

# Formale objektorientierte Software-Entwicklung

---

Prof. Dr. Rolf Hennicker

29.06.2006

# **Kapitel 4: Semantik von Komponentenspezifikationen**

# Ziele

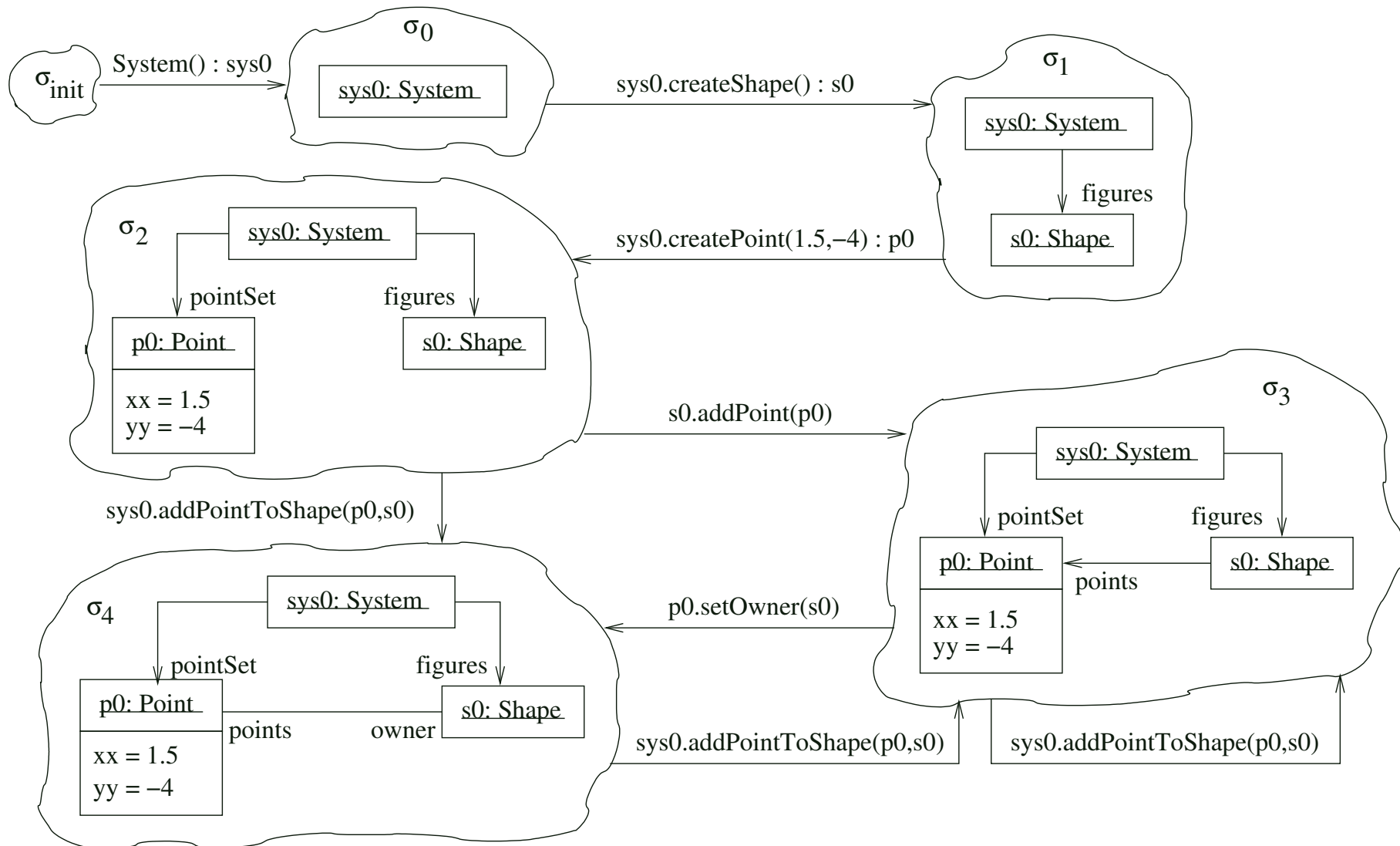
- Über Klassensignaturen gebildete Transitionssysteme kennen ( $\Sigma_{\Delta}$ -Transitionssysteme).
- Verstehen, wann ein  $\Sigma_{\Delta}$ -Transitionssystem eine Operationsspezifikation erfüllt (insbesondere unter Einbeziehung von Invarianten).
- Die Semantik einer Komponentenspezifikation angeben können.
- Eigenschaften von Komponentenspezifikationen analysieren können.

## Grundideen zur Semantik von Komponentenspezifikationen

Sei  $CompSpec = (\langle M, \Delta \rangle, Invs^e, OpSpecs)$  eine Komponentenspezifikation mit formaler Repräsentation  $FRep(CompSpec) = (\langle M, \Sigma_\Delta \rangle, Invs, OpSpecs)$ .

1. Ein **Modell** von  $CompSpec$  ist ein (zu  $\Sigma_\Delta$  passendes) Transitionssystem, das alle Operationsspezifikationen aus  $OpSpecs$  (unter Einbeziehung der Invarianten  $Invs$ ) erfüllt.
2. Die Semantik von  $CompSpec$  ist gegeben durch die Klasse aller Modelle von  $CompSpec$ . Jedes einzelne Modell kann als abstrakte Repräsentation einer korrekten Realisierung von  $CompSpec$  verstanden werden.

# Beispiel: Ausschnitt eines Transitionssystems $\mathcal{T}_{\text{PointsAndShapes}}$



## 4.1 $\Sigma_{\Delta}$ -Transitionssysteme

### Definition (Markiertes Transitionssystem mit Output):

1. Ein markiertes Transitionssystem mit Output ist ein Quintupel

$\mathcal{T} = (S, \sigma_{init}, \mathcal{L}, \Omega, R)$  bestehend aus

- einer Menge  $S$  von Zuständen,
- einem Anfangszustand  $\sigma_{init} \in S$ ,
- einer Menge  $\mathcal{L}$  von Markierungen (Labels),
- einer Menge  $\Omega$  von Outputs und
- einer Transitionsrelation  $R \subseteq S \times \mathcal{L} \times (S \times \Omega)^{\perp}$  wobei  $(S \times \Omega)^{\perp} = (S \times \Omega) \cup \{\perp\}$

2. Der Domain  $Dom(\mathcal{T}) \subseteq S \times \mathcal{L}$  von  $\mathcal{T}$  ist die Menge

$Dom(\mathcal{T}) = \{(\sigma^-, l) \in S \times \mathcal{L} \mid \text{es gibt } \sigma \in S, \omega \in \Omega \text{ so dass } (\sigma^-, l, (\sigma, \omega)) \in R\}$

### Notation

Wir schreiben  $\sigma^- \xrightarrow{l:\omega} \sigma \in R$  für  $(\sigma^-, l, (\sigma, \omega)) \in R$  und  $\sigma^- \xrightarrow{l} \perp \in R$  für  $(\sigma^-, l, \perp) \in R$ .

## ***Grundideen zur Definition von $\Sigma_{\Delta}$ -Transitionssystemen***

Sei  $\Delta$  ein Klassendiagramm mit Klassensignatur  $\Sigma_{\Delta}$  und mit Operationen  $Ops_{\Delta} = M_{\Delta} \cup Q_{\Delta} \cup Con_{\Delta}$ .

Ein  $\Sigma_{\Delta}$ -Transitionssystem ist eine abstrakte Repräsentation eines nicht-deterministischen, objekt-orientierten Programms, das

- alle Operationen aus  $Ops_{\Delta}$  unterstützt und
- bei Operationsaufrufen Zustandsübergänge auf  $State_{\Delta}$  durchführt.

**Definition** ( $\sigma_{init}, Label_{\Delta}, \Omega_{\Delta}$ ):

Sei  $\Delta$  ein Klassendiagramm.

- $C_{\sigma_{init}} = \emptyset$  für alle  $C \in Class_{\Delta}$
- $Label_{\Delta} = \{o.op_C(v_1, \dots, v_n) \mid (op : C \times T_1 \times \dots \times T_n \rightarrow T) \in M_{\Delta} \cup Q_{\Delta},$   
 $o \in \llbracket C \rrbracket, v_i \in \llbracket T_i \rrbracket \text{ für } i = 1, \dots, n\} \cup$   
 $\{C(v_1, \dots, v_n) \mid (C : T_1 \times \dots \times T_n \rightarrow C) \in Con_{\Delta},$   
 $v_i \in \llbracket T_i \rrbracket \text{ für } i = 1, \dots, n\}$
- $\Omega_{\Delta} = RVal_{\Delta} \cup \llbracket Void \rrbracket$  wobei  $\llbracket Void \rrbracket = \{*\}$

**Notation**

Falls der Name  $op$  in  $\Delta$  eindeutig ist, schreiben wir  $o.op(v_1, \dots, v_n)$  statt  $o.op_C(v_1, \dots, v_n)$ .

**Definition ( $\Sigma_{\Delta}$ -Transitionssystem):**

Sei  $\Delta$  ein Klassendiagramm mit Klassensignatur  $\Sigma_{\Delta}$ .

Ein  $\Sigma_{\Delta}$ -*Transitionssystem* ist ein markiertes Transitionssystem mit Output

$$\mathcal{T}_{\Delta} = (\text{State}_{\Delta}, \sigma_{init}^{\Delta}, \text{Label}_{\Delta}, \Omega_{\Delta}, R_{\Delta})$$

so dass

$$R_{\Delta} \subseteq \text{State}_{\Delta} \times \text{Label}_{\Delta} \times (\text{State}_{\Delta} \times \Omega_{\Delta})^{\perp}$$

die drei folgenden Bedingungen erfüllt:

- (wdef) Für alle  $\sigma^{-} \xrightarrow{o.op_C(v_1, \dots, v_n):r} \sigma \in R_{\Delta}$   
 mit  $(op : C \times T_1 \times \dots \times T_n \longrightarrow T) \in M_{\Delta} \cup Q_{\Delta}$  gilt:
- (1)  $(o, v_1, \dots, v_n) \in \text{Valid}(\sigma^{-}, \llbracket C \rrbracket \times \llbracket T_1 \rrbracket \times \dots \times \llbracket T_n \rrbracket)$
  - (2)  $o \neq \text{null}$
  - (3) falls  $T \neq \text{Void}$  dann  $r \in \text{Valid}(\sigma, \llbracket T \rrbracket)$
  - (4) falls  $T = \text{Void}$  dann  $r = *$

(query) Für alle  $\sigma^- \xrightarrow{o.q_C(v_1, \dots, v_n):r} \sigma \in R_{\Delta}$   
 mit  $(q : C \times T_1 \times \dots \times T_n \longrightarrow T) \in Q_{\Delta}$  gilt:  
 $\sigma = \sigma^-$

(con) Für alle  $\sigma^- \xrightarrow{C(v_1, \dots, v_n):o} \sigma \in R_{\Delta}$   
 mit  $(C : T_1 \times \dots \times T_n \longrightarrow C) \in Con_{\Delta}$  gilt:  
 (1)  $(v_1, \dots, v_n) \in Valid(\sigma^-, \llbracket T_1 \rrbracket \times \dots \times \llbracket T_n \rrbracket)$   
 (2)  $o \in C_{\sigma}$  und  $o \notin D_{\sigma}$  für alle  $D < C$   
 (3)  $o \notin C_{\sigma^-}$

## Notation

Für Methoden ohne Ergebnis schreiben wir auch  $\sigma^- \xrightarrow{o.m_C(v_1, \dots, v_n)} \sigma$   
 statt  $\sigma^- \xrightarrow{o.m_C(v_1, \dots, v_n):*} \sigma$ .

## Bemerkung

- Eine Transition  $\sigma^- \xrightarrow{o.op(v_1, \dots, v_n)} \sigma \in R_{\Delta}$  modelliert die Tatsache, dass
  - im Zustand  $\sigma^-$  der Operationsaufruf  $o.op(v_1, \dots, v_n)$  ausführbar (enabled) ist und dass
  - $\sigma$  ein möglicher (wohldefinierter) Nachfolgezustand ist, der nach Ausführung der Operation erreicht werden kann.
- Eine Transition  $\sigma^- \xrightarrow{o.op(v_1, \dots, v_n)} \perp \in R_{\Delta}$  modelliert die Tatsache, dass
  - im Zustand  $\sigma^-$  der Operationsaufruf  $o.op(v_1, \dots, v_n)$  ausführbar (enabled) ist und dass
  - die Ausführung der Operation nicht terminieren kann oder zu einem Laufzeitfehler führen kann.

## 4.2 Erfüllung von Operationsspezifikationen

### *Grundideen (noch ohne Einbeziehung von Invarianten)*

1. Der Implementierer einer Operation  $op$  kann davon ausgehen, dass beim Aufruf der Operation die Vorbedingung von  $op$  gilt.
2. Der Implementierer muss garantieren, dass dann
  - die Operation ausführbar ist,
  - jede Ausführung der Operation terminiert und zu keinem Laufzeitfehler führt und
  - nach jeder Ausführung der Operation deren Nachbedingung gilt.

### **Beispiel** ( $\mathcal{T}_{\text{PointsAndShapes}}$ )

- $\sigma_2 \xrightarrow{\text{sys0.addPointToShape}(p_0, s_0)} \sigma_4$  ist zulässig, da Vor- und Nachbed. erfüllt sind,
- $\sigma_4 \xrightarrow{\text{sys0.addPointToShape}(p_0, s_0)} \sigma_3$  ist zulässig, da die Vorbed. nicht erfüllt ist.

## Definition (Erfüllung von Operationsspezifikationen):

Sei  $\Sigma_\Delta$  eine Klassensignatur und sei

$\mathcal{T}_\Delta = (State_\Delta, \sigma_{init}, Label_\Delta, \Omega_\Delta, R_\Delta)$  ein  $\Sigma_\Delta$ -Transitionssystem.

### 1. Methoden ohne Rückgabewert:

Sei  $(op : C \times T_1 \times \dots \times T_n \rightarrow Void) \in M_\Delta$ .

$$\mathcal{T}_\Delta \models \text{context } C :: op(x_1 : T_1, \dots, x_n : T_n)$$

$$\text{pre} : P$$

$$\text{post} : Q$$

falls für alle  $(\sigma^-, o, v_1, \dots, v_n) \in State_\Delta \times ([C] \times [T_1] \times \dots \times [T_n])$  mit  $o \neq null$  gilt:

Falls  $\llbracket P \rrbracket_{\beta, \sigma^-, \sigma^-} = true$  (mit  $\beta(self) = o, \beta(x_i) = v_i$ ) dann gilt:

(a) Es existiert  $\sigma \in State_\Delta$  mit  $\sigma^- \xrightarrow{o.op_C(v_1, \dots, v_n)} \sigma \in R_\Delta$

(b)  $\sigma^- \xrightarrow{o.op_C(v_1, \dots, v_n)} \perp \notin R_\Delta$

(c) Für alle  $\sigma^- \xrightarrow{o.op_C(v_1, \dots, v_n)} \sigma \in R_\Delta$  ist  $\llbracket Q \rrbracket_{\beta, \sigma^-, \sigma} = true$  (mit demselben  $\beta$  wie oben)

## 2. Methoden mit Rückgabewert:

Sei  $(op : C \times T_1 \times \dots \times T_n \rightarrow T) \in M_\Delta$ .

$$\begin{array}{l} \mathcal{T}_\Delta \models \text{context } C :: op(x_1 : T_1, \dots, x_n : T_n) : T \quad (T \neq Void) \\ \quad \text{pre} : P \\ \quad \text{post} : Q \end{array}$$

falls für alle  $(\sigma^-, o, v_1, \dots, v_n) \in State_\Delta \times ([C] \times [T_1] \times \dots \times [T_n])$  mit  $o \neq null$  gilt:

Falls  $\llbracket P \rrbracket_{\beta, \sigma^-, \sigma^-} = true$  (mit  $\beta(self) = o$ ,  $\beta(x_i) = v_i$ ) dann gilt:

(a) Es existiert  $(\sigma, r) \in State_\Delta \times [T]$  mit  $\sigma^- \xrightarrow{o.op_C(v_1, \dots, v_n):r} \sigma \in R_\Delta$

(b)  $\sigma^- \xrightarrow{o.op_C(v_1, \dots, v_n)} \perp \notin R_\Delta$

(c) Für alle  $\sigma^- \xrightarrow{o.op_C(v_1, \dots, v_n):r} \sigma \in R_\Delta$  ist  $\llbracket Q \rrbracket_{\beta[result \mapsto r], \sigma^-, \sigma} = true$  (mit  $\beta$  wie oben)

### 3. Queryoperationen:

Sei  $(q : C \times T_1 \times \dots \times T_n \rightarrow T) \in Q_\Delta$ .

$$\begin{array}{l} \mathcal{T}_\Delta \models \text{context } C :: q(x_1 : T_1, \dots, x_n : T_n) : T \\ \quad \text{pre} : P \\ \quad \text{post} : Q \end{array}$$

falls für alle  $(\sigma^-, o, v_1, \dots, v_n) \in \text{State}_\Delta \times ([C] \times [T_1] \times \dots \times [T_n])$  mit  $o \neq \text{null}$  gilt:

Falls  $\llbracket P \rrbracket_{\beta, \sigma^-, \sigma^-} = \text{true}$  (mit  $\beta(\text{self}) = o$ ,  $\beta(x_i) = v_i$ ) dann gilt:

- (a) Es existiert  $r \in [T]$  mit  $\sigma^- \xrightarrow{o.q_C(v_1, \dots, v_n):r} \sigma^- \in R_\Delta$
- (b)  $\sigma^- \xrightarrow{o.q_C(v_1, \dots, v_n)} \perp \notin R_\Delta$
- (c) Für alle  $\sigma^- \xrightarrow{o.q_C(v_1, \dots, v_n):r} \sigma^- \in R_\Delta$  ist  $\llbracket Q \rrbracket_{\beta[\text{result} \mapsto r], \sigma^-, \sigma^-} = \text{true}$  (mit  $\beta$  wie oben)

#### 4. Konstruktoren:

Sei  $(C : T_1 \times \dots \times T_n \rightarrow C) \in \text{Con}_\Delta$ .

$$\begin{array}{l} T_\Delta \models \text{context } C :: C(x_1 : T_1, \dots, x_n : T_n) \\ \quad \text{pre} : P \\ \quad \text{post} : Q \end{array}$$

falls für alle  $(\sigma^-, v_1, \dots, v_n) \in \text{State}_\Delta \times ([T_1] \times \dots \times [T_n])$  gilt:

Falls  $\llbracket P \rrbracket_{\beta, \sigma^-, \sigma^-} = \text{true}$  (mit  $\beta(x_i) = v_i$ ) dann gilt:

(a) Es existiert  $(\sigma, o) \in \text{State}_\Delta \times [C]$  mit  $\sigma^- \xrightarrow{C(v_1, \dots, v_n):o} \sigma \in R_\Delta$

(b)  $\sigma^- \xrightarrow{C(v_1, \dots, v_n)} \perp \notin R_\Delta$

(c) Für alle  $\sigma^- \xrightarrow{C(v_1, \dots, v_n):o} \sigma \in R_\Delta$  ist  $\llbracket Q \rrbracket_{\beta[\text{self} \mapsto o], \sigma^-, \sigma} = \text{true}$  (mit demselben  $\beta$  wie oben)

## 4.3 Einbeziehung von Invarianten

1. Der Implementierer einer Operation  $op$  kann davon ausgehen, dass beim Aufruf der Operation die Vorbedingung von  $op$  gilt und
  - falls die Operation komponenten-öffentlich ist: alle Komponenteninvarianten und alle Klasseninvarianten (für alle existierenden Objekte) gelten,
  - falls die Operation komponenten-privat ist: alle Klasseninvarianten (für alle existierenden Objekte) gelten.
  
