Kapitel 2

Syntax und Semantik von OCL-Ausdrücken

Prof. Dr. Rolf Hennicker

29.04.2010

Ziele

- Grundbegriffe wie 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 (und verstehen).

2.1 Formale Grundlagen: Signaturen und Terme

Definition (Signatur)

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

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

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

  $s_1 \times \ldots \times s_n \rightarrow s$ heißt *Funktionalität* von f.

Definition (Sorten-geordnete Signatur)

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

Beispiel (Keller)

$$
\begin{array}{lcl}
S & = & \{Stack, NeStack, Elem\} \\
\leq & = & \{NeStack \leq Stack, \\
 & & Stack \leq Stack, \\
 & & NeStack \leq NeStack, \\
 & & Elem \leq Elem\} \\
F & = & \{emptyStack : \rightarrow Stack, \\
 & & push : Stack \times Elem \rightarrow NeStack, \\
 & & pop : NeStack \rightarrow Stack, \\
 & & top : NeStack \rightarrow Elem\} \\
\Sigma_{STACK} & = & (S, \leq, F)
\end{array}
$$

Abkürzung: $\leq = \{NeStack \leq Stack\}$

Definition (Terme)

Sei $\Sigma = (S, \leq, F)$ eine sorten-geordnete Signatur und sei X eine Menge von getypten Variablen (der Form $x : s$ mit $s \in S$).

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

Anmerkungen:

- Funktionssymbole der Form $\_\ f\ \_ : s_1 \times s_2 \rightarrow s$ weisen auf Mixfixdarstellung hin.
- Namen von Funktionssymbolen und Variablen können überladen sein.

Beispiel (Keller)

Seien $e : \mathit{Elem}, s : \mathit{Stack}, ns : \mathit{NeStack}$ getypte Variablen.
Terme sind z.B.

$\mathit{push}(s, e)$

$\mathit{pop}(\mathit{push}(s, e))$

$\mathit{top}(\mathit{push}(\mathit{emptyStack}, e))$

$\mathit{pop}(ns)$

Definition (Freie Variablen)

Sei $t \in T(\Sigma, X)$.
$FV(t) =_{def} \{x : s \in X \mid x \ \mathit{kommt\ in}\ t\ \mathit{vor}\}$

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

2.2 OCL-Signatur

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

Definition (OCL-Signatur)

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

1.

$$
\begin{aligned}
S_\Delta^{OCL} &= (Base^{OCL} \cup Class_\Delta \cup Coll_\Delta^{OCL}) \text{ wobei} \\
Base^{OCL} &= \{OclAny, OclVoid, \\
&\quad Real, Integer, Boolean, String, Null\} \text{ OCL-Grundtypen} \\
Class_\Delta &= \{C \mid C \text{ ist der Name einer Klasse aus } \Delta\} \\
Coll_\Delta^{OCL} &= \{Collection(T) \mid T \in Base^{OCL} \cup Class_\Delta\} \cup \\
&\quad \{Set(T) \mid T \in Base^{OCL} \cup Class_\Delta\} \cup \\
&\quad \{Sequence(T) \mid T \in Base^{OCL} \cup Class_\Delta\} \cup \\
&\quad \{Bag(T) \mid T \in Base^{OCL} \cup Class_\Delta\}
\end{aligned}
$$

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

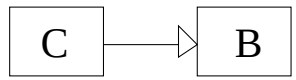
   (a) $\leq$ ist reflexiv und transitiv,

   (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.

$$F_\Delta^{OCL} = (Std_\Delta^{OCL} \cup Inst_\Delta^{OCL} \cup A_\Delta) \text{ wobei}$$

$$Std_\Delta^{OCL} = \text{OCL-Operationen auf OclAny}$$
$$(\text{z.B. } \_ = \_ : OclAny \times OclAny \to Boolean,$$
$$\_.oclIsUndefined() : OclAny \to Boolean,$$
$$\_.oclIsTypeOf(T) : OclAny \to Boolean \text{ mit } T \in S_\Delta^{OCL})$$

$$\cup \quad \text{OCL-Operationen auf Grundtypen}$$
$$(\text{z.B. } \_ + \_ : Real \times Real \to Real,$$
$$\_and\_ : Boolean \times Boolean \to Boolean)$$

$$\cup \quad \text{OCL-Operationen auf Kollektionstypen}$$
$$(\text{z.B. } \_ \to size() : Collection(OclAny) \to Integer,$$
$$\_ \to isempty() : Collection(OclAny) \to Boolean,$$
$$\_ = \_ : Collection(OclAny) \times Collection(OclAny) \to Boolean,$$
$$\_ \to includes(\_) : Collection(OclAny) \times OclAny \to Boolean,$$
$$\_ \to excludes(\_) : Collection(OclAny) \times OclAny \to Boolean,$$
$$Set\{\} :\to Set(T),$$
$$\_ \to including(\_) : Set(T) \times T \to Set(T)),$$
$$\_ \to intersection(\_) : Set(T) \times Set(T) \to Set(T))$$

$$\cup \quad \text{null-Konstante} \quad (null :\to Null)$$

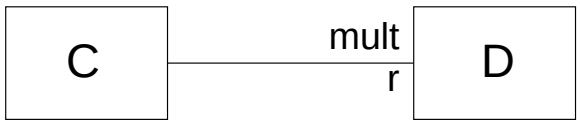
$$Inst_\Delta^{OCL} = \{C.allInstances() :\to Set(C) \mid C \in Class_\Delta\}$$

A_Δ enthält

- für jedes Attribut $a : T$ einer Klasse C aus Δ

  $\_.a : C \rightarrow T$

- für jeden Rollennamen r an einem (navigierbaren) Assoziationsende



  1. $\_.r : C \rightarrow D \quad falls\ mult <= 1$
  2. $\_.r : C \rightarrow Set(D) \quad falls\ mult > 1$
  3. $\_.r : C \rightarrow Sequence(D) \quad falls\ mult > 1\ mit\ Property\ \{ordered, unique\}$

  Der Rollenname r kann auch der implizite Rollenname sein.

