

Formale objektorientierte Software-Entwicklung

Prof. Dr. Rolf Hennicker

22.04.2004

Syntax und Semantik von OCL-Ausdrücken

Ziele

- Grundbegriffe wie z.B. Signaturen, Sorten (Typen) und Terme kennen.
- OCL-Signaturen und die Syntax von OCL-Ausdrücken kennen.
- Die Formalisierung von Systemzuständen verstehen.
- Die Semantik von OCL-Ausdrücken kennen.

2.1 Formale Grundlagen: Signaturen und Terme

Definition (Signatur):

Eine Signatur $\Sigma = (S, OP)$ besteht aus

- einer Menge S von Sorten, auch Typen genannt (Namen für Datenbereiche)
- einer Menge OP von Operationen (Namen für Funktionen) der Form

$$op : s_1 \times \dots \times s_n \rightarrow s \text{ mit } s_1, \dots, s_n, s \in S \ (n \geq 0)$$

Definition (Sorten-geordnete Signatur):

Eine sorten-geordnete Signatur $\Sigma = (S, \leq, OP)$ besitzt zusätzlich eine partielle Ordnung \leq auf der Menge S der Sorten (i.e. $\leq \subseteq S \times S$)

Beispiel (Keller):

$$S = \{Stack, Elem, EStack, EElem\}$$

$$\leq = \{Stack \leq EStack, \\ Elem \leq EElem, \\ Stack \leq Stack, \\ Elem \leq Elem, \\ EStack \leq EStack, \\ EElem \leq EElem\}$$

$$OP = \{emptyStack : \rightarrow Stack, \\ push : Stack \times Elem \rightarrow Stack, \\ pop : Stack \rightarrow EStack, \\ top : Stack \rightarrow EElem, \\ errorStack : \rightarrow EStack, \\ errorElem : \rightarrow EElem\}$$

$$\Sigma_{STACK} = (S, \leq, OP)$$

Abkürzung:

$$\leq = \{Stack \leq EStack, Elem \leq EElem\}$$

Definition (Terme):

Sei $\Sigma = (S, \leq, OP)$ eine sorten-geordnete Signatur und sei $X = (X_s)_{s \in S}$ eine Familie von Mengen X_s von Variablen des Typs s .

Die Familie $T(\Sigma, X) = (T(\Sigma, X)_s)_{s \in S}$ von Mengen $T(\Sigma, X)_s$ von (Σ, X) -Termen des Typs s wird induktiv definiert wie folgt:

1. Falls $x \in X_s$ dann $x \in T(\Sigma, X)_s$
2. Falls $(op : s_1, \dots, s_n \rightarrow s) \in OP$ ($n \geq 0$)
und $t_1 \in T(\Sigma, X)_{r_1}, \dots, t_n \in T(\Sigma, X)_{r_n}$ mit $r_i \leq s_i$ (für $i = 1, \dots, n$)
dann $op(t_1, \dots, t_n) \in T(\Sigma, X)_s$

Anmerkung:

Operationssymbole der Form $_op_ : s_1 \times s_2 \rightarrow s$ weisen auf Mixfixdarstellung hin.

Beispiel (Keller):

Seien $e : Elem, s : Stack$ getypte Variablen.

$push(s, e), pop(push(s, e)),$
 $top(push(emptyStack, e))$ sind Terme der Typen $Stack, EStack, EElem$.

Definition (Freie Variablen):

Sei $t \in T(\Sigma, X)$.

$FV(t) =_{def} \{x \in X \mid x \text{ kommt in } t \text{ vor}\}$

z.B. $FV(push(emptyStack, e)) = \{e\}$

2.2 OCL-Signatur

Gegeben sei ein Klassendiagramm Δ . Die OCL-Signatur Σ_{Δ}^{OCL} stellt die Typen und Operationen zur Verfügung, welche für OCL-Ausdrücke benötigt werden.

Beispiel (Punkte und Formen):

$$\begin{aligned} \Sigma_{\Delta}^{OCL} &= (S_{\Delta}^{OCL}, \leq, OP_{\Delta}^{OCL}) \\ S_{\Delta}^{OCL} &= \{OclAny, OclVoid, Real, Integer, Boolean, String, Null, \\ &\quad Point, CPoint, Shape, System, \\ &\quad Collection(OclAny), \dots, Collection(System), \\ &\quad Set(OclAny), \dots, Set(System), \dots\} \\ \leq &= \{Real \leq OclAny, \dots, System \leq OclAny, \\ &\quad Integer \leq Real, CPoint \leq Point, \\ &\quad Set(Point) \leq Collection(Point), \\ &\quad Set(CPoint) \leq Set(Point), \dots\} \\ OP_{\Delta}^{OCL} &= F_{\Delta}^{OCL} \cup Inst_{\Delta}^{OCL} \cup A_{\Delta} \text{ wobei} \\ F_{\Delta}^{OCL} &= \{ _ = _ : OclAny \times OclAny \rightarrow Boolean, \\ &\quad _ = _ : Set(Point) \times Set(Point) \rightarrow Boolean, \\ &\quad _ + _ : Real \times Real \rightarrow Real, \dots \} \end{aligned}$$

$$\begin{aligned}
 & _and_ : Boolean \times Boolean \rightarrow Boolean, \dots \\
 & _ \rightarrow includes(_) : Collection(OclAny) \times OclAny \rightarrow Boolean, \\
 & _ \rightarrow including(_) : Set(Point) \times Point \rightarrow Set(Point), \\
 & _ \rightarrow including(_) : Set(Shape) \times Shape \rightarrow Set(Shape), \dots, \\
 & null : \rightarrow Null\}
 \end{aligned}$$

$$\begin{aligned}
 Inst_{\Delta}^{OCL} = & \{Point.allInstances() : \rightarrow Set(Point), \\
 & CPoint.allInstances() : \rightarrow Set(CPoint), \\
 & Shape.allInstances() : \rightarrow Set(Shape), \\
 & System.allInstances() : \rightarrow Set(System)\}
 \end{aligned}$$

$$\begin{aligned}
 A_{\Delta} = & \{ _ .xx : Point \rightarrow Real, \\
 & _ .yy : Point \rightarrow Real, \\
 & _ .owner : Point \rightarrow Shape, \\
 & _ .coulor : CPoint \rightarrow Real, \\
 & _ .points : Shape \rightarrow Set(Point), \\
 & _ .pointSet : System \rightarrow Set(Point), \\
 & _ .figures : System \rightarrow Set(Shape)\}
 \end{aligned}$$

Terme sind z.B. $self.xx$, $self.xx + 7 = self.yy$
 $p.owner.points \rightarrow includes(p)$
wobei $self : Point$ und $p : Point$ Variablen sind.