2. Der Implementierer muss garantieren, dass dann
  - die Operation ausführbar ist,
  - jede Ausführung der Operation terminiert und zu keinem Laufzeitfehler führt und
  - nach jeder Ausführung der Operation deren Nachbedingung gilt und
    - ▷ falls die Operation komponenten-öffentlich ist: alle Komponenteninvarianten und alle Klasseninvarianten (für alle existierenden Objekte) gelten,
    - ▷ falls die Operation komponenten-privat ist: alle Klasseninvarianten (für alle existierenden Objekte) gelten.

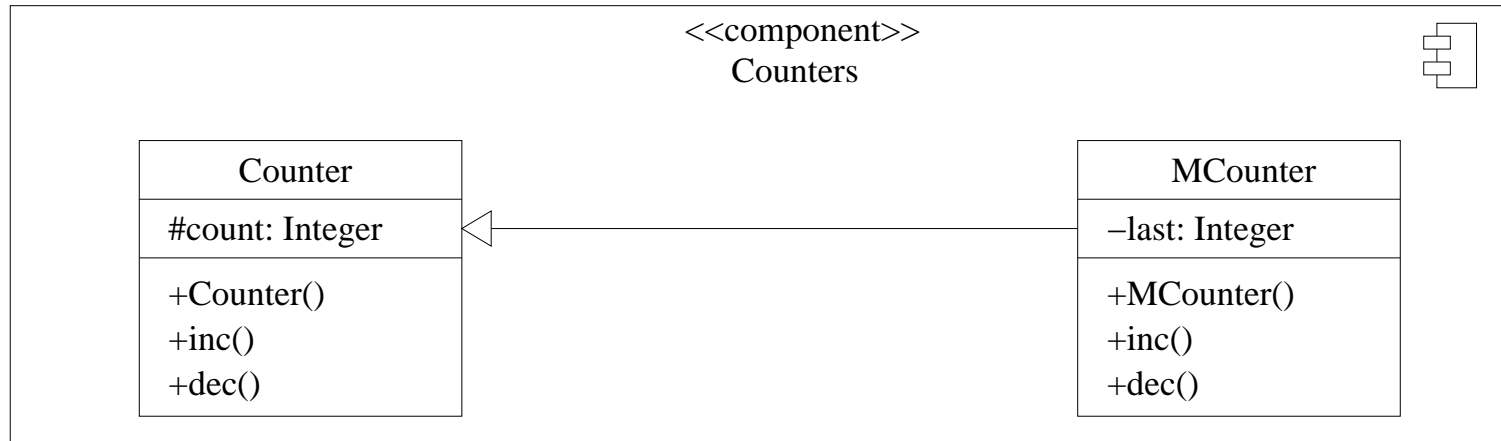
## Beispiel ( $\mathcal{T}_{\text{PointsAndShapes}}$ )

Die durch komponenten-öffentliche Operationen verursachten Zustandsübergänge erhalten die Komponenteninvariante `invBidirect` (und die anderen Komponenteninvarianten `invSameSystem` und `invOneSystem`), wenn die Vorbedingung der Operation erfüllt ist.

### **Beachte:**

- $\sigma_3 \xrightarrow{\text{sys0.addPointToShape}(p0,s0)} \sigma_3$  ist zulässig, da in  $\sigma_3$  die Komponenteninvariante `invBidirect` nicht erfüllt ist.
- $\sigma_4 \xrightarrow{\text{sys0.addPointToShape}(p0,s0)} \sigma_3$  erhält `invBidirect` nicht, ist aber zulässig, da in  $\sigma_4$  die Vorbedingung nicht erfüllt ist.
- $\sigma_2 \xrightarrow{s0.addPoint(p0)} \sigma_3$  erhält `invBidirect` nicht, ist aber zulässig, da `addPoint` komponenten-privat ist.

## Beispiel: Komponentenspezifikation für "Counters"



```
context Counter
  inv: count >= 0

context Counter::Counter()
  post: count = 0

context Counter::inc()
  post: count = count@pre + 1

context Counter::dec()
  pre: count > 0
  post: count = count@pre - 1
```

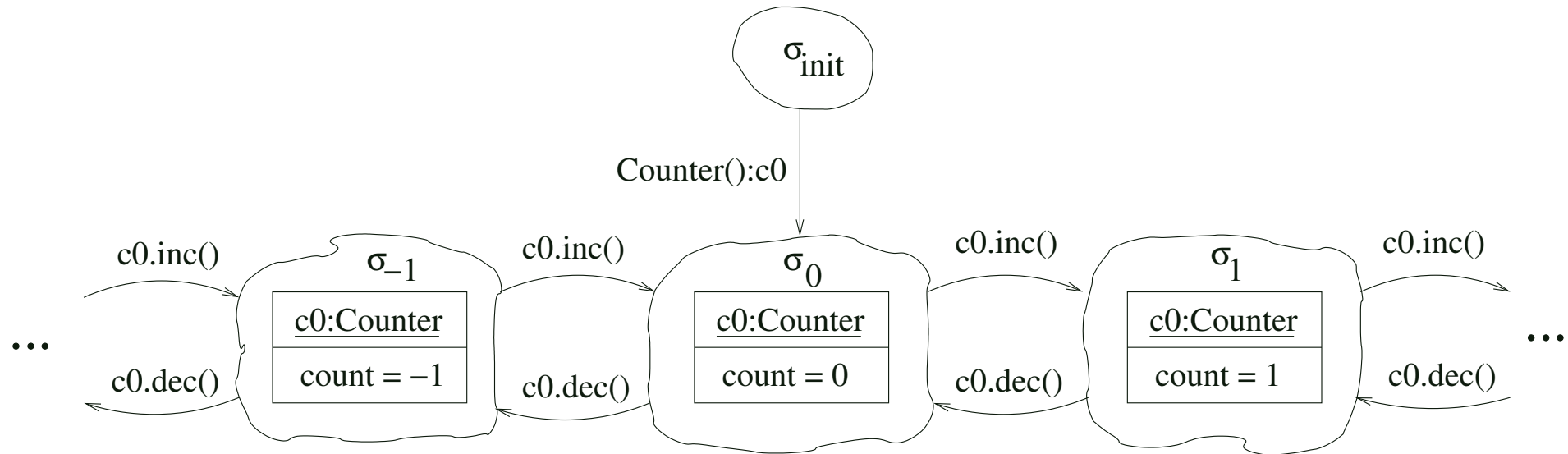
```
context MCounter
  inv: last >= 0

context MCounter::MCounter()
  post: count = 0 and last = 0

context MCounter::inc()
  post: count = count@pre + 1 and
        last = count@pre

context MCounter::dec()
  pre: count > 0
  post: count = count@pre - 1 and
        last = count@pre
```