Beispiel (Punkte und Formen)

$$
\begin{array}{lcl}
S_\Delta^{OCL} & = & (Base^{OCL} \cup Class_\Delta \cup Coll_\Delta^{OCL}) \text{ wobei} \\
Base^{OCL} & = & \{OclAny, OclVoid, Real, Integer, Boolean, String, Null\} \\
Class_\Delta & = & \{Point, CPoint, Shape, System\} \\
Coll_\Delta^{OCL} & = & \{Collection(OclAny), \ldots, Collection(System), \\
 & & \ Set(OclAny), \ldots, Set(System), \ldots\} \\
\leq & = & \{Real \leq OclAny, \ldots, System \leq OclAny, \\
 & & \ Integer \leq Real, CPoint \leq Point, \\
 & & \ Set(Point) \leq Collection(Point), \\
 & & \ Set(CPoint) \leq Set(Point), \ldots\} \\
F_\Delta^{OCL} & = & (Std_\Delta^{OCL} \cup Inst_\Delta^{OCL} \cup A_\Delta) \text{ wobei} \\
Std_\Delta^{OCL} & = & \{\_ = \_ : OclAny \times OclAny \to Boolean, \\
 & & \ \_ + \_ : Real \times Real \to Real, \\
 & & \ \_and\_ : Boolean \times Boolean \to Boolean, \ldots \\
 & & \ \_ = \_ : Collection(Point) \times Collection(Point) \to Boolean, \\
 & & \ \_ \to includes(\_) : Collection(OclAny) \times OclAny \to Boolean, \\
 & & \ \_ \to including(\_) : Set(Point) \times Point \to Set(Point), \\
 & & \ \_ \to including(\_) : Set(Shape) \times Shape \to Set(Shape), \ldots, \\
 & & \ null :\to Null\}
\end{array}
$$

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

Seien $self : Point$ und $p : Point$ getypte Variablen.

Terme sind z.B.

$self.xx$

$self.xx + 7 = self.yy$

$p.owner.points \rightarrow includes(p)$

$p.owner.points \rightarrow including(p)$

2.3 OCL-Ausdrücke

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

Definition (OCL-Ausdrücke)

Sei X eine Menge von getypten Variablen, so dass X
für jedes $C \in Class_\Delta$ eine spezielle Variable $self : C$ des Typs C und
für jedes $T \in S_\Delta^{OCL}$ eine spezielle Variable $result : T$ des Typs T enthält.

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

1. Falls $x : T \in X$, dann $x \in EXP_T^{OCL}$.

2. Falls $f \in F_\Delta^{OCL}$, $f : T_1 \times \ldots \times T_n \to T$ (mit $n \geq 0$)
   und $e_1 \in EXP_{U_1}^{OCL}, \ldots, e_n \in EXP_{U_n}^{OCL}$ mit $U_i \leq T_i$ (für $i = 1, \ldots, n$),
   dann $f(e_1, \ldots, e_n) \in EXP_T^{OCL}$ (ggf. in Mixfix-Notation).

   Beachte: OCL-Ausdrücke, die nur mit den Regeln 1. und 2. gebildet werden, sind genau die Terme über Σ_Δ^{OCL} (mit Variablen aus X).

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 $\mathit{if}\ e_B\ \mathit{then}\ e_1\ \mathit{else}\ e_2\ \mathit{endif} \in EXP_T^{OCL}$.

4. Falls $e \in EXP_{Coll(T)}^{OCL}$ (mit $Coll \in \{Set, Sequence, Bag\}$),
   $i : T \in X$, $acc : S \in X$, $e_{\text{start}} \in EXP_{S1}^{OCL}$, $S1 \leq S$ und $e_{\text{body}} \in EXP_S^{OCL}$,
   dann $e \to \mathit{iterate}(i : T;\ acc : S = e_{\text{start}} \mid e_{\text{body}}) \in EXP_S^{OCL}$.

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

5. Vordefinierte Iterator-Ausdrücke:

Falls $e \in EXP^{OCL}_{Coll(T)}$(mit $Coll \in \{Set, Sequence, Bag\}$)
und $x : T \in X$ und $e_B \in EXP^{OCL}_{Boolean}$,
dann

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

Auswertungsalgorithmus für $e \longrightarrow \mathit{iterate}(i : T;\ acc : S = e_{\mathbf{start}} \mid e_{\mathbf{body}})$

```
S acc = e_start;
T i;
forall i in e do
  acc := e_body;
return acc;
```

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 $Coll \in \{Set, Sequence, Bag\}$.

  $e \rightarrow forAll(x \mid e_B)$ steht für $e \rightarrow forAll(x : T \mid e_B)$
  und Analoges gilt für *exists*, *select*, *any*, *one*, *iterate*.
- $e \rightarrow forAll(x, y \mid e_B)$ steht für $e \rightarrow forAll(x \mid e \rightarrow forAll(y \mid e_B))$

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 Regeln 1. - 5. für OCL-Ausdrücke, wobei EXP durch $EEXP$ ersetzt wird, und durch die folgenden zusätzlichen Regeln:

6. Falls $C \in Class_\Delta$, dann $C.allInstances@pre() \in EEXP_{Set(C)}^{OCL}$.

7. Falls $C \in Class_\Delta$, $(\_.a : C \to T) \in A_\Delta$ 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

   $e.oclIsNew() =_{def}$
   $C.allInstances@pre() \to excludes(e)$ *and* $C.allInstances() \to includes(e)$
   and $e.oclIsTypeOf(C)$

Anmerkung

Die obigen Definitionen umfassen die wesentlichen Konstrukte (für Ausdrücke) der Sprache OCL. Die vollständige Syntax findet sich in der OCL Spezifikation.

