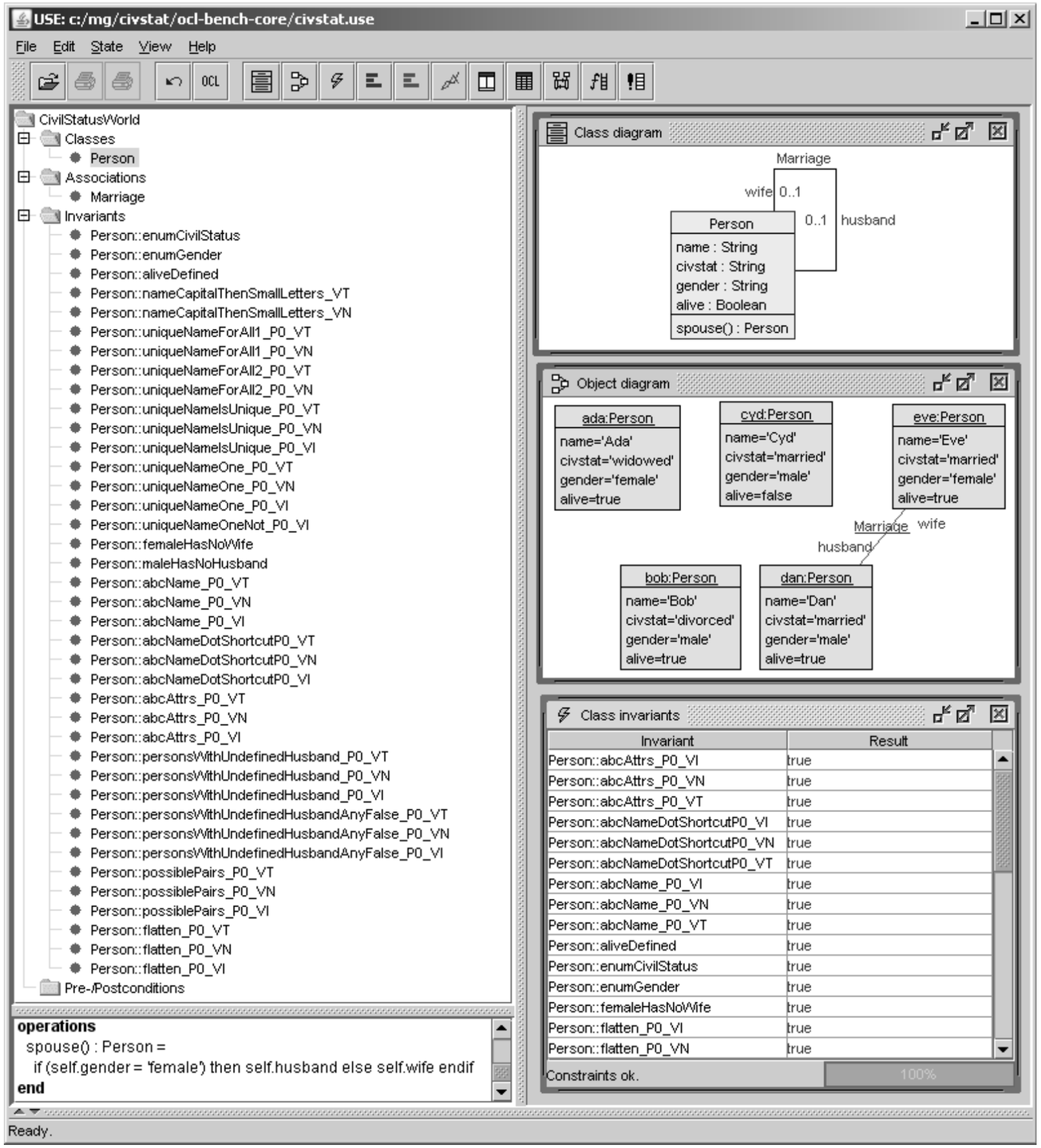
OCL Benchmark - Core (B1)

Benchmark **Core** des OCL Benchmark

Modell



Ergebnisse der Werkzeuge

Invarianten

	Name	USE	Eclipse MDT OCL	RoclET	Octopus	DresdenOCL-Parser	OCLE	ATL[1]
1	**context Person inv enumCivilStatus: self.civstat='single' or self.civstat='married' or self.civstat='divorced' or self.civstat='widowed'**	true	true	true	true		**Boolean= true**	true
2	**context Person inv enumGender: self.gender='female' or self.gender='male'**	true	true	true	true		**Boolean= true**	true
3	**context Person inv aliveDefined: self.alive=true or self.alive=false**	true	true	true	true		**Boolean= true**	true
4	**context Person inv nameCapitalThenSmallLetters_VT: let small:Set(String)= Set{'a','b','c','d','e','f','g','h','i','j','k','l','m', 'n','o','p','q','r','s','t','u','v','w','x','y','z'} in let capital:Set(String)= Set{'A','B','C','D','E','F','G','H','I','J','K','L','M', 'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'} in capital->includes(self.name.substring(1,1)) and Set{2..self.name.size()}->forAll(i:Integer| small->includes(self.name.substring(i,i))) and self.name.size()>=1**	true	true	true	true		**Boolean= true**[2]	Syntaxfehler. [3]
5	**context Person inv nameCapitalThenSmallLetters_VN: let small= Set{'a','b','c','d','e','f','g','h','i','j','k','l','m', 'n','o','p','q','r','s','t','u','v','w','x','y','z'} in let capital= Set{'A','B','C','D','E','F','G','H','I','J','K','L','M', 'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'} in capital->includes(self.name.substring(1,1)) and Set{2..self.name.size()}->forAll(i| small->includes(self.name.substring(i,i))) and self.name.size()>=1**	true	N/A[4]	true	N/A Typisierung notwendig	Typisierung notwendig	**Boolean= true** [2]	Syntaxfehler. [3] [5]
6	**context Person inv uniqueNameForAll1_P0_VT: Person.allInstances->forAll(self2:Person| self<>self2 implies self.name<>self2.name)**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	**Boolean= true**	Syntaxfehler. [8]
7	**context Person inv uniqueNameForAll1_P0_VN: Person.allInstances->forAll(self2| self<>self2 implies self.name<>self2.name)**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	**Boolean= true**	N/A[9]
8	**context Person inv uniqueNameForAll1_P1_VT: Person.allInstances()->forAll(self2:Person| self<>self2 implies self.name<>self2.name)**	N/A	true	true	true		Boolean= true	Syntaxfehler. [8]
9	**context Person inv uniqueNameForAll1_P1_VN: Person.allInstances()->forAll(self2| self<>self2 implies self.name<>self2.name)**	N/A	true	true	true		Boolean= true	true

10	**context Person inv uniqueNameForAll2_P0_VT: Person.allInstances->forAll(p1,p2:Person| p1<>p2 implies p1.name<>p2.name)**	true	N/A[6]	N/A	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler. [8] [9] [10]
11	**context Person inv uniqueNameForAll2_P0_VN: Person.allInstances->forAll(p1,p2| p1<>p2 implies p1.name<>p2.name)**	true	N/A[6]	N/A	N/A[6]	Syntaxfehler[7]	Boolean= true	N/A[9]
12	**context Person inv uniqueNameForAll2_P1_VT: Person.allInstances()->forAll(p1,p2:Person| p1<>p2 implies p1.name<>p2.name)**	N/A	true	N/A	true		Boolean= true	Syntaxfehler.[8][10]
13	**context Person inv uniqueNameForAll2_P1_VN: Person.allInstances()->forAll(p1,p2| p1<>p2 implies p1.name<>p2.name)**	N/A	true	N/A	true		Boolean= true	N/A[10]
14	**context Person inv uniqueNameIsUnique_P0_VT: Person.allInstances->isUnique(p:Person|p.name)**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler. [8] [9]
15	**context Person inv uniqueNameIsUnique_P0_VN: Person.allInstances->isUnique(p|p.name)**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	N/A[9]
16	**context Person inv uniqueNameIsUnique_P0_VI: Person.allInstances->isUnique(name)**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler.[9] [11]
17	**context Person inv uniqueNameIsUnique_P1_VT: Person.allInstances()->isUnique(p:Person|p.name)**	N/A	true	true	true		Boolean= true	Syntaxfehler. [8]
18	**context Person inv uniqueNameIsUnique_P1_VN: Person.allInstances()->isUnique(p|p.name)**	N/A	true	true	true		Boolean= true	true
19	**context Person inv uniqueNameIsUnique_P1_VI: Person.allInstances()->isUnique(name)**	N/A	true	true	true		Boolean= true	Syntaxfehler.[11]
20	**context Person inv uniqueNameOne_P0_VT: Person.allInstances->one(p:Person|self.name=p.name)**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler. [8][9]
21	**context Person inv uniqueNameOne_P0_VN: Person.allInstances->one(p|self.name=p.name)**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	N/A[9]
22	**context Person inv uniqueNameOne_P0_VI: Person.allInstances->one(self.name=name)**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler. [9][11]
23	**context Person inv uniqueNameOne_P1_VT: Person.allInstances()->one(p:Person|self.name=p.name)**	N/A	true	true	true		Boolean= true	Syntaxfehler.[8]
24	**context Person inv uniqueNameOne_P1_VN: Person.allInstances()->one(p|self.name=p.name)**	N/A	true	true	true		Boolean= true	true

25	**context Person inv uniqueNameOne_P1_VI: Person.allInstances()->one(self.name=name)**	N/A	true	true	true		Boolean= true	Syntaxfehler.[11]
26	**context Person inv uniqueNameOneNot_P0_VI: not Person.allInstances->one(name=name)**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler.[9][11]
27	**context Person inv uniqueNameOneNot_P1_VI: not Person.allInstances()->one(name=name)**	N/A	true	true	true		Boolean= true	Syntaxfehler.[11]
28	**context Person inv femaleHasNoWife: self.gender='female' implies self.wife->isEmpty()**	true	true	true	true	Info: adding type-conversion 'asSet' to 'isEmpty' with source type 'Person'.	Boolean= true	N/A [12]
29	**context Person inv maleHasNoHusband: self.gender='male' implies self.husband->isEmpty()**	true	true	true	true	Info: adding type-conversion 'asSet' to 'isEmpty' with source type 'Person'.	Boolean= true	N/A [12]
30	**context Person inv abcName_P0_VT: let ada:Person=Person.allInstances->any(p:Person|p.name='Ada') in let bob:Person=Person.allInstances->any(p:Person|p.name='Bob') in let cyd:Person=Person.allInstances->any(p:Person|p.name='Cyd') in Set{ada,bob,cyd}->collect(p:Person|p.name)=Bag{'Ada','Bob','Cyd'}**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler.[8][9]
31	**context Person inv abcName_P0_VN: let ada=Person.allInstances->any(p|p.name='Ada') in let bob=Person.allInstances->any(p|p.name='Bob') in let cyd=Person.allInstances->any(p|p.name='Cyd') in Set{ada,bob,cyd}->collect(p|p.name)=Bag{'Ada','Bob','Cyd'}**	true	N/A[6][4]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler.[5][9]
32	**context Person inv abcName_P0_VI: let ada=Person.allInstances->any(name='Ada') in let bob=Person.allInstances->any(name='Bob') in let cyd=Person.allInstances->any(name='Cyd') in Set{ada,bob,cyd}->collect(name)=Bag{'Ada','Bob','Cyd'}**	true	N/A[6][4]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler.[5][9][11]
33	**context Person inv abcName_P1_VT: let ada:Person=Person.allInstances()->any(p:Person|p.name='Ada') in let bob:Person=Person.allInstances()->any(p:Person|p.name='Bob') in let cyd:Person=Person.allInstances()->any(p:Person|p.name='Cyd') in Set{ada,bob,cyd}->collect(p:Person|p.name)=Bag{'Ada','Bob','Cyd'}**	N/A	true	true	true		Boolean= true	Syntaxfehler.[8]
34	**context Person inv abcName_P1_VN: let ada=Person.allInstances()->any(p|p.name='Ada') in let bob=Person.allInstances()->any(p|p.name='Bob') in let cyd=Person.allInstances()->any(p|p.name='Cyd') in Set{ada,bob,cyd}->collect(p|p.name)=Bag{'Ada','Bob','Cyd'}**	N/A	N/A[4]	Syntax Fehler	N/A[4]	Typisierung fehlt	Boolean= true	Syntaxfehler.[5]
35	**context Person inv abcName_P1_VI: let ada=Person.allInstances()->any(name='Ada') in let bob=Person.allInstances()->any(name='Bob') in let cyd=Person.allInstances()->any(name='Cyd') in Set{ada,bob,cyd}->collect(name)=Bag{'Ada','Bob','Cyd'}**	N/A	N/A[4]	Syntax Fehler	N/A[4]	Typisierung fehlt[13]	Boolean= true	Syntaxfehler.[5][11]

36	**context Person inv abcNameDotShortcutP0_VT: let ada:Person=Person.allInstances->any(p:Person|p.name='Ada') in let bob:Person=Person.allInstances->any(p:Person|p.name='Bob') in let cyd:Person=Person.allInstances->any(p:Person|p.name='Cyd') in Set{ada,bob,cyd}.name=Bag{'Ada','Bob','Cyd'}**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler.[8][9]
37	**context Person inv abcNameDotShortcutP0_VN: let ada=Person.allInstances->any(p|p.name='Ada') in let bob=Person.allInstances->any(p|p.name='Bob') in let cyd=Person.allInstances->any(p|p.name='Cyd') in Set{ada,bob,cyd}.name=Bag{'Ada','Bob','Cyd'}**	true	N/A[4][6]	true	N/A[4]	Syntaxfehler[7]	Boolean= true	Syntaxfehler.[5][9]
38	**context Person inv abcNameDotShortcutP0_VI: let ada=Person.allInstances->any(name='Ada') in let bob=Person.allInstances->any(name='Bob') in let cyd=Person.allInstances->any(name='Cyd') in Set{ada,bob,cyd}.name=Bag{'Ada','Bob','Cyd'}**	true	N/A[4][6]	true	N/A[4][6]	Syntaxfehler[7]	Boolean= true	Syntaxfehler.[5][9][11]
39	**context Person inv abcNameDotShortcutP1_VT: let ada:Person=Person.allInstances()->any(p:Person|p.name='Ada') in let bob:Person=Person.allInstances()->any(p:Person|p.name='Bob') in let cyd:Person=Person.allInstances()->any(p:Person|p.name='Cyd') in Set{ada,bob,cyd}.name=Bag{'Ada','Bob','Cyd'}**	N/A	true	true	true		Boolean= true	Syntaxfehler.[8]
40	**context Person inv abcNameDotShortcutP1_VN: let ada=Person.allInstances()->any(p|p.name='Ada') in let bob=Person.allInstances()->any(p|p.name='Bob') in let cyd=Person.allInstances()->any(p|p.name='Cyd') in Set{ada,bob,cyd}.name=Bag{'Ada','Bob','Cyd'}**	N/A	N/A[4]	Syntax Fehler	N/A[4]	Typisierung fehlt	Boolean= true	Syntaxfehler.[5]
41	**context Person inv abcNameDotShortcutP1_VI: let ada=Person.allInstances()->any(name='Ada') in let bob=Person.allInstances()->any(name='Bob') in let cyd=Person.allInstances()->any(name='Cyd') in Set{ada,bob,cyd}.name=Bag{'Ada','Bob','Cyd'}**	N/A	N/A[4]	Syntax Fehler	N/A[4]	Typisierung fehlt	Boolean= true	Syntaxfehler.[5][11]
42	**context Person inv abcAttrs_P0_VT: let ada:Person=Person.allInstances->any(p:Person|p.name='Ada') in let bob:Person=Person.allInstances->any(p:Person|p.name='Bob') in let cyd:Person=Person.allInstances->any(p:Person|p.name='Cyd') in Set{ada,bob,cyd}->collect(p:Person|Sequence{p.name,p.civstat,p.gender,p.alive})= Bag{Sequence{'Ada','widowed' ,'female',true }, Sequence{'Bob','divorced','male' ,true }, Sequence{'Cyd','married' ,'male' ,false}}**	true	N/A[6]	Type mismatch. No common supertype: (String), (Boolean)	N/A [4]	Syntaxfehler[7]	Boolean= false	Syntaxfehler.[8][9]