## Beispiel: Ausschnitt eines Transitionssystems $\mathcal{T}_{\text{Counters}}$



Die Klasseninvariante bleibt bei allen durch "korrekte" Operationsaufrufe verursachten Zustandsübergängen erhalten.

### **Beachte:**

$\sigma_0 \xrightarrow{c0.dec()} \sigma_{-1}$  erhält die Klasseninvariante nicht, ist aber zulässig, da in  $\sigma_0$  die Vorbedingung von *dec* nicht gilt.

## Allgemeine Definitionen zur Einbeziehung von Invarianten

Im Folgenden sei immer  $CompSpec = (\langle M, \Delta \rangle, Invs^e, OpSpecs)$  eine Komponentenspezifikation mit formaler Repräsent.  $FRep(CompSpec) = (\langle M, \Sigma_\Delta \rangle, Invs, OpSpecs)$ .

**Definition ( $INV_C, CLASSINV_\Delta, COMPINV_M$ ):**

Sei  $C \in Class_\Delta$ .

$$INV_C = \bigwedge_{(context\ C\ inv: Inv) \in Invs} Inv$$

$$CLASSINV_\Delta = \bigwedge_{C \in Class_\Delta} C.allInstances() \rightarrow forAll(self \mid INV_C)$$

$$COMPINV_M = \bigwedge_{(context\ M\ inv: Inv) \in Invs} Inv$$

Dabei steht  $\bigwedge_{i \in \{1, \dots, n\}} E_i$  für  $E_1$  and ... and  $E_n$ .

**Lemma:**

Sei  $\sigma \in State_{\Delta}$  und  $\beta$  eine beliebige Belegung.

$\llbracket CLASSINV_{\Delta} \rrbracket_{\beta, \sigma, \sigma} = true$  genau dann, wenn

für alle  $C \in Class_{\Delta}$ , für alle  $(context\ C\ inv : Inv) \in Invs$  und

für alle  $o \in C_{\sigma}$  gilt:  $\llbracket Inv \rrbracket_{\beta[self \mapsto o], \sigma, \sigma} = true$ .

**Bemerkung:**

Klasseninvarianten von Oberklassen müssen auch für alle Objekte von Unterklassen gelten.

**Definition ( $E^+$ ,  $E^{\sim}$ ,  $E^-$ ):**

Sei  $E$  ein OCL-Ausdruck vom Typ Boolean.

$$E^+ = E \text{ and } COMPINV_M \text{ and } CLASSINV_{\Delta}$$

$$E^{\sim} = E \text{ and } CLASSINV_{\Delta}$$

$$E^- = E$$

**Definition:**

Ein  $\Sigma_{\Delta}$ -Transitionssystem  $\mathcal{T}_{\Delta}$  erfüllt eine Operationsspezifikation

context  $C :: op(x_1 : T_1, \dots, x_n : T_n)$  pre:  $P$  post:  $Q$

unter Einbeziehung der Invarianten  $Invs$ , wenn gilt:

$$\begin{aligned} \mathcal{T}_{\Delta} \models & \text{context } C :: op(x_1 : T_1, \dots, x_n : T_n) \\ & \text{pre: } P^{Visibility(op)} \\ & \text{post: } Q^{Visibility(op)} \end{aligned}$$

## Beispiel

Betrachte die Komponentenspezifikation für Counters und das Transitionssystem  $\mathcal{T}_{\text{Counters}}$ .

```
 $\mathcal{T}_{\text{Counters}} \models$  context Counter::dec()  
    pre: count > 0 and  
        Counter.allInstances() -> forAll(self |  
            self.count >= 0) and  
        MCounter.allInstances() -> forAll(self |  
            self.last >= 0)  
    post: count = count@pre - 1 and  
        Counter.allInstances() -> forAll(self |  
            self.count >= 0) and  
        MCounter.allInstances() -> forAll(self |  
            self.last >= 0)
```

## 4.4 Modelle und Semantik von Komponentenspezifikationen

### *Grundideen des Modellbegriffs*

Ein Modell einer Komponentenspezifikation  $CompSpec$  ist ein  $\Sigma_{\Delta}$ -Transitionssystem, das alle Operationsspezifikationen von  $CompSpec$  unter Einbeziehung der gegebenen (expliziten und impliziten) Invarianten erfüllt.

Außerdem müssen im Anfangszustand  $\sigma_{init}$  alle Komponenteninvarianten gelten (die Klasseninvarianten gelten trivialerweise in  $\sigma_{init}$ ).

**Definition (Modell):**

Sei  $CompSpec$  eine Komponentenspezifikation mit formaler Repräsentation  $FRep(CompSpec) = (\langle M, \Sigma_\Delta \rangle, Invs, OpSpecs)$  und sei  $\mathcal{T}_\Delta$  ein  $\Sigma_\Delta$ -Transitionssystem.