Definition (Freie Variablen)

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

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

2.4 Semantische Bereiche für OCL-Typen

$$
\begin{array}{lcll}
[\![\mathit{OclVoid}]\!] & = & \{\bot\} & \textit{undefiniert} \\
[\![\mathit{Integer}]\!] & = & \mathbb{Z} \cup \{\bot\} & \textit{ganze Zahlen (mit } \bot\textit{)} \\
[\![\mathit{Real}]\!] & = & \mathbb{R} \cup \{\bot\} & \textit{reelle Zahlen} \\
[\![\mathit{Boolean}]\!] & = & \{\mathit{true},\ \mathit{false},\ \bot\} & \textit{Wahrheitswerte} \\
[\![\mathit{String}]\!] & = & \mathcal{A}^* \cup \{\bot\} & \textit{endliche Strings über einem Alphabet } \mathcal{A} \\
[\![\mathit{Null}]\!] & = & \{\mathit{null},\ \bot\} &
\end{array}
$$

Für jeden Klassennamen $C \in \mathit{Class}_\Delta$ sei OId_C eine abzählbar unendliche Menge von Objektidentitäten des Typs C, so dass $\mathit{OId}_C \cap \mathit{OId}_D = \emptyset$ falls $C \neq D$.
Sei $\mathit{OID}_C = \bigcup \{\mathit{OId}_D \mid D \in \mathit{Class}_\Delta,\ D \leq C\}$.

$$
\begin{array}{lcll}
[\![C]\!] & = & \mathit{OID}_C \cup \{\mathit{null}, \bot\} & \\
[\![\mathit{OclAny}]\!] & = & \bigcup \{[\![T]\!] \mid T \in \mathit{Base}^{OCL} \cup \mathit{Class}_\Delta,\ T \neq \mathit{OclAny}\} & \\
[\![\mathit{Set}(T)]\!] & = & \mathcal{F}([\![T]\!]) \cup \{\bot\} & \textit{endliche Mengen über } [\![T]\!] \\
[\![\mathit{Sequence}(T)]\!] & = & [\![T]\!]^* \cup \{\bot\} & \textit{endliche Folgen über } [\![T]\!] \\
[\![\mathit{Bag}(T)]\!] & = & \mathcal{B}([\![T]\!]) \cup \{\bot\} & \textit{endl. Multimengen über } [\![T]\!] \\
[\![\mathit{Collection}(T)]\!] & = & \mathcal{F}([\![T]\!]) \cup [\![T]\!]^* \cup \mathcal{B}([\![T]\!]) \cup \{\bot\} &
\end{array}
$$

Lemma (Semantische Bereiche sind verträglich mit $\leq$)
Für alle $T, T' \in S_\Delta^{OCL}$ gilt:
Falls $T \leq T'$ dann $[\![T]\!] \subseteq [\![T']\!]$.

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

2.5 Interpretation von Standard OCL-Funktionen

Die Interpretation eines Funktionssymbols $(f : T_1 \times \ldots \times T_n \to T) \in Std_\Delta^{OCL}$ ist eine Funktion $[\![f]\!] : [\![T_1]\!] \times \ldots \times [\![T_n]\!] \to [\![T]\!]$.

Beispiel (Interpretation von $\_ + \_$)
$\_ + \_ : Real \times Real \to Real$ wird interpretiert durch
$[\![\_ + \_]\!] : [\![Real]\!] \times [\![Real]\!] \to [\![Real]\!]$,

$$[\![\_ + \_]\!](x, y) = \begin{cases} x + y & \text{falls } x \neq \bot \text{ und } y \neq \bot \\ \bot & \text{falls } x = \bot \text{ oder } y = \bot \end{cases}$$

Definition (Striktheit)
Eine Funktion $[\![f]\!] : [\![T_1]\!] \times \ldots \times [\![T_n]\!] \to [\![T]\!]$ heißt strikt, falls aus $(x_1 = \bot \text{ oder } \ldots \text{ oder } x_n = \bot)$ folgt $[\![f]\!](x_1, \ldots, x_n) = \bot$.

1. Operationen auf OclAny

$$[\![\_ = \_]\!] : [\![OclAny]\!] \times [\![OclAny]\!] \to [\![Boolean]\!]$$

$$[\![\_ = \_]\!](x, y) = \begin{cases} true & \text{falls } x = y \text{ und } x \neq \bot, y \neq \bot \\ false & \text{falls } x \neq y \text{ und } x \neq \bot, y \neq \bot \\ \bot & \text{falls } x = \bot \text{ oder } y = \bot \end{cases}$$

$$[\![\_.oclIsUndefined()]\!] : [\![OclAny]\!] \to [\![Boolean]\!]$$

$$[\![\_.oclIsUndefined()]\!](x) = \begin{cases} true & \text{falls } x = \bot \\ false & \text{sonst} \end{cases}$$

$$[\![\_.oclIsTypeOf(T)]\!] : [\![OclAny]\!] \to [\![Boolean]\!]$$

$$[\![\_.oclIsTypeOf(T)]\!](x) = \begin{cases} true & \text{falls } x \in [\![T]\!] \setminus \bigcup \{ [\![T']\!] \mid T' < T \} \\ false & \text{sonst} \end{cases}$$

2. Arithmetische Operationen

werden durch die strikten Fortsetzungen ihrer Standardbedeutungen interpretiert.

3. Boolsche Operationen

$[\![not\_]\!] : [\![Boolean]\!] \to [\![Boolean]\!]$

$$[\![not\_]\!](x) = \begin{cases} true & \text{falls } x = false \\ false & \text{falls } x = true \\ \bot & \text{sonst} \end{cases}$$

$[\![\_and\_]\!] : [\![Boolean]\!] \times [\![Boolean]\!] \to [\![Boolean]\!]$

$$[\![\_and\_]\!](x, y) = \begin{cases} false & \text{falls } x = false \text{ oder } y = false \\ true & \text{falls } x = true \text{ und } y = true \\ \bot & \text{sonst} \end{cases}$$

$[\![\_or\_]\!] : [\![Boolean]\!] \times [\![Boolean]\!] \to [\![Boolean]\!]$

$$[\![\_or\_]\!](x, y) = \begin{cases} true & \text{falls } x = true \text{ oder } y = true \\ false & \text{falls } x = false \text{ und } y = false \\ \bot & \text{sonst} \end{cases}$$

$[\![\_implies\_]\!] : [\![Boolean]\!] \times [\![Boolean]\!] \to [\![Boolean]\!]$

$$[\![\_implies\_]\!](x, y) = \begin{cases} true & \text{falls } x = false \text{ oder } (x = true \text{ und } y = true) \\ false & \text{falls } x = true \text{ und } y = false \\ \bot & \text{sonst} \end{cases}$$

Anmerkung: $[\![\_implies\_]\!](x, y) = [\![\_or\_]\!]([\![not]\!](x), [\![\_and\_]\!](x, y))$

4. Operationen auf Kollektionstypen

werden durch strikte Funktionen interpretiert, z.B.

$$[\![\_ \rightarrow \mathit{includes}(\_)]\!] : [\![\mathit{Collection}(\mathit{OclAny})]\!] \times [\![\mathit{OclAny}]\!] \rightarrow [\![\mathit{Boolean}]\!]$$

$$[\![\_ \rightarrow \mathit{includes}(\_)]\!](s, x) = \begin{cases} \mathit{true} & \text{falls } x \in s \text{ und } x \neq \bot, s \neq \bot \\ \mathit{false} & \text{falls } x \notin s \text{ und } x \neq \bot, s \neq \bot \\ \bot & \text{falls } x = \bot \text{ oder } s = \bot \end{cases}$$

$$[\![\_ \rightarrow \mathit{including}(\_)]\!] : [\![\mathit{Set}(T)]\!] \times [\![T]\!] \rightarrow [\![\mathit{Set}(T)]\!]$$

$$[\![\_ \rightarrow \mathit{including}(\_)]\!](s, x) = \begin{cases} s \cup \{x\} & \text{falls } s \neq \bot \text{ und } x \neq \bot \\ \bot & \text{falls } s = \bot \text{ oder } x = \bot \end{cases}$$

$$[\![\_ \rightarrow \mathit{including}(\_)]\!] : [\![\mathit{Sequence}(T)]\!] \times [\![T]\!] \rightarrow [\![\mathit{Sequence}(T)]\!]$$

$$[\![\_ \rightarrow \mathit{including}(\_)]\!](s, x) = \begin{cases} s \circ \langle x \rangle & \text{falls } s \neq \bot, x \neq \bot \\ \bot & \text{falls } s = \bot \text{ oder } x = \bot \end{cases}$$

5. Null-Konstante

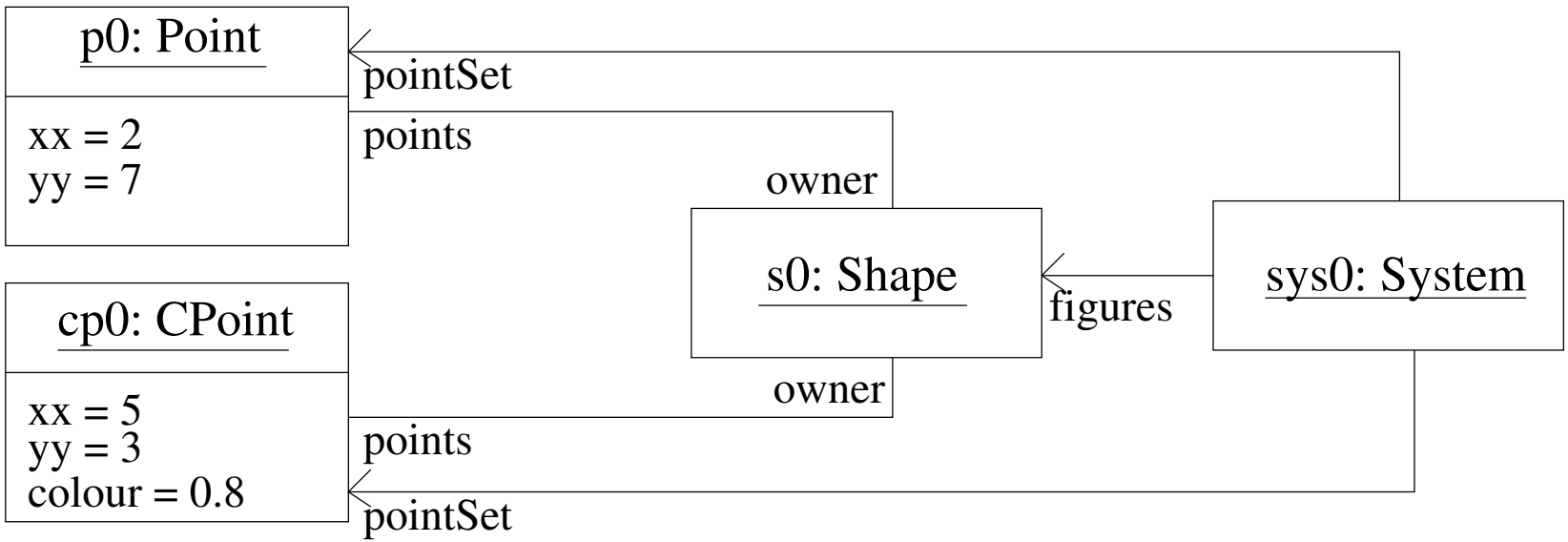
$[\![\mathit{null}]\!] = \mathit{null}$

2.6 Formalisierung von Systemzuständen

Im Folgenden sei Δ ein Klassendiagramm.

Graphische Darstellung von Systemzuständen
Ein Zustand des (von Δ beschriebenen) Systems zu einem bestimmten Zeitpunkt kann durch ein Objektdiagramm 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, die 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}

$$p0.xx \longmapsto 2 \qquad cp0.xx \longmapsto 5 \qquad s0.points \longmapsto \{p0, cp0\}$$
$$p0.yy \longmapsto 7 \qquad cp0.yy \longmapsto 3 \qquad sys0.figures \longmapsto \{s0\}$$
$$p0.owner \longmapsto s0 \qquad cp0.owner \longmapsto s0 \qquad sys0.pointSet \longmapsto \{p0, cp0\}$$
$$cp0.colour \longmapsto 0.8$$

Definition (Links- und Rechtswerte)

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

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

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

$$RVal_\Delta = \bigcup_{T \in S_\Delta^{OCL}} [\![T]\!]$$

Anmerkung:

$RVal_\Delta = [\![OclAny]\!] \cup [\![Collection(OclAny)]\!]$

Definition (Systemzustand)

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

1. $\sigma_{Instances} : Class_\Delta \to \bigcup_{C \in Class_\Delta} \mathcal{F}(OID_C)$, so dass gilt:

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

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

   Notation:
   Wir schreiben C_σ für $\sigma_{Instances}(C)$.

2. $\sigma_{Val} :\ LVal_\Delta \to RVal_\Delta$,
   so dass für jedes $o \in OID_C$ und $(\_.a : C \to T) \in A_\Delta$ gilt:

   $$\sigma_{Val}(\, o.a \,) \in [\![T]\!]$$

   Notation:
   Wir schreiben $o.a_\sigma$ für $\sigma_{Val}(o.a)$

Definition

Die Menge aller Systemzustände bzgl. eines Klassendiagramms Δ ist gegeben durch

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

Notation:

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

$$\sigma[o.a \longmapsto v]$$

für den Zustand $\sigma' = (\sigma_{Instances},\ \sigma'_{Val})$ mit

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

d.h. σ wird mit dem (neuen) Wert v für $o.a$ aktualisiert.

Notation:

Für $\sigma \in State_\Delta$, $C \in Class_\Delta$, $o \in Old_C$, $o \notin C_\sigma$ schreiben wir

$$\sigma\,[o = \mathrm{new}_C]$$

für den Zustand $\sigma' = (\sigma'_{Instances}, \sigma'_{Val})$ mit

$$\sigma'_{Instances}(B) = \begin{cases} \sigma_{Instances}(B) \cup \{o\} & \text{falls } B \geq C \\ \sigma_{Instances}(B) & \text{sonst} \end{cases}$$

$$\begin{array}{lcll} \sigma'_{Val}(o.a) & = & default_T & \text{für alle } (a : B \to T) \in A_\Delta \text{ mit } B \geq C \\ \sigma'_{Val}(o'.a) & = & \sigma_{Val}(o'.a) & \text{für alle } o'.a \in LVal_\Delta \text{ mit } o' \neq o \end{array}$$

wobei

$$\begin{array}{l} default_{Integer} = default_{Real} = 0, \\ default_{Boolean} = false, \\ default_C = null \text{ für } C \in Class_\Delta, \\ default_{Set(T)} = \emptyset, \ \ldots \end{array}$$

Notationen:

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

1. $(State_\Delta \text{ X } (\llbracket T_1 \rrbracket \times \ldots \times \llbracket T_n \rrbracket) =_{def}$

$$\{(\sigma, v_1, \ldots, v_n) \in State_\Delta \times \llbracket T_1 \rrbracket \times \ldots \times \llbracket T_n \rrbracket \mid$$

$$v_i \neq \perp \text{ für } i = 1, \ldots, n \text{ und}$$
$$\text{falls } T_i \in Class_\Delta \text{ dann } v_i \in (T_i)_\sigma \cup \{null\} \text{ und}$$
$$\text{falls } T_i = Coll(D) \text{ mit } D \in Class_\Delta \text{ dann } v_i \subseteq D_\sigma \cup \{null\}\}$$

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

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

$$Valid\,(\,\sigma, \llbracket T_1 \rrbracket \times \ldots \times \llbracket T_n \rrbracket) =_{def}$$

$$\{(v_1, \ldots, v_n) \in \llbracket T_1 \rrbracket \times \ldots \times \llbracket T_n \rrbracket \mid$$

$$(\sigma, v_1, \ldots, v_n) \in State_\Delta \text{ X } (\llbracket T_1 \rrbracket \times \ldots \times \llbracket T_n \rrbracket)\}$$

2.7 Denotationelle Semantik von OCL-Ausdrücken

Grundidee
Die Bedeutung (Denotation) eines OCL-Ausdrucks e vom Typ T ist ein Wert des semantischen Bereichs $[\![T]\!]$.

