrieben werden
- Klassendiagramme allein beschreiben nur die prinzipielle Struktur von Objektzuständen
- Oft sollen Eigenschaften von Objektstrukturen detailliert beschrieben werden:
  - Einschränkung der Objektzustände (Attributwerte)
  - Beispiel: *Die Dauer eines Flugs ist kürzer als 24 Stunden*
  - Beschreibung von Abhängigkeiten zwischen Attributen/Assoziationen
  - Beispiel: E*in Angestellter ist nur mit den Flügen einer einzigen Fluggesellschaft verbunden*
- Zusätzlich sollen die Eigenschaften von Operationen unabhängig von der Implementierung beschrieben werden
  - In welchem Zustand kann die Operation ausgeführt werden?
  - Welchen Effekt hat die Ausführung der Operation auf die Objektzustände?

Beschreibung von Objekteigenschaften

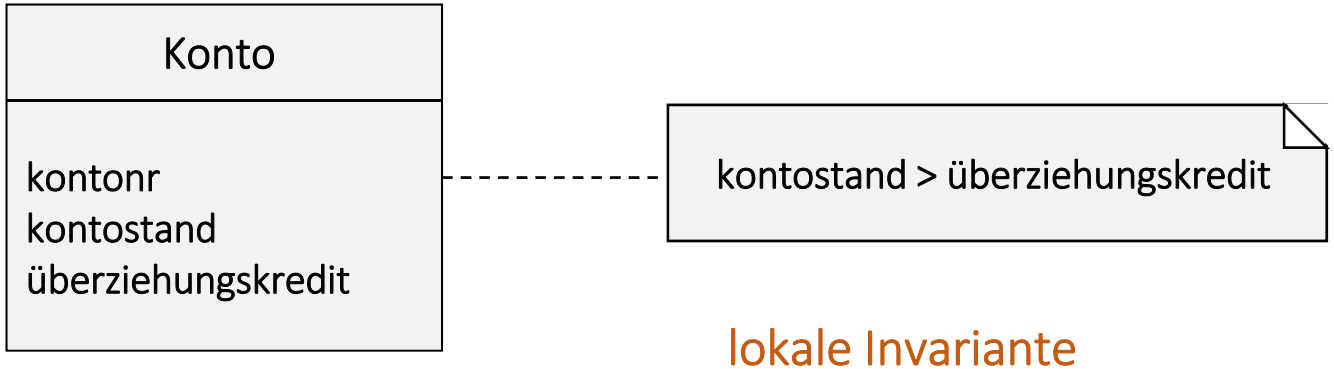
- Informeller Text
- graphisch als Teil der Klassendiagramme
- Formal und ausführbar durch OCL-Ausdrücke
  - OCL-Ausdrücke „navigieren“ durch Objektstrukturen und beschreiben damit Eigenschaften oder Anfragen über Objektstrukturen
  - Vor- und Nachbedingungen von Operationen
  - Invarianten

Invarianten

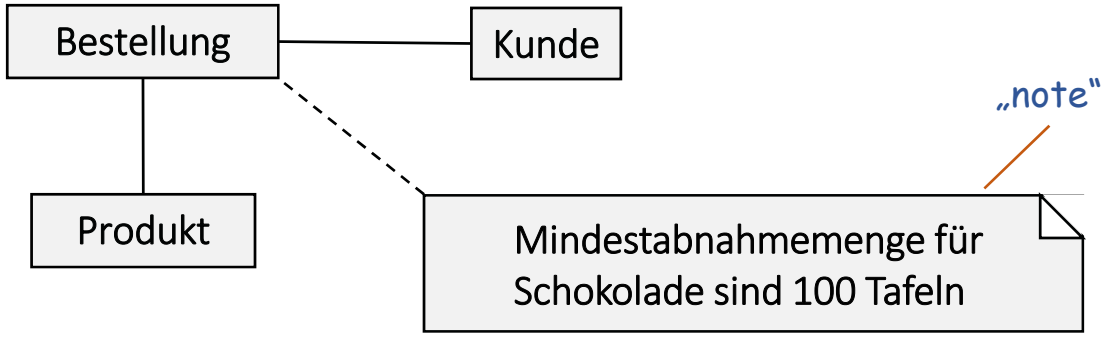
- Eine Invariante beschreibt eine Eigenschaft, die die Objekte des Systems
  - nach ihrer Kreierung und
  - nach der Ausführung jeder (öffentlichen) Methode

  erfüllen.
- Eine lokale Invariante hängt vom Zustand eines einzigen Objekts ab
- Eine globale Invariante hängt vom Zustand mehrerer Objekte ab

Lokale und globale Invarianten - Beispiele

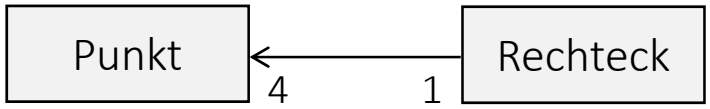


lokale Invariante

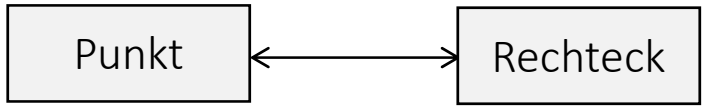


globale Invariante

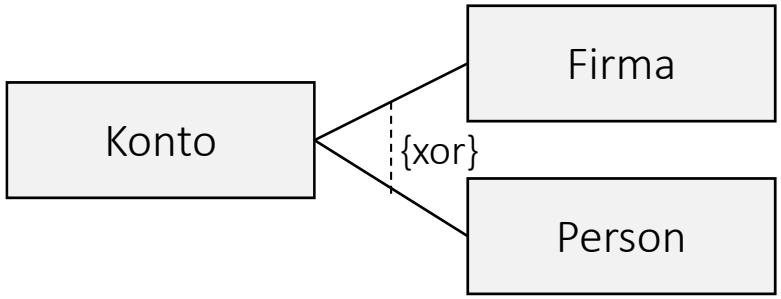
Graphische Repräsentation von Invarianten - Beispiele



Vielfachheiten



bidirektionale Assoziation



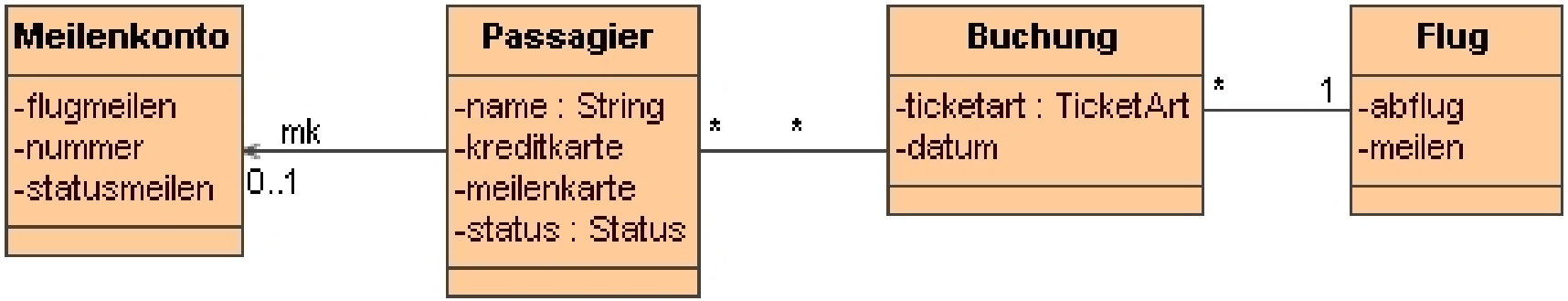
Xor-Assoziation

OCL-Ausdrücke

- OCL-Ausdrücke
  - beziehen sich auf ein gegebenes Klassendiagramm
  - sind statisch, d.h. verändern den Zustand der Objekte nicht
  - sind getypt: jeder OCL-Ausdruck besitzt einen Typ
- OCL-Typen:
  - Grunddatentypen (z.B. Boolean, Integer)
  - Klassen (z.B. Kunde, Person, …)
  - Collection-Typen (z.B. Set, Bag, Sequence)

- Im folgenden wird ein grober Überblick über OCL gegeben

Klassendiagramm - Beispiel



Die Beispiele in diesem Kapitel sind dem Buch H. Störrle: „UML2 für Studenten“ entnommen