$\mathcal{T}_\Delta$  ist ein *Modell* von  $CompSpec$ , wenn gilt:

1.  $\llbracket COMPINV_M \rrbracket_{\beta, \sigma_{init}, \sigma_{init}} = true$ .
2. Für alle  $(op : C \times T_1 \times \dots \times T_n \rightarrow Void) \in Opns_\Delta$  mit  $(context\ C :: op(x_1 : T_1, \dots, x_n : T_n)\ pre: P\ post: Q) \in OpSpecs$  gilt:

$$\begin{aligned} \mathcal{T}_\Delta \models & \text{context } C :: op(x_1 : T_1, \dots, x_n : T_n) \\ & \text{pre: } P^{Visibility(op)} \\ & \text{post: } Q^{Visibility(op)} \end{aligned}$$

und analog für Methoden mit Rückgabewert, Queryoperationen und Konstruktoren, wobei bei Queryoperationen  $Q^{Visibility(op)} = Q$  genommen werden kann.

**Definition (Semantik von Komponentenspezifikationen):**

Sei  $CompSpec$  eine Komponentenspezifikation.

$$\llbracket CompSpec \rrbracket = \{ \mathcal{T}_\Delta \mid \mathcal{T}_\Delta \text{ ist ein Modell von } CompSpec \},$$

d.h. die Semantik von  $CompSpec$  ist gegeben durch die Klasse aller Modelle von  $CompSpec$ .

**Definition (Korrekte Realisierung):**

Sei  $CompSpec$  eine Komponentenspezifikation.

Ein  $\Sigma_\Delta$ -Transitionssystem  $\mathcal{T}_\Delta$  ist eine *korrekte Realisierung* von  $CompSpec$ , wenn  $\mathcal{T}_\Delta \in \llbracket CompSpec \rrbracket$ .

**Beispiel** ( $\mathcal{T}_{\text{Counters}}$ )

Gegeben sei die Komponentenspezifikation für Counters.

Das  $\Sigma_{\Delta}$ -Transitionssystem  $\mathcal{T}_{\text{Counters}} = (State_{\Delta}, \sigma_{init}, Label_{\Delta}, \Omega_{\Delta}, R_{\Delta})$  hat folgende Transitionsrelation  $R_{\Delta}$ :

$$\begin{aligned}
 R_{\Delta} = & \left\{ \sigma^- \xrightarrow{\text{Counter}():vc} \sigma \mid \sigma^- \in State_{\Delta}, vc \in OId_{\text{Counter}}, vc \notin Counter_{\sigma^-}, \right. \\
 & \left. \sigma = \sigma^- [vc = \text{new}_{\text{Counter}}] \right\} \\
 \cup & \left\{ \sigma^- \xrightarrow{vc.inc_{\text{Counter}}()} \sigma \mid \sigma^- \in State_{\Delta}, vc \in Counter_{\sigma^-}, \right. \\
 & \left. \sigma = \sigma^- [vc.count \mapsto vc.count_{\sigma^-} + 1] \right\} \\
 \cup & \left\{ \sigma^- \xrightarrow{vc.dec_{\text{Counter}}()} \sigma \mid \sigma^- \in State_{\Delta}, vc \in Counter_{\sigma^-}, \right. \\
 & \left. \sigma = \sigma^- [vc.count \mapsto vc.count_{\sigma^-} - 1] \right\}
 \end{aligned}$$

$$\cup \left\{ \sigma^- \xrightarrow{MCounter():mc} \sigma \quad \left| \quad \begin{array}{l} \sigma^- \in State_{\Delta}, mc \in OId_{MCounter}, mc \notin MCounter_{\sigma^-} \\ \sigma = \sigma^- [ mc = \text{new}_{MCounter} ] \end{array} \right. \right\}$$

$$\cup \left\{ \sigma^- \xrightarrow{mc.inc_{MCounter}()} \sigma \quad \left| \quad \begin{array}{l} \sigma^- \in State_{\Delta}, mc \in MCounter_{\sigma^-}, \\ \sigma = \sigma^- [ mc.count \mapsto mc.count_{\sigma^-} + 1, \\ mc.last \mapsto mc.count_{\sigma^-} ] \end{array} \right. \right\}$$

$$\cup \left\{ \sigma^- \xrightarrow{mc.dec_{MCounter}()} \sigma \quad \left| \quad \begin{array}{l} \sigma^- \in State_{\Delta}, mc \in MCounter_{\sigma^-}, \\ \sigma = \sigma^- [ mc.count \mapsto mc.count_{\sigma^-} - 1, \\ mc.last \mapsto mc.count_{\sigma^-} ] \end{array} \right. \right\}$$

$\mathcal{T}_{\text{Counters}}$  ist eine korrekte Realisierung (Modell) der Counters Spezifikation.

## 4.5 Eigenschaften von Komponentenspezifikationen

### Definition (Konsistenz):

Eine Komponentenspezifikation  $CompSpec$  ist *konsistent*, wenn  $\llbracket CompSpec \rrbracket \neq \emptyset$ , d.h. es gibt mindestens eine korrekte Realisierung von  $CompSpec$ .

### Bemerkung:

Die in Abschnitt 3.4 angegebene Methodik zur Erstellung von Operationsspezifikationen (unter Berücksichtigung der Invarianten) unterstützt die Erstellung von konsistenten Komponentenspezifikationen.

**Satz (Charakterisierung der Konsistenz):**

Eine Komponentenspezifikation  $CompSpec$  ist konsistent genau dann, wenn gilt:

1.  $\llbracket COMPINV_M \rrbracket_{\beta, \sigma_{init}, \sigma_{init}} = true$
2. Für alle  $(op : C \times T_1 \times \dots \times T_n \rightarrow Void) \in Opns_{\Delta}$  mit  
(context  $C :: op(x_1 : T_1, \dots, x_n : T_n)$  pre:  $P$  post:  $Q$ )  $\in OpSpecs$  gilt:

Für alle  $(\sigma^-, o, v_1, \dots, v_n) \in State_{\Delta} \times ([C] \times [T_1] \times \dots \times [T_n])$   
mit  $o \neq null$  gilt:

Falls  $\llbracket P^{Visibility(op)} \rrbracket_{\beta, \sigma^-, \sigma^-} = true$  (mit  $\beta(self) = o, \beta(x_i) = v_i$ )  
dann existiert  $\sigma \in State_{\Delta}$ , so dass  $\llbracket Q^{Visibility(op)} \rrbracket_{\beta, \sigma^-, \sigma} = true$  (mit demselben  $\beta$ )

und analog für Methoden mit Rückgabewert, Queryoperationen und Konstruktoren.