43	**context Person inv abcAttrs_P0_VN: let ada=Person.allInstances->any(p|p.name='Ada') in let bob=Person.allInstances->any(p|p.name='Bob') in let cyd=Person.allInstances->any(p|p.name='Cyd') in Set{ada,bob,cyd}->collect(p|Sequence{p.name,p.civstat,p.gender,p.alive})= Bag{Sequence{'Ada','widowed','female',true}, Sequence{'Bob','divorced','male',true}, Sequence{'Cyd','married','male',false}}**	true	N/A[4][6]	Type mismatch. No common supertype: (String), (Boolean)	N/A [4]	Syntaxfehler[7]	Boolean= false	Syntaxfehler.[5][9]
44	**context Person inv abcAttrs_P0_VI: let ada=Person.allInstances->any(name='Ada') in let bob=Person.allInstances->any(name='Bob') in let cyd=Person.allInstances->any(name='Cyd') in Set{ada,bob,cyd}->collect(Sequence{name,civstat,gender,alive})= Bag{Sequence{'Ada','widowed','female',true}, Sequence{'Bob','divorced','male',true}, Sequence{'Cyd','married','male',false}}**	true	N/A[4][6]	Type mismatch. No common supertype: (String), (Boolean)	N/A[4]	Syntaxfehler[7]	Boolean= false	Syntaxfehler.[5][9][11]
45	**context Person inv abcAttrs_P1_VT: let ada:Person=Person.allInstances()->any(p:Person|p.name='Ada') in let bob:Person=Person.allInstances()->any(p:Person|p.name='Bob') in let cyd:Person=Person.allInstances()->any(p:Person|p.name='Cyd') in Set{ada,bob,cyd}->collect(p:Person|Sequence{p.name,p.civstat,p.gender,p.alive})= Bag{Sequence{'Ada','widowed','female',true}, Sequence{'Bob','divorced','male',true}, Sequence{'Cyd','married','male',false}}**	N/A	Type mismatch. No common supertype: (String), (Boolean)	Syntax Fehler	N/A [type mismatch].All parts of a collection literal should have the same (super)type.	java.lang.NullPointerException	Boolean= false	Syntax Fehler.[8]
46	**context Person inv abcAttrs_P1_VN: let ada=Person.allInstances()->any(p|p.name='Ada') in let bob=Person.allInstances()->any(p|p.name='Bob') in let cyd=Person.allInstances()->any(p|p.name='Cyd') in Set{ada,bob,cyd}->collect(p|Sequence{p.name,p.civstat,p.gender,p.alive})= Bag{Sequence{'Ada','widowed','female',true}, Sequence{'Bob','divorced','male',true}, Sequence{'Cyd','married','male',false}}**	N/A	N/A[4]	Syntax Fehler	N/A[4]	Typisierung fehlt	Boolean= false	Syntax Fehler.[5]
47	**context Person inv abcAttrs_P1_VI: let ada=Person.allInstances()->any(name='Ada') in let bob=Person.allInstances()->any(name='Bob') in let cyd=Person.allInstances()->any(name='Cyd') in Set{ada,bob,cyd}->collect(Sequence{name,civstat,gender,alive})= Bag{Sequence{'Ada','widowed','female',true}, Sequence{'Bob','divorced','male',true}, Sequence{'Cyd','married','male',false}}**	N/A	N/A[4]	Type mismatch. No common supertype: (String), (Boolean)	N/A[4]	Typisierung fehlt	Boolean= false	Syntax Fehler.[5][11]
48	**context Person inv personsWithUndefinedHusband_P0_VT: Person.allInstances->select(p:Person|p.husband=Person.allInstances->any(p:Person|p.wife->isEmpty()).wife)->collect(p:Person|p.name)=Bag{'Ada','Bob','Cyd','Dan'}**	true	N/A[6]	N/A	N/A[6]	Syntaxfehler[7]	Boolean= Undefined	Syntax Fehler.[8][9][12]

49	**context Person inv personsWithUndefinedHusband_P0_VN: Person.allInstances->select(p| p.husband= Person.allInstances->any(p|p.wife->isEmpty()).wife)->collect(p|p.name)=Bag{'Ada','Bob','Cyd','Dan'}**	true	N/A[6]	N/A	N/A[6]	Syntaxfehler[7]	Boolean= Undefined	N/A[9][12]
50	**context Person inv personsWithUndefinedHusband_P0_VI: Person.allInstances->select(husband= Person.allInstances->any(wife->isEmpty()).wife)->collect(name)=Bag{'Ada','Bob','Cyd','Dan'}**	true	N/A[6]	N/A	N/A[6]	Syntaxfehler[7]	Boolean= Undefined	Syntax Fehler.[9][11][12]
51	**context Person inv personsWithUndefinedHusband_P1_VT: Person.allInstances()->select(p:Person| p.husband= Person.allInstances()->any(p:Person|p.wife->isEmpty()).wife)->collect(p:Person|p.name)=Bag{'Ada','Bob','Cyd','Dan'}**	N/A	N/A[14] (true)	Syntax Fehler	N/A [14]	Info: adding type-conversion 'asSet' to 'isEmpty' with source type 'Person'.	Boolean= Undefined	Syntax Fehler.[8]
52	**context Person inv personsWithUndefinedHusband_P1_VN: Person.allInstances()->select(p| p.husband= Person.allInstances()->any(p|p.wife->isEmpty()).wife)->collect(p|p.name)=Bag{'Ada','Bob','Cyd','Dan'}**	N/A	N/A[14] (true)	Syntax Fehler	N/A [14]	Info: adding type-conversion 'asSet' to 'isEmpty' with source type 'Person'.	Boolean= Undefined	N/A[12]
53	**context Person inv personsWithUndefinedHusband_P1_VI: Person.allInstances()->select(husband= Person.allInstances()->any(wife->isEmpty()).wife)->collect(name)=Bag{'Ada','Bob','Cyd','Dan'}**	N/A	true	N/A	N/A [14]	Info: adding type-conversion 'asSet' to 'isEmpty' with source type 'Person'.	Boolean= Undefined	Syntax Fehler.[11]
54	**context Person inv personsWithUndefinedHusbandAnyFalse_P0_VT: Person.allInstances->select(p:Person| p.husband=Person.allInstances->any(p:Person|false))->collect(p:Person|p.name)=Bag{'Ada','Bob','Cyd','Dan'}**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= Undefined	Syntax Fehler.[8][9]
55	**context Person inv personsWithUndefinedHusbandAnyFalse_P0_VN: Person.allInstances->select(p| p.husband=Person.allInstances->any(p|false))->collect(p|p.name)=Bag{'Ada','Bob','Cyd','Dan'}**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= Undefined	N/A[9]
56	**context Person inv personsWithUndefinedHusbandAnyFalse_P0_VI: Person.allInstances->select(husband=Person.allInstances->any(false))->collect(name)=Bag{'Ada','Bob','Cyd','Dan'}**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= Undefined	Syntax Fehler.[9][11]
57	**context Person inv personsWithUndefinedHusbandAnyFalse_P1_VT: Person.allInstances()->select(p:Person| p.husband=Person.allInstances()->any(p:Person|false)**	N/A	N/A[14] (true)	Syntax Fehler	N/A [14]		Boolean= Undefined	Syntax Fehler.[8]

	)->collect(p:Person|p.name)=Bag{'Ada','Bob','Cyd','Dan'}							
58	**context Person inv personsWithUndefinedHusbandAnyFalse_P1_VN: Person.allInstances()->** **select(p|** **p.husband=Person.allInstances()->any(p|false)** **)->collect(p|p.name)=Bag{'Ada','Bob','Cyd','Dan'}**	N/A	N/A[14] (true)	Syntax Fehler	N/A [14]		Boolean= Undefined	N/A [15]
59	**context Person inv personsWithUndefinedHusbandAnyFalse_P1_VI: Person.allInstances()->** **select(** **husband=Person.allInstances()->any(false)** **)->collect(name)=Bag{'Ada','Bob','Cyd','Dan'}**	N/A	true	true	N/A [14]		Boolean= Undefined	Syntax Fehler.[8]
60	**context Person inv possiblePairs_P0_VT: let ada:Person=Person.allInstances->any(p:Person|p.name='Ada') in let emptySeq:Sequence(Person)=Sequence{ada}->excluding(ada) in Person.allInstances->iterate(w,h:Person; res:Bag(Sequence(Person))=Bag{emptySeq}->excluding(emptySeq)| if w.gender='female' and w.alive and w.civstat<>'married' and h.gender='male' and h.alive and h.civstat<>'married' then res->including(Sequence{w,h}) else res endif)->** **collect(pair:Sequence(Person)|pair->collect(p:Person|p.name))= Bag{Sequence{'Ada','Bob'}}**	true	N/A[16][6]	Syntax Fehler	N/A [6]	Syntaxfehler[7]	Boolean= false	Syntax Fehler.[8][9]
61	**context Person inv possiblePairs_P0_VN: let ada=Person.allInstances->any(p|p.name='Ada') in let emptySeq=Sequence{ada}->excluding(ada) in Person.allInstances->iterate(w,h; res:Bag(Sequence(Person))=Bag{emptySeq}->excluding(emptySeq)| if w.gender='female' and w.alive and w.civstat<>'married' and h.gender='male' and h.alive and h.civstat<>'married' then res->including(Sequence{w,h}) else res endif)->** **collect(pair|pair->collect(p|p.name))= Bag{Sequence{'Ada','Bob'}}**	true	N/A[16][4][6]	Syntax Fehler	N/A[4][6]	Syntaxfehler[7]	Boolean= false	Syntax Fehler[5][9]
62	**context Person inv possiblePairs_P0_VI: let ada=Person.allInstances->any(name='Ada') in let emptySeq=Sequence{ada}->excluding(ada) in Person.allInstances->iterate(w,h; res:Bag(Sequence(Person))=Bag{emptySeq}->excluding(emptySeq)| if w.gender='female' and w.alive and w.civstat<>'married' and h.gender='male' and h.alive and h.civstat<>'married' then res->including(Sequence{w,h}) else res endif)->** **collect(pair|pair->collect(name))= Bag{Sequence{'Ada','Bob'}}**	true	N/A[16][4][6]	Syntax Fehler	N/A[4][6]	Syntaxfehler[7]	Boolean= false	Syntax Fehler[5][9][11]
63	**context Person inv possiblePairs_P1_VT: let ada:Person=Person.allInstances()->any(p:Person|p.name='Ada') in let emptySeq:Sequence(Person)=Sequence{ada}->excluding(ada) in Person.allInstances()->iterate(w,h:Person; res:Bag(Sequence(Person))=Bag{emptySeq}->excluding(emptySeq)| if w.gender='female' and w.alive and w.civstat<>'married' and h.gender='male' and h.alive and h.civstat<>'married' then res->including(Sequence{w,h}) else res endif)->**	N/A	N/A[16]	Syntax Fehler	Syntax Fehler.Collectiontype Bag(String) has no operation or iterator =(Bag(Sequence(String)))	java.lang.AssertionError: type of source expression must not be null	Boolean= false	Syntax Fehler[8]

	collect(pair:Sequence(Person)|pair->collect(p:Person|p.name))= Bag{Sequence{'Ada','Bob'}}							
64	**context Person inv possiblePairs_P1_VN: let ada=Person.allInstances()->any(p|p.name='Ada') in let emptySeq=Sequence{ada}->excluding(ada) in Person.allInstances()->iterate(w,h; res:Bag(Sequence(Person))=Bag{emptySeq}->excluding(emptySeq)| if w.gender='female' and w.alive and w.civstat<>'married' and h.gender='male' and h.alive and h.civstat<>'married' then res->including(Sequence{w,h}) else res endif)-> collect(pair|pair->collect(p|p.name))= Bag{Sequence{'Ada','Bob'}}**	N/A	N/A[16][4]	Syntax Fehler	N/A[4]	Typisierung fehlt	Boolean= false	Syntax Fehler[5]
65	**context Person inv possiblePairs_P1_VI: let ada=Person.allInstances()->any(name='Ada') in let emptySeq=Sequence{ada}->excluding(ada) in Person.allInstances()->iterate(w,h; res:Bag(Sequence(Person))=Bag{emptySeq}->excluding(emptySeq)| if w.gender='female' and w.alive and w.civstat<>'married' and h.gender='male' and h.alive and h.civstat<>'married' then res->including(Sequence{w,h}) else res endif)-> collect(pair|pair->collect(name))= Bag{Sequence{'Ada','Bob'}}**	N/A	N/A[16][4]	Syntax Fehler	N/A[4]	Typisierung fehlt	Boolean= Undefined	Syntax Fehler[5][11]
66	**context Person inv flatten_P0_VT: let dan:Person=Person.allInstances->any(p:Person|p.name='Dan') in let eve:Person=Person.allInstances->any(p:Person|p.name='Eve') in Set{Bag{eve}, Bag{eve.spouse()}, Bag{eve.spouse().spouse()}, Bag{eve.spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse().spouse()}}-> flatten()=Set{dan,eve}**	true	N/A[6]	true	N/A[6]	Syntaxfehler[7]	Boolean= true	Syntax Fehler[8][9]
67	**context Person inv flatten_P0_VN: let dan=Person.allInstances->any(p|p.name='Dan') in let eve=Person.allInstances->any(p|p.name='Eve') in Set{Bag{eve}, Bag{eve.spouse()}, Bag{eve.spouse().spouse()}, Bag{eve.spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse().spouse()}}-> flatten()=Set{dan,eve}**	true	N/A[4][6]	true	N/A[4][6]	Syntaxfehler[7]	Boolean= true	Syntax Fehler[5][9]
68	**context Person inv flatten_P0_VI: let dan=Person.allInstances->any(name='Dan') in let eve=Person.allInstances->any(name='Eve') in Set{Bag{eve}, Bag{eve.spouse()}, Bag{eve.spouse().spouse()}, Bag{eve.spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse().spouse()}}-> flatten()=Set{dan,eve}**	true	N/A[4][6]	true	N/A[4][6]	Syntaxfehler[7]	Boolean= true	Syntax Fehler[5][9][11]

69	**context Person inv flatten_P1_VT: let dan:Person=Person.allInstances()->any(p:Person|p.name='Dan') in let eve:Person=Person.allInstances()->any(p:Person|p.name='Eve') in Set{Bag{eve}, Bag{eve.spouse()}, Bag{eve.spouse().spouse()}, Bag{eve.spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse().spouse()}}->flatten()=Set{dan,eve}**	N/A	true	true	true	java.lang.AssertionError: type of source expression must not be null	Boolean=true	Syntax Fehler[8]
70	**context Person inv flatten_P1_VN: let dan=Person.allInstances()->any(p|p.name='Dan') in let eve=Person.allInstances()->any(p|p.name='Eve') in Set{Bag{eve}, Bag{eve.spouse()}, Bag{eve.spouse().spouse()}, Bag{eve.spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse().spouse()}}->flatten()=Set{dan,eve}**	N/A	N/A[4]	Syntax Fehler	N/A[4]	(Typisierung fehlt)	Boolean=true	Syntax Fehler[5]
71	**context Person inv flatten_P1_VI: let dan=Person.allInstances()->any(name='Dan') in let eve=Person.allInstances()->any(name='Eve') in Set{Bag{eve}, Bag{eve.spouse()}, Bag{eve.spouse().spouse()}, Bag{eve.spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse()}, Bag{eve.spouse().spouse().spouse().spouse().spouse()}}->flatten()=Set{dan,eve}**	N/A	N/A[4]	Syntax Fehler	N/A[4]	(Typisierung fehlt)	Boolean=true	Syntax Fehler[5][11]