OCL-Ausdrücke - Beispiele

Kontext des Ausdrucks:
bestimmt Startobjekt der Navigation

Invariante

```
context Flug inv:
        self.meilen > 0
```

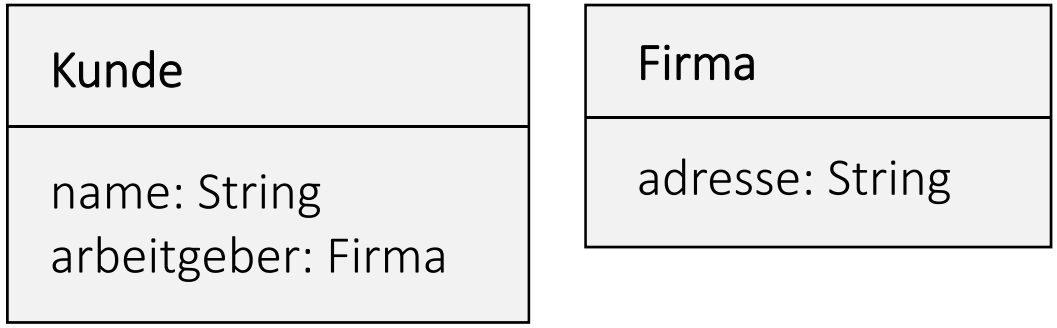
„self." darf auch fehlen

alternativ:

```
context f: Flug inv:
        f.meilen > 0
```

Navigation

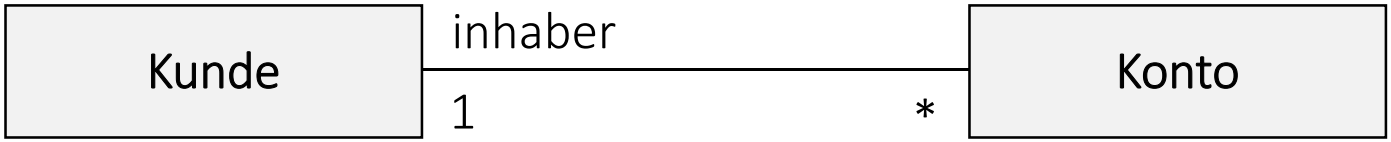
- Navigation entlang von
  - Attributen
  - Assoziationen



Beispiele Navigation entlang von Attributen:

jim.name -- Typ String
jim.arbeitger – Typ Firma
jim.arbeitgeber.adresse – Typ String

Beispiel Navigation entlang von Assoziationen

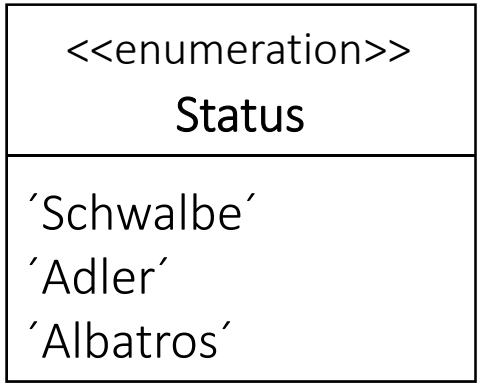


mein_konto.inhaber Typ Kunde

jim.konto Typ **Set(Konto)** (Mengen von Konto-Objekten)

Invariante - Beispiel

```
context Passagier inv:
      status = #Adler
         implies mk.statusmeilen >= 10000
and   status = #Albatros
         implies mk.statusmeilen >= 100000
and   mk.statusmeilen < 10000
         implies status = #Schwalbe
```



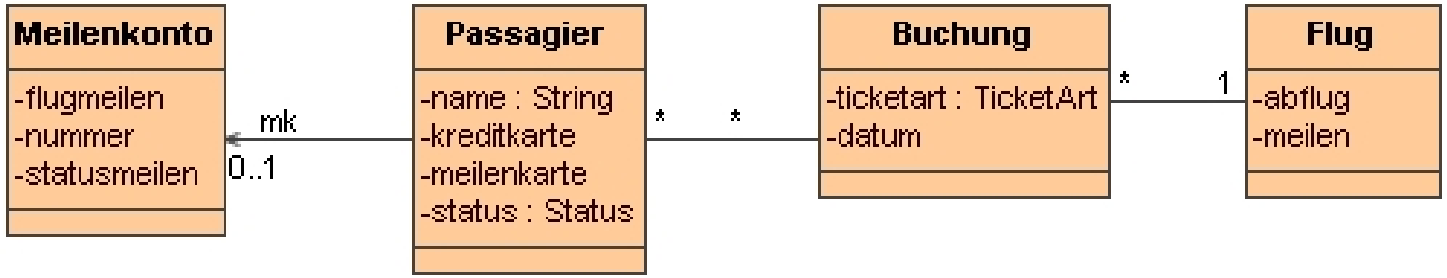
Beispiele OCL-Ausdrücke

```
Passagier.allInstances()
```
-- Menge aller im aktuellen Zustand existierender Passagier-Objekte

```
p.buchung->select(b:Buchung | b.datum = HEUTE)
```
-- Filtern aller heutigen Buchungen eines Passagiers p

```
p.buchung->exists(datum = HEUTE)
```
-- Boolescher Ausdruck mit Existenzquantor: gibt es eine heute durchgeführte Buchung des Passagiers p?
-- Analog dazu prüft der Operator `forall`, ob eine Bedingung für alle Instanzen einer Menge gilt

Beispiel – Spezifikation von Vor-/Nachbedingungen



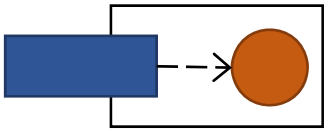

```
context Passagier :: meilenGutschreiben(b:Buchung)
-- Spezifikation der Vor- und Nachbedingung der Methode meilenGutschreiben in Klasse
Passagier
pre:     mk<>oclIsUndefined
-- Passagier hat ein Meilenkonto
post: let  – Definition von Teilausdrücken
           fm = mk.flugmeilen
           meilen = b.flug.meilen
      in
           fm = fm@pre + meilen
-- der Wert von fm nach Ausführung der Methode ist gleich dem Wert vor Ausführung der
Methode (fm@pre) addiert mit den Meilen der Buchung b
```

Hoare-Kalkül vs OCL

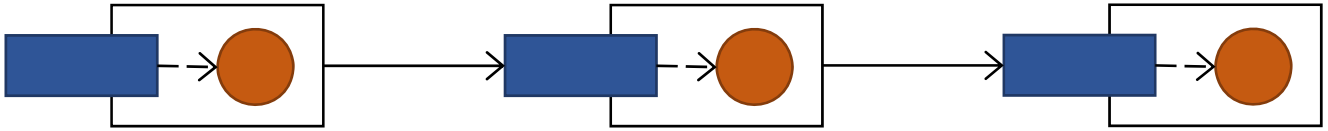
- Gemeinsamkeit: in beiden Ansätzen werden Eigenschaften über Systemzuständen durch Prädikate beschrieben, in Form von Vor- und Nachbedingungen und Invarianten
- Unterschiede:
  - Hoare-Kalkül:
    - Nicht ausführbar
    - Unendlich viele Zustände (int x kann unendlich viele Werte annehmen)
    - Mathematischer Korrektheitsbeweis
  - OCL:
    - Ausführbare Spezifikation
    - Endliche viele Zustände (ein OCL-Ausdruck bezieht sich auf einen Systemzustand mit endlich vielen Objekten)
    - Check der Spezifikation für gegebene Instanz des Klassendiagramms (wie assert-Statement in Java), kein Korrektheitsbeweis

Operationen – lokaler und globaler Effekt

- Idee von Objektorientierung: Eine Operation verändert die lokalen Daten (d.h. die Attribute)



- Aber: Eine Operation kann Operationen anderer Objekte aufrufen



- Insgesamt betrachtet kann eine Operation somit die Zustände vieler Objekte ändern
  = globaler Effekt
- In der Vor- und Nachbedingung wird der globale Effekt einer Operation beschrieben

Bewertung - Invarianten

- Invarianten sind ein wichtiges Instrumentarium zur Dokumentation von Objekteigenschaften
  - Modellierung fachlicher Regeln im fachlichen Klassendiagramm
  - Modellierung von Abhängigkeiten und Einschränkungen im technischen Klassendiagramm

- Invarianten in realen Beispielen führen zu komplexen OCL-Ausdrücken -> in vielen Fällen genügt die textuelle Beschreibung von Invarianten.

Bewertung – Vor- und Nachbedingungen

- Mit Vor- und Nachbedingungen kann der Effekt einer Operation unabhängig von der Implementierung beschrieben werden.
- Verwendung in der Dokumentation zur Spezifikation von Schnittstellen
- Dem praktischen Einsatz von formal beschriebenen Vor- und Nachbedingungen sind Grenzen gesetzt.
- Auch die textuelle Beschreibung von Vor-und Nachbedingungen bringt großen Nutzen.

Textuelle Beschreibung einer Methode

- Vorbedingung:
  - Welche Bedingungen werden an die Parameter bzw. an den Zustand des aufgerufenen Objekts (d.h. den Zustand der Attribute) gestellt?

- Nachbedingung:
  - Welchen Effekt hat die Ausführung der Methode?
  - Welcher Rückgabewert wird zurückgegeben?
  - Wann werden Exceptions geworfen?