**Beweis:** " $\Leftarrow$ ":

Wir definieren ein  $\Sigma_\Delta$ -Transitionssystem  $\mathcal{T}_\Delta$  durch die Relation

$$R_\Delta = \left\{ \sigma^- \xrightarrow{o.op_C(v_1, \dots, v_n)} \sigma \mid (op : C \times T_1 \times \dots \times T_n \rightarrow Void) \in M_\Delta, \right. \\ \left. \sigma^-, \sigma \in State_\Delta, (\sigma^-, o, v_1, \dots, v_n) \in State_\Delta \times ([C] \times [T_1] \times \dots \times [T_n]), \right. \\ \left. o \neq null, \llbracket Q^{Visibility(op)} \rrbracket_{\beta, \sigma^-, \sigma} = true \text{ mit } \beta(self) = o, \beta(x_i) = v_i \right\}$$

∪ "entsprechende Transitionen für Methoden mit Rückgabewert, Queryoperationen und Konstruktoren"

Für alle  $op \in Opns_\Delta$  mit

(context  $C :: op(x_1 : T_1, \dots, x_n : T_n)$  pre:  $P$  post:  $Q$ )  $\in OpSpecs$  gilt:

$$\mathcal{T}_\Delta \models \text{context } C :: op(x_1 : T_1, \dots, x_n : T_n) \text{ pre: } P^{Visibility(op)} \text{ post: } Q^{Visibility(op)}$$

da die Bedingung (a) in der Definition von " $\models$ " aus der Voraussetzung und der Konstruktion von  $R_\Delta$  folgt und die Bedingungen (b) und (c) in der Def. von " $\models$ " schon alleine aus der Konstruktion von  $R_\Delta$  folgen. Folglich ist  $\mathcal{T}_\Delta \in [CompSpec]$ .

" $\Rightarrow$ ":

Da *CompSpec* konsistent ist, existiert  $\mathcal{T}_\Delta \in \llbracket \text{CompSpec} \rrbracket$ .

Folglich ist  $\llbracket \text{COMPINV}_M \rrbracket_{\beta, \sigma_{init}, \sigma_{init}} = true$

und für alle (context  $C :: op(x_1 : T_1, \dots, x_n : T_n)$  pre:  $P$  post:  $Q$ )  $\in OpSpecs$  gilt:

$$\mathcal{T}_\Delta \models \text{context } C :: op(x_1 : T_1, \dots, x_n : T_n) \text{ pre: } P^{Visibility(op)} \text{ post: } Q^{Visibility(op)}$$

Sei nun  $(\sigma^-, o, v_1, \dots, v_n) \in State_\Delta \times (\llbracket C \rrbracket \times \llbracket T_1 \rrbracket \times \dots \times \llbracket T_n \rrbracket)$ ,  $o \neq null$

und gelte  $\llbracket P^{Visibility(op)} \rrbracket_{\beta, \sigma^-, \sigma^-} = true$  (mit  $\beta(self) = o$ ,  $\beta(x_i) = v_i$ ).

Dann existiert, wegen (a) in der Def. von " $\models$ ", ein  $\sigma \in State_\Delta$

mit  $\sigma^- \xrightarrow{o.op_C(v_1, \dots, v_n)} \sigma \in R_\Delta$  und, wegen (c) in der Def. von " $\models$ ",

ist  $\llbracket Q^{Visibility(op)} \rrbracket_{\beta, \sigma^-, \sigma} = true$ .

**Beispiel (Counters)**

Ersetze in der Komponentenspezifikation für Counters die Operationsspezifikation von `dec` durch:

```
context Counter::dec()
  pre: true
  post: count = count@pre - 1
```

Dann ist die resultierende Komponentenspezifikation inkonsistent.

**Beweis:**

Wir zeigen, dass die Bedingung (2) des Satzes **nicht** erfüllt ist:

Sei  $\sigma^- \in State_\Delta$ ,  $Counter_{\sigma^-} = \{c0\}$ ,  $c0.count_{\sigma^-} = 0$ .

Dann gilt (für  $\beta(self) = c0$ ):

$$\begin{aligned} \llbracket P^{Visibility(dec)} \rrbracket_{\beta, \sigma^-, \sigma^-} &= \llbracket true \text{ and } COMPINV_\Delta \text{ and } CLASSINV_\Delta \rrbracket_{\beta, \sigma^-, \sigma^-} = \\ \llbracket Counter.allInstances() \rightarrow \text{forAll}(self \mid self.count \geq 0) \text{ and} \\ MCounter.allInstances() \rightarrow \text{forAll}(self \mid last.count \geq 0) \rrbracket_{\beta, \sigma^-, \sigma^-} &= true. \end{aligned}$$

**Annahme:** Es gibt  $\sigma \in State_{\Delta}$  mit  $\llbracket Q^{Visibility(dec)} \rrbracket_{\beta, \sigma^-, \sigma} = true$ .

Es ist also  $\llbracket Q^{Visibility(dec)} \rrbracket_{\beta, \sigma^-, \sigma} =$   
 $\llbracket self.count = self.count@pre - 1 \text{ and } COMPINV_{\Delta} \text{ and } CLASSINV_{\Delta} \rrbracket_{\beta, \sigma^-, \sigma} =$   
 $\llbracket self.count = self.count@pre - 1 \text{ and}$   
 $Counter.allInstances() \rightarrow forAll(self \mid self.count \geq 0) \text{ and}$   
 $MCounter.allInstances() \rightarrow forAll(self \mid last.count \geq 0) \rrbracket_{\beta, \sigma^-, \sigma} = true$ .

Dann ist  $\llbracket self.count = self.count@pre - 1 \rrbracket_{\beta, \sigma^-, \sigma} = true$   
 (also  $\llbracket self.count \rrbracket_{\beta, \sigma^-, \sigma} = -1$ ) und  
 $\llbracket Counter.allInstances() \rightarrow forAll(self \mid self.count \geq 0) \rrbracket_{\beta, \sigma^-, \sigma} = true$ .