Beispiel (Semantik von OCL-Ausdrücken)

- $3 + 2$ bedeutet $5 \in [\![\mathit{Integer}]\!] \quad (= \mathbb{Z} \cup \{\bot\})$
- $3 = 2$ bedeutet $\mathit{false} \in [\![\mathit{Boolean}]\!]$
- Sei $mx : \mathit{Real}$ eine Variable des Typs Real.
  $2 + mx$ kann nur bezüglich einer Variablenbelegung β interpretiert werden.

  z.B. für $\beta(mx) = 4.5$ hat $2 + mx$ die Bedeutung 6.5.
- Sei $p : \mathit{Point}$ eine Variable des Typs Point.
  $p.xx$ kann nur bezüglich einer Variablenbelegung β und bezüglich eines Zustands $\sigma \in \mathit{State}_\Delta$ interpretiert werden.

  z.B. für $\beta(p) = p0 \in [\![\mathit{Point}]\!]$ und für σ mit $p0 \in \mathit{Point}_\sigma$ 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 $\beta(self) = p0,\ \beta(mx) = 4.5,$
  $p0.xx_{\sigma^-} = 2,\ p0.xx_\sigma = 5.3$

  hat $self.xx = self.xx@pre + mx$ 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: $[\![e]\!]_{\beta,\sigma^-,\sigma}$

Definition (Variablenbelegung)

Sei X eine Menge von getypten Variablen.
Eine Belegung β ist eine Funktion

$$\beta : X \to \bigcup_{T \in S_\Delta^{OCL}} [\![T]\!]$$

so dass $\beta(x : T) \in [\![T]\!]$.

Notationen:

- Die Menge aller Belegungen wird mit Env_Δ bezeichnet.
- Sei $\beta \in Env_\Delta$ und $x : T \in X$. Wir schreiben häufig $\beta(x)$ statt $\beta(x : T)$.
- Sei $\beta \in Env_\Delta$, $y : T \in X$ und $v \in [\![T]\!]$.
  Wir schreiben $\beta[y \mapsto v]$ für die Belegung β' mit

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

Definition (Semantik von OCL-Ausdrücken)

Sei $EEXP^{OCL} = (EEXP_T^{OCL})_{T \in S_\Delta^{OCL}}$ die 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 $(\llbracket \_ \rrbracket_{\_,\_,\_,T})_{T \in S_\Delta^{OCL}}$ wobei für jedes $T \in S_\Delta^{OCL}$

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

Die Funktionen $\llbracket \_ \rrbracket_{\_,\_,\_,T}$ werden gemäß der Struktur von OCL-Ausdrücken induktiv definiert. Im Folgenden wird diese Definition im Detail dargestellt, wobei zur notationellen Vereinfachung der Index "T" weggelassen wird.

Induktive Definition von $[\![e]\!]_{\beta,\sigma^-,\sigma}$

1. $[\![x]\!]_{\beta,\sigma^-,\sigma} = \beta(x)$ für $x : T \in X$
2. Sei $f \in F_\Delta^{OCL} = Std_\Delta^{OCL} \cup Inst_\Delta^{OCL} \cup A_\Delta$.
   (a) $f \in Std_\Delta^{OCL}$:

$$[\![f(e_1,\ldots,e_n)]\!]_{\beta,\sigma^-,\sigma} = [\![f]\!]([\![e_1]\!]_{\beta,\sigma^-,\sigma},\ldots,\ [\![e_n]\!]_{\beta,\sigma^-,\sigma})$$

z.B. sei $\beta(self) = p0,\ \beta(mx) = 4.5,$
$p0.xx_{\sigma^-} = 2,\ p0.xx_\sigma = 5.3.$

$$[\![self.xx = self.xx@pre + mx]\!]_{\beta,\sigma^-,\sigma} =$$

$$[\![\_ = \_]\!]([\![self.xx]\!]_{\beta,\sigma^-,\sigma},\ [\![self.xx@pre + mx]\!]_{\beta,\sigma^-,\sigma}) =$$

$$[\![\_ = \_]\!]([\![self.xx]\!]_{\beta,\sigma^-,\sigma},\ [\![\_ + \_]\!]([\![self.xx@pre]\!]_{\beta,\sigma^-,\sigma},\ [\![mx]\!]_{\beta,\sigma^-,\sigma})) =$$

$$[\![\_ = \_]\!]([\![self]\!]_{\beta,\sigma^-,\sigma}.xx_\sigma,\ [\![\_ + \_]\!]([\![self]\!]_{\beta,\sigma^-,\sigma}.xx_{\sigma^-},\ [\![mx]\!]_{\beta,\sigma^-,\sigma})) =$$

$$[\![\_ = \_]\!](\beta(self).xx_\sigma,\ [\![\_ + \_]\!](\beta(self).xx_{\sigma^-},\ \beta(mx))) =$$

$$[\![\_ = \_]\!](p0.xx_\sigma,\ [\![\_ + \_]\!](p0.xx_{\sigma^-},\ 4.5)) =$$

$$[\![\_ = \_]\!](5.3,\ [\![\_ + \_]\!](2,\ 4.5)) =$$

$$[\![\_ = \_]\!](5.3,\ 6.5) =$$

$$false$$

(b) $[\![C.allInstances()]\!]_{\beta,\sigma^-,\sigma} = C_\sigma$

z.B. $[\![Point.allInstances()]\!]_{\beta,\sigma^-,\sigma} = \{p0, cp0\}$ mit σ wie oben

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

$$[\![e.a]\!]_{\beta,\sigma^-,\sigma} = \begin{cases} \bot & \text{falls } [\![e]\!]_{\beta,\sigma^-,\sigma} = \bot \text{ oder } [\![e]\!]_{\beta,\sigma^-,\sigma} = null \\ [\![e]\!]_{\beta,\sigma^-,\sigma}.a_\sigma & \text{sonst} \end{cases}$$

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

$[\![p.owner.points]\!]_{\beta,\sigma^-,\sigma} =$

$[\![p.owner]\!]_{\beta,\sigma^-,\sigma}.points_\sigma =$

$[\![p]\!]_{\beta,\sigma^-,\sigma}.owner_\sigma.points_\sigma =$

$\beta(p).owner_\sigma.points_\sigma =$

$p0.owner_\sigma.points_\sigma =$

$s0.points_\sigma =$

$\{p0, cp0\}$

3.

$$[\![\text{if } e_B \text{ then } e_1 \text{ else } e_2 \text{ endif}]\!]_{\beta,\sigma^-,\sigma} = \begin{cases} [\![e_1]\!]_{\beta,\sigma^-,\sigma} & \text{falls } [\![e_B]\!]_{\beta,\sigma^-,\sigma} = \mathit{true} \\ [\![e_2]\!]_{\beta,\sigma^-,\sigma} & \text{falls } [\![e_B]\!]_{\beta,\sigma^-,\sigma} = \mathit{false} \\ \bot & \text{falls } [\![e_B]\!]_{\beta,\sigma^-,\sigma} = \bot \end{cases}$$

4. Sei $e \in EEXP^{OCL}_{Set(T)}$, $e_s \in EEXP^{OCL}_{S1}$ mit $S1 \leq S$ und $e_b \in EEXP^{OCL}_{S}$.

$$[\![e \rightarrow \mathit{iterate}(i : T; \mathit{acc} : S = e_s \mid e_b)]\!]_{\beta,\sigma^-,\sigma} = \mathit{it}_{e_b,\beta,\sigma^-,\sigma}([\![e]\!]_{\beta,\sigma^-,\sigma}, [\![e_s]\!]_{\beta,\sigma^-,\sigma}),$$

wobei $\mathit{it}_{e_b,\beta,\sigma^-,\sigma} : [\![\mathit{Set}(T)]\!] \times [\![S]\!] \rightarrow [\![S]\!]$ rekursiv definiert ist durch

$$\mathit{it}_{e_b,\beta,\sigma^-,\sigma}(m, w) = \begin{cases} w & \text{falls } m = \emptyset \\ \mathit{it}_{e_b,\beta,\sigma^-,\sigma}(m \setminus \{v\}, [\![e_b]\!]_{\beta[i \mapsto v,\ \mathit{acc} \mapsto w],\sigma^-,\sigma}) & \text{falls } m \neq \bot, v \in m \\ \bot & \text{sonst} \end{cases}$$

In ähnlicher Weise definiert man die Semantik von Iteratorausdrücken, 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 ausgeführt wird.

5. $[\![e \rightarrow forAll(x : T \mid e_B)]\!]_{\beta,\sigma^-,\sigma} =$
$[\![e \rightarrow iterate(x : T; res = true \mid res\ and\ e_B)]\!]_{\beta,\sigma^-,\sigma}$

$[\![e \rightarrow exists(x : T \mid e_B)]\!]_{\beta,\sigma^-,\sigma} =$
$[\![e \rightarrow iterate(x : T; res = false \mid res\ or\ e_B)]\!]_{\beta,\sigma^-,\sigma}$

$[\![e \rightarrow select(x : T \mid e_B)]\!]_{\beta,\sigma^-,\sigma} =$
$[\![e \rightarrow iterate(x : T; res : Coll(T) = Coll\{\} \mid$
$\quad if\ e_B\ then\ res \rightarrow including(x)\ else\ res\ endif)]\!]_{\beta,\sigma^-,\sigma}$

$[\![e \rightarrow any(x : T \mid e_B)]\!]_{\beta,\sigma^-,\sigma} =$
$[\![e \rightarrow select(x : T \mid e_B) \rightarrow asSequence() \rightarrow first()]\!]_{\beta,\sigma^-,\sigma}$

$[\![e \rightarrow one(x : T \mid e_B)]\!]_{\beta,\sigma^-,\sigma} =$
$[\![e \rightarrow select(x : T \mid e_B) \rightarrow size() = 1]\!]_{\beta,\sigma^-,\sigma}$

6. $[\![C.allInstances@pre()]\!]_{\beta,\sigma^-,\sigma} = C_{\sigma^-}$

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

$$[\![e.a@pre]\!]_{\beta,\sigma^-,\sigma} = \begin{cases} \bot & \text{falls } [\![e]\!]_{\beta,\sigma^-,\sigma} = \bot \text{ oder } [\![e]\!]_{\beta,\sigma^-,\sigma} = null \\ [\![e]\!]_{\beta,\sigma^-,\sigma}.a_{\sigma^-} & \text{sonst} \end{cases}$$

z.B. sei $\beta(self) = p0, \ p0.xx_{\sigma^-} = 2$

$[\![self.xx@pre]\!]_{\beta,\sigma^-,\sigma} =$

$[\![self]\!]_{\beta,\sigma^-,\sigma}.xx_{\sigma^-} =$

$\beta(self).xx_{\sigma^-} =$

$p0.xx_{\sigma^-} = 2$

8. Sei $C \in Class_\Delta, \ e \in EEXP_C^{OCL}$.

$[\![e.oclIsNew()]\!]_{\beta,\sigma^-,\sigma} =$

$[\![C.allInstances@pre() \to excludes(e) \ and \ C.allInstances() \to includes(e)$
$\quad and \ e.oclIsTypeOf(C)]\!]_{\beta,\sigma^-,\sigma}$

Lemma:

1. $[\![e \to \mathit{forAll}(x : T \mid e_B)]\!]_{\beta,\sigma^-,\sigma} =$

$$\begin{cases} \mathit{true} & \text{falls } [\![e]\!]_{\beta,\sigma^-,\sigma} \neq \bot \text{ und für alle } v \in [\![e]\!]_{\beta,\sigma^-,\sigma} \\ & [\![e_B]\!]_{\beta[x \mapsto v],\sigma^-,\sigma} = \mathit{true} \\ \mathit{false} & \text{falls } [\![e]\!]_{\beta,\sigma^-,\sigma} \neq \bot \text{ und es existiert } v \in [\![e]\!]_{\beta,\sigma^-,\sigma} \\ & \text{so dass } [\![e_B]\!]_{\beta[x \mapsto v],\sigma^-,\sigma} = \mathit{false} \\ \bot & \text{sonst} \end{cases}$$

2. $[\![e \to \mathit{exists}(x : T \mid e_B)]\!]_{\beta,\sigma^-,\sigma} =$

$$\begin{cases} \mathit{true} & \text{falls } [\![e]\!]_{\beta,\sigma^-,\sigma} \neq \bot \text{ und es existiert } v \in [\![e]\!]_{\beta,\sigma^-,\sigma} \\ & \text{so dass } [\![e_B]\!]_{\beta[x \mapsto v],\sigma^-,\sigma} = \mathit{true} \\ \mathit{false} & \text{falls } [\![e]\!]_{\beta,\sigma^-,\sigma} \neq \bot \text{ und für alle } v \in [\![e]\!]_{\beta,\sigma^-,\sigma} \\ & [\![e_B]\!]_{\beta[x \mapsto v],\sigma^-,\sigma} = \mathit{false} \\ \bot & \text{sonst} \end{cases}$$

3.

$$[\![e.\mathit{oclIsNew}()]\!]_{\beta,\sigma^-,\sigma} = \begin{cases} \bot & \text{falls } [\![e]\!]_{\beta,\sigma^-,\sigma} = \bot \\ \mathit{true} & \text{falls } [\![e]\!]_{\beta,\sigma^-,\sigma} \in C_\sigma, [\![e]\!]_{\beta,\sigma^-,\sigma} \notin C_{\sigma^-} \text{ und} \\ & [\![e]\!]_{\beta,\sigma^-,\sigma} \notin D_\sigma \text{ für } D < C \\ \mathit{false} & \text{sonst} \end{cases}$$

Beachte:

$[\![e \to \mathit{exists}(x : T \mid e_B)]\!]_{\beta,\sigma^-,\sigma} = [\![\mathit{not}(e \to \mathit{forAll}(x : T \mid \mathit{not}(e_B)))]\!]_{\beta,\sigma^-,\sigma}$

Beispiel

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

$[\![self.points \rightarrow exists(q : Point \mid q.xx > 4)]\!]_{\beta,\sigma^-,\sigma} = true$

weil für $cp0 \in [\![self.points]\!]_{\beta,\sigma^-,\sigma} = \{p0, cp0\}$,

$[\![q.xx > 4]\!]_{\beta[q\mapsto cp0],\sigma^-,\sigma} = ([\![q.xx]\!]_{\beta[q\mapsto cp0],\sigma^-,\sigma} > [\![4]\!]_{\ldots}) =$

$([\![q]\!]_{\beta[q\mapsto cp0],\sigma^-,\sigma}.xx_\sigma > [\![4]\!]_{\ldots}) =$

$(\beta[q \mapsto cp0](q).xx_\sigma > 4) = (cp0.xx_\sigma > 4) = (5 > 4) = true$

Beispiel

Sei $result : Point,\ \beta(result) = p1,\ p1 \notin Point_{\sigma^-},\ p1 \in Point_\sigma \setminus CPoint_\sigma$.

Dann gilt: $[\![result.oclIsNew()]\!]_{\beta,\sigma^-,\sigma} = true$

Lemma:

Sei e ein (erweiterter) OCL-Ausdruck mit $FV(e) = \{x_1 : T_1, \ldots, x_n : T_n\}$.
Seien $\beta, \beta' \in Env_\Delta$ zwei Belegungen, so dass $\beta(x_i) = \beta'(x_i)$ für $i = 1, \ldots, n$.
Dann gilt für alle $\sigma^-, \sigma \in State_\Delta$:

$$[\![e]\!]_{\beta,\sigma^-,\sigma} = [\![e]\!]_{\beta',\sigma^-,\sigma}$$

2.8 Zusammenfassung

Sei Δ ein Klassendiagramm.

- Die OCL-Signatur $\Sigma_\Delta^{OCL} = (S_\Delta^{OCL}, \leq, F_\Delta^{OCL})$ stellt die Typen und Operationen zur Verfügung, die in (erweiterten) OCL-Ausdrücken verwendet werden können.
- (Erweiterte) OCL-Ausdrücke besitzen - zusätzlich zu Termen - Konstrukte für bedingte Ausdrücke (*if-then-else-endif*), Iteration (*iterate, forAll, exists, ...*), zur Referenzierung von Vorzuständen (*@pre*) und für die Erzeugung neuer Objekte (*oclIsNew*).
- Jedem Typ $T \in S_\Delta^{OCL}$ ist ein semantischer Bereich $[\![T]\!]$ zugeordnet, so dass gilt: $T' \leq T$ impliziert $[\![T']\!] \subseteq [\![T]\!]$.
- $State_\Delta = \{\sigma \mid \sigma = (\sigma_{Instances},\ \sigma_{Val})\}$ ist die Menge der Systemzustände bezüglich Δ.
- Die Semantik $[\![e]\!]_{\beta,\sigma^-,\sigma}$ eines (erweiterten) OCL-Ausdrucks ist definiert bezüglich
  - einer Belegung β (für die in e frei vorkommenden Variablen),
  - zweier Zustände $\sigma^-,\ \sigma \in State_\Delta$.