Issues and gimmicks around OCL's collection operation closure

- closure computes the transitive, reflexive closure of a relation in form of a fixpoint (f(a)=a) by applying an OCL closure expression iteratively until no more `new' solution elements are found

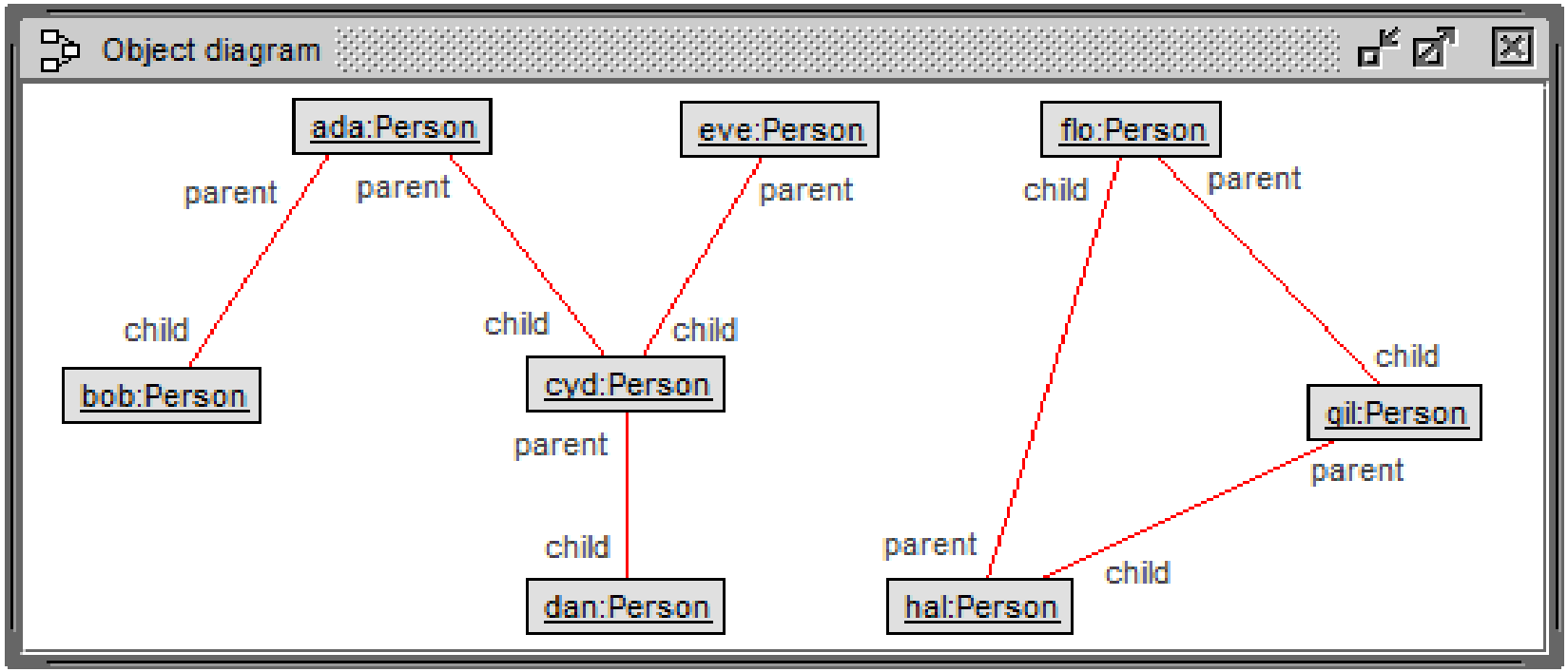
- in principle, closure can only be defined meaningful on Set(T) and Ord(T), but with some additional devices, closure is defined on Bag(T) and Seq(T) as well

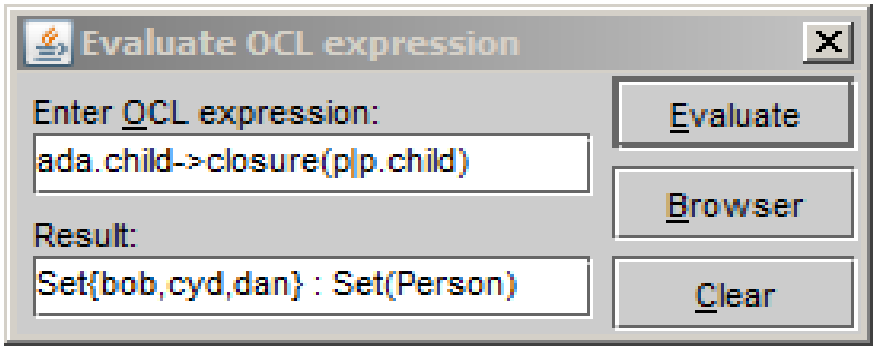
- the OCL closure expression can be collection-typed or single-value typed, however the result of a closure is always a collection of type Set(T) or Ord(T)

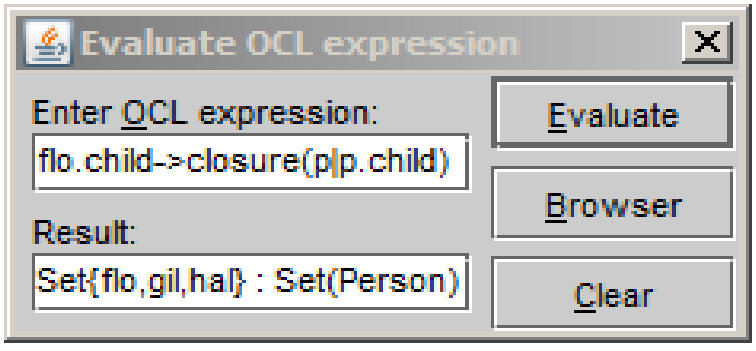
- for an argument of type Ord(T), the fixpoint is in general not unique and in order to achieve a unique solution, the solution, which has to be returned by an OCL evaluator, is required to be the depth-first preorder solution

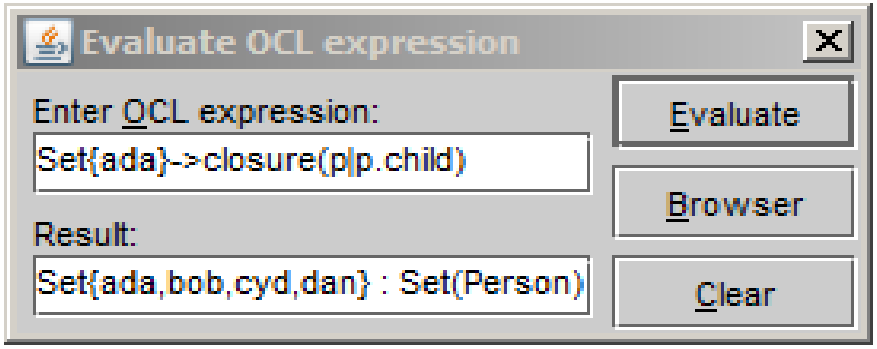
- closure has syntactic variations (as all other collection operations) with or without explicit variables and types for them

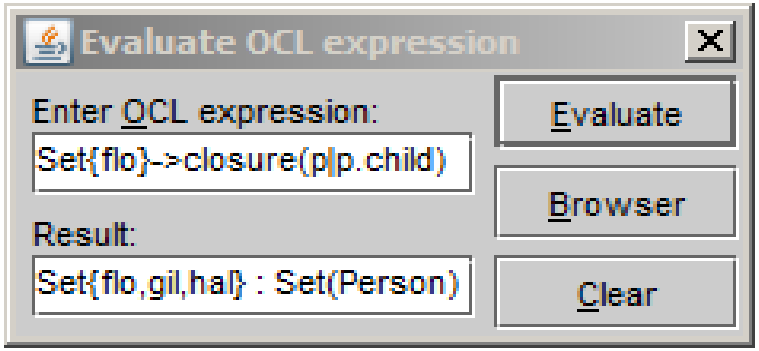
closure on Set(T)



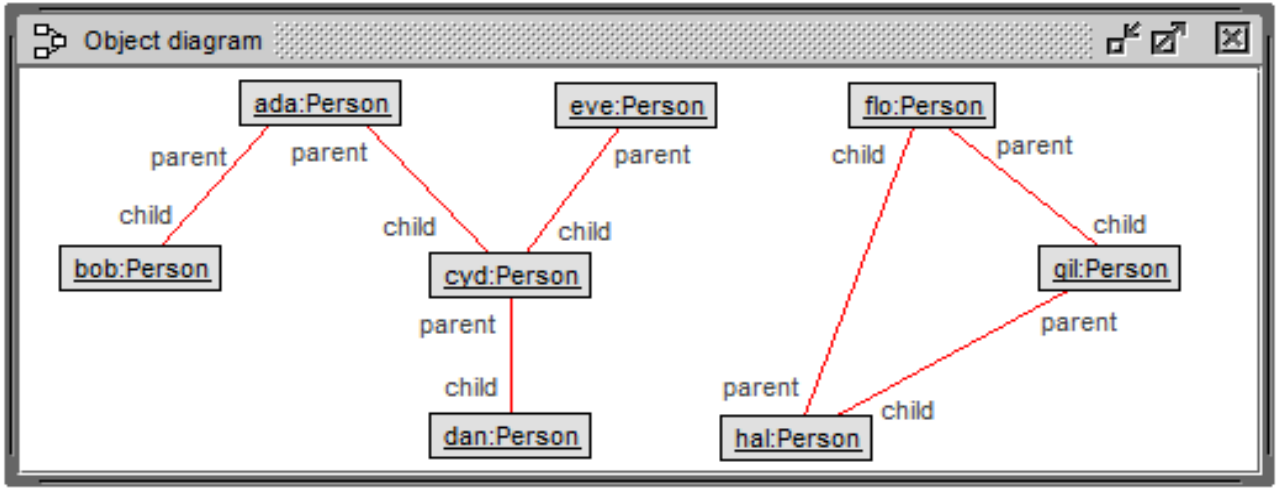








closure on OrderedSet(T)



Evaluate OCL expression

Enter OCL expression:
`OrderedSet{bob,cyd}->closure(p|p.child)`

Result:
`OrderedSet{bob,cyd,dan} : OrderedSet(Person)`

Evaluate | Browser | Clear

Evaluate OCL expression

Enter OCL expression:
`flo.child->asOrderedSet()->closure(p|p.child)`

Result:
`OrderedSet{gil,hal,flo} : OrderedSet(Person)`

Evaluate | Browser | Clear

Evaluate OCL expression

Enter OCL expression:
`OrderedSet{cyd,bob}->closure(p|p.child)`

Result:
`OrderedSet{cyd,bob,dan} : OrderedSet(Person)`

Evaluate | Browser | Clear

Evaluate OCL expression

Enter OCL expression:
`Set{flo}->asOrderedSet()->closure(p|p.child)`

Result:
`OrderedSet{flo,gil,hal} : OrderedSet(Person)`

Evaluate | Browser | Clear

Evaluate OCL expression

Enter OCL expression:
`Set{ada}->asOrderedSet()->closure(p|p.child)`

Result:
`OrderedSet{ada,bob,cyd,dan} : OrderedSet(Person)`

Evaluate | Browser | Clear

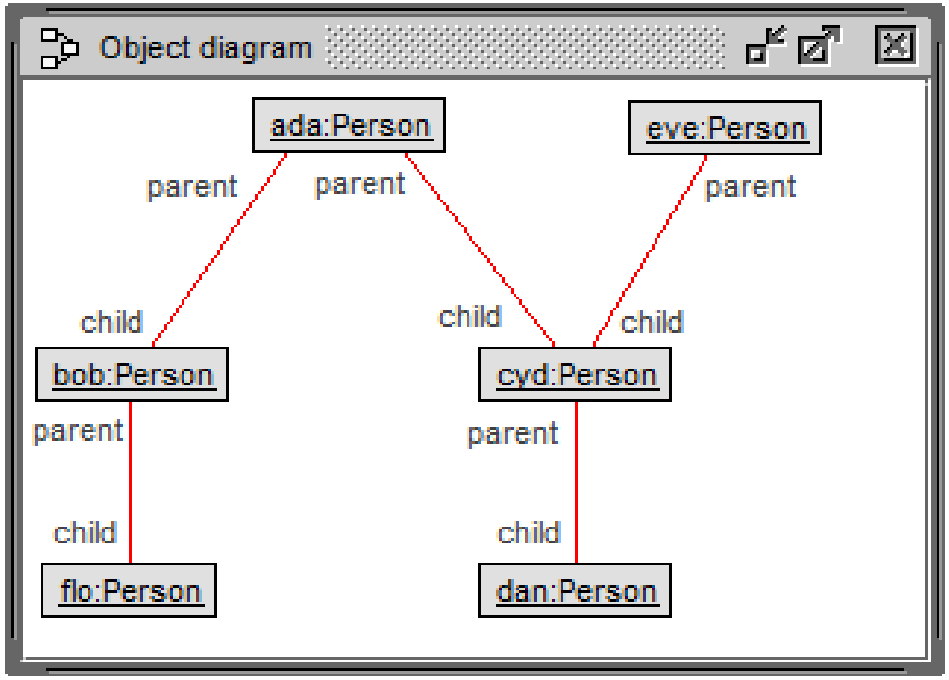
Evaluate OCL expression

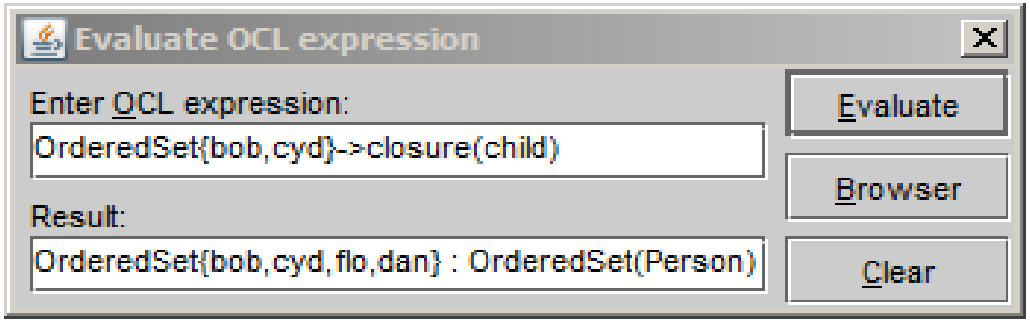
Enter OCL expression:
`Sequence{Set{bob,cyd}->asOrderedSet(),Set{cyd,bob}->asOrderedSet()}`

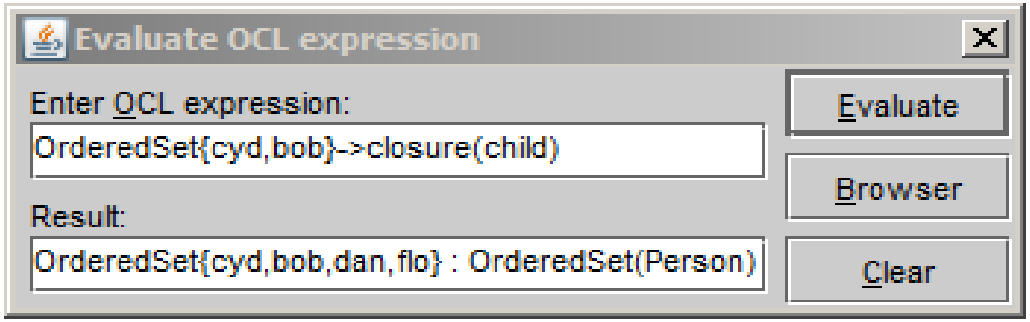
Result:
`Sequence{OrderedSet{bob,cyd},OrderedSet{bob,cyd}} : Sequence(OrderedSet(Person))`

Evaluate | Browser | Clear

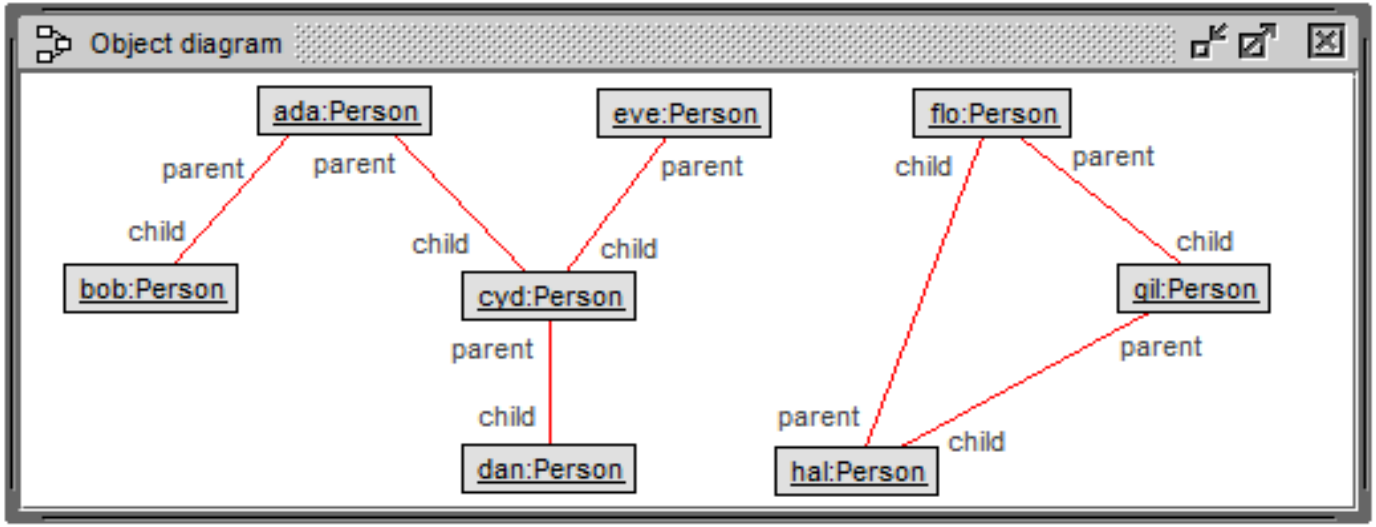
closure on OrderedSet(T)







closure on Bag(T)



Evaluate OCL expression

Enter OCL expression:
Bag{flo}->closure(p|p.child)
Result:
Set{flo,gil,hal} : Set(Person)
Evaluate
Browser
Clear

Evaluate OCL expression

Enter OCL expression:
Bag{flo}->union(flo.child)
Result:
Bag{flo,gil} : Bag(Person)
Evaluate
Browser
Clear

Evaluate OCL expression

Enter OCL expression:
Bag{flo}->union(flo.child)->union(flo.child.child)
Result:
Bag{flo,gil,hal} : Bag(Person)
Evaluate
Browser
Clear

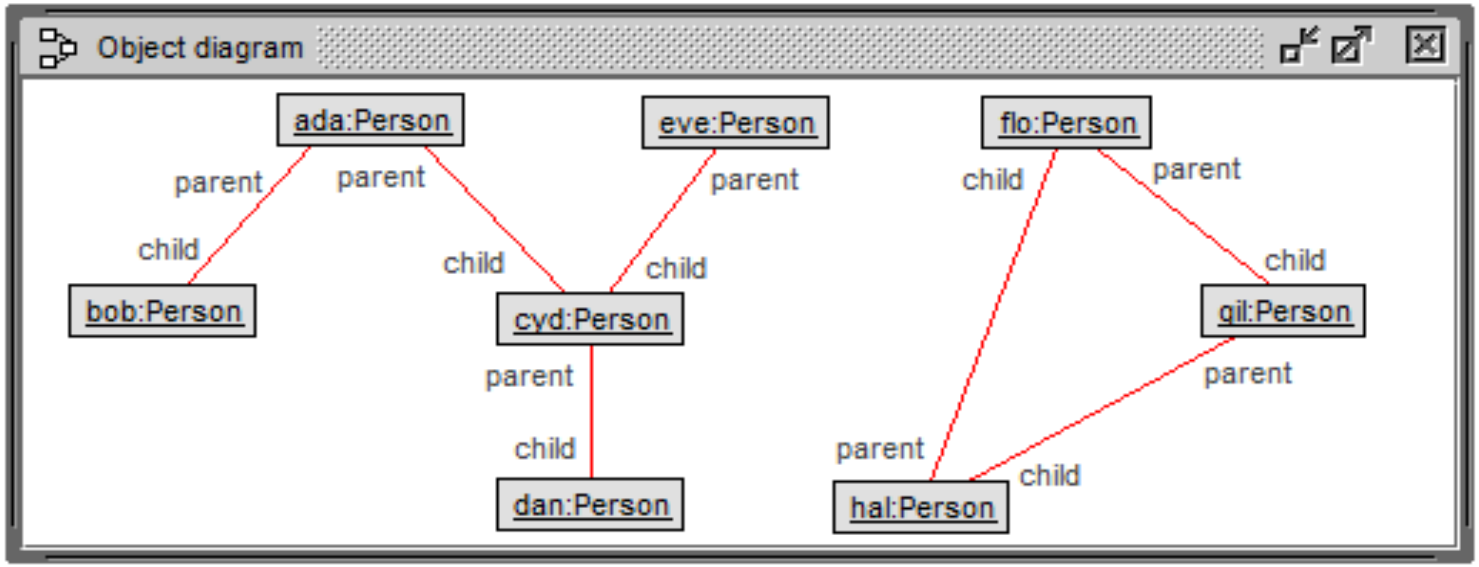
Evaluate OCL expression

Enter OCL expression:
Bag{flo}->union(flo.child)->union(flo.child.child)->union(flo.child.child.child)
Result:
Bag{flo,flo,gil,hal} : Bag(Person)
Evaluate
Browser
Clear

Evaluate OCL expression

Enter OCL expression:
Bag{flo}->union(flo.child)->union(flo.child.child)->union(flo.child.child.child)->union(flo.child.child.child.child)
Result:
Bag{flo,flo,gil,gil,hal} : Bag(Person)
Evaluate
Browser
Clear

closure on Sequence(T)



Evaluate OCL expression

Enter OCL expression:
`Sequence{bob,cyd}->closure(p|p.child)`

Result:
`OrderedSet{bob,cyd,dan} : OrderedSet(Person)`

Evaluate | Browser | Clear

Evaluate OCL expression

Enter OCL expression:
`flo.child->asSequence()->closure(p|p.child)`

Result:
`OrderedSet{gil,hal,flo} : OrderedSet(Person)`

Evaluate | Browser | Clear

Evaluate OCL expression

Enter OCL expression:
`Sequence{cyd,bob}->closure(p|p.child)`

Result:
`OrderedSet{cyd,bob,dan} : OrderedSet(Person)`

Evaluate | Browser | Clear

Evaluate OCL expression

Enter OCL expression:
`Set{flo}->asSequence()->closure(p|p.child)`

Result:
`OrderedSet{flo,gil,hal} : OrderedSet(Person)`

Evaluate | Browser | Clear

Evaluate OCL expression

Enter OCL expression:
`Sequence{ada}->closure(p|p.child)`

Result:
`OrderedSet{ada,bob,cyd,dan} : OrderedSet(Person)`

Evaluate | Browser | Clear

Evaluate OCL expression

Enter OCL expression:
`Sequence{ada,ada}->closure(p|p.child)`

Result:
`OrderedSet{ada,bob,cyd,dan} : OrderedSet(Person)`

Evaluate | Browser | Clear

Syntax and evaluation of closure

```
COLEXPR->closure(ELEMVAR | CLOSEXPR)

type[[ COLEXPR  ]] in {Set(T), Bag(T), Ord(T), Seq(T)}
type[[ CLOSEXPR ]] in {Set(T), Bag(T), Ord(T), Seq(T), T}
type[[ ELEMVAR  ]] = T
```

type[[COLEXPR]]	type[[COLEXPR->closure(ELEMVAR | CLOSEXPR)]]
Set(T),Bag(T)	Set(T) [COLEXPR:Bag(T) implicit typecast to Set(T)]
Ord(T),Seq(T)	Ord(T) [COLEXPR:Seq(T) implicit typecast to Ord(T)]

type[[COLEXPR]]	type[[CLOSEXPR]]	implicit typecast for CLOSEXPR
Set(T),Bag(T)	Set(T)	-
	Bag(T)	asSet()
	Ord(T)	asSet()
	Seq(T)	asSet()
	T	-
Ord(T),Seq(T)	Set(T)	asOrd()
	Bag(T)	asOrd()
	Ord(T)	-
	Seq(T)	asOrd()
	T	-

Sufficient to define the result of COLEXPR->closure(ELEMVAR | CLOSEXPR)
for the following cases

COLEXPR	CLOSEXPR
Set(T)	Set(T)
Set(T)	T
Ord(T)	Ord(T)
Ord(T)	T

```
RES:=eval[[COLEXPR]]
NEXT:=eval[[CLOSEXPR]] -- has type Set(T) # Ord(T) # T
while eval[[(NEXT-RES # Set{NEXT}-RES)->notEmpty()]] do
  RES:=eval[[RES->union(NEXT) # RES->including(NEXT)]]
  NEXT:=eval[[CLOSEXPR]]
  od
```

In the case of COLEXPR:Ord(T), RES must be the depth-first preorder solution

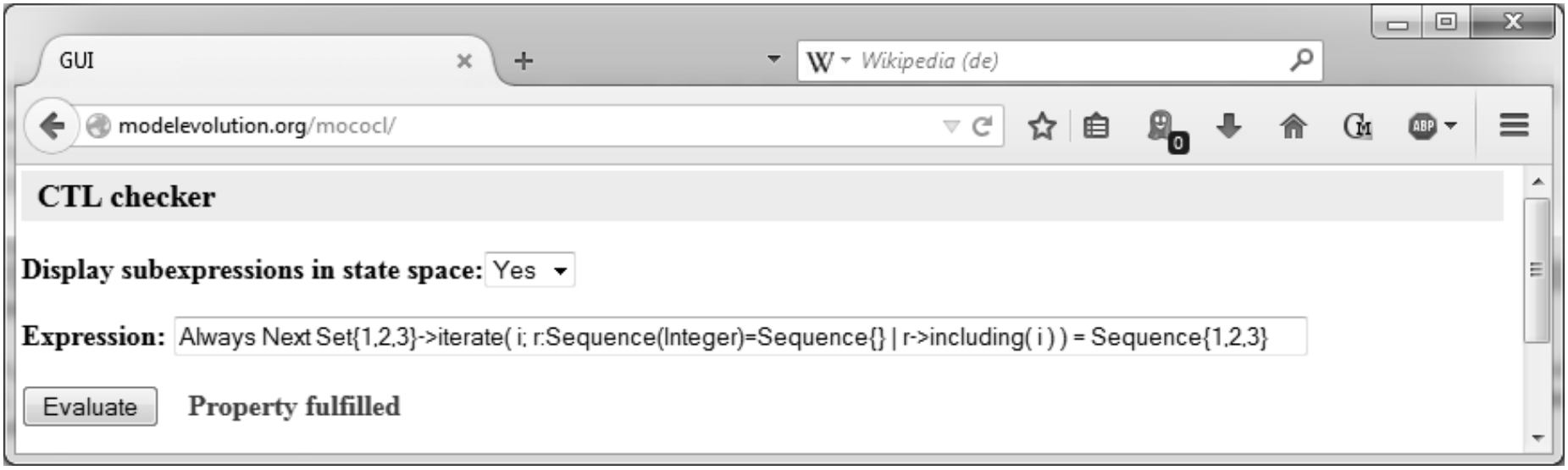
Variations: COLEXPR->closure(CLOSEXPR)
COLEXPR->closure(ELEMVAR:ELEMTYPE | CLOSEXPR)

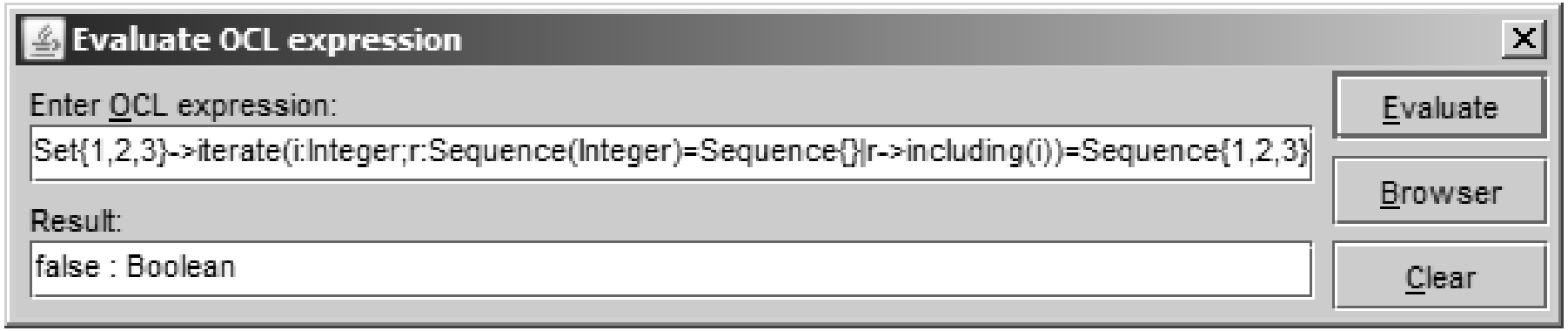
```
Examples: assume Person::child:Set(Person)
==========================================

use> ?ada.child->closure(p | p.child)
     Set{bob,cyd,dan} : Set(Person)

use> ?ada.child->asSet()->closure(p | p.child->asSequence())
     Set{bob,cyd,dan} : Set(Person)

use> ?ada.child->asBag()->closure(p | p.child->asSequence())
     Set{bob,cyd,dan} : Set(Person)

     COLEXPR->closure(E | CLOSEXPR) : Set(T)
        :                    :
      Bag(T)              Sequence(T)

use> ?ada.child->asOrderedSet()->closure(p | p.child->asSequence())
     OrderedSet{bob,cyd,dan} : OrderedSet(Person)

use> ?ada.child->asSequence()->closure(p | p.child->asSequence())
     OrderedSet{bob,cyd,dan} : OrderedSet(Person)
```

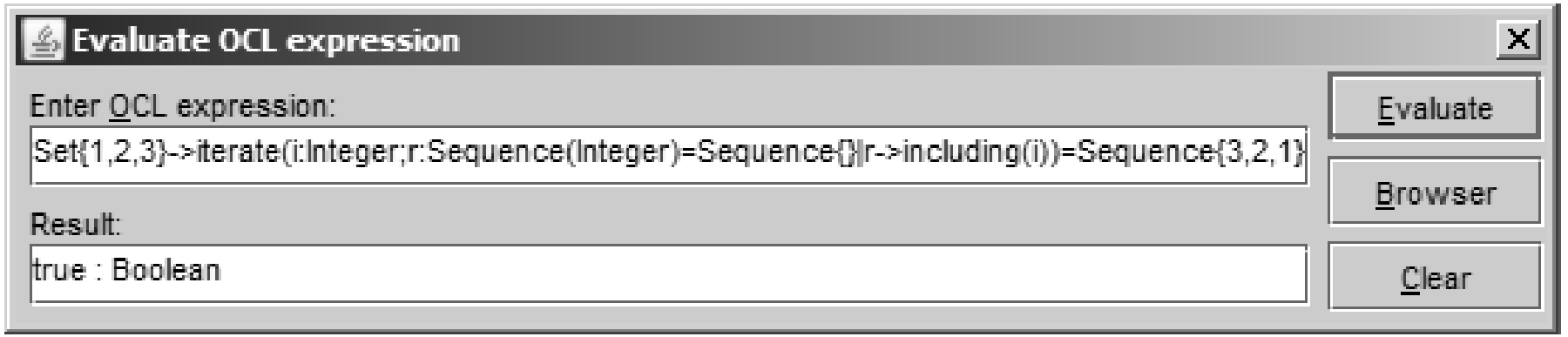
```
Operation closure with single-valued CLOSTERM
=============================================

class Person
end

association ParentChild between
  Person[0..1] role parent
  Person[0..*] role child
end

!create ada,bel,cyd,dan,eve,flo,gil:Person
!insert (ada,bel) into ParentChild
!insert (ada,cyd) into ParentChild
!insert (bel,dan) into ParentChild
!insert (bel,eve) into ParentChild
!insert (cyd,flo) into ParentChild
!insert (cyd,gil) into ParentChild

          ada
        /     \
      bel      cyd
     /   \    /   \
   dan eve flo gil

use> ?Set{dan}->closure(parent)
     Set{Undefined,ada,bel,dan} : Set(Person)

use> ?Set{dan.parent}->closure(parent)
     Set{Undefined,ada,bel} : Set(Person)
```

```
Depth-first preorder result for closure on OrderedSet
=====================================================

class Person
attributes
  child:OrderedSet(Person)
end

!create ada,bel,cyd,dan,eve,flo,gil:Person
!set ada.child:=OrderedSet{bel,cyd}   ...
!set dan.child:=OrderedSet{}          ...

          ada          preorder    ada bel dan eve cyd flo gil
        /     \
      bel      cyd     inorder     dan bel eve ada flo cyd gil
     /   \    /   \
    dan eve flo gil   postorder   dan eve bel flo gil cyd ada

use> ?OrderedSet{ada}->closure(child)
     OrderedSet{ada,bel,dan,eve,cyd,flo,gil} : OrderedSet(Person)
     -- depth-first preorder

use> ?OrderedSet{ada,bel,dan,eve,cyd,flo,gil}->closure(child)
     OrderedSet{ada,bel,dan,eve,cyd,flo,gil} : OrderedSet(Person)
     -- depth-first preorder

use> ?OrderedSet{ada,bel,cyd,dan,eve,flo,gil}->closure(child)
     OrderedSet{ada,bel,cyd,dan,eve,flo,gil} : OrderedSet(Person)
     -- breadth-first

use> ?OrderedSet{gil,flo,eve,dan,cyd,bel,ada}->closure(child)
     OrderedSet{gil,flo,eve,dan,cyd,bel,ada} : OrderedSet(Person)
     -- reverse breadth-first
```

Nachtrag zur Wohldefiniertheit von iterate







```
use> ?Set{'1','2','3'}->iterate(s:String;r:String=''|r+s)
     '321' : String

use> ?Set{'2','3','4'}->iterate(s:String;r:String=''|r+s)
     '324' : String

-- op(init,e) := init+e
-- op(op(init,e1),e2) =?= op(op(init,e2),e1)
-- init+e1+e2 =?= init+e2+e1

use> ?(('')+'1')+'2')
     '12' : String

use> ?(('')+'2')+'1')
     '21' : String

use> ?Set{'123','132','213','231','312','321'}->
       includes(Set{'1','2','3'}->iterate(s:String;r:String=''|r+s))
     true : Boolean
```

```
use> ?Bag{3,2,1}->iterate(i:Integer;r:Integer=0|r*10+i)
     321 : Integer

use> ?Bag{4,3,2}->iterate(i:Integer;r:Integer=0|r*10+i)
     324 : Integer

-- op(init,e) := init*10+e
-- op(op(init,e1),e2) =?= op(op(init,e2),e1)
-- ((init*10)+e1)*10+e2 =?= ((init*10)+e2)*10+e1


--             | commutative ITEREXPR | non-commutative ITEREXPR
-- -----------+----------------------+-------------------------
-- OrderedSet |           +          |              +
-- Sequence   |           +          |              +
-- Set        |           +          |              !
-- Bag        |           +          |              !

-- evaluation of iterate with non-commutative ITEREXPR
-- implementation-dependent within an OCL evaluator
-- for example: Set{'1','2','3'}->iterate(s:String;r:String=''|r+s) =
--              '123' or '132' or '213' or '231' or '312' or '321'

use> ?Set{1,2,3}->iterate(i:Integer;r:Integer=0|r+i)
     6 : Integer

-- op(init,e) := init+e
-- op(op(init,e1),e2) =?= op(op(init,e2),e1)
-- init+e1+e2 = init+e2+e1 => commutative OCL ITEREXPR
```