**Widerspruch!**

**Definition (Verhaltensverträglichkeit bzgl. Subtypen):**

Eine Komponentenspezifikation  $CompSpec$  ist *verhaltensverträglich bzgl. Subtypen*, wenn für alle  $C, D \in Class_{\Delta}$  mit  $D < C$  und für alle  $(op : C \times T_1 \times \dots \times T_n \longrightarrow Void)$ ,  $(op : D \times T_1 \times \dots \times T_n \longrightarrow Void) \in Opns_{\Delta}$  mit Operationsspezifikationen

$$\begin{aligned} \text{context } C &:: op(x_1 : T_1, \dots, x_n : T_n) \text{ pre} : P_{op_C} \text{ post} : Q_{op_C} \\ \text{context } D &:: op(x_1 : T_1, \dots, x_n : T_n) \text{ pre} : P_{op_D} \text{ post} : Q_{op_D} \end{aligned}$$

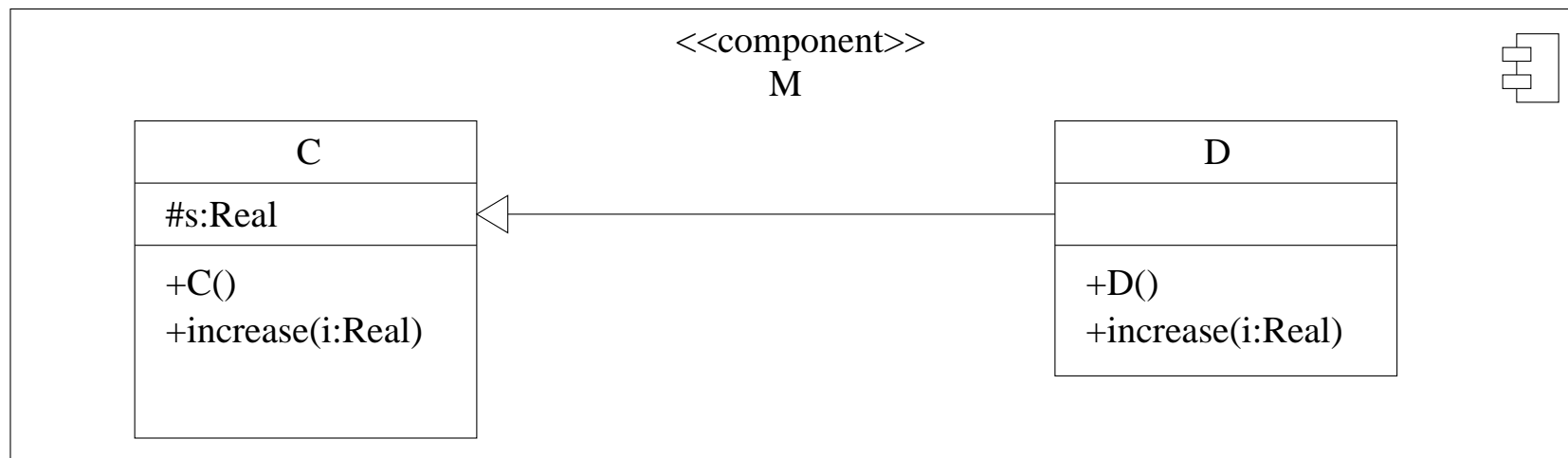
(entsprechend für Methoden mit Rückgabewert und Queryoperationen) gilt:

Für alle  $(\sigma^-, d, v_1, \dots, v_n) \in State_{\Delta} \times ([D] \times [T_1] \times \dots \times [T_n])$  mit  $d \neq null$  und für alle  $\sigma \in State_{\Delta}$  gilt:

1. Falls  $\llbracket P_{op_C}^{Visibility(op_C)} \rrbracket_{\beta, \sigma^-, \sigma^-} = true$ , dann ist  $\llbracket P_{op_D}^{Visibility(op_D)} \rrbracket_{\beta, \sigma^-, \sigma^-} = true$  (mit  $\beta(self) = d$ ,  $\beta(x_i) = v_i$ ).
2. Falls  $\llbracket P_{op_D}^{Visibility(op_D)} \rrbracket_{\beta, \sigma^-, \sigma^-} = true$  und  $\llbracket Q_{op_D}^{Visibility(op_D)} \rrbracket_{\beta, \sigma^-, \sigma} = true$ , dann ist  $\llbracket Q_{op_C}^{Visibility(op_C)} \rrbracket_{\beta, \sigma^-, \sigma} = true$  (mit  $\beta(self) = d$ ,  $\beta(x_i) = v_i$ ).

**Beispiel:**

Die folgende Komponentenspezifikation ist verhaltensverträglich bzgl. Subtypen:



```
context C
```

```
  inv: s >= 0 and s <= 20
```

```
context C::increase(i:Real)
```

```
  pre: i >= 0 and s+i <= 20
```

```
  post: s <= s@pre + i and s > s@pre
```

```
context D::increase(i:Real)
```

```
  pre: i >= 0 and s+i/2 <= 20
```

```
  post: s = s@pre + i/2
```

## Definition (Lokalitätsprinzip für Klasseninvarianten):

- Eine Klasseninvariante  $\text{context } C \text{ inv: Inv}$  erfüllt das Lokalitätsprinzip, wenn sie keine Eigenschaften für "entfernte" Objekte, d.h. für Objekte, die verschieden sind von der Interpretation von "self", verlangt.
- Eine Komponentenspezifikation  $\text{CompSpec} = (\langle M, \Delta \rangle, \text{OpSpecs}, \text{Invs}^e)$  erfüllt das Lokalitätsprinzip für Klasseninvarianten, wenn jede Klasseninvariante in  $\text{Invs}^e$  das Lokalitätsprinzip erfüllt.

**Beispiel:**

```
context A
  inv: self.myB.y >= 0
```

```
context B
  inv: A.allInstances() -> forAll(a |
    a.x >= 0)
```

Keine der beiden Invarianten erfüllt das Lokalitätsprinzip!

## 4.6 Zusammenfassung

- Ein  $\Sigma_{\Delta}$ -Transitionssystem  $\mathcal{T}_{\Delta}$  ist ein markiertes, nicht-deterministisches Transitionssystem mit Output, das als eine Abstraktion eines nicht-deterministischen, objekt-orientierten Programms verstanden werden kann, das alle Operationen von  $\Sigma_{\Delta}$  unterstützt.
- Ein  $\Sigma_{\Delta}$ -Transitionssystem  $\mathcal{T}_{\Delta}$  ist ein Modell einer Komponentenspezifikation *CompSpec*, wenn die Komponenteninvarianten im Anfangszustand gelten und  $\mathcal{T}_{\Delta}$  alle Operationsspezifikationen von *CompSpec* erfüllt (wobei entsprechend der Sichtbarkeit einer Operation die Klassen- und Komponenteninvarianten einbezogen werden müssen).
- Jedes Modell einer Komponentenspezifikation *CompSpec* kann als eine mögliche, korrekte Realisierung von *CompSpec* verstanden werden.
- Wichtige Eigenschaften von Komponentenspezifikationen sind Konsistenz, Verhaltensverträglichkeit bzgl. Subtypen und das Lokaliätsprinzip für Klasseninvarianten.