Definition (OCL-Signatur):

Sei Δ ein Klassendiagramm. Die sorten-geordnete *OCL-Signatur* bezüglich Δ ist definiert als $\Sigma_{\Delta}^{OCL} = (S_{\Delta}^{OCL}, \leq, OP_{\Delta}^{OCL})$ wobei

1.

$$S_{\Delta}^{OCL} = (Base^{OCL} \cup Class_{\Delta} \cup Coll_{\Delta}^{OCL}) \text{ wobei}$$

$$Base^{OCL} = \{OclAny, OclVoid, \\ Real, Integer, Boolean, String, Null\} \text{ OCL – Grundtypen}$$

$$Class_{\Delta} = \{C \mid C \text{ ist der Name einer Klasse aus } \Delta\}$$

$$Coll_{\Delta}^{OCL} = \{Collection(T) \mid T \in Base^{OCL} \cup Class_{\Delta}\} \cup \\ \{Set(T) \mid T \in Base^{OCL} \cup Class_{\Delta}\} \cup \\ \{Sequence(T) \mid T \in Base^{OCL} \cup Class_{\Delta}\} \cup \\ \{Bag(T) \mid T \in Base^{OCL} \cup Class_{\Delta}\}$$

2. $\leq \subseteq S_{\Delta}^{OCL} \times S_{\Delta}^{OCL}$ ist die (bezüglich \subseteq) kleinste Relation, welche die folgenden Bedingungen erfüllt:

(a) \leq ist reflexiv und transitiv,

(b) *Integer* \leq *Real*,

(c) für jedes $T \in Base^{OCL} \cup Class_{\Delta}$ gilt $T \leq OclAny$,

(d) für jedes $T \in S_{\Delta}^{OCL}$ gilt $OclVoid \leq T$,

(e) für jedes $C \in Class_{\Delta}$ gilt $Null \leq C$,

(f) falls $B, C \in Class_{\Delta}$ und  in Δ vorkommt, dann $C \leq B$,

(g) falls $Coll \in \{Set, Sequence, Bag\}$ und $T \in Base^{OCL} \cup Class_{\Delta}$, dann $Coll(T) \leq Collection(T)$,

(h) falls $Coll \in \{Set, Sequence, Bag, Collection\}$ und $R, T \in Base^{OCL} \cup Class_{\Delta}$, und $R \leq T$ dann $Coll(R) \leq Coll(T)$.

3.

$$\begin{aligned}
 OP_{\Delta}^{OCL} &= (F_{\Delta}^{OCL} \cup Inst_{\Delta}^{OCL} \cup A_{\Delta}) \text{ wobei} \\
 F_{\Delta}^{OCL} &= \text{OCL-Operationen auf OclAny} \\
 &\quad (\text{z.B. } _ = _ : OclAny \times OclAny \rightarrow Boolean, \\
 &\quad _.oclIsUndefined() : OclAny \rightarrow Boolean, \\
 &\quad _.oclIsTypeOf(T) : OclAny \rightarrow Boolean \text{ fuer } T \in S_{\Delta}^{OCL}) \\
 &\cup \text{OCL-Operationen auf Grundtypen} \\
 &\quad (\text{z.B. } +, -, *, \textit{and}, \textit{or}, \dots) \\
 &\cup \text{OCL-Operationen auf Kollektionstypen} \\
 &\quad (\text{z.B. } _ \rightarrow size() : Collection(OclAny) \rightarrow Integer, \\
 &\quad _ \rightarrow includes(_) : Collection(OclAny) \times OclAny \rightarrow Boolean, \\
 &\quad _ \rightarrow includesAll(_) : \\
 &\quad \quad Collection(OclAny) \times Collection(OclAny) \rightarrow Boolean, \\
 &\quad _ = _ : Set(T) \times Set(T) \rightarrow Boolean, \\
 &\quad _ \rightarrow including(_) : Set(T) \times T \rightarrow Set(T)) \\
 &\cup \text{null-Konstante } (null : \rightarrow Null) \\
 Inst_{\Delta}^{OCL} &= \{C.allInstances() : \rightarrow Set(C) \mid C \in Class_{\Delta}\}
 \end{aligned}$$

A_Δ enthält

- für jedes Attribut $a : T$ einer Klasse C aus Δ
 $_.a : C \rightarrow T$
- für jeden Rollennamen r an einem Assoziationsende



- $_.r : C \rightarrow D$ falls $mult \leq 1$
- $_.r : C \rightarrow Set(D)$ falls $mult > 1$
- $_.r : C \rightarrow Sequence(D)$ falls $mult > 1$ mit *Property* {ordered}

2.3 OCL-Ausdrücke

Gegeben sei ein Klassendiagramm Δ sowie die zugehörige OCL-Signatur $\Sigma_{\Delta}^{OCL} = (S_{\Delta}^{OCL}, \leq, OP_{\Delta}^{OCL})$.

Definition (OCL-Ausdrücke):

Sei $X = (X_T)_{T \in S_{\Delta}^{OCL}}$ eine Familie von Mengen X_T von Variablen des Typs T , so dass für jedes $C \in Class_{\Delta}$ eine spezielle Variable *self* des Typs C und für jedes $T \in S_{\Delta}^{OCL}$ eine spezielle Variable *result* des Typs T gegeben ist (i.e. $self \in X_C$, $result \in X_T$).

Die Familie $EXP^{OCL} = (EXP_T^{OCL})_{T \in S_{\Delta}^{OCL}}$ von Mengen EXP_T^{OCL} von OCL-Ausdrücken des Typs T ist induktiv definiert wie folgt:

1. Falls $x \in X_T$ dann $x \in EXP_T^{OCL}$.
2. Falls $op \in OP_{\Delta}^{OCL}$, $op : T_1 \times \dots \times T_n \rightarrow T$ (mit $n \geq 0$)
und $e_1 \in EXP_{U_1}^{OCL}, \dots, e_n \in EXP_{U_n}^{OCL}$ mit $U_i \leq T_i$ (für $i = 1, \dots, n$)
dann $op(e_1, \dots, e_n) \in EXP_T^{OCL}$ (eventuell auch in Mixfix-Notation).

z.B. $e1 = e2$, $b1$ and $b2$, $b1$ implies $b2$, not $b1$
 $Point.allInstances() \rightarrow includes(p)$
 $s.oclIsTypeOf(CPoint)$
 $self.xx = 0$
 $result = self.points \rightarrow includes(p)$
3. Falls $e_B \in EXP_{Boolean}^{OCL}$ und $e_1 \in EXP_{T_1}^{OCL}$, $e_2 \in EXP_{T_2}^{OCL}$
so dass ein kleinster Obertyp T von T_1 und T_2 existiert
dann $if\ e_B\ then\ e_1\ else\ e_2\ endif \in EXP_T^{OCL}$.
4. Falls $e \in EXP_{Coll(T)}^{OCL}$ (mit $Coll \in \{Set, Sequence, Bag\}$), $i \in X_T$, $acc \in X_S$
und $e_1 \in EXP_{S_1}^{OCL}$, $e_2 \in EXP_S^{OCL}$ mit $S_1 \leq S$
dann $e \rightarrow iterate(i : T; acc : S = e_1 \mid e_2) \in EXP_S^{OCL}$.

z.B. $self.points \rightarrow iterate(p : Point; acc : Real = 0 \mid acc + p.xx)$

5. Vordefinierte Iterator-Ausdrücke:

Falls $e, e' \in EXP_{Coll(T)}^{OCL}$ (mit $Coll \in \{Set, Sequence, Bag\}$)

und $x \in X_T$ und $e_B \in EXP_{Boolean}^{OCL}$

dann

- $e \rightarrow forAll(x : T \mid e_B) =_{def}$
 $e \rightarrow iterate(x : T; res : Boolean = true \mid res \text{ and } e_B) \in EXP_{Boolean}^{OCL}$
- $e \rightarrow exists(x : T \mid e_B) =_{def}$
 $e \rightarrow iterate(x : T; res : Boolean = false \mid res \text{ or } e_B) \in EXP_{Boolean}^{OCL}$
- $e \rightarrow select(x : T \mid e_B) =_{def}$
 $e \rightarrow iterate(x : T; res : Coll(T) = Coll\{\} \mid$
 $if e_B \text{ then } res \rightarrow including(x) \text{ else } res \text{ endif}) \in EXP_{Coll(T)}^{OCL}$
- $e \rightarrow any(x : T \mid e_B) =_{def}$
 $e \rightarrow select(x : T \mid e_B) \rightarrow asSequence() \rightarrow first() \in EXP_T^{OCL}$
- $e \rightarrow one(x : T \mid e_B) =_{def}$
 $e \rightarrow select(x : T \mid e_B) \rightarrow size() = 1 \in EXP_{Boolean}^{OCL}$

Auswertungsalgorithmus für $e \rightarrow \text{iterate}(i : T; \text{acc} : S = e_1 \mid e_2)$

```
S acc = e1;  
T i;  
forall i in e do  
  acc = e2;  
return acc;
```

Anmerkung

Die obige Definition umfasst die wesentlichen Konstrukte der Sprache OCL. Die vollständige Syntax findet sich in der OCL 2.0 Spezifikation.

Abkürzungen

- Sei $(_.a : C \rightarrow T) \in A_\Delta$. Dann steht a für $self.a$
- Sei e ein OCL Ausdruck des Typs $Coll(T)$ mit $Coll \in \{Set, Sequence, Bag\}$.
Dann steht $e \rightarrow forall(x | e_B)$ für $e \rightarrow forall(x : T | e_B)$
und Analoges gilt für $exists, select, any, one, iterate$

Die Spezifikation von Nachbedingungen erfordert zusätzliche Konstrukte zur

- Bezugnahme auf "vorherige" Werte von Attributen und Rollen ($@pre$)
- Beschreibung der Erzeugung von neuen Objekten ($oclIsNew()$)

Definition (Erweiterte OCL-Ausdrücke):

Die Familie $EEXP^{OCL} = (EEXP_T^{OCL})_{T \in S_{\Delta}^{OCL}}$ von Mengen $EEXP_T^{OCL}$ von *erweiterten OCL-Ausdrücken* ist definiert durch die Bedingungen (1) - (5) für OCL-Ausdrücke (unter Ersetzung von EXP gegen $EEXP$) und durch die folgenden zusätzlichen Bedingungen:

6. Falls $C \in Class_{\Delta}$, dann $C.allInstances@pre() \in EEXP_{Set(C)}^{OCL}$.
7. Falls $C \in Class_{\Delta}$, $(_a : C \rightarrow T) \in A_{\Delta}$ und $e \in EEXP_D^{OCL}$ mit $D \leq C$
dann $e.a@pre \in EEXP_T^{OCL}$
8. Falls $C \in Class_{\Delta}$, und $e \in EEXP_C^{OCL}$
dann $e.oclIsNew() =_{def}$
 $C.allInstances@pre() \rightarrow excludes(e)$ and $C.allInstances() \rightarrow includes(e)$

Definition (Freie Variablen):

Für jeden (erweiterten) OCL-Ausdruck e ist die Menge $FV(e)$ der *freien Variablen* von e induktiv definiert wie folgt:

- $FV(x) = \{x\} \quad (x \in X)$
- $FV(op(e_1, \dots, e_n)) = FV(e_1) \cup \dots \cup FV(e_n) \quad \text{mit } (op \in OP_{\Delta}^{OCL})$
- $FV(\text{if } e_B \text{ then } e_1 \text{ else } e_2) = FV(e_B) \cup FV(e_1) \cup FV(e_2)$
- $FV(e \rightarrow \text{iterate}(i : T; acc : S = e_1 \mid e_2)) = FV(e) \cup ((FV(e_1) \cup FV(e_2)) \setminus \{i, acc\})$
- $FV(e \rightarrow \text{forAll}(x : T \mid e_B)) = FV(e) \cup (FV(e_B) \setminus \{x\})$
und Analoges gilt für *exists*, *select*, *any*, *one*
- $FV(C.allInstances@pre()) = \emptyset$
- $FV(e.a@pre) = FV(e)$
- $FV(e.oclIsNew()) = FV(e)$

2.4 Semantische Bereiche für OCL-Typen

$\llbracket OclVoid \rrbracket$	$= \{\perp\}$	<i>undefiniert</i>
$\llbracket Integer \rrbracket$	$= \mathbb{Z} \cup \{\perp\}$	<i>ganze Zahlen (mit \perp)</i>
$\llbracket Real \rrbracket$	$= \mathbb{R} \cup \{\perp\}$	<i>reelle Zahlen</i>
$\llbracket Boolean \rrbracket$	$= \{true, false, \perp\}$	<i>Wahrheitswerte</i>
$\llbracket String \rrbracket$	$= \mathcal{A}^* \cup \{\perp\}$	<i>endliche Strings über einem Alphabet \mathcal{A}</i>
$\llbracket Null \rrbracket$	$= \{null, \perp\}$	

Für jeden Klassennamen $C \in Class_{\Delta}$ sei OId_C eine abzählbar unendliche Menge von Objektidentitäten des Typs C und $OID_C = \bigcup \{OId_D \mid D \in Class_{\Delta}, D \leq C\}$.

$\llbracket C \rrbracket$	$= OId_C \cup \{null, \perp\}$	
$\llbracket OclAny \rrbracket$	$= \bigcup \{\llbracket T \rrbracket \mid T \in Base^{OCL} \cup Class_{\Delta}, T \neq OclAny\}$	
$\llbracket Set(T) \rrbracket$	$= \mathcal{F}(\llbracket T \rrbracket) \cup \{\perp\}$	<i>endliche Mengen ueber $\llbracket T \rrbracket$</i>
$\llbracket Sequence(T) \rrbracket$	$= \llbracket T \rrbracket^* \cup \{\perp\}$	<i>endliche Sequenzen ueber $\llbracket T \rrbracket$</i>
$\llbracket Bag(T) \rrbracket$	$= \mathcal{B}(\llbracket T \rrbracket) \cup \{\perp\}$	<i>endl. Multimengen ueber $\llbracket T \rrbracket$</i>
$\llbracket Collection(T) \rrbracket$	$= \mathcal{F}(\llbracket T \rrbracket) \cup \llbracket T \rrbracket^* \cup \mathcal{B}(\llbracket T \rrbracket) \cup \{\perp\}$	

Lemma 1 (Semantische Bereiche sind verträglich mit \leq):

Für alle $T, T' \in S_{\Delta}^{OCL}$ gilt:

Falls $T \leq T'$ dann $\llbracket T \rrbracket \subseteq \llbracket T' \rrbracket$.

(Beweis durch Induktion über die Definition von \leq .)

2.5 Interpretationen für Standard-OCL-Operationen

Die Interpretation einer Standard-Operation $(op : T_1 \times \dots \times T_n \rightarrow T) \in F_{\Delta}^{OCL}$ ist eine Funktion $\llbracket op \rrbracket : \llbracket T_1 \rrbracket \times \dots \times \llbracket T_n \rrbracket \rightarrow \llbracket T \rrbracket$.

Beispiel (Interpretation von $_ + _$):

$_ + _ : Real \times Real \rightarrow Real$ wird interpretiert als

$\llbracket _ + _ \rrbracket : \llbracket Real \rrbracket \times \llbracket Real \rrbracket \rightarrow \llbracket Real \rrbracket$,

$$\llbracket _ + _ \rrbracket(x, y) = \begin{cases} x + y & \text{falls } x \neq \perp \text{ und } y \neq \perp \\ \perp & \text{falls } x = \perp \text{ oder } y = \perp \end{cases}$$

Definition (Striktheit):

Eine Funktion $\llbracket op \rrbracket : \llbracket T_1 \rrbracket \times \dots \times \llbracket T_n \rrbracket \rightarrow \llbracket T \rrbracket$ heißt strikt,

falls aus $x_1 = \perp \vee \dots \vee x_n = \perp$ stets folgt $\llbracket op \rrbracket(x_1, \dots, x_n) = \perp$.

1. Operationen auf *OclAny*

$$\llbracket - = - \rrbracket : \llbracket OclAny \rrbracket \times \llbracket OclAny \rrbracket \rightarrow \llbracket Boolean \rrbracket$$

$$\llbracket - = - \rrbracket(x, y) = \begin{cases} true & \text{falls } x = y \text{ und } x \neq \perp, y \neq \perp \\ false & \text{falls } x \neq y \text{ und } x \neq \perp, y \neq \perp \\ \perp & \text{falls } x = \perp \text{ oder } y = \perp \end{cases}$$

$$\llbracket -.oclIsUndefined() \rrbracket : \llbracket OclAny \rrbracket \rightarrow \llbracket Boolean \rrbracket$$

$$\llbracket -.oclIsUndefined() \rrbracket(x) = \begin{cases} true & \text{falls } x = \perp \\ false & \text{sonst} \end{cases}$$

$$\llbracket -.oclIsTypeOf(T) \rrbracket : \llbracket OclAny \rrbracket \rightarrow \llbracket Boolean \rrbracket$$

$$\llbracket -.oclIsTypeOf(T) \rrbracket(x) = \begin{cases} true & \text{falls } x \in \llbracket T \rrbracket \setminus \bigcup \{ \llbracket T' \rrbracket \mid T' < T \} \\ false & \text{sonst} \end{cases}$$

2. **Arithmetische Operationen**

werden durch die strikten Fortsetzungen ihrer Standardbedeutungen interpretiert.

3. **Boolsche Operationen**

$$\llbracket \text{not_} \rrbracket : \llbracket \text{Boolean} \rrbracket \rightarrow \llbracket \text{Boolean} \rrbracket$$

$$\llbracket \text{not_} \rrbracket(x) = \begin{cases} \text{true} & \text{falls } x = \text{false} \\ \text{false} & \text{falls } x = \text{true} \\ \perp & \text{sonst} \end{cases}$$

$$\llbracket \text{_and_} \rrbracket : \llbracket \text{Boolean} \rrbracket \times \llbracket \text{Boolean} \rrbracket \rightarrow \llbracket \text{Boolean} \rrbracket$$

$$\llbracket \text{_and_} \rrbracket(x, y) = \begin{cases} \text{false} & \text{falls } x = \text{false} \text{ oder } y = \text{false} \\ \text{true} & \text{falls } x = \text{true} \text{ und } y = \text{true} \\ \perp & \text{sonst} \end{cases}$$

$$\llbracket \text{_or_} \rrbracket : \llbracket \text{Boolean} \rrbracket \times \llbracket \text{Boolean} \rrbracket \rightarrow \llbracket \text{Boolean} \rrbracket$$

$$\llbracket \text{_or_} \rrbracket(x, y) = \begin{cases} \text{true} & \text{falls } x = \text{true} \text{ oder } y = \text{true} \\ \text{false} & \text{falls } x = \text{false} \text{ und } y = \text{false} \\ \perp & \text{sonst} \end{cases}$$

$$\llbracket _ \text{implies} _ \rrbracket : \llbracket Boolean \rrbracket \times \llbracket Boolean \rrbracket \rightarrow \llbracket Boolean \rrbracket$$

$$\llbracket _ \text{implies} _ \rrbracket(x, y) = \begin{cases} true & \text{falls } x = false \text{ oder } (x = true \text{ und } y = true) \\ false & \text{falls } x = true \text{ und } y = false \\ \perp & \text{sonst} \end{cases}$$

Anmerkung: $\llbracket _ \text{implies} _ \rrbracket(x, y) = \llbracket _ \text{or} _ \rrbracket(\llbracket not \rrbracket(x), \llbracket _ \text{and} _ \rrbracket(x, y))$

4. Operationen auf Kollektionstypen

werden durch strikte Funktionen interpretiert, z.B.

$$\llbracket _ \rightarrow \text{includes}(_) \rrbracket : \llbracket Collection(OclAny) \rrbracket \times \llbracket OclAny \rrbracket \rightarrow \llbracket Boolean \rrbracket$$

$$\llbracket _ \rightarrow \text{includes}(_) \rrbracket(s, x) = \begin{cases} true & \text{falls } x \in s \text{ und } x \neq \perp, s \neq \perp \\ false & \text{falls } x \notin s \text{ und } x \neq \perp, s \neq \perp \\ \perp & \text{falls } x = \perp \text{ oder } s = \perp \end{cases}$$

$$\llbracket - \rightarrow including(-) \rrbracket : \llbracket Set(T) \rrbracket \times \llbracket T \rrbracket \rightarrow \llbracket Set(T) \rrbracket$$

$$\llbracket - \rightarrow including(-) \rrbracket (s, x) = \begin{cases} s \cup \{x\} & \text{falls } s \neq \perp \text{ und } x \neq \perp \\ \perp & \text{falls } s = \perp \text{ oder } x = \perp \end{cases}$$

$$\llbracket - \rightarrow including(-) \rrbracket : \llbracket Sequence(T) \rrbracket \times \llbracket T \rrbracket \rightarrow \llbracket Sequence(T) \rrbracket$$

$$\llbracket - \rightarrow including(-) \rrbracket (s, x) = \begin{cases} s \circ \langle x \rangle & \text{falls } s \neq \perp, x \neq \perp \\ \perp & \text{falls } s = \perp \text{ oder } x = \perp \end{cases}$$

5. Null-Konstante

$$\llbracket null \rrbracket = null$$

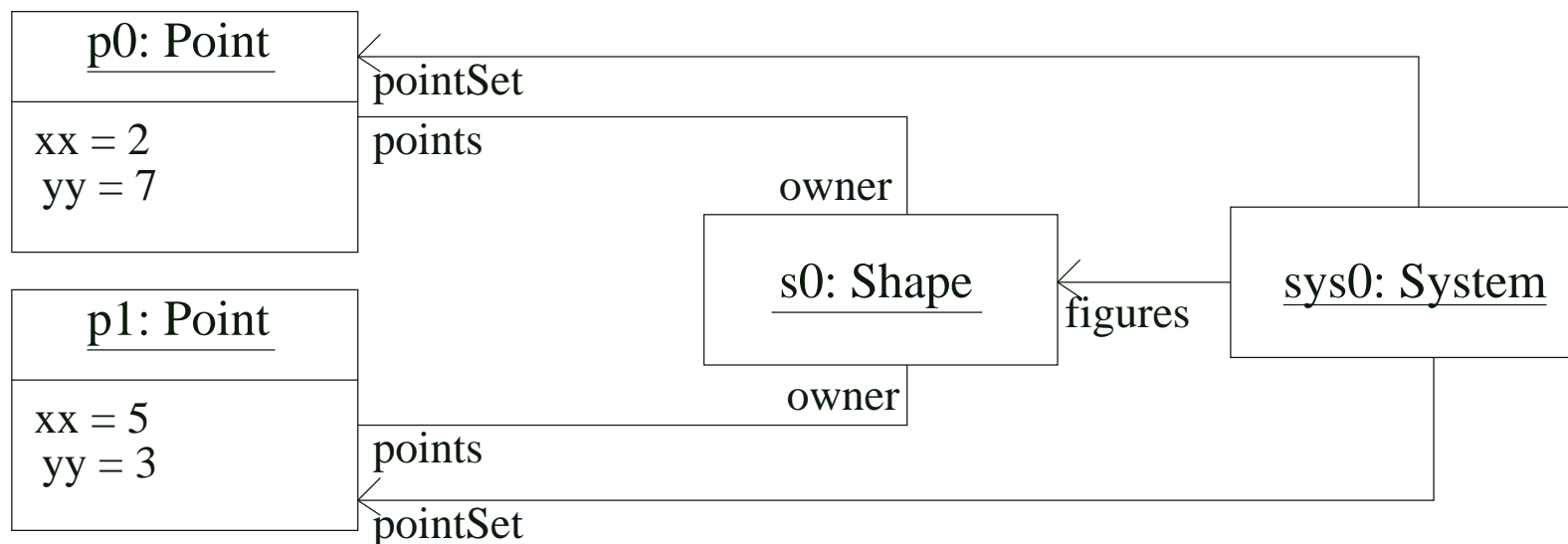
2.6 Formalisierung von Systemzuständen

Im Folgenden sei Δ ein Klassendiagramm.

Graphische Darstellung von Systemzuständen

Der Zustand des (von Δ beschriebenen) Systems zu einem bestimmten Zeitpunkt kann als ein Objektdiagramm für Δ dargestellt werden ("Snapshot").

Beispiel (Punkte und Formen):



Mathematische Formalisierung (Grundidee)

Ein Systemzustand besteht aus

1. einer Menge augenblicklich existierender Objekte (für jede Klasse $C \in Class_{\Delta}$),
2. einer Funktion, welche für jedes Paar $o.a$ (mit einem Objekt o und einem Attribut oder Rollennamen a) einen Wert liefert.

Beispiel (Punkte und Formen):

existierende Objekte der Klasse `Point`: {p0, cp0}

existierende Objekte der Klasse `CPoint`: {cp0}

existierende Objekte der Klasse `Shape`: {s0}

existierende Objekte der Klasse `System`: {sys0}

$p0.xx \mapsto 2$

$cp0.xx \mapsto 5$

$s0.points \mapsto \{p0, cp0\}$

$p0.yy \mapsto 7$

$cp0.yy \mapsto 3$

$sys0.figures \mapsto \{s0\}$

$p0.owner \mapsto s0$

$cp0.owner \mapsto s0$

$sys0.pointSet \mapsto \{p0, cp0\}$

$cp0.colour \mapsto 0.8$

Definition (Links- und Rechtswerte):

1. Die Menge $LVal_{\Delta}$ der *Linkswerte* (bezüglich Δ) ist gegeben durch

$$LVal_{\Delta} = \{o.a \mid o \in OID_C \text{ mit} \\ C \in Class_{\Delta} \text{ und } (..a : C \rightarrow T) \in A_{\Delta}\}$$

2. Die Menge $RVal_{\Delta}$ der *Rechtswerte* (bezüglich Δ) ist gegeben durch

$$RVal_{\Delta} = \bigcup \{ \llbracket T \rrbracket \mid T \in S_{\Delta}^{OCL} \}$$

Anmerkung:

$$RVal_{\Delta} = \llbracket OclAny \rrbracket \cup \llbracket Collection(OclAny) \rrbracket$$

Definition (Systemzustand):

Ein Systemzustand (bezüglich Δ) ist ein Paar $\sigma = (\sigma_{Instances}, \sigma_{Val})$ von Funktionen

- $\sigma_{Instances} : Class_{\Delta} \rightarrow \bigcup \{ \mathcal{F}(OID_C) \mid C \in Class_{\Delta} \}$

so dass gilt:

(a) $\sigma_{Instances}(C) \subseteq OID_C$ für jedes $C \in Class_{\Delta}$

(b) $\sigma_{Instances}(D) = \{ o \in \sigma_{Instances}(C) \mid o \in OID_D \}$ für alle $C, D \in Class_{\Delta}$ mit $D \leq C$

Notation:

Wir schreiben C_{σ} für $\sigma_{Instances}(C)$.

- $\sigma_{Val} : LVal_{\Delta} \rightarrow RVal_{\Delta}$

so dass für jedes $o \in OID_C$ und $(_.a : C \rightarrow T) \in A_{\Delta}$ gilt:

$$\sigma_{Val}(o.a) \in \llbracket T \rrbracket$$

Definition:

$State_{\Delta} = \{\sigma \mid \sigma = (\sigma_{Instances}, \sigma_{Val}) \text{ ist Systemzustand bezüglich } \Delta\}$

Notationen

Für $\sigma \in State_{\Delta}$, $o.a \in LVal_{\Delta}$ und $v \in RVal_{\Delta}$ schreiben wir

- $o.a_{\sigma}$ für $\sigma_{Val}(o.a)$
- $\sigma[o.a \mapsto v]$ für den Zustand $\sigma' = (\sigma_{Instances}, \sigma'_{Val})$ mit

$$\begin{aligned}\sigma'_{Val}(o.a) &= v \\ \sigma'_{Val}(x) &= \sigma_{Val}(x) \text{ falls } x \neq o.a\end{aligned}$$

i.e. σ wird mit dem neuen Wert v für $o.a$ aktualisiert

Notationen

Seien $T_1, \dots, T_n \in S_{\Delta}^{OCL}$.

1. $(State_{\Delta} \times ([T_1] \times \dots \times [T_n])) =_{def}$

$$\{(\sigma, v_1, \dots, v_n) \in State_{\Delta} \times [T_1] \times \dots \times [T_n] \mid$$

$$v_i \neq \perp \text{ für } i = 1, \dots, n \text{ und}$$

$$\text{falls } T_i \in Class_{\Delta} \text{ dann } v_i \in (T_i)_{\sigma} \cup \{null\} \text{ und}$$

$$\text{falls } T_i = Coll(D) \text{ mit } D \in Class_{\Delta} \text{ dann } v_i \subseteq (T_i)_{\sigma} \cup \{null\}\}$$

wobei $Coll \in \{Set, Sequence, Bag, Collection\}$

2. Sei $\sigma \in State_{\Delta}$.

$$Valid(\sigma, [T_1] \times \dots \times [T_n]) =_{def}$$

$$\{(v_1, \dots, v_n) \in [T_1] \times \dots \times [T_n] \mid$$

$$(\sigma, v_1, \dots, v_n) \in State_{\Delta} \times ([T_1] \times \dots \times [T_n])\}$$

2.7 Denotationelle Semantik für OCL-Ausdrücke

Grundidee

Die Denotation (Bedeutung) eines OCL-Ausdrucks e vom Typ T ist ein Wert des semantischen Bereiches $\llbracket T \rrbracket$.

Beispiel (Semantik für OCL-Ausdrücke):

- $3 + 2$ bedeutet $5 \in \llbracket Integer \rrbracket$ ($= \mathbb{Z} \cup \{\perp\}$)
- $3 = 2$ bedeutet $false \in \llbracket Boolean \rrbracket$
- Sei $mx : Real$ eine Variable des Typs $Real$.
 $mx + 4.5$ kann nur bezüglich einer Belegung $\beta : X_{Real} \rightarrow \llbracket Real \rrbracket$ interpretiert werden.
z.B. für $\beta(mx) = 2.3$ hat $mx + 4.5$ die Bedeutung 6.8.

- Sei $p : Point$ eine Variable des Typs $Point$.
 $p.xx$ kann nur bezüglich einer Belegung $\beta : X_{Point} \rightarrow \llbracket Point \rrbracket$ und bezüglich eines Zustandes $\sigma \in State_{\Delta}$ interpretiert werden.

z.B. für $\beta(p) = p0$ und für σ mit $p0 \in Point_{\sigma}$ und $p0.xx_{\sigma} = 2$ hat $p.xx$ die Bedeutung 2 (bezüglich β und σ).

- $self.xx = self.xx@pre + mx$ kann nur bezüglich einer Belegung β und bezüglich zweier Zustände $\sigma^{-}, \sigma \in State_{\Delta}$ interpretiert werden.

z.B. für $self \in X_{Point}, mx \in X_{Real}$,
 $\beta(self) = p0, \beta(mx) = 7.4$,
 $p0 \in Point_{\sigma^{-}}, p0.xx_{\sigma^{-}} = 2$,
 $p0 \in Point_{\sigma}, p0.xx_{\sigma} = 5.3$,

besitzt $self.xx = self.xx@pre + mx$ die Bedeutung *false* (bezüglich β, σ^{-} und σ).

Allgemein gilt:

Die Semantik eines (erweiterten) OCL-Ausdruckes e ist definiert bezüglich

- einer Belegung β (für die Variablen)
- zweier Zustände σ^- und σ

Notation: $\llbracket e \rrbracket_{\beta, \sigma^-, \sigma}$

Definition (Variablenbelegung):

Sei $X = (X_T)_{T \in S_{\Delta}^{OCL}}$ eine Familie von Mengen X_T von Variablen des Typs T . Eine Belegung β (der Variablen X) ist eine Familie von Funktionen

$$\beta = (\beta_T : X_T \rightarrow \llbracket T \rrbracket)_{T \in S_{\Delta}^{OCL}}$$

Notationen

- Die Menge aller Belegungen wird mit Env_{Δ} bezeichnet.
- Sei $\beta \in Env_{\Delta}$ und $x \in X_T$. Wir schreiben $\beta(x)$ anstelle von $\beta_T(x)$.
- Sei $\beta \in Env_{\Delta}$, $y \in X_T$ und $v \in \llbracket T \rrbracket$. Wir schreiben $\beta[y \mapsto v]$ für die Belegung β' mit

$$\begin{aligned}\beta'(y) &= v \\ \beta'(x) &= \beta(x) \text{ falls } x \neq y\end{aligned}$$

Definition (Semantik für OCL-Ausdrücke):

Sei $EEXP^{OCL} = (EEXP_T^{OCL})_{T \in S_{\Delta}^{OCL}}$ eine Familie von Mengen $EEXP_T^{OCL}$ von erweiterten OCL-Ausdrücken des Typs T .

Die Semantik von erweiterten OCL-Ausdrücken ist gegeben durch eine Familie von Funktionen

$$\llbracket - \rrbracket_{-, -, -, -} = (\llbracket - \rrbracket_{-, -, -, -, T})_{T \in S_{\Delta}^{OCL}} \text{ wobei für jedes } T \in S_{\Delta}^{OCL}$$

$$\llbracket - \rrbracket_{-, -, -, -, T} : EEXP_T^{OCL} \times Env_{\Delta} \times State_{\Delta} \times State_{\Delta} \rightarrow \llbracket T \rrbracket.$$

Die Funktionen $\llbracket - \rrbracket_{-, -, -, -, T}$ werden gemäß der Struktur erweiterter OCL-Ausdrücke induktiv definiert. Im Folgenden wird diese Definition im Detail dargestellt, wobei aus Gründen der Einfachheit der Index "T" weggelassen wird.

Induktive Definition von $\llbracket e \rrbracket_{\beta, \sigma^-, \sigma}$

1. $\llbracket x \rrbracket_{\beta, \sigma^-, \sigma} = \beta(x)$ für $x \in X_T$
2. Sei $op \in OP_{\Delta}^{OCL} = F_{\Delta}^{OCL} \cup Inst_{\Delta}^{OCL} \cup A_{\Delta}$
 - (a) $op \in F_{\Delta}^{OCL}$:

$$\llbracket op(e_1, \dots, e_n) \rrbracket_{\beta, \sigma^-, \sigma} = \llbracket op \rrbracket(\llbracket e_1 \rrbracket_{\beta, \sigma^-, \sigma}, \dots, \llbracket e_n \rrbracket_{\beta, \sigma^-, \sigma})$$

z.B.

$$\begin{aligned} \llbracket self.xx = self.xx@pre + mx \rrbracket_{\beta, \sigma^-, \sigma} &= \\ \llbracket - = - \rrbracket(\llbracket self.xx \rrbracket_{\beta, \sigma^-, \sigma}, \llbracket self.xx@pre + mx \rrbracket_{\beta, \sigma^-, \sigma}) &= \\ \llbracket - = - \rrbracket(\llbracket self.xx \rrbracket_{\beta, \sigma^-, \sigma}, \llbracket - + - \rrbracket(\llbracket self.xx@pre \rrbracket_{\beta, \sigma^-, \sigma}, \llbracket mx \rrbracket_{\beta, \sigma^-, \sigma})) &= \\ \llbracket - = - \rrbracket(\llbracket self.xx \rrbracket_{\beta, \sigma^-, \sigma}, \llbracket self.xx@pre \rrbracket_{\beta, \sigma^-, \sigma} + \llbracket mx \rrbracket_{\beta, \sigma^-, \sigma}) &= \\ \llbracket - = - \rrbracket(\llbracket self \rrbracket_{\beta, \sigma^-, \sigma}.xx_{\sigma}, \llbracket self \rrbracket_{\beta, \sigma^-, \sigma}.xx_{\sigma^-} + \llbracket mx \rrbracket_{\beta, \sigma^-, \sigma}) &= \\ \llbracket - = - \rrbracket(\beta(self).xx_{\sigma}, \beta(self).xx_{\sigma^-} + \beta(mx)) &= \end{aligned}$$

$$\left\{ \begin{array}{ll} true & \text{falls } \beta(self).xx_{\sigma} = \beta(self).xx_{\sigma^-} + \beta(mx) \\ & \text{und beide Seiten sind } \neq \perp \\ false & \text{falls } \beta(self).xx_{\sigma} \neq \beta(self).xx_{\sigma^-} + \beta(mx) \\ & \text{und beide Seiten sind } \neq \perp \\ \perp & \text{falls eine Seite gleich } \perp \text{ ist} \end{array} \right.$$

$$(b) \llbracket C.allInstances() \rrbracket_{\beta, \sigma^-, \sigma} = C_\sigma$$

z.B. $\llbracket Point.allInstances() \rrbracket_{\beta, \sigma^-, \sigma} = \{p0, cp0\}$ mit σ wie oben

(c) op ist von der Form $(_.a : C \rightarrow T) \in A_\Delta$:

$$\llbracket e.a \rrbracket_{\beta, \sigma^-, \sigma} = \begin{cases} \perp & \text{falls } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} = \perp \text{ oder } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} = null \\ \llbracket e \rrbracket_{\beta, \sigma^-, \sigma}.a_\sigma & \text{sonst} \end{cases}$$

z.B. sei σ der Systemzustand des obigen Beispiels und sei $\beta(p) = p0$,

$$\llbracket p.owner.points \rrbracket_{\beta, \sigma^-, \sigma} = \llbracket p.owner \rrbracket_{\beta, \sigma^-, \sigma}.points_\sigma =$$

$$\llbracket p \rrbracket_{\beta, \sigma^-, \sigma}.owner_\sigma.points_\sigma = \beta(p).owner_\sigma.points_\sigma =$$

$$p0.owner_\sigma.points_\sigma = s0.points_\sigma = \{p0, cp0\}$$

3.

$$\llbracket \text{if } e_B \text{ then } e_1 \text{ else } e_2 \text{ endif} \rrbracket_{\beta, \sigma^-, \sigma} = \begin{cases} \llbracket e_1 \rrbracket_{\beta, \sigma^-, \sigma} & \text{falls } \llbracket e_B \rrbracket_{\beta, \sigma^-, \sigma} = \text{true} \\ \llbracket e_2 \rrbracket_{\beta, \sigma^-, \sigma} & \text{falls } \llbracket e_B \rrbracket_{\beta, \sigma^-, \sigma} = \text{false} \\ \perp & \text{falls } \llbracket e_B \rrbracket_{\beta, \sigma^-, \sigma} = \perp \end{cases}$$

4. Sei $e \in \mathit{EEXP}_{\mathit{Set}(T)}^{\mathit{OCL}}$.

$$\llbracket e \rightarrow \text{iterate}(i : T; \text{acc} : S = e_1 \mid e_2) \rrbracket_{\beta, \sigma^-, \sigma} = \text{it}(\llbracket e \rrbracket_{\beta, \sigma^-, \sigma}, \llbracket e_1 \rrbracket_{\beta, \sigma^-, \sigma})$$

wobei $\text{it} : \llbracket \mathit{Set}(T) \rrbracket \times \llbracket S \rrbracket \rightarrow \llbracket S \rrbracket$ rekursiv definiert wird:

$$\text{it}(m, w) = \begin{cases} w & \text{falls } m = \emptyset \\ \text{it}(m \setminus \{v\}, \llbracket e_2 \rrbracket_{\beta[i \mapsto v, \text{acc} \mapsto w], \sigma^-, \sigma}) & \text{falls } m \neq \perp \text{ und } v \in m \\ \perp & \text{sonst} \end{cases}$$

In ähnlicher Weise definiert man die Semantik von Iteratorausdrücken auch dann, wenn e vom Typ $\mathit{Bag}(T)$ oder $\mathit{Sequence}(T)$ ist, wobei im letzten Fall die Iteration gemäß der Ordnung der Elemente in der Sequenz durchgeführt wird.

5. $\llbracket e \rightarrow \text{forAll}(x : T \mid e_B) \rrbracket_{\beta, \sigma^-, \sigma} =$
 $\llbracket e \rightarrow \text{iterate}(x : T; \text{res} = \text{true} \mid \text{res and } e_B) \rrbracket_{\beta, \sigma^-, \sigma}$
- $\llbracket e \rightarrow \text{exists}(x : T \mid e_B) \rrbracket_{\beta, \sigma^-, \sigma} =$
 $\llbracket e \rightarrow \text{iterate}(x : T; \text{res} = \text{false} \mid \text{res or } e_B) \rrbracket_{\beta, \sigma^-, \sigma}$
- $\llbracket e \rightarrow \text{select}(x : T \mid e_B) \rrbracket_{\beta, \sigma^-, \sigma} =$
 $\llbracket e \rightarrow \text{iterate}(x : T; \text{res} : \text{Coll}(T) = \text{Coll}\{\} \mid$
 $\text{if } e_B \text{ then res} \rightarrow \text{including}(x) \text{ else res endif}) \rrbracket_{\beta, \sigma^-, \sigma}$
- $\llbracket e \rightarrow \text{any}(x : T \mid e_B) \rrbracket_{\beta, \sigma^-, \sigma} =$
 $\llbracket e \rightarrow \text{select}(x : T \mid e_B) \rightarrow \text{asSequence}() \rightarrow \text{first}() \rrbracket_{\beta, \sigma^-, \sigma}$
- $\llbracket e \rightarrow \text{one}(x : T \mid e_B) \rrbracket_{\beta, \sigma^-, \sigma} =$
 $\llbracket e \rightarrow \text{select}(x : T \mid e_B) \rightarrow \text{size}() = 1 \rrbracket_{\beta, \sigma^-, \sigma}$
6. $\llbracket C.\text{allInstances}@pre() \rrbracket_{\beta, \sigma^-, \sigma} = C_{\sigma^-}$

7. Sei $(_.a : C \rightarrow T) \in A_\Delta$.

$$\llbracket e.a@pre \rrbracket_{\beta, \sigma^-, \sigma} = \begin{cases} \perp & \text{falls } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} = \perp \text{ oder } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} = \text{null} \\ \llbracket e \rrbracket_{\beta, \sigma^-, \sigma}.a_{\sigma^-} & \text{sonst} \end{cases}$$

z.B. sei $\beta(\text{self}) = p0$, $\beta(mx) = 7.4$,
 $p0 \in \text{Point}_{\sigma^-}$, $p0.xx_{\sigma^-} = 2$,
 $p0 \in \text{Point}_{\sigma}$, $p0.xx_{\sigma} = 5.3$

$\llbracket \text{self}.xx = \text{self}.xx@pre + mx \rrbracket_{\beta, \sigma^-, \sigma} = \text{false}$ weil
 $\beta(\text{self}).xx_{\sigma} = p0.xx_{\sigma} = 5.3$ und
 $\beta(\text{self}).xx_{\sigma^-} + \beta(mx) = p0.xx_{\sigma^-} + 7.4 = 9.4$

8. Sei $C \in \text{Class}_\Delta$, $e \in \text{EEXP}_C^{\text{OCL}}$.

$$\llbracket e.oclIsNew() \rrbracket_{\beta, \sigma^-, \sigma} = \llbracket C.allInstances@pre() \rightarrow \text{excludes}(e) \text{ and } C.allInstances() \rightarrow \text{includes}(e) \rrbracket_{\beta, \sigma^-, \sigma}$$

z.B. sei $\text{result} : \text{Point}$, $\beta(\text{result}) = p2$, $p2 \notin \text{Point}_{\sigma^-}$, $p2 \in \text{Point}_{\sigma}$
 $\llbracket \text{result.oclIsNew}() \rrbracket_{\beta, \sigma^-, \sigma} = \text{true}$

Lemma:

$$1. \llbracket e \rightarrow \text{forall}(x : T \mid e_B) \rrbracket_{\beta, \sigma^-, \sigma} =$$

$$\left\{ \begin{array}{l} \text{true} \quad \text{falls } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} \neq \perp \text{ und für alle } v \in \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} \\ \quad \llbracket e_B \rrbracket_{\beta[x \mapsto v], \sigma^-, \sigma} = \text{true} \\ \text{false} \quad \text{falls } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} \neq \perp \text{ und es existiert } v \in \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} \\ \quad \text{so dass } \llbracket e_B \rrbracket_{\beta[x \mapsto v], \sigma^-, \sigma} = \text{false} \\ \perp \quad \text{sonst} \end{array} \right.$$

$$2. \llbracket e \rightarrow \text{exists}(x : T \mid e_B) \rrbracket_{\beta, \sigma^-, \sigma} =$$

$$\left\{ \begin{array}{l} \text{true} \quad \text{falls } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} \neq \perp \text{ und es existiert } v \in \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} \\ \quad \text{so dass } \llbracket e_B \rrbracket_{\beta[x \mapsto v], \sigma^-, \sigma} = \text{true} \\ \text{false} \quad \text{falls } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} \neq \perp \text{ und für alle } v \in \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} \\ \quad \llbracket e_B \rrbracket_{\beta[x \mapsto v], \sigma^-, \sigma} = \text{false} \\ \perp \quad \text{sonst} \end{array} \right.$$

Note: $\llbracket e \rightarrow \text{exists}(x : T \mid e_B) \rrbracket_{\beta, \sigma^-, \sigma} =$
 $\llbracket \text{not}(e \rightarrow \text{forAll}(x : T \mid \text{not}(e_B))) \rrbracket_{\beta, \sigma^-, \sigma}$

3.

$$\llbracket e.\text{oclIsNew}() \rrbracket_{\beta, \sigma^-, \sigma} = \begin{cases} \perp & \text{falls } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} = \perp \\ \text{true} & \text{falls } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} \in C_\sigma \text{ und } \llbracket e \rrbracket_{\beta, \sigma^-, \sigma} \notin C_{\sigma^-} \\ \text{false} & \text{sonst} \end{cases}$$

Beispiel:

Sei σ wie oben und $\beta'(self) = s0$,

$$\llbracket self.points \rightarrow \text{exists}(q : Point \mid q.xx > 4) \rrbracket_{\beta', \sigma^-, \sigma} = \text{true}$$

weil für $cp0 \in \llbracket self.points \rrbracket_{\beta', \sigma^-, \sigma} = \{p0, cp0\}$,

$$\begin{aligned} \llbracket q.xx > 4 \rrbracket_{\beta'[q \mapsto cp0], \sigma^-, \sigma} &= (\llbracket q.xx \rrbracket_{\beta'[q \mapsto cp0], \sigma^-, \sigma} > \llbracket 4 \rrbracket \dots) \\ &= (\beta'[q \mapsto cp0](q).xx_\sigma > 4) = (cp0.xx_\sigma > 4) = (5 > 4) = \text{true} \end{aligned}$$

Lemma: Sei e ein (erweiterter) OCL-Ausdruck mit $FV(e) = \{x_1, \dots, x_n\}$.
Seien $\beta, \beta' \in Env_\Delta$ zwei Belegungen, so dass $\beta(x_i) = \beta'(x_i)$ für $i = 1, \dots, n$.
Dann gilt für alle $\sigma^-, \sigma \in State_\Delta$

$$\llbracket e \rrbracket_{\beta, \sigma^-, \sigma} = \llbracket e \rrbracket_{\beta', \sigma^-, \sigma}$$

2.8 Zusammenfassung

Sei Δ ein Klassendiagramm

- Die OCL-Signatur $\Sigma_{\Delta}^{OCL} = (S_{\Delta}^{OCL}, \leq, OP_{\Delta}^{OCL})$ stellt die Typen und Operationen zur Verfügung, welche für (erweiterte) OCL-Ausdrücke benötigt werden.
- Erweiterte OCL-Ausdrücke besitzen zusätzliche Konstrukte für bedingte Ausdrücke (*if-then-else-endif*), Iteration (*iterate, forAll, exists,*), für die Erzeugung neuer Objekte (*oclIsNew*) und zur Referenzierung von Vorzuständen (*@pre*).
- Jeder Typ $T \in S_{\Delta}^{OCL}$ besitzt einen semantischen Bereich $\llbracket T \rrbracket$, so dass aus $T' \leq T$ stets folgt $\llbracket T' \rrbracket \subseteq \llbracket T \rrbracket$.
- $State_{\Delta} = \{\sigma \mid \sigma = (\sigma_{Instances}, \sigma_{Val})\}$ ist die Menge der Systemzustände bezüglich Δ .
- Die Semantik $\llbracket e \rrbracket_{\beta, \sigma^{-}, \sigma}$ eines (erweiterten) OCL-Ausdrucks ist definiert bezüglich
 - einer Belegung β (für die in e frei auftretenden Variablen),
 - zweier Zustände $\sigma^{-}, \sigma \in State_{\Delta}$.