Anmerkungen

1. ↑ in ATL ein Klassentyp müssen mit den zugehörigen Package zusammen nutzen. z.B: Set{Person} durch Set{CivilStatusWorld!Person} ersetzen.
2. ↑ OCLE: Indexes are 0-based, but works if we adjust the indexes in the expression
3. ↑ ATL: der Ausdruck "Set{2..self.name.size()}" wird nicht erkannt, genauer gesagt, das " .. " wird nicht erkannt.
4. ↑ Eclipse: let-Variablen immer typisiert
5. ↑ ATL: Typisierung notwendig.
6. ↑ Eclipse: allInstances() mit Operatorklammern
7. ↑ DresdenOCL: allInstances() mit Operatorklammern
8. ↑ ATL: in Operation wird Variable nicht deklariert.
9. ↑ ATL: allInstances existiert nicht in MOF!EClass, also allInstances mit Klammern: allInstances()
10. ↑ ATL unterstützt keine multi Iterator für Operator exists() und forALL().
11. ↑ ATL: Attribute müssen mit Objekt verbunden.
12. ↑ ATL: Fehlermeldung: "could not find operation isEmpty on Void having supertypes: [OclAny]".
13. ↑ DresdenOCL: Diese Kombination tut: `let ada:Person=Person::allInstances()->any(name='Ada') in ...`
14. ↑ Eclipse: Variablennamen dürfen bei Verschachtelung nur einmal verwendet werden
15. ↑ ATL: Fehlermeldung: "could not find operation including on Bag(OclAny) having supertypes: [Collection(OclAny)]."
16. ↑ Eclipse: Iterate nur mit einer Iterationsvariablen

Von "https://muse.informatik.uni-bremen.de/wiki/index.php/OCL_Benchmark_-_Core"

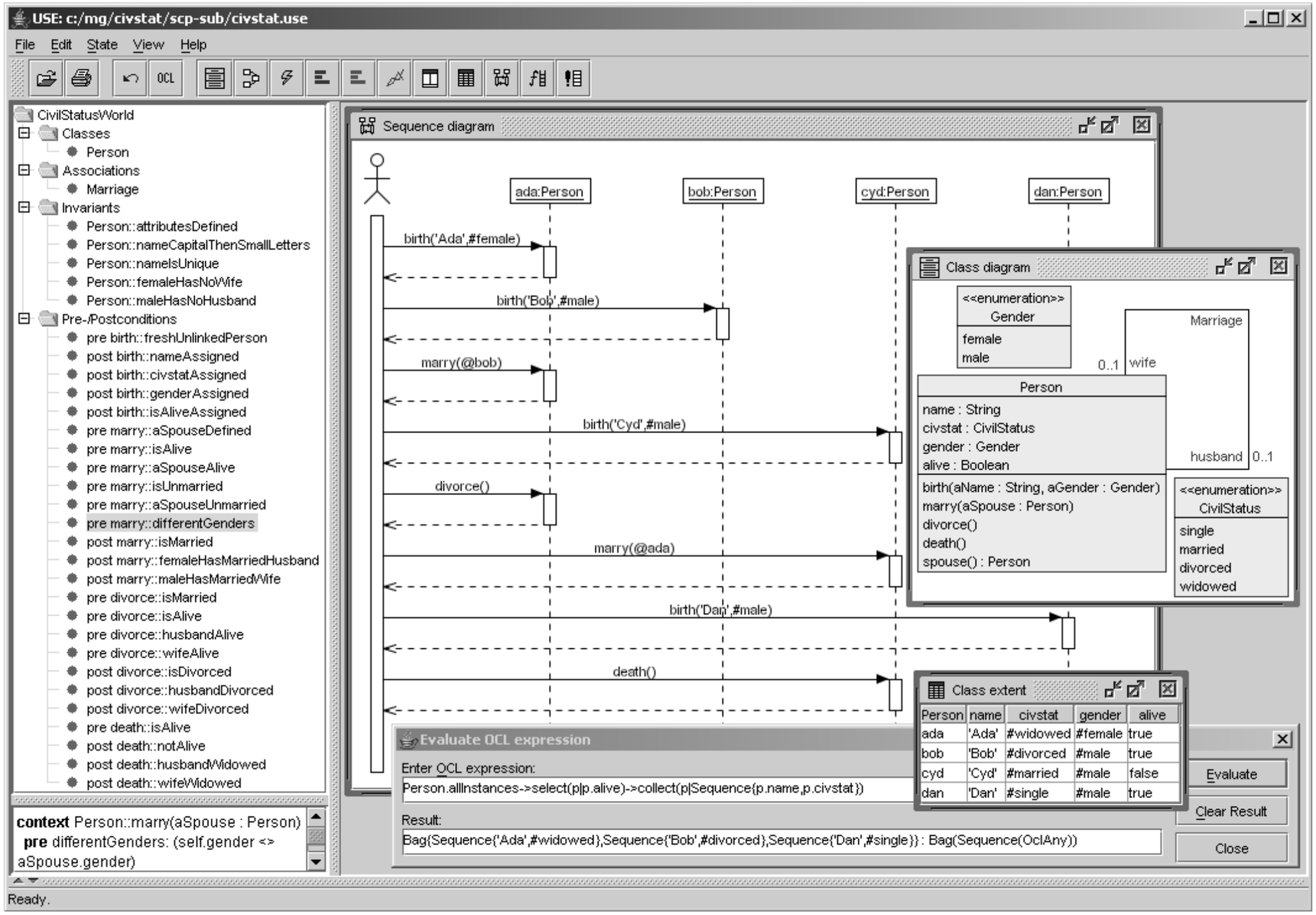
Kategorie: OCLBenchmark

OCL Benchmark 0 - CivilStatus (B2)

Benchmark 0 des OCL Benchmark

Inhaltsverzeichnis

Modell



Ergebnisse der Werkzeuge

Dresden-OCL macht schon beim Parsen von Invarianten und Operationen Probleme.

	USE	Eclipse MDT OCL	RoclET	OCLE	Octopus	...
?Set{ada,bob,cyd,dan}.name	Bag{'Ada','Bob','Cyd','Dan'} : Bag(String)	Bag{Dan : String, Cyd : String, Bob : String, Ada : String}[1]	Bag(String)= Bag{ 'Ada' , 'Bob' , 'Cyd' , 'Dan' } [2]	Bag(String)= Bag{ 'Ada' , 'Bob' , 'Cyd' , 'Dan' } [3]	Bag{'Ada','Bob','Cyd','Dan'}[4]	
?Person.allInstances.name	Bag{'Ada','Bob','Cyd','Dan'} : Bag(String)	Bag{Dan : String, Cyd : String, Bob : String, Ada : String}[5]	Bag{'Ada','Bob','Cyd','Dan'} [6]	Bag(String)= Bag{ *Bob* , *Ada* , *Cyd* , *Dan* }	[Ada, Bob, Cyd, Dan]	
?Set{ada,bob,cyd,dan}->collect(name)	Bag{'Ada','Bob','Cyd','Dan'} : Bag(String)	Bag{Dan : String, Cyd : String, Bob : String, Ada : String}[1]	Bag{'Ada','Bob','Cyd','Dan'} [7]	Bag(String)= Bag{ Bob , Ada , Cyd , Dan }[8]	Bag{'Ada','Bob','Cyd','Dan'} [9]	

?Set{ada,bob,cyd,dan}->collect(p|p.name)	Bag{'Ada','Bob','Cyd','Dan'} : Bag(String)	Bag{Dan : String, Cyd : String, Bob : String, Ada : String}[1]	Bag{'Ada','Bob','Cyd','Dan'} [10]	Bag(String)= Bag{ Bob , Ada , Cyd , Dan }[8]	Bag{'Ada','Bob','Cyd','Dan'} [11]	
?Set{ada,bob,cyd,dan}->collect(Sequence{name,civstat,gender,alive})	Bag{Sequence{'Ada',#widowed, #female,true }, Sequence{'Bob',#divorced,#male, true }, Sequence{'Cyd',#married, #male, false}, Sequence{'Dan',#single, #male, true }} : Bag(Sequence(OclAny))	Type mismatch. No common supertype: (String), (CivilStatus)[1]	Type mismatch. No common supertype: (String), (Boolean)	n/a[12]	n/a[13]	
?Set{ada,bob,cyd,dan}->collect(p|Sequence{p.name,p.civstat,p.gender,p.alive})	Bag{Sequence{'Ada',#widowed, #female,true }, Sequence{'Bob',#divorced,#male, true }, Sequence{'Cyd',#married, #male, false}, Sequence{'Dan',#single, #male, true }}: Bag(Sequence(OclAny))	Type mismatch. No common supertype: (String), (CivilStatus)[1]	Type mismatch. No common supertype: (String), (Boolean)	n/a[12]	n/a[13]	
?Person.allInstances->forAll(p1,p2|p1<>p2 implies p1.name<>p2.name)	true : Boolean	true : Boolean[5]	true [14]	Boolean= true	true : Boolean [5]	
?Person.allInstances->isUnique(p|p.name)	true : Boolean	true : Boolean [5]	true	Boolean= true	true : Boolean [5]	
?Person.allInstances->forAll(p|(p.gender=#female implies p.wife->isEmpty) and (p.gender=#male implies p.husband->isEmpty))	true : Boolean	true : Boolean [5]	true [15]	Boolean= true	true : Boolean [5]	
?Person.allInstances->select(p|p.alive and p.civstat<>#widowed).civstat	Bag{#divorced,#single} : Bag(CivilStatus)	Bag {single, divorced}[16]	true [17]	Bag(CivilStatus)= Bag{ single , divorced }	[single, divorced][5]	
?let p1=Person.allInstances->any(true) in let p2=Person.allInstances->excluding(p1)->any(true) in let p3=Person.allInstances->excluding(p1)->excluding(p2)->any(true) in let p4=Person.allInstances->excluding(p1)->excluding(p2)->excluding(p3)->any(true) in Sequence{p1,p2,p3,p4}=Person.allInstances->asSequence()	true : Boolean	true : Boolean [18]	true[19]	Boolean= true	false : Boolean (Reihenfolge unterscheidet sich bei beiden Sequenzen) [5][18]	
?let o:OclAny=ada in o	ada : Person	Ada : Person[1]	true[20]	OclAny= ada	ada : Person [21]	
?Person.allInstances->iterate(w,h:Person; res:Set(Tuple(bride:Person,bridegroom:Person))= oclEmpty(Set(Tuple(bride:Person,bridegroom:Person)))| if w.gender=#female and h.gender=#male and w.alive and h.alive and w.civstat<>#married and h.civstat<>#married then res->including(Tuple{bride:w,bridegroom:h}) else res endif)	Set{Tuple{bride:ada,bridegroom:bob}, Tuple{bride:ada,bridegroom:dan}} : Set(Tuple(bride:Person,bridegroom:Person))	true mit diesem Ausdruck OCL Benchmark 0 Ausdruck 13 - Eclipse MDT	immer Syntaxfehler, Type mismatch. No common supertype: (Boolean), (Set(Tuple(bride: Person, bridegroom: Person)))[22]	Set(Tuple(bride:Person,bridegroom:Person))= Set{ Tuple{ ada, bob }, Tuple{ ada, dan } }[23]	n/a [24]	
?Sequence{ada.name,ada.civstat,ada.alive,ada.gender, dan.name,dan.civstat,dan.alive,dan.gender}	Sequence{'Ada',#widowed,true,#female, 'Dan',#single,true,#male} : Sequence(OclAny)	Type mismatch. No common supertype: (String), (CivilStatus)[25]	Type mismatch. No common supertype: (String), (Boolean)[26]	Sequence(OclAny)= Sequence{ 'Ada' , widowed , true , female , 'Dan' , single , true , male }	n/a [13]	

!set dan.civstat:=#married !set ada.civstat:=#married !insert (ada,dan) into Marriage ?Sequence{ada,ada.spouse(),dan,dan.spouse()}	Sequence{ada,dan,dan,ada} : Sequence(Person)	Sequence{Ada:Person, Dan:Person, Dan:Person, Ada:Person} [27]	true[28]	Sequence(Person)= Sequence{ ada , dan , dan , ada }	[Ada,Dan,Dan,Ada][29]	
?Sequence{ada,ada.spouse(),ada.spouse().spouse(), ada.spouse().spouse().spouse()}	Sequence{ada,dan,ada,dan} : Sequence(Person)	Sequence{Ada:Person, Dan:Person, Ada:Person, Dan:Person} [30]	true[31]	Sequence(Person)= Sequence{ ada , dan , ada , dan }	[Ada, Dan, Ada, Dan][32]	
?Person.allInstances-> select(husband=Person.allInstances-> any(wife->isEmpty).wife)	Set{bob,cyd,dan} : Set(Person)	Set{dan,cyd,bob}	N/A	Set(Person)=Undefined	n/a [33]	

Person_<...>.cmd

```
-- Person::birth(aName:String,aGender:Gender)
!set self.name:=aName
!set self.civstat:=#single
!set self.gender:=aGender
!set self.alive:=true
```

```
-- Person::marry(aSpouse:Person)
!set self.civstat:=#married
!set aSpouse.civstat:=#married
!insert (if self.gender=#female then self else aSpouse endif, if self.gender=#female then aSpouse else self endif) into Marriage
```

```
-- Person::divorce()
!set self.civstat:=#divorced
!set self.spouse().civstat:=#divorced
!delete (if self.gender=#female then self else self.wife endif, if self.gender=#female then self.husband else self endif) from Marriage
```

```
-- Person::death() -- for married Person objects
!set self.alive:=false
!set self.spouse().civstat:=#widowed
!delete (if self.gender=#female then self else self.wife endif,if self.gender=#female then self.husband else self endif) from Marriage
```

```
-- Person::death() -- for unmarried Person objects
!set self.alive:=false
```

abcd_op.cmd

```
-- read c:/mg/civstat/scp-sub/abcd_cmd.cmd

open c:/mg/civstat/scp-sub/civstat.use

!create ada:Person

!openter ada birth('Ada',#female)
read c:/mg/civstat/scp-sub/Person_birth.cmd
!opexit

!create bob:Person

!openter bob birth('Bob',#male)
read c:/mg/civstat/scp-sub/Person_birth.cmd
!opexit

!openter ada marry(bob)
read c:/mg/civstat/scp-sub/Person_marry.cmd
!opexit

!create cyd:Person

!openter cyd birth('Cyd',#male)
read c:/mg/civstat/scp-sub/Person_birth.cmd
!opexit

!openter ada divorce()
read c:/mg/civstat/scp-sub/Person_divorce.cmd
!opexit

!openter cyd marry(ada)
read c:/mg/civstat/scp-sub/Person_marry.cmd
!opexit

!create dan:Person

!openter dan birth('Dan',#male)
read c:/mg/civstat/scp-sub/Person_birth.cmd
!opexit

!openter cyd death()
read c:/mg/civstat/scp-sub/Person_death_married.cmd
!opexit
```

abcd_flat.cmd

```
-- read c:/mg/civstat/scp-sub/abcd_flat.cmd

open c:/mg/civstat/scp-sub/civstat.use

!create ada:Person

-- !openter ada birth('Ada',#female)
-- read c:/mg/civstat/scp-sub/Person_birth.cmd
-- !opexit

!set ada.name:='Ada'
!set ada.civstat:=#single
!set ada.gender:=#female
!set ada.alive:=true

!create bob:Person

-- !openter bob birth('Bob',#male)
-- read c:/mg/civstat/scp-sub/Person_birth.cmd
-- !opexit

!set bob.name:='Bob'
!set bob.civstat:=#single
!set bob.gender:=#male
!set bob.alive:=true

-- !openter ada marry(bob)
-- read c:/mg/civstat/scp-sub/Person_marry.cmd
-- !opexit

!set ada.civstat:=#married
!set bob.civstat:=#married
!insert (ada,bob) into Marriage

!create cyd:Person

-- !openter cyd birth('Cyd',#male)
-- read c:/mg/civstat/scp-sub/Person_birth.cmd
-- !opexit

!set cyd.name:='Cyd'
!set cyd.civstat:=#single
!set cyd.gender:=#male
!set cyd.alive:=true

-- !openter ada divorce()
-- read c:/mg/civstat/scp-sub/Person_divorce.cmd
-- !opexit

!set ada.civstat:=#divorced
!set bob.civstat:=#divorced
!delete (ada,bob) from Marriage

-- !openter cyd marry(ada)
-- read c:/mg/civstat/scp-sub/Person_marry.cmd
-- !opexit

!set cyd.civstat:=#married
!set ada.civstat:=#married
!insert (ada,cyd) into Marriage

!create dan:Person

-- !openter dan birth('Dan',#male)
-- read c:/mg/civstat/scp-sub/Person_birth.cmd
-- !opexit

!set dan.name:='Dan'
!set dan.civstat:=#single
!set dan.gender:=#male
!set dan.alive:=true

-- !openter cyd death()
-- read c:/mg/civstat/scp-sub/Person_death_married.cmd
-- !opexit

!set cyd.alive:=false
!set ada.civstat:=#widowed
!delete (ada,cyd) from Marriage
```

Anmerkungen

1. ↑ Eclipse: Kein direkter Zugriff auf Objekte, daher über Person.allInstances()->any()
2. ↑

RoclET: Ausdruck:context Person inv Bench0_test1:Set{Person.allInstances()->any(name='Ada'), Person.allInstances()->any(name='Bob'), Person.allInstances()->any(name='Cyd'), Person.allInstances()->any(name='Dan')}.name=Bag{'Dan', 'Cyd', 'Bob', 'Ada'}

3. ↑ OCLE: Kein direkter Zugriff auf ObjectIds: Set{Person.allInstances()->any(p|p.name='Ada'), Person.allInstances()->any(p|p.name='Bob'), Person.allInstances()->any(p|p.name='Cyd'), Person.allInstances()->any(p|p.name='Dan')}.name
4. ↑
Octopus: Kein direkter Zugriff auf Objekte, daher über let ada:Person =Person.allInstances()->any(name='Ada'),bob:Person=Person.allInstances()->any(name='Bob'), cyd:Person = Person.allInstances()->any(name='Cyd'),dan:Person = Person.allInstances()->any(name='Dan') in Set{ada,bob,cyd,dan}.name = Bag{'Ada','Bob','Cyd','Dan'}
5. ↑ Eclipse: Abweichende Syntax (allInstances())
6. ↑ RoclET: durch invariant berechnet. context Person inv test1:Person.allInstances().name=Bag{'Ada', 'Bob','Cyd','Dan'}
7. ↑ RoclET: context Person inv test2:Person.allInstances()->collect(name)=Bag{'Ada', 'Bob','Cyd','Dan'}
8. ↑ OCLE: mittels let und manuelles setzen des Kontext
9. ↑
Octopus: Kein direkter Zugriff auf Objekte, daher über let ada:Person =Person.allInstances()->any(name='Ada'),bob:Person=Person.allInstances()->any(name='Bob'), cyd:Person = Person.allInstances()->any(name='Cyd'),dan:Person = Person.allInstances()->any(name='Dan') in Set{ada,bob,cyd,dan}->collect(name)=Bag{'Ada','Bob','Cyd','Dan'}
10. ↑ RoclET: context Person inv test3:Person.allInstances()->collect(p|p.name)=Bag{'Ada', 'Bob','Cyd','Dan'}
11. ↑
Octopus: Kein direkter Zugriff auf Objekte, daher über let ada:Person =Person.allInstances()->any(name='Ada'),bob:Person=Person.allInstances()->any(name='Bob'), cyd:Person = Person.allInstances()->any(name='Cyd'),dan:Person = Person.allInstances()->any(name='Dan') in Set{ada,bob,cyd,dan}->collect(p|p.name)=Bag{'Ada','Bob','Cyd','Dan'}
12. ↑ OCLE: Mit Bag(OclAny) als return type funktionierts
13. ↑ Octopus: Type mismatch.All parts of a collection literal should have the same (super)type.
14. ↑ RoclET: context Person inv test4:Person.allInstances()->forAll(p1:Person|Person.allInstances()->forAll(p2:Person|p1<>p2 implies p1.name<>p2.name))
15. ↑ RoclET: ohne Enumeration getestet. context Person inv test6:Person.allInstances()->forAll(p|(p.gender='female' implies p.wife->isEmpty()) and (p.gender='male' implies p.husband->isEmpty()))
16. ↑ Eclipse: Person.allInstances()->select(p|p.alive and p.civstat<>CivilStatus::widowed).civstat
17. ↑ RoclET: context Person inv Bench_test10:Person.allInstances()->select(p|p.alive and p.civstat<>'widowed').civstat=Bag{'divorced','single'}
18. ↑ Eclipse: Typ muss angegeben werden let p1 : Person ...
19. ↑
RoclET: Ausdruck:context Person inv test10:let p1:Person=Person.allInstances()->any(true) in let p2:Person=Person.allInstances()->excluding(p1)->any(true) in let p3:Person=Person.allInstances()->excluding(p1)->excluding(p2)->any(true) in let p4:Person=Person.allInstances()->excluding(p1)->excluding(p2)->excluding(p3)->any(true) in Sequence{p1,p2,p3,p4}=Person.allInstances()->asSequence()
20. ↑ RoclET: Ausdruck:context Person inv test11:let o:OclAny=Person.allInstances()->any(name='Ada') in o=Person.allInstances()->any(name='Ada')
21. ↑ Octopus: Ausdruck:let o:OclAny=Person.allInstances()->any(name = 'Ada') in o=Person.allInstances()->any(name='Ada')
22. ↑
RoclET: Ausdruck: context Person inv Bench_test13:let ada:Person=Person.allInstances()->any(name='Ada') in let bob:Person=Person.allInstances()->any(name='Bob') in let dan:Person=Person.allInstances()->any(name='Dan') in Person.allInstances()->iterate(w:Person; res:Set(Tuple(bride:Person,bridegroom:Person)) = Set{Tuple{bride:Person=ada, bridegroom:Person=bob}}->excludes(Tuple{bride:Person=ada, bridegroom:Person=bob})| res->union(Person.allInstances()->iterate(h:Person; res2:Set(Tuple(bride:Person,bridegroom:Person)) = Set{Tuple{bride:Person=ada, bridegroom:Person=bob}}->excludes(Tuple{bride:Person=ada, bridegroom:Person=bob}) | if w.gender='female' and h.gender='male' and w.alive and h.alive and w.civstat<>'married' and h.civstat<>'married' then res2->including(Tuple{bride:Person=w,bridegroom:Person=h}) else res2 endif))) = Set{Tuple{bride:Person=ada, bridegroom:Person=bob}, Tuple{bride:Person=ada, bridegroom:Person=dan}}

23. ↑ OCLE: Anstelle von Tuple TupleType im Definitionsteil verwenden
24. ↑ Octopus: Der Parser kennt den Typ "Tupel" nicht.
25. ↑ Eclipse: Ausdruck: let ada:Person=Person.allInstances()->any(name='Ada'), dan:Person = Person.allInstances()->any(name='Dan') in Sequence{ada.name,ada.civstat,ada.alive,ada.gender,dan.name,dan.civstat,dan.alive,dan.gender}
26. ↑
RoclET: context Person inv test8:Sequence{Person.allInstances()->any(name='Ada').name, Person.allInstances()->any(name='Ada').civstat, Person.allInstances()->any(name='Ada').alive, Person.allInstances()->any(name='Ada').gender, Person.allInstances()->any(name='Dan').name, Person.allInstances()->any(name='Dan').civstat, Person.allInstances()->any(name='Dan').alive, Person.allInstances()->any(name='Dan').gender} = Sequence{'Ada', 'widowed', true, 'female', 'Dan', 'single', true, 'male'}
27. ↑ Eclipse: Ausdruck: let ada:Person=Person.allInstances()->any(name='Ada'), dan:Person = Person.allInstances()->any(name='Dan') in Sequence{ada,ada.spouse(),dan,dan.spouse()}
Details: spouse() wurde in Java implementiert.
Das Ergebnis ist intern eine ArrayList. Demnach eine Sequence.
28. ↑
RoclET: Insert Befehl muss manuell setzen und Ausdruck: context Person inv test12:Sequence{Person.allInstances()->any(name='Ada'), Person.allInstances()->any(name='Ada').spouse(), Person.allInstances()->any(name='Ada').spouse().spouse(), Person.allInstances()->any(name='Ada').spouse().spouse().spouse()} = Sequence{Person.allInstances()->any(name='Ada'), Person.allInstances()->any(name='Dan'), Person.allInstances()->any(name='Ada'), Person.allInstances()->any(name='Ada')}
29. ↑ Octopus: Ausdruck:let ada:Person=Person.allInstances()->any(name='Ada'), dan:Person = Person.allInstances()->any(name='Dan') in Sequence{ada,ada.spouse(),dan,dan.spouse()}
30. ↑ Eclipse: Ausdruck: let ada:Person=Person.allInstances()->any(name='Ada') in Sequence{ada,ada.spouse(),dan,dan.spouse()}
Details: spouse() wurde in Java implementiert.
Das Ergebnis ist intern eine ArrayList. Demnach eine Sequence.
31. ↑
RoclET: Ausdruck:context Person inv Bench_test16:Sequence{Person.allInstances()->any(name='Ada'), Person.allInstances()->any(name='Ada').spouse(), Person.allInstances()->any(name='Ada').spouse().spouse(), Person.allInstances()->any(name='Ada').spouse().spouse().spouse()} = Sequence{Person.allInstances()->any(name='Ada'), Person.allInstances()->any(name='Dan'), Person.allInstances()->any(name='Ada'), Person.allInstances()->any(name='Dan')}
32. ↑ Octopus: Ausdruck: let ada:Person = Person.allInstances()->any(name='Ada'), dan:Person = Person.allInstances()->any(name='Dan') in Sequence{ada,ada.spouse(), ada.spouse().spouse(), ada.spouse().spouse().spouse()}
33. ↑
Octopus: java.lang.NullPointerException während der Laufzeit. Octopus macht keine Überprüfung auf null: für einen eigenmächtigen Zustand von Person darf die Methode p.getHusband() null zurückliefern; wenn die Anweisung "Vergleich" darauf ausgeführt wird, verursacht es den NullPointer-Fehler.

Von "https://muse.informatik.uni-bremen.de/wiki/index.php/OCL_Benchmark_0_-_CivilStatus"

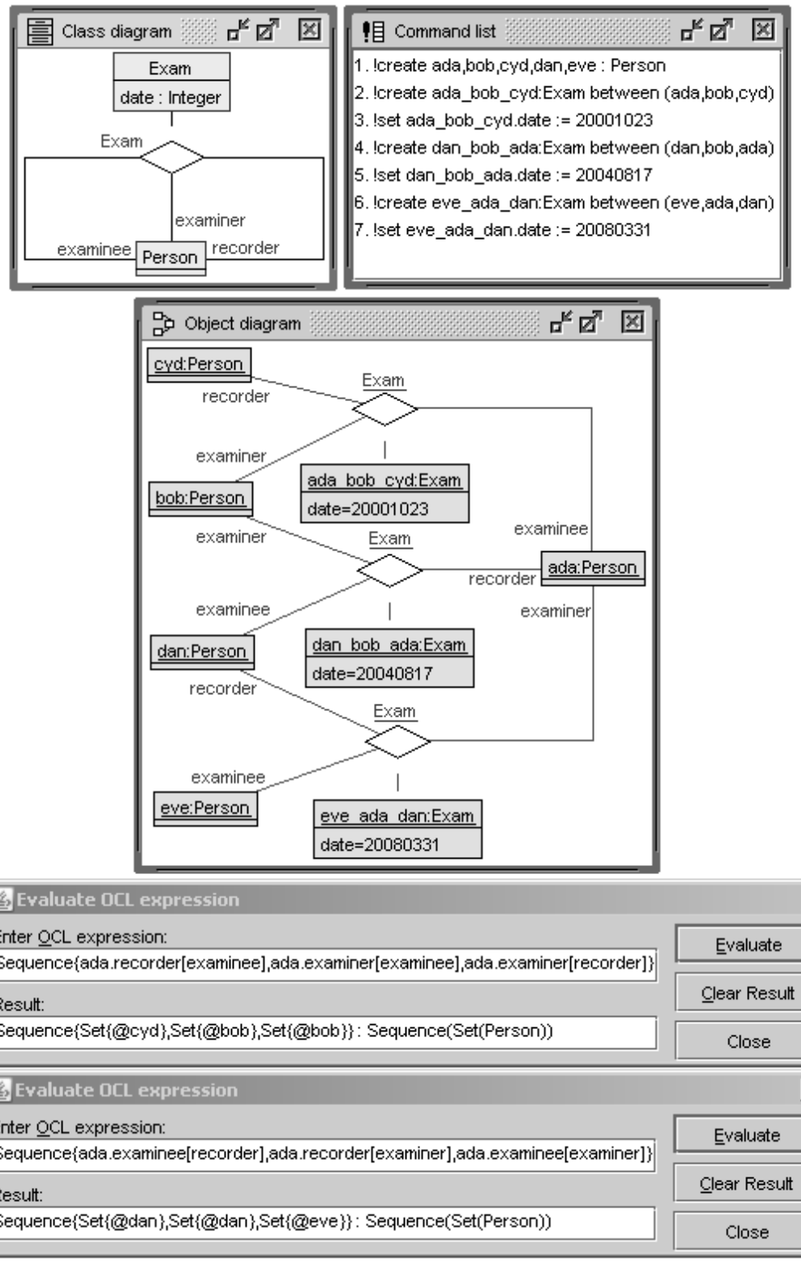
Kategorie: OCLBenchmark

OCL Benchmark 1 - Exams (B3)

Benchmark 1 des OCL Benchmark

Inhaltsverzeichnis

Modell



Ergebnisse der Werkzeuge

***RoclET* und *OCLE* unterstützen keine ternären Assoziationen.**
Dresden-OCL eigentlich schon, trotzdem gibt's Probleme.

	USE	**Eclipse MDT OCL**	**...**
?ada_bob_cyd.examinee	@ada : Person	ada : Person [1]	
?ada_bob_cyd.examiner	@bob : Person	bob : Person [2]	

?ada_bob_cyd.recorder	@cyd : Person	cyd : Person [3]	
?Exam.allInstances-> select(e|e.examinee=ada)	Set{@ada_bob_cyd} : Set(Exam)	Set{ada_bob_cyd : Exam} [4]	
?Exam.allInstances-> select(e|e.recorder=ada)	Set{@dan_bob_ada} : Set(Exam)	Set{dan_bob_ada : Exam} [5]	
?Exam.allInstances-> select(e|e.examiner=ada)	Set{@eve_ada_dan} : Set(Exam)	Set{eve_ada_dan : Exam} [6]	
?ada.examinee[examiner]	Set{@eve} : Set(Person)	ERROR in (variableExpCS): (Unrecognized variable: (examiner)) [7] Sequence{bob : Person}[8]	
?ada.examinee[recorder]	Set{@dan} : Set(Person)	ERROR in (variableExpCS): (Unrecognized variable: (recorder))[9]	
?ada.examiner[examinee]	Set{@bob} : Set(Person)	ERROR in (variableExpCS): (Unrecognized variable: (examinee))[10]	
?ada.examiner[recorder]	Set{@bob} : Set(Person)	ERROR in (variableExpCS): (Unrecognized variable: (recorder))[11]	
?ada.recorder[examinee]	Set{@cyd} : Set(Person)	ERROR in (variableExpCS): (Unrecognized variable: (examinee))[12]	
?ada.recorder[examiner]	Set{@dan} : Set(Person)	ERROR in (variableExpCS): (Unrecognized variable: (examiner))[13]	
?ada.recorder[examiner]	Set{@dan} : Set(Person)	ERROR in (variableExpCS): (Unrecognized variable: (examiner))[14]	
?Exam.allInstances-> select(e|e.examiner=ada)-> collect(e|e.recorder)	Bag{@dan,@dan} : Bag(Person)	Bag{dan : Person, dan : Person} [15]	

Exam_<...>.cmd

Zustand 1

```
!create ada,bob,cyd,dan,eve:Person

-- Exam(examinee,examiner,recorder)
!create ada_bob_cyd:Exam between (ada,bob,cyd)
!set ada_bob_cyd.date:=20001023

?ada_bob_cyd.examinee
?ada_bob_cyd.examiner
?ada_bob_cyd.recorder
```

Zustand 2

```
!create dan_bob_ada:Exam between (dan,bob,ada)
!set dan_bob_ada.date:=20040817

!create eve_ada_dan:Exam between (eve,ada,dan)
!set eve_ada_dan.date:=20080331

?Exam.allInstances->select(e|e.examinee=ada)
?Exam.allInstances->select(e|e.recorder=ada)
?Exam.allInstances->select(e|e.examiner=ada)

?ada.examinee[examiner]
?ada.examinee[recorder]
?ada.examiner[examinee]
?ada.examiner[recorder]
?ada.recorder[examinee]
?ada.recorder[examiner]
```

Zustand 3

```
!create flo:Person
!create flo_ada_dan:Exam between (flo,ada,dan)
!set flo_ada_dan.date:=20080331

?ada.recorder[examiner]


?Exam.allInstances->select(e|e.examiner=ada)->collect(e|e.recorder)
```

Anmerkungen

1. ↑ Eclipse: Ausdruck: Exam.allInstances()->any(examID='ada_bob_cyd').examinee
2. ↑ Eclipse: Ausdruck: Exam.allInstances()->any(examID='ada_bob_cyd').examiner
3. ↑ Eclipse: Ausdruck: Exam.allInstances()->any(examID='ada_bob_cyd').recorder
4. ↑ Eclipse: Ausdruck: Exam.allInstances()->select(e|e.examinee=Person.allInstances()->any(name='ada'))
5. ↑ Eclipse: Ausdruck: Exam.allInstances()->select(e|e.recorder=Person.allInstances()->any(name='ada'))
6. ↑ Eclipse: Ausdruck: Exam.allInstances()->select(e|e.examiner=Person.allInstances()->any(name='ada'))
7. ↑ Eclipse: Person.allInstances()->any(name='ada').examinee[examiner]
8. ↑ Eclipse: Ausdruck: Person.allInstances()->any(name='ada').examinee.examiner
9. ↑ Eclipse: Person.allInstances()->any(name='ada').examinee[recorder]
10. ↑ Eclipse: Person.allInstances()->any(name='ada').examiner[examinee]
11. ↑ Eclipse: Person.allInstances()->any(name='ada').examiner[recorder]
12. ↑ Eclipse: Person.allInstances()->any(name='ada').recorder[examinee]
13. ↑ Eclipse: Person.allInstances()->any(name='ada').recorder[examiner]
14. ↑ Eclipse: Person.allInstances()->any(name='ada').recorder[examiner]
15. ↑ Eclipse: Ausdruck: Exam.allInstances()->select(e|e.examiner=Person.allInstances()->any(name='ada'))->collect(e|e.recorder)

OCL Benchmark 3 - Undefined (B4)

Benchmark 3 des OCL Benchmark

Ergebnisse der Werkzeuge

	USE	**Eclipse MDT OCL[1]**	**RoclET [2]**	**OCLE**	**Octopus**	**ATL Simple OCL Web Tester**	**Dresden-OCL Parser**
?true or Sequence{true}->excluding(true)->last()	true : Boolean	true	true	Boolean= true	true : Boolean	true	
?Sequence{true}->excluding(true)->last() or true	true : Boolean	Invalid Class[3]	N/A	Boolean=last() called for an empty sequence	true : Boolean	wird nicht ausgewertet[4]	
?let B=Set{Sequence{true}->excluding(true)->last(),false,true} in B->iterate(b1,b2:Boolean;r:Sequence(Boolean) =oclEmpty(Sequence(Boolean))|r->including(b1 or b2))	Sequence{Undefined, Undefined, true, Undefined, false, true, true, true, true} : Sequence(Boolean)	",b2:Boolean misplaced construct(s)" Sequence(false, true, true, true) bei diesem Ausdruck OCL Benchmark 3 - Ausdruck 3 Eclipse MDT[5]	Type mismatch. No common supertype: (Boolean), (Sequence(Boolean))	Sequence(Boolean)=last() called for an empty sequence	N/A primitive parameter vs. objektwertige Verwendung.	wird nicht ausgewertet[6]	Syntaxfehler[7]
?false and Sequence{true}->excluding(true)->last()	false : Boolean	false	false	Boolean=false	false: Boolean	false	
?Sequence{true}->excluding(true)->last() and false	false : Boolean	Invalid Class[3]	N/A	Boolean=last() called for an empty sequence	false : Boolean	wird nicht ausgewertet[4]	
?let B=Set{Sequence{true}->excluding(true)->last(),false,true} in B->iterate(b1,b2:Boolean;r:Sequence(Boolean) =oclEmpty(Sequence(Boolean))|r->including(b1 and b2))	Sequence{Undefined, false, Undefined, false, false, false, Undefined, false, true} : Sequence(Boolean)	",b2:Boolean misplaced construct(s)"Sequence(false, true, true, true) bei diesem Ausdruck OCL Benchmark 3 - Ausdruck 6 Eclipse MDT[5]	Type mismatch. No common supertype: (Boolean), (Sequence(Boolean))	Sequence(Boolean)=last() called for an empty sequence	N/A primitive parameter vs. objektwertige Verwendung.	wird nicht ausgewertet[8]	Syntaxfehler[7]
?false implies Sequence{true}->excluding(true)->last()	true : Boolean	true	true	Boolean=true	true : Boolean	true	
?Sequence{true}->excluding(true)->last() implies true	true : Boolean	Invalid Class[3]	N/A	Boolean=last() called for an empty sequence	true : Boolean	wird nicht ausgewertet[4]	
?let B=Set{Sequence{true}->excluding(true)->last(),false,true} in B->iterate(b1,b2:Boolean;r:Sequence(Boolean) =oclEmpty(Sequence(Boolean))|r->including(b1 implies b2))	Sequence{Undefined, Undefined, true, true, true, true, Undefined, false, true} : Sequence(Boolean)	",b2:Boolean misplaced construct(s)" Sequence(true, true, false, true) bei diesem Ausdruck OCL Benchmark 3 - Ausdruck 9 Eclipse MDT[5]	N/A	Sequence(Boolean)=last() called for an empty sequence	N/A primitive parameter vs. objektwertige Verwendung.	wird nicht ausgewertet	Syntaxfehler[7]
?if true then false else Sequence{true}->excluding(true)->last() endif	false : Boolean	false	false	Boolean=false	false: Boolean	false	
?if false then Sequence{true}->excluding(true)->last() else true endif	true : Boolean	true	true	Boolean=true	true : Boolean	true	
?Sequence{true}-> excluding(true)-> last().oclIsUndefined()	true : Boolean (mit isUndefined)	true	true	Boolean.OclIsUndefined not found	false: Boolean	wird nicht ausgewertet	

Anmerkungen

1. ↑ Eclipse: Ausdrücke immer typisiert: let c:Set(Integer)|Bag(Integer)|Sequence(Integer) ...
2. ↑ RoclET: Ausdrücke immer typisiert: let c:Set(Integer)|Bag(Integer)|Sequence(Integer) ...
3. ↑ Eclipse: Aufruf von or auf oclUndefined liefert oclInvalid
4. ↑ ATL: Sequence{true}->excluding(true) lieft Sequence{}, Sequence{}->last() wird auf ein leeres Sequence aufgerufen
5. ↑ Eclipse: Collections erlauben kein oclUndefined als Element, daher vier Elemente (2*2)
6. ↑
   ATL: Sequence{true}->excluding(true)->last() wird auf ein leeres Sequence aufgerufen. Zum Testzweck wird die OCL-Ausdücke let B=Set{Sequence{true}->last(),false,true} in

B->iterate(b1,b2:Boolean;r:Sequence(Boolean)=oclEmpty(Sequence(Boolean))|r->including(b1 or b2)) erfolgreich bewertet, wird Set{true,true} als Result herausgekommen.
7. ↑ DresdenOCL: Syntaxfehler bei `oclEmpty(Sequence(Boolean))`
8. ↑
ATL: Sequence{true}->excluding(true)->last() wird auf ein leeres Sequence aufgerufen. Zum Testzweck wird die OCL-Ausdücke let B=Set{Sequence{true}->last(),false,true} in B->iterate(b1,b2:Boolean;r:Sequence(Boolean)=oclEmpty(Sequence(Boolean))|r->including(b1 and b2)) erfolgreich bewertet, wird Set{false} als Result herausgekommen.

Von "https://muse.informatik.uni-bremen.de/wiki/index.php/OCL_Benchmark_3_-_Undefined"

Kategorie: OCLBenchmark

OCL Benchmark 4 - Teil 1 (B5)

Teil 1 von OCL Benchmark 4 - Collection Operations

Inhaltsverzeichnis

p04_e01: Reject 2 Select

	USE	Eclipse MDT OCL [1] [2] [3]	RoclET[1]	OCLE	Octopus [1] [2] [4]	ATL Simple OCL Web Tester [1] [5]
?let c = oclEmpty(Set(Integer)) in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true[7]	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer)} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer)} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	N/A[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Set{1} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true

?let c = Set{4} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 1, 2, 3} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undeined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer), 1, 2, 3} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Set{1, 2, 3} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 1} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 1} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{2, 3, 4, 5} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 4} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 2} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 4} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 2} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5} in c->reject(i|i<4) = c->select(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true

p04_e02: Select 2 Reject

	USE	Eclipse MDT OCL[1]	RoclET[1]	OCLE	Octopus[1] [2][4]	ATL Simple OCL Web Tester [1] [5]
?let c = oclEmpty(Set(Integer)) in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer)} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer)} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	N/A[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Set{1} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 1, 2, 3} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer), 1, 2, 3} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Set{1, 2, 3} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 1} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 1} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true

?let c = Sequence{1, 2, 3} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{2, 3, 4, 5} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 4} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 2} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 4} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 2} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5} in c->select(i|i<4) = c->reject(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true

p04_e03: Exists 2 forAll

	USE	**Eclipse MDT OCL[1]**	**RoclET[1]**	**OCLE**	**Octopus[1] [2][4]**	**ATL Simple OCL Web Tester [1] [5]**
?let c = oclEmpty(Set(Integer)) in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Set(Integer)) in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer)} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer)} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	false[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Set{1} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true

?let c = Sequence{4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Set{1, 2, 3, 4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 4, 1} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 4, 1} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4, 5, 6} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4, 5, 6, 4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4, 5, 6} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4, 5, 6, 4} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4, 5, 6} in c->exists(i|i<4) = not c->forAll(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true

p04_e04: forAll 2 Exists

	USE	**Eclipse MDT OCL[1]**	**RoclET[1]**	**OCLE**	**Octopus [1] [2][4]**	**ATL Simple OCL Web Tester [5]**
?let c = oclEmpty(Set(Integer)) in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true[6]	Boolean= true	true	true

?let c = Set{oclUndefined(Integer)} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer)} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer)} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	false[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Set{1} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	false[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	false[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Set{1, 2, 3} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 1} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 1} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{2, 3, 4, 5} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 4} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true

?let c = Bag{2, 3, 4, 5, 2} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 4} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 2} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5} in c->forAll(i|i<4) = not c->exists(i|not(i<4))	true : Boolean	true	true	Boolean= true	true	true

p04_e05: one 2 exists forAll (extern)

	USE	**Eclipse MDT OCL[1]**	**RoclET[1]**	**OCLE**	Octopus [1][2][4]	**ATL Simple OCL Web Tester [5]**
?let c = oclEmpty(Set(Integer)) in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true[6]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer)} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer)} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	N/A[8]	Boolean= Undefined	true	wird nicht ausgewertet[9]
?let c = Set{1} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Bag{1} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Sequence{1} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Set{4} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Bag{4} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Sequence{4} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Set{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	N/A[8]	Boolean= Undefined	true	wird nicht ausgewertet[9][10]

?let c = Bag{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	N/A[8]	Boolean= Undefined	true	wird nicht ausgewertet[9][10]
?let c = Bag{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	N/A[8]	Boolean= Undefined	true	wird nicht ausgewertet[9][10]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	N/A[8]	Boolean= Undefined	true	wird nicht ausgewertet[9][10]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	N/A[8]	Boolean= Undefined	true	wird nicht ausgewertet[9][10]
?let c = Set{1, 4, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	false [4]	wird nicht ausgewertet[10]
?let c = Bag{1, 4, 5, 4} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	false [4]	wird nicht ausgewertet[10]
?let c = Bag{1, 4, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	false [4]	wird nicht ausgewertet[10]
?let c = Sequence{1, 4, 5, 4} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	false [4]	wird nicht ausgewertet[10]
?let c = Sequence{1, 4, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	false [4]	wird nicht ausgewertet[10]
?let c = Set{1, 2, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Bag{1, 2, 5, 1} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Bag{1, 2, 5, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Bag{1, 2, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Sequence{1, 2, 5, 1} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Sequence{1, 2, 5, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Sequence{1, 2, 5} in c->one(i|i<4) = (c->exists(i|i<4) and c->forAll(x,y|x<4 and y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]

p04_e06: one 2 exists forAll (intern)

	USE	**Eclipse MDT OCL[1]**	**RocIET[1]**	**OCLE**	**Octopus [1][2][11]**	**ATL Simple OCL Web Tester [1] [5]**
?let c = oclEmpty(Set(Integer)) in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true[6]	Boolean= true	true [4]	true
?let c = oclEmpty(Bag(Integer)) in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true[6]	Boolean= true	true [4]	true

?let c = oclEmpty(Sequence(Integer)) in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true[6]	Boolean= true	true [4]	true
?let c = Set{oclUndefined(Integer)} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true[8]	Boolean= Undefined	N/A [11]	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer)} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true[8]	Boolean= Undefined	n/a [11]	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer)} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true[8]	Boolean= Undefined	n/a [11]	wird nicht ausgewertet[9]
?let c = Set{1} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	N/A [11]	true
?let c = Bag{1} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Sequence{1} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Set{4} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Bag{4} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true [4]	true
?let c = Sequence{4} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	true [4]	true
?let c = Set{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true[8]	Boolean= Undefined	N/A [11]	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true[8]	Boolean= Undefined	n/a [11]	wird nicht ausgewertet[9]
?let c = Bag{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true[8]	Boolean= Undefined	n/a [11]	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	false[8]	Boolean= Undefined	n/a [11]	wird nicht ausgewertet[9]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	false[8]	Boolean= Undefined	n/a [11]	wird nicht ausgewertet[9]
?let c = Set{1, 4, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Bag{1, 4, 5, 4} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Bag{1, 4, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Sequence{1, 4, 5, 4} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Sequence{1, 4, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Set{1, 2, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true

?let c = Bag{1, 2, 5, 1} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Bag{1, 2, 5, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Bag{1, 2, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Sequence{1, 2, 5, 1} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Sequence{1, 2, 5, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true
?let c = Sequence{1, 2, 5} in c->one(i|i<4) = c->exists(x|x<4 and c->forAll(y|y<4 implies x = y))	true : Boolean	true	true	Boolean= true	n/a [11]	true

Anmerkungen

1. ↑ Ausdrücke immer typisiert: let c:Set(Integer)|Bag(Integer)|Sequence(Integer) ...
2. ↑ Statt oclEmpty(...) wird c:ColType(Type) = ColType{} verwendet. Beispiel: c:Set(Integer) = Set{}
3. ↑ Eclipse: Statt oclUndefined(...) wird c:ColType(Type) = ColType{ColType{}->any(false)}. Beispiel: c:Set(Integer) = Set{Set{}->any(false)}
4. ↑ Octopus: Statt ColType{oclUndefined(Integer)} wird ColType{ColType{1}->any(false)}. Beispiel: Set{oclUndefined(Integer)}= Set{Set{1}->any(false)}
5. ↑ ATL: statt oclEmpty(Set(Integer)), oclEmpty(Bag(Integer)), oclEmpty(Sequence(Integer)) wird durch Set{},Bag{},Sequence{} ersetzen
6. ↑ RoclET: oclEmpty(Set(Integer)), oclEmpty(Bag(Integer)), oclEmpty(Sequence(Integer)) durch Set{1}->reject(i|i=1), Bag{1}->reject(i|i=1),Sequence{1}->reject(i|i=1) ersetzen
7. ↑ Eclipse: Vergleicht zwei leere Sets. Siehe Ergebnisdokument
8. ↑ RoclET: oclUndefined(Integer) durch Set{1}->any(false) ersetzen
9. ↑ ATL: Collections erlauben kein oclUndefined als Element, hier kann nicht typisiert
10. ↑ Die OCL Spezifikation erlaubt multi Iterationen für exists() und forAll(), aber es wird nicht von aktueller ATL Implementation unterstützt. Es wird nur 1 Iteration erlaubt. Vergleichen mit obene Teste, die oclEmpty(...) nutzt, weil es in Collection keine Elemente gibt und wird 2 leere Collection vergleichen, sowieso wird true. Bei diesem Fall wird es sich nicht berücksichtigt, was in exists() und forAll() innnen steht.
11. ↑ im Ausdruck sieht wie (forAll(..|..<.. implies .. = ...))) aus.Octopus lässt den Teil (implies ...=...) immer weg.Es generiert keine Code dafür.

Von "https://muse.informatik.uni-bremen.de/wiki/index.php/OCL_Benchmark_4_-_Teil_1"

Kategorie: OCLBenchmark

OCL Benchmark 4 - Teil 2 (B5)

Teil 2 von OCL Benchmark 4 - Collection Operations

Inhaltsverzeichnis

p04_e07: exists 2 reject

	USE	Eclipse MDT OCL [1] [2] [3]	RoclET[1]	OCLE [4]	Octopus [1][5]	ATL Simple OCL Web Tester [6]
?let c = oclEmpty(Set(Integer)) in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true[7]	Boolean= true	true	true [8]
?let c = oclEmpty(Bag(Integer)) in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true[7]	Boolean= true	true	true[8]
?let c = oclEmpty(Sequence(Integer)) in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	N/A[7]	Boolean= true	true	true[8]
?let c = Set{oclUndefined(Integer)} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer)} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer)} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true	Boolean= true	true	true[8]
?let c = Bag{1} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true	Boolean= true	true	true[8]
?let c = Sequence{1} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	N/A	Boolean= true	true	true[8]
?let c = Set{4} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true	Boolean= true	true	true[8]
?let c = Bag{4} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true	Boolean= true	true	true[8]
?let c = Sequence{4} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	N/A	Boolean= true	true	true[8]
?let c = Set{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1, 2, 3, 4} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true	Boolean= true	true	true[8]
?let c = Bag{1, 2, 3, 4, 1} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true	Boolean= true	true	true[8]
?let c = Bag{1, 2, 3, 4} in c->exists(i|i<4) = c->reject(i|i<4)->size()<c->size()	true : Boolean	true	true	Boolean= true	true	true[8]

p04_e08: exists 2 select

	USE	Eclipse MDT OCL[1]	RoclET[1]	OCLE	Octopus [1][9]	ATL Simple OCL Web Tester [6]
?let c = oclEmpty(Set(Integer)) in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true[7]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true[7]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true[7]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer)} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer)} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1, 2, 3, 4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 4, 1} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true

?let c = Bag{1, 2, 3, 4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 4, 1} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4, 5, 6} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4, 5, 6, 4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4, 5, 6} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4, 5, 6, 4} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4, 5, 6} in c->exists(i|i<4) = c->select(i|i<4)->notEmpty()	true : Boolean	true	true	Boolean= true	true	true

p04_e09: forAll 2 reject

	USE	**Eclipse MDT OCL[1]**	**RoclET[1]**	**OCLE**	**Octopus[1][9]**	**ATL Simple OCL Web Tester [6]**
?let c = oclEmpty(Set(Integer)) in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true[7]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true[7]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true[7]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer)} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer)} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true

?let c = Sequence{4} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1, 2, 3} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 1} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 1} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{2, 3, 4, 5} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 4} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 2} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 4} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 2} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5} in c->forAll(i|i<4) = c->reject(i|i<4)->isEmpty()	true : Boolean	true	true	Boolean= true	true	true

p04_e10: forAll 2 select

	USE	**Eclipse MDT OCL[1]**	**RoclET[1]**	**OCLE**	**Octopus[1][9]**	**ATL Simple OCL Web Tester [6]**
?let c = oclEmpty(Set(Integer)) in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true[7]	Boolean= true	true	true

?let c = oclEmpty(Bag(Integer)) in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true[7]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true[7]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer)} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer)} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1, 2, 3} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 1} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 1} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true

?let c = Set{2, 3, 4, 5} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 4} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 2} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 4} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 2} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5} in c->forAll(i|i<4) = (c->select(i|i<4) = c)	true : Boolean	true	true	Boolean= true	true	true

p04_e11: one 2 reject

	USE	**Eclipse MDT OCL**	**RoclET**[1]	**OCLE**	**Octopus** [1][9]	**ATL Simple OCL Web Tester** [6]
?let c = oclEmpty(Set(Integer)) in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true[7]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true[7]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	N/A[7]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer)} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer)} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Set{4} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true

?let c = Set{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	N/A[9]	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1, 4, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 4, 5, 4} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 4, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 4, 5, 4} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Sequence{1, 4, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Set{1, 2, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 5, 1} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 5, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 5, 1} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Sequence{1, 2, 5, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Sequence{1, 2, 5} in c->one(i|i<4) = (c->reject(i|i<4)->size() = c->size()-1)	true : Boolean	true	N/A	Boolean= true	true	true

p04_e12: one 2 select

	USE	Eclipse MDT OCL[1]	RoclET	OCLE	Octopus [1][9]	ATL Simple OCL Web Tester [6]
?let c = oclEmpty(Set(Integer)) in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true

?let c = oclEmpty(Sequence(Integer)) in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer)} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer)} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Set{4} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Bag{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[10]
?let c = Set{1, 4, 5} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 4, 5, 4} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 4, 5} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 4, 5, 4} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Sequence{1, 4, 5} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Set{1, 2, 5} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true

?let c = Bag{1, 2, 5, 1} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 5, 5} in c->one(i|i<4) = (c->select(i|i<4)->size() = 1)	true : Boolean	true	true	Boolean= true	true	true

Anmerkungen

1. ↑ Ausdrücke immer typisiert: let c:Set(Integer)|Bag(Integer)|Sequence(Integer) ...
2. ↑ Statt oclEmpty(...) wird c:ColType(Type) = ColType{} verwendet. Beispiel: c:Set(Integer) = Set{}
3. ↑ Statt oclUndefined(...) wird c:ColType(Type) = ColType{ColType{}->any(false)}. Beispiel: c:Set(Integer) = Set{Set{}->any(false)}
4. ↑ Hier müssen wir beide Vergleichsseiten des '<'-Operators in Klammern setzen, da die Datei ansonsten nicht kompiliert.
5. ↑ Octopus: Statt ColType{oclUndefined(Integer)} wird ColType{ColType{1}->any(false)}. Beispiel: Set{oclUndefined(Integer)}= Set{Set{1}->any(false)}
6. ↑ Statt oclEmpty(Set(Integer)), oclEmpty(Bag(Integer)), oclEmpty(Sequence(Integer)) wird durch Set{}, Bag{},Sequence{} ersetzen
7. ↑ oclEmpty(Set(Integer)), oclEmpty(Bag(Integer)), oclEmpty(Sequence(Integer)) durch Set{1}->reject(i|i=1), Bag{1}->reject(i|i=1),Sequence{1}->reject(i|i=1) ersetzen
8. ↑ Kleine Bemerkung: um die OCL-Ausdücke rechtig bewerten, muss c->reject(i|i<4)->size()<c->size() als (c->reject(i|i<4)->size()<c->size()) eingeben.
9. ↑ oclUndefined(Integer) durch Set{1}->any(false) ersetzen
10. ↑ Collections erlauben kein oclUndefined als Element, hier kann nicht typisiert.

Von "https://muse.informatik.uni-bremen.de/wiki/index.php/OCL_Benchmark_4_-_Teil_2"

Kategorie: OCLBenchmark

OCL Benchmark 4 - Teil 3

Teil 3 von OCL Benchmark 4 - Collection Operations

Inhaltsverzeichnis

p04_e13: exists 2 collect & includes

	USE	Eclipse MDT OCL	RocIET[1]	OCLE	Octopus[1] [2][3]	ATL Simple OCL Web Tester[4]
?let c = oclEmpty(Set(Integer)) in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]

?let c = Sequence{oclUndefined(Integer)} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Set{1} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Set{1, 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 4, 1} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 4, 1} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4, 5, 6} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4, 5, 6, 4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4, 5, 6} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4, 5, 6, 4} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true

?let c = Sequence{4, 5, 6} in c->exists(i|i<4) = c->collect(i|i<4)->includes(true)	true : Boolean	true	true	Boolean= true	true	true

p04_e14: exists 2 collect & one

	USE	**Eclipse MDT OCL**	**RoclET[1]**	**OCLE**	Octopus [1][6] [2]	**ATL Simple OCL Web Tester [4]**
?let c = oclEmpty(Set(Integer)) in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Set{1} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	false[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	false[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	false[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	false[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	false[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Set{1, 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	false	Boolean= true	true	false

?let c = Bag{1, 2, 3, 4, 1} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	false	Boolean= true	true	false
?let c = Bag{1, 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	false	Boolean= true	true	false
?let c = Sequence{1, 2, 3, 4, 1} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	false	Boolean= true	true	false
?let c = Sequence{1, 2, 3, 4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	false	Boolean= true	true	false
?let c = Set{4, 5, 6} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4, 5, 6, 4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4, 5, 6} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4, 5, 6, 4} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4, 5, 6} in c->exists(i|i<4) = c->collect(i|i<4)->asSet()->one(e|e)	true : Boolean	true	true	Boolean= true	true	true

p04_e15: forAll 2 collect & excludes

	USE	**Eclipse MDT OCL**	**RocIET[1]**	**OCLE**	Octopus [1][6] [2][5]	**ATL Simple OCL Web Tester [4]**
?let c = oclEmpty(Set(Integer)) in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	false[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Set{1} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true

?let c = Bag{4} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	false[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	false[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Set{1, 2, 3} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 1} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 1} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{2, 3, 4, 5} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 4} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 2} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 4} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 2} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5} in c->forAll(i|i<4) = c->collect(i|i<4)->excludes(false)	true : Boolean	true	true	Boolean= true	true	true

p04_e16: forAll 2 collect & one

	USE	**Eclipse MDT OCL**	**RoclET[1]**	**OCLE**	**Octopus**	**ATL Simple OCL Web Tester [4]**

?let c = oclEmpty(Set(Integer)) in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true[5]	Boolean= true[8]	true	wird nicht ausgewertet[9]
?let c = oclEmpty(Bag(Integer)) in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true[5]	Boolean= true [8]	true	wird nicht ausgewertet[9]
?let c = oclEmpty(Sequence(Integer)) in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true[5]	Boolean= true [8]	true	wird nicht ausgewertet[9]
?let c = Set{oclUndefined(Integer)} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Set{1} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Bag{1} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Sequence{1} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Set{4} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Bag{4} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Sequence{4} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Set{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	false[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	false[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	false[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Set{1, 2, 3} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	false	Boolean= true	true	wird nicht ausgewertet[11]
?let c = Bag{1, 2, 3, 1} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	false	Boolean= true	true	wird nicht ausgewertet[11]
?let c = Bag{1, 2, 3} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	false	Boolean= true	true	wird nicht ausgewertet[11]

?let c = Sequence{1, 2, 3, 1} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	false	Boolean= true	true	wird nicht ausgewertet[11]
?let c = Sequence{1, 2, 3} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	false	Boolean= true	true	wird nicht ausgewertet[11]
?let c = Set{2, 3, 4, 5} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Bag{2, 3, 4, 5, 4} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Bag{2, 3, 4, 5, 2} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Bag{2, 3, 4, 5} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Sequence{2, 3, 4, 5, 4} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Sequence{2, 3, 4, 5, 2} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]
?let c = Sequence{2, 3, 4, 5} in c->forAll(i|i<4) = (let s = c->collect(i|i<4)->asSet() in c->notEmpty() implies s->one(true) and s->one(e|e))	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[10]

p04_e17: one 2 collect

	USE	Eclipse MDT OCL	RoclET[1]	OCLE	Octupus	ATL Simple OCL Web Tester [4]
?let c = oclEmpty(Set(Integer)) in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	false[5]	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true[5]	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true[6]	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Set{1} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true

?let c = Bag{4} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true[6]	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Set{1, 4, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 4, 5, 4} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 4, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 4, 5, 4} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 4, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{1, 2, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	false	Boolean= true	true	true
?let c = Bag{1, 2, 5, 1} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	false	Boolean= true	true	true
?let c = Bag{1, 2, 5, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	false	Boolean= true	true	true
?let c = Bag{1, 2, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	false	Boolean= true	true	true
?let c = Sequence{1, 2, 5, 1} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	false	Boolean= true	true	true
?let c = Sequence{1, 2, 5, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	false	Boolean= true	true	true
?let c = Sequence{1, 2, 5} in c->one(i|i<4) = (c->collect(i|i<4)->count(true) = 1)	true : Boolean	true	false	Boolean= true	true	true

Anmerkungen

1. ↑ Ausdrücke immer typisiert: let c:Set(Integer)|Bag(Integer)|Sequence(Integer) ...
2. ↑ oclEmpty(Set(Integer)), oclEmpty(Bag(Integer)), oclEmpty(Sequence(Integer)) durch Set{}, Bag{},Sequence{} ersetzen
3. ↑ Octopus: Statt ColType{oclUndefined(Integer)} wird ColType{ColType{1}->any(false)}. Beispiel: Set{oclUndefined(Integer)}= Set{Set{1}->any(false)}
4. ↑ statt oclEmpty(Set(Integer)), oclEmpty(Bag(Integer)), oclEmpty(Sequence(Integer)) wird durch Set{},Bag{},Sequence{} ersetzen

5. ↑ oclEmpty(Set(Integer)), oclEmpty(Bag(Integer)), oclEmpty(Sequence(Integer)) durch Set{1}->reject(i|i=1), Bag{1}->reject(i|i=1),Sequence{1}->reject(i|i=1) ersetzen
6. ↑ oclUndefined(Integer) durch Set{1}->any(false) ersetzen
7. ↑ Collections erlauben kein oclUndefined als Element, hier kann nicht typisiert
8. ↑ Der 'implies'-Ausdruck muss geklammert werden, ansonsten liefert Ausdruck 'Boolean= false'
9. ↑ c->collect(i|i<4)->asSet() wird eine leere Collection aufgerufen.
10. ↑ Syntax wird nicht erkannt, wenn diesen Ausdruck ohne "s->one(true)" zu bewerten, wird "true" ausliefern.
11. ↑ Syntax wird nicht erkannt, wenn diesen Ausdruck ohne "s->one(true)" zu bewerten , wird "false" ausliefern.

Von "https://muse.informatik.uni-bremen.de/wiki/index.php/OCL_Benchmark_4_-_Teil_3"

Kategorie: OCLBenchmark

OCL Benchmark 4 - Teil 4 (B5)

Teil 4 von OCL Benchmark 4 - Collection Operations

Inhaltsverzeichnis

p05_o01: Collect 2 Iterate

	USE	Eclipse MDT OCL	RoclET[1]	OCLE	Octopus [2] [3] [4]	ATL Simple OCL Web Tester [5]
?let c = oclEmpty(Set(Integer)) in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = oclEmpty(Bag(Integer)) in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = oclEmpty(Sequence(Integer)) in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(i*i))	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = oclEmpty(Sequence(Integer)) in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->append(i*i))	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Set{oclUndefined(Integer)} in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	true	Boolean= true	n/a [4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	true	Boolean= true	n/a [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->append(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	wird nicht ausgewertet[7]

?let c = Set{2} in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	ture
?let c = Bag{2} in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	true
?let c = Sequence{2} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	true
?let c = Sequence{2} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->append(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	true
?let c = Set{oclUndefined(Integer), 0, 1, 2} in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 0, 1, 2, oclUndefined(Integer)} in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 0, 1, 2} in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 0, 1, 2, oclUndefined(Integer)} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 0, 1, 2} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 0, 1, 2, oclUndefined(Integer)} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->append(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 0, 1, 2} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->append(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	wird nicht ausgewertet[7]
?let c = Set{-1, 0, 1, 2} in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	true
?let c = Bag{-1, 0, 1, 2, 1} in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	true
?let c = Bag{-1, 0, 1, 2} in c->collect(i|i*i) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	true
?let c = Sequence{-1, 0, 1, 2, 1} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	true
?let c = Sequence{-1, 0, 1, 2} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	true
?let c = Sequence{-1, 0, 1, 2, 1} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->append(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	true
?let c = Sequence{-1, 0, 1, 2} in c->collect(i|i*i) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->append(i*i))	true : Boolean	true	false	Boolean= true	n/a [4]	true

p05_o02: Exists 2 Iterate

	USE	Eclipse MDT OCL	RoclET	OCLE	Octopus [2]	ATL Simple OCL Web Tester [5]

					[3] [8]	
?let c = oclEmpty(Set(Integer)) in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Set{1} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	N/A	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	N/A	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Set{1, 2, 3, 4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 4, 1} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true

?let c = Sequence{1, 2, 3, 4, 1} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4, 5, 6} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4, 5, 6, 4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4, 5, 6} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4, 5, 6, 4} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{4, 5, 6} in c->exists(i|i<4) = c->iterate(i;r:Boolean = false|r or i<4)	true : Boolean	true	true	Boolean= true	true	true

p05_o03: ForAll 2 Iterate

	USE	**Eclipse MDT OCL**	**RoclET**	**OCLE**	**Octopus[2] [3] [8]**	**ATL Simple OCL Web Tester[5]**
?let c = oclEmpty(Set(Integer)) in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = oclEmpty(Bag(Integer)) in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = oclEmpty(Sequence(Integer)) in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Set{1} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{4} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{4} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true

?let c = Sequence{4} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7]
?let c = Set{1, 2, 3} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3, 1} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{1, 2, 3} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3, 1} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{1, 2, 3} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{2, 3, 4, 5} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 4} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5, 2} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Bag{2, 3, 4, 5} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 4} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5, 2} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true
?let c = Sequence{2, 3, 4, 5} in c->forAll(i|i<4) = c->iterate(i;r:Boolean = true|r and i<4)	true : Boolean	true	true	Boolean= true	true	true

p05_o04: One 2 Iterate

	USE	**Eclipse MDT OCL**	**RoclET**	**OCLE**	**Octopus** [2] [3] [8] [4]	**ATL Simple OCL Web Tester** [5]

?let c = oclEmpty(Set(Integer)) in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = oclEmpty(Bag(Integer)) in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = oclEmpty(Sequence(Integer)) in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Set{oclUndefined(Integer)} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= Undefined	true[6]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	N/A	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Set{1} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{1} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{1} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Set{4} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{4} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{4} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Set{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	N/A	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]

?let c = Sequence{oclUndefined(Integer), 1, 4, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	N/A	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Set{1, 4, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{1, 4, 5, 4} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{1, 4, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{1, 4, 5, 4} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{1, 4, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Set{1, 2, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{1, 2, 5, 1} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{1, 2, 5, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{1, 2, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{1, 2, 5, 1} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{1, 2, 5, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{1, 2, 5} in c->one(i|i<4) = c->iterate(i;r:Sequence(Boolean)=Sequence{false,false}|if r->first() then Sequence{true,false} else Sequence{r->last() and i<4, (r->last() and i<4) xor (r->last() or i<4)} endif)->last()	true : Boolean	true	true	Boolean= true	N/A [4]	true

p05_o05: Reject 2 Iterate

	USE	**Eclipse MDT OCL**	**RoclET[1]**	**OCLE**	**Octopus [2] [3] [8] [4]**	**ATL Simple OCL Web Tester [5]**
?let c = oclEmpty(Set(Integer)) in c->reject(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	true[6]	true

?let c = oclEmpty(Bag(Integer)) in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = oclEmpty(Sequence(Integer)) in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Set{oclUndefined(Integer)} in c->reject(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= Undefined	true[6]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= Undefined	true[6]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	N/A	Boolean= Undefined	true[6]	wird nicht ausgewertet[7]
?let c = Set{1} in c->reject(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Bag{1} in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Sequence{1} in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Set{4} in c->reject(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Bag{4} in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Sequence{4} in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Set{oclUndefined(Integer), 1, 2, 3} in c->reject(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= Undefined	N/A[4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= Undefined	N/A[4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 2, 3} in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= Undefined	N/A[4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	N/A	Boolean= Undefined	N/A[4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	N/A	Boolean= Undefined	N/A[4]	wird nicht ausgewertet[7]
?let c = Set{1, 2, 3} in c->reject(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Bag{1, 2, 3, 1} in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Bag{1, 2, 3} in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Sequence{1, 2, 3, 1} in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true

?let c = Sequence{1, 2, 3} in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Set{2, 3, 4, 5} in c->reject(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Bag{2, 3, 4, 5, 4} in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Bag{2, 3, 4, 5, 2} in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Bag{2, 3, 4, 5} in c->reject(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Sequence{2, 3, 4, 5, 4} in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Sequence{2, 3, 4, 5, 2} in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true
?let c = Sequence{2, 3, 4, 5} in c->reject(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r else r->including(i) endif)	true : Boolean	true	true	Boolean= true	N/A[4]	true

p05_o06: Select 2 Iterate

	USE	**Eclipse MDT OCL**	**RoclET**	**OCLE**	**Octopus [2] [3] [8] [4]**	**ATL Simple OCL Web Tester [5]**
?let c = oclEmpty(Set(Integer)) in c->select(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = oclEmpty(Bag(Integer)) in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = oclEmpty(Sequence(Integer)) in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Set{oclUndefined(Integer)} in c->select(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= Undefined	true[6]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	N/A	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Set{1} in c->select(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{1} in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{1} in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true

?let c = Set{4} in c->select(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Bag{4} in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Sequence{4} in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	true[6]	true
?let c = Set{oclUndefined(Integer), 1, 2, 3} in c->select(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 1, 2, 3} in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	N/A	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	N/A	Boolean= Undefined	N/A [4]	wird nicht ausgewertet[7]
?let c = Set{1, 2, 3} in c->select(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{1, 2, 3, 1} in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{1, 2, 3} in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{1, 2, 3, 1} in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{1, 2, 3} in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Set{2, 3, 4, 5} in c->select(i|i<4) = c->iterate(i;r:Set(Integer) = oclEmpty(Set(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{2, 3, 4, 5, 4} in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{2, 3, 4, 5, 2} in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Bag{2, 3, 4, 5} in c->select(i|i<4) = c->iterate(i;r:Bag(Integer) = oclEmpty(Bag(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{2, 3, 4, 5, 4} in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{2, 3, 4, 5, 2} in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true
?let c = Sequence{2, 3, 4, 5} in c->select(i|i<4) = c->iterate(i;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|if i<4 then r->including(i) else r endif)	true : Boolean	true	true	Boolean= true	N/A [4]	true

Anmerkungen

1. ↑ Kein Iterate mit 2 Iteratorvariablen
2. ↑ Ausdrücke immer typisiert: let c:Set(Integer)|Bag(Integer)|Sequence(Integer) ...
3. ↑ Statt oclEmpty(...) wird c:ColType(Type) = ColType{} verwendet. Beispiel: c:Set(Integer) = Set{}
4. ↑ Octopus Parser hat die objektwertige Variable als primitive parameter in Java Code transformiert.Es verursacht syntax Fehler in Java Code. Der fehlerhafter Code könnt nicht richtig ausgeführt werden.(z.B. (objekt)Integer vs.(primitive Typ)int,(objekt)Boolean vs. (primitive Typ)boolean)
5. ↑ statt oclEmpty(Set(Integer)), oclEmpty(Bag(Integer)), oclEmpty(Sequence(Integer)) wird durch Set{},Bag{},Sequence{} ersetzen
6. ↑ Manchmal hat der Ausdruck das Problem (objektwertige Variable vs. primitive parameter),in den generierten Javacode taucht auch Fehler auf. Aber der Code könnt ausgeführt wird,weil die fehlerhafter Code wird nicht völlständig aufgerufen.
7. ↑ Collections erlauben kein oclUndefined als Element, hier kann nicht typisiert
8. ↑ Statt ColType{oclUndefined(Integer)} wird ColType{ColType{1}->any(false)}. Beispiel: Set{oclUndefined(Integer)}= Set{Set{1}->any(false)}

Von "https://muse.informatik.uni-bremen.de/wiki/index.php/OCL_Benchmark_4_-_Teil_4"

Kategorie: OCLBenchmark

OCL Benchmark 4 - Teil 5 (B6)

Teil 5 von OCL Benchmark 4 - Collection Operations

p09_e02: iterate 2 iterate let

	USE	Eclipse MDT OCL	RoclET	OCLE	Octopus [1] [2] [3] [4]	ATL Simple OCL Web Tester [5]
?let c = oclEmpty(Set(Integer)) in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	true[6]	true
?let c = oclEmpty(Bag(Integer)) in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	true[6]	true
?let c = oclEmpty(Sequence(Integer)) in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	true[6]	true
?let c = Set{oclUndefined(Integer)} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer)} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer)} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	wird nicht ausgewertet[7]
?let c = Set{2} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	true
?let c = Bag{2} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	true

?let c = Sequence{2} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	true
?let c = Set{oclUndefined(Integer), 2, 3, -10} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 2, 3, -10, oclUndefined(Integer)} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	wird nicht ausgewertet[7]
?let c = Bag{oclUndefined(Integer), 2, 3, -10} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 2, 3, -10, oclUndefined(Integer)} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	wird nicht ausgewertet[7]
?let c = Sequence{oclUndefined(Integer), 2, 3, -10} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	wird nicht ausgewertet[7]
?let c = Set{1, 2, 3, -10} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	true
?let c = Bag{1, 2, 3, -10, 1} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	true
?let c = Bag{1, 2, 3, -10} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	true
?let c = Sequence{1, 2, 3, -10, 1} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	true
?let c = Sequence{1, 2, 3, -10} in c->iterate(elem;r:Bag(Integer) = Bag{1}|r->including(r->size()+elem)) = c->iterate(elem;r:Bag(Integer) = Bag{1}|let res = r->including(r->size()+elem) in res)	true : Boolean	true	N/A	Boolean= true	N/A [4]	true

p09_e03: singleton 2 value

	USE	**Eclipse MDT OCL**	**RoclET[8]**	**OCLE**	**Octopus [9] [1] [2] [3]**	**ATL Simple OCL Web Tester [5]**
?let c = oclEmpty(Set(Integer)) in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = oclEmpty(Set(Integer)) in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]

?let c = oclEmpty(Bag(Integer)) in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = oclEmpty(Bag(Integer)) in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]
?let c = oclEmpty(Sequence(Integer)) in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = oclEmpty(Sequence(Integer)) in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Set{oclUndefined(Integer)} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = Set{oclUndefined(Integer)} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Bag{oclUndefined(Integer)} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = Bag{oclUndefined(Integer)} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Sequence{oclUndefined(Integer)} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = Sequence{oclUndefined(Integer)} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Set{2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[7][10]
?let c = Set{2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= true	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Bag{2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[7][10]
?let c = Bag{2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= true	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Sequence{2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= true	true	wird nicht ausgewertet[7][10]
?let c = Sequence{2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= true	N/A[9]	wird nicht ausgewertet[7][10]

?let c = Set{oclUndefined(Integer), 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = Set{oclUndefined(Integer), 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Bag{oclUndefined(Integer), 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	true	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = Bag{oclUndefined(Integer), 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Sequence{oclUndefined(Integer), 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = Sequence{oclUndefined(Integer), 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Set{1, 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = Set{1, 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Bag{1, 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = Bag{1, 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]
?let c = Sequence{1, 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = if c->size() = 1 then c->iterate(elem;r:Integer = oclUndefined(Integer)|elem) else oclUndefined(Integer) endif	true : Boolean	true	N/A	Boolean= Undefined	true	wird nicht ausgewertet[7][10]
?let c = Sequence{1, 2} in if c->size() = 1 then c->any(true) else oclUndefined(Integer) endif = c->iterate(elem;r:Sequence(OclAny) = Sequence{oclUndefined(Integer),false}|if r->at(2) = false then Sequence{elem,true} else Sequence{oclUndefined(Integer),true} endif)->at(1)	true : Boolean	true	Type mismatch. No common supertype: (Integer), (Boolean)	Boolean= Undefined	N/A[9]	wird nicht ausgewertet[7][10]

p10_e01: conversion 2 sequence

Durchgestrichen sind Ergebniss, die nicht stimmen können. Die Befehle haben keine zwei Iterationsvariablen.

	USE	**Eclipse MDT OCL**	**RoclET**	**OCLE**	**Octopus [1] [2] [3] [11]**	**ATL Simple OCL Web Tester [5]**
?let c = oclEmpty(Set(Integer)) in c->asSequence() = c->asBag()->asSequence()	true : Boolean	true	true	Boolean= true	true	true

?let c = oclEmpty(Set(Integer)) in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))	true : Boolean	true	true	Boolean= true	true [6]	true
?let c = oclEmpty(Set(Integer)) in c->asSequence() = c->iterate(u;r:Tuple(theSet:Set(Integer),theSeq:Sequence(Integer)) = Tuple{theSet:c,theSeq:oclEmpty(Sequence(Integer))}|let e = r.theSet->any(true) in Tuple{theSet:r.theSet->excluding(e),theSeq:r.theSeq->including(e)}).theSeq	true : Boolean	false[2]	Syntaxfehler		N/A[11]	wird nicht ausgewertet[10]
?let c = oclEmpty(Set(Integer)) in c->asSequence() = Sequence{c}->flatten()	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Set{oclUndefined(Integer)} in c->asSequence() = c->asBag()->asSequence()	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Set{oclUndefined(Integer)} in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))	true : Boolean	true	true	Boolean= true	N/A[11]	wird nicht ausgewertet[7][10]
?let c = Set{oclUndefined(Integer)} in c->asSequence() = c->iterate(u;r:Tuple(theSet:Set(Integer),theSeq:Sequence(Integer)) = Tuple{theSet:c,theSeq:oclEmpty(Sequence(Integer))}|let e = r.theSet->any(true) in Tuple{theSet:r.theSet->excluding(e),theSeq:r.theSeq->including(e)}).theSeq	true : Boolean	false[2]	Syntaxfehler		N/A[11]	wird nicht ausgewertet[7][10]
?let c = Set{oclUndefined(Integer)} in c->asSequence() = Sequence{c}->flatten()	true : Boolean	true	N/A	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Set{1} in c->asSequence() = c->asBag()->asSequence()	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{1} in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))	true : Boolean	true	true	Boolean= true	N/A [4]	wird nicht ausgewertet[10]
?let c = Set{1} in c->asSequence() = c->iterate(u;r:Tuple(theSet:Set(Integer),theSeq:Sequence(Integer)) = Tuple{theSet:c,theSeq:oclEmpty(Sequence(Integer))}|let e = r.theSet->any(true) in Tuple{theSet:r.theSet->excluding(e),theSeq:r.theSeq->including(e)}).theSeq	true : Boolean	false[2]	Syntaxfehler		N/A [4]	wird nicht ausgewertet[10]
?let c = Set{1} in c->asSequence() = Sequence{c}->flatten()	true : Boolean	true	N/A	Boolean= true	true	true
?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->asSequence() = c->asBag()->asSequence()	true : Boolean	true	true	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))	true : Boolean	true	true	Boolean= true	N/A [4]	wird nicht ausgewertet[7]
?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->asSequence() = c->iterate(u;r:Tuple(theSet:Set(Integer),theSeq:Sequence(Integer)) = Tuple{theSet:c,theSeq:oclEmpty(Sequence(Integer))}|let e = r.theSet->any(true) in Tuple{theSet:r.theSet->excluding(e),theSeq:r.theSeq->including(e)}).theSeq	true : Boolean	false[2]	Syntaxfehler		N/A [4]	wird nicht ausgewertet[7][10]
?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->asSequence() = Sequence{c}->flatten()	true : Boolean	true	N/A	Boolean= true	true	wird nicht ausgewertet[7]
?let c = Set{1, -1, 3, 2, 0} in c->asSequence() = c->asBag()->asSequence()	true : Boolean	true	true	Boolean= true	true	true
?let c = Set{1, -1, 3, 2, 0} in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))	true : Boolean	true	true	Boolean= true	N/A [4]	false[12]
?let c = Set{1, -1, 3, 2, 0} in c->asSequence() = c->iterate(u;r:Tuple(theSet:Set(Integer),theSeq:Sequence(Integer)) = Tuple{theSet:c,theSeq:oclEmpty(Sequence(Integer))}|let e = r.theSet->any(true) in Tuple{theSet:r.theSet->excluding(e),theSeq:r.theSeq->including(e)}).theSeq	true : Boolean	false[2]	Syntaxfehler		N/A [4]	wird nicht ausgewertet[10]
?let c = Set{1, -1, 3, 2, 0} in c->asSequence() = Sequence{c}->flatten()	true : Boolean	true	N/A	Boolean= true	true	true

Anmerkungen

1. ↑ Ausdrücke immer typisiert: let c:Set(Integer)|Bag(Integer)|Sequence(Integer) ...
2. ↑ Statt oclEmpty(...) wird c:ColType(Type) = ColType{} verwendet. Beispiel: c:Set(Integer) = Set{}
3. ↑ Statt ColType{oclUndefined(Integer)} wird ColType{ColType{1}->any(false)}. Beispiel: Set{oclUndefined(Integer)}= Set{Set{1}->any(false)}
4. ↑ Octopus Parser hat die objektwertige Variable als primitive parameter in Java Code transformiert.Es verursacht syntax Fehler in Java Code. Die fehlerhaft Code könnt nicht richtig ausgeführt werden.(z.B. (objekt)Integer vs.(primitive Typ)int,(objekt)Boolean vs. (primitive Typ)boolean)
5. ↑ statt oclEmpty(Set(Integer)), oclEmpty(Bag(Integer)), oclEmpty(Sequence(Integer)) wird durch Set{},Bag{},Sequence{} ersetzen
6. ↑ Manchmal der Ausdruck hat das Problem (objektwertige Variable vs. primitive parameter) und in den Generierung des Javacode taucht es auch Fehler auf. Aber der Code könnt ausgeführt werden,weil der fehlerhafter Code nicht aufgerufen wird.
7. ↑ Collections erlauben kein oclUndefined als Element, hier kann nicht typisiert
8. ↑ Kein Iterate mit 2 Iteratorvariablen
9. ↑ Type mismatch.All parts of a collection literal should have the same (super)type.
10. ↑ Syntax wird nicht erkannt
11. ↑ Der Parser von Octopus kennen Tuple Typ nicht.
12. ↑ es wird "let c : Set = Set {2, 1, 3, 0, - 1} in c->asSequence() = c->iterate(elem; r : Sequence = Sequence {} | r->including(elem))" bewertet.

Von "https://muse.informatik.uni-bremen.de/wiki/index.php/OCL_Benchmark_4_-_Teil_5"

Kategorie: OCLBenchmark

OCL Benchmark 5 - Determinateness (B6)

Benchmark 5 des OCL Benchmark

Ergebnisse der Werkzeuge

	Ausdrücke	USE	Eclipse MDT OCL	RoclET	Octopus [1] [2]	ATL Simple OCL Web Tester [3]	OCLE
1	**?let c = oclEmpty(Set(Integer)) in c->any(true) = c->asBag()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= Undefined
2	**?let c = oclEmpty(Set(Integer)) in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= Undefined
3	**?let c = oclEmpty(Set(Integer)) in c->any(true) = c->asBag()->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= Undefined
4	**?let c = oclEmpty(Set(Integer)) in c->any(true) = c->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= first() called for an empty sequence
5	**?let c = oclEmpty(Set(Integer)) in c->any(true) = c->asSequence()->last()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= last() called for an empty sequence
6	**?let c = oclEmpty(Set(Integer)) in c->any(true) = c->asBag()->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= first() called for an empty sequence
7	**?let c = oclEmpty(Set(Integer)) in c->any(true) = c->asBag()->asSequence()->last()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= last() called for an empty sequence
8	**?let c = oclEmpty(Set(Integer)) in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	N/A	true[2]	wird nicht ausgewertet[4]	Boolean= first() called for an empty sequence
9	**?let c = oclEmpty(Set(Integer)) in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	true	Boolean= true
10	**?let c = oclEmpty(Set(Integer)) in Sequence{c}->flatten() = c->asBag()->asSequence()**	true:Boolean	true	N/A	true	true	Boolean= true

11	**?let c = oclEmpty(Bag(Integer)) in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))**	true:Boolean	true	true	true[2]	true	Boolean= true
12	**?let c = oclEmpty(Bag(Integer)) in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= Undefined
13	**?let c = oclEmpty(Bag(Integer)) in c->any(true) = c->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= first() called for an empty sequence
14	**?let c = oclEmpty(Bag(Integer)) in c->any(true) = c->asSequence()->last()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= last() called for an empty sequence
15	**?let c = oclEmpty(Bag(Integer)) in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	N/A	true[2]	wird nicht ausgewertet[4]	Boolean= first() called for an empty sequence
16	**?let c = oclEmpty(Bag(Integer)) in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	true	Boolean= Undefined
17	**?let c = Set{oclUndefined(Integer)} in c->any(true) = c->asBag()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
18	**?let c = Set{oclUndefined(Integer)} in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
19	**?let c = Set{oclUndefined(Integer)} in c->any(true) = c->asBag()->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
20	**?let c = Set{oclUndefined(Integer)} in c->any(true) = c->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
21	**?let c = Set{oclUndefined(Integer)} in c->any(true) = c->asSequence()->last()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
22	**?let c = Set{oclUndefined(Integer)} in c->any(true) = c->asBag()->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
23	**?let c = Set{oclUndefined(Integer)} in c->any(true) = c->asBag()->asSequence()->last()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
24	**?let c = Set{oclUndefined(Integer)} in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	N/A	N/A[1]	wird nicht ausgewertet[4][5]	Boolean= Undefined
25	**?let c = Set{oclUndefined(Integer)} in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[5]	Boolean= true
26	**?let c = Set{oclUndefined(Integer)} in Sequence{c}->flatten() = c->asBag()->asSequence()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[5]	Boolean= true
27	**?let c = Bag{oclUndefined(Integer)} in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))**	true:Boolean	true	true	N/A[1]	wird nicht ausgewertet[5]	Boolean= Undefined
28	**?let c = Bag{oclUndefined(Integer)} in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
29	**?let c = Bag{oclUndefined(Integer)} in c->any(true) = c->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
30	**?let c = Bag{oclUndefined(Integer)} in c->any(true) = c->asSequence()->last()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined

31	**?let c = Bag{oclUndefined(Integer)} in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	N/A	N/A[1]	wird nicht ausgewertet[4][5]	Boolean= Undefined
32	**?let c = Bag{oclUndefined(Integer)} in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[5]	Boolean= true
33	**?let c = Set{1} in c->any(true) = c->asBag()->any(true)**	true:Boolean	true	true	true	wird nicht ausgewertet[4]	Boolean= true
34	**?let c = Set{1} in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	true	true	wird nicht ausgewertet[4]	Boolean= true
35	**?let c = Set{1} in c->any(true) = c->asBag()->asSequence()->any(true)**	true:Boolean	true	true	true	wird nicht ausgewertet[4]	Boolean= true
36	**?let c = Set{1} in c->any(true) = c->asSequence()->first()**	true:Boolean	true	true	true	wird nicht ausgewertet[4]	Boolean= true
37	**?let c = Set{1} in c->any(true) = c->asSequence()->last()**	true:Boolean	true	true	true	wird nicht ausgewertet[4]	Boolean= true
38	**?let c = Set{1} in c->any(true) = c->asBag()->asSequence()->first()**	true:Boolean	true	true	true	wird nicht ausgewertet[4]	Boolean= true
39	**?let c = Set{1} in c->any(true) = c->asBag()->asSequence()->last()**	true:Boolean	true	true	true	wird nicht ausgewertet[4]	Boolean= true
40	**?let c = Set{1} in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	true	N/A[1]	wird nicht ausgewertet[4]	Boolean= true
41	**?let c = Set{1} in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	true	Boolean= true
42	**?let c = Set{1} in Sequence{c}->flatten() = c->asBag()->asSequence()**	true:Boolean	true	N/A	true	true	Boolean= true
43	**?let c = Bag{1} in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))**	true:Boolean	true	false	N/A[1]	true	Boolean= true
44	**?let c = Bag{1} in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	true	true	wird nicht ausgewertet[4]	Boolean= true
45	**?let c = Bag{1} in c->any(true) = c->asSequence()->first()**	true:Boolean	true	true	true	wird nicht ausgewertet[4]	Boolean= true
46	**?let c = Bag{1} in c->any(true) = c->asSequence()->last()**	true:Boolean	true	true	true	wird nicht ausgewertet[4]	Boolean= true
47	**?let c = Bag{1} in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	true	N/A[1]	wird nicht ausgewertet[4]	Boolean= true
48	**?let c = Bag{1} in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	true	Boolean= true
49	**?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->asBag()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
50	**?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	N/A	N/A[1]	wird nicht ausgewertet[4][5]	Boolean= Undefined
51	**?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->asBag()->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined

52	**?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
53	**?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->last()**	false:Boolean	false	N/A	false	wird nicht ausgewertet[4][5]	Boolean= Undefined
54	**?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->asBag()->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
55	**?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->asBag()->asSequence()->last()**	false:Boolean	false	N/A	false	wird nicht ausgewertet[4][5]	Boolean= Undefined
56	**?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	N/A	N/A[1]	wird nicht ausgewertet[4][5]	Boolean= Undefined
57	**?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[5]	Boolean= true
58	**?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in Sequence{c}->flatten() = c->asBag()->asSequence()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[5]	Boolean= true
59	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0, oclUndefined(Integer)} in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))**	true:Boolean	true	false	N/A[1]	wird nicht ausgewertet[5]	Boolean= true
60	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0, oclUndefined(Integer)} in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
61	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0, oclUndefined(Integer)} in c->any(true) = c->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
62	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0, oclUndefined(Integer)} in c->any(true) = c->asSequence()->last()**	false:Boolean	false	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
63	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0, oclUndefined(Integer)} in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	N/A	N/A[1]	wird nicht ausgewertet[4][5]	Boolean= Undefined
64	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0, oclUndefined(Integer)} in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[5]	Boolean= true
65	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))**	true:Boolean	true	false	N/A[1]	wird nicht ausgewertet[5]	Boolean= Undefined
66	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
67	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
68	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->last()**	false:Boolean	false	N/A	true	wird nicht ausgewertet[4][5]	Boolean= Undefined
69	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0} in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	N/A	N/A[1]	wird nicht ausgewertet[4][5]	Boolean= Undefined
70	**?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0} in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4][5]	Boolean= true
71	**?let c = Set{1, -1, 3, 2, 0} in c->any(true) = c->asBag()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= true

72	**?let c = Set{1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= true
73	**?let c = Set{1, -1, 3, 2, 0} in c->any(true) = c->asBag()->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= true
74	**?let c = Set{1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= true
75	**?let c = Set{1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->last()**	false:Boolean	false	N/A	false		Boolean= false
76	**?let c = Set{1, -1, 3, 2, 0} in c->any(true) = c->asBag()->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= true
77	**?let c = Set{1, -1, 3, 2, 0} in c->any(true) = c->asBag()->asSequence()->last()**	false:Boolean	false	N/A	false		Boolean= false
78	**?let c = Set{1, -1, 3, 2, 0} in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	N/A	N/A[1]	wird nicht ausgewertet[4]	Boolean= true
79	**?let c = Set{1, -1, 3, 2, 0} in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	true	Boolean= true
80	**?let c = Set{1, -1, 3, 2, 0} in Sequence{c}->flatten() = c->asBag()->asSequence()**	true:Boolean	true	N/A	true	true	Boolean= true
81	**?let c = Bag{1, -1, 3, 2, 0, 1} in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))**	true:Boolean	true	false	N/A[1]	false	Boolean= true
82	**?let c = Bag{1, -1, 3, 2, 0, 1} in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= true
83	**?let c = Bag{1, -1, 3, 2, 0, 1} in c->any(true) = c->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= true
84	**?let c = Bag{1, -1, 3, 2, 0, 1} in c->any(true) = c->asSequence()->last()**	false:Boolean	false	N/A	true	wird nicht ausgewertet[4]	Boolean= true
85	**?let c = Bag{1, -1, 3, 2, 0, 1} in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	N/A	N/A[1]	wird nicht ausgewertet[4]	Boolean= true
86	**?let c = Bag{1, -1, 3, 2, 0, 1} in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	true	Boolean= true
87	**?let c = Bag{1, -1, 3, 2, 0} in c->asSequence() = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))**	true:Boolean	true	false	N/A[1]	false	Boolean= true
88	**?let c = Bag{1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->any(true)**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= true
89	**?let c = Bag{1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->first()**	true:Boolean	true	N/A	true	wird nicht ausgewertet[4]	Boolean= true
90	**?let c = Bag{1, -1, 3, 2, 0} in c->any(true) = c->asSequence()->last()**	false:Boolean	false	N/A	false	wird nicht ausgewertet[4]	Boolean= false
91	**?let c = Bag{1, -1, 3, 2, 0} in c->any(true) = c->iterate(elem;r:Sequence(Integer) = oclEmpty(Sequence(Integer))|r->including(elem))->first()**	true:Boolean	true	N/A	N/A[1]	wird nicht ausgewertet[4]	Boolean= true
92	**?let c = Bag{1, -1, 3, 2, 0} in Sequence{c}->flatten() = c->asSequence()**	true:Boolean	true	N/A	true	true	Boolean= true

Anmerkungen

1. ↑
   Octopus-Parser hat die objektwertige Variable als primitiven Parameter in Java Code transformiert. Es verursacht Syntaxfehler in Java Code. Der fehlerhafte Code konnte nicht richtig ausgeführt werden. (Z.B. (objekt)Integer vs. (primitiver Typ)int, (objekt)Boolean vs. (primitiver Typ)boolean)
2. ↑ Octopus: Manchmal der Ausdruck hat das Problem (objektwertige Variable vs. primitive parameter) und in den Generierung des Javacode taucht es auch Fehler auf. Aber der Code könnt ausgeführt werden,weil der fehlerhafter Code nicht aufgerufen wird.
3. ↑ ATL: statt oclEmpty(Set(Integer)), oclEmpty(Bag(Integer)), oclEmpty(Sequence(Integer)) wird durch Set{},Bag{},Sequence{} ersetzen
4. ↑ ATL: Syntax wird nicht erkannt
5. ↑ ATL: Collections erlauben kein oclUndefined als Element, hier kann nicht typisiert

Von "https://muse.informatik.uni-bremen.de/wiki/index.php/OCL_Benchmark_5_-_Determinateness"

Kategorie: OCLBenchmark