Checking mOdCL with a benchmark for OCL

Francisco Durán      Manuel Roldán

Abstract

The mOdCL tool is an OCL evaluator for Maude with different applications and which has been used in different contexts and projects. There are an increasing number of OCL engines and it is important to guarantee that their implementation of OCL conforms with the OCL specification. For this reason Kuhlmann et al. [8] proposed an OCL benchmark with different tests for checking the quality and conformance of different implementations of OCL engines with the OCL Specification. This report presents our experience after checking the mOdCL evaluator with the mentioned benchmark. All the proposed tests have been successfully executed, however we think that some test proposed in the benchmark contain mistakes or errors. We have identified such cases and they have been transformed before their execution with mOdCL.

1 Introduction

The Object Constraint Language (OCL) [10] is a textual formal language which provides the necessary formal notation to complete UML diagrams with precise and unambiguous specifications. The growing interest in OCL, mainly thanks to its use in Model-Driven Software Development (MDSD), has motivated the development of a number of tools supporting OCL. There are commercial tools as OCL MagicDraw or Together, open source tools as ATL [18] or Eclipse [9] and an increasing number of academic projects (see e.g, the OCL Portal [11]).

One of these tools is the mOdCL evaluator [3, 15] which allows the evaluation of OCL expressions in Maude [1] scope. It can be used with different purposes and it can be integrated in other Maude tools which represent UML models as object configurations. It has already been used to validate OCL constraints on given object diagrams, to dynamically validate such constraints during the execution of Maude prototypes of UML models [15], to analyze models using the Maude reachability tool [2] and from external tools where mOdCL is a black box component used to evaluate OCL expressions in general (it has been successfully integrated in e-Motions [19, 13] and Maudeling [16]).

The increasing number of tools with support for OCL motivated the proposal of a benchmark [8] with which to check the conformance of the different tools with the OCL Standard Specification [10]. This report presents our experience after checking the mOdCL evaluator with the benchmark. Although we provide a basic introduction to mOdCL and its general characteristics, this
description is not complete (see [4, 3] for detailed information about the design, implementation and use of mOdCL).

This document first introduces the basic characteristics of mOdCL and presents the benchmark, later it describes the process followed to test the mOdCL evaluator using expressions proposed in the benchmark and finally, it shows the results obtained. All the UML models and OCL expressions used in the benchmark are included in appendices as well as the corresponding mOdCL tests and results.

2 The mOdCL evaluator

The mOdCL evaluator is an interpreter of OCL for Maude [1] which allows the evaluation of OCL expressions on configurations of objects representing UML models. Expressions are evaluated by means of an eval function, which can be used with three different interfaces, depending on the context required to evaluate a given expression.

\[
\begin{align*}
\text{op eval : OclExp -> OclType .} \\
\text{op eval : OclExp Configuration -> OclType .} \\
\text{op eval : OclExp Configuration Configuration -> OclType .}
\end{align*}
\]

If the OCL expression is not referred to a given UML state the eval function only needs the expression to be evaluated. Otherwise, it refers to a given state and the eval function needs the object configuration representing such a state. Finally, those postcondition expressions using the @pre operator refer to two states (the actual state and the previous state referred by means of @pre) and therefore the eval function uses two objects configurations.

2.1 The representation of the UML model

The mOdCL evaluator assumes the representation of object oriented systems as usual in Maude. Thus, system states are represented as configurations of Maude objects (of sort Configuration), where objects are identified by elements of sort Oid. This is completed with the use of sorts specifically defined in mOdCL for declaring the name of attributes (AttributeName), the name of operations (OpName) and their arguments (Arg).

Each class in the UML class diagram is represented by a new sort, which should be named by using the name of the class in the diagram and must be subsort of the predefined sort Cid. The new sort must be used to declare a constant named with the name of the class. Furthermore, constants of sort AttributeName must be declared to represent each attribute or association. For example, given the following model CivilStatusWorld\(^1\), used in benchmarks B1 and B2.

\(^1\)Models are described here using the notation of the USE tool, as it is included in the benchmark [8].
model CivilStatusWorld

class Person
attributes
  name:String
  civstat:String
  gender:String
  alive:Boolean
operations
  spouse():Person=
    if self.gender='female' then self.husband else self.wife endif
end
association Marriage between
  Person [0..1] role wife
  Person [0..1] role husband
end

The following declarations must be provided and included in a Maude system module with the static structure of the UML model for representing the name of the class Person and its attributes:

sort Person .
subsort Person < Cid .
op Person : -> Person [ctor] .
op civstat : -> AttributeName [ctor] .
op gender : -> AttributeName [ctor] .
op alive : -> AttributeName [ctor] .

An operation \texttt{op (arg}_1: type_1, \ldots, arg}_n: type_n) : type is represented in mOdCL as a constant \texttt{op}, of sort \texttt{OpName} (of operation names), and constants arg_1, \ldots, arg_n, of sort \texttt{Arg} (of arguments). In this example we have a \texttt{spouse} operation without arguments, so that we would declare:

\texttt{op spouse : -> OpName} .

The body of the operation is an OCL expression which is written as the right hand of an equation whose left hand is the name of the operation (of sort \texttt{OpName}) and suitable arguments. For example, the operation \texttt{spouse} in the model \texttt{CivilStatusWorld} should be written in Maude as

\texttt{eq spouse() = if gender = "female" then husband else wife endif} .

Although this model does not define inheritance relationships among classes, if necessary, they should be represented through a predicate \texttt{isSubClass(A,B)}, which determines if a class \texttt{A} is a subclass of a class \texttt{B}.

\texttt{op isSubClass : Cid Cid -> Bool} .

For example, if the model would include a class \texttt{Worker} which is subclass of \texttt{Person}, then the following equation should be included in the Maude module.
eq isSubClass(Worker, Person) = true .

Association classes are represented and handled as any other class, defining a new sort for the class and declaring constants of sort AttributeName for their attributes and associations. For example, the TernExam model, used in the benchmark B3 includes the following association class Exam:

associationclass Exam between
    Person[0..*] role examinee
    Person[0..*] role examiner
    Person[0..*] role recorder
attributes
    date:Integer
end

which should be represented in mOdCL by including the following declarations in the Maude module which represents the model:

sort Exam .
subsort Exam < Cid .
op Exam : -> Person [ctor] .
op date : -> AttributeName [ctor] .
op examinee : -> AttributeName [ctor] .
op examiner : -> AttributeName [ctor] .
op recorder : -> AttributeName [ctor] .

Appendix A contains the complete definition of the model CivilStatusWorld for USE and mOdCL (benchmark B1). Appendix G contains more examples (MathLib and MapWorld) of UML models and their representation on USE and mOdCL.

2.2 The representation of a system state

As it was previously mentioned, a given system state must be represented by a Configuration of Maude objects. For example, in the model CivilStatusWorld, the following configuration represents the object diagram of Figure 1.
Figure 2: Object diagram representing a postcondition state in benchmark B2.

```plaintext
op state : -> Configuration.
eq state =
  < ada : Person | name : "Ada", gender : "female", alive : true,
  civstat : "widowed", wife : null, husband : null >
  < bob : Person | name : "Bob", gender : "male", alive : true,
  civstat : "divorced", wife : null, husband : null >
  < cyd : Person | name : "Cyd", gender : "male", alive : false,
  civstat : "married", wife : null, husband : null >
  < dan : Person | name : "Dan", gender : "male", alive : true,
  civstat : "married", wife : eve, husband : null >
  < eve : Person | name : "Eve", gender : "female", alive : true,
endm

Finally, if the configuration is used to evaluate pre- or post-condition expressions, it must include an env message to represent the object invoking the operation concerned (env(self <- object)) and an OpEnv message, containing pairs with the name and the actual value of the arguments used to invoke the operation (OpEnv(arg(name_1, value_1),...,arg(name_n, value_n)). For example, the invocation `ada.birth("Ada", female)` used in the benchmark B2, results in a system state represented by the object diagram of Figure 2. It would be represented by using the `ada Oid` as the value of `self` in the `env` message and by placing its arguments as the values of arguments `aName` and `aGender` in the `OpEnv` message.

```plaintext
eq state-A-post =
  < ada : Person | name : "Ada", civstat : single, gender : female,
  alive : true, wife : null, husband : null >
  env(self <- ada)
  OpEnv((arg(aName, "Ada"), arg(aGender, female))).

2.3 Lexical issues

OCL expressions must be written having into account minor lexical limitations in mOdCL, as follows:
```
• Tokens must be broken by spaces. Because of Maude lexical limitations it is necessary to include spaces to break any token in expressions. For example, the following expression

\[
\text{let } c = \text{Sequence}\{1, 2, 3\} \text{ in } c \rightarrow \text{reject}(i | i < 4) = c \rightarrow \text{select}(i | \text{not}(i < 4))
\]

must be written in mOdCL as

\[
\text{let } c = \text{Sequence}\{1, 2, 3\} \text{ in } c \rightarrow \text{reject}(i | i < 4) = c \rightarrow \text{select}(i | \text{not}(i < 4))
\]

• Values of type \text{String} must be written using double quote instead of single quote.

3 The benchmark

The benchmark deals with accuracy, determinateness and efficiency aspects and contains about 950 OCL expressions handling general expressions, invariants and pre- and post-conditions. It is structured in seven parts: the parts B1 to B5 are about accuracy, to evaluate the completeness and correctness of the realization of OCL in a given OCL engine, having into account syntactic and semantic features; the part B6 deals with determinateness; finally, the part B7 deals with efficiency.

Parts B1, B2 and B3 contain expressions referring to an UML model, dealing with navigation, ternary associations, invariants and pre- and post-conditions. Parts B4 and B5 contain OCL queries independent of any model, with expressions to check the three valued OCL logic (B4) and all the possible combinations of operations on collections (B5). Part B6 contains expressions to evaluate the OCL non-deterministic operations as \textit{any} of flatten and finally, part B7 proposes expressions to evaluate the efficiency by means of a mathematical library, including with an OCL function to determine if a given number is prime and a model which defines a graph representing towns and roads. The first one is used to check primitive OCL data types and the second one to check user defined types, by computing the transitive closure and the connected components in the graph.

The benchmark does not explicitly states the OCL specification version to which it is targeted. However, it explicitly proposes some checks to test backward compatibility from OCL 2.0 version to OCL 1.3 version. For example, it proposes versions of the same expression using \textit{allInstances} (OCL 1.3 Specification) and \textit{allInstances()} (OCL 2.0 Specification).

We found some expressions for which we think that the correct result is different from the one proposed in the benchmark, and some syntactic differences with the OCL Specification:
• Some tests make use of the iterate operator with several iteration variables. We think that this does not conform with the concrete syntax of the OCL specification (see clause 9.4.26 in [10]).

• The proposed result when applying the \texttt{collect} operator does not flatten the resulting collection, however the OCL standard specifies that it must be flatten (see clause 11.9.1.6 in [10]).

• Some tests try to compare the \texttt{invalid} value. In such a case the result proposed is \texttt{true}. However, according to the OCL specification, in general an expression where one of the parts is \texttt{invalid} will be itself \texttt{invalid} (see clause 7.5.11 in [10])). Thus, for example the following expression should produce \texttt{invalid} whereas the benchmark specifies that the expected result is \texttt{true}.

\begin{verbatim}
let c = Set{} in
c -> any(true) = c -> asSequence() -> last()
\end{verbatim}

• It writes empty collections by using an \texttt{oclEmpty} operator which is not in the OCL Specification. Thus the benchmark write

\begin{verbatim}
 - Set{} as oclEmpty(Set(TypeName))
 - Bag{} as oclEmpty(Bag(TypeName))
 - Sequence{} as oclEmpty(Sequence(TypeName))
\end{verbatim}

• It uses an \texttt{isUndefined} operator to check the \texttt{undefined} value. This operator is not in the OCL Standard Specification, which defines the \texttt{oclIsUndefined()} instead (see clause 11.3 in [10]).

• It uses the \texttt{isDefined} operator, which is not part of the OCL 2.0 Specification. To be conformed with the OCL specification it should be replaced by the expressions \texttt{not oclIsUndefined()}

• It used the \texttt{empty} operator without parentheses.

We suppose that those operators not contained in the OCL standard are used because the authors of the benchmark need them in their OCL engine (the USE tool) and that the rest of differences are mistakes. Thus, we will adapt such mistakes before executing the test with mOdCL.

4 The process used to test mOdCL with the benchmark

The benchmark has been proposed by the USE team and they provide a bunch of files [7] adapted to be used as source files for the USE tool. Thus, we have files defining the proposed UML models and the test expressions with USE syntax. We have adapted this files to be used as the input to mOdCL, this required the following steps:
• To create the Maude model from the corresponding USE model. We did this by hand, by defining Maude modules with the specification of the model. The benchmark defines four UML models, the CivilStatusWorld model is used in the benchmark B1 (see its definition for USE and mOdCL in Appendix A) and is extended to be used in the benchmark B2 (see Appendix B). The model TernExam is used in benchmark B3 to check ternary association (see Appendix C), and the models MathLib and MapWorld are defined to check efficiency in benchmark B7 (see Appendix G).

• To modify the source USE files with queries, invariants and pre- and post-conditions in order to adapt them to the lexical characteristics of mOdCL (see Section 2.3) and to remove the characteristics which are not in the OCL Specification (see Section 3). This process is made automatically, by using emacs macros and a simple preprocessor. These changes are:

1. To replace terms oclEmpty(Set(Integer)) by Set{}.
2. To replace terms oclEmpty(Bag(Integer)) by Bag{}.
3. To replace terms oclEmpty(Sequence(Integer)) by Sequence{}.
4. To replace terms isUndefined by oclIsUndefined().
5. To replace terms isDefined by not oclIsUndefined().
6. To replace terms empty by empty().
7. To include spaces around tokens (e.g. reject(i|i<4) must be reject(i | i < 4)).
8. To use double quote around string literals.
9. To include Maude comments when necessary around USE code.
10. To enclose the expression to be evaluated in the eval command.

For example, we transform the following expression:

\[
\text{let } c = \text{oclEmpty(Set(Integer))} \text{ in }
\text{c->reject(i|i<4) = c->select(i|not(i<4))}
\]

into the following one, using the eval operator of mOdCL to evaluate the expression

\[
\text{red eval(}
\text{let } c = \text{Set{} in }
\text{c -> reject(i | i < 4) = c -> select(i | not(i < 4))}
\text{) .}
\]

• Finally, we only have to execute the Maude command on the suitable file. For example:
5 Evaluation results

In this section we discuss the results obtained after executing the tests included in the benchmark. All the tests and the results of their execution are included in Appendixes A to G.

5.1 Core Benchmark (B1)

This benchmark checks basic OCL and UML characteristics. It uses a simple UML model named CivilStatusWorld defining a class with simple attributes, an user defined operation without side-effects and a reflexive binary association. It avoids advanced characteristics like enumerations, empty collections and undefined values and includes different test expressions using three different syntax for naming and declaring variables in collection operations.

1. Typed variables: \texttt{aPersonSet->select(p:Person/\texttt{p.gender}="female")}

2. Named variables: \texttt{aPersonSet->select(p/\texttt{p.gender}="female")}

3. Implicit variables: \texttt{aPersonSet->select(\texttt{gender}="female")}

All tests in this part produced the expected result, however:

- To be conform with the OCL specification we had to substitute the \texttt{empty} operator by \texttt{empty()}.
Some tests using `collect` were modified for using `collectNested` because we think it is wrong as we discussed at Section 3.

- We excluded some tests which use more than one iteration variables in `iterate` expressions as we discussed at Section 3 (nevertheless, we transformed such expressions into equivalent expressions as nested iterate and the test produced the expected result).

### 5.2 Extended Core Benchmark (B2)

This benchmark extends the core benchmark with enumerations, side-effects operations and pre- and post-conditions. Furthermore, it considers `undefined` values in expressions.

The USE team simulates side effects operations by means of USE commands with which they create the states concerned in the OCL expressions to be tested. This way, the states needed to evaluate pre- and post-conditions are written by hand. The mOdCL evaluator can be used to dynamically validate side effect operations specified in Maude [15]. Thus, we do not need to produce the target states by hand, they could be dynamically captured during the system execution. However, as we only want to reproduce the benchmark here, we have declared different configurations corresponding to the object diagrams used in this benchmark. For example, to check pre- and post-conditions of the `birth` operation we declare two constants corresponding to the object diagrams before and after the execution of such an operation:

\[
\text{ops state-A-pre state-A-post : } \rightarrow \text{Configuration} .
\]

\[
\begin{align*}
\text{eq state-A-pre} & = <\text{ada} : \text{Person} | \text{name} : \text{null}, \text{civstat} : \text{null}, \text{gender} : \text{null}, \\
& \quad \text{alive} : \text{null}, \text{wife} : \text{null}, \text{husband} : \text{null} > \\
& \quad \text{env}(\text{self} <- \text{ada}) \text{OpEnv}(\text{arg}(\text{aName}, \text{"Ada"}), \text{arg}(\text{aGender}, \text{female})) .
\end{align*}
\]

\[
\begin{align*}
\text{eq state-A-post} & = <\text{ada} : \text{Person} | \text{name} : \text{"Ada"}, \text{civstat} : \text{single}, \text{gender} : \text{female}, \\
& \quad \text{alive} : \text{true}, \text{wife} : \text{null}, \text{husband} : \text{null} > \\
& \quad \text{env}(\text{self} <- \text{ada}) \text{OpEnv}(\text{arg}(\text{aName}, \text{"Ada"}), \text{arg}(\text{aGender}, \text{female})) .
\end{align*}
\]

and we check the invocation to `birth` by executing

Maude> red eval(pre(birth), state-A-pre) .
result Bool: true

Maude> red eval(post(birth), state-A-post) .
result Bool: true

All the tests in this benchmark produced the expected result, except the one concerning the `collect` which we think that proposes an erroneous result.

### 5.3 Advanced Modeling Benchmark (B3)

This part deals with ternary and high-order associations. We do not locate any clause in the OCL specification where it is described what to do with this kind of
associations so that currently mOdCL does not implement these characteristics. The paper [8] describing the benchmark shows the results for 7 selected OCL engines and says that only one supports ternary associations. The mOdCL can represent ternary associations but does not implement all the syntax required to handle them so that some of the proposed test have not been executed.

5.4 Three-valued Logic Benchmark (B4)

OCL supports three valued logic, some expressions can produce undefined values when evaluated. In general, an expression for which one of the parts is undefined will itself be undefined. However there are some exceptions as the logical operators or the \texttt{if then \textit{else} endif} operator. For example, when the expression \texttt{E1 or E2} is evaluated, the result must be true if any of its operands is true, even if the other is undefined. This benchmark checks such OCL operators which deal with three-valued logic.

When we used mOdCL to evaluate the B4 benchmark we find two differences:

- For the following expression

  \texttt{?let B=Set\{Sequence\{true\} \rightarrow excluding(true) \rightarrow last(),false,true\} in}
  \texttt{B \rightarrow iterate(b1,b2:Boolean;}
  \texttt{r:Sequence(Boolean)=oclEmpty(Sequence(Boolean)) |}
  \texttt{r \rightarrow including(b1 \text{ and } b2))}

the benchmark proposes the following result

\texttt{Sequence\{Undefined, false, Undefined, false, false,}
\texttt{Undefined, false, true\} : Sequence(Boolean)}

However we think that this is wrong because according to the OCL 2.3.1 Specification any expression containing undefined must produce undefined. From the OCL 2.0 Specification the \texttt{undefined} value was changed by two values: \texttt{invalid} to represent an error and \texttt{null} to represent and undefined (not erroneous) value. Any expression can contain \texttt{null}, but only logical expression can contain \texttt{invalid}. The mentioned expression defines a \texttt{Sequence} which contains \texttt{Undefined} (representing an error), so that in our opinion this expression must produce \texttt{invalid}, as mOdCL did\textsuperscript{2}:

\texttt{reduce in TEST-B4 : eval\{let B = Set\{Sequence\{true\} \rightarrow excluding(true) \rightarrow last(), true, false\} in B \rightarrow iterate (b1 : Boolean ;}
\texttt{r = Sequence\{} | B \rightarrow iterate (b2 : Boolean; r1 = r | r1
\texttt{\rightarrow including(b1 \text{ and } b2)))\} .}
\texttt{rewrites: 78 in Oms cpu (Oms real) (~ rewrites/second)}
\texttt{result OclInvalid: invalid}

\textsuperscript{2}In fact this expression does not conform with the OCL syntax because it uses 2 variables in the iterate expression(as we discussed in Section 3). However it has been have executed once it has been transformed into an equivalent expression which uses 2 nested iterate
• For the following expression

```ocl
use> ?false implies Sequence{true}->excluding(true)->last()
```

the benchmark proposes `false` as the result

However we think that the result must be `true` as mOdCL did

```
reduce in mOdCL : eval(false implies Sequence{true} -> excluding(true) -> last()) .
rewrites: 18 in Oms cpu (Oms real) (~ rewrites/second)
result Bool: true
```

### 5.5 OCL Laws Benchmark (B5)

This benchmark checks the correct implementation of individual operations. It combines different collection operators to check the result in a number of situations, including empty collections, collections containing undefined and others. This benchmark is organized in 25 files, one for each possible combination of operators on collections (for example, `select` and `reject`, `select` and `iterate`, etc.). Each file contains about 29 test expressions.

Some expressions include `undefined` in collections. As we support OCL 2.3.1 and we have `invalid` and `null` instead, we should decide if to evaluate such expressions as `invalid` or as `null`. It does not make sense to choose `invalid` because any expression containing `invalid` becomes `invalid` (see clause 7.5.11 in [10]). However, if we interpret `undefined` as `null`, how to evaluate expressions as `null < 4`? Having that into account, we exclude such test expression. As an example we show one of such expressions

```ocl
?let c = Set{oclUndefined(Integer)} in
  c->reject(i|i<4) = c->select(i|not(i<4))
```

**Expected Results:**

```
true : Boolean
```

The rest of tests were correctly executed and their results were the expected ones.

### 5.6 Engine Determinateness Properties (B6)

This benchmark deals with the implementation of nondeterministic operations, as for example `any` or `flatten`. As the intended result is not deterministic we study the result of any test to conclude if the result produced by mOdCL is right or not. All the tests in this benchmark were fine however, as we previously mentioned, we disagree with the result proposed in the benchmark for those tests which deal with `invalid` in expressions.
5.7 OCL Engine Efficiency (B7)

This benchmark checks efficiency. As they do not provide time result or detailed information about the execution environment of such efficiency test, we only execute them to proof that mOdCL evaluates correctly the mentioned tests.

Two different tests are proposed which underly in several side-effect free functions defined as OCL functions. Thus, they are a good test to check the use of Maude equations to implement OCL defined functions.

The first test proposes a mathematical library MathLib with functions to calculate the root of a number with a given precision and to determine if a number is prime. The following model StringReal is proposed:

```maude
model StringReal

class MathLib
operations

root(arg:Real,precision:Real):Real=  
  if arg>0 then
    rootAux(arg,  
      precision,  
      if arg>=1 then 1.0 else arg endif,  
      if arg>=1 then arg else 1.0 endif)
  else 0.0 endif

rootAux(arg:Real,precision:Real,low:Real,high:Real):Real=  
  let new:Real=(low+high)/2 in
  if (arg-new*new).abs<=precision  
    then new  
    else rootAux(arg,  
      precision,  
      if new*new<arg then new else low endif,  
      if new*new<arg then high else new endif)
  endif

isPrime(arg:Integer):Boolean=  
  if arg<=1 then false else
    if arg=2 then true else isPrimeAux(arg,2,arg div 2) endif endif

isPrimeAux(arg:Integer,cur:Integer,top:Integer):Boolean=  
  if arg.mod(cur)=0 then false else  
    if cur+1<=top then isPrimeAux(arg,cur+1,top) else true endif
  endif

which has been specified by the following Maude module:

```
eq root(a, precision)
  = if a > 0 then
      rootAux(a,
        precision,
        if a \geq 1 then 1.0 else a endif,
        if a \geq 1 then a else 1.0 endif)
  else 0.0 endif .

eq rootAux(a, precision, low, high)
  = (let new := (low + high) / 2 in
      if (a - new * new) . abs() \leq precision
      then new
      else rootAux(a,
                   precision,
                   if new * new < a then new else low endif,
                   if new * new < a then high else new endif)
      endif) .

vars arg cur top : Nat .
op isPrime : -> OpName [ctor] .
eq isPrime(arg)
  = if arg \leq 1 then false else
    if arg = 2 then true else isPrimeAux(arg, 2, arg . div (2)) endif endif .

op isPrimeAux : -> OpName [ctor] .
eq isPrimeAux(arg, cur, top)
  = if arg . mod(cur) = 0 then false else
    if cur + 1 \leq top then isPrimeAux(arg, cur + 1, top) else true endif
  endif .
endm

All expressions referring this benchmark were correctly calculated, except one expression using \texttt{oclUndefined} in collections. However, one expression computes the root of 10.89 with precision $1/10^{16}$ and mOdCL only can calculate the right result with precision $1/10^{14}$.

The second test proposes the model \texttt{MapWorld}, with towns and roads represented as a graph. It computes the transitive closure and the maximal node sets in which all nodes are connected directly or indirectly. Following we show the USE model

def model MapWorld

class Town
  attributes
    name: String -- key attribute
  operations
    connect(): Set(Town)=

fst->union(snd)
connectPlus():Set(Town)=
    connectPlusAux(connect())
connectPlusAux(aSet:Set(Town)):Set(Town)=
    let oneStep:Set(Town)=aSet->collect(t|t.connect())->flatten()->asSet() in
    if oneStep->exists(t|aSet->excludes(t)) then
        connectPlusAux(aSet->union(oneStep))
    else aSet endif
end

association Road between
town[0..*] role fst
town[0..*] role snd
end

and its Maude specification

mod CLASSES-BENCHMARK-B7-B is
    pr mOdCL .

    ---- class Town
    sort Town .
    subsort Town < Cid .
    op Town : -> Town [ctor] .

    ---- Attributes for Town
    op name : -> AttributeName [ctor] .
    op fst : -> AttributeName [ctor] .
    op snd : -> AttributeName [ctor] .

    ---- Operations for Town
    op connect : -> OpName [ctor] .
eq connect()
        = fst -> union(snd) .

    op connectPlus : -> OpName [ctor] .
eq connectPlus()
        = connectPlusAux(connect()) .

    ops t oneStep : -> Vid .
    var aSet : Set .

eq connectPlusAux(aSet)
        = let oneStep := aSet -> collect(t | t . connect()) -> flatten() -> asSet() in
            if oneStep -> exists(t | aSet -> excludes(t)) then
                connectPlusAux(aSet -> union(oneStep))
            else aSet endif .
endm

All the tests in this benchmark were correctly executed, however one test
was intended to repeat a simple expression $1024 \times 1024$ times and mOdCL only could perform such test $200000$ times before stack overflow. Finally, we had to convert some expression using collect to one equivalent but using collectNested as we previously did.

6 Empirical Evaluation of the Benchmark in different OCL machines

Kuhlmann et al. proposed this benchmark to show the applicability of the benchmark concepts and to make a contribution for improvement of OCL engines. For this reason, they do not provide concrete results of the evaluation of different OCL engines evaluated but general results. We revise here their conclusions to put mOdCL in the context of the existing OCL engines. We want to thank the effort spent in the benchmark development because it has allowed us to detect and fix a number of implementation errors.

They checked 7 OCL evaluation engines (ATL OCL [18], Dresden OCL [17], Eclipse MDT OCL, [9], OCLE [12], Octopus [6] RoclET [14], and USE [5]) and one of them was only partially checked because of resource limitations. They reported that only 401 accuracy and determinateness queries (46.8%) were correctly evaluated by all engines (see [7] to get their checks) and only one accepted all syntactic variations. In our case, mOdCL correctly executes all the proposed test. During the process we found a small number of syntactic and semantic errors, but the initial execution of the benchmark correctly executed almost all the proposed test. After fixing the mentioned bugs mOdCL correctly executes all the proposed test (excluding those which require some extra syntax to deal with ternary associations).

References


A Core Benchmark (B1)

--- model CivilStatusWorld
---
--- class Person
---
--- attributes
--- name:String
--- civstat:String
--- gender:String
--- alive:Boolean
---
--- operations
--- spouse():Person=
---   if self.gender='female' then self.husband else self.wife endif
--- end
---
--- association Marriage between
---   Person [0..1] role wife
---   Person [0..1] role husband
--- end
---

mod CLASSES-BENCHMARK-B1 is
pr mOdCL.

---- class Person
sort Person.
subsort Person < Cid.
op Person : -> Person [ctor].

---- Attributes for Person
----   name: String, civstat: String, gender: String, alive:Boolean
op name : -> AttributeName [ctor].
op civstat : -> AttributeName [ctor].
op gender : -> AttributeName [ctor].
op alive : -> AttributeName [ctor].

---- Associations for Person
----   wife [0..1] Person
----   husband {0..1} Person
op wife : -> AttributeName [ctor].
op husband : -> AttributeName [ctor].

---- Operations
op spouse : -> OpName.
eq spouse() = if gender = "female" then husband else wife endif.
endm

---

mod BENCHMARK-TEST is
pr CLASSES-BENCHMARK-B1.

ops small capital i self2 o p p1 p2 : -> Vid.

19
ops adav emptySeq w h res res1 pair : -> Vid .
ops ada bob cyd dan eve : -> Oid .

op state : -> Configuration .
eq state
  = < ada : Person | name : "Ada", gender : "female", alive : true,
    civstat : "widowed", wife : null, husband : null >
  < bob : Person | name : "Bob", gender : "male", alive : true,
    civstat : "divorced", wife : null, husband : null >
  < cyd : Person | name : "Cyd", gender : "male", alive : false,
    civstat : "married", wife : null, husband : null >
  < dan : Person | name : "Dan", gender : "male", alive : true,
    civstat : "married", wife : eve, husband : null >
  < eve : Person | name : "Eve", gender : "female", alive : true,
endm

---- Invariants from the domain ----------------------------------------
--- 1
--- context Person inv enumCivilStatus:
--- self.civstat='single' or self.civstat='married' or
--- self.civstat='divorced' or self.civstat='widowed'
red eval(
  context Person inv enumCivilStatus:
  self . civstat = "single" or self . civstat = "married" or
  self . civstat = "divorced" or self . civstat = "widowed"
, state) .
--- reduce in BENCHMARK-TEST : eval(context Person inv enumCivilStatus: ((self .
  -- civstat = "divorced" or self . civstat = "widowed") or self . civstat =
  -- "married") or self . civstat = "single", state) .
--- rewrites: 328 in 4ms cpu (0ms real) (82000 rewrites/second)
--- result Bool: true

--- 2
--- context Person inv enumGender:
--- self.gender='female' or self.gender='male'
red eval(
  context Person inv enumGender:
  self . gender = "female" or self . gender = "male"
, state) .
--- reduce in BENCHMARK-TEST : eval(context Person inv enumGender: self . gender =
  -- "female" or self . gender = "male", state) .
--- rewrites: 188 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- 3
--- context Person inv aliveDefined:
--- self.alive=true or self.alive=false
red eval(
  context Person inv enumGender:
  self . gender = "female" or self . gender = "male"
, state) .
--- reduce in BENCHMARK-TEST : eval(context Person inv enumGender: self . gender =
  -- "female" or self . gender = "male", state) .
--- rewrites: 188 in 0ms cpu (0ms real) (~ rewrites/second)
```plaintext
--- result Bool: true
--- 4 OK
--- 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)=
--- 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
red eval(
context Person inv nameCapitalThenSmallLettersVT:
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) =
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
, state)
--- rewrites: 2038 in 12ms cpu (14ms real) (169819 rewrites/second)
--- result Bool: true
--- 5
--- context Person inv nameCapitalThenSmallLetters_VN:
--- 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)=
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
, state)
--- rewrites: 2038 in 12ms cpu (14ms real) (169819 rewrites/second)
--- result Bool: true
--- 5
--- context Person inv nameCapitalThenSmallLetters_VW:
--- 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)=
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
red eval(
context Person inv nameCapitalThenSmallLetters_VW:
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) =
--- result Bool: true
--- 4 OK
--- 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)=
--- 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
red eval(
context Person inv nameCapitalThenSmallLettersVT:
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) =
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
, state)
--- rewrites: 2038 in 12ms cpu (14ms real) (169819 rewrites/second)
--- result Bool: true
--- 5
--- context Person inv nameCapitalThenSmallLetters_VN:
--- 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)=
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
red eval(
context Person inv nameCapitalThenSmallLetters_VN:
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) =
--- result Bool: true
--- 5
--- context Person inv nameCapitalThenSmallLetters_VW:
--- 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)=
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
red eval(
context Person inv nameCapitalThenSmallLetters_VW:
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) =
```
\$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, state).

--- ==========================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
--- rewrites: 2037 in 16ms cpu (15ms real) (127304 rewrites/second)
--- result Bool: true
--- -------------------------------------------------------------------------------
--- 6
--- context Person inv uniqueNameForAll1_P0_VT:
--- Person . allInstances->forAll(self2:Person | self<>self2 implies self . name<>self2 . name)
red eval(
context Person inv uniqueNameForAll1-P0-VT:
Person . allInstances -> forAll(self2 : Person | self <> self2 implies self . name <> self2 . name)
, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameForAll1-P0-VT:
--- Person . allInstances -> forAll (self2 : Person | self <> self2 implies self . name <> self2 . name), state).
--- rewrites: 1176 in 4ms cpu (2ms real) (294000 rewrites/second)
--- result Bool: true
--- ==============================================================
--- 7
--- context Person inv uniqueNameForAll1_P0_VN:
--- Person . allInstances->forAll(self2:Person | self<>self2 implies self . name<>self2 . name)
red eval(
context Person inv uniqueNameForAll1-P0-VN:
Person . allInstances -> forAll(self2 | self <> self2 implies self . name <> self2 . name)
, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameForAll1-P0-VN:
--- Person . allInstances -> forAll (self2 | self <> self2 implies self . name <> self2 . name), state).
--- rewrites: 1175 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
--- ==============================================================
--- 8
--- context Person inv uniqueNameForAll1_P1_VT:
--- Person.allInstances() -> forAll(self2:Person | self <> self2 implies self . name <> self2 . name)
red eval(
context Person inv uniqueNameForAll1-P0-VT:
Person . allInstances() -> forAll(self2 : Person |
self <> self2 implies self . name <> self2 . name
, state).
--- ==================================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameForAll1-P0-VT:
---     Person . allInstances() -> forAll (self2 : Person | self <> self2 implies
---     self . name <> self2 . name), state).
--- rewrites: 1176 in 4ms cpu (2ms real) (294000 rewrites/second)
--- result Bool: true
--- ==================================================================
--- 9
--- context Person inv uniqueNameForAll1_P1_VN:
---     Person.allInstances() -> forAll(self2 |
---     self <> self2 implies self.name <> self2.name)
red eval(
context Person inv uniqueNameForAll1-P0-VN:
     Person . allInstances() -> forAll(self2 |
     self <> self2 implies self . name <> self2 . name)
, state).
--- ==================================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameForAll1-P0-VN:
---     Person . allInstances() -> forAll (self2 | self <> self2 implies self .
---     name <> self2 . name), state).
--- rewrites: 1175 in 4ms cpu (2ms real) (293750 rewrites/second)
--- result Bool: true
--- ==================================================================
--- 10
--- context Person inv uniqueNameForAll1-P0_VT:
---     Person.allInstances>forAll(p1,p2:Person |
---     p1<p2 implies p1.name>p2.name)
red eval(
context Person inv uniqueNameForAll1-P0-VN:
     Person . allInstances -> forAll(p1, p2 : Person |
     p1 <> p2 implies p1 . name <> p2 . name)
, state).
--- ==================================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameForAll1-P0-VT:
---     Person . allInstances -> forAll (p1, p2 : Person | p1 <> p2 implies p1 .
---     name <> p2 . name), state).
--- rewrites: 5904 in 12ms cpu (16ms real) (492000 rewrites/second)
--- result Bool: true
--- ==================================================================
--- 11
--- context Person inv uniqueNameForAll1_P0_VN:
---     Person.allInstances>forAll(p1,p2 |
---     p1<p2 implies p1.name>p2.name)
red eval(
context Person inv uniqueNameForAll1-P0-VN:
     Person . allInstances -> forAll(p1, p2 : Person |
     p1 <> p2 implies p1 . name <> p2 . name)
, state).
--- ==================================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameForAll1-P0-VN:
---     Person . allInstances -> forAll (p1, p2 | p1 <> p2 implies p1 . name <> p2
---     . name), state).
--- rewrites: 5903 in 16ms cpu (23ms real) (368914 rewrites/second)
--- result Bool: true
--- . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
--- 12
---
--- context Person inv uniqueNameForAll2_P1_VT:
--- Person.allInstances()->forAll(p1,p2:Person |
--- p1<>p2 implies p1.name<>p2.name)
red eval(
context Person inv uniqueNameForAll2-P0-VT:
  Person . allInstances() -> forAll(p1, p2 : Person |
  p1 <> p2 implies p1 . name <> p2 . name)
  , state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameForAll2-P0-VT:
--- Person . allInstances() -> forAll (p1, p2 : Person | p1 <> p2 implies p1 .
--- name <> p2 . name), state).
--- rewrites: 5904 in 12ms cpu (10ms real) (492000 rewrites/second)
--- result Bool: true
---========================================================================
--- 13
--- context Person inv uniqueNameForAll2_P1_VN:
--- Person.allInstances()->forAll(p1,p2 |
--- p1<>p2 implies p1.name<>p2.name)
red eval(
context Person inv uniqueNameForAll2-P0-VN:
  Person . allInstances() -> forAll(p1, p2 |
  p1 <> p2 implies p1 . name <> p2 . name)
  , state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameForAll2-P0-VN:
--- Person . allInstances() -> forAll (p1, p2 | p1 <> p2 implies p1 . name <
--- p2 . name), state).
--- rewrites: 5903 in 12ms cpu (18ms real) (491916 rewrites/second)
--- result Bool: true
---========================================================================
--- 14
--- context Person inv uniqueNameIsUnique_P0_VT:
--- Person.allInstances->isUnique(p:Person | p.name)
red eval(
context Person inv uniqueNameIsUnique-P0-VT:
  Person . allInstances -> isUnique(p : Person | p . name)
  , state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameIsUnique-P0-VT:
--- Person . allInstances -> isUnique (p : Person | p . name), state).
--- rewrites: 556 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
---========================================================================
--- 15
--- context Person inv uniqueNameIsUnique_P0_VN:
--- Person.allInstances->isUnique(p | p.name)
red eval(
context Person inv uniqueNameIsUnique-P0-VN:
  Person . allInstances -> isUnique(p | p . name)
  , state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameIsUnique-P0-VN:
--- Person . allInstances -> isUnique (p | p . name), state).
--- rewrites: 555 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---========================================================================
--- 16
--- context Person inv uniqueNameIsUnique_P0_VI:
--- Person.allInstances -> isUnique(name)
red eval(
    context Person inv uniqueNameIsUnique-P0-VI:
        Person . allInstances -> isUnique(name)
        , state).
--- ===============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameIsUnique-P0-VI:
---     Person . allInstances -> isUnique(name), state).
--- rewrites: 706 in 4ms cpu (0ms real) (176500 rewrites/second)
--- result Bool: true

------------------------------------------------------------------------
--- 17
--- context Person inv uniqueNameIsUnique_P1_VT:
---     Person.allInstances() -> isUnique(p:Person | p.name)
red eval(
    context Person inv uniqueNameIsUnique-P1-VT:
        Person . allInstances() -> isUnique(p : Person | p . name)
        , state).
--- ===============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameIsUnique-P1-VT:
---     Person . allInstances() -> isUnique (p : Person | p . name), state).
--- rewrites: 556 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

------------------------------------------------------------------------
--- 18 allInstances()
--- context Person inv uniqueNameIsUnique_P1_VN:
---     Person.allInstances() -> isUnique(p | p.name)
red eval(
    context Person inv uniqueNameIsUnique-P1-VN:
        Person . allInstances() -> isUnique(p | p . name)
        , state).
--- ===============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameIsUnique-P1-VN:
---     Person . allInstances() -> isUnique (p | p . name), state).
--- rewrites: 555 in 4ms cpu (33ms real) (138750 rewrites/second)
--- result Bool: true

------------------------------------------------------------------------
--- 19 allInstances()
--- context Person inv uniqueNameIsUnique_P1_VI:
---     Person.allInstances() -> isUnique(name)
red eval(
    context Person inv uniqueNameIsUnique-P1-VI:
        Person . allInstances() -> isUnique(name)
        , state).
--- ===============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameIsUnique-P1-VI:
---     Person . allInstances() -> isUnique(name), state).
--- rewrites: 706 in 4ms cpu (2ms real) (176500 rewrites/second)
--- result Bool: true

------------------------------------------------------------------------
--- 20
--- context Person inv uniqueNameOne_P0_VT:
---     Person.allInstances -> one(p:Person | self.name=p.name)
red eval(
    context Person inv uniqueNameOne-P0-VT:
        Person . allInstances -> one(p : Person | self . name = p . name)
        , state).
--- ==============================================================

25
--- reduce in BENCHMARK-TEST : eval(context Person inv uniqueNameOne-P0-VT: Person

  . allInstances -> one (p : Person | self . name = p . name), state).
--- rewrites: 911 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true

------------------------------------------------------------------------

--- 21
--- context Person inv uniqueNameOne_P0_VN:
---   Person.allInstances->one(p | self.name=p.name)
red eval(
  context Person inv uniqueNameOne-P0-VN:
    Person . allInstances -> one(p | self . name = p . name)
  , state).
--- =========================================================
--- reduce in BENCHMARK-TEST : eval(context Person inv uniqueNameOne-P0-VN: Person

  . allInstances -> one (p | self . name = p . name), state).
--- rewrites: 808 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true

------------------------------------------------------------------------

--- 22
--- context Person inv uniqueNameOne_P0_VI:
---   Person.allInstances-> one(self.name=name)
red eval(
  context Person inv uniqueNameOne-P0-VI:
    Person . allInstances -> one(self . name = name)
  , state).
--- =========================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameOne-P0-VI:

  . allInstances -> one (self . name = name), state).
--- rewrites: 1061 in Oms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
--- . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

------------------------------------------------------------------------

--- 23
--- context Person inv uniqueNameOne_P1_VT:
---   Person.allInstances()->one(p:Person | self.name=p.name)
red eval(
  context Person inv uniqueNameOne-P1-VT:
    Person . allInstances() -> one(p : Person | self . name = p . name)
  , state).
--- =========================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameOne-P1-VT:

  . allInstances() -> one (p : Person | self . name = p . name), state).
--- rewrites: 911 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true

------------------------------------------------------------------------

--- 24
--- context Person inv uniqueNameOne_P1_VN:
---   Person.allInstances()->one(p|self.name=p.name)
red eval(
  context Person inv uniqueNameOne-P1-VN:
    Person . allInstances() -> one(p | self . name = p . name)
  , state).
--- =========================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameOne-P1-VN:

  . allInstances() -> one (p | self . name = p . name), state).
--- rewrites: 910 in Oms cpu (1ms real) (~ rewrites/second)
--- result Bool: true

------------------------------------------------------------------------

26
--- 25
--- context Person inv uniqueNameOne_P1_VI:
--- Person.allInstances() -> one(self.name=name)
red eval(
  context Person inv uniqueNameOne-P1-VI:
    Person . allInstances() -> one(self . name = name)
  , state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameOne-P1-VI:
---   Person . allInstances() -> one(self . name = name), state).
--- rewrites: 1061 in 4ms cpu (2ms real) (265250 rewrites/second)
--- result Bool: true
------------------------------------------------------------------------
--- 26
--- context Person inv uniqueNameOneNot_P0_VI:
--- not Person.allInstances->one(name=name)
red eval(
  context Person inv uniqueNameOneNot-P0-VI:
    not Person . allInstances -> one(name = name)
  , state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameOneNot-P0-VI:
---   not Person . allInstances -> one(name = name), state).
--- rewrites: 591 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
------------------------------------------------------------------------
--- 27
--- context Person inv uniqueNameOneNot_P1_VI:
--- not Person.allInstances()->one(name=name)
red eval(
  context Person inv uniqueNameOne-P1-VI:
    not Person . allInstances() -> one(name = name)
  , state).
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv uniqueNameOne-P1-VI:
---   not Person . allInstances() -> one(name = name), state).
--- rewrites: 591 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
------------------------------------------------------------------------
--- 28
--- context Person inv femaleHasNoWife:
--- self.gender='female' implies self.wife->isEmpty()
red eval(
  context Person inv femaleHasNoWife:
    self . gender = "female" implies self . wife -> isEmpty()
  , state).
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv femaleHasNoWife: self .
--- gender = "female" implies self . wife -> isEmpty(), state).
--- rewrites: 178 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
------------------------------------------------------------------------
--- 29
--- context Person inv maleHasNoHusband:
--- self.gender='male' implies self.husband -> isEmpty()
red eval(
  context Person inv maleHasNoHusband:
    self . gender = "male" implies self . husband -> isEmpty()
  , state).
--- ==============================================================
--- reduce in BENCHMARK-TEST : eval(context Person inv maleHasNoHusband: self .  
--- gender = "male" implies self . husband -> isEmpty(), state) .  
--- rewrites: 178 in 0ms cpu (0ms real) (~ rewrites/second)  
--- result Bool: true

------------------------------------------------------------------------
---- Invariants encoding queries ----------------------------------------
------------------------------------------------------------------------

--- 30  
--- context Person inv abcName_P0_VT:  
--- let ada=Person.allInstances -> any(p:Person | p.name='Ada') in  
--- let bob=Person.allInstances -> any(p:Person | p.name='Bob') in  
--- let cyd=Person.allInstances -> any(p:Person | p.name='Cyd') in  
--- Set{ada,bob, cyd} -> collect(p:Person | p.name)=Bag{'Ada','Bob','Cyd'}
red eval(
context Person inv abcName-P0-VT:  
let ada:Vid = Person . allInstances -> any (p : Person | p . name = "Ada") in  
let bob:Vid = Person . allInstances -> any (p : Person | p . name = "Bob") in  
let cyd:Vid = Person . allInstances -> any (p : Person | p . name = "Cyd") in  
--- rewrites: 2049 in 8ms cpu (6ms real) (256092 rewrites/second)  
--- result Bool: true

------------------------------------------------------------------------
--- 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'}
red eval(
context Person inv abcName-P0-VN:  
let ada:Vid = Person . allInstances -> any (p : Person | p . name = "Ada") in  
let bob:Vid = Person . allInstances -> any (p : Person | p . name = "Bob") in  
let cyd:Vid = Person . allInstances -> any (p : Person | p . name = "Cyd") in  
--- rewrites: 1318 in 0ms cpu (1ms real) (~ rewrites/second)  
--- result Bool: true

------------------------------------------------------------------------
--- 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'}
red eval(
  context Person inv abcName-P0-VI:
  let ada:Vid = Person . allInstances -> any(name = "Ada") in
  let bob:Vid = Person . allInstances -> any(name = "Bob") in
  let cyd:Vid = Person . allInstances -> any(name = "Cyd") in
  Set{ada:Vid, bob:Vid, cyd:Vid} -> collect(name) = Bag{"Ada", "Bob", "Cyd"}
  , state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcName-P0-VI: let
--- ada:Vid = Person . allInstances -> any(name = "Ada") in let bob:Vid =
--- Person . allInstances -> any(name = "Bob") in let cyd:Vid = Person .
--- allInstances -> any(name = "Cyd") in Set{ada:Vid, bob:Vid, cyd:Vid} ->
--- rewrites: 2502 in 4ms cpu (6ms real) (625343 rewrites/second)
--- result Bool: true
--- .................................
--- 33
--- context Person inv abcName_P1_VT:
--- let ada=Person.allInstances()->any(p:Person|p.name='Ada') in
--- let bob=Person.allInstances()->any(p:Person|p.name='Bob') in
--- let cyd=Person.allInstances()->any(p:Person|p.name='Cyd') in
--- Set{ada,bob,cyd}->collect(p:Person|p.name)=Bag{'Ada','Bob','Cyd'}
red eval(
  context Person inv abcName-P1-VT:
  let ada:Vid = Person . allInstances() -> any(p : Person | p . name = "Ada") in
  let bob:Vid = Person . allInstances() -> any(p : Person | p . name = "Bob") in
  let cyd:Vid = Person . allInstances() -> any(p : Person | p . name = "Cyd") in
  , state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcName-P1-VT: let
--- ada:Vid = Person . allInstances() -> any(p : Person | p . name = "Ada") in
--- let bob:Vid = Person . allInstances() -> any(p : Person | p . name = "Bob") in
--- let cyd:Vid = Person . allInstances() -> any(p : Person | p . name = "Cyd") in
--- rewrites: 2049 in 4ms cpu (2ms real) (512250 rewrites/second)
--- result Bool: true
--- .................................................................
--- 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'}
red eval(
  context Person inv abcName-P1-VT:
  let ada:Vid = Person . allInstances() -> any(p | p . name = "Ada") in
  let bob:Vid = Person . allInstances() -> any(p | p . name = "Bob") in
  let cyd:Vid = Person . allInstances() -> any(p | p . name = "Cyd") in
  Set{ada:Vid, bob:Vid, cyd:Vid} -> collect(p | p . name) = Bag{"Ada", "Bob", "Cyd"}
  , state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcName-P1-VT: let
--- ada:Vid = Person . allInstances() -> any(p | p . name = "Ada") in
--- let bob:Vid = Person . allInstances() -> any(p | p . name = "Bob") in
--- let cyd:Vid = Person . allInstances() -> any(p | p . name = "Cyd") in
--- Set{ada:Vid, bob:Vid, cyd:Vid} -> collect(p | p . name) = Bag{"Ada", "Bob", "Cyd"}
  , state).
--- bob:Vid = Person . allInstances() -> any (p | p . name = "Bob") in let
--- cyd:Vid = Person . allInstances() -> any (p | p . name = "Cyd") in Set{
--- ada:Vid, bob:Vid, cyd:Vid} -> collect (p | p . name) = Bag("Ada", "Bob",
--- "Cyd"), state).
--- rewrites: 2048 in 4ms cpu (5ms real) (512000 rewrites/second)
--- result Bool: true
--------------------------------------------------------------------------------------------------------
--- 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'}
red eval(
context Person inv abcName_P1_VI:
let ada:Vid = Person . allInstances() -> any(name = "Ada") in
let bob:Vid = Person . allInstances() -> any(name = "Bob") in
let cyd:Vid = Person . allInstances() -> any(name = "Cyd") in
Set{ada,bob,cyd} -> collect(name) = Bag("Ada","Bob","Cyd")
, state).
--- rewrites in BENCHMARK-B1-TEST : eval(context Person inv abcName_P1-VI: let
--- ada:Vid = Person . allInstances() -> any(name = "Ada") in let bob:Vid =
--- Person . allInstances() -> any(name = "Bob") in let cyd:Vid = Person .
--- allInstances() -> any(name = "Cyd") in Set{ada, bob, cyd} -> collect(name)
--- = Bag{"Ada","Bob","Cyd"}, state).
--- rewrites: 2502 in 4ms cpu (3ms real) (625343 rewrites/second)
--- result Bool: true
--------------------------------------------------------------------------------------------------------
--- 36
--- context Person inv abcNameDotShortcutP0_VT:
--- let ada=Person.allInstances->any(p:Person|p.name='Ada') in
--- let bob=Person.allInstances->any(p:Person|p.name='Bob') in
--- let cyd=Person.allInstances ->any(p:Person|p.name='Cyd') in
--- Set{ada,bob,cyd}.name=Bag{'Ada','Bob','Cyd'}
red eval(
context Person inv abcNameDotShortcutP0_VT:
let ada:Vid = Person . allInstances -> any(p : Person | p . name = "Ada") in
let bob:Vid = Person . allInstances -> any(p : Person | p . name = "Bob") in
let cyd:Vid = Person . allInstances -> any(p : Person | p . name = "Cyd") in
Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob", "Cyd")
, state).
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcNameDotShortcutP0-VT: let
--- ada:Vid = Person . allInstances -> any(p : Person | p . name = "Ada")
--- in let bob:Vid = Person . allInstances -> any(p : Person | p . name =
--- "Bob") in let cyd:Vid = Person . allInstances -> any(p : Person | p . name
--- = "Cyd") in Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob",
--- "Cyd"), state).
--- rewrites: 2148 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
--------------------------------------------------------------------------------------------------------
--- 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'}
red eval(
context Person inv abcNameDotShortcutP0_VN:
let ada:Vid = Person . allInstances -> any(p : Person | p . name = "Ada") in
let bob:Vid = Person . allInstances -> any(p : Person | p . name = "Bob") in
let cyd:Vid = Person . allInstances -> any(p : Person | p . name = "Cyd") in
Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob", "Cyd")
, state).
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcNameDotShortcutP0-VN: let
--- ada:Vid = Person . allInstances -> any (p : Person | p . name = "Ada")
--- in let bob:Vid = Person . allInstances -> any (p : Person | p . name =
--- "Bob") in let cyd:Vid = Person . allInstances -> any (p : Person | p . name
--- = "Cyd") in Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob",
--- "Cyd"), state).
--- rewrites: 2148 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
--------------------------------------------------------------------------------------------------------
--- 38
--- 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'}
red eval(
context Person inv abcNameDotShortcutP0_VN:
let ada:Vid = Person . allInstances -> any(p : Person | p . name = "Ada") in
let bob:Vid = Person . allInstances -> any(p : Person | p . name = "Bob") in
let cyd:Vid = Person . allInstances -> any(p : Person | p . name = "Cyd") in
Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob", "Cyd")
, state).
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcNameDotShortcutP0-VN: let
--- ada:Vid = Person . allInstances -> any (p : Person | p . name = "Ada")
--- in let bob:Vid = Person . allInstances -> any (p : Person | p . name =
--- "Bob") in let cyd:Vid = Person . allInstances -> any (p : Person | p . name
--- = "Cyd") in Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob",
--- "Cyd"), state).
--- rewrites: 2148 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
--------------------------------------------------------------------------------------------------------
context Person inv abcNameDotShortcutP0-VN:
  let ada:Vid = Person . allInstances -> any(p | p . name = "Ada") in
  let bob:Vid = Person . allInstances -> any(p | p . name = "Bob") in
  let cyd:Vid = Person . allInstances -> any(p | p . name = "Cyd") in
  Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob", "Cyd")
, state).
--- ==============================================================
--- reduce in BENCHMARK-TEST : eval(context Person inv abcNameDotShortcutP0-VN: let
--- ada:Vid = Person . allInstances -> any (p | p . name = "Ada") in let
--- bob:Vid = Person . allInstances -> any (p | p . name = "Bob") in let
--- cyd:Vid = Person . allInstances -> any (p | p . name = "Cyd") in Set{
--- rewrites: 1398 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
--- ==============================================================
--- 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'} red eval(
context Person inv abcNameDotShortcutP0-VI:
  let ada:Vid = Person . allInstances -> any(name = "Ada") in
  let bob:Vid = Person . allInstances -> any(name = "Bob") in
  let cyd:Vid = Person . allInstances -> any(name = "Cyd") in
  Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob", "Cyd")
, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcNameDotShortcutP0-VI: let
--- ada:Vid = Person . allInstances -> any(name = "Ada") in let bob:Vid =
--- Person . allInstances -> any(name = "Bob") in let cyd:Vid = Person .
--- allInstances -> any(name = "Cyd") in Set{ada:Vid, bob:Vid, cyd:Vid} . name
--- = Bag("Ada", "Bob", "Cyd"), state) .
--- rewrites: 2510 in Oms cpu (6ms real) (~ rewrites/second)
--- result Bool: true
--- ==============================================================
--- 39
--- context Person inv abcNameDotShortcutP1_VT:
--- let ada=Person.allInstances()->any(p:Person|p.name='Ada') in
--- let bob=Person.allInstances()->any(p:Person|p.name='Bob') in
--- let cyd=Person.allInstances()->any(p:Person|p.name='Cyd') in
--- Set{ada,bob,cyd}.name=Bag('Ada','Bob','Cyd') red eval(
context Person inv abcNameDotShortcutP1-VT:
  let ada:Vid = Person . allInstances () -> any(p : Person | p . name = "Ada") in
  let bob:Vid = Person . allInstances () -> any(p : Person | p . name = "Bob") in
  let cyd:Vid = Person . allInstances () -> any(p : Person | p . name = "Cyd") in
  Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob", "Cyd")
, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcNameDotShortcutP1-VT:
--- let ada:Vid = Person . allInstances() -> any(p : Person | p . name =
--- name = "Bob") in let cyd:Vid = Person . allInstances() -> any(p : Person |
--- p . name = "Cyd") in Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada",
--- "Bob", "Cyd"), state).
--- rewrites: 2148 in 4ms cpu (2ms real) (537000 rewrites/second)
--- result Bool: true

--- 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'}
red eval(
  context Person inv abcNameDotShortcutP1-VN:
  let ada:Vid = Person . allInstances() -> any(p | p . name = "Ada") in
  let bob:Vid = Person . allInstances() -> any(p | p . name = "Bob") in
  let cyd:Vid = Person . allInstances() -> any(p | p . name = "Cyd") in
  Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob", "Cyd")

--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcNameDotShortcutP1-VN:
---   let ada:Vid = Person . allInstances() -> any(name = "Ada") in
---   let bob:Vid = Person . allInstances() -> any(name = "Bob") in
---   let cyd:Vid = Person . allInstances() -> any(name = "Cyd") in
---   Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob", "Cyd")
---   state)
--- rewrites: 2147 in 4ms cpu (30ms real) (536615 rewrites/second)
--- result Bool: true

--- 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'}
red eval(
  context Person inv abcNameDotShortcutP1-VI:
  let ada:Vid = Person . allInstances() -> any(name = "Ada") in
  let bob:Vid = Person . allInstances() -> any(name = "Bob") in
  let cyd:Vid = Person . allInstances() -> any(name = "Cyd") in
  Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob", "Cyd")

--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcNameDotShortcutP1-VI:
---   let ada:Vid = Person . allInstances() -> any(name = "Ada") in
---   let bob:Vid = Person . allInstances() -> any(name = "Bob") in
---   let cyd:Vid = Person . allInstances() -> any(name = "Cyd") in
---   Set{ada:Vid, bob:Vid, cyd:Vid} . name = Bag("Ada", "Bob", "Cyd")
---   state)
--- rewrites: 2510 in 4ms cpu (8ms real) (627500 rewrites/second)
--- result Bool: true

--- 42
--- context Person inv abcAttrs_P0_VT:
---   let ada=Person.allInstances -> any(p:Person | p.name='Ada') in
---   let bob=Person.allInstances -> any(p:Person | p.name='Bob') in
---   let cyd=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' ,true },false}
red eval(
  context Person inv abcAttrs-P0-VT:
  let ada:Vid = Person . allInstances -> any(p : Person | p . name = "Ada") in
let bob:Vid = Person . allInstances -> any(p : Person | p . name = "Bob") in
let cyd:Vid = Person . allInstances -> any(p : Person | p . name = "Cyd") in
Set(ada:Vid, bob:Vid, cyd:Vid) ->
collectNested(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)}
, state).
--- =========================================================
    Sequence("Bob", 'divorced', 'male', true ),
    Sequence("Cyd", 'married', 'male', false)}, state).
--- rewrites: 2879 in Oms cpu (3ms real) (~ rewrites/second)
--- result Bool: true
------------------------------------------------------------------------
--- 44
--- 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)}
red eval(
    context Person inv abcAttrs-P0-VN:
    let ada:Vid = Person . allInstances -> any(p | p . name = "Ada") in
    let bob:Vid = Person . allInstances -> any(p | p . name = "Bob") in
    let cyd:Vid = Person . allInstances -> any(p | p . name = "Cyd") in
    Set(ada, bob, cyd) ->
collectNested(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)}, state).
--- ==============================================================
--- reduce in BENCHMARK-TEST : eval(context Person inv abcAttrs-P0-VN: let ada:Vid
--- = Person . allInstances -> any(p | p . name = "Ada") in let bob:Vid =
--- Person . allInstances -> any(p | p . name = "Bob") in let cyd:Vid =
--- Person . allInstances -> any(p | p . name = "Cyd") in Set(ada:Vid,
--- bob:Vid, cyd:Vid) -> collectNested(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}), state).)
--- rewrites: 2048 in Oms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
-------------------------------------------------------------------------------
--- 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})
red eval(
  context Person inv abcAttrs-P0-VI:
  let ada:Vid = Person.allInstances() -> any(name = "Ada") in
  let bob:Vid = Person.allInstances() -> any(name = "Bob") in
  let cyd:Vid = Person.allInstances() -> any(name = "Cyd") in
  Set(ada,bob, cyd) ->
  collectNested(Sequence{name,civstat,gender,alive}) =
  Bag(Sequence{"Ada", "widowed", "female",true },
      Sequence{"Bob", "divorced", "male",true },
      Sequence{"Cyd", "married", "male",false}), state).
--- =========================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcAttrs-P0-VI: let
--- ada:Vid = Person.allInstances() -> any(name = "Ada") in let bob:Vid =
--- Person.allInstances() -> any(name = "Bob") in let cyd:Vid = Person.
--- allInstances() -> any(name = "Cyd") in
--- Set(ada,bob, cyd) ->
--- collectNested(Sequence{name,civstat,gender,alive}) = Bag(Sequence{"Ada",
--- "widowed", "female",true }, Sequence{"Bob", "divorced", "male",true},
--- Sequence{"Cyd", "married", "male",false}), state).
--- rewrites: 3602 in 4ms cpu (4ms real) (900500 rewrites/second)
--- result Bool: true
----- . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
--- 45
--- context Person inv abcAttrs_P1-VT:
--- let ada=Person.allInstances()->any(p:Person|p.name='Ada') in
--- let bob=Person.allInstances()->any(p:Person|p.name='Bob') in
--- let cyd=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}), state).
--- rewrites: 2879 in 4ms cpu (2ms real) (719750 rewrites/second)
--- result Bool: true

--- context Person inv abcAttrs-P1-VT:
let ada:Vid = Person.allInstances() -> any(p : Person | p . name = "Ada") in
let bob:Vid = Person.allInstances() -> any(p : Person | p . name = "Bob") in
let cyd:Vid = Person.allInstances() -> any(p : Person | p . name = "Cyd") in
Set(ada,bob, cyd) ->
collectNested(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}), state).
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcAttrs-P1-VT: let
--- ada:Vid = Person.allInstances() -> any(p : Person | p . name = "Ada") in
--- let bob:Vid = Person.allInstances() -> any(p : Person | p . name =
--- "Bob") in let cyd:Vid = Person.allInstances() -> any(p : Person | p .
--- name = "Cyd") in
--- Set(ada,bob, cyd) ->
--- collectNested(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}), state).
--- rewrites: 2879 in 4ms cpu (2ms real) (719750 rewrites/second)
--- result Bool: true
--- 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}}
red eval(
  context Person inv abcAttrs-P1-VN:
  let ada:Vid = Person . allInstances() -> any(p | p . name = "Ada") in
  let bob:Vid = Person . allInstances() -> any(p | p . name = "Bob") in
  let cyd:Vid = Person . allInstances() -> any(p | p . name = "Cyd") in
  Set{ada:Vid, bob:Vid, cyd:Vid } ->
    collectNested(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}}, state).
--- ==================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcAttrs-P1-VN: let
---   ada:Vid = Person . allInstances() -> any(p | p . name = "Ada") in
---   bob:Vid = Person . allInstances() -> any(p | p . name = "Bob") in
---   cyd:Vid = Person . allInstances() -> any(p | p . name = "Cyd") in
---   Set{ada:Vid, bob:Vid, cyd:Vid } ->
---     collectNested(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}}, state).
--- result Bool: true
--- ==================================================
--- 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}}
red eval(
  context Person inv abcAttrs-P1-VI:
  let ada:Vid = Person . allInstances() -> any(name = "Ada") in
  let bob:Vid = Person . allInstances() -> any(name = "Bob") in
  let cyd:Vid = Person . allInstances() -> any(name = "Cyd") in
  Set{ada:Vid, bob:Vid, cyd:Vid } ->
    collectNested(Sequence{name,civstat,gender,alive}) =
    Bag{Sequence{"Ada","widowed" ,"female",true },
         Sequence{"Bob","divorced","male" ,true },
         Sequence{"Cyd","married" ,"male" ,false}}, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv abcAttrs-P1-VI: let
---   ada:Vid = Person . allInstances() -> any(name = "Ada") in
---   bob:Vid = Person . allInstances() -> any(name = "Bob") in
---   cyd:Vid = Person . allInstances() -> any(name = "Cyd") in
---   Set{ada:Vid, bob:Vid, cyd:Vid } ->
---     collectNested(Sequence{name,civstat,gender,alive}) =
---     Bag{Sequence{"Ada","widowed" ,"female",true },
---          Sequence{"Bob","divorced","male" ,true },
---          Sequence{"Cyd","married" ,"male" ,false}}, state).
--- result Bool: true
--- ==============================================================
--- allInstances() -> any(name = "Cyd") in Set{ada:Vid, bob:Vid, cyd:Vid} ->
--- collectNested(Sequence(name, civstat, gender, alive)) = Bag{Sequence("Ada",
--- "widowed", "female", true), Sequence("Bob", "divorced", "male", true),
--- Sequence("Cyd", "married", "male", false), state).
--- rewrites: 3602 in 8ms cpu (9ms real) (450250 rewrites/second)
--- result Bool: true
------------------------------------------------------------------------
--- 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"}
red eval(context Person inv personsWithUndefinedHusband-P0-VT:
Person . allInstances ->
select(p : Person |
 p . husband =
 Person . allInstances -> any(p | p . wife -> isNull()) . wife
 ) -> collect(p : Person | p . name) = Bag("Ada", "Bob", "Cyd", "Dan")
, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
--- personsWithUndefinedHusband-P0-VT: Person . allInstances -> select (p :
--- Person | p . husband = Person . allInstances -> any (p | p . wife ->
--- isNull()) . wife) -> collect (p : Person | p . name) = Bag("Ada", "Bob",
--- "Cyd", "Dan"), state).
--- rewrites: 2348 in 4ms cpu (2ms real) (587000 rewrites/second)
--- result Bool: true
------------------------------------------------------------------------
--- 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"}
red eval(context Person inv personsWithUndefinedHusband-P0-VN:
Person . allInstances ->
select(p |
 p . husband =
 Person . allInstances -> any(p | p . wife -> isNull()) . wife
 ) -> collect(p | p . name) = Bag("Ada", "Bob", "Cyd", "Dan")
, state).
--- ==============================================================
--- reduce in BENCHMARK-TEST : eval(context Person inv
--- personsWithUndefinedHusband-P0-VN: Person . allInstances -> select (p | p .
--- husband = Person . allInstances -> any (p | p . wife -> isNull()) . wife)
--- rewrites: 2128 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
------------------------------------------------------------------------
--- 50
--- context Person inv personsWithUndefinedHusband_P0_VI:
--- Person.allInstances->
--- select(
--- husband=
--- Person.allInstances->any(wife->isEmpty()) . wife
--- )->collect(name)=Bag(‘Ada’, ’Bob’, ’Cyd’, ’Dan’)
red eval(
  context Person inv personsWithUndefinedHusband-P0-VI:
  Person . allInstances ->
  select(
    husband =
    Person . allInstances -> any(wife -> isNotEmpty()) . wife
  ) -> collect(name)=Bag(‘Ada’, ’Bob’, ’Cyd’, ’Dan’)
, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
--- personsWithUndefinedHusband-P0-VI: Person . allInstances -> select(husband
--- = Person . allInstances -> any(wife -> isNotEmpty()) . wife) -> collect(name)
--- rewrites: 2770 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
--- . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
--- ==============================================================
--- 51
--- context Person inv personsWithUndefinedHusband-P1-VT:
--- Person.allInstances()->
--- select(p:Person|
--- p.husband=
--- Person.allInstances()->any(p:Person|p.wife->isEmpty()).wife
red eval(
  context Person inv personsWithUndefinedHusband-P1-VT:
  Person . allInstances() ->
  select(p : Person |
    p . husband =
    Person . allInstances() -> any(p | p . wife -> isNotEmpty()) . wife
, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
--- personsWithUndefinedHusband-P1-VT: Person . allInstances() -> select (p :
--- Person | p . husband = Person . allInstances() -> any (p | p . wife ->
--- isNotEmpty()) . wife) -> collect (p : Person | p . name) = Bag(“Ada”, “Bob”,
--- rewrites: 2348 in 4ms cpu (3ms real) (587000 rewrites/second)
--- result Bool: true
--- . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
--- ==============================================================
--- 52
--- context Person inv personsWithUndefinedHusband-P1-VN:
--- Person.allInstances()->
--- select(p|
--- p.husband=
--- Person.allInstances()->any(p|p.wife->isEmpty()).wife
red eval(
  context Person inv personsWithUndefinedHusband-P1-VN:
  Person . allInstances() ->
  select(p |
    p . husband =
    Person . allInstances() -> any(p | p . wife -> isNotEmpty()) . wife
, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
--- personsWithUndefinedHusband-P1-VN: Person . allInstances() -> select (p | p
--- . husband = Person . allInstances() -> any (p | p . wife -> isEmpty()) .
--- rewrites: 2347 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
--- ==============================================================
--- 53
--- context Person inv personsWithUndefinedHusband_P1_VI:
--- Person.allInstances()->
--- select(
--- husband=
--- Person.allInstances()->any(wife->isEmpty()).wife
--- )->collect(name)=Bag("Ada","Bob","Cyd","Dan")
red eval(
context Person inv personsWithUndefinedHusband-P1-VI:
Person . allInstances() ->
select(

husband =
Person . allInstances -> any(wife -> isEmpty()) . wife
) ->
collect(name) = Bag("Ada", "Bob", "Cyd", "Dan")

, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
--- personsWithUndefinedHusband-P1-VI: Person . allInstances() -> select(
--- husband = Person . allInstances -> any(wife -> isEmpty()) . wife) ->
--- rewrites: 2770 in 4ms cpu (4ms real) (692500 rewrites/second)
--- result Bool: true
--- ==============================================================
--- 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"}
red eval(
context Person inv personsWithUndefinedHusbandAnyFalse-P0-VT:
Person . allInstances ->
select(p : Person |

p . husband = Person . allInstances -> any(p | false)
) ->
collect(p : Person | p . name) = Bag("Ada", "Bob", "Cyd", "Dan")

, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
--- personsWithUndefinedHusbandAnyFalse-P0-VN: Person . allInstances -> select
--- (p | p . husband = Person . allInstances -> any (p | false)) -> collect (p
--- rewrites: 2647 in 4ms cpu (4ms real) (661750 rewrites/second)
--- result Bool: true
--- ==============================================================
--- 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"}
red eval(
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")
, state).
--- ============
--- reduce in BENCHMARK-TEST : eval(context Person inv
---   personsWithUndefinedHusbandAnyFalse-P0-VN: Person . allInstances -> select
---     (p | p . husband = Person . allInstances -> any (p | false)) -> collect (p
--- rewrites: 2528 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
---
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
---   personsWithUndefinedHusbandAnyFalse-P0-VI: Person . allInstances -> select
---     (p | p . husband = Person . allInstances -> any (p | false)) -> collect (p
--- rewrites: 2920 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
---
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
---   personsWithUndefinedHusbandAnyFalse-P1-VT: Person . allInstances -> select
---     (p : Person | p . husband = Person . allInstances() -> any(p : Person | false)
---       ) -> collect(p : Person | p . name) = Bag("Ada", "Bob", "Cyd", "Dan")
--- rewrites: 2648 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
---
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
---   personsWithUndefinedHusbandAnyFalse-P1-VN: Person . allInstances() -> select
---     (p : Person | p . husband = Person . allInstances() -> any(p | false)
---       ) -> collect(p : Person | p . name) = Bag("Ada", "Bob", "Cyd", "Dan")
--- rewrites: 2648 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
---
--- 56
--- context Person inv personsWithUndefinedHusbandAnyFalse_P0_VI:
---   Person.allInstances->
---     select(
---       husband=Person.allInstances->any(false)
---     )->collect(name)=Bag('Ada','Bob','Cyd','Dan')
red eval(
context Person inv personsWithUndefinedHusbandAnyFalse_P0_VI:
  Person . allInstances ->
  select(
    husband = Person . allInstances -> any(false)
  ) -> collect(name) = Bag("Ada", "Bob", "Cyd", "Dan")
, state).
--- ============
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
---   personsWithUndefinedHusbandAnyFalse-P0-VI: Person . allInstances -> select(
---     husband = Person . allInstances -> any(false)) -> collect(name) = Bag(
--- rewrites: 2920 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
---
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
---   personsWithUndefinedHusbandAnyFalse-P1-VT: Person . allInstances() -> select(
---     p : Person | p . husband = Person . allInstances() -> any (p |
---       false)) -> collect (p : Person | p . name) = Bag("Ada", "Bob", "Cyd",
---       "Dan"), state).
--- rewrites: 2648 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
---
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
---   personsWithUndefinedHusbandAnyFalse-P1-VN: Person . allInstances() -> select(
---     p : Person | p . husband = Person . allInstances() -> any (p |
---       false)) -> collect (p : Person | p . name) = Bag("Ada", "Bob", "Cyd",
---       "Dan"), state).
--- rewrites: 2648 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
---
--- 57
--- context Person inv personsWithUndefinedHusbandAnyFalse_P1_VT:
---   Person.allInstances()->
---     select(p:Person|
---       p.husband=Person.allInstances()->any(p:Person|false)
---     )->collect(p:Person|p.name)=Bag('Ada','Bob','Cyd','Dan')
red eval(
context Person inv personsWithUndefinedHusbandAnyFalse_P1_VT:
  Person . allInstances() ->
  select(p : Person |
    p . husband = Person . allInstances() -> any(p | false)
  ) -> collect(p : Person | p . name) = Bag("Ada", "Bob", "Cyd", "Dan")
, state).
--- ============
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
---   personsWithUndefinedHusbandAnyFalse-P1-VT: Person . allInstances() -> select(
--- rewrites: 2648 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
---
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
---   personsWithUndefinedHusbandAnyFalse-P1-VN: Person . allInstances() -> select(
--- rewrites: 2648 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
---
--- 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"}
red eval(
context Person inv personsWithUndefinedHusbandAnyFalse-P1-VN: Person . allInstances() ->
select(p | p . husband = Person . allInstances() -> any(p | false)) ->
--- ===================================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
--- personsWithUndefinedHusbandAnyFalse-P1-VN: Person . allInstances() ->
--- select (p | p . husband = Person . allInstances() -> any (p | false)) ->
--- rewrites: 2647 in 4ms cpu (2ms real) (661750 rewrites/second)
--- result Bool: true
--- ---------------------------------------------------------------------
--- 59
--- context Person inv personsWithUndefinedHusbandAnyFalse_P1_VI:
--- Person.allInstances() ->
--- select(
--- husband=Person.allInstances()->any(false)
--- )->collect(name)=Bag{"Ada", "Bob", "Cyd", "Dan"}
red eval(
context Person inv personsWithUndefinedHusbandAnyFalse-P0-VI: Person . allInstances() ->
select(
husband = Person . allInstances -> any(false)
--- ===================================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv
--- personsWithUndefinedHusbandAnyFalse-P0-VI: Person . allInstances() ->
--- select(husband = Person . allInstances -> any(false)) -> collect(name) =
--- rewrites: 2920 in 8ms cpu (8ms real) (365000 rewrites/second)
--- result Bool: true
--- ---------------------------------------------------------------------
--- 60
--- context Person inv possiblePairs_P0_VT:
--- let ada=Person.allInstances -> any(p:Person | p.name='Ada') in
--- let emptySeq=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"})
--- WARNING. Two variables in iterate.
--- This does not conform the OCL specification
--- it should be syntactically incorrect in OCL
--- ---------------------------------------------------------------------
--- 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'}}
--- WARNING. Two variables in iterate.
--- This does not comform the OCL specification
--- it should be syntactically incorrect in OCL
------------------------------------------------------------------------
--- 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'}}
--- WARNING. Two variables in iterate.
--- This does not comform the OCL specification
--- it should be syntactically incorrect in OCL
------------------------------------------------------------------------
--- 63
--- context Person inv possiblePairs_P1_VT:
--- let ada=Person.allInstances () -> any(p:Person | p.name='Ada') in
--- let emptySeq=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'}}
--- WARNING. Two variables in iterate.
--- This does not comform the OCL specification
--- it should be syntactically incorrect in OCL
------------------------------------------------------------------------
--- 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'}}
--- WARNING. Two variables in iterate.
--- This does not conform the OCL specification
--- it should be syntactically incorrect in OCL

--- 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'}}
--- WARNING. Two variables in iterate.
--- This does not conform the OCL specification
--- it should be syntactically incorrect in OCL

--- 66
--- context Person inv flatten_P0_VT:
--- let dan=Person.allInstances->any(p:Person|p.name='Dan') in
--- let eve=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}
red eval(
context Person inv flatten-P0-VT:
let dan:Vid = Person . allInstances -> any (p : Person | p . name = "Dan") in
let eve:Vid = Person . allInstances -> any (p : Person | p . name = "Eve") in
Set{Bag{eve:Vid},
Bag{eve:Vid . spouse()},
Bag{eve:Vid . spouse() . spouse()},
Bag{eve:Vid . spouse() . spouse() . spouse()},
Bag{eve:Vid . spouse() . spouse() . spouse().spouse()},
Bag{eve:Vid . spouse() . spouse().spouse().spouse().spouse().spouse()}} ->
flatten() = Set{dan:Vid, eve:Vid}
, state) .
--- ---------------------------------------------
--- reduce in BENCHMARK-B1-TEST: eval(context Person inv flatten-P0-VT: let
--- dan:Vid = Person . allInstances -> any (p : Person | p . name = "Dan") in
--- let eve:Vid = Person . allInstances -> any (p : Person | p . name = "Eve")
--- in Set{Bag{eve:Vid}, Bag{eve:Vid . spouse()}, Bag{eve:Vid . spouse() .
--- spouse()}, Bag{eve:Vid . spouse() . spouse() . spouse() . spouse()},
--- Bag{eve:Vid . spouse() . spouse() . spouse().spouse()},
--- Bag{eve:Vid . spouse() . spouse().spouse().spouse().spouse().spouse()}} ->
flatten() = Set{dan:Vid, eve:Vid}, state) .
--- rewrites: 3647 in 4ms cpu (2ms real) (911750 rewrites/second)
--- result Bool: true
--- ---------------------------------------------

--- 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},

--- 42
--- 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}
red eval(
context Person inv flatten-P0-VN:
let dan:Vid = Person . allInstances -> any(p | p . name = "Dan") in
let eve:Vid = Person . allInstances -> any(p | p . name = "Eve") in
Set{Bag{eve:Vid},
Bag{eve:Vid . spouse()},
Bag{eve:Vid . spouse() . spouse()},
Bag{eve:Vid . spouse() . spouse().spouse()},
Bag{eve:Vid . spouse() . spouse().spouse().spouse()},
Bag{eve:Vid . spouse() . spouse().spouse().spouse().spouse().spouse()} } ->
flatten() = Set{dan:Vid, eve:Vid}
, state).
--- ==================================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv flatten-P0-VN: let
dan:Vid = Person . allInstances -> any(p | p . name = "Dan") in
let eve:Vid = Person . allInstances -> any(p | p . name = "Eve") in
Set{Bag{eve:Vid},
Bag{eve:Vid . spouse()},
Bag{eve:Vid . spouse() . spouse()},
Bag{eve:Vid . spouse() . spouse().spouse()},
Bag{eve:Vid . spouse() . spouse().spouse().spouse()},
Bag{eve:Vid . spouse() . spouse().spouse().spouse().spouse().spouse()}} ->
flatten() = Set{dan:Vid, eve:Vid}
, state).
--- result Bool: true

--- 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().spouse()}} ->
--- flatten()=Set{dan,eve}
red eval(
context Person inv flatten-P0-VI:
let dan:Vid = Person . allInstances -> any(name = "Dan") in
let eve:Vid = Person . allInstances -> any(name = "Eve") in
Set{Bag{eve:Vid},
Bag{eve:Vid . spouse()},
Bag{eve:Vid . spouse() . spouse()},
Bag{eve:Vid . spouse() . spouse().spouse()},
Bag{eve:Vid . spouse() . spouse().spouse().spouse()},
Bag{eve:Vid . spouse() . spouse().spouse().spouse().spouse().spouse()} } ->
flatten() = Set{dan:Vid, eve:Vid}
, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv flatten-P0-VI: let
dan:Vid = Person . allInstances -> any(name = "Dan") in
let eve:Vid = Person . allInstances -> any(name = "Eve") in
Set{Bag{eve:Vid},
Bag{eve:Vid . spouse()},
Bag{eve:Vid . spouse() . spouse()},
Bag{eve:Vid . spouse() . spouse().spouse()},
Bag{eve:Vid . spouse() . spouse().spouse().spouse()},
Bag{eve:Vid . spouse() . spouse().spouse().spouse().spouse().spouse()}} ->
flatten() = Set{dan:Vid, eve:Vid}
, state).
--- result Bool: true

--- 68
--- spouse()}) -> flatten() = Set{dan:Vid, eve:Vid}, state).
--- rewrites: 4188 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
--- . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
--- ==============================================================
--- 69
--- context Person inv flatten_P1_VT:
--- let dan=Person.allInstances() -> any(p:Person | p.name='Dan') in
--- let eve=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}
red eval(
context Person inv flatten-P1-VT:
let dan:Vid = Person . allInstances -> any(p : Person | p . name = "Dan") in
let eve:Vid = Person . allInstances -> any(p : Person | p . name = "Eve") in
Set{Bag(eve:Vid),
    Bag(eve:Vid . spouse()),
    Bag(eve:Vid . spouse() . spouse()),
    Bag(eve:Vid . spouse() . spouse() . spouse()),
    Bag(eve:Vid . spouse() . spouse() . spouse() . spouse()),
    Bag(eve:Vid . spouse() . spouse() . spouse() . spouse() . spouse())} ->
flatten() = Set{dan:Vid, eve:Vid}
, state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv flatten-P1-VT: let
--- dan:Vid = Person . allInstances -> any(p : Person | p . name = "Dan") in
--- let eve:Vid = Person . allInstances -> any(p : Person | p . name = "Eve") in
--- in Set{Bag(eve:Vid), Bag(eve:Vid . spouse()), Bag(eve:Vid . spouse() .
--- spouse() . spouse() . spouse() . spouse())} -> flatten() = Set{dan:Vid,
--- eve:Vid}, state).
--- rewrites: 3647 in 4ms cpu (3ms real) (911750 rewrites/second)
--- result Bool: true
--- ==============================================================
--- 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}
red eval(
context Person inv flatten-P1-VN:
let dan:Vid = Person . allInstances() -> any(p | p . name = "Dan") in
let eve:Vid = Person . allInstances() -> any(p | p . name = "Eve") in
Set{Bag(eve:Vid),
    Bag(eve:Vid . spouse()),
    Bag(eve:Vid . spouse() . spouse()),
    Bag(eve:Vid . spouse() . spouse() . spouse()),

```
Bag(eve:Vid . spouse() . spouse() . spouse() . spouse()),
Bag(eve:Vid . spouse() . spouse() . spouse() . spouse() . spouse()) ->
flatten() = Set{dan:Vid, eve:Vid}

--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv flatten-P1-VN: let
--- dan:Vid = Person . allInstances() -> any (p | p . name = "Dan") in let
--- eve:Vid = Person . allInstances() -> any (p | p . name = "Eve") in Set{Bag{
--- eve:Vid}, Bag(eve:Vid . spouse()), Bag(eve:Vid . spouse() . spouse()), Bag{
--- eve:Vid . spouse() . spouse() . spouse()), Bag(eve:Vid . spouse() . spouse(
--- ) . spouse() . spouse()), Bag(eve:Vid . spouse() . spouse() . spouse() . spouse(),
--- spouse() . spouse()) . spouse() . spouse()) . spouse() . spouse() . spouse()
--- rewrites: 3646 in 4ms cpu (3ms real) (911500 rewrites/second)
--- result Bool: true
--- ==============================================================
--- 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}
--- red eval(
--- 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:Vid, eve:Vid}
--- , state).
--- ==============================================================
--- reduce in BENCHMARK-B1-TEST : eval(context Person inv flatten-P1-VI: let
--- dan:Vid = Person . allInstances() -> any (name = "Dan") in let eve:Vid =
--- Person . allInstances() -> any (name = "Eve") in Set{Bag{eve:Vid}, Bag{
--- rewrite: 4188 in 4ms cpu (4ms real) (1047000 rewrites/second)
--- result Bool: true
--- ==============================================================

--- 45
B Extended Core Benchmark (B2)

--- model CivilStatusWorld
---
--- enum CivilStatus {single, married, divorced, widowed}
--- enum Gender {female, male}
---
--- class Person
---
--- attributes
---    name:String
---    civstat:CivilStatus
---    gender:Gender
---    alive:Boolean
---
--- operations
---
--- birth(aName:String, aGender:Gender)
---    pre freshUnlinkedPerson: name.isUndefined and civstat.isUndefined and
---        gender.isUndefined and alive.isUndefined and
---        wife.isUndefined and husband.isUndefined
---    post nameAssigned: name=aName -- equivalent to 'aName=self.name'
---    post civstatAssigned: civstat=#single
---    post genderAssigned: gender=aGender
---    post isAliveAssigned: alive=true -- equivalent to 'alive'
---
--- marry(aSpouse:Person)
---    pre aSpouseDefined: aSpouse.isDefined
---    pre isAlive: alive
---    pre aSpouseAlive: aSpouse.alive
---    pre isUnmarried: civstat<>#married
---    pre aSpouseUnmarried: aSpouse.civstat<>#married
---    pre differentGenders: gender<>aSpouse.gender
---    post isMarried: civstat=#married
---    post femaleHasMarriedHusband: gender=#female implies
---        husband=aSpouse and husband.civstat=#married
---    post maleHasMarriedWife: gender=#male implies
---        wife=aSpouse and wife.civstat=#married
---
--- divorce()
---    pre isMarried: civstat=#married
---    pre isAlive: alive
---    pre husbandAlive: gender=#female implies husband.alive
---    pre wifeAlive: gender=#male implies wife.alive
---    post isDivorced: civstat=#divorced
---    post husbandDivorced: gender=#female implies
---        husband.isUndefined and husband@pre.civstat=#divorced
---    post wifeDivorced: gender=#male implies
---        wife.isUndefined and wife@pre.civstat=#divorced
---
--- death()
---    pre isAlive: alive
---    post notAlive: not(alive)
---    post husbandWidowed: gender=#female and husband.isDefined implies
---        husband@pre.wife.isUndefined and husband@pre.civstat=#widowed
---    post wifeWidowed: gender=#male and wife.isDefined implies
---        wife@pre.husband.isUndefined and wife@pre.civstat=#widowed
---
--- spouse():Person=if gender=#female then husband else wife endif
--- constraints
--- inv attributesDefined: name.isDefined and civstat.isDefined and
gender.isDefined and alive.isDefined
--- inv nameCapitalThenSmallLetters:
---   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(name.substring(1,1)) and
---   Set{2..name.size}->forAll(i |
---     small->includes(name.substring(i,i)))
--- inv nameIsUnique: Person.allInstances->forAll(self |
---     small->includes(name.substring(i,i)))
--- inv femaleHasNoWife: gender=#female implies wife.isUndefined
--- inv maleHasNoHusband: gender=#male implies husband.isUndefined
--- end
---
--- association Marriage between
--- Person [0..1] role wife
--- Person [0..1] role husband
--- end
---
mod CLASSES-BENCHMARK-B2 is
pr mOdCL.

---- class Person
sort Person.
subsort Person < Cid.
op Person : -> Person [ctor].

---- Attributes for Person
---- name: String, civstat: String, gender: String, alive:Boolean
op name : -> AttributeName [ctor].
op civstat : -> AttributeName [ctor].
op gender : -> AttributeName [ctor].
op alive : -> AttributeName [ctor].

---- Associations for Person
---- wife [0..1] Person
---- husband {0..1] Person
op wife : -> AttributeName [ctor].
op husband : -> AttributeName [ctor].

---- enum CivilStatus {single, married, divorced, widowed}
sort CivilStatus.
op single : -> CivilStatus [ctor].
op married : -> CivilStatus [ctor].
op divorced : -> CivilStatus [ctor].
op widowed : -> CivilStatus [ctor].

---- enum Gender {female, male}

47
sort Gender .
op female : -> Gender .
op male : -> Gender .

subsorts CivilStatus Gender < Enum .

op pre : OpName -> OclExp .
op post : OpName -> OclExp .

---- Operations
--- birth(aName:String, aGender:Gender)
op birth : -> OpName .
op aName : -> Arg .
op aGender : -> Arg .

eq pre(birth) = (name . oclIsUndefined() and civstat . oclIsUndefined()
    and gender . oclIsUndefined() and alive . oclIsUndefined()
    and wife . oclIsUndefined() and husband . oclIsUndefined()) .
eq post(birth) = (name = aName
    and civstat = single
    and gender = aGender
    and alive = true) .

--- marry(aSpouse:Person)
op marry : -> OpName .
op aSpouse : -> Arg .

eq pre(marry) = (not (aSpouse . oclIsUndefined())
    and alive
    and aSpouse . alive
    and civstat <> married
    and aSpouse . civstat <> married
    and gender <> aSpouse . gender) .
eq post(marry) = (civstat = married
    and (gender = female implies husband = aSpouse and husband . civstat = married)
    and (gender = male implies wife = aSpouse and wife . civstat = married)) .

--- divorce()
op divorce : -> OpName .
eq pre(divorce) = (civstat = married
    and alive
    and (gender = female implies husband . alive)
    and (gender = male implies wife . alive)) .
eq post(divorce) = (civstat = divorced
    and (gender = female implies husband . oclIsUndefined()
        and husband @pre . civstat = divorced)
    and (gender = male implies wife . oclIsUndefined()
        and wife @pre . civstat = divorced)) .

--- death()
op death : -> OpName .
eq pre(death) = alive .

48
eq post(death) = (not (alive)
and (gender = female
    and not husband.oclIsUndefined()
    implies husband @pre . wife.oclIsUndefined()
and husband @pre . civstat = widowed)
and (gender = male
    and not wife.oclIsUndefined()
    implies wife @pre . husband.oclIsUndefined()
and wife @pre . civstat = widowed)).

--- spouse():Person
op spouse : -> OpName .
eq spouse() = if gender = female then husband else wife endif.
endm

mod BENCHMARK-TEST-B2 is
inc CLASSES-BENCHMARK-B2 .
ops ada bob cyd dan eve: -> Oid .
ops o p p1 p2 p3 p4: -> Vid .

--- Test State - For Part 1
op state-1 : -> Configuration .
eq state-1
= < ada : Person | name : "Ada", civstat : widowed, gender : female, alive : true,
    wife : null, husband : null >
< bob : Person | name : "Bob", civstat : divorced, gender : male, alive : true,
    wife : null, husband : null >
< cyd : Person | name : "Cyd", civstat : married, gender : male, alive : false,
    wife : ada, husband : null >
< dan : Person | name : "Dan", civstat : single, gender : male, alive : true,
    wife : null, husband : null > .

--- Test State - For Part 2
op state-2 : -> Configuration .
eq state-2
= < ada : Person | name : "Ada", civstat : married, gender : female, alive : true,
    wife : null, husband : dan >
< bob : Person | name : "Bob", civstat : divorced, gender : male, alive : true,
    wife : null, husband : null >
< cyd : Person | name : "Cyd", civstat : married, gender : male, alive : false,
    wife : ada, husband : null >
< dan : Person | name : "Dan", civstat : married, gender : male, alive : true,
    wife : ada, husband : null > .

--- Test State A. For precondition
op state-A-pre : -> Configuration .
eq state-A-pre
= < ada : Person | name : null, civstat : null, gender : null, alive : null,
    wife : null, husband : null >
env(self <- ada)
OpEnv((arg(aName, "Ada"), arg(aGender, female))).

--- Test State A. For postcondition
eq state-A-post
= < ada : Person | name : "Ada", civstat : single, gender : female, alive : true,
    wife : null, husband : null >
```
env(self <- ada)
OpEnv((arg(aName, "Ada"), arg(aGender, female))) .

--- ----------------------------------------
--- Test State B. For precondition
op state-B-pre : -> Configuration .
eq state-B-pre
  = < ada : Person | name : "Ada", civstat : single, gender : female, alive : true,
    wife : null, husband : null >
  < bob : Person | name : null, civstat : null, gender : null, alive : null,
    wife : null, husband : null >
env(self <- bob)
OpEnv((arg(aName, "Bob"), arg(aGender, male))) .

--- Test State B. For postcondition
op state-B-post : -> Configuration .
eq state-B-post
  = < ada : Person | name : "Ada", civstat : single, gender : female, alive : true,
    wife : null, husband : null >
  < bob : Person | name : "Bob", civstat : single, gender : male, alive : true,
    wife : null, husband : null >
env(self <- bob)
OpEnv((arg(aName, "Bob"), arg(aGender, male))) .

--- ----------------------------------------
--- Test State C. For precondition
op state-C-pre : -> Configuration .
eq state-C-pre
  = < ada : Person | name : "Ada", civstat : single, gender : female, alive : true,
    wife : null, husband : null >
  < bob : Person | name : "Bob", civstat : single, gender : male, alive : true,
    wife : null, husband : null >
  < cyd : Person | name : null, civstat : null, gender : null, alive : null,
    wife : null, husband : null >
env(self <- cyd)
OpEnv((arg(aName, "Cyd"), arg(aGender, male))) .

--- Test State C. For postcondition
op state-C-post : -> Configuration .
eq state-C-post
  = < ada : Person | name : "Ada", civstat : married, gender : female, alive : true,
    wife : null, husband : bob >
  < bob : Person | name : "Bob", civstat : married, gender : male, alive : true,
    wife : ada, husband : null >
env(self <- ada)
OpEnv(arg(aSpouse, bob)) .

--- ----------------------------------------
--- Test State D. For precondition
op state-D-pre : -> Configuration .
eq state-D-pre
  = < ada : Person | name : "Ada", civstat : married, gender : female, alive : true,
    wife : null, husband : bob >
  < bob : Person | name : "Bob", civstat : married, gender : male, alive : true,
    wife : ada, husband : null >
  < cyd : Person | name : null, civstat : null, gender : null, alive : null,
    wife : null, husband : null >
env(self <- cyd)
OpEnv((arg(aName, "Cyd"), arg(aGender, male))) .

--- Test State D. For postcondition
```
--- Test State D. For precondition
op state-D-pre : -> Configuration .
eq state-D-pre
 = < ada : Person | name : "Ada", civstat : married, gender : female, alive : true,
  wife : null, husband : bob >
 < bob : Person | name : "Bob", civstat : married, gender : male, alive : true,
  wife : ada, husband : null >
 < cyd : Person | name : "Cyd", civstat : single, gender : male, alive : true,
  wife : null, husband : null >
 env(self <- cyd)
 OpEnv((arg(aName, "Cyd"), arg(aGender, male))) .
--- ----------------------------------------
--- Test State E. For precondition
op state-E-pre : -> Configuration .
eq state-E-pre
 = < ada : Person | name : "Ada", civstat : married, gender : female, alive : true,
  wife : null, husband : bob >
 < bob : Person | name : "Bob", civstat : married, gender : male, alive : true,
  wife : ada, husband : null >
 < cyd : Person | name : "Cyd", civstat : single, gender : male, alive : true,
  wife : null, husband : null >
 env(self <- ada)
 OpEnv(empty) .
--- Test State E. For postcondition
op state-E-post : -> Configuration .
eq state-E-post
 = < ada : Person | name : "Ada", civstat : divorced, gender : female, alive : true,
  wife : null, husband : null >
 < bob : Person | name : "Bob", civstat : divorced, gender : male, alive : true,
  wife : null, husband : null >
 < cyd : Person | name : "Cyd", civstat : single, gender : male, alive : true,
  wife : null, husband : null >
 env(self <- ada)
 OpEnv(empty) .
--- Test State F. For precondition
op state-F-pre : -> Configuration .
eq state-F-pre
 = < ada : Person | name : "Ada", civstat : divorced, gender : female, alive : true,
  wife : null, husband : null >
 < bob : Person | name : "Bob", civstat : divorced, gender : male, alive : true,
  wife : null, husband : null >
 < cyd : Person | name : "Cyd", civstat : single, gender : male, alive : true,
  wife : null, husband : null >
 env(self <- cyd)
 OpEnv(arg(aSpouse, ada)) .
--- Test State F. For postcondition
op state-F-post : -> Configuration .
eq state-F-post
 = < ada : Person | name : "Ada", civstat : married, gender : female, alive : true,
  wife : null, husband : cyd >
 < bob : Person | name : "Bob", civstat : divorced, gender : male, alive : true,
  wife : null, husband : null >
 < cyd : Person | name : "Cyd", civstat : married, gender : male, alive : true,
  wife : ada, husband : null >
 env(self <- cyd)
OpEnv(arg(aSpouse, ada)) .

--- Test State G. For precondition
op state-G-pre : -> Configuration .
eq state-G-pre
  = < ada : Person | name : "Ada", civstat : married, gender : female, alive : true,
    wife : null, husband : cyd >
  < bob : Person | name : "Bob", civstat : divorced, gender : male, alive : true,
    wife : null, husband : null >
  < cyd : Person | name : "Cyd", civstat : married, gender : male, alive : true,
    wife : ada, husband : null >
  < dan : Person | name : null, civstat : null, gender : null, alive : null,
    wife : null, husband : null >
  env(self <- dan)
  OpEnv((arg(aName, "Dan"), arg(aGender, male))).

--- Test State G. For postcondition
op state-G-post : -> Configuration .
eq state-G-post
  = < ada : Person | name : "Ada", civstat : married, gender : female, alive : true,
    wife : null, husband : cyd >
  < bob : Person | name : "Bob", civstat : divorced, gender : male, alive : true,
    wife : null, husband : null >
  < cyd : Person | name : "Cyd", civstat : married, gender : male, alive : true,
    wife : ada, husband : null >
  < dan : Person | name : "Dan", civstat : single, gender : male, alive : true,
    wife : null, husband : null >
  env(self <- dan)
  OpEnv((arg(aName, "Dan"), arg(aGender, male))).

--- Test State H. For precondition
op state-H-pre : -> Configuration .
eq state-H-pre
  = < ada : Person | name : "Ada", civstat : married, gender : female, alive : true,
    wife : null, husband : cyd >
  < bob : Person | name : "Bob", civstat : divorced, gender : male, alive : true,
    wife : null, husband : null >
  < cyd : Person | name : "Cyd", civstat : married, gender : male, alive : true,
    wife : ada, husband : null >
  < dan : Person | name : "Dan", civstat : single, gender : male, alive : true,
    wife : null, husband : null >
  env(self <- cyd)
  OpEnv(empty).
3.4 Pre- and Post-conditions

--- 1
--- ada.birth("Ada", female)
red eval(pre(birth), state-A-pre) .
red eval(post(birth), state-A-post) .
--- rewrites: 135 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
--- rewrites: 77 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- 2
--- bob.birth("Bob", male)
red eval(pre(birth), state-B-pre) .
red eval(post(birth), state-B-post) .
--- reduce in BENCHMARK-TEST-B2 : eval(pre(birth), state-B-pre) .
--- rewrites: 135 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
--- rewrites: 77 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- 3
--- ada.marry(bob)
red eval(pre(marry), state-C-pre) .
red eval(post(marry), state-C-post) .
--- reduce in BENCHMARK-TEST-B2 : eval(pre(marry), state-C-pre) .
--- rewrites: 114 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
--- reduce in BENCHMARK-TEST-B2 : eval(post(marry), state-C-post) .
--- rewrites: 146 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- 4
--- cyd.birth("Cyd", male)
red eval(pre(birth), state-D-pre) .
red eval(post(birth), state-D-post) .
--- reduce in BENCHMARK-TEST-B2 : eval(pre(birth), state-D-pre) .
--- rewrites: 135 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
--- rewrites: 77 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- 5
--- ada.divorce
red eval(pre(divorce), state-E-pre) .
red eval(post(divorce), state-E-post, state-E-pre).

--- rewrites: 111 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- rewrites: 157 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- 6
--- cyd.marry(ada)
red eval(pre(marry), state-F-pre).
red eval(post(marry), state-F-post, state-F-pre).

--- reduce in BENCHMARK-TEST-B2 : eval(pre(marry), state-F-pre).
--- rewrites: 114 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- rewrites: 153 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- 7
--- dan.birth("Dan", male)
red eval(pre(birth), state-G-pre).
red eval(post(birth), state-G-post).

--- rewrites: 135 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- rewrites: 77 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- 8
--- cyd.death()
red eval(pre(death), state-H-pre).

--- rewrites: 16 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- rewrites: 224 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- 3.4.1. Queries Based On Test State - Part 1

--- 1
--- use> ?Set{ada,bob, cyd, dan}.name
red eval(
    Set {ada, bob, cyd, dan} . name
, state-1).

--- reduce in BENCHMARK-TEST-B2 : eval(Set{ada, bob, cyd, dan} . name, state-1).
--- rewrites: 123 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bag: Bag{"Ada", "Bob", "Cyd", "Dan"}
---
--- Bag{"Ada", "Bob", "Cyd", "Dan"} : Bag(String)
---
--- 2
--- use> ?Person.allInstances.name
red eval(
    Person . allInstances . name
, state-1).
---
--- reduce in BENCHMARK-TEST-B2 : eval(Person . allInstances . name, state-1).
---
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bag: Bag{"Dan", "Cyd", "Bob", "Ada"}
---
--- Bag{"Ada", "Bob", "Cyd", "Dan"} : Bag(String)
---
--- 3
--- use> ?Set{ada,bob,cyd,dan}->collect(name)
red eval(
    Set{ada,bob,cyd,dan} -> collect(name)
, state-1).
---
--- reduce in BENCHMARK-TEST-B2 : eval(Set{ada, bob, cyd, dan} -> collect(name), state-1).
---
--- rewrites: 124 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bag: Bag{"Ada", "Bob", "Cyd", "Dan"}
---
--- Bag{"Ada", "Bob", "Cyd", "Dan"} : Bag(String)
---
--- 4
--- use> ?Set{ada,bob,cyd,dan}->collect(p|p.name)
red eval(
    Set{ada, bob, cyd, dan} -> collect (p | p . name)
, state-1).
---
--- reduce in BENCHMARK-TEST-B2 : eval(Set{ada, bob, cyd, dan} -> collect (p | p . name), state-1).
---
--- rewrites: 99 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bag: Bag{"Ada", "Bob", "Cyd", "Dan"}
---
--- Bag{"Ada", "Bob", "Cyd", "Dan"} : Bag(String)
---
--- 5
--- use> ?Set{ada,bob,cyd,dan}->collect(Sequence{name,civstat,gender,alive})
red eval(
    Set{ada, bob, cyd, dan} -> collect(Sequence{name, civstat, gender, alive})
, state-1).
---
--- reduce in BENCHMARK-TEST-B2 : eval(Set{ada, bob, cyd, dan} -> collect(Sequence{name, civstat, gender, alive}), state-1).
---
--- rewrites: 368 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bag: Bag{"Ada", widowed, female, true, "Bob", divorced, male, true, "Cyd", married, male, false, "Dan", single, male, true}
---
--- 6
--- use> ?Set{ada,bob,cyd,dan}->
---
--- collect(p|Sequence{p.name,p.civstat,p.gender,p.alive})
red eval(
    Set{ada, bob, cyd, dan} ->
    collectNested(p | Sequence{p . name, p . civstat, p . gender, p . alive})
--- reduce in BENCHMARK-TEST-B2 : eval(Set{ada, bob, cyd, dan} -> collectNested (p
--- | Sequence(p . name, p . civstat, p . gender, p . alive)), state-1) .
--- rewrites: 257 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bag: Bag{Sequence("Ada", widowed, female, true), Sequence{"Bob",
--- divorced, male, true}, Sequence{"Cyd", married, male, false}, Sequence{
--- "Dan", single, male, true})
---
--- Bag{Sequence{'Ada', #widowed, #female, true },
--- Sequence{'Bob', #divorced, #male, true },
--- Sequence{'Cyd', #married, #male, false},
--- Sequence{'Dan', #single, #male, true }): Bag{Sequence(OclAny)}
--- WARNING. mOdCL uses collectNested instead of collect. We think that
--- this expressions is wrong in the benchmark
---
--- 7
--- use> ?Person.allInstances->forAll(p1,p2|p1<>p2 implies p1.name<>p2.name)
red eval(
   Person . allInstances -> forAll(p1 |
      Person . allInstances -> forAll(p2 | p1 <> p2 implies p1 . name <> p2 . name))
   ,state-1) .
--- rewrites: 770 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
---
--- 8
--- use> ?Person.allInstances->isUnique(p|p.name)
red eval(
   Person . allInstances -> isUnique(p | p . name)
   ,state-1) .
--- rewrites: 87 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- 9
--- use> ?Person.allInstances->forAll(p|
--- (p.gender=#female implies p.wife->isEmpty) and
--- (p.gender=#male implies p.husband->isEmpty))
red eval(
   Person . allInstances -> forAll(p |
      (p . gender = female implies p . wife -> isEmpty()) and
      (p . gender = male implies p . husband -> isEmpty()))
   ,state-1) .
--- rewrites: 281 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- true : Boolean
---
--- 10
--- use> ?Person.allInstances->
--- select(p|p.alive and p.civstat<>#widowed).civstat
red eval(
    Person . allInstances ->
      select (p | p . alive and p . civstat <> widowed) . civstat
, state-1).
---
--- alive and p . civstat <> widowed) . civstat, state-1).
--- rewrites: 219 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bag: Bag{single, divorced}
---
--- Bag{#divorced,#single} : Bag(CivilStatus)
---
--- 11
--- use> ?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()
---
red eval(
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()
, state-1).
---
--- reduce in BENCHMARK-TEST-B2 : eval(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(), state-1).
---
rewrites: 192 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- true : Boolean
---
--- 12
--- use> ?let o:OclAny=ada in o
red eval(
  let o = ada in o
, state-1).
---
--- reduce in BENCHMARK-TEST-B2 : eval(let o = ada in o, state-1).
--- rewrites: 8 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Oid: ada
---
--- ada : Person
--- 13
--- use> ?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))
--- WARNING. It uses two variables in iterate.
--- It should not conform with the OCL Specification

--- 14
--- use> ?Sequence{ada.name,ada.civstat,ada.alive,ada.gender,
--- dan.name,dan.civstat,dan.alive,dan.gender}
red eval(
    Sequence{ada . name, ada . civstat, ada . alive, ada . gender ,
    dan . name, dan . civstat, dan . alive, dan . gender},
    state-1) .
--- reduce in BENCHMARK-TEST-B2 : eval(Sequence{ada . name, ada . civstat, ada .
--- alive, ada . gender, dan . name, dan . civstat, dan . alive, dan . gender},
--- state-1) .
--- rewrites: 93 in 4ms cpu (1ms real) (~ rewrites/second)
--- result Sequence: Sequence{Ada" , widowed, true, female, "Dan", single, true,
--- male}
---
--- Sequence{Ada",#widowed,true,#female,'Dan',#single,true,#male} :
--- Sequence(OclAny)
---
--- 3.4.2. Queries Based On Test State - Part 2
---
--- use> !set dan.civstat=#married
red eval( Sequence{ada, ada . spouse(), dan, dan . spouse()}
    ,state-1) .
--- reduce in BENCHMARK-TEST-B2 : eval(Sequence{ada . spouse(), dan, dan .
--- spouse()}, state-1) .
--- rewrites: 83 in 4ms cpu (1ms real) (20744 rewrites/second)
--- result Sequence: Sequence{ada, null, dan, null}
---
--- use> ?Sequence{ada,ada.spouse(),dan,dan.spouse()}
red eval( Sequence{ada, ada . spouse(), ada . spouse() . spouse(),
    ada . spouse() . spouse() . spouse()}
    ,state-2) .
--- reduce in BENCHMARK-TEST-B2 : eval(Sequence{ada, ada . spouse(), dan, dan . spouse()}, state-2) .
--- rewrites: 83 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Sequence: Sequence{ada, dan, dan, ada}
---
--- Sequence{ada,dan,dan,ada} : Sequence(Person)
------------------------------------------------------------------------
--- 3
--- use> ?Sequence{ada,ada.spouse(),ada.spouse().spouse(),
--- ada.spouse().spouse().spouse()}
red eval(
Sequence{ada, ada . spouse(), ada . spouse() . spouse(),
     ada . spouse() . spouse() . spouse()})
, state-2) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B2 : eval(Sequence{ada, ada . spouse(), ada . spouse() .
--- . spouse(), ada . spouse() . spouse() . spouse()}, state-2) .
--- rewrites: 207 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Sequence: Sequence{ada, dan, ada, dan}
---
--- Sequence{ada,dan,ada,dan} : Sequence(Person)
------------------------------------------------------------------------
--- 4
--- use> ?Person.allInstances->
--- select(husband=Person.allInstances->any(wife->isEmpty).wife)
red eval(
Person . allInstances ->
select(husband = Person . allInstances -> any(wife -> isEmpty()) . wife),
state-2) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B2 : eval(Person . allInstances -> select(husband =
--- Person . allInstances -> any(wife -> isEmpty()) . wife), state-2) .
--- rewrites: 522 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Set: Set{dan, cyd, bob}
---
--- Set{bob,cyd,dan} : Set(Person)
------------------------------------------------------------------------
C Advanced Modeling Benchmark (B3)

--- model TernExam
---
--- class Person
---   attributes
---     name: String
---   end
---
--- associationclass Exam between
---   Person[0..*] role examinee
---   Person[0..*] role examiner
---   Person[0..*] role recorder
--- attributes
---     date: Integer
---   end
---
--- constraints
---
--- context Exam inv threeParticipants:
---   self.examinee<>self.examiner and
---   self.examinee<>self.recorder and
---   self.examiner<>self.recorder
---
--- context Exam inv threeParticipants2:
---   Set(self.examinee, self.examiner, self.recorder)->size() = 3

in mOdCL.maude mod CLASSES-BENCHMARK-B3 is
pr mOdCL.

----- --------------------------------------------------------------------------------------------------------

----- class Person
sort Person .
subsort Person < Cid .
op Person : -> Person [ctor] .

----- Attributes for Person
-----       name: String, civstat: String, gender: String, alive:Boolean
op name : -> AttributeName [ctor] .
op exam : -> AttributeName [ctor] .

----- association class Exam
sort Exam .
subsort Exam < Cid .
op Exam : -> Exam [ctor] .

   op examinee : -> AttributeName [ctor] .
op examiner : -> AttributeName [ctor] .
op recorder : -> AttributeName [ctor] .
op date : -> AttributeName [ctor] .
endm

----- --------------------------------------------------------------------------------------------------------

mod BENCHMARK-TEST-B3 is
inc CLASSES-BENCHMARK-B3 .

ops ada bob cyd dan eve flo : -> Oid .
ops ada-bob-cyd dan-bob-ada eve-ada-dan flo-ada-dan : -> Oid .
op state-1 : -> Configuration.

eq state-1 =
  < ada : Person | name : "Ada", exam : Set{ada-bob-cyd, dan-bob-ada, eve-ada-dan} >
  < bob : Person | name : "Bob", exam : Set{ada-bob-cyd, dan-bob-ada} >
  < cyd : Person | name : "Cyd", exam : Set{ada-bob-cyd} >
  < dan : Person | name : "Dan", exam : Set{dan-bob-ada, eve-ada-dan, flo-ada-dan} >
  < eve : Person | name : "Eve", exam : Set{eve-ada-dan} >
  < ada-bob-cyd : Exam | date : 20001023, examinee : ada, examiner : bob, recorder : cyd >
  < dan-bob-ada : Exam | date : 20040817, examinee : dan, examiner : bob, recorder : ada >

--- Including Flo

op state-2 : -> Configuration.

eq state-2 =
  < flo : Person | name : "Flo", exam : Set{flo-ada-dan} >
  < ada-bob-cyd : Exam | date : 20001023, examinee : ada, examiner : bob, recorder : cyd >
  < dan-bob-ada : Exam | date : 20040817, examinee : dan, examiner : bob, recorder : ada >
  < eve-ada-dan : Exam | date : 20080331, examinee : eve, examiner : ada, recorder : dan >

--- 1
--- use> ?Exam.allInstances->select(e|e.examinee=ada)
red eval(Exam . allInstances -> select(e | e . examinee = ada), state-1).

--- 2
--- use> ?Exam.allInstances->select(e|e.recorder=ada)
red eval(Exam . allInstances -> select(e | e . recorder = ada), state-1).

--- 3
--- use> ?Exam.allInstances->select(e|e.examiner=ada)
red eval(Exam . allInstances -> select(e | e . examiner = ada), state-1).
--- reduce in BENCHMARK-TEST-B3 : eval(Exam . allInstances -> select (e | e .
--- examiner = ada), state-1).
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Set: Set(eve-ada-dan)
---
--- Set{@eve_ada_dan} : Set(Exam)

--- 4
--- use> ?ada.examinee[examiner]
---    Set{@eve} : Set(Person)
--- NO. mOdCL does not support this syntax

--- 5
--- use> ?ada.examinee[recorder]
---    Set{@dan} : Set(Person)
--- NO. mOdCL does not support this syntax

--- 6
--- use> ?ada.examiner[examinee]
---    Set{@bob} : Set(Person)
--- NO. mOdCL does not support this syntax

--- 7
--- use> ?ada.examiner[recorder]
---    Set{@bob} : Set(Person)
--- NO. mOdCL does not support this syntax

--- 8
--- use> ?ada.recorder[examinee]
---    Set{@cyd} : Set(Person)
--- NO. mOdCL does not support this syntax

--- 9
--- use> ?ada.recorder[examiner]
---    Set{@dan} : Set(Person)
--- NO. mOdCL does not support this syntax

--- 10
--- use> !create flo:Person
--- use> !create flo_ada_dan:Exam between (flo,ada,dan)
--- use> !set flo_ada_dan.date:=20080331

--- 11
--- use> ?Exam.allInstances->select(e|e.examiner=ada)->collect(e|e.recorder)
red eval(Exam . allInstances -> select (e | e . examiner = ada) -> collect (e | e . recorder)
       , state-2).
--- reduce in BENCHMARK-TEST-B3 : eval(Exam . allInstances -> select (e | e .
---   examiner = ada) -> collect (e | e . recorder), state-2).
--- rewrites: 197 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bag: Bag{dan, dan}
--- Bag{@dan,@dan} : Bag(Person)
D Three-Valued Logic Benchmark (B4)

--- 5.1. OCL Queries

--- or

--- 1
--- use> ?true or Sequence{true} -> excluding(true) -> last()
red eval(true or Sequence{true} -> excluding(true) -> last())

--- 2
--- use> ?Sequence{true} -> excluding(true) -> last() or true
red eval(Sequence{true} -> excluding(true) -> last() or true)

--- 3
--- use> ?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)
--- WARNING. Two variables in iterate.
--- This does not comform the OCL specification
--- it should be syntactically incorrect in OCL

--- and

--- 4
--- use> ?false and Sequence{true} -> excluding(true) -> last()
red eval(false and Sequence{true} -> excluding(true) -> last())

--- result Bool: false
--- false : Boolean

--- 5
--- use> ?Sequence{true} -> excluding(true) -> last() and false
red eval(Sequence{true} -> excluding(true) -> last() and false).
--- ================================================================
--- reduce in mOdCL : eval(false and Sequence{true} -> excluding(true) -> last(---
--- )).  
--- rewrites: 18 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: false
---
--- false : Boolean

--- 6
--- use> ?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)
--- WARNING. Two variables in iterate.
--- This does not conform the OCL specification
--- it should be syntactically incorrect in OCL
--- implies

--- 7
--- use> ?false implies Sequence{true} -> excluding(true) -> last()
red eval(
false implies Sequence{true} -> excluding(true) -> last() )
--- ================================================================
--- reduce in mOdCL : eval(false implies Sequence{true} -> excluding(true) ->
--- last()) .
--- rewrites: 18 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- false : Boolean
--- WARNING. The result obtained here by mOdCL is different of the one proposed
--- in the benchmark. We think the benchmak is wrong in this test

--- 8
--- use> ?Sequence{true} -> excluding(true) -> last() implies true
red eval(Sequence{true} -> excluding(true) -> last() implies true )
--- ================================================================
--- reduce in mOdCL : eval(Sequence{true} -> excluding(true) -> last() implies
--- true) .
--- rewrites: 18 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- true : Boolean

--- 9
--- use> ?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))
--- WARNING. Two variables in iterate.
--- This does not confrom the OCL specification
--- it should be syntactically incorrect in OCL

--- if then else endif

--- 10
--- use> ?if true then false
--- else Sequence(true) -> excluding(true) -> last() endif
red eval(
  if true then false
  else Sequence(true) -> excluding(true) -> last() endif
).
--- ==============================================================
--- reduce in mOdCL : eval(if true then false else Sequence(true) -> excluding(
--- true) -> last() endif).
--- rewrites: 7 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: false
---
--- false : Boolean
--- ==============================================================

--- 11
--- use> ?if false then Sequence(true) -> excluding(true) -> last()
--- else true endif
red eval(
  if false then Sequence(true) -> excluding(true) -> last()
  else true endif
).
--- ==============================================================
--- reduce in mOdCL : eval(if false then Sequence(true) -> excluding(true) ->
--- last() else true endif).
--- rewrites: 7 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- true : Boolean
--- oclIsUndefined

--- 12
--- use> ?Sequence(true) -> excluding(true) -> last().oclIsUndefined()
red eval(
  Sequence(true) -> excluding(true) -> last() . oclIsUndefined() )
.
--- ==============================================================
--- reduce in mOdCL : eval(Sequence(true) -> excluding(true) -> last() .
--- oclIsUndefined())
--- rewrites: 24 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- true : Boolean
--- ==============================================================
--- Collection Operations
--- --- DB related
--- --- --- reject to select
--- --- --- --- Empty
--- --- --- Set
--- 1
?let c = oclEmpty(Set(Integer)) in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
  let c = Set{} in
    c -> reject(i | i < 4) = c -> select(i | not(i < 4))
) .
--- ------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> reject (i | i < 4) =
--- c -> select (i | not i < 4)) .
--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- Bag
--- 2
?let c = oclEmpty(Bag(Integer)) in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
  let c = Bag{} in
    c -> reject(i | i < 4) = c -> select(i | not(i < 4))
) .
--- ------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> reject (i | i < 4) =
--- c -> select (i | not i < 4)) .
--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- Sequence
--- 3
?let c = oclEmpty(Sequence(Integer)) in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
  let c = Sequence{} in
    c -> reject(i | i < 4) = c -> select(i | not(i < 4))
) .
--- ------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> reject (i | i <
--- 4) = c -> select (i | not i < 4)) .
--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- Singleton
---
--- --- --- Includes Undefined
---
--- --- --- Set -----------------------------------------------
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
---   c -> reject(i | i < 4) = c -> select(i | not(i < 4))
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections
---
--- --- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
---   c -> reject(i | i < 4) = c -> select(i | not(i < 4))
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections
---
--- --- --- --- Sequence
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
---   c -> reject(i | i < 4) = c -> select(i | not(i < 4))
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections
---
--- --- --- Excludes Undefined
---
--- --- --- Expression e fulfilled
---
--- --- --- Set -----------------------------------------------
--- 7
--- ?let c = Set{1} in
---   c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
   let c = Set{1} in
   c -> reject(i | i < 4) = c -> select(i | not(i < 4))
)
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1} in c -> reject (i | i < 4) =
---   c -> select (i | not i < 4)).
--- rewrites: 50 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- Bag
--- 8
--- ```
--- ?let c = Bag{1} in
---  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(  
let c = Bag{1} in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
).  
--- ========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> reject (i | i < 4) = c -> select (i | not i < 4)) .
--- rewrites: 50 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- Sequence
--- 9
--- ```
--- ?let c = Sequence{1} in
---  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(  
let c = Sequence{1} in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
).  
--- ========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> reject (i | i < 4) = c -> select (i | not i < 4)) .
--- rewrites: 50 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- Expression e not fulfilled
---
--- --- --- --- Set
--- 10
--- ```
--- ?let c = Set{4} in
---  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(  
let c = Set{4} in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
).  
--- ========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> reject (i | i < 4) = c -> select (i | not i < 4)) .
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- Bag
--- 11
--- ```
--- ?let c = Bag{4} in
---  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(  
let c = Bag{4} in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
).  
--- --------------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> reject (i | i < 4) =
--- c -> select (i | not i < 4)) .
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
  let c = Sequence{4} in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4} in c -> reject (i | i <
--- 4) = c -> select (i | not i < 4)) .
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- Many Elements

--- --- --- Includes Undefined
---
--- --- --- Set ---------------------------------------------
--- 13
--- ?let c = Set{oclUndefined(Integer), 1, 2, 3} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- Bag

--- --- --- Equal Values
--- 14
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3,
--- oclUndefined(Integer)} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- Sequence

--- --- --- --- Equal Values
--- 16
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in
---   c -> reject(i | i < 4) = c -> select(i | not(i < 4))
---
--- Expected Results:
---   true : Boolean
---  --- --- --- No Equal Values
--- 17
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in
---   c -> reject(i | i < 4) = c -> select(i | not(i < 4))
---
--- Expected Results:
---   true : Boolean
---  --- --- Excludes Undefined
---  --- --- --- Expression e fulfilled

--- --- --- --- Set ------------------------------------------
--- 18
--- ?let c = Set{1, 2, 3} in
---   c -> reject(i | i < 4) = c -> select(i | not(i < 4))
--- red eval(let c = Set{1, 2, 3} in c -> reject (i | i < 4) = c -> select (i | not i < 4)) .
---
--- Expected Results:
---   true : Boolean
---  --- --- --- Bag
---  --- --- --- Equal Values
--- 19
--- ?let c = Bag{1, 2, 3, 1} in
---   c -> reject(i | i < 4) = c -> select(i | not(i < 4))
--- red eval(let c = Bag{1, 2, 3, 1} in c -> reject (i | i < 4) = c -> select (i | not i < 4)) .
---
--- Expected Results:
---   true : Boolean
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- No Equal Values
--- 20
--- ?let c = Bag{1, 2, 3} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval( let c = Bag{1, 2, 3} in
    c -> reject(i | i < 4) = c -> select(i | not(i < 4))
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3} in c -> reject (i |
--- i < 4) = c -> select (i | not i < 4)) .
--- rewrites: 108 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- Sequence
--- --- --- --- Equal Values
--- 21
--- ?let c = Sequence{1, 2, 3, 1} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval( let c = Sequence{1, 2, 3, 1} in
    c -> reject(i | i < 4) = c -> select(i | not(i < 4))
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 1} in c -> reject
--- (i | i < 4) = c -> select (i | not i < 4)) .
--- rewrites: 137 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- No Equal Values
--- 22
--- ?let c = Sequence{1, 2, 3} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval( let c = Sequence{1, 2, 3} in
    c -> reject(i | i < 4) = c -> select(i | not(i < 4))
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3} in c -> reject (i |
--- i < 4) = c -> select (i | not i < 4)) .
--- rewrites: 108 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- Expression e not fulfilled
--- 72
--- Set ------------------------------------------
--- 23
--- ?let c = Set{2, 3, 4, 5} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
  let c = Set{2, 3, 4, 5} in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{2, 3, 4, 5} in c -> reject (i | i < 4) = c -> select (i | not i < 4)).
--- rewrites: 145 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- Bag
---
--- Equal Values (both violating)
--- 24
--- ?let c = Bag{2, 3, 4, 5, 4} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
  let c = Bag{2, 3, 4, 5, 4} in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 4, 5} in c -> reject (i | i < 4) = c -> select (i | not i < 4)).
--- rewrites: 178 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- Equal Values (both not violating)
--- 25
--- ?let c = Bag{2, 3, 4, 5, 2} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
  let c = Bag{2, 3, 4, 2, 5} in
  c -> reject(i | i < 4) = c -> select(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 2, 5} in c -> reject (i | i < 4) = c -> select (i | not i < 4)).
--- rewrites: 174 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- No Equal Values
--- 26
--- ?let c = Bag{2, 3, 4, 5} in
--- c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(}
let c = Bag{2, 3, 4, 5} in
c -> reject(i | i < 4) = c -> select(i | not(i < 4))
)
.

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5} in c -> reject (i | i < 4) = c -> select (i | not i < 4)) .
--- rewrites: 145 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Equal Values (both violating)
--- 27
--- ?let c = Sequence{2, 3, 4, 5, 4} in
c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
let c = Sequence{2, 3, 4, 5, 4} in
c -> reject(i | i < 4) = c -> select(i | not(i < 4))
)
.

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 4} in c -> reject (i | i < 4) = c -> select (i | not i < 4)) .
--- rewrites: 178 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?let c = Sequence{2, 3, 4, 5, 2} in
c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
let c = Sequence{2, 3, 4, 5, 2} in
c -> reject(i | i < 4) = c -> select(i | not(i < 4))
)
.

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 2} in c -> reject (i | i < 4) = c -> select (i | not i < 4)) .
--- rewrites: 174 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- No Equal Values
--- 29
--- ?let c = Sequence{2, 3, 4, 5} in
c -> reject(i | i < 4) = c -> select(i | not(i < 4))
red eval(
let c = Sequence{2, 3, 4, 5} in
c -> reject(i | i < 4) = c -> select(i | not(i < 4))
)
.

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5} in c -> reject (i | i < 4) = c -> select (i | not i < 4)) .
--- Collection Operations

--- --- DB related

--- --- --- select to reject

--- --- --- --- Empty

--- --- --- --- --- Set -----------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Set{} in
  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> select (i | i < 4) =
--- c -> reject (i | not i < 4)) .
--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Bag{} in
  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> select (i | i < 4) =
--- c -> reject (i | not i < 4)) .
--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Sequence{} in
c -> select(i | i < 4) = c -> reject(i | not(i < 4))

--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{} in c -> select (i | i <
--- 4) = c -> reject (i | not i < 4)).
--- rewrites: 22 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- Singleton
---
--- --- --- Includes Undefined
---
--- --- --- Set -----------------------------------------------
---
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
---
--- NO. invalid or null in collections
---
--- --- --- Bag
---
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
---
--- NO. invalid or null in collections
---
--- --- --- Sequence
---
--- 6
--- ?let c = Sequence{oclUndefined(Integer)}
--- in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
---
--- NO. invalid or null in collections
---
--- --- --- Excludes Undefined
---
--- --- --- Expression e fulfilled
---
--- --- --- --- Set ------------------------------------------
---
--- 7
--- ?let c = Set{1} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
let c = Set{1} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{1} in c -> select (i | i < 4) =
--- c -> reject (i | not i < 4)).
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
---
--- 76
--- Expected Results:
--- true : Boolean

--- --- --- Bag
--- 8
--- ?let c = Bag{1} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Bag{1} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> select (i | i < 4) =
--- c -> reject (i | not i < 4)).
--- rewrites: 54 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Sequence{1} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> select (i | i <
--- 4) = c -> reject (i | not i < 4)).
--- rewrites: 54 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- Expression e not fulfilled

--- --- --- Set ------------------------------------------
--- 10
--- ?let c = Set{4} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Set{4} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> select (i | i < 4) =
--- c -> reject (i | not i < 4)).
--- rewrites: 50 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- Bag
--- 11
--- ?let c = Bag{4} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Bag{4} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> select (i | i < 4) =
--- c -> reject (i | not i < 4)).
---
--- Expected Results:
--- true : Boolean

77
--- Expected Results:
  --- true : Boolean

--- --- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Sequence{4} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
) .
--- =========================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{4} in c -> select (i | i < 4) = c -> reject (i | not i < 4)) .
--- rewrites: 50 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
  --- true : Boolean

--- --- Many Elements

--- --- --- Includes Undefined

--- --- --- Set ---------------------------------------------
--- 13
--- ?let c = Set{oclUndefined(Integer), 1, 2, 3} in
---  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
---  true : Boolean
---  NO. invalid or null in collections

--- --- Bag

--- --- --- Equal Values
--- 14
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in
---  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
---  true : Boolean
---  NO. invalid or null in collections

--- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3} in
---  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
---  true : Boolean
---  NO. invalid or null in collections

--- --- --- Sequence

--- --- --- --- Equal Values
--- 16
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- No Equal Values
--- 17
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- Excludes Undefined
--- --- --- Expression e fulfilled
--- --- --- --- Set ------------------------------------------
--- 18
--- ?let c = Set{1, 2, 3} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- --- --- --- Bag
--- --- --- --- Equal Values
--- 19
--- ?let c = Bag{1, 2, 3, 1} in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- --- --- --- No Equal Values
--- 20
--- ?let c = Bag{1, 2, 3} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- No Equal Values
--- 21
--- ?let c = Bag{1, 2, 3} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
let c = Bag{1, 2, 3} in c -> select(i | i < 4) = c -> reject(i | not(i < 4))

--- rewrites: 120 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- Sequence
--- --- --- --- Equal Values
--- 21
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
---
--- rewrites: 153 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- No Equal Values
--- 22
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
---
--- rewrites: 120 in 4ms cpu (0ms real) (29992 rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- Expression e not fulfilled
--- --- --- --- Set
--- 23
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
---
--- rewrites: 120 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- i | i < 4 = c -> reject (i | not i < 4) .
--- rewrites: 145 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Bag
---
--- --- --- --- Equal Values (both violating)
--- 24
--- ?let c = Bag{2, 3, 4, 5, 4} in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Bag{2, 3, 4, 5, 4} in
  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5, 4} in c ->
--- select (i | i < 4) = c -> reject (i | not i < 4)) .
--- rewrites: 174 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Equal Values (both not violating)
--- 25
--- ?let c = Bag{2, 3, 4, 5, 2} in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Bag{2, 3, 4, 5, 2} in
  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5, 2} in c ->
--- select (i | i < 4) = c -> reject (i | not i < 4)) .
--- rewrites: 175 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- No Equal Values
--- 26
--- ?let c = Bag{2, 3, 4, 5} in
--- c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
  let c = Bag{2, 3, 4, 5} in
  c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5} in c -> select (i | i < 4) = c -> reject (i | not i < 4)) .
--- rewrites: 145 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
81
--- --- --- --- Sequence
--- --- --- --- Equal Values (both violating)
--- 27
--- ?let c = Sequence{2, 3, 4, 5, 4} in
c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
    let c = Sequence{2, 3, 4, 5, 4} in
c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{2, 3, 4, 5, 4} in c ->
    select(i | i < 4) = c -> reject(i | not(i < 4)) .
--- rewrites: 174 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
--- --- --- --- Equal Values (both not violating)
--- 28
--- ?let c = Sequence{2, 3, 4, 5, 2} in
c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
    let c = Sequence{2, 3, 4, 5, 2} in
c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{2, 3, 4, 5, 2} in c ->
    select(i | i < 4) = c -> reject(i | not(i < 4)) .
--- rewrites: 178 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
--- --- --- --- No Equal Values
--- 29
--- ?let c = Sequence{2, 3, 4, 5} in
c -> select(i | i < 4) = c -> reject(i | not(i < 4))
red eval(
    let c = Sequence{2, 3, 4, 5} in
c -> select(i | i < 4) = c -> reject(i | not(i < 4))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{2, 3, 4, 5} in c -> select
    (i | i < 4) = c -> reject(i | not(i < 4)) .
--- rewrites: 145 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- File: p04_e03_exists2forAll.maude
--- Collection Operations

--- --- Logic

--- --- --- exists to forAll

--- --- --- --- Empty

--- --- --- --- Set -----------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
---   c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval(
  let c = Set{} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- ================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> exists (i | i < 4) =
---   (not c -> forAll (i | not i < 4))) .
--- rewrites: 25 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
---   c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval(
  let c = Bag{} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> exists (i | i < 4) =
---   (not c -> forAll (i | not i < 4))) .
--- rewrites: 25 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
---   c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval(
  let c = Sequence{} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> exists (i | i <
--- 4) = (not c -> forAll (i | not i < 4))) .
--- rewrites: 25 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- Singleton
--- --- --- --- Includes Undefined
--- --- --- --- --- Set ---------------------------------------------
--- 4
--- ?let c = Set{6oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = not c -> forall(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
---
--- NO. invalid or null in collections

--- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = not c -> forall(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- Sequence
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = not c -> forall(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- Excludes Undefined
--- --- --- Expression e fulfilled
--- --- --- --- Set ---------------------------------------------
--- 7
--- ?let c = Set{1} in
--- c -> exists(i | i < 4) = not c -> forall(i | not(i < 4))
red eval(
let c = Set{1} in
  c -> exists(i | i < 4) = (not c -> forall(i | not(i < 4)))
) .
--- -----------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1} in c -> exists (i | i < 4) =
--- . (not c -> forall (i | not i < 4))) .
--- rewrites: 45 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
--- c -> exists(i | i < 4) = not c -> forall(i | not(i < 4))
red eval(
let c = Bag{1} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- ⊓⊔
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> exists (i | i < 4) =
  (not c -> forAll (i | not i < 4))) .
--- rewrites: 45 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
---  c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval(
let c = Sequence{1} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- ⊓⊔
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> exists (i | i <
  4) = (not c -> forAll (i | not i < 4))) .
--- rewrites: 45 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Expression e not fulfilled
--- --- --- --- Set ------------------------------------------
--- 10
--- ?let c = Set{4} in
---  c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval(
let c = Set{4} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- ⊓⊔
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> exists (i | i < 4) =
  (not c -> forAll (i | not i < 4))) .
--- rewrites: 53 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
---  c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval(
let c = Bag{4} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- ⊓⊔
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> exists (i | i < 4) =
  (not c -> forAll (i | not i < 4))) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Sequence
---
--- ?let c = Sequence{4} in
---  c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval(
   let c = Sequence{4} in
   c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
)
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4} in c -> exists (i | i < 4) = (not c -> forAll (i | not i < 4))) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- Many Elements
--- --- --- Includes Undefined
--- --- --- Set ---------------------------------------------
---
--- ?let c = Set{oclUndefined(Integer), 2, 3, 4} in
--- c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- Bag
--- --- --- --- Equal Values
---
--- ?let c = Bag{oclUndefined(Integer), 2, 3, 4,
--- oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- No Equal Values
---
--- ?let c = Bag{oclUndefined(Integer), 2, 3, 4} in
--- c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- Sequence
Equal Values

?let c = Sequence{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in
c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))

Expected Results:
true : Boolean
NO. invalid or null in collections

No Equal Values

?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in
c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))

Expected Results:
true : Boolean
NO. invalid or null in collections

Excludes Undefined

Expression e fulfilled

Set

?let c = Set{1, 2, 3, 4} in
c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))

Expected Results:
true : Boolean

Bag

?let c = Bag{1, 2, 3, 4, 1} in
c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))

Expected Results:
true : Boolean
--- No Equal Values ---

21

?-let c = Bag{1, 2, 3, 4} in
   c -> exists(i | i < 4) = not c -> forall(i | not(i < 4))
red eval(
   let c = Bag{1, 2, 3, 4} in
   c -> exists(i | i < 4) = (not c -> forall(i | not(i < 4)))
).

=========================================
reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3, 4} in c -> exists (i | i < 4) = (not c -> forall (i | not i < 4))) .
rewrites: 57 in Oms cpu (Oms real) (~ rewrites/second)
result Bool: true

--- Expected Results:
true : Boolean

--- Sequence ---

--- Equal Values ---

22

?-let c = Sequence{1, 2, 3, 4, 1} in
   c -> exists(i | i < 4) = not c -> forall(i | not(i < 4))
red eval(
   let c = Sequence{1, 2, 3, 4, 1} in
   c -> exists(i | i < 4) = (not c -> forall(i | not(i < 4)))
).

=========================================
reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 4, 1} in c -> exists (i | i < 4) = (not c -> forall (i | not i < 4))) .
rewrites: 61 in Oms cpu (Oms real) (~ rewrites/second)
result Bool: true

--- Expected Results:
true : Boolean

--- No Equal Values ---

23

?-let c = Sequence{1, 2, 3, 4} in
   c -> exists(i | i < 4) = not c -> forall(i | not(i < 4))
red eval(
   let c = Sequence{1, 2, 3, 4} in
   c -> exists(i | i < 4) = (not c -> forall(i | not(i < 4)))
).

=========================================
reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 4} in c -> exists (i | i < 4) = (not c -> forall (i | not i < 4))) .
rewrites: 57 in Oms cpu (Oms real) (~ rewrites/second)
result Bool: true

--- Expected Results:
true : Boolean

--- Expression e not fulfilled ---

--- Set ---

24

?-let c = Set{4, 5, 6} in
   c -> exists(i | i < 4) = not c -> forall(i | not(i < 4))
red eval(
let c = Set{4, 5, 6} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- usleep
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4, 5, 6} in c -> exists (i | i < 4) = (not c -> forAll (i | not i < 4)))
--- rewrites: 111 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- Bag
---
--- --- --- --- Equal Values (both violating)
--- 25
--- ?let c = Bag{4, 5, 6, 4} in
---   c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval(
let c = Bag{4, 5, 6, 4} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- usleep
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4, 5, 6, 4} in c -> exists (i | i < 4) = (not c -> forAll (i | not i < 4)))
--- rewrites: 140 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- No Equal Values
--- 26
--- ?let c = Bag{4, 5, 6} in
---   c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval(
let c = Bag{4, 5, 6} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- usleep
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4, 5, 6} in c -> exists (i | i < 4) = (not c -> forAll (i | not i < 4)))
--- rewrites: 111 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- Sequence
---
--- --- --- --- Equal Values (both violating)
--- 27
--- ?let c = Sequence{4, 5, 6, 4} in
---   c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval(
let c = Sequence{4, 5, 6, 4} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).
--- usleep
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4, 5, 6, 4} in c -> exists (i | i < 4) = (not c -> forAll (i | not i < 4)))
--- rewrites: 111 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- Sequence
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4, 5, 6} in c -> exists
---. (i | i < 4) = (not c -> forAll (i | not i < 4))) .
--- rewrites: 140 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- No Equal Values
--- 28
--- ?let c = Sequence{4, 5, 6} in
--- c -> exists(i | i < 4) = not c -> forAll(i | not(i < 4))
red eval
let c = Sequence{4, 5, 6} in
  c -> exists(i | i < 4) = (not c -> forAll(i | not(i < 4)))
).---
--- Collection Operations
--- --- Logic
--- --- --- forAll to exists
--- --- --- --- Empty
--- --- --- --- --- Set ------------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval
let c = Set{} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).---
--- Expected Results:
--- true : Boolean

File: p04_e04_forAll2exists.maude

--- Collection Operations
--- --- Logic
--- --- --- forAll to exists
--- --- --- --- Empty
--- --- --- --- --- Set ------------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval
let c = Set{} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).---
--- Expected Results:
--- true : Boolean

--- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
let c = Bag{} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> forAll (i | i < 4) =
--- . (not c -> exists (i | not i < 4))) .
--- rewrites: 25 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
---   c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
let c = Sequence{} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> forAll (i | i <
--- 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 25 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- Singleton
--- --- --- --- --- Includes Undefined
--- --- --- --- --- --- Set
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
---   c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
---
--- Expected Results:
---  true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
---   c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
---
--- Expected Results:
---  true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Sequence
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
---   c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined

--- --- --- --- --- Expression e fulfilled

--- --- --- --- --- --- Set

--- 7
--- ?let c = Set{1} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
    let c = Set{1} in
    c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1} in c -> forAll (i | i < 4) =
--- . (not c -> exists (i | not i < 4))) .
--- rewrites: 53 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- --- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Bag

--- 8
--- ?let c = Bag{1} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
    let c = Bag{1} in
    c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> forAll (i | i < 4) =
--- . (not c -> exists (i | not i < 4))) .
--- rewrites: 53 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- --- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Sequence

--- 9
--- ?let c = Sequence{1} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
    let c = Sequence{1} in
    c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> forAll (i | i <
--- 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 53 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- --- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 10
--- ?let c = Set{4} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
let c = Set{4} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> forAll (i | i < 4) =
--- . (not c -> exists (i | not i < 4))) .
--- rewrites: 45 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
let c = Bag{4} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> forAll (i | i < 4) =
--- . (not c -> exists (i | not i < 4))) .
--- rewrites: 45 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
let c = Sequence{4} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4} in c -> forAll (i | i <
--- 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 45 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Many Elements
--- --- --- --- Includes Undefined
--- --- --- --- --- --- Set ------------------------------------------
--- 13
--- ?let c = Set{oclUndefined(Integer), 1, 2, 3} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
  let c = Set{null, 1, 2, 3} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{null, 1, 2, 3} in c ->
--- forAll (i | i < 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 136 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: false
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- Bag

--- --- --- --- --- --- --- Equal Values
--- 14
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3,
---   oclUndefined(Integer)} in
---  c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
---
--- Expected Results:
---  true : Boolean
---  NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3} in
---  c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
---
--- Expected Results:
---  true : Boolean
---  NO. invalid or null in collections

--- --- --- --- --- --- --- Sequence

--- --- --- --- --- --- --- Equal Values
--- 16
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3,
---   oclUndefined(Integer)} in
---  c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
---
--- Expected Results:
---  true : Boolean
---  NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 16
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in
---  c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
---
--- Expected Results:
---  true : Boolean
---  NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined
--- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- Set ------------------------------------------
--- 17
--- ?let c = Set{1, 2, 3} in
c -> forall(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
let c = Set{1, 2, 3} in
c -> forall(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 3} in c -> forall (i |
i < 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 111 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values
--- 18
--- ?let c = Bag{1, 2, 3, 1} in
c -> forall(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
let c = Bag{1, 2, 3, 1} in
c -> forall(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 1, 3} in c -> forall (i |
i < 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 140 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- No Equal Values
--- 19
--- ?let c = Bag{1, 2, 3} in
c -> forall(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
let c = Bag{1, 2, 3} in
c -> forall(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3} in c -> forall (i |
i < 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 111 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- --- Equal Values
--- 20
--- ?let c = Sequence{1, 2, 3, 1} in
c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))

red eval(
let c = Sequence{1, 2, 3, 1} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).

--- Expected Results:
  true : Boolean

--- --- --- --- --- --- --- --- No Equal Values
--- 21
--- ?let c = Sequence{1, 2, 3} in
--- c -> forAll(i | i < 4) = exists(i | not(i < 4))
red eval(
let c = Sequence{1, 2, 3} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).

--- Expected Results:
  true : Boolean

--- --- --- --- --- --- --- --- Expression e not fulfilled
--- 22
--- ?let c = Set{2, 3, 4, 5} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
let c = Set{2, 3, 4, 5} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).

--- Expected Results:
  true : Boolean

--- --- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 23
--- ?let c = Bag{2, 3, 4, 5, 4} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
let c = Bag{2, 3, 4, 5, 4} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
)
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 4, 5} in c ->
--- forAll (i | i < 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 111 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 24
--- ?let c = Bag{2, 3, 4, 5, 2} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
  let c = Bag{2, 3, 4, 5, 2} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5, 2} in c ->
--- forAll (i | i < 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 136 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 25
--- ?let c = Bag{2, 3, 4, 5} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
  let c = Bag{2, 3, 4, 5} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5} in c -> forAll (
--- i | i < 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 107 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 26
--- ?let c = Sequence{2, 3, 4, 5, 4} in
--- c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(
  let c = Sequence{2, 3, 4, 5, 4} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 4} in c ->
--- forAll (i | i < 4) = (not c -> exists (i | not i < 4))) .
--- rewrites: 111 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- 97
---
---  Expected Results:  
---    true : Boolean

---  ---  ---  ---  ---  ---  ---  ---  ---  Equal Values (both not violating)
---  27
---  ?let c = Sequence{2, 3, 4, 5, 2} in
---    c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(  
let c = Sequence{2, 3, 4, 5, 2} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))  
).
---  ==============================================================
---  reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 2} in c ->  
---    forAll (i | i < 4) = (not c -> exists (i | not i < 4)))  .
---  rewrites: 111 in 0ms cpu (0ms real) (~ rewrites/second)
---  result Bool: true
---  ---
---  Expected Results:  
---    true : Boolean

---  ---  ---  ---  ---  ---  ---  ---  ---  No Equal Values
---  28
---  ?let c = Sequence{2, 3, 4, 5} in
---    c -> forAll(i | i < 4) = not c -> exists(i | not(i < 4))
red eval(  
let c = Sequence{2, 3, 4, 5} in
  c -> forAll(i | i < 4) = (not c -> exists(i | not(i < 4)))  
).
---  ==============================================================
---  reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5} in c -> forAll  
---    (i | i < 4) = (not c -> exists (i | not i < 4)))  .
---  rewrites: 107 in 0ms cpu (0ms real) (~ rewrites/second)
---  result Bool: true
---  ---
---  Expected Results:  
---    true : Boolean

--- Collection Operations
---  --- Logic
---  ---  --- one to exists and forAll (extern)
---  ---  ---  --- Empty
---  ---  ---  ---  --- Set  -------------------------------------
---  1
---  ?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))
red eval(  
let c = Set{} in
  c -> one(i | i < 4) =

--- File: p04_e05_one2exists_forAllExtern.maude

--- Collection Operations
---  --- Logic
---  ---  --- one to exists and forAll (extern)
---  ---  ---  --- Empty
---  ---  ---  ---  --- Set  -------------------------------------
---  1
---  ?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))
red eval(  
let c = Set{} in
  c -> one(i | i < 4) =

--- Collection Operations
---  --- Logic
---  ---  --- one to exists and forAll (extern)
---  ---  ---  --- Empty
---  ---  ---  ---  --- Set  -------------------------------------
---  1
---  ?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))
red eval(  
let c = Set{} in
  c -> one(i | i < 4) =

98
(c -> exists(i | i < 4) and c -> forAll(x,y | x < 4 and y < 4 implies x = y))

--- Expected Results:
--- true : Boolean
---

--- --- --- --- --- Bag
--- 2
--- ?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))
---
red eval(
let c = Bag{} in
  c -> one(i | i < 4) =
    (c -> exists(i | i < 4) and c -> forAll(x,y | x < 4 and y < 4 implies x = y))
).

--- Expected Results:
--- true : Boolean
---

--- --- --- --- --- Sequence
--- 3
--- ?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))
red eval(
let c = Sequence{} in
  c -> one(i | i < 4) =
    (c -> exists(i | i < 4) and c -> forAll(x,y | x < 4 and y < 4 implies x = y))
).

--- Expected Results:
--- true : Boolean
---

--- --- --- --- --- Singleton
--- --- --- --- --- Includes Undefined
--- --- --- --- --- --- Set ---------------------------------------------
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
--- c \rightarrow one(i \mid i < 4) =
--- (c \rightarrow exists(i \mid i < 4) and c \rightarrow forAll(x,y \mid x < 4 and y < 4 implies x = y))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
--- c \rightarrow one(i \mid i < 4) =
--- (c \rightarrow exists(i \mid i < 4) and c \rightarrow forAll(x,y \mid x < 4 and y < 4 implies x = y))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
--- c \rightarrow one(i \mid i < 4) =
--- (c \rightarrow exists(i \mid i < 4) and c \rightarrow forAll(x,y \mid x < 4 and y < 4 implies x = y))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- 7
--- ?let c = Set{1} in
--- c \rightarrow one(i \mid i < 4) =
--- (c \rightarrow exists(i \mid i < 4) and c \rightarrow forAll(x,y \mid x < 4 and y < 4 implies x = y))
---
--- Expected Results:
--- true : Boolean
---
--- 8
--- ?let c = Bag{1} in
--- c \rightarrow one(i \mid i < 4) =
--- (c \rightarrow exists(i \mid i < 4) and c \rightarrow forAll(x,y \mid x < 4 and y < 4 implies x = y))
---
--- Expected Results:
--- true : Boolean
red eval(
  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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> one (i | i < 4) = (c
---    -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i <
---    4)))
--- rewrites: 97 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 9
--- ?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))
red eval(
  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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> one (i | i < 4) =
---    (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i <
---        4)))
--- rewrites: 96 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 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))
red eval(
  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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> one (i | i < 4) = (c
---    -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i <
---    4)))
--- rewrites: 99 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
--- ?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))
red eval(
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))
).  
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> one (i | i < 4) = (c
---   -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i <
---   4)) ).
--- rewrites: 99 in Oms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 12
--- ?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))
red eval(
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))
).  
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4} in c -> one (i | i < 4)
---   = (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i |
---   i < 4)).
--- rewrites: 98 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- Many Elements
---
--- --- --- --- --- --- --- Includes Undefined
---
--- --- --- --- --- --- --- Set ---------------------------------------------
--- 13
--- ?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))
---
--- Expected Results:
---   true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- --- Bag
---
--- --- --- --- --- --- --- Equal Values
--- 14
--- ?let c = Bag{oclUndefined(Integer), 1, 4, 5,
---   oclUndefined(Integer)} in
--- c -> one(i | i < 4) =
---
--- 102
--- (c -> exists(i | i < 4) and c -> forall(x,y | x < 4 and y < 4 implies x = y))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- --- No Equal Values
--- 15
--- ?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))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- ---- Sequence
--- --- --- --- --- --- --- Equal Values
--- 16
--- ?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))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- --- No Equal Values
--- 17
--- ?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))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- --- Set -----------------------------
--- 18
--- ?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))
red eval(
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))
) .
---.reduce in BENCHMARK-TEST-BS : eval(let c = Set{1, 4, 5} in c -> one (i | i < 4)
--- = (c -> forall (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i < 4))) .
--- rewrites: 353 in 4ms cpu (1ms real) (88250 rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
---
--- --- --- --- --- --- --- --- Equal Values
---
--- 19
--- ?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))
red eval(
  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))
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 4, 5, 4} in c -> one (i | i < 4) = (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i < 4))) .
--- rewrites: 556 in Oms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
---
--- 20
--- ?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))
red eval(
  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))
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 4, 5} in c -> one (i | i < 4) = (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i < 4))) .
--- rewrites: 353 in Oms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- --- --- Equal Values
---
--- 21
--- ?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))
red eval(
  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))
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 4, 5, 4} in c -> one (i | i < 4) = (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i < 4))) .
--- rewrites: 353 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 4, 5, 4} in c -> one (i | i < 4) = (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i < 4))) .
--- rewrites: 552 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- No Equal Values
--- 22
--- ?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))
red eval( 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))
).
--- --------------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 4, 5} in c -> one (i | i < 4) = (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i < 4))) .
--- rewrites: 350 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- --- Set ------------------------------------------------------
--- 23
--- ?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))
red eval( let c = Set{1, 2, 5} in
c -> one(i | i < 4) =
   (c -> exists(i | i < 4) and c -> forAll(x | c -> forAll (y | x < 4 and y < 4 implies x = y))
c -> forAll(y | x < 4 and y < 4 implies x = y)))
).
--- --------------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 5} in c -> one (i | i < 4) = (c -> forAll (x, c -> forAll (y | x < 4 and y < 4 implies x = y)) and c
--- exists (i | i < 4)) .
--- rewrites: 129 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values (violating)
--- 24
--- ?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))
red eval(
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))
).
---=================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 5, 1} in c -> one (i | i < 4) =
--- (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i < 4))).
--- rewrites: 136 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (not violating)
--- 25
--- ?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))
red eval(
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))
).
---=================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 5, 5} in c -> one (i | i < 4) =
--- (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i < 4))).
--- rewrites: 136 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 26
--- ?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))
red eval(
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))
).
---=================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 5} in c -> one (i | i < 4) =
--- (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists (i | i < 4))).
--- rewrites: 131 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 27
--- ?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))
red eval(
    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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5, 1} in c -> one (i
---   | i < 4) = (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c ->
---   exists (i | i < 4))).
--- rewrites: 132 in Oms cpu (Oms real) (^ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?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))
red eval(
    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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5, 5} in c -> one (i
---   | i < 4) = (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c ->
---   exists (i | i < 4))).
--- rewrites: 132 in Oms cpu (Oms real) (^ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- No Equal Values
--- 29
--- ?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))
red eval(
    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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5} in c -> one (i | i
---   < 4) = (c -> forAll (x, y | x < 4 and y < 4 implies x = y) and c -> exists
---   . (i | i < 4))).
--- rewrites: 128 in Oms cpu (Oms real) (^ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 107
--- Collection Operations

--- --- Logic

--- --- --- one to exists and forAll (intern)

--- --- --- --- Empty

--- --- --- --- --- Set ---------------------------------------------

--- 1
--- ?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))
red eval(
let c = Set{} in
  c -> one(i | i < 4) =
    c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
).

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> one (i | i < 4) = c
--- -> exists (x | x < 4 and c -> forAll (y | y < 4 implies x = y))).

--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true

--- Expected Results:
--- true : Boolean

--- --- --- --- --- Bag

--- 2
--- ?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))
red eval(
let c = Bag{} in
  c -> one(i | i < 4) =
    c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
).

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> one (i | i < 4) = c
--- -> exists (x | x < 4 and c -> forAll (y | y < 4 implies x = y))).

--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true

--- Expected Results:
--- true : Boolean

--- --- --- --- --- Sequence

--- 3
--- ?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))
red eval(
let c = Sequence{} in
  c -> one(i | i < 4) =
    c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
red eval(let c = Sequence{} in c -> one (i | i < 4) = c -> exists (x | x < 4 and c -> forAll (y | y < 4 implies x = y))) .
--- Expected Results:
--- true : Boolean
--- --- --- --- Singleton
--- --- --- --- Includes Undefined
--- --- --- --- --- Set ---------------------------------------------
--- 4
--- ?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))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- --- Bag
--- 5
--- ?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))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- --- Sequence
--- 6
--- ?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))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- --- Excludes Undefined
--- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- Set ------------------------------------------
--- 7
--- ?let c = Set{1} in
--- c -> one(i | i < 4) =
--- c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
--- red eval(
--- let c = Set{1} in
--- c -> one(i | i < 4) =
--- c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
--- ) .
--- ==================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1} in c -> one (i | i < 4) = c
--- -> exists (x | x < 4 and c -> forAll (y | y < 4 implies x = y))) .
--- rewrites: 71 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
--- c -> one(i | i < 4) =
--- c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
red eval(
let c = Bag{1} in
   c -> one(i | i < 4) =
       c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> one (i | i < 4) = c
--- -> exists (x | x < 4 and c -> forAll (y | y < 4 implies x = y))) .
--- rewrites: 71 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
--- c -> one(i | i < 4) =
--- c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
red eval(
let c = Sequence{1} in
   c -> one(i | i < 4) =
       c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> one (i | i < 4)
--- = c -> exists (x | x < 4 and c -> forAll (y | y < 4 implies x = y))) .
--- rewrites: 71 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Expression e not fulfilled

--- --- --- --- --- --- Set
--- 10
--- ?let c = Set{4} in
--- c -> one(i | i < 4) =
--- c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
red eval(
let c = Set{4} in
   c -> one(i | i < 4) =
       c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
).
--- ==============================================================

110
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> one (i | i < 4) = c
--- -> exists (x | x < 4 and c -> forAll (y | y < 4 implies x = y))) .
--- rewrites: 73 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
--- c -> one(i | i < 4) =
--- c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
red eval(
  let c = Bag{4} in
  c -> one(i | i < 4) =
  c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
).
--- ---------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> one (i | i < 4) = c
--- -> exists (x | x < 4 and c -> forAll (y | y < 4 implies x = y))) .
--- rewrites: 73 in 4ms cpu (0ms real) (18250 rewrites/second)
--- result Bool: true

--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in
--- c -> one(i | i < 4) =
--- c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
red eval(
  let c = Sequence{4} in
  c -> one(i | i < 4) =
  c -> exists(x | x < 4 and c -> forAll(y | y < 4 implies x = y))
).
--- ---------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4} in c -> one (i | i < 4)
--- = c -> exists (x | x < 4 and c -> forAll (y | y < 4 implies x = y))) .
--- rewrites: 73 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

--- Expected Results:
--- true : Boolean

--- --- --- --- Many Elements
--- --- --- --- Includes Undefined

--- --- --- --- Set ---------------------------------------------
--- 13
--- ?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))
--- Expected Results:
--- true : Boolean
Bag

Equal Values

?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))

Expected Results:
  true : Boolean
  NO. invalid or null in collections

No Equal Values

?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))

Expected Results:
  true : Boolean
  NO. invalid or null in collections

Sequence

Equal Values

?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))

Expected Results:
  true : Boolean
  NO. invalid or null in collections

No Equal Values

?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))

Expected Results:
  true : Boolean
  NO. invalid or null in collections

Excludes Undefined

Expression e fulfilled

Set

?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))

red eval(
  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))
)
--- expected results:
--- true : boolean
--- --- --- --- --- --- --- no equal values
--- 20
--- ?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))
red eval(
  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))
) .
--- expected results:
--- true : boolean
--- --- --- --- --- --- --- sequence
--- --- --- --- --- --- --- equal values
--- 21
--- ?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))
red eval(
  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))
) .
--- expected results:
--- true : boolean
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))
).
-- =========================================
-- reduce in BENCHMARK-TEST-B5 : eval(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))) .
-- rewrites: 170 in Oms cpu (Oms real) (~ rewrites/second)
-- result Bool: true
--
-- Expected Results:
--   true : Boolean
--
-- --- --- --- --- --- --- --- No Equal Values
-- 22
-- ?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))
red eval(
  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))
).
-- =========================================
-- reduce in BENCHMARK-TEST-B5 : eval(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)))
--
-- rewrites: 137 in Oms cpu (Oms real) (~ rewrites/second)
-- result Bool: true
--
-- Expected Results:
--   true : Boolean
--
-- --- --- --- --- --- --- --- Expression e not fulfilled
--
-- --- --- --- --- --- --- --- Set ------------------------------------------
-- 23
-- ?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))
red eval(
  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))
).
-- =========================================
-- reduce in BENCHMARK-TEST-B5 : eval(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))) .
-- rewrites: 170 in Oms cpu (Oms real) (~ rewrites/second)
-- result Bool: true
--
-- Expected Results:
--   true : Boolean
--
-- --- --- --- --- --- --- --- Bag
-- --- --- --- --- --- --- --- Equal Values (violating)
-- 24
--- ?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))
red eval(
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)) 
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 1, 5} in c -> one (i | i < 4) = c -> exists (x | x < 4 and c -> forAll (y | y < 4 implies x = y)))
--- rewrites: 261 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (not violating)
--- 25
--- ?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))
red eval(
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)) 
).
--- reduce in BENCHMARK-TEST-B5 : eval(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)))
--- rewrites: 207 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 26
--- ?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))
red eval(
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)) 
).
--- reduce in BENCHMARK-TEST-B5 : eval(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)))
--- rewrites: 170 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
Equal Values (both violating)

?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))
red eval(
  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))
).
--- reduce in BENCHMARK-TEST-B5 : eval(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))).
--- rewrites: 225 in Oms cpu (Oms real) (\^ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

Equal Values (both not violating)

?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))
red eval(
  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))
).
--- reduce in BENCHMARK-TEST-B5 : eval(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))).
--- rewrites: 207 in Oms cpu (Oms real) (\^ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

No Equal Values

?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))
red eval(
  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))
).
--- reduce in BENCHMARK-TEST-B5 : eval(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)))
--- rewrites: 170 in Oms cpu (Oms real) (\^ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- Collection Operations
--- --- Interdisciplinary
--- --- --- exists to reject
--- --- --- --- Empty

--- --- --- --- --- Set ---------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(let c = Set{} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> exists (i | i < 4) =
--- c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 32 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(let c = Bag{} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> exists (i | i < 4) =
--- c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 32 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(let c = Sequence{} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> exists (i | i <
--- 4) = c -> reject (i | i < 4) -> size() < c -> size()).
--- rewrites: 32 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Singleton
---
--- --- --- --- --- Includes Undefined
---
--- --- --- --- --- --- Set -------------------------------------------
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- Sequence
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- --- --- Expression e fulfilled
---
--- --- --- --- --- --- --- Set -----------------------------------------
--- 7
--- ?let c = Set{1} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval( let c = Set{1} in
   c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
);
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1} in c -> exists (i | i < 4) =
--- c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 55 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
  let c = Bag{1} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
) .
--- =========================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> exists (i | i < 4) =
--- c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 55 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
  let c = Sequence{1} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
) .
--- =========================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> exists (i | i <
--- 4) = c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 55 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
---
--- --- --- --- --- --- --- Set ------------------------------------------
--- 10
--- ?let c = Set{4} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
  let c = Set{4} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
) .
--- =========================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> exists (i | i < 4) =
--- c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 63 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
  let c = Bag{4} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> exists (i | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 63 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in
---   c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
   let c = Sequence{4} in
   c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- --- Many Elements
--- --- --- --- --- Includes Undefined
--- --- --- --- --- --- --- Set
--- 13
--- ?let c = Set{oclUndefined(Integer), 2, 3, 4} in
---   c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- --- Bag
--- --- --- --- --- Equal Values
--- 14
--- ?let c = Bag{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in
---   c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 2, 3, 4} in
---   c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values
--- 16
--- ?let c = Sequence(oclUndefined(Integer), 2, 3, 4,
--- oclUndefined(Integer)) in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- No Equal Values
--- 17
--- ?let c = Sequence(oclUndefined(Integer), 2, 3, 4) in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- --- Expression e fulfilled

--- --- --- --- --- --- --- Set ------------------------------------------
--- 18
--- ?let c = Set{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
--- red eval(
--- let c = Set{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
--- ) .
---
--- Expected Results: false : Boolean

--- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values
--- 19
--- ?let c = Bag{1, 2, 3, 4, 1} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
--- red eval(
--- let c = Bag{1, 2, 3, 4, 1} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
--- ) .
---
--- Expected Results: true : Boolean

--- --- --- --- --- --- --- No Equal Values
--- 20
--- ?let c = Bag{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
let c = Bag{1, 2, 3, 4} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3, 4} in c -> exists (i | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()).
--- rewrites: 110 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
--- 21
--- ?let c = Sequence{1, 2, 3, 4, 1} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
let c = Sequence{1, 2, 3, 4, 1} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 4, 1} in c -> exists (i | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()).
--- rewrites: 131 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 22
--- ?let c = Sequence{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
let c = Sequence{1, 2, 3, 4} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 4} in c -> exists (i | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()).
--- rewrites: 113 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Expression e not fulfilled
---
--- --- --- --- --- --- --- --- Set ------------------------------------------
--- 23
--- ?let c = Set{4, 5, 6} in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
let c = Set{4, 5, 6} in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).  
--- ===========================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4, 5, 6} in c -> exists (i |
--- i < 4) = c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 127 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- --- Bag
---
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- ?let c = Bag{4, 5, 6, 4} in
---   c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
  let c = Bag{4, 5, 6, 4} in
    c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
  ).  
--- ===========================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4, 5, 4, 6} in c -> exists (i |
--- i | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 159 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 25
--- ?let c = Bag{4, 5, 6} in
---   c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
  let c = Bag{4, 5, 6} in
    c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
  ).  
--- ===========================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4, 5, 6} in c -> exists (i |
--- i | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 127 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 26
--- ?let c = Sequence{4, 5, 6, 4} in
---   c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
  let c = Sequence{4, 5, 6, 4} in
    c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
  ).  
--- ===========================================================================
--- Collection Operations

--- --- Interdisciplinary

--- --- --- exists to select

--- --- --- --- Empty

--- --- --- --- --- Set ------------------------------------------------

--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
let c = Set{} in
  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).
---=============================================

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(4, 5, 6, 4) in c -> exists
--- (i | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()).
--- rewrites: 165 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 27
--- ?let c = Sequence(4, 5, 6) in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
let c = Sequence(4, 5, 6) in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).
---=============================================

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(4, 5, 6) in c -> exists (i
--- | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 131 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- File: p04_e08_exists2select.maude

--- Collection Operations

--- --- Interdisciplinary

--- --- --- exists to select

--- --- --- --- Empty

--- --- --- --- --- Set ------------------------------------------------

--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
let c = Set{} in
  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).
---=============================================

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(4, 5, 6, 4) in c -> exists
--- (i | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()).
--- rewrites: 165 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 27
--- ?let c = Sequence(4, 5, 6) in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
let c = Sequence(4, 5, 6) in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).
---=============================================

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(4, 5, 6) in c -> exists (i
--- | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 131 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- File: p04_e08_exists2select.maude

--- Collection Operations

--- --- Interdisciplinary

--- --- --- exists to select

--- --- --- --- Empty

--- --- --- --- --- Set ------------------------------------------------

--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
let c = Set{} in
  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).
---=============================================

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(4, 5, 6, 4) in c -> exists
--- (i | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()).
--- rewrites: 165 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 27
--- ?let c = Sequence(4, 5, 6) in
--- c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
red eval(
let c = Sequence(4, 5, 6) in
  c -> exists(i | i < 4) = c -> reject(i | i < 4) -> size() < c -> size()
).
---=============================================

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(4, 5, 6) in c -> exists (i
--- | i < 4) = c -> reject (i | i < 4) -> size() < c -> size()) .
--- rewrites: 131 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- File: p04_e08_exists2select.maude
red eval(
  let c = Bag{} in
  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).

--- =========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> exists (i | i < 4) = c
---  -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 28 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
---  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
  let c = Sequence{} in
  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).

--- =========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> exists (i | i <
---  4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 28 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- Singleton
---
--- --- --- --- --- Includes Undefined
---
--- --- --- --- --- Set ---------------------------------------------
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
---  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- Expected Results:
---  true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
---  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- Expected Results:
---  true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Sequence
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
---  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- Expected Results:
---  true : Boolean
---
--- 125
--- NO. invalid or null in collections
--- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- --- Set
--- 7
--- ?let c = Set{1} in
---   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
   let c = Set{1} in
   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{1} in c -> exists (i | i < 4) =
--- c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 51 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
---   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
   let c = Bag{1} in
   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1} in c -> exists (i | i < 4) =
--- c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 51 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
---   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
   let c = Sequence{1} in
   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{1} in c -> exists (i | i <
--- 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 51 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 10
--- ?let c = Set{4} in
--- c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
 let c = Set{4} in
  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
)
--- ============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> exists (i | i < 4) =
--- c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
--- c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
 let c = Bag{4} in
  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
)
--- ============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> exists (i | i < 4) =
--- c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in
--- c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
 let c = Sequence{4} in
  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
)
--- ============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4} in c -> exists (i | i <
--- 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- Many Elements

--- --- --- --- Includes Undefined

--- --- --- --- --- --- Set ---------------------------------------------
--- 13
--- ?let c = Set{oclUndefined(Integer), 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- Bag
---
--- --- --- --- --- --- Equal Values
--- 14
--- ?let c = Bag{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in
---   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 2, 3, 4} in
---   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- Equal Values
--- 16
--- ?let c = Sequence{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in
---   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- No Equal Values
--- 17
--- ?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in
---   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- Expression e fulfilled
---
--- --- --- --- --- --- Set -------------------------------------------------------------
--- 18
--- ?let c = Set{1, 2, 3, 4} in
---   c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
  let c = Set{1, 2, 3, 4} in
  c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
)
---
--- BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 3, 4} in c -> exists (i | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 100 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- Bag

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3, 1, 4} in c -> exists (i | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- Equal Values

--- 19
--- ?let c = Bag{1, 2, 3, 4, 1} in
--- c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3, 1, 4} in c ->
--- exists (i | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- No Equal Values

--- 20
--- ?let c = Bag{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3, 4} in c -> exists (i
--- | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 100 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- Sequence

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 4} in c -> exists (i
--- | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 100 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- Equal Values

--- 21
--- ?let c = Sequence{1, 2, 3, 4, 1} in
--- c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 4, 1} in c ->
--- exists (i | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---  No Equal Values
--- 22
--- ?let c = Sequence{1, 2, 3, 4} in
c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
let c = Sequence{1, 2, 3, 4} in
c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{1, 2, 3, 4} in c -> exists
 --- (i | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 100 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- Expression e not fulfilled
---
--- Set ------------------------------------------
--- 23
--- ?let c = Set{4, 5, 6} in
c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
let c = Set{4, 5, 6} in
c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{4, 5, 6} in c -> exists (i | i
 --- < 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 105 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- Bag
---
--- Equal Values (both violating)
--- 24
--- ?let c = Bag{4, 5, 6, 4} in
c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
let c = Bag{4, 5, 6, 4} in
c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{4, 5, 6, 4} in c -> exists (i
 --- | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
--- rewrites: 131 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
25

?let c = Bag{4, 5, 6} in
        c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
    let c = Bag{4, 5, 6} in
    c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).                        

========================================
reduce in BENCHMARK-TEST-BS : eval(let c = Bag{4, 5, 6} in c -> exists (i | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
rewrite: 105 in Oms cpu (Oms real) (~ rewrites/second)
result Bool: true

--- Expected Results:
--- true : Boolean

26

?let c = Sequence{4, 5, 6, 4} in
        c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
    let c = Sequence{4, 5, 6, 4} in
    c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).                        

========================================
reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{4, 5, 6, 4} in c -> exists (i | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
rewrite: 131 in Oms cpu (Oms real) (~ rewrites/second)
result Bool: true

--- Expected Results:
--- true : Boolean

27

?let c = Sequence{4, 5, 6} in
        c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
red eval(
    let c = Sequence{4, 5, 6} in
    c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
).                        

========================================
reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{4, 5, 6} in c -> exists (i | i < 4) = c -> select (i | i < 4) -> notEmpty()) .
rewrite: 105 in Oms cpu (Oms real) (~ rewrites/second)
result Bool: true

--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- No Equal Values
--- 25
--- ?let c = Bag{4, 5, 6} in
---        c -> exists(i | i < 4) = c -> select(i | i < 4) -> notEmpty()
--- Collection Operations

--- --- Interdisciplinary

--- --- --- forAll to reject

--- --- --- --- Empty

--- --- --- --- Set

--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(
  let c = Set{} in
  c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
).

--- --- --- --- Bag

--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(
  let c = Bag{} in
  c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
).

--- --- --- --- Sequence

--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(
  let c = Sequence{} in
  c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
).

--- Expected Results:
--- true : Boolean

132
--- Singleton

--- Includes Undefined

--- Set

--- 4
--- \(?let c = \{\text{oclUndefined}(\text{Integer})\} \) in
--- \(c \rightarrow \text{forall}(i \mid i < 4) = c \rightarrow \text{reject}(i \mid i < 4) \rightarrow \text{isEmpty}()\)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- Bag

--- 5
--- \(?let c = \{\text{oclUndefined}(\text{Integer})\} \) in
--- \(c \rightarrow \text{forall}(i \mid i < 4) = c \rightarrow \text{reject}(i \mid i < 4) \rightarrow \text{isEmpty}()\)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- Sequence

--- 6
--- \(?let c = \{\text{oclUndefined}(\text{Integer})\} \) in
--- \(c \rightarrow \text{forall}(i \mid i < 4) = c \rightarrow \text{reject}(i \mid i < 4) \rightarrow \text{isEmpty}()\)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- Excludes Undefined

--- Expression e fulfilled

--- Set

--- 7
--- \(?let c = \{1\} \) in
--- \(c \rightarrow \text{forall}(i \mid i < 4) = c \rightarrow \text{reject}(i \mid i < 4) \rightarrow \text{isEmpty}()\)
---
--- Expected Results:
--- true : Boolean
---

--- Bag

--- 8
--- \(?let c = \{1\} \) in
--- \(c \rightarrow \text{forall}(i \mid i < 4) = c \rightarrow \text{reject}(i \mid i < 4) \rightarrow \text{isEmpty}()\)
---
--- Expected Results:
--- true : Boolean
---

--- Bag

--- 9
--- \(?let c = \{1\} \) in
--- \(c \rightarrow \text{forall}(i \mid i < 4) = c \rightarrow \text{reject}(i \mid i < 4) \rightarrow \text{isEmpty}()\)
---
--- Expected Results:
--- true : Boolean
---

--- Sequence

--- 10
--- \(?let c = \{1\} \) in
--- \(c \rightarrow \text{forall}(i \mid i < 4) = c \rightarrow \text{reject}(i \mid i < 4) \rightarrow \text{isEmpty}()\)
---
--- Expected Results:
--- true : Boolean
---

--- Set

--- 11
--- \(?let c = \{1\} \) in
--- \(c \rightarrow \text{forall}(i \mid i < 4) = c \rightarrow \text{reject}(i \mid i < 4) \rightarrow \text{isEmpty}()\)
---
--- Expected Results:
--- true : Boolean
---

--- Bag

--- 12
--- \(?let c = \{1\} \) in
--- \(c \rightarrow \text{forall}(i \mid i < 4) = c \rightarrow \text{reject}(i \mid i < 4) \rightarrow \text{isEmpty}()\)
---
--- Expected Results:
--- true : Boolean
---

--- Sequence

--- 13
--- \(?let c = \{1\} \) in
--- \(c \rightarrow \text{forall}(i \mid i < 4) = c \rightarrow \text{reject}(i \mid i < 4) \rightarrow \text{isEmpty}()\)
---
--- Expected Results:
--- true : Boolean
---
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1} in c -> forall (i | i < 4) =
--- c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 51 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
---   c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(
   let c = Sequence{1} in
c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
).
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{1} in c -> forall (i | i <
--- 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 51 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 10
--- ?let c = Set{4} in
---   c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(
   let c = Set{4} in
c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
).
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{4} in c -> forall (i | i < 4) =
--- c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 49 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
---   c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(
   let c = Bag{4} in
c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
).
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{4} in c -> forall (i | i < 4) =
--- c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 49 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in
--- c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(
  let c = Sequence{4} in
  c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
)
---
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{4} in c -> forall (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 49 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Many Elements
---
--- --- --- --- Includes Undefined
---
--- --- --- --- Set ---------------------------------------------
--- 13
--- ?let c = Set{oclUndefined(Integer), 1, 2, 3} in
--- c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- Bag
---
--- --- --- --- Equal Values
--- 14
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3,
--- oclUndefined(Integer)} in
--- c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3} in
--- c -> forall(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- Equal Values
--- 16
--- ?let c = Sequence(oclUndefined(Integer), 1, 2, 3,  
--- oclUndefined(Integer)) in  
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()  
---
--- Expected Results:  
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- No Equal Values
--- 17
--- ?let c = Sequence(oclUndefined(Integer), 1, 2, 3) in  
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()  
---
--- Expected Results:  
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 18
--- ?let c = Set{1, 2, 3} in  
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()  
red eval(let c = Set{1, 2, 3} in  
   c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()) .
--- ================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{1, 2, 3} in c -> forAll (i | i  
--- < 4) = c -> reject (i | i < 4) -> isEmpty()) .  
--- rewrites: 103 in 0ms cpu (0ms real) (~ rewrites/second)  
--- result Bool: true
---
--- Expected Results:  
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values
--- 19
--- ?let c = Bag{1, 2, 3, 1} in  
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()  
red eval(let c = Bag{1, 2, 1, 3} in  
   c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()) .
--- ================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1, 2, 1, 3} in c -> forAll (i  
--- | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .  
--- rewrites: 129 in 0ms cpu (0ms real) (~ rewrites/second)  
--- result Bool: true
---
--- Expected Results:  
--- true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 20
--- ?let c = Bag{1, 2, 3} in
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(  
let c = Bag{1, 2, 3} in  
  c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()  
) .
--- -----------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3} in c -> forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 103 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values
--- 21
--- ?let c = Sequence{1, 2, 3, 1} in
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(  
let c = Sequence{1, 2, 3, 1} in  
  c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()  
) .
--- -----------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 1} in c -> forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 129 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 22
--- ?let c = Sequence{1, 2, 3} in
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(  
let c = Sequence{1, 2, 3} in  
  c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()  
) .
--- -----------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3} in c -> forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 103 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 23
--- ?let c = Set{2, 3, 4, 5} in
--- c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(  
let c = Set{2, 3, 4, 5} in  
) .
c -> forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty() 

--- ===------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{2, 3, 4, 5} in c -> forAll (i
--- | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 118 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- ?let c = Bag{2, 3, 4, 5, 4} in
--- c -> forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()
red eval(
let c = Bag{2, 3, 4, 5, 4} in
  c -> forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- ===------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5} in c ->
---                             forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 135 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 25
--- ?let c = Bag{2, 3, 4, 5, 2} in
--- c -> forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()
red eval(
let c = Bag{2, 3, 4, 5, 2} in
  c -> forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- ===------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 2, 5} in c ->
---                             forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 144 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 26
--- ?let c = Bag{2, 3, 4, 5} in
--- c -> forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()
red eval(
let c = Bag{2, 3, 4, 5} in
  c -> forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- ===------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5} in c -> forAll (i
--- | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 118 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- --- Equal Values (both violating)
--- 27
--- ?let c = Sequence{2, 3, 4, 5, 4} in
c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(
let c = Sequence{2, 3, 4, 5, 4} in
  c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
)
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 4} in c ->
--- forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 135 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?let c = Sequence{2, 3, 4, 5, 2} in
c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(
let c = Sequence{2, 3, 4, 5, 2} in
  c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
)
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 2} in c ->
--- forAll (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 133 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- --- --- --- --- --- --- No Equal Values
--- 29
--- ?let c = Sequence{2, 3, 4, 5} in
c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
red eval(
let c = Sequence{2, 3, 4, 5} in
  c -> forAll(i | i < 4) = c -> reject(i | i < 4) -> isEmpty()
)
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5} in c -> forAll
--- (i | i < 4) = c -> reject (i | i < 4) -> isEmpty()) .
--- rewrites: 118 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- --- --- --- --- --- ---
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- ---
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- ---
---
--- Expected Results:
--- true : Boolean
--- Collection Operations
--- --- Interdisciplinary
--- --- --- forAll to select
--- --- --- --- Empty
--- --- --- --- --- Set

--- 1
--- ?let c = oclEmpty(Set(Integer)) in
c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
  let c = Set{} in
  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> forAll (i | i < 4) = (c -> select (i | i < 4) = c)) .
--- rewrites: 26 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
  let c = Bag{} in
  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> forAll (i | i < 4) = (c -> select (i | i < 4) = c)) .
--- rewrites: 26 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
  let c = Sequence{} in
  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> forAll (i | i < 4) = (c -> select (i | i < 4) = c)) .
--- rewrites: 26 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- Singleton

--- --- --- --- Includes Undefined

--- --- --- --- --- Set -------------------------------
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
---  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
---  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- Sequence
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
---  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- Excludes Undefined

--- --- --- --- --- --- Expression e fulfilled

--- --- --- --- --- --- Set -------------------------------
--- 7
--- ?let c = Set{1} in
---  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
    let c = Bag{1} in
    c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- ==================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> forAll (i | i < 4) =
--- . (c -> select (i | i < 4) = c)) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
---  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
    let c = Sequence{1} in
    c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- ==================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> forAll (i | i <
--- 4) = (c -> select (i | i < 4) = c)) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- Expression e not fulfilled
---
--- --- --- --- --- --- Set -------------------------------------------
--- 10
--- ?let c = Set{4} in
---  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
    let c = Set{4} in
    c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- ==================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> forAll (i | i < 4) =
--- . (c -> select (i | i < 4) = c)) .
--- rewrites: 47 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
---  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
    let c = Bag{4} in
    c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- ==================================================

142
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> forAll (i | i < 4) =
--- . (c -> select (i | i < 4)) .
--- rewrites: 47 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
let c = Sequence{4} in
  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4} in c -> forAll (i | i <
--- 4) = (c -> select (i | i < 4)) .
--- rewrites: 47 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- --- Many Elements
--- --- --- --- --- Includes Undefined

--- --- --- --- --- --- --- --- --- Set ---------------------------------------------
--- 13
--- ?let c = Set{oclUndefined(Integer), 1, 2, 3} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- --- Bag

--- --- --- --- --- --- --- --- Equal Values
--- 14
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3,
--- oclUndefined(Integer)} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- Sequence
--- Equal Values
--- 16
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3,
--- oclUndefined(Integer)} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- 17
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
let c = Sequence{null, 1, 2, 3} in
  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
). 
---
--- Expected Results:
--- true : Boolean
---
--- Excludes Undefined
--- Expression e fulfilled
---
--- Set ------------------------------------------
--- 18
--- ?let c = Set{1, 2, 3} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
let c = Set{1, 2, 3} in
  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
). 
---
--- Expected Results:
--- true : Boolean
---
--- Bag
--- Equal Values
--- 19
--- ?let c = Bag{1, 2, 3, 1} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
let c = Bag{1, 2, 3, 1} in
  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
--- Expected Results:
--- true : Boolean

--- ----- ----- ----- ----- ----- ----- ----- No Equal Values
--- 20
--- ?let c = Bag{1, 2, 3} in
c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
 let c = Bag{1, 2, 3} in
c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- ----- ----- ----- ----- ----- ----- ----- Equal Values
--- 21
--- ?let c = Sequence{1, 2, 3, 1} in
c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
 let c = Sequence{1, 2, 3, 1} in
c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- ----- ----- ----- ----- ----- ----- ----- No Equal Values
--- 22
--- ?let c = Sequence{1, 2, 3} in
c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
 let c = Sequence{1, 2, 3} in
c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Expression e not fulfilled

--- --- --- --- --- --- Set ---------------------------------------------
--- 23
--- let c = Set{2, 3, 4, 5} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
  let c = Set{2, 3, 4, 5} in
  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{2, 3, 4, 5} in c -> forAll (i
--- | i < 4) = (c -> select (i | i < 4) = c)) .
--- rewrites: 118 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- let c = Bag{2, 3, 4, 5, 4} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
---
--- red eval(
  let c = Bag{2, 3, 4, 4, 5} in
  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 4, 5} in c ->
--- forAll (i | i < 4) = (c -> select (i | i < 4) = c)) .
--- rewrites: 133 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 25
--- let c = Bag{2, 3, 4, 5, 2} in
--- c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
  let c = Bag{2, 3, 4, 2, 5} in
  c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 2, 5} in c ->
--- forAll (i | i < 4) = (c -> select (i | i < 4) = c)) .
--- rewrites: 146 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- No Equal Values
--- 26
--- ?let c = Bag{2, 3, 4, 5} in
--- c -> forall(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
  let c = Bag{2, 3, 4, 5} in
  c -> forall(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5} in c -> forall (i
--- | i < 4) = (c -> select (i | i < 4) = c)).
--- rewrites: 118 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 27
--- ?let c = Sequence{2, 3, 4, 5, 4} in
--- c -> forall(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
  let c = Sequence{2, 3, 4, 5, 4} in
  c -> forall(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 4} in c ->
--- forall (i | i < 4) = (c -> select (i | i < 4) = c)).
--- rewrites: 133 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?let c = Sequence{2, 3, 4, 5, 2} in
--- c -> forall(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(
  let c = Sequence{2, 3, 4, 5, 2} in
  c -> forall(i | i < 4) = (c -> select(i | i < 4) = c)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 2} in c ->
--- forall (i | i < 4) = (c -> select (i | i < 4) = c)).
--- rewrites: 135 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- No Equal Values
--- 29
--- ?let c = Sequence{2, 3, 4, 5} in
--- c -> forall(i | i < 4) = (c -> select(i | i < 4) = c)
red eval(

147
let c = Sequence{2, 3, 4, 5} in
c -> forAll(i | i < 4) = (c -> select(i | i < 4) = c)
).

--- 
--- Collection Operations
--- --- Interdisciplinary
--- --- --- one to reject
--- --- --- --- Empty
--- --- --- --- Set -------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
  let c = Set{} in
c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
).

--- 
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5} in c -> forAll
--- . (i | i < 4) = (c -> select (i | i < 4) = c) .
--- rewrites: 118 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
  let c = Bag{} in
c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
).

--- 
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> one (i | i < 4) = (c
--- . reject (i | i < 4) -> size() = c -> size() - 1)) .
--- rewrites: 36 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Sequence{} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> one (i | i < 4) =
--- . (c -> reject (i | i < 4) -> size() = c -> size() - 1)) .
--- rewrites: 36 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Singleton
---
--- --- --- --- Includes Undefined
---
--- --- --- --- --- Set
--- --- --- --- --- Includes Undefined
--- --- 4
--- ?let c = Set{oclUndefined(Integer)} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- Bag
--- --- 5
--- ?let c = Bag{oclUndefined(Integer)} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- Sequence
--- --- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- Excludes Undefined
---
--- --- --- --- --- Expression e fulfilled
---
--- --- --- --- --- --- Set
--- --- --- --- --- --- Includes Undefined
--- --- --- --- --- --- 7
--- ?let c = Set{1} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Set{1} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
)
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1} in c \to one (i \mid i < 4) = (c
--- \to reject (i \mid i < 4) \to size() = c \to size() - 1)).
--- rewrites: 65 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
--- c \to one(i \mid i < 4) = (c \to reject(i \mid i < 4) \to size() = c \to size() - 1)
red eval(
let c = Bag{1} in
  c \to one(i \mid i < 4) = (c \to reject(i \mid i < 4) \to size() = c \to size() - 1)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c \to one (i \mid i < 4) = (c
--- \to reject (i \mid i < 4) \to size() = c \to size() - 1)).
--- rewrites: 65 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
--- c \to one(i \mid i < 4) = (c \to reject(i \mid i < 4) \to size() = c \to size() - 1)
red eval(
let c = Sequence{1} in
  c \to one(i \mid i < 4) = (c \to reject(i \mid i < 4) \to size() = c \to size() - 1)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c \to one (i \mid i < 4)
--- = (c \to reject (i \mid i < 4) \to size() = c \to size() - 1)).
--- rewrites: 65 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Expression e not fulfilled

--- --- --- --- --- --- Set
--- 10
--- ?let c = Set{4} in
--- c \to one(i \mid i < 4) = (c \to reject(i \mid i < 4) \to size() = c \to size() - 1)
red eval(
let c = Set{4} in
  c \to one(i \mid i < 4) = (c \to reject(i \mid i < 4) \to size() = c \to size() - 1)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c \to one (i \mid i < 4) = (c
--- \to reject (i \mid i < 4) \to size() = c \to size() - 1)).
--- rewrites: 67 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true

150
--- Expected Results:  
--- true : Boolean  

--- --- --- Bag
--- 11
--- ?let c = Bag{4} in  
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)  
red eval(  
  let c = Bag{4} in  
   c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)  
) .  
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{4} in c -> one(i | i < 4) = (c  
--- -> reject(i | i < 4) -> size() = c -> size() - 1)) .  
--- rewrites: 67 in 0ms cpu (0ms real) (~ rewrites/second)  
--- result Bool: true
---

--- Expected Results:  
--- true : Boolean  

--- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in  
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)  
red eval(  
  let c = Sequence{4} in  
   c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)  
) .  
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{4} in c -> one(i | i < 4) = (c  
--- -> reject(i | i < 4) -> size() = c -> size() - 1)) .  
--- rewrites: 67 in 0ms cpu (0ms real) (~ rewrites/second)  
--- result Bool: true
---

--- Expected Results:  
--- true : Boolean  

--- --- --- Many Elements
--- --- --- Includes Undefined

--- --- --- Set
--- 13
--- ?let c = Set(oclUndefined(Integer), 1, 4, 5) in  
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
---
--- Expected Results:  
--- true : Boolean  
--- NO. invalid or null in collections

--- --- --- Bag
--- 14
--- ?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)
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 1, 4, 5} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values
--- 16
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 17
--- ?let c = Sequence{oclUndefined(Integer), 1, 4, 5} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- --- Expression e fulfilled

--- --- --- --- --- --- --- Set ---------------------------------------------------
--- 18
--- ?let c = Set{1, 4, 5} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Set{1, 4, 5} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
).
--- -----------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 4, 5} in c -> one (i | i < 
--- 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)) .
--- rewrites: 129 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- Equal Values
--- 19
--- ?let c = Bag{1, 4, 5, 4} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Bag{1, 4, 5, 4} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
) .
--- =========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 4, 4, 5} in c -> one (i |
--- i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)) .
--- rewrites: 161 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- No Equal Values
--- 20
--- ?let c = Bag{1, 4, 5} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Bag{1, 4, 5} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
) .
--- =========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 4, 5} in c -> one (i |
--- i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)) .
--- rewrites: 129 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- Sequence
--- 21
--- ?let c = Sequence{1, 4, 5, 4} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Sequence{1, 4, 5, 4} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
) .
--- =========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 4, 5, 4} in c -> one (i |
--- i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)) .
--- rewrites: 166 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- No Equal Values
--- 22
--- ?let c = Sequence{1, 4, 5} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Sequence{1, 4, 5} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
) .

153
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(1, 4, 5) in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)).
--- rewrites: 132 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled

--- --- --- --- --- --- --- Set
--- 23
--- ?let c = Set{1, 2, 5} in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Set{1, 2, 5} in
c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)
).
---

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 5} in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)).
--- rewrites: 110 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Bag
--- 24
--- ?let c = Bag{1, 2, 5, 1} in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Bag{1, 2, 5, 1} in
c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)
).
---

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 1, 5} in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)).
--- rewrites: 127 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Equal Values (not violating)
--- 25
--- ?let c = Bag{1, 2, 5, 5} in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Bag{1, 2, 5, 5} in
c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)
).
---

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 5, 5} in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)).
--- rewrites: 131 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 26
--- ?let c = Bag{1, 2, 5} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
  let c = Bag{1, 2, 5} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 5} in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)).
--- rewrites: 110 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 27
--- ?let c = Sequence{1, 2, 5, 1} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
  let c = Sequence{1, 2, 5, 1} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5, 1} in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)).
--- rewrites: 130 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?let c = Sequence{1, 2, 5, 5} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
  let c = Sequence{1, 2, 5, 5} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5, 5} in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)).
--- rewrites: 135 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 29
--- ?let c = Sequence{1, 2, 5} in
--- c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
red eval(
let c = Sequence{1, 2, 5} in
  c -> one(i | i < 4) = (c -> reject(i | i < 4) -> size() = c -> size() - 1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5} in c -> one (i | i < 4) = (c -> reject (i | i < 4) -> size() = c -> size() - 1)).
--- rewrites: 112 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- Collection Operations
--- --- Interdisciplinary
--- --- --- one to select
--- --- --- --- Empty
--- --- --- --- --- Set -----------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Set{} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> one (i | i < 4) = (c -> select (i | i < 4) -> size() = 1)).
--- rewrites: 29 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Bag{} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> one (i | i < 4) = (c -> select (i | i < 4) -> size() = 1)).
--- rewrites: 29 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---

--- Expected Results:
--- true : Boolean

--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
---  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Sequence{} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- ===================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{} in c -> one (i | i < 4) =
---  . (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 29 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Singleton

--- --- --- --- Includes Undefined
--- --- --- --- --- Set -----------------------------------------------
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
---  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
---  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Sequence
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
---  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- Excludes Undefined
--- --- --- --- --- Expression e fulfilled

--- --- --- --- --- Set ---------------------------------------------
--- 7
--- ?let c = Set{1} in

157
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Set{1} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
)
--- ============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1} in c -> one (i | i < 4) = (c
--- -> select (i | i < 4) -> size() = 1))
--- rewrites: 60 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
---  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Bag{1} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
)
--- ============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> one (i | i < 4) = (c
--- -> select (i | i < 4) -> size() = 1))
--- rewrites: 60 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
---  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Sequence{1} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
)
--- ============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> one (i | i < 4)
--- = (c -> select (i | i < 4) -> size() = 1))
--- rewrites: 60 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 10
--- ?let c = Set{4} in
---  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Set{4} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
)
--- ============================================================

158
reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> one (i | i < 4) = (c -> select (i | i < 4) -> size() = 1)) .
rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true

Expected Results:
true : Boolean

--- --- --- --- --- --- Bag

11
?let c = Bag{4} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Bag{4} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
) .

Expected Results:
true : Boolean

--- --- --- --- --- --- Sequence

12
?let c = Sequence{4} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Sequence{4} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
) .

Expected Results:
true : Boolean

--- --- --- --- Many Elements

--- --- --- --- Includes Undefined

--- --- --- --- Set

13
?let c = Set{oclUndefined(Integer), 1, 4, 5} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)

Expected Results:
true : Boolean

NO. invalid or null in collections

--- --- --- --- Bag

--- --- --- --- Equal Values

14
--- ?let c = Bag{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in
---   c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections
---
--- 15
--- ?let c = Bag{oclUndefined(Integer), 1, 4, 5} in
---   c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
---
--- 16
--- ?let c = Bag{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in
---   c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections

--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values
---
--- 16
--- ?let c = Sequence{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in
---   c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
---
--- 17
--- ?let c = Sequence{oclUndefined(Integer), 1, 4, 5} in
---   c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined
--- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- Set
---
--- 18
--- ?let c = Set{1, 4, 5} in
---   c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
---
--- Expected Results:
---   true : Boolean
--- Bag
--- Equal Values
--- 19
--- let c = Bag{1, 4, 5, 4} in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Bag{1, 4, 5, 4} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- rewrites: 138 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- No Equal Values
--- 20
--- let c = Bag{1, 4, 5} in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Bag{1, 4, 5} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- rewrites: 112 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- Sequence
--- Equal Values
--- 21
--- let c = Sequence{1, 4, 5, 4} in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Sequence{1, 4, 5, 4} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- rewrites: 138 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- No Equal Values
--- 22
--- let c = Sequence{1, 4, 5} in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Sequence{1, 4, 5} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 4, 5} in c -> one (i | i
--- < 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 112 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
---
--- --- --- --- --- --- --- Set --------------------------------------------------------
--- 23
--- ?let c = Set{1, 2, 5} in
---   c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Set{1, 2, 5} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 5} in c -> one (i | i <
--- 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 101 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- --- Bag -----------------------------------------------------
--- 24
--- ?let c = Bag{1, 2, 5, 1} in
---   c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Bag{1, 2, 5, 1} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 1, 5} in c -> one (i | i
--- < 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 120 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (violating) -----------------------------------
--- 25
--- ?let c = Bag{1, 2, 5, 5} in
---   c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Bag{1, 2, 5, 5} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 1, 5} in c -> one (i | i
--- < 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 120 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (not violating) -----------------------------------
--- 25
--- ?let c = Bag{1, 2, 5, 5} in
---   c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Bag{1, 2, 5, 5} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 5, 5} in c -> one (i | i < 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 116 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- No Equal Values
--- 26
--- ?let c = Bag{1, 2, 5} in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Bag{1, 2, 5} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 5} in c -> one (i | i < 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 101 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- Sequence
--- 27
--- ?let c = Sequence{1, 2, 5, 1} in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Sequence{1, 2, 5, 1} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5, 1} in c -> one (i | i < 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 122 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?let c = Sequence{1, 2, 5, 5} in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Sequence{1, 2, 5, 5} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5, 5} in c -> one (i | i < 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- Equal Values (both violating)
--- 29
--- ?let c = Sequence{1, 2, 5, 1} in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Sequence{1, 2, 5, 1} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5, 1} in c -> one (i | i < 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 122 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- Equal Values (both violating)
--- 30
--- ?let c = Sequence{1, 2, 5, 5} in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
  let c = Sequence{1, 2, 5, 5} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5, 5} in c -> one (i | i < 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 29
--- ?let c = Sequence{1, 2, 5} in
--- c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
red eval(
let c = Sequence{1, 2, 5} in
  c -> one(i | i < 4) = (c -> select(i | i < 4) -> size() = 1)
).
--- ================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5} in c -> one (i | i
--- < 4) = (c -> select (i | i < 4) -> size() = 1)) .
--- rewrites: 102 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- Collection Operations
---
--- --- Collect
--- --- --- exists to collect and includes
--- --- --- --- Empty
--- --- --- --- --- Set ------------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Set{} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
).
--- ================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> exists (i | i < 4) = c
--- -> collect (i | i < 4) -> includes(true)) .
--- rewrites: 26 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Bag{} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
).
--- ================

File: p04.e13_exists2collect_includes.maude
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> exists (i | i < 4) = c
--- -> collect (i | i < 4) -> includes(true)) .
--- rewrites: 26 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- --- true : Boolean
---
--- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Sequence{} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> exists (i | i <
--- 4) = c -> collect (i | i < 4) -> includes(true)) .
--- rewrites: 26 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- --- true : Boolean
---
--- --- --- --- Singleton
--- --- --- --- Includes Undefined
--- --- --- --- --- Set
--- --- --- --- --- --- Bag
--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- --- Expression e fulfilled
---
--- --- --- --- --- --- --- Set ------------------------------------------
--- 7
--- ?let c = Set{1} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
  let c = Set{1} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1} in c -> exists (i | i < 4) =
  c -> collect (i | i < 4) -> includes(true)).
--- rewrites: 48 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
  let c = Bag{1} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> exists (i | i < 4) =
  c -> collect (i | i < 4) -> includes(true)).
--- rewrites: 48 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
  let c = Sequence{1} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> exists (i | i <
  4) = c -> collect (i | i < 4) -> includes(true)).
--- rewrites: 48 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 10
--- ?let c = Set{4} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Set(4) in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)

--- Expected Results:
--- true : Boolean

--- ??? --- Bag
--- 11
--- ?let c = Bag{4} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Bag{4} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
)

--- Expected Results:
--- true : Boolean

--- ??? --- Sequence
--- 12
--- ?let c = Sequence{4} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Sequence{4} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
)

--- Expected Results:
--- true : Boolean

--- ??? --- Many Elements

--- ??? --- Includes Undefined

--- ??? --- Set

--- 13
--- ?let c = Set(oclUndefined(Integer), 2, 3, 4) in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)

--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
Bag

Equal Values

?let c = Bag{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)

Expected Results:
true : Boolean
NO. invalid or null in collections

No Equal Values

?let c = Bag{oclUndefined(Integer), 2, 3, 4} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)

Expected Results:
true : Boolean
NO. invalid or null in collections

Sequence

Equal Values

?let c = Sequence{oclUndefined(Integer), 2, 3, 4, oclUndefined(Integer)} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)

Expected Results:
true : Boolean
NO. invalid or null in collections

No Equal Values

?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)

Expected Results:
true : Boolean
NO. invalid or null in collections

Excludes Undefined

Expression e fulfilled

Set

?let c = Set{1, 2, 3, 4} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval
let c = Set{1, 2, 3, 4} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
)

reduce in BENCHMARK-TEST-BS : eval(let c = Set{1, 2, 3, 4} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> includes(true))
rewrites: 99 in 0ms cpu (Oms real) (~ rewrites/second)
result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- Equal Values
--- 19
--- ?let c = Bag{1, 2, 3, 4, 1} in
---   c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Bag{1, 2, 3, 4, 1} in
   c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3, 1, 4} in c ->
---   exists (i | i < 4) = c -> collect (i | i < 4) -> includes(true)) .
--- rewrites: 116 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 20
--- ?let c = Bag{1, 2, 3, 4} in
---   c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Bag{1, 2, 3, 4} in
   c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3, 4} in c -> exists (i
---   | i < 4) = c -> collect (i | i < 4) -> includes(true)) .
--- rewrites: 99 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values
--- 21
--- ?let c = Sequence{1, 2, 3, 4, 1} in
---   c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Sequence{1, 2, 3, 4, 1} in
   c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 4, 1} in c ->
---   exists (i | i < 4) = c -> collect (i | i < 4) -> includes(true)) .
--- rewrites: 116 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- ---
--- --- --- --- --- --- --- --- No Equal Values
--- 22
--- ?let c = Sequence{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
    let c = Sequence{1, 2, 3, 4} in
    c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 4} in c -> exists
--- . (i | i < 4) = c -> collect (i | i < 4) -> includes(true)).
--- rewrites: 99 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- --- Set ------------------------------------------------------
--- 23
--- ?let c = Set{4, 5, 6} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
    let c = Set{4, 5, 6} in
    c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4, 5, 6} in c -> exists (i | i
--- < 4) = c -> collect (i | i < 4) -> includes(true)).
--- rewrites: 108 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- ?let c = Bag{4, 5, 6, 4} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
    let c = Bag{4, 5, 6, 4} in
    c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4, 5, 4, 6} in c -> exists (i
--- | i < 4) = c -> collect (i | i < 4) -> includes(true)).
--- rewrites: 136 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 25
--- ?let c = Bag{4, 5, 6} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Bag{4, 5, 6} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
).
--- =========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4, 5, 6} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> includes(true)) .
--- rewrites: 108 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values (both violating)
--- 26
--- ?let c = Sequence{4, 5, 6, 4} in
---   c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Sequence{4, 5, 6, 4} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4, 5, 6, 4} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> includes(true)) .
--- rewrites: 136 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 27
--- ?let c = Sequence{4, 5, 6} in
---   c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
red eval(
let c = Sequence{4, 5, 6} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> includes(true)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4, 5, 6} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> includes(true)) .
--- rewrites: 108 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---

File: p04_e14_exists2collect_one.maude

--- -- Collection Operations
--- --- -- Collect
--- --- --- exists to collect and one
--- --- --- --- Empty

--- --- --- --- --- Set ------------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Set{} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).
--- =========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> exists (i | i < 4) = c
--- -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 29 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Bag{} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).
--- =========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> exists (i | i < 4) = c
--- -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 29 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Sequence{} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).
--- =========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> exists (i | i <
--- 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 30 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- Singleton

--- --- --- Includes Undefined

--- --- --- --- --- Set ------------------------------------------------
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Sequence
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Excludes Undefined
--- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- Set ------------------------------------------
--- 7
--- ?let c = Set{1} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Set{1} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{1} in c -> exists (i | i < 4) =
--- c -> collect (i | i < 4) -> asSet() -> one (e | e)).
--- rewrites: 60 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Bag{1} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1} in c -> exists (i | i < 4) =
--- c -> collect (i | i < 4) -> asSet() -> one (e | e)).
--- rewrites: 60 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Sequence{1} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 62 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- Set ---------------------------------------------
--- 10
--- ?let c = Set{4} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Set{4} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 62 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- --- --- --- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Bag{4} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 62 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- --- --- --- --- --- --- Sequence

174
red eval(
  let c = Sequence{4} in 
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).

reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .

result Bool: true

Expected Results:
  true : Boolean

--- --- --- --- Many Elements

--- --- --- --- --- Includes Undefined

--- 12
--- ?let c = Sequence{4} in 
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Sequence{4} in 
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).

--- --- --- --- --- Set ---------------------------------------------

--- 13
--- ?let c = Set{oclUndefined(Integer), 2, 3, 4} in 
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)

--- Expected Results:
  true : Boolean
  NO. invalid or null in collections

--- --- --- --- --- --- Bag

--- --- --- --- --- --- Equal Values
--- 14
--- ?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)

--- Expected Results:
  true : Boolean
  NO. invalid or null in collections

--- --- --- --- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 2, 3, 4} in 
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)

--- Expected Results:
  true : Boolean
  NO. invalid or null in collections

--- --- --- --- --- --- Sequence

--- --- --- --- --- --- Equal Values
--- 16
--- ?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)

--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- --- --- --- --- No Equal Values
--- 17
--- ?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- --- Set ------------------------------------------
--- 18
--- ?let c = Set{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Set{1, 2, 3, 4} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).
--- ================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 3, 4} in c -> exists (i |
--- | i < 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 120 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values
--- 19
--- ?let c = Bag{1, 2, 3, 4, 1} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Bag{1, 2, 3, 4, 1} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).
--- ================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3, 1, 4} in c ->
--- exists (i | i < 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 138 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- No Equal Values
--- 20
--- ?let c = Bag{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Bag{1, 2, 3, 4} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3, 4} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 120 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- --- Equal Values
--- 21
--- ?let c = Sequence{1, 2, 3, 4, 1} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
--- red eval(let c = Sequence{1, 2, 3, 4, 1} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)) .
--- -----------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 4, 1} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 144 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 22
--- ?let c = Sequence{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
--- red eval(let c = Sequence{1, 2, 3, 4} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)) .
--- -----------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 4} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 125 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Expression e not fulfilled
--- 23
--- ?let c = Set{4, 5, 6} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
--- red eval(let c = Set{4, 5, 6} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)) .
--- -----------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4, 5, 6} in c -> exists (i | i < 4) = c -> collect (i | i < 4) -> asSet() -> one (e | e)) .
--- rewrites: 120 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Bag

--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- ?let c = Bag{4, 5, 6, 4} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Bag{4, 5, 6, 4} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4, 5, 4, 6} in c -> exists (i
--- | i < 4) = c -> collect (i | i < 4) -> asSet () -> one (e | e)) .
--- rewrites: 149 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- No Equal Values
--- 25
--- ?let c = Bag{4, 5, 6} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Bag{4, 5, 6} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4, 5, 6} in c -> exists (i | i
--- < 4) = c -> collect (i | i < 4) -> asSet () -> one (e | e)) .
--- rewrites: 120 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- Sequence

--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 26
--- ?let c = Sequence{4, 5, 6, 4} in
--- c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
red eval(
  let c = Sequence{4, 5, 6, 4} in
  c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4, 5, 4, 6} in c -> exists
--- . (i | i < 4) = c -> collect (i | i < 4) -> asSet () -> one (e | e)) .
--- rewrites: 154 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
true : Boolean

--- No Equal Values
--- 27
--- ?let c = Sequence{4, 5, 6} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)

red eval(
  let c = Sequence{4, 5, 6} in
c -> exists(i | i < 4) = c -> collect(i | i < 4) -> asSet() -> one(e | e)
).

--- reduce in BENCHMARK-TEST-B5: eval(let c = Sequence{4, 5, 6} in c -> exists (i
| i < 4) = c -> collect (i | i < 4) -> asSet () -> one (e | e)).
--- rewrites: 124 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- Collection Operations
--- Collect
--- forAll to collect and excludes
--- Empty
--- Set
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)

red eval(
  let c = Set{} in
c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
).

--- reduce in BENCHMARK-TEST-B5: eval(let c = Set{} in c -> forAll (i | i < 4) = c
| collect (i | i < 4) -> excludes(false)).
--- rewrites: 28 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)

red eval(
  let c = Bag{} in
c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
).

--- reduce in BENCHMARK-TEST-B5: eval(let c = Bag{} in c -> forAll (i | i < 4) = c
--- \( \rightarrow \) collect (\( i \mid i < 4 \) \( \rightarrow \) excludes(false)) .
--- rewrites: 28 in 0ms cpu (0ms real) (\~ rewrites/second)
--- result Bool: true 
--- 
--- Expected Results:
--- true : Boolean 
--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
--- c \( \rightarrow \) forall(i \mid i < 4) = c \( \rightarrow \) collect(i \mid i < 4) \( \rightarrow \) excludes(false) 
red eval(
let c = Sequence{} in 
  c \( \rightarrow \) forall(i \mid i < 4) = c \( \rightarrow \) collect(i \mid i < 4) \( \rightarrow \) excludes(false) ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c \( \rightarrow \) forall (i \mid i < 
--- 4) = c \( \rightarrow \) collect (i \mid i < 4) \( \rightarrow \) excludes(false)) .
--- rewrites: 28 in 0ms cpu (0ms real) (\~ rewrites/second)
--- result Bool: true 
--- 
--- Expected Results:
--- true : Boolean 
--- --- --- --- Singleton
--- --- --- --- --- Includes Undefined
--- --- --- --- --- --- Set ---------------------------------------------
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
--- c \( \rightarrow \) forall(i \mid i < 4) = c \( \rightarrow \) collect(i \mid i < 4) \( \rightarrow \) excludes(false) 
--- 
--- Expected Results:
--- true : Boolean 
--- --- --- --- NO. invalid or null in collections
--- --- --- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
--- c \( \rightarrow \) forall(i \mid i < 4) = c \( \rightarrow \) collect(i \mid i < 4) \( \rightarrow \) excludes(false) 
--- 
--- Expected Results:
--- true : Boolean 
--- --- --- --- NO. invalid or null in collections
--- --- --- --- --- Sequence
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
--- c \( \rightarrow \) forall(i \mid i < 4) = c \( \rightarrow \) collect(i \mid i < 4) \( \rightarrow \) excludes(false) 
--- 
--- Expected Results:
--- true : Boolean 
--- --- --- --- NO. invalid or null in collections
--- --- --- --- --- Excludes Undefined

180
--- --- --- --- --- --- Expression e fulfilled

--- --- --- --- --- --- Set

--- 7
--- ?let c = Set{1} in
---   c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Set{1} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
) .
--- ==================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{1} in c -> forAll (i | i < 4) =
---   c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- Bag

--- 8
--- ?let c = Bag{1} in
---   c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Bag{1} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
) .
--- ==================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1} in c -> forAll (i | i < 4) =
---   c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- Sequence

--- 9
--- ?let c = Sequence{1} in
---   c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Sequence{1} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
) .
--- ==================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{1} in c -> forAll (i | i <
---   4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- --- --- --- --- --- Expression e not fulfilled

--- --- --- --- --- --- Set

--- 10
--- ?let c = Set{4} in
red eval(
  let c = Set{4} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
).

--- Expected Results:
--  true : Boolean

--- --- --- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
---   c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Bag{4} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
).

--- Expected Results:
--  true : Boolean

--- --- --- --- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in
---   c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Sequence{4} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
).

--- Expected Results:
--  true : Boolean

--- --- --- --- Many Elements

--- --- --- --- Includes Undefined

--- --- --- --- Set

--- 13
--- ?let c = Set{oclUndefined(Integer), 1, 2, 3} in
---   c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)

--- Expected Results:
--  true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- Bag

--- --- --- --- --- --- --- Equal Values
--- 14
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3} in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- Sequence

--- --- --- --- --- --- --- Equal Values
--- 16
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 17
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined

--- --- --- --- --- Expression e fulfilled

--- --- --- --- --- Set ------------------------------------------
--- 18
--- ?let c = Set{1, 2, 3} in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
let c = Set{1, 2, 3} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{1, 2, 3} in c -> forAll (i | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 110 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- --- --- --- --- --- Bag
--- --- --- --- --- --- --- Equal Values
--- 19
--- ?let c = Bag{1, 2, 3, 1} in
---    c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
    let c = Bag{1, 2, 3, 1} in
    c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
) .
---sorry in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 1, 3} in c -> forAll (i | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 138 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 20
--- ?let c = Bag{1, 2, 3} in
---    c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
    let c = Bag{1, 2, 3} in
    c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
) .
---reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3} in c -> forAll (i | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 110 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values
--- 21
--- ?let c = Sequence{1, 2, 3, 1} in
---    c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
    let c = Sequence{1, 2, 3, 1} in
    c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
) .
---sorry in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 1} in c -> forAll (i | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 138 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- No Equal Values
--- 22
--- ?let c = Sequence{1, 2, 3} in
--- c -> forall(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Sequence{1, 2, 3} in
  c -> forall(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
)
--- =====================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3} in c -> forall (i
--- | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 110 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- --- Set ---------------------------------------------
--- 23
--- ?let c = Set{2, 3, 4, 5} in
--- c -> forall(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Set{2, 3, 4, 5} in
  c -> forall(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
)
--- =====================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{2, 3, 4, 5} in c -> forall (i
--- | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 123 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- ?let c = Bag{2, 3, 4, 5, 4} in
--- c -> forall(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Bag{2, 3, 4, 4, 5} in
  c -> forall(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
)
--- =====================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 4, 5} in c ->
--- forall (i | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 140 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 25
--- ?let c = Bag{2, 3, 4, 5, 2} in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Bag{2, 3, 4, 5, 2} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
).
--- ========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5, 2} in c ->
--- forAll (i | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 151 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 26
--- ?let c = Bag{2, 3, 4, 5} in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Bag{2, 3, 4, 5} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
).
--- ========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5} in c ->
--- forAll (i | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 123 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 27
--- ?let c = Sequence{2, 3, 4, 5, 4} in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Sequence{2, 3, 4, 5, 4} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
).
--- ========================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 4} in c ->
--- forAll (i | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 140 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?let c = Sequence{2, 3, 4, 5, 2} in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
  let c = Sequence{2, 3, 4, 5, 2} in
  c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
).
---

186
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(2, 3, 4, 5, 2) in c ->
--- forAll (i | i < 4) = c -> collect (i | i < 4) -> excludes(false)) .
--- rewrites: 140 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 29
--- ?let c = Sequence(2, 3, 4, 5) in
--- c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
red eval(
    let c = Sequence(2, 3, 4, 5) in
    c -> forAll(i | i < 4) = c -> collect(i | i < 4) -> excludes(false)
) .
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(2, 3, 4, 5) in c -> forAll
--- . (i | i < 4) = c -> collect(i | i < 4) -> excludes(false)) .
--- rewrites: 123 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- Collection Operations
---
--- --- Collect
--- --- --- forAll to collect and one
--- --- --- --- Empty
--- --- --- --- --- Set ------------------------------------------------
--- 1
--- ?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))
red eval(
    let c = Set{} 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))
) .
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> forAll (i | i < 4) = (c
--- let s = c -> collect (i | i < 4) -> asSet() in c -> notEmpty() implies s ->
--- one (e | e) and s -> one(true))) .
--- rewrites: 55 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
true : Boolean

--- Expected Results:
--- true : Boolean

--- --- --- --- --- Bag
--- 2
--- ?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))
red eval(
let c = Bag{} in

--- Expected Results:
--- true : Boolean

--- --- --- --- --- Sequence
--- 3
--- ?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))
red eval(
let c = Sequence{} in

--- Expected Results:
--- true : Boolean

--- --- --- --- --- Singleton
--- --- --- --- --- Includes Undefined
--- --- --- --- --- --- Set ---------------------------------------------
--- 4
--- ?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))
--- Expected Results:
true : Boolean
NO. invalid or null in collections

?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))

Expected Results:
true : Boolean
NO. invalid or null in collections

?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))

Expected Results:
true : Boolean
NO. invalid or null in collections

Excludes Undefined

Expression e fulfilled

?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))
red eval(
  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))
) .

========================================
reduce in BENCHMARK-TEST-BS : eval(let c = Set{1} in c -> forAll (i | i < 4) =
  (let s = c -> collect (i | i < 4) -> asSet() in c -> notEmpty() implies s
  -> one (e | e) and s -> one(true))) .
rewrites: 99 in Oms cpu (Oms real) (~ rewrites/second)
result Bool: true

Expected Results:
true : Boolean

?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))
red eval(
  let c = Bag{1} in

\[
c \rightarrow \text{forAll}(i \mid i < 4) = 
\begin{align*}
& \text{(let } s = c \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } \\
& c \rightarrow \text{notEmpty()} \text{ implies } s \rightarrow \text{one(true)} \text{ and } s \rightarrow \text{one(e | e)})
\end{align*}
\]

---

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c \rightarrow \text{forAll}(i \mid i < 4) = 
--- . \text{(let } s = c \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } c \rightarrow \text{notEmpty()} \text{ implies } s 
--- \rightarrow \text{one(e | e)} \text{ and } s \rightarrow \text{one(true)}) .
--- rewrites: 99 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---

--- Expected Results:
--- true : Boolean
---

--- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
--- c \rightarrow \text{forAll}(i \mid i < 4) = 
--- \text{(let } s = c \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } \\
--- c \rightarrow \text{notEmpty()} \text{ implies } s \rightarrow \text{one(true)} \text{ and } s \rightarrow \text{one(e | e)})
red eval(let c = Sequence{1} in c \rightarrow \text{forAll}(i \mid i < 4) = 
\begin{align*}
& \text{(let } s = c \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } \\
& c \rightarrow \text{notEmpty()} \text{ implies } s \rightarrow \text{one(true)} \text{ and } s \rightarrow \text{one(e | e)})
\end{align*}
).
---

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c \rightarrow \text{forAll}(i \mid i < 4) = 
--- . \text{(let } s = c \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } c \rightarrow \text{notEmpty()} 
--- \text{ implies } s \rightarrow \text{one(e | e)} \text{ and } s \rightarrow \text{one(true)}) .
--- rewrites: 99 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---

--- Expected Results:
--- true : Boolean
---

--- --- --- --- --- --- Expression e not fulfilled
---

--- --- --- --- --- --- Set
--- 10
--- ?let c = Set{4} in
--- c \rightarrow \text{forAll}(i \mid i < 4) = 
--- \text{(let } s = c \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } \\
--- c \rightarrow \text{notEmpty()} \text{ implies } s \rightarrow \text{one(true)} \text{ and } s \rightarrow \text{one(e | e)})
red eval(let c = Set{4} in c \rightarrow \text{forAll}(i \mid i < 4) = 
\begin{align*}
& \text{(let } s = c \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } \\
& c \rightarrow \text{notEmpty()} \text{ implies } s \rightarrow \text{one(true)} \text{ and } s \rightarrow \text{one(e | e)})
\end{align*}
).
---

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c \rightarrow \text{forAll}(i \mid i < 4) = 
--- . \text{(let } s = c \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } c \rightarrow \text{notEmpty()} \text{ implies } s 
--- \rightarrow \text{one(e | e)} \text{ and } s \rightarrow \text{one(true)}) .
--- rewrites: 93 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---

--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- Bag
--- 11
--- ?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))
red eval(
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))
).
--- ===================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> forAll (i | i < 4) =
---     (let s = c -> collect (i | i < 4) -> asSet() in c -> notEmpty()) implies s
---     -> one (e | e) and s -> one(true)) .
--- rewrites: 93 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 12
--- ?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))
red eval(
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))
).
--- ===================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4} in c -> forAll (i | i <
---     4) = (let s = c -> collect (i | i < 4) -> asSet() in c -> notEmpty())
---     implies s -> one (e | e) and s -> one(true)) .
--- rewrites: 93 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- Many Elements
--- --- --- --- Includes Undefined
--- --- --- --- --- Set ---------------------------------------------
--- 13
--- ?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))
---
--- Expected Results:
---  true : Boolean
---
--- NO. invalid or null in collections

191
Bag

--- Equal Values

?let c = Bag{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in

\( \text{c} \rightarrow \forall (i \mid i < 4) = (\text{let } s = \text{c} \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } \text{c} \rightarrow \text{notEmpty()} \text{ implies } s \rightarrow \text{one(true)} \text{ and } s \rightarrow \text{one(e \mid e)}) \)

Expected Results:
true : Boolean

--- No Equal Values

?let c = Bag{oclUndefined(Integer), 1, 2, 3} in

\( \text{c} \rightarrow \forall (i \mid i < 4) = (\text{let } s = \text{c} \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } \text{c} \rightarrow \text{notEmpty()} \text{ implies } s \rightarrow \text{one(true)} \text{ and } s \rightarrow \text{one(e \mid e)}) \)

Expected Results:
true : Boolean

--- Sequence

--- Equal Values

?let c = Sequence{oclUndefined(Integer), 1, 2, 3, oclUndefined(Integer)} in

\( \text{c} \rightarrow \forall (i \mid i < 4) = (\text{let } s = \text{c} \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } \text{c} \rightarrow \text{notEmpty()} \text{ implies } s \rightarrow \text{one(true)} \text{ and } s \rightarrow \text{one(e \mid e)}) \)

Expected Results:
true : Boolean

--- No Equal Values

?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in

\( \text{c} \rightarrow \forall (i \mid i < 4) = (\text{let } s = \text{c} \rightarrow \text{collect}(i \mid i < 4) \rightarrow \text{asSet()} \text{ in } \text{c} \rightarrow \text{notEmpty()} \text{ implies } s \rightarrow \text{one(true)} \text{ and } s \rightarrow \text{one(e \mid e)}) \)

Expected Results:
true : Boolean

--- Excludes Undefined

--- Expression e fulfilled

--- Set ------------------------------------------

?let c = Set{1, 2, 3} in

\( \text{c} \rightarrow \forall (i \mid i < 4) = \)
--- (let s = c -> collect(i | i < 4) -> asSet() in
c -> notEmpty() implies s -> one(true) and s -> one(e | e))
red eval(
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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(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 (e | e) and s -> one(true)).
--- rewrites: 157 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- --- --- Equal Values
--- 19
--- ?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))
red eval(
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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(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 (e | e) and s -> one(true)).
--- rewrites: 186 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- --- --- No Equal Values
--- 20
--- ?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))
red eval(
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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(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 (e | e) and s -> one(true)).
--- rewrites: 157 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- --- Equal Values
--- 21
--- ?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))
---
--- red eval(
--- 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))
---   ).
---
--- reduce in BENCHMARK-TEST-B5 : eval(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 (e | e) and s -> one(true)) .
--- rewrites: 183 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- --- --- --- --- --- --- No Equal Values
--- 22
--- ?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))
---
--- red eval(
--- 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))
---   ).
---
--- reduce in BENCHMARK-TEST-B5 : eval(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 (e | e) and s -> one(true)) .
--- rewrites: 155 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- --- --- --- --- --- --- Expression e not fulfilled
--- 23
--- ?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))
red eval(
  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))
) .
--- ============================
--- reduce in BENCHMARK-TEST-B5 : eval(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 (e | e) and s -> one(true)) .
--- rewrites: 180 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- ?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))
red eval(
  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))
) .
--- ============================
--- reduce in BENCHMARK-TEST-B5 : eval(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 (e | e) and s -> one(true)) .
--- rewrites: 198 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 25
--- ?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))
red eval(
  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))
) .
--- ============================
--- reduce in BENCHMARK-TEST-B5 : eval(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 (e | e) and s -> one(true)) .
--- rewrites: 198 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- No Equal Values
--- 26
--- ?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))
red eval(
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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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 (e | e) and s -> one(true)) .
--- rewrites: 180 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- Sequence

--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 27
--- ?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))
red eval(
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))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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 (e | e) and s -> one(true)) .
--- rewrites: 194 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?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))
red eval(
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))
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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 (e | e) and s -> one(true)) .
--- rewrites: 194 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- ---
c -> notEmpty() implies s -> one(true) and s -> one(e | e))

--- Collection Operations

--- Collect

--- --- one to collect

--- --- --- Empty

--- --- --- --- Set --------------------------------------------------------

--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
let c = Set{} in
  c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
) .

--- reduce in BENCHMARK-TEST-B5 : eval(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(e | e) and s -> one(true)) .
--- rewrites: 194 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 29
--- ?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))
red eval(
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))
) .

--- reduce in BENCHMARK-TEST-B5 : eval(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(e | e) and s -> one(true)) .
--- rewrites: 177 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---

--- Collection Operations

--- --- Collect

--- --- --- one to collect

--- --- --- --- Empty

--- --- --- --- --- Set --------------------------------------------------------

--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
let c = Set{} in
  c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
) .

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> one(i | i < 4) = (c
  -> collect(i | i < 4) -> count(true) = 1)) .
--- rewrites: 30 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
  let c = Bag{} in
c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{} in c -> one (i | i < 4) = (c
--- -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 30 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
  let c = Sequence{} in
c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{} in c -> one (i | i < 4) =
--- . (c -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 30 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- --- --- Singleton
--- --- --- Includes Undefined
--- --- --- --- --- Set
--- --- --- --- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
---
--- Expected Results:
--- --- Bag
--- --- NO. invalid or null in collections

--- --- --- --- ---
--- NO. invalid or null in collections
---
--- --- --- --- --- --- Sequence
---
--- ?let c = Sequence{oclUndefined(Integer)} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- Excludes Undefined
---
--- --- --- --- --- --- Expression e fulfilled
---
--- --- --- --- --- --- --- Set
---
--- --- --- --- --- --- --- Bag
---
--- --- --- --- --- --- --- Sequence
---
--- ?let c = Bag{1} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
---
--- --- --- --- --- --- --- Sequence
---
--- ?let c = Sequence{1} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
---
--- Expected Results:
--- true : Boolean
---
--- = (c -> collect {i | i < 4} -> count(true) = 1)) .
--- rewrites: 61 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- Expression e not fulfilled
---
--- --- --- --- --- --- --- Set -------------------------------
--- 10
--- ?let c = Set{4} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
  let c = Set{4} in
  c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
).
--- =============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{4} in c -> one (i | i < 4) = (c
---   -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 56 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag-------------------------------
--- 11
--- ?let c = Bag{4} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
  let c = Bag{4} in
  c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
).
--- =============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{4} in c -> one (i | i < 4) = (c
---   -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 56 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence------------------------
--- 12
--- ?let c = Sequence{4} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
  let c = Sequence{4} in
  c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{4} in c -> one (i | i < 4)
---   = (c -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 56 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
---
Many Elements

Includes Undefined

Set

?let c = Set{oclUndefined(Integer), 1, 4, 5} in
   c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)

Expected Results:
   true : Boolean
   NO. invalid or null in collections

Bag

Equal Values

?let c = Bag{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in
   c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)

Expected Results:
   true : Boolean
   NO. invalid or null in collections

No Equal Values

?let c = Bag{oclUndefined(Integer), 1, 4, 5} in
   c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)

Expected Results:
   true : Boolean
   NO. invalid or null in collections

Sequence

Equal Values

?let c = Sequence{oclUndefined(Integer), 1, 4, 5, oclUndefined(Integer)} in
   c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)

Expected Results:
   true : Boolean
   NO. invalid or null in collections

No Equal Values

?let c = Sequence{oclUndefined(Integer), 1, 4, 5} in
   c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)

Expected Results:
   true : Boolean
   NO. invalid or null in collections

Excludes Undefined

Expression e fulfilled
--- 18
--- let c = Set{1, 4, 5} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
--- eval(let c = Set{1, 4, 5} in c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1))
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 4, 5} in c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1))
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 19
--- let c = Bag{1, 4, 4, 5} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
--- eval(let c = Bag{1, 4, 4, 5} in c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1))
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 4, 4, 5} in c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1))
--- rewrites: 145 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 20
--- let c = Bag{1, 4, 5} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
--- eval(let c = Bag{1, 4, 5} in c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1))
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 4, 5} in c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1))
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
    let c = Sequence{1, 4, 5, 4} in
    c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
).
--- ================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 4, 5, 4} in c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)).
--- rewrites: 145 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 22
--- ?let c = Sequence{1, 4, 5} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
    let c = Sequence{1, 4, 5} in
    c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
).
--- ================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 4, 5} in c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)).
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Expression e not fulfilled
--- 23
--- ?let c = Set{1, 2, 5} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
    let c = Set{1, 2, 5} in
    c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
).
--- ================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 5} in c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)).
--- rewrites: 105 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- Equal Values (violating)
--- 24
--- ?let c = Bag{1, 2, 5, 1} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
    let c = Bag{1, 2, 5, 1} in
    c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
)
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1, 2, 1, 5} in c -> one (i | i < 4) = (c -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 125 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (not violating)
--- 25
--- ?let c = Bag{1, 2, 5, 5} in
---   c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
  let c = Bag{1, 2, 5, 5} in
  c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
) .
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1, 2, 5, 5} in c -> one (i | i < 4) = (c -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 122 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 26
--- ?let c = Bag{1, 2, 5} in
---   c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
  let c = Bag{1, 2, 5} in
  c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
) .
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1, 2, 5} in c -> one (i | i < 4) = (c -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 105 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 27
--- ?let c = Sequence{1, 2, 5, 1} in
---   c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
  let c = Sequence{1, 2, 5, 1} in
  c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
) .
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{1, 2, 5, 1} in c -> one (i | i < 4) = (c -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 125 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true

204
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?let c = Sequence{1, 2, 5, 5} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
let c = Sequence{1, 2, 5, 5} in
c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5, 5} in c -> one (i | i < 4) = (c -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 122 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- No Equal Values
--- 28
--- ?let c = Sequence{1, 2, 5} in
--- c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
red eval(
let c = Sequence{1, 2, 5} in
c -> one(i | i < 4) = (c -> collect(i | i < 4) -> count(true) = 1)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 5} in c -> one (i | i < 4) = (c -> collect (i | i < 4) -> count(true) = 1)) .
--- rewrites: 105 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- Collection Operations
--- --- Iterate
--- --- --- collect
--- --- --- --- Empty
--- --- --- --- Set ------------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> collect(i | i * i) =
--- c -> iterate(i; r:Bag(Integer) = oclEmpty(Bag(Integer)) |
--- r -> including(i * i))
red eval(
let c = Set{} in

File: p05_o01_collect2iterate.maude
---

--- Collection Operations
--- --- Iterate
--- --- --- collect
--- --- --- --- Empty
--- --- --- --- Set ------------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> collect(i | i * i) =
--- c -> iterate(i; r:Bag(Integer) = oclEmpty(Bag(Integer)) |
--- r -> including(i * i))
red eval(
let c = Set{} in

205
c -> collect(i | i * i) =
   c -> iterate(i ; r : Bag(Integer) = Bag{} | r -> including(i * i))
).
---========================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> collect (i | i * i) =
   c -> iterate (i ; r : Bag(Integer) = Bag{} | r -> including(i * i))) .
--- rewrites: 30 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
---   c -> collect(i | i * i) =
---     c -> iterate(i ; r : Bag(Integer) = oclEmpty(Bag(Integer)) |
---       r -> including(i * i))
red eval(
let c = Bag{} in
c -> collect(i | i * i) =
c -> iterate(i ; r : Bag(Integer) = Bag{} |
  r -> including(i * i))
).
---========================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> collect (i | i * i) =
   c -> iterate (i ; r : Bag(Integer) = Bag{} | r -> including(i * i))) .
--- rewrites: 29 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Sequence A
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
---   c -> collect(i | i * i) =
---     c -> iterate(i;
---       r : Sequence(Integer) = oclEmpty(Sequence(Integer)) |
---       r -> including(i * i))
red eval(
let c = Sequence{} in
c -> collect(i | i * i) =
c -> iterate(i ;
  r : Sequence(Integer) = Sequence{} |
  r -> including(i * i))
).
---========================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> collect (i | i * i) =
   c -> iterate (i ; r : Sequence(Integer) = Sequence{} | r -> including(i * i))) .
--- rewrites: 28 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Sequence B
--- 4
--- ?let c = oclEmpty(Sequence(Integer)) in
---   c -> collect(i | i * i) =
---   c -> iterate(i; 
---       r:Sequence(Integer) = oclEmpty(Sequence(Integer)) | 
---       r -> append(i * i))
red eval(
let c = Sequence{} in
   c -> collect(i | i * i) =
   c -> iterate(i;
       r : Sequence(Integer) = Sequence{} |
       r -> append(i * i))
).
--- ==================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> collect (i | i * 
---   i) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | r -> append(i * 
---   i)) .
--- rewrites: 28 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- Singleton
--- --- --- --- --- Includes Undefined
--- --- --- --- --- --- Set ---------------------------------------------
--- 5
--- ?let c = Set{oclUndefined(Integer)} in
---   c -> collect(i | i * i) =
---   c -> iterate(i; r:Bag(Integer) = oclEmpty(Bag(Integer)) |
---     r -> including(i * i))
---
--- Expected Results:
---   true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- Bag
--- 6
--- ?let c = Bag{oclUndefined(Integer)} in
---   c -> collect(i | i * i) =
---   c -> iterate(i; r:Bag(Integer) = oclEmpty(Bag(Integer)) |
---     r -> including(i * i))
---
--- Expected Results:
---   true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- Sequence A
--- 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))
---
--- Expected Results:
---   true : Boolean
---
--- 207
--- NO. invalid or null in collections

--- --- --- --- --- --- Sequence B

--- 8
--- ?let c = Sequence{oclUndefined(Integer)} in
--- c -> collect(i | i * i) =
--- c -> iterate(i;
--- r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r -> append(i * i))
red eval(
let c = Sequence{null} in
  c -> collect(i | i * i) =
  c -> iterate(i ;
      r : Sequence(Integer) = Sequence{} |
      r -> append(i * i))
).

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{null} in c -> collect (i |
--- i * i) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | r ->
--- append(i * i)) .
--- rewrites: 56 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- false : Boolean

--- --- --- --- --- --- Excludes Undefined

--- --- --- --- --- --- Set ---------------------------------------------

--- 9
--- ?let c = Set{2} in
--- c -> collect(i | i * i) =
--- c -> iterate(i ;
--- r : Sequence(Integer) = Sequence{} |
--- r -> append(i * i))
red eval(
let c = Set{2} in
  c -> collect(i | i * i) =
  c -> iterate(i ;
      r : Bag(Integer) = Bag{} |
      r -> including(i * i))
).

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{2} in c -> collect (i | i * i) =
--- c -> iterate (i ; r : Bag(Integer) = Bag{} | r -> including(i * i)) .
--- rewrites: 61 in 4ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Bag

--- 10
--- ?let c = Bag{2} in
--- c -> collect(i | i * i) =
--- c -> iterate(i ;
--- r : Bag(Integer) = Bag{} |
--- r -> including(i * i))
red eval(
let c = Bag{2} in
  c -> collect(i | i * i) =
  c -> iterate(i ;
      r : Bag(Integer) = Bag{} |
r -> including(i * i))

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2} in c -> collect (i | i * i) =
--- c -> iterate (i ; r : Bag(Integer) = Bag{} | r -> including(i * i))) .
--- rewrites: 61 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Sequence A
--- 11
--- ?let c = Sequence{2} in
--- c -> collect(i | i * i) =
--- c -> iterate(i;
--- r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r -> including(i * i))
red eval(let c = Sequence{2} in
  c -> collect(i | i * i) =
  c -> iterate(i;
    r : Sequence(Integer) = Sequence{} |
    r -> including(i * i))
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2} in c -> collect (i | i *
--- i) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | r -> including(
--- i * i))) .
--- rewrites: 59 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Sequence B
--- 12
--- ?let c = Sequence{2} in
--- c -> collect(i | i * i) =
--- c -> iterate(i;
--- r : Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r -> append(i * i))
red eval(let c = Sequence{2} in
  c -> collect(i | i * i) =
  c -> iterate(i;
    r : Sequence(Integer) = Sequence{} |
    r -> append(i * i))
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2} in c -> collect (i | i *
--- i) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | r -> append(i *
--- i))) .
--- rewrites: 58 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

209
--- --- --- --- Many Elements
--- --- --- --- Includes Undefined
--- --- --- --- --- Set
--- 13
--- ?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))
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- Bag
--- --- --- --- --- --- Equal Values
--- 14
--- ?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))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- --- --- --- No Equal Values
--- 15
--- ?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))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- --- --- --- Sequence A
--- --- --- --- --- --- Equal Values
--- 16
--- ?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))
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- --- --- --- No Equal Values
--- 17
--- ?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))
---

red eval(
  let c = Sequence(null, 0, 1, 2) in
  c -> collect(i | i * i) =
  c -> iterate(i;
    r = Sequence{} |
      r -> including(i * i))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(null, 0, 1, 2) in c ->
---    collect (i | i * i) = c -> iterate (i, r = Sequence{} | r -> including(i *
---    i)))
--- rewrites: 152 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- --- Sequence B
---
--- ?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))
---
--- Expected Results:
---   false : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- --- No Equal Values
---
--- ?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))
red eval(
  let c = Sequence(null, 0, 1, 2) in
  c -> collect(i | i * i) =
  c -> iterate(i;
    r : Sequence(Integer) = Sequence{} |
      r -> append(i * i))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(null, 0, 1, 2) in c ->
---    collect (i | i * i) = c -> iterate (i ; r : Sequence(Integer) = Sequence{}
---    | r -> append(i * i)))
--- rewrites: 149 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
false : Boolean

--- Excludes Undefined

Set -----------------------------

--- 20
--- ?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))
red eval(let c = Set{-1, 0, 1, 2} in
c -> collect(i | i * i) =
c -> iterate(i ; r : Bag(Integer) = Bag{} | r -> including(i * i))
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{-1, 0, 1, 2} in c -> collect (i | i * i) = c -> iterate (i ; r : Bag(Integer) = Bag{} | r -> including(i * i))).
--- rewrites: 160 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
--- Expected Results:
--- true : Boolean
---
--- Equal Values
---
--- 21
--- ?let c = Bag{-1, 0, 1, 2, 1} in
c -> collect(i | i * i) =
c -> iterate(i ; r : Bag(Integer) = Bag{} | r -> including(i * i))
red eval(let c = Bag{-1, 0, 1, 2, 1} in
c -> collect(i | i * i) =
c -> iterate(i ; r : Bag(Integer) = Bag{} | r -> including(i * i))
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{-1, 0, 1, 2, 1} in c -> collect (i | i * i) = c -> iterate (i ; r : Bag(Integer) = Bag{} | r -> including(i * i))).
--- rewrites: 193 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
--- Expected Results:
--- true : Boolean
---
--- No Equal Values
---
--- 22
--- ?let c = Bag{-1, 0, 1, 2} in
c -> collect(i | i * i) =
c -> iterate(i ; r : Bag(Integer) = Bag{} | r -> including(i * i))
red eval(let c = Bag{-1, 0, 1, 2} in
c -> collect(i | i * i) =
c -> iterate(i ; r : Bag(Integer) = Bag{} |
\[ r \rightarrow \text{including}(i \times i) \]

---

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag\{-1, 0, 1, 2\} in c \rightarrow collect (i | i \times i) = c \rightarrow iterate (i ; r : Bag(Integer) = Bag{} | r \rightarrow including(i \times i))) .
--- rewrites: 160 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence A
---
--- 23
--- ?let c = Sequence\{-1, 0, 1, 2, 1\} in
--- c \rightarrow collect(i | i \times i) =
--- c \rightarrow iterate(i ;
--- r : Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r \rightarrow including(i \times i))
red eval(
let c = Sequence\{-1, 0, 1, 2, 1\} in
  c \rightarrow collect(i | i \times i) =
  c \rightarrow iterate(i ;
    r : Sequence(Integer) = Sequence{} |
    r \rightarrow including(i \times i))
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence\{-1, 0, 1, 2, 1\} in c \rightarrow collect (i | i \times i) = c \rightarrow iterate (i ; r : Sequence(Integer) = Sequence{} | r \rightarrow including(i \times i))) .
--- rewrites: 187 in 4ms cpu (0ms real) (46738 rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
---
--- 24
--- ?let c = Sequence\{-1, 0, 1, 2\} in
--- c \rightarrow collect(i | i \times i) =
--- c \rightarrow iterate(i ;
--- r : Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r \rightarrow including(i \times i))
red eval(
let c = Sequence\{-1, 0, 1, 2\} in
  c \rightarrow collect(i | i \times i) =
  c \rightarrow iterate(i ;
    r : Sequence(Integer) = Sequence{} |
    r \rightarrow including(i \times i))
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence\{-1, 0, 1, 2\} in c \rightarrow collect (i | i \times i) = c \rightarrow iterate (i ; r : Sequence(Integer) = Sequence{} | r \rightarrow including(i \times i))) .
--- rewrites: 155 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
---
--- Expected Results: 
--- true : Boolean 

--- --- --- --- --- --- --- Sequence B
--- --- --- --- --- --- --- Equal Values
--- 25
--- ?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)) 
red eval(
let c = Sequence{-1, 0, 1, 2, 1} in 
c -> collect(i | i * i) = 
c -> iterate(i ; 
     r : Sequence(Integer) = Sequence{} | 
     r -> append(i * i)) 
).                      
--- =================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{-1, 0, 1, 2, 1} in c -> 
--- collect (i | i * i) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} 
--- | r -> append(i * i))).
--- rewrites: 182 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true 
---
--- Expected Results: 
--- true : Boolean 

--- --- --- --- --- --- --- No Equal Values
--- 26
--- ?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)) 
red eval(
let c = Sequence{-1, 0, 1, 2} in 
c -> collect(i | i * i) = 
c -> iterate(i ; 
     r : Sequence(Integer) = Sequence{} | 
     r -> append(i * i)) 
).                      
--- =================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{-1, 0, 1, 2} in c -> 
--- collect (i | i * i) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} 
--- | r -> append(i * i))).
--- rewrites: 151 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true 
---
--- Expected Results: 
--- true : Boolean 

--- Collection Operations
--- --- Iterate
--- --- --- exists
--- --- --- --- Empty
--- --- --- --- --- Set
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
let c = Set{} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
--- !=============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> exists (i | i < 4) = c
---    -> iterate (i ; r : Boolean = false | r or i < 4)).
--- rewrites: 28 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
let c = Bag{} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
--- !=============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> exists (i | i < 4) = c
---    -> iterate (i ; r : Boolean = false | r or i < 4)).
--- rewrites: 28 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
let c = Sequence{} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
--- !=============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> exists (i | i <
--- 4) = c -> iterate (i ; r : Boolean = false | r or i < 4)).
--- rewrites: 28 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- Singleton
--- --- --- --- --- Includes Undefined

--- --- --- --- --- --- Set ---------------------------------------------
--- 4
--- ?let c = Set{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = c -> iterate(i; r: Boolean = false | r or i < 4)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- Bag
--- 5
--- ?let c = Bag{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = c -> iterate(i; r: Boolean = false | r or i < 4)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- Sequence
--- 6
--- ?let c = Sequence{oclUndefined(Integer)} in
--- c -> exists(i | i < 4) = c -> iterate(i; r: Boolean = false | r or i < 4)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined

--- --- --- --- --- --- --- Set ---------------------------------------------
--- 7
--- ?let c = Set{1} in
--- c -> exists(i | i < 4) = c -> iterate(i; r: Boolean = false | r or i < 4)
red eval(
  let c = Set{1} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
) .
---
--- Expected Results:
--- true : Boolean
---

--- --- --- --- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
--- c -> exists(i | i < 4) = c -> iterate(i; r: Boolean = false | r or i < 4)
red eval(
  let c = Bag{1} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
) .
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> exists (i | i < 4) =
--- c -> iterate (i ; r : Boolean = false | r or i < 4)) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
let c = Sequence{1} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> exists (i | i < 4) =
--- c -> iterate (i ; r : Boolean = false | r or i < 4)) .
--- rewrites: 52 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
---
--- --- --- --- --- --- --- Set
--- 10
--- ?let c = Set{4} in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
let c = Set{4} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> exists (i | i < 4) =
--- c -> iterate (i ; r : Boolean = false | r or i < 4)) .
--- rewrites: 57 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
let c = Bag{4} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> exists (i | i < 4) =
--- c -> iterate (i ; r : Boolean = false | r or i < 4)) .
--- rewrites: 57 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:  
--- true : Boolean  

--- --- --- --- --- --- --- Sequence  
--- 12  
--- ?let c = Sequence(4) in  
--- c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)  
red eval(  
let c = Sequence(4) in  
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)  
) .  
--- ===============================  
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(4) in c -> exists (i | i < 4) = c -> iterate (i ; r : Boolean = false | r or i < 4)) .  
--- rewrites: 56 in 0ms cpu (0ms real) (~ rewrites/second)  
--- result Bool: true  
---  
--- Expected Results:  
--- true : Boolean  

--- --- --- --- Many Elements  

--- --- --- --- --- Includes Undefined  

--- --- --- --- --- Set  
--- ---------------------------------------------------------------  
--- 13  
--- ?let c = Set(oclUndefined(Integer), 2, 3, 4) in  
--- c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)  
---  
--- Expected Results:  
--- true : Boolean  
--- NO. invalid or null in collections  

--- --- --- --- Bag  

--- --- --- --- --- --- Equal Values  
--- 14  
--- ?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)  
---  
--- Expected Results:  
--- true : Boolean  
--- NO. invalid or null in collections  

--- --- --- --- --- No Equal Values  
--- 15  
--- ?let c = Bag(oclUndefined(Integer), 2, 3, 4) in  
--- c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)  
---  
--- Expected Results:  
--- true : Boolean  
--- NO. invalid or null in collections  

--- --- --- --- --- Sequence  

--- --- --- --- --- --- Equal Values  
--- 16  
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- --- No Equal Values
---
--- ?let c = Sequence{oclUndefined(Integer), 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- --- Set ----------------------------------------------
---
--- ?let c = Set{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
  let c = Set{1, 2, 3, 4} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
)
---
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{1, 2, 3, 4} in c -> exists (i |
--- i < 4) = c -> iterate (i ; r : Boolean = false | r or i < 4)) .
--- rewrites: 110 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values
---
--- ?let c = Bag{1, 2, 3, 4, 1} in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
  let c = Bag{1, 2, 3, 4, 1} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
)
---
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1, 2, 3, 4, 1} in c -> exists (i |
--- i < 4) = c -> iterate (i ; r : Boolean = false | r or i < 4)) .
--- rewrites: 129 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
---
--- ?let c = Bag{1, 2, 3, 4} in
red eval(
  let c = Bag{1, 2, 3, 4} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
---

--- Expected Results:
--- true : Boolean
---

--- --- --- --- --- --- --- Sequence
---
--- ?let c = Sequence{1, 2, 3, 4, 1} in
--- c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
red eval(
  let c = Sequence{1, 2, 3, 4, 1} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
---

--- Expected Results:
--- true : Boolean
---

--- --- --- --- --- --- --- No Equal Values
---
--- ?let c = Sequence{1, 2, 3, 4} in
--- c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
red eval(
  let c = Sequence{1, 2, 3, 4} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
---

--- Expected Results:
--- true : Boolean
---

--- --- --- --- --- --- --- Expression e not fulfilled
---
--- ?let c = Set{4, 5, 6} in
--- c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
red eval(
  let c = Set{4, 5, 6} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
)
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4, 5, 6} in c -> exists (i | i < 4) = c -> iterate (i ; r : Boolean = false | r or i < 4)).
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- ?let c = Bag{4, 5, 6, 4} in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
  let c = Bag{4, 5, 6, 4} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4, 5, 6, 4} in c -> exists (i | i < 4) = c -> iterate (i ; r : Boolean = false | r or i < 4)).
--- rewrites: 147 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 25
--- ?let c = Bag{4, 5, 6} in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
  let c = Bag{4, 5, 6} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4, 5, 6} in c -> exists (i | i < 4) = c -> iterate (i ; r : Boolean = false | r or i < 4)).
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values (both violating)
--- 26
--- ?let c = Sequence{4, 5, 6, 4} in
--- c -> exists(i | i < 4) = c -> iterate(i;r:Boolean = false | r or i < 4)
red eval(
  let c = Sequence{4, 5, 6, 4} in
  c -> exists(i | i < 4) = c -> iterate(i ; r : Boolean = false | r or i < 4)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4, 5, 6, 4} in c -> exists (i | i < 4) = c -> iterate (i ; r : Boolean = false | r or i < 4)).
--- rewrites: 117 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- ---
File: p05_o03_forAll2iterate.maude

--- Collection Operations
--- --- Iterate
--- --- --- forAll
--- --- --- --- Empty
--- --- --- --- --- Set ------------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
let c = Set{} in
 c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> forAll (i | i < 4) = c
---   -> iterate (i ; r : Boolean = true | r and i < 4)).
--- rewrites: 28 in 4ms cpu (Oms real) (7000 rewrites/second)
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
let c = Bag{} in
 c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
).
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Singleton

--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Bag

--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Sequence

--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined

--- --- --- --- --- --- Bag
--- --- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- Set ------------------------------------------
--- 7
--- let c = Set{1} in
c -> forAll( i | i < 4 ) = c -> iterate( i ; r : Boolean = true | r and i < 4 )
red eval(
let c = Set{1} in
c -> forAll( i | i < 4 ) = c -> iterate( i ; r : Boolean = true | r and i < 4 )
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1} in c -> forAll ( i | i < 4 ) =
--- c -> iterate ( i ; r : Boolean = true | r and i < 4 )).
--- rewrites: 57 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- 8
--- let c = Bag{1} in
c -> forAll( i | i < 4 ) = c -> iterate( i ; r : Boolean = true | r and i < 4 )
red eval(
let c = Bag{1} in
c -> forAll( i | i < 4 ) = c -> iterate( i ; r : Boolean = true | r and i < 4 )
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> forAll ( i | i < 4 ) =
--- c -> iterate ( i ; r : Boolean = true | r and i < 4 )).
--- rewrites: 57 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 9
--- let c = Sequence{1} in
c -> forAll( i | i < 4 ) = c -> iterate( i ; r : Boolean = true | r and i < 4 )
red eval(
let c = Sequence{1} in
c -> forAll( i | i < 4 ) = c -> iterate( i ; r : Boolean = true | r and i < 4 )
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> forAll ( i | i < 4 )
--- c -> iterate ( i ; r : Boolean = true | r and i < 4 )).
--- rewrites: 56 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 10
--- let c = Set{4} in
--- c -> forall(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
  let c = Set{4} in
  c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
).
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> forall (i | i < 4) =
--- c -> iterate (i ; r : Boolean = true | r and i < 4)).
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- Bag
--- 11
--- ?let c = Bag{4} in
---  c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
  let c = Bag{4} in
  c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
).
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{4} in c -> forall (i | i < 4) =
--- c -> iterate (i ; r : Boolean = true | r and i < 4)).
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- --- Sequence
--- 12
--- ?let c = Sequence{4} in
---  c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
  let c = Sequence{4} in
  c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
).
--- ===============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{4} in c -> forall (i | i <
--- 4) = c -> iterate (i ; r : Boolean = true | r and i < 4)).
--- rewrites: 52 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- Many Elements
--- --- --- --- Includes Undefined
--- --- --- --- --- Set
--- 13
--- ?let c = Set{oclUndefined(Integer), 1, 2, 3} in
---  c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
---
--- Expected Results:
---  true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- Bag

--- --- --- --- --- --- --- Equal Values
--- 14
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 15
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3} in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- Sequence

--- --- --- --- --- --- --- Equal Values
--- 16
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 17
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3} in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined

--- --- --- --- --- --- Expression e fulfilled

--- --- --- --- --- --- --- Set ------------------------------------------
--- 18
--- ?let c = Set{1, 2, 3} in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
    let c = Set{1, 2, 3} in
    c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 3} in c -> forAll (i | i <
--- 4) = c -> iterate (i ; r : Boolean = true | r and i < 4)) .

226
--- rewrites: 117 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- Equal Values
--- 19
--- ?let c = Bag{1, 2, 3, 1} in
---   c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
  let c = Bag{1, 2, 3, 1} in
  c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
)
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 20
--- ?let c = Bag{1, 2, 3} in
---   c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
  let c = Bag{1, 2, 3} in
  c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
)
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- --- Equal Values
--- 21
--- ?let c = Sequence{1, 2, 3, 1} in
---   c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
  let c = Sequence{1, 2, 3, 1} in
  c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
)
---
--- Expected Results:
---   true : Boolean

---
---

---

---

--- 227
--- true : Boolean
--- --- --- --- --- No Equal Values
--- 22
--- ?let c = Sequence{1, 2, 3} in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
let c = Sequence{1, 2, 3} in
  c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
)
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{1, 2, 3} in c -> forAll (i
--- | i < 4) = c -> iterate (i ; r : Boolean = true | r and i < 4)) .
--- rewrites: 114 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- --- Set ------------------------------------------
--- 23
--- ?let c = Set{2, 3, 4, 5} in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
let c = Set{2, 3, 4, 5} in
  c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
)
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{2, 3, 4, 5} in c -> forAll (i |
--- i < 4) = c -> iterate (i ; r : Boolean = true | r and i < 4)) .
--- rewrites: 132 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- Bag
--- --- --- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- ?let c = Bag{2, 3, 4, 5, 4} in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
let c = Bag{2, 3, 4, 5, 4} in
  c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
)
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{2, 3, 4, 5, 4} in c -> forAll (i |
--- i < 4) = c -> iterate (i ; r : Boolean = true | r and i < 4)) .
--- rewrites: 151 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 25
--- ?let c = Bag{2, 3, 4, 5, 2} in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
  let c = Bag{2, 3, 4, 5, 2} in
    c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5, 2} in c -> forAll (i
---   | i < 4) = c -> iterate (i ; r : Boolean = true | r and i < 4)) .
--- rewrites: 151 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 26
--- ?let c = Bag{2, 3, 4, 5} in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
  let c = Bag{2, 3, 4, 5} in
    c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5} in c -> forAll (i |
---   i < 4) = c -> iterate (i ; r : Boolean = true | r and i < 4)) .
--- rewrites: 132 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 27
--- ?let c = Sequence{2, 3, 4, 5, 4} in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
  let c = Sequence{2, 3, 4, 5, 4} in
    c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 4} in c ->
---   forAll (i | i < 4) = c -> iterate (i ; r : Boolean = true | r and i < 4)) .
--- rewrites: 146 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?let c = Sequence{2, 3, 4, 5, 2} in
--- c -> forAll(i | i < 4) = c -> iterate(i;r:Boolean = true | r and i < 4)
red eval(
  let c = Sequence{2, 3, 4, 5, 2} in
    c -> forAll(i | i < 4) = c -> iterate(i ; r : Boolean = true | r and i < 4)
) .

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(2, 3, 4, 5, 2) in c ->
--- forAll (i | i < 4) = c -> iterate (i ; r : Boolean = true | r and i < 4)) .
--- rewrites: 146 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- Expected Results:
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(2, 3, 4, 5) in c -> forAll
--- . (i | i < 4) = c -> iterate (i ; r : Boolean = true | r and i < 4)) .
--- rewrites: 128 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
--- true : Boolean

--- Collection Operations

--- --- Iterate
--- --- one
--- --- --- --- Empty
--- --- --- --- --- Set ------------------------------------------------
--- 1
--- ?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()
red eval(
let c = Set{} in
c -> one(i | i < 4) =

--- Collection Operations

--- --- Iterate
--- --- one
--- --- --- --- Empty
--- --- --- --- --- Set ------------------------------------------------
--- 1
--- ?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()
red eval(
let c = Set{} in

c -> iterate(i ; r:Sequence(Boolean) = Sequence(false,false) |
--- 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() .
--- rewrites: 39 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- Bag
---
--- 2
--- ?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()
red eval(
let c = Bag{} 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()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} 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()).
--- rewrites: 39 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- --- --- --- --- Sequence
---
--- 3
--- ?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()
red eval(
let c = Sequence{} 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()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c → one (i | i < 4) =
--- 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()).
--- rewrites: 39 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
true : Boolean

Singleton

Includes Undefined

Set

4
?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()

Expected Results:
  true : Boolean
  NO. invalid or null in collections

Bag

5
?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()

Expected Results:
  true : Boolean
  NO. invalid or null in collections

Sequence

6
?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()

Expected Results:
  true : Boolean
  NO. invalid or null in collections

Excludes Undefined

Expression e fulfilled

Set

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()
red eval(
  let c = Set{1} in

232
\[ c \rightarrow \text{one}(i \mid i < 4) = \]
\[ c \rightarrow \text{iterate}(i; r: \text{Sequence}(\text{Boolean}) = \text{Sequence}(false, false) \mid \]
\[ \quad \text{if } r \rightarrow \text{first()} \text{ then } \text{Sequence}(true, false) \]
\[ \quad \text{else } \text{Sequence}(r \rightarrow \text{last()} \text{ and } i < 4, \]
\[ \quad \quad (r \rightarrow \text{last()} \text{ and } i < 4) \text{ xor } (r \rightarrow \text{last()} \text{ or } i < 4) \} \text{ endif} \rightarrow \text{last()} \]
\].

--- ========================
--- reduce in BENCHMARK-TEST-B5 : eval(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()).
--- rewrites: 112 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- 8
--- ?let c = Bag{1} in
--- c -> one(1 | i < 4) =
--- c -> iterate(1; 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()
red eval(
let c = Bag{1} in
\)
\].
--- ========================
--- reduce in BENCHMARK-TEST-B5 : eval(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()).
--- rewrites: 112 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{1} in
--- c -> one(1 | i < 4) =
--- c -> iterate(1; 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()
red eval(
let c = Sequence{1} in
\)
\].
--- --- --- --- --- --- --- Sequence
--- 233
--- reduce in BENCHMARK-TEST-B5 : eval(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()) .
--- rewrites: 111 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
---
--- --- --- --- --- --- --- Bag
--- 11
--- ?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()
red eval(
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()
).
-> 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()

rewrites: 110 in 0ms cpu (0ms real) (~ rewrites/second)
result Bool: true

---

Expected Results:
true : Boolean

--- 12
?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()

red eval(
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()
).

---

Expected Results:
true : Boolean

---

--- Many Elements

---

--- Includes Undefined

--- 13
?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()

---

Expected Results:
true : Boolean

--- NO. invalid or null in collections

--- Bag

---

--- Equal Values

--- 14
?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,
---M (r -> last() and i < 4) xor (r -> last() or i < 4) endif) -> last()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 15
--- ?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,
---M (r -> last() and i < 4) xor (r -> last() or i < 4) endif) -> last()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- Sequence

--- --- --- --- --- --- --- Equal Values
--- 16
--- ?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,
---M (r -> last() and i < 4) xor (r -> last() or i < 4) endif) -> last()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- --- --- No Equal Values
--- 17
--- ?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,
---M (r -> last() and i < 4) xor (r -> last() or i < 4) endif) -> last()
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined

--- --- --- --- --- --- --- Expression e fulfilled

--- --- --- --- --- --- --- Set -----------------------------
--- 18
--- ?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()
red eval(
  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()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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()) .
--- rewrites: 256 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Bag
---
--- ?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()
red eval(
  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()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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()) .
--- rewrites: 328 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- No Equal Values
--- 20
--- ?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}

237
--- else Sequence{r -> last() and i < 4, (r -> last() and i < 4) xor (r -> last() or i < 4)} endif) -> last()
red eval(
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()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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()) .
--- rewrites: 256 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- --- —— Sequence
--- --- --- --- --- --- --- --- --- --- --- Equal Values
--- 21
--- ?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()
red eval(
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()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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())
--- .
--- rewrites: 324 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- --- --- --- No Equal Values
--- 22
--- ?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()
red eval(
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() ) .
--- =========================================
--- reduce in BENCHMARK-TEST-B5 : eval(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() ) .
--- rewrites: 253 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
---
--- --- --- --- --- --- Set ------------------------------------------
--- 23
--- ?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()
red eval(
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() ) .
--- reduce in BENCHMARK-TEST-B5 : eval(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() ) .
--- rewrites: 208 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag ------------------------------------------
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- ?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()
red eval(
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()
  .

--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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, 
--- 4, r -> last() and i < 4 xor (r -> last() or i < 4)) endif) -> last()) .
--- rewrites: 236 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 25
--- ?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()
red eval(
  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()
  .

--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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, 
--- 4, r -> last() and i < 4 xor (r -> last() or i < 4)) endif) -> last()) .
--- rewrites: 236 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 26
--- ?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()
red eval(
  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()
(r -> last() and i < 4) xor (r -> last() or i < 4) endif) -> last()

--- ==========================================---
--- reduce in BENCHMARK-TEST-B5 : eval(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()) .
--- rewrites: 208 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- --- Equal Values (both violating)
--- 27
--- ?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()
red eval(
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()
).
--- ==========================================---
--- reduce in BENCHMARK-TEST-B5 : eval(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,
--- i < 4, r -> last() and i < 4 xor (r -> last() or i < 4) endif) -> last()) .
---
--- rewrites: 232 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?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()
red eval(
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()
).

241
--- Collection Operations

--- --- Iterate

--- --- --- reject

--- --- --- --- Empty

--- --- --- --- Set -------------------------------

--- 1

--- ?let c = oclEmpty(Set(Integer)) in

--- c -> reject(i | i < 4) =

--- c -> iterate(i ; r:Set(Integer) = Set{} |

--- if i < 4 then r else r -> including(i) endif) red eval(

--- let c = oclEmpty(Set(Integer)) in

--- c -> iterate(i ; r:Set(Integer) = Set{} |

--- if i < 4 then r else r -> including(i) endif) red eval(

--- File: p05_o05_reject2iterate.maude

--- Expected Results:

--- true : Boolean

--- --- --- --- --- --- No Equal Values

--- 205 in Oms cpu (Oms real) (~ rewrites/second)
let c = Set{} in
  c -> reject(i | i < 4) =
    c -> iterate(i ; r : Set(Integer) = Set{} | if i < 4 then r else r -> including(i) endif)

--- ============
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> reject(i | i < 4) = c
--- -> iterate (i ; r : Set(Integer) = Set{} | if i < 4 then r else r -> including(i) endif)) .
--- rewrites: 28 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> reject(i | i < 4) =
--- c -> iterate(i; r:Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)
red eval(
let c = Bag{} in
  c -> reject(i | i < 4) =
    c -> iterate(i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)
).
--- ============
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> reject (i | i < 4) = c
--- -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)) .
--- rewrites: 28 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
--- c -> reject(i | i < 4) =
--- c -> iterate(i; r:Sequence(Integer) = Sequence{} | if i < 4 then r else r -> including(i) endif)
red eval(
let c = Sequence{} in
  c -> reject(i | i < 4) =
    c -> iterate(i ; r : Sequence(Integer) = Sequence{} | if i < 4 then r else r -> including(i) endif)
).
--- ============
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> reject (i | i < 4) = c
--- -> iterate (i ; r : Sequence(Integer) = Sequence{} | if i < 4 then r else r -> including(i) endif)) .
--- rewrites: 28 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- Singleton

--- --- --- --- Includes Undefined

--- --- --- --- --- Set -------------------------------
--- 4
--- ?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)
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections

--- --- --- --- --- Bag
--- 5
--- ?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)
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections

--- --- --- --- --- Sequence
--- 6
--- ?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)
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined

--- --- --- --- --- Expression e fulfilled

--- --- --- --- --- Set -------------------------------
--- 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)
red eval(
let c = Set{1} in
  c -> reject(i | i < 4) =
    c -> iterate(i ; r : Set(Integer) = Set{} |
      if i < 4 then r else r -> including(i) endif)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{1} in c -> reject (i | i < 4) =
--- c -> iterate (i ; r : Set(Integer) = Set{} | if i < 4 then r else r ->
--- including(i) endif)) .
--- rewrites: 58 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- Bag
--- 8
--- ?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)
red eval(
let c = Bag{1} in
c -> reject(i | i < 4) =
c -> iterate(i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)
).
--- =============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1} in c -> reject (i | i < 4) =
c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)) .
--- rewrites: 58 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- Sequence
--- 9
--- ?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)
red eval(
let c = Sequence{1} in
c -> reject(i | i < 4) =
c -> iterate(i ; r : Sequence(Integer) = Sequence{} | if i < 4 then r else r -> including(i) endif)
).
--- =============================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> reject (i | i <
--- 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | if i < 4 then r else r ->
--- including(i) endif)) .
--- rewrites: 57 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- Expression e not fulfilled
---
--- --- --- --- --- --- Set -------------------------------
--- 10
--- ?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)
red eval(let c = Set{4} in 
c -> reject(i | i < 4) =
c -> iterate(i ; r : Set(Integer) = Set{} | 
if i < 4 then r else r -> including(i) endif)) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{4} in c -> reject (i | i < 4) =
--- c -> iterate (i ; r : Set(Integer) = Set{} | if i < 4 then r else r ->
--- including(i) endif)) .
--- rewrites: 65 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- 11
--- ?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)
red eval(let c = Bag{4} in 
c -> reject(i | i < 4) =
c -> iterate(i ; r : Bag(Integer) = Bag{} | 
if i < 4 then r else r -> including(i) endif)) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{4} in c -> reject (i | i < 4) =
--- c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r ->
--- including(i) endif)) .
--- rewrites: 65 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 12
--- ?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)
red eval(let c = Sequence{4} in 
c -> reject(i | i < 4) =
c -> iterate(i ; r : Sequence(Integer) = Sequence{} | 
if i < 4 then r else r -> including(i) endif)) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{4} in c -> reject (i | i <
--- 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | if i < 4 then r
--- else r -> including(i) endif)) .
--- rewrites: 64 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- Many Elements
---
--- Includes Undefined
---

--- Set
---
--- 13
--- ?let c = Set{oclUndefined(Integer), 1, 2, 3} in
---  c -> iterate(i; r:Set(Integer) = oclEmpty(Set(Integer)) |
---       if i < 4 then r else r -> including(i) endif)
---
--- Expected Results:
---  true : Boolean
--- NO. invalid or null in collections
---
--- Bag
---
--- 14
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3,
---               oclUndefined(Integer)} in
---  c -> iterate(i; r:Bag(Integer) = oclEmpty(Bag(Integer)) |
---       if i < 4 then r else r -> including(i) endif)
---
--- Expected Results:
---  true : Boolean
--- NO. invalid or null in collections
---
--- No Equal Values
---
--- 15
--- ?let c = Bag{oclUndefined(Integer), 1, 2, 3} in
---  c -> iterate(i; r:Bag(Integer) = oclEmpty(Bag(Integer)) |
---       if i < 4 then r else r -> including(i) endif)
---
--- Expected Results:
---  true : Boolean
--- NO. invalid or null in collections
---
--- Sequence
---
--- 16
--- ?let c = Sequence{oclUndefined(Integer), 1, 2, 3,
---                     oclUndefined(Integer)} in
---  c -> iterate(i;
---             r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
---             if i < 4 then r else r -> including(i) endif)
---
--- NO. invalid or null in collections
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- No Equal Values
--- 17
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- --- Set ---------------------------------------------
--- 18
--- ?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)
red eval(
let c = Set{1, 2, 3} in
  c -> reject(i | i < 4) =
    c -> iterate(i ; r : Set(Integer) = Set{ } |
        if i < 4 then r else r -> including(i) endif)
).
--- ================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 3} in c -> reject (i | i <
--- 4) = c -> iterate (i ; r : Set(Integer) = Set{ } | if i < 4 then r else r ->
--- including(i) endif)) .
--- rewrites: 118 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Bag

--- --- --- --- --- --- --- --- Equal Values
--- 19
--- ?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)
red eval(
let c = Bag{1, 2, 3, 1} in
  c -> reject(i | i < 4) =
    c -> iterate(i ; r : Bag(Integer) = Bag{} |
        if i < 4 then r else r -> including(i) endif)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1, 2, 3, 1} in c -> reject (i |
--- i < 4) = c -> iterate (i ; r : Bag=Integer) = Bag{} | if i < 4 then r else
--- r -> including(i) endif)).
--- rewrites: 148 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 20
--- ?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)
red eval(
   let c = Bag{1, 2, 3} in
       c -> reject(i | i < 4) =
           c -> iterate(i ; r : Bag(Integer)
               = Bag{} | 
               if i < 4 then r else r -> including(i) endif)
).
--- ==================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3} in c -> reject (i | i < 
--- 4) = c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r ->
---     including(i) endif)) .
--- rewrites: 118 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- --- Equal Values
--- 21
--- ?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)
red eval(
   let c = Sequence{1, 2, 3, 1} in
       c -> reject(i | i < 4) =
           c -> iterate(i ;
               r : Sequence(Integer) = Sequence{} | 
               if i < 4 then r else r -> including(i) endif)
).
--- ==================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 1} in c -> reject
---  . (i | i < 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | if i <
---     4 then r else r -> including(i) endif)) .
--- rewrites: 144 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- --- No Equal Values
--- 22
--- ?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)
red eval(
let c = Sequence{1, 2, 3} in
  c -> reject(i | i < 4) =
  c -> iterate(i ;
       r = Sequence{} |  
       if i < 4 then r else r -> including(i) endif)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{1, 2, 3} in c -> reject (i | i < 4) = c -> iterate (i ; r = Sequence{} | if i < 4 then r else r -> including(i) endif)) .
--- rewrites: 114 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5, 4} in c -> reject (i | i < 4) = c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)) .
--- rewrites: 199 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- --- --- --- --- --- --- Equal Values (both not violating)
--- 25
---  ?let c = Bag{2, 3, 4, 5, 2} in
---    c -> reject(i | i < 4) =
---    c -> iterate(i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)
red eval(
    let c = Bag{2, 3, 4, 5, 2} in
        c -> reject(i | i < 4) =
            c -> iterate(i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5, 2} in c -> reject (i | i < 4) = c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)) .
--- rewrites: 192 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 26
---  ?let c = Bag{2, 3, 4, 5} in
---    c -> reject(i | i < 4) =
---    c -> iterate(i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)
red eval(
    let c = Bag{2, 3, 4, 5} in
        c -> reject(i | i < 4) =
            c -> iterate(i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5} in c -> reject (i | i < 4) = c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r else r -> including(i) endif)) .
--- rewrites: 162 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- --- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- --- Equal Values (both violating)
--- 27
---  ?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)
red eval(
let c = Sequence{2, 3, 4, 5, 4} in
c -> reject(i | i < 4) =
c -> iterate(i;
    r : Sequence(Integer) = Sequence{} |
    if i < 4 then r else r -> including(i) endif)
).

--- expected results:

--- true : Boolean

--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?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)
red eval(
let c = Sequence{2, 3, 4, 5, 2} in
c -> reject(i | i < 4) =
c -> iterate(i;
    r : Sequence(Integer) = Sequence{} |
    if i < 4 then r else r -> including(i) endif)
).

--- expected results:

--- true : Boolean

--- --- --- --- --- --- --- --- No Equal Values
--- 29
--- ?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)
red eval(
let c = Sequence{2, 3, 4, 5} in
c -> reject(i | i < 4) =
c -> iterate(i;
    r : Sequence(Integer) = Sequence{} |
    if i < 4 then r else r -> including(i) endif)
).

--- expected results:
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(2, 3, 4, 5) in c -> reject
--- . (i | i < 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | if i <
--- 4 then r else r -> including(i) endif)) .
--- rewrites: 158 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- Collection Operations
--- --- Iterate
--- --- --- select
--- --- --- --- Empty
--- --- --- --- --- Set ------------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> select(i | i < 4) =
--- c -> iterate(i ; r : Set(Integer) = Set{} | if i < 4 then r -> including(i) else r endif)
red eval(
let c = Set{} in
 c -> select(i | i < 4) =
  c -> iterate(i ; r : Set(Integer) = Set{} | if i < 4 then r -> including(i) else r endif)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} in c -> select (i | i < 4) = c
--- -> iterate (i ; r : Set(Integer) = Set{} | if i < 4 then r -> including(i)
--- else r endif)).
--- rewrites: 28 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- Bag
--- 2
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> select(i | i < 4) =
--- c -> iterate(i ; r : Bag(Integer) = Bag{} | if i < 4 then r -> including(i) else r endif)
red eval(
let c = Bag{} in
 c -> select(i | i < 4) =
  c -> iterate(i ; r : Bag(Integer) = Bag{} | if i < 4 then r -> including(i) else r endif)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in c -> select (i | i < 4) = c
--- -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r -> including(i)}
--- else r endif)) .
--- rewrites: 28 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Sequence
--- 3
--- ?let c = oclEmpty(Sequence(Integer)) in
--- c -> select(i | i < 4) =
--- c -> iterate(i ;
--- r:Sequence(Integer) = Sequence{} |
--- if i < 4 then r -> including(i) else r endif)
red eval(
let c = Sequence{} in
  c -> select(i | i < 4) =
  c -> iterate(i ;
    r : Sequence(Integer) = Sequence{} |
    if i < 4 then r -> including(i) else r endif)) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in c -> select (i | i <
--- 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | if i < 4 then r
--- -> including(i) else r endif)) .
--- rewrites: 28 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Singleton
---
--- --- --- --- --- Includes Undefined
---
--- --- --- --- --- --- Set ..............................
--- 4
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Bag
--- 6
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Sequence
--- 6
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
--- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 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)
red eval(
let c = Set{1} in
  c -> select(i | i < 4) =
    c -> iterate(i ; r : Set(Integer) = Set{} |
      if i < 4 then r -> including(i) else r endif)
).  
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{1} in c -> select (i | i < 4) =
---   c -> iterate (i ; r : Set(Integer) = Set{} | if i < 4 then r -> including(
---     i) else r endif)).
--- rewrites: 65 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- Bag
--- 8
--- ?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)
red eval(
let c = Bag{1} in
  c -> select(i | i < 4) =
    c -> iterate(i ; r : Bag(Integer) = Bag{} |
      if i < 4 then r -> including(i) else r endif)
).  
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1} in c -> select (i | i < 4) =
---   c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r -> including(
---     i) else r endif)).
--- rewrites: 65 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- Sequence
--- 255
--- 9
--- ?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)
red eval(
let c = Sequence{1} in
  c -> select(i | i < 4) =
  c -> iterate(i;
    r : Sequence(Integer) = Sequence{} |
    if i < 4 then r -> including(i) else r endif)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1} in c -> select (i | i < 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | if i < 4 then r -> including(i) else r endif)) .
--- rewrites: 64 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- Set ---------------------------------------------
--- 10
--- ?let c = Set{4} in
--- c -> select(i | i < 4) =
--- c -> iterate(i;
--- r : Set(Integer) = Set{} |
--- if i < 4 then r -> including(i) else r endif)
red eval(
let c = Set{4} in
  c -> select(i | i < 4) =
  c -> iterate(i;
    r : Set(Integer) = Set{} |
    if i < 4 then r -> including(i) else r endif)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{4} in c -> select (i | i < 4) = c -> iterate (i ; r : Set(Integer) = Set{} | if i < 4 then r -> including(i) else r endif)) .
--- rewrites: 58 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
--- 11
--- ?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)
red eval(
let c = Bag{4} in
  c -> select(i | i < 4) =
  c -> iterate(i;
    r : Bag(Integer) = Bag{} |
    if i < 4 then r -> including(i) else r endif)
).
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag\{4\} in c -> select (i | i < 4) =
---   c -> iterate (i ; r : Bag\(\text{Integer}\) = Bag\{\} | if i < 4 then r -> including(i) else r endif)) .
--- rewrites: 58 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- 12
--- ?let c = Sequence\{4\} in
---   c -> select(i | i < 4) =
---     c -> iterate(i;
---       r:Sequence\(\text{Integer}\) = oclEmpty(Sequence\(\text{Integer}\)) | 
---       if i < 4 then r -> including(i) else r endif)
---
red eval(
let c = Sequence\{4\} in
  c -> select(i | i < 4) =
  c -> iterate(i ;
       r : Sequence\(\text{Integer}\) = Sequence\{\} | 
       if i < 4 then r -> including(i) else r endif)
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence\{4\} in c -> select (i | i < 4) =
---   c -> iterate (i ; r : Sequence\(\text{Integer}\) = Sequence\{\} | if i < 4 then r -> including(i) else r endif)) .
--- rewrites: 57 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---   NO. invalid or null in collections
--- --- --- --- --- --- Many Elements
--- --- --- --- --- --- Includes Undefined
--- --- --- --- --- --- Set -------------------------------
--- 13
--- ?let c = Set\(\text{oclUndefined}(\text{Integer}), 1, 2, 3\) in
---   c -> select(i | i < 4) =
---     c -> iterate(i; r:Set\(\text{Integer}\) = oclEmpty(Set\(\text{Integer}\)) | 
---     if i < 4 then r -> including(i) else r endif)
---
--- Expected Results:
---   true : Boolean
--- --- --- --- --- --- Bag
--- --- --- --- --- --- Equal Values
--- 14
--- ?let c = Bag\(\text{oclUndefined}(\text{Integer}), 1, 2, 3,\)
---   \text{oclUndefined}(\text{Integer})\} in
---   c -> select(i | i < 4) =
---     c -> iterate(i; r:Bag\(\text{Integer}\) = oclEmpty(Bag\(\text{Integer}\)) | 
---     if i < 4 then r -> including(i) else r endif)

--- 257
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- No Equal Values
--- 15
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- --- No Equal Values
--- 16
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- --- No Equal Values
--- 17
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- --- Expression e fulfilled
--- --- --- --- --- --- --- Set ------------------------------------------
--- 18
--- ?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)
---
---
--- red eval(
let c = Set{1, 2, 3} in
c -> select(i | i < 4) =
    c -> iterate(i ; r = Set{} |
        if i < 4 then r -> including(i) else r endif)
).
--- ==================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 3} in c -> select (i | i
--- < 4) = c -> iterate (i, r = Set{} | if i < 4 then r -> including(i) else r
--- endif)).
--- rewrites: 128 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- Bag
---
--- --- --- --- --- --- --- Equal Values
--- 19
--- ?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)
red eval(
let c = Bag{1, 2, 3, 1} in
c -> select(i | i < 4) =
    c -> iterate(i; r : Bag(Integer) = Bag{} |
        if i < 4 then r -> including(i) else r endif)
).
--- ==================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3, 1} in c -> select (i | i
--- < 4) = c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r ->
---     including(i) else r endif)).
--- rewrites: 176 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 20
--- ?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)
red eval(
let c = Bag{1, 2, 3} in
    c -> select(i | i < 4) =
        c -> iterate(i ; r : Bag(Integer) = Bag{} |
            if i < 4 then r -> including(i) else r endif)
).
--- ==================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2, 3} in c -> select (i | i
--- < 4) = c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r ->
---     including(i) else r endif)).
--- rewrites: 139 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- 259
--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values
--- 21
--- ?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)
red eval(
let c = Sequence{1, 2, 3, 1} in
c -> select(i | i < 4) =
c -> iterate(i;
     r : Sequence(Integer) = Sequence{} |
     if i < 4 then r -> including(i) else r endif)
).
--- -------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3, 1} in c -> select
--- (i | i < 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | if i <
--- 4 then r -> including(i) else r endif)) .
--- rewrites: 172 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- No Equal Values
--- 22
--- ?let c = Sequence{1, 2, 3} in
c -> select(i | i < 4) =
c -> iterate(i;
     r : Sequence(Integer) = Sequence{} |
     if i < 4 then r -> including(i) else r endif)
red eval(
let c = Sequence{1, 2, 3} in
c -> select(i | i < 4) =
c -> iterate(i;
     r : Sequence(Integer) = Sequence{} |
     if i < 4 then r -> including(i) else r endif)
).
--- -------------------------------------------
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2, 3} in c -> select
--- (i | i < 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | if i < 4
--- then r -> including(i) else r endif)) .
--- rewrites: 136 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- --- --- Expression e not fulfilled
--- --- --- --- --- --- --- --- Set ------------------------------------------
--- 23
--- ?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)
red eval(
  let c = Set[2, 3, 4, 5] in 
  c -> select(i | i < 4) = 
    c -> iterate(i ; r : Set(Integer) = Set{} | 
      if i < 4 then r -> including(i) else r endif)
).
--- ====================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set[2, 3, 4, 5] in c -> select (i | 
--- i < 4) = c -> iterate (i ; r : Set(Integer) = Set{} | if i < 4 then r -> 
--- including(i) else r endif)) .
--- rewrites: 162 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- Bag
---
--- --- --- --- --- --- --- Equal Values (both violating)
--- 24
--- ?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)
red eval(
  let c = Bag[2, 3, 4, 5, 4] in 
  c -> select(i | i < 4) = 
    c -> iterate(i ; r : Bag(Integer) = Bag{} | 
      if i < 4 then r -> including(i) else r endif)
).
--- ====================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag[2, 3, 4, 5, 4] in c -> select (i | 
--- i < 4) = c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r -> 
--- including(i) else r endif)) .
--- rewrites: 192 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Equal Values (both not violating)
--- 25
--- ?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)
red eval(
  let c = Bag[2, 3, 4, 5, 2] in 
  c -> select(i | i < 4) = 
    c -> iterate(i ; r : Bag(Integer) = Bag{} | 
      if i < 4 then r -> including(i) else r endif)
).
--- ====================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag[2, 3, 4, 5, 2] in c -> select (i | 
--- i < 4) = c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r -> 
--- including(i) else r endif)) .
--- rewrites: 199 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
---
---
---
---
---

261
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- No Equal Values
--- 26
--- ?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)
red eval(
let c = Bag{2, 3, 4, 5} in
c -> select(i | i < 4) =
c -> iterate(i ; r : Bag(Integer) = Bag{} |
    if i < 4 then r -> including(i) else r endif)
).
--- =====================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2, 3, 4, 5} in c -> select (i |
--- i < 4) = c -> iterate (i ; r : Bag(Integer) = Bag{} | if i < 4 then r ->
--- including(i) else r endif)) .
--- rewrites: 162 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values (both violating)
--- 27
--- ?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)
red eval(
let c = Sequence{2, 3, 4, 5, 4} in
c -> select(i | i < 4) =
c -> iterate(i ;
    r : Sequence(Integer) = Sequence{} |
    if i < 4 then r -> including(i) else r endif)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 4} in c ->
--- select (i | i < 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} |
--- if i < 4 then r -> including(i) else r endif)) .
--- rewrites: 187 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- Equal Values (both not violating)
--- 28
--- ?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)
red eval(

let c = Sequence{2, 3, 4, 5, 2} in
  c -> select(i | i < 4) =
    c -> iter(i;
      r : Sequence(Integer) = Sequence{} |
       if i < 4 then r -> including(i) else r endif)
).

---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5, 2} in c ->
  select (i | i < 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} |
  if i < 4 then r -> including(i) else r endif)).
--- rewrites: 194 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- --- --- No Equal Values
--- 29
--- 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)
red eval(
  let c = Sequence{2, 3, 4, 5} in
  c -> select(i | i < 4) =
    c -> iterate(i;
      r : Sequence(Integer) = Sequence{} |
       if i < 4 then r -> including(i) else r endif)
).

--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2, 3, 4, 5} in c -> select
  (i | i < 4) = c -> iterate (i ; r : Sequence(Integer) = Sequence{} | if i <
  4 then r -> including(i) else r endif)).
--- rewrites: 158 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- Technical Relationships
--- --- iterate with let
--- --- --- Empty
--- --- --- --- Set ---------------------------------------------------
--- 1
--- 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)
red eval(
let c = Set{} 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)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{} 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).
--- rewrites: 42 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Bag
---
--- ?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)
red eval(
let c = Bag{} 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)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} 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).
--- rewrites: 42 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Sequence
---
--- ?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)
red eval(
let c = Sequence{} 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)
).

--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} 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)) .
--- rewrites: 42 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- Singleton
--- --- --- --- Includes Undefined

--- --- --- --- --- Set -----------------------------------------------
--- 4
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Bag
--- 5
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- --- Sequence
--- 6
--- ?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)
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections
---
--- --- --- --- Excludes Undefined

265
--- --- --- --- --- Set ------------------------------------------------
--- 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)
red eval(
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)
).
--- ---------------------------------------------------------------
--- reduce in BENCHMARK-TEST-BS : eval(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))
--- rewrites: 92 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Bag
--- 8
--- ?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)
red eval(
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)
).
--- ---------------------------------------------------------------
--- reduce in BENCHMARK-TEST-BS : eval(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))
--- rewrites: 92 in 4ms cpu (0ms real) (23000 rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Sequence
--- 9
--- ?let c = Sequence{2} in
c -> iterate(elem; r:Bag(Integer) = Bag{1} |
   r -> including(r -> size()+elem))

--- --- --- --- --- --
red eval(
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)
).

reduce in BENCHMARK-TEST-B5 : eval(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)).

rewrites: 91 in Oms cpu (Oms real) (~ rewrites/second)
result Bool: true

Expected Results:
  true : Boolean

--- --- --- Many Elements

--- --- --- --- Includes Undefined

--- --- --- --- --- Set -----------------------------------------------

10
?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)

Expected Results:
  true : Boolean
  NO. invalid or null in collections

--- --- --- --- Bag

--- --- --- --- Equal Values

11
?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)

Expected Results:
  true : Boolean
  NO. invalid or null in collections

--- --- --- --- No Equal Values

12
?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
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Sequence

--- --- --- --- --- Equal Values
--- 13
--- ?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
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- No Equal Values
--- 14
--- ?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
---
--- Expected Results:
--- true : Boolean
--- NO. invalid or null in collections

--- --- --- --- --- Excludes Undefined

--- --- --- --- --- Set --------------------------------------------------
--- 15
--- ?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
red eval(
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)
).
--- ===============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2, 3, -10} in c -> iterate ( elem ; r : Bag(Integer) = Bag{1} | r -> including(r -> size() + elem)) = c

268
--- iterate (elem ; r : Bag(Integer) = Bag{1} | let res = r -> including(r
--- size()+elem) in res)) .
--- rewrites: 281 in 0ms cpu (2ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- Bag
---
--- Equal Values
--- 16
---
--- 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)
red eval(
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)
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(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)) .
--- rewrites: 356 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- No Equal Values
--- 17
---
--- 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)
red eval(
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)
).
---
--- reduce in BENCHMARK-TEST-B5 : eval(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)) .
--- rewrites: 281 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Sequence

--- --- --- --- --- --- Equal Values
--- 18
--- ?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)
red eval(
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)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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)) .
--- rewrites: 351 in 4ms cpu (1ms real) (87750 rewrites/second)
--- result Bool: true
---

--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- No Equal Values
--- 19
--- ?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)
red eval(
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)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(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)) .
--- rewrites: 277 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---

--- Expected Results:
--- true : Boolean
--- Technical Relationships
--- --- Singleton to Value
--- --- --- Empty
--- --- --- --- Set

--- 1
--- ?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

red eval( let c = Set{} in
    if c -> size() = 1 then c -> any(x | true)
    else null endif =
    if c -> size() = 1 then
    c -> iterate(elem ; r : Integer = null | elem)
    else null endif
)

--- expected results:
--- true : Boolean

--- 2
--- ?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) | elem)

red eval( let c = Set{} in
    if c -> size() = 1 then c -> any(x | true)
    else null endif =
    c -> iterate(elem ;
    r : Sequence(OclAny) = Sequence(null,false) |
    if r -> at(2) = false then Sequence(elem,true)
    else Sequence(null,true) endif) -> at(1)
)

--- expected results:
--- true : Boolean

--- reduce in BENCHMARK-TEST BS : eval(let c = Set{} in if c -> size() = 1 then c
--- -> any(x | true) else null endif = if c -> size() = 1 then c -> iterate (elem ;
--- r : Integer = null | elem) else null endif) .
--- rewrites: 32 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true

--- reduce in BENCHMARK-TEST BS : eval(let c = Set{} in if c -> size() = 1 then c
--- -> any(x | true) else null endif = c -> iterate (elem ;
--- r : Sequence(OclAny) = Sequence(null,false) |
--- if r -> at(2) = false then Sequence(elem,true)
--- else Sequence(null,true) endif) -> at(1) ) .
--- rewrites: 32 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true

--- expected results:
--- true : Boolean

--- 271
--- rewrites: 46 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Bag
--- 3
--- ?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
---
red eval(
let c = Bag{} in
if c -> size() = 1 then c -> any(x | true)
else null endif =
if c -> size() = 1 then
  c -> iterate(elem ; r : Integer = null | elem)
else null endif
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in if c -> size() = 1 then c
---  
--- -> any (x | true) else null endif = if c -> size() = 1 then c -> iterate ( 
---  
--- elem ; r : Integer = null | elem) else null endif).
--- rewrites: 32 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Bag
--- 4
--- ?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{null,true} endif) -> at(1)
---
red eval(
let c = Bag{} in
if c -> size() = 1 then c -> any(x | true)
else null endif =
if c -> size() = 1 then
  c -> iterate(elem ;
  
  r : Sequence(OclAny) = Sequence{null,false} | 
  
  if r -> at(2) = false then Sequence{elem,true}
  
  else Sequence{null,true} endif) -> at(1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{} in if c -> size() = 1 then c
---  
--- -> any (x | true) else null endif = c -> iterate (elem ; r : Sequence(
---  
--- OclAny) = Sequence{null, false} | if r -> at(2) = false then Sequence{elem,
---  
--- true} else Sequence{null, true} endif) -> at(1)) .
--- rewrites: 46 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
272
--- --- --- --- Sequence
--- 5
--- ?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
--- end eval

red eval(
let c = Sequence{} in
    if c -> size() = 1 then c -> any(x | true)
    else null endif =
    if c -> size() = 1 then
    c -> iterate(elem; r : Integer = null | elem)
    else null endif
).
--- ==================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in if c -> size() = 1
---     then c -> any (x | true) else null endif = if c -> size() = 1 then c ->
---     iterate (elem ; r : Integer = null | elem) else null endif).
--- rewrites: 32 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- 6
--- ?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{null,true} endif) -> at(1)
--- end eval

red eval(
let c = Sequence{} in
    if c -> size() = 1 then c -> any(x | true)
    else null endif =
    c -> iterate(elem ;
        r : Sequence(OclAny) = Sequence{null,false} |
        if r -> at(2) = false then Sequence{elem,true}
        else Sequence{null,true} endif) -> at(1)
).
--- ==================================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{} in if c -> size() = 1
---     then c -> any (x | true) else null endif = c -> iterate (elem ; r :
---         Sequence(OclAny) = Sequence{null, false} | if r -> at(2) = false then
---         Sequence{elem, true} else Sequence{null, true} endif) -> at(1)) .
--- rewrites: 46 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- Singleton
--- --- --- --- Includes Undefined
--- --- --- --- --- Set ---------------------------------------------------
--- 7
--- ?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
red eval(
let c = Set{null} in
  if c -> size() = 1 then c -> any(x | true)
  else null endif =
  if c -> size() = 1 then
    c -> iterate(elem ; r : Integer = null | elem)
    else null endif
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{null} in if c -> size() = 1 then
--- c -> any (x | true) else null endif = if c -> size() = 1 then c -> iterate
--- . (elem ; r : Integer = null | elem) else null endif).
--- rewrites: 68 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 8
--- ?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{null,true} endif) -> at(1)
red eval(
let c = Set{null} in
  if c -> size() = 1 then c -> any(x | true)
  else null endif =
  c -> iterate(elem ;
    r : Sequence(OclAny) = Sequence{null,false} |
    if r -> at(2) = false then Sequence{elem,true}
    else Sequence{null,true} endif) -> at(1)
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{null} in if c -> size() = 1 then
--- c -> any (x | true) else null endif = c -> iterate (elem ; r : Sequence(
--- OclAny) = Sequence(null, false) | if r -> at(2) = false then Sequence(elem,
--- true) else Sequence(null, true) endif) -> at(1)) .
--- rewrites: 95 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- Bag
--- 9
--- ?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
red eval(
let c = Bag{null} in
if c -> size() = 1 then c -> any(x | true)
else null endif =
if c -> size() = 1 then
c -> iterate(elem; r : Integer = null | elem)
else null endif
).  
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{null} in if c -> size() = 1 then
--- c -> any (x | true) else null endif = if c -> size() = 1 then c -> iterate
--- . (elem ; r : Integer = null | elem) else null endif).
--- rewrites: 68 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 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(null,true) endif) -> at(1)
red eval(
let c = Bag{null} in
if c -> size() = 1 then c -> any(x | true)
else null endif =
c -> iterate(elem;
 r : Sequence(OclAny) = Sequence(null,false) |
 if r -> at(2) = false then Sequence{elem,true}
 else Sequence(null,true) endif) -> at(1)
).  
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{null} in if c -> size() = 1 then
--- c -> any (x | true) else null endif = c -> iterate (elem ; r : Sequence(
--- OclAny) = Sequence(null, false) | if r -> at(2) = false then Sequence(elem,
--- true) else Sequence(null, true) endif) -> at(1)) .
--- rewrites: 95 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- Sequence
--- 11
--- ?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
red eval(
let c = Sequence{null} in
if c -> size() = 1 then c -> any(x | true) else null endif =
if c -> size() = 1 then
    c -> iterate(elem ; r : Integer = null | elem)
else null endif
).
--- ================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{null} in if c -> size() = 1
---    then c -> any(x | true) else null endif = if c -> size() = 1 then c ->
---    iterate (elem ; r : Integer = null | elem) else null endif).
--- rewrites: 67 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
--- 12
--- ?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(null,true,endef) -> at(1)
red eval(
let c = Sequence{null} in
    if c -> size() = 1 then c -> any(x | true)
else null endif =
    c -> iterate(elem ;
        r : Sequence(OclAny) = Sequence{null,false} |
        if r -> at(2) = false then Sequence{elem,true}
else Sequence(null,true,endef) -> at(1)
).
--- ================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Sequence{null} in if c -> size() = 1
---    then c -> any(x | true) else null endif = c -> iterate (elem ; r :
---        Sequence(OclAny) = Sequence{null,false} | if r -> at(2) = false then
---        Sequence(elem,true) else Sequence(null,true,endef) -> at(1)) .
--- rewrites: 94 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
--- --- --- --- Excludes Undefined
--- --- --- --- --- Set -------------------------------
--- 13
--- ?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
red eval(
let c = Set{2} in
    if c -> size() = 1 then c -> any(x | true)
else null endif =
    if c -> size() = 1 then
276
c -> iterate(elem ; r : Integer = null | elem) 
    else null endif
)

--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{2} in if c -> size() = 1 then c
---   -> any (x | true) else null endif = if c -> size() = 1 then c -> iterate 
---   elem ; r : Integer = null | elem) else null endif)
--- rewrites: 68 in 0ms cpu (0ms real) (` rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 14
--- ?let c = Set{2} in
---   if c -> size() = 1 then c -> any(true)
---   else oclUndefined(Integer) endif =
---   c -> iterate(elem ;
---     r : Sequence(OclAny) = Sequence{null, false} | if r -> at(2) = false then Sequence{elem, true} 
---     else Sequence{null, true} endif) -> at(1)
red eval(
let c = Set{2} in
   if c -> size() = 1 then c -> any(x | true)
   else null endif =
   c -> iterate(elem ;
     r : Sequence(OclAny) = Sequence{null, false} | if r -> at(2) = false then Sequence{elem, true} 
     else Sequence{null, true} endif) -> at(1)
)

--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{2} in if c -> size() = 1 then c
---   -> iterate(elem ; r : Integer = null | elem) else null endif = if c -> size() = 1 then c -> iterate 
---     elem ; r : Integer = null | elem) else null endif)
--- rewrites: 95 in 0ms cpu (0ms real) (` rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 15
--- ?let c = Bag{2} in
---   if c -> size() = 1 then c -> any(true)
---   else oclUndefined(Integer) endif =
---   c -> iterate(elem ;
---     r : Integer = oclUndefined(Integer) | elem)
---     else oclUndefined(Integer) endif
eval(
let c = Bag{2} in
   if c -> size() = 1 then c -> any(x | true)
   else null endif =
   if c -> size() = 1 then
   c -> iterate(elem ;
     r : Integer = null | elem)
     else null endif
)

--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2} in if c -> size() = 1 then c
---   -> iterate(elem ; r : Integer = null | elem) else null endif = if c -> size() = 1 then c -> iterate 
---     elem ; r : Integer = null | elem) else null endif)
--- rewrites: 95 in 0ms cpu (0ms real) (` rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 277
---> any (x | true) else null endif = if c -> size() = 1 then c -> iterate (elem ; r : Integer = null | elem) else null endif).
---> result Bool: true
--->
---> Expected Results:
---> true : Boolean

---> 16
---> ?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{null,true} endif) -> at(1)
red eval(
let c = Bag{2} in
    if c -> size() = 1 then c -> any(x | true)
    else null endif =
        c -> iterate(elem;  
        r : Sequence(OclAny) = Sequence{null,false} |
            if r -> at(2) = false then Sequence{elem,true}
            else Sequence{null,true} endif) -> at(1)
).
---> reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{2} in if c -> size() = 1 then c
---> -> any (x | true) else null endif = c -> iterate (elem ; r : Sequence(  
---> OclAny) = Sequence{null, false} | if r -> at(2) = false then Sequence{elem,  
---> true} else Sequence{null, true} endif) -> at(1)).
---> rewrites: 95 in Oms cpu (Oms real) (~ rewrites/second)
---> result Bool: true
--->
---> Expected Results:
---> true : Boolean

---> --- --- --- --- Sequence
---> 17
---> ?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
red eval(
let c = Sequence{2} in
    if c -> size() = 1 then c -> any(x | true)
    else null endif =
        if c -> size() = 1 then
            c -> iterate(elem ; r : Integer = null | elem)
            else null endif
).
---> reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2} in if c -> size() = 1
---> then c -> any (x | true) else null endif = if c -> size() = 1 then c ->  
---> iterate (elem ; r : Integer = null | elem) else null endif).
---> rewrites: 67 in Oms cpu (Oms real) (~ rewrites/second)
---> result Bool: true
--->
--- Expected Results:
--- true : Boolean
---
--- 18
--- ?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{null,true} endif) -> at(1)
red eval(
let c = Sequence{2} in
if c -> size() = 1 then c -> any(x | true)
else null endif =
c -> iterate(elem;
r : Sequence(OclAny) = Sequence{null,false} |
if r -> at(2) = false then Sequence{elem,true}
else Sequence{null,true} endif) -> at(1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{2} in if c -> size() = 1
--- then c -> any (x | true) else null endif = c -> iterate (elem ; r :
--- Sequence(OclAny) = Sequence{null,false} | if r -> at(2) = false then
--- Sequence{elem,true} else Sequence{null,true} endif) -> at(1)) .
--- rewrites: 94 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- Many Elements
--- --- --- --- Includes Undefined
--- --- --- --- --- Set -----------------------------------------------
--- 19
--- ?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
red eval(
let c = Set{null, 2} in
if c -> size() = 1 then c -> any(x | true)
else null endif =
if c -> size() = 1 then
  c -> iterate(elem ; r : Integer = null | elem)
else null endif
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{null, 2} in if c -> size() = 1
--- then c -> any (x | true) else null endif = if c -> size() = 1 then c ->
--- iterate (elem ; r : Integer = null | elem) else null endif .
--- rewrites: 53 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 20
--- ?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{null,true} endif) -> at(1)
red eval(
  let c = Set{null, 2} in
  if c -> size() = 1 then c -> any(x | true) else
    null endif =
    c -> iterate(elem;
      r : Sequence(OclAny) = Sequence{null,false} |
      if r -> at(2) = false then Sequence{elem,true}
      else Sequence{null,true} endif) -> at(1)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{null, 2} in if c -> size() = 1
---   then c -> any(x | true) else null endif = c -> iterate (elem ; r :
---       Sequence(OclAny) = Sequence{null,false} | if r -> at(2) = false then
---       Sequence{elem,true} else Sequence{null,true} endif) -> at(1)) .
--- rewrites: 127 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- --- Expected Results:
--- true : Boolean

--- --- --- --- --- Bag
--- 21
--- ?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
red eval(
  let c = Bag{null, 2} in
  if c -> size() = 1 then c -> any(x | true) else
    null endif =
    if c -> size() = 1 then
      c -> iterate(elem ; r : Integer = null | elem)
    else null endif
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{null, 2} in if c -> size() = 1
---   then c -> any(x | true) else null endif = if c -> size() = 1 then c ->
---   iterate (elem ; r : Integer = null | elem) else null endif) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- --- Expected Results:
--- true : Boolean

--- 22
--- ?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(null,true) endif) -> at(1)
red eval(
let c = Bag{null, 2} in
if c -> size() = 1 then c -> any(x | true)
else null endif =
c -> iterate(elem; r : Sequence(OclAny) = Sequence(null,false) |
if r -> at(2) = false then Sequence(elem,true)  
else Sequence(null,true) endif) -> at(1)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{null, 2} in if c -> size() = 1
--- then c -> any (x | true) else null endif = c -> iterate (elem ; r :
--- Sequence(0clAny) = Sequence(null, false) ) if r -> at(2) = false then
--- Sequence(elem, true) else Sequence(null, true) endif) -> at(1)) .
--- rewrites: 127 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- --- Sequence
--- 23
--- ?let c = Sequence(oclUndefined(Integer), 2) in
--- if c -> size() = 1 then c -> any(true)
--- else oclUndefined(Integer) endif =
--- c -> iterate(elem; r:Integer = oclUndefined(Integer) | elem)
--- else oclUndefined(Integer) endif
red eval(
let c = Sequence{null, 2} in
if c -> size() = 1 then c -> any(x | true)
else null endif =
if c -> size() = 1 then
  c -> iterate(elem ; r : Integer = null | elem)
else null endif
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{null, 2} in if c -> size()
--- = 1 then c -> any (x | true) else null endif = if c -> size() = 1 then c ->
--- iterate (elem ; r : Integer = null | elem) else null endif) .
--- rewrites: 51 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 24
--- ?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(null,true) endif) -> at(1)
---
--- 281
red eval(
let c = Sequence(null, 2) in
  if c -> size() = 1 then c -> any(x | true)
  else null endif =
  c -> iterate(elem ;
    r : Sequence(OclAny) = Sequence(null,false) |
    if r -> at(2) = false then Sequence(elem,true)
    else Sequence(null,true) endif) -> at(1)
).  
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence(null, 2) in if c -> size() = 1 then
c -> any (x | true) else null endif = c -> iterate (elem ; r :
Sequence(OclAny) = Sequence(null, false) | if r -> at(2) = false then
Sequence(elem,true) else Sequence(null, true) endif) -> at(1)) .
--- rewrites: 125 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Excludes Undefined
---
--- --- --- --- --- Set  ------------------------------------------------
--- 25
--- ?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
red eval(
let c = Set{1, 2} in
  if c -> size() = 1 then c -> any(x | true)
  else null endif =
  if c -> size() = 1 then
  c -> iterate(elem; r:Integer = null | elem)
  else null endif
).  
--- ==============================================================
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Set{1, 2} in if c -> size() = 1 then
c -> any (x | true) else null endif = if c -> size() = 1 then
c -> iterate (elem ; r : Integer = null | elem) else null endif).
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 26
--- ?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(null,true) endif) -> at(1)
red eval(
let c = Set{1, 2} in
  if c -> size() = 1 then c -> any(x | true)
else null endif =
c -> iterate(elem ;
   r : Sequence(OclAny) = Sequence(null,false) |
   if r -> at(2) = false then Sequence(elem,true)
   else Sequence(null,true) endif) -> at(1)
).

--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Set{1, 2} in if c -> size() = 1 then
c -> any (x | true) else null endif = c -> iterate (elem ; r : Sequence(
OclAny) = Sequence(null, false) | if r -> at(2) = false then Sequence(elem,
true) else Sequence(null, true) endif) -> at(1)) .
--- rewrites: 127 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- cancer --- --- --- --- Bag
--- 27
--- ?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 = null | elem)
--- else oclUndefined(Integer) endif
red eval(
let c = Bag{1, 2} in
   if c -> size() = 1 then c -> any(x | true)
   else null endif =
   if c -> size() = 1 then
   c -> iterate(elem ; r : Integer = null | elem)
   else null endif
).

--- ==============================================================
--- reduce in BENCHMARK-TEST-BS : eval(let c = Bag{1, 2} in if c -> size() = 1 then
c -> any (x | true) else null endif = if c -> size() = 1 then c -> iterate
elem ; r : Integer = oclUndefined(Integer) | elem)
--- else oclUndefined(Integer) endif .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- cancer --- --- --- --- Bag
--- 28
--- ?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(null,true) endif) -> at(1)
red eval(
let c = Bag{1, 2} in
   if c -> size() = 1 then c -> any(x | true)
   else null endif =
c -> iterate(elem ;
   r : Sequence(OclAny) = Sequence(null,false) |
   if r -> at(2) = false then Sequence(elem,true)
   else Sequence(null,true) endif) -> at(1)
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Bag{1, 2} in if c -> size() = 1 then
---   c -> any (x | true) else null endif = c -> iterate (elem ; r : Sequence(OclAny) = Sequence{null, false} | if r -> at(2) = false then Sequence(elem, true) else Sequence(null, true) endif) -> at(1)).
--- rewrites: 127 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- Sequence
--- 29
--- ?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
--- red eval(
--- let c = Sequence{1, 2} in
---   if c -> size() = 1 then c -> any(x | true)
---     else null endif =
---     if c -> size() = 1 then
---       c -> iterate(elem ; r : Integer = null | elem)
---     else null endif
--- ) .
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2} in if c -> size() = 1 then
---   c -> any (x | true) else null endif = if c -> size() = 1 then c ->
---     iterate (elem ; r : Integer = null | elem) else null endif).
--- rewrites: 51 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- Sequence
--- 30
--- ?let c = Sequence{1, 2} in
---   if c -> size() = 1 then c -> any(x | 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(null,true) endif) -> at(1)
--- red eval(
--- let c = Sequence{1, 2} in
---   if c -> size() = 1 then c -> any(x | true)
---     else null endif =
---     c -> iterate(elem ;
---       r : Sequence(OclAny) = Sequence{null,false} |
---       if r -> at(2) = false then Sequence(elem,true)
---     else Sequence(null,true) endif) -> at(1)
--- ) .
---
--- reduce in BENCHMARK-TEST-B5 : eval(let c = Sequence{1, 2} in if c -> size() = 1 then
---   c -> any (x | true) else null endif = c -> iterate (elem ; r :
---     Sequence(OclAny) = Sequence{null,false} | if r -> at(2) = false then
--- Sequence(elem, true) else Sequence(null, true) endif -> at(1)) .
--- rewrites: 125 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- Technical Relationships
--- --- Conversion to Sequences
--- --- --- Empty
--- --- --- --- Set ---------------------------------------------------
--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> asSequence() = c -> asBag() -> asSequence() red eval( let c = Set{} in c -> asSequence() = c -> asBag() -> asSequence() ) . --- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in c -> asSequence() = c -> asBag() -> asSequence()) . --- rewrites: 20 in 0ms cpu (0ms real) (~ rewrites/second) --- result Bool: true --- Expected Results:
--- true : Boolean --- 2
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> asSequence() =
--- c -> iterate(elem; --- r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |  --- r -> including(elem)) red eval( let c = Set{} in c -> asSequence() = c -> iterate(elem;  --- r : Sequence(Integer) = Sequence{} | r -> including(elem)) ) . --- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in c -> asSequence() = c -> iterate (elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem))) --- rewrites: 26 in 0ms cpu (0ms real) (~ rewrites/second) --- result Bool: true --- Expected Results:
--- true : Boolean --- 3
--- ?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 red eval( let c = Set{} in c -> asSequence() = c -> iterate(u ;
\[ r : \text{Tuple(\text{theSet} : \text{Set(Integer)}, \text{theSeq} : \text{Sequence(Integer)})} = \]
\[ \text{Tuple(\text{theSet} = c, \text{theSeq} = \text{Sequence{}{})} | \]
\[ \text{let e} = r . \text{theSet} \rightarrow \text{any}(x \mid \text{true}) \in \]
\[ \text{Tuple(\text{theSet} = r . \text{theSet} \rightarrow \text{excluding(e)}, \]
\[ \text{theSeq} = r . \text{theSeq} \rightarrow \text{including(e)}) . \text{theSeq}} \]

--- ================================
--- reduce in \text{BENCHMARK-TEST-B6} : eval(let c = \text{Set{} in c -> asSequence()} = c \rightarrow
--- iterate (u ; r : \text{Tuple(\text{theSet} : \text{Set(Integer)}, \text{theSeq} : \text{Sequence(Integer)})} =
--- \text{Tuple(\text{theSet} = c, \text{theSeq} = \text{Sequence{}{})} | let e = r . \text{theSet} \rightarrow \text{any}(x \mid
--- \text{true}) \in \text{Tuple(\text{theSet} = r . \text{theSet} \rightarrow \text{excluding(e)}, \text{theSeq} = r . \text{theSeq} \rightarrow
--- \text{including(e)}) . \text{theSeq}} .
--- \text{rewrites: 40 in Oms cpu (Oms real) (~ rewrites/second)}
--- \text{result Bool: true}
---
--- Expected Results:
--- \text{true : Boolean}
---
---\text{4}
--- \text{?let c = oclEmpty(Set(Integer{})) in}
--- \text{c -> asSequence()} = \text{Sequence{c} -> flatten()}
\text{red eval( let c = \text{Set{}} in}
\text{c -> asSequence()} = \text{Sequence{c} -> flatten()}
\text{)} .
--- ================================
--- reduce in \text{BENCHMARK-TEST-B6} : eval(let c = \text{Set{}} in c \rightarrow \text{asSequence()} =
--- \text{Sequence{c} -> flatten()}} .
--- \text{rewrites: 24 in Oms cpu (Oms real) (~ rewrites/second)}
--- \text{result Bool: true}
---
--- Expected Results:
--- \text{true : Boolean}
---
------ --- --- Singleton
------ --- --- --- Includes Undefined
------ --- --- --- --- Set -------------------------------
------ --- --- --- ---\text{5}
------ --- --- --- --- \text{?let c = Set{oclUndefined(Integer{})) in}
------ --- --- --- \text{c -> asSequence()} = c \rightarrow \text{asBag()} \rightarrow \text{asSequence()}
\text{red eval( let c = \text{Set{null{} in}
\text{c -> asSequence()} = c \rightarrow \text{asBag()} \rightarrow \text{asSequence()}} .
--- ================================
--- reduce in \text{BENCHMARK-TEST-B6} : eval(let c = \text{Set{null{}} in c \rightarrow \text{asSequence()} =
--- \text{asBag()} \rightarrow \text{asSequence()}} .
--- \text{rewrites: 25 in Oms cpu (Oms real) (~ rewrites/second)}
--- \text{result Bool: true}
---
--- Expected Results:
--- \text{true : Boolean}
---
------ --- ---\text{6}
------ --- --- \text{?let c = Set{oclUndefined(Integer{})) in}
------ --- --- \text{c -> asSequence() =}
------ --- --- \text{iterate(elem;}
------ --- --- \text{r:Sequence(Integer) = oclEmpty(Sequence(Integer{})) |}
------ --- --- \text{r -> including(elem))}

287
red eval(let c = Set{null} in
c -> asSequence() =
c -> iterate(elem ;
    r : Sequence(Integer) = Sequence{}
    | r -> including(elem)));
).
--- ==========================================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> asSequence() =
---     c -> iterate (elem ; r : Sequence(Integer) = Sequence{}
---     | r -> including(elem));
--- rewrites: 42 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 7
--- ?let c = Set{oclUndefined(Integer)} in
---     c -> asSequence() =
---     c -> iterate(u;
---        r : Tuple(theSet : Set(Integer), theSeq : Sequence(Integer)) =
---        Tuple{theSet = c, theSeq = Sequence{}}
---        | let e = r . theSet -> any(true) in
---        Tuple{theSet = r . theSet -> excluding(e),
---          theSeq = r . theSeq -> including(e)}).theSeq
red eval(let c = Set{null} in
c -> asSequence() =
c -> iterate(u;
    r : Tuple(theSet : Set(Integer), theSeq : Sequence(Integer)) =
    Tuple{theSet = c, theSeq = Sequence{}}
    | let e = r . theSet -> any(true) in
    Tuple{theSet = r . theSet -> excluding(e),
          theSeq = r . theSeq -> including(e)}).theSeq
).
--- ==========================================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> asSequence() =
---     c -> iterate (u ; r : Tuple(theSet : Set(Integer), theSeq : Sequence(---     Integer)) = Tuple{theSet = c, theSeq = Sequence{}}
---     | any (x | true) in Tuple{theSet = r . theSet -> excluding(e), theSeq = r . ---     theSeq -> including(e)}).theSeq).
--- rewrites: 97 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 8
--- ?let c = Set{oclUndefined(Integer)} in
---     c -> asSequence() = Sequence(c) -> flatten()
red eval(let c = Set{null} in
c -> asSequence() = Sequence(c) -> flatten())
).
--- ==========================================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> asSequence() =
---     Sequence(c) -> flatten()).
--- rewrites: 29 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- Excludes Undefined

--- --- --- --- Set ---------------------------------------------
--- 9
--- ?let c = Set{1} in c -> asSequence() = c -> asBag() -> asSequence()
red eval( let c = Set{1} in c -> asSequence() = c -> asBag() ->
    asSequence() ).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c ->
--- asBag() -> asSequence() ).
--- rewrites: 25 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 10
--- ?let c = Set{1} in
--- c -> asSequence() =
--- c -> iterate(elem;
--- --- r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- --- r -> including(elem))
red eval( let c = Set{1} in
    c -> asSequence() =
    c -> iterate(elem ;
        r : Sequence(Integer) = Sequence{} | r -> including(elem))
). 
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> asSequence() = c ->
--- iterate (elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem)))
---
--- rewrites: 42 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 11
--- ?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
red eval( let c = Set{1} in
    c -> asSequence() =
    c -> iterate(u;
        r : Tuple(theSet : Set(Integer), theSeq : Sequence(Integer)) =
            Tuple{theSet = c, theSeq = Sequence{}} |
        let e = r . theSet -> any(x | true) in
            Tuple{theSet = r . theSet -> excluding(e),
                theSeq = r . theSeq -> including(e)} . theSeq
    )
). reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> asSequence() = c ->
--- iterate (u ; r : Tuple(theSet : Set(Integer), theSeq : Sequence(Integer)) =
--- Tuple{theSet = c, theSeq = Sequence{}} | let e = r . theSet -> any (x |
--- true) in Tuple{theSet = r . theSet -> excluding(e), theSeq = r . theSeq ->
--- including(e)) . theSeq). 
--- rewrites: 97 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 12
--- ?let c = Set{1} in c -> asSequence() = Sequence{c} -> flatten()
red eval( let c = Set{1} in c -> asSequence() = Sequence{c} ->
flatten() ).
--- rewrites in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> asSequence() =
--- Sequence(c) -> flatten() ).
--- rewrites: 29 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- ---- ---- Many Elements
---
--- ---- ---- ---- Includes Undefined
---
--- 13
--- ?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in
--- c -> asSequence() = c -> asBag() -> asSequence()
red eval( let c = Set{null, 1, -1, 3, 2, 0} in
--- c -> asSequence() = c -> asBag() -> asSequence() )
---
--- rewrites in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 0, 2} in c
--- -> asSequence() = c -> asBag() -> asSequence() ).
--- rewrites: 55 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 14
--- ?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))
red eval( let c = Set{null, 1, -1, 3, 2, 0} in
--- c -> asSequence() =
--- c -> iterate(elem ;
--- r : Sequence(Integer) = Sequence{} |
--- r -> including(elem))
---
--- rewrites in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 2, 0} in c ->
--- asSequence() = c -> iterate (elem ; r : Sequence(Integer) = Sequence{} | r
--- -> including(elem)) ).
---
--- 290
--- rewrites: 127 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 15
--- ?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(r.theSet -> excluding(e),
--- theSeq = r.theSeq -> including(e))).theSeq
red eval(let c = Set{null, 1, -1, 3, 2, 0} in
  c -> asSequence() =
  c -> iterate(u;
  r : Tuple(theSet : Set(Integer), theSeq : Sequence(Integer)) =
  Tuple(theSet = c, theSeq = Sequence{}) |
  let e = r.theSet -> any(x | true) in
  Tuple(r.theSet = r.theSet -> excluding(e),
  theSeq = r.theSeq -> including(e))).theSeq
)
--- ==========================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 2, 0} in c ->
--- iterate (u ; r : Tuple(theSet : Set(Integer), theSeq : Sequence(Integer)) =
--- Tuple(theSet = c, theSeq = Sequence{}) | let e = r.theSet -> any (x |
--- true) in Tuple(r.theSet = r.theSet -> excluding(e),
--- theSeq = r.theSeq -> including(e))) . theSeq)
--- rewrites: 380 in 4ms cpu (1ms real) (95000 rewrites/second)
--- result Sequence: Sequence{null, 1, -1, 3, 2, 0}
---
--- Expected Results:
--- true : Boolean
---
--- 16
--- ?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in
--- c -> asSequence() = Sequence(c) -> flatten()
red eval(let c = Set{null, 1, -1, 3, 2, 0} in
  c -> asSequence() = Sequence(c) -> flatten()
)
--- ==========================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 0, 2} in c
--- -> asSequence() = Sequence(c) -> flatten()) .
--- rewrites: 59 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- Excludes Undefined
---
--- --- --- --- --- Set ---------------------------------------------
--- 17
--- ?let c = Set{1, -1, 3, 2, 0} in
--- c -> asSequence() = c -> asBag() -> asSequence()
red eval(let c = Set{1, -1, 3, 2, 0} in
  c -> asSequence() = c -> asBag() -> asSequence()
reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1, -1, 3, 0, 2} in c ->
  asSequence() = c -> iterate(elem ;
  r : Sequence(Integer) = oclEmpty(Sequence(Integer)) |
  r -> including(elem))) .

--- Expected Results:
--- true : Boolean
---
--- 18
--- ?let c = Set{1, -1, 3, 2, 0} in
--- c -> asSequence() =
--- c -> iterate(elem ;
--- r : Sequence(Integer) = Sequence{} |
--- r -> including(elem))
red eval( let c = Set{1, -1, 3, 2, 0} in
  c -> asSequence() =
  c -> iterate(elem ;
  r : Sequence(Integer) = Sequence{} |
  r -> including(elem))
)

--- Expected Results:
--- true : Boolean
---
--- 19
--- ?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
red eval( 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 : Sequence{}} |
  let e = r . theSet -> any(x | true) in
  Tuple{theSet : r . theSet -> excluding(e),
  theSeq : r . theSeq -> including(e))}.theSeq)

--- Expected Results:
--- true : Boolean
--- Expected Results:
--- true : Boolean

--- 20
--- ?let c = Set{1, -1, 3, 2, 0} in
---  c -> asSequence() = Sequence(c) -> flatten()
red eval( let c = Set{1, -1, 3, 2, 0} in
  c -> asSequence() = Sequence(c) -> flatten() )
--- ==================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1, -1, 3, 0, 2} in c ->
---  asSequence() = Sequence(c) -> flatten()) .
--- rewrites: 53 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- Technical Relationships

--- --- Determinateness Constraints

--- --- --- In Addition to the Ocl4All 2007 equivalences

--- --- --- --- Empty

--- --- --- --- Set

--- 1
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> any(true) = c -> asBag() -> any(true)
red eval( let c = Set{} in
   c -> any(true) = c -> asBag() -> any(true)
) .
--- =========================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in c -> any(true) = c ->
--- asBag() -> any(true)) .
--- rewrites: 28 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- 2
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Set{} in
   c -> any(true) = c -> asSequence() -> any(true)
) .
--- =========================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in c -> any(true) = c ->
--- asSequence() -> any(true)) .
--- rewrites: 29 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

--- 3
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> any(true) = c -> asBag() -> asSequence() -> any(true)
red eval( let c = Set{} in
   c -> any(true) = c -> asBag() -> asSequence() -> any(true)
) .
--- =========================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in c -> any(true) = c ->
--- asBag() -> asSequence() -> any(true)) .
--- rewrites: 31 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean

---
--- 4
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> any(true) = c -> asSequence() -> first()
red eval( let c = Set{} in
     c -> any(true) = c -> asSequence() -> first()
).
--- =============================================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in c -> any(true) = c ->
---     asSequence() -> first()) .
--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result OclInvalid: invalid
---
--- Expected Results:
--- true : Boolean
--- WARNING. We think that this test should return invalid instead of true.
--- According to the OCL Specification, any expression containing invalid
--- becomes invalid.
--- invalid = invalid should return invalid!
---
--- 5
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> any(true) = c -> asSequence() -> last()
red eval( let c = Set{} in
     c -> any(true) = c -> asSequence() -> last()
).
--- =============================================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in c -> any(true) = c ->
---     asSequence() -> first()) .
--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result OclInvalid: invalid
---
--- Expected Results:
--- true : Boolean
---
--- 6
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> any(true) = c -> asBag() -> asSequence() -> first()
red eval( let c = Set{} in
     c -> any(true) = c -> asBag() -> asSequence() -> first()
).
--- =============================================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in c -> any(true) = c ->
---     asBag() -> asSequence() -> first()) .
--- rewrites: 24 in Oms cpu (Oms real) (~ rewrites/second)
--- result OclInvalid: invalid
---
--- Expected Results:
--- true : Boolean
---
--- 7
--- ?let c = oclEmpty(Set(Integer)) in
--- c -> any(true) = c -> asBag() -> asSequence() -> last()
red eval( let c = Set{} in
     c -> any(true) = c -> asBag() -> asSequence() -> last()
).
--- =============================================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in c -> any(true) = c ->
---     asBag() -> asSequence() -> last()) .
--- rewrites: 24 in Oms cpu (Oms real) (~ rewrites/second)
--- Expected Results:
--- true : Boolean
--- 8
--- ?let c = oclEmpty(Set(Integer)) in
---  c -> any(true) =
---      c -> iterate(elem;
---                   r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
---                   r -> including(elem)) -> first()
red eval( let c = Set{} in
         c -> any(true) =
         c -> iterate(elem ;
                   r : Sequence(Integer) = Sequence{} |
                   r -> including(elem)) -> first()
).
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in c -> any(true) = c ->
--- iterate (elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem))
--- -> first()) .
--- rewrites: 30 in 0ms cpu (0ms real) (~ rewrites/second)
--- result OclInvalid: invalid
---
--- Expected Results:
--- true : Boolean
--- 9
--- ?let c = oclEmpty(Set(Integer)) in
---  Sequence{c} -> flatten() = c -> asSequence()
red eval( let c = Set{} in
         Sequence{c} -> flatten() = c -> asSequence()
).
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in Sequence{c} -> flatten() =
---  c -> asSequence()) .
--- rewrites: 24 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 10
--- ?let c = oclEmpty(Set(Integer)) in
---  Sequence{c} -> flatten() = c -> asBag() -> asSequence()
red eval( let c = Set{} in
         Sequence{c} -> flatten() = c -> asBag() -> asSequence()
).
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{} in Sequence{c} -> flatten() =
---  c -> asBag() -> asSequence()) .
--- rewrites: 26 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- Bag
--- 11
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> asSequence() =
--- c -> iterate(elem;
--- r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r -> including(elem))
red eval( let c = Bag{} in
  c -> asSequence() =
  c -> iterate(elem ;
    r : Sequence(Integer) = Sequence{} | r -> including(elem)) )
  ).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{} in c -> asSequence() = c ->
--- iterate (elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem)))
--- .
--- rewrites: 26 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 12
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Bag{} in
  c -> any(true) = c -> asSequence() -> any(true) )
  ).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{} in c -> any(true) = c ->
--- asSequence() -> any(true))
--- .
--- rewrites: 29 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 13
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> any(true) = c -> asSequence() -> first()
red eval( let c = Bag{} in
  c -> any(true) = c -> asSequence() -> first() )
  ).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{} in c -> any(true) = c ->
--- asSequence() -> first())
--- .
--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result OclInvalid: invalid
---
--- Expected Results:
--- true : Boolean
--- 14
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> any(true) = c -> asSequence() -> last()
red eval( let c = Bag{} in
  c -> any(true) = c -> asSequence() -> last() )
  ).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{} in c -> any(true) = c ->
--- asSequence() -> last() ).
--- rewrites: 22 in Oms cpu (Oms real) (~ rewrites/second)
--- result OclInvalid: invalid
---
--- Expected Results:
--- true : Boolean
---
--- 15
--- ?let c = oclEmpty(Bag(Integer)) in
--- c -> any(true) =
--- c -> iterate(elem;
--- r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r -> including(elem)) -> first()
red eval( let c = Bag{} in
c -> any(true) =
c -> iterate(elem ;
    r : Sequence(Integer) = Sequence{} | r -> including(elem)) -> first()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{} in c -> any(true) = c ->
--- iterate (elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem))
--- -> first()) .
--- rewrites: 30 in Oms cpu (Oms real) (~ rewrites/second)
--- result OclInvalid: invalid
---
--- Expected Results:
--- true : Boolean
---
--- 16
--- ?let c = oclEmpty(Bag(Integer)) in
--- Sequence{c} -> flatten() = c -> asSequence()
red eval( let c = Bag{} in
    Sequence{c} -> flatten() = c -> asSequence()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{} in Sequence{c} -> flatten() =
--- c -> asSequence()) .
--- rewrites: 24 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- Sequence
--- --- --- --- --- Singleton
--- --- --- --- --- Includes Undefined
--- --- --- --- --- --- Set
---
--- 17
--- ?let c = Set{oclUndefined(Integer)} in
--- c -> any(true) = c -> asBag() -> any(true)
---
red eval( let c = Set{null} in
c -> any(true) = c -> asBag() -> any(true)
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> asBag() -> any(true)) .
--- rewrites: 24 in 0ms cpu (0ms real) (~ rewrites/second)
--- result OclVoid: null
---
--- Expected Results:
---  true : Boolean
---
--- 18
--- ?let c = Set{oclUndefined(Integer)} in
---  c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Set{null} in
         c -> any(true) = c -> asSequence() -> any(true)
   ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> any(true) = c ->
--- asSequence() -> any(true)) .
--- rewrites: 39 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- 19
--- ?let c = Set{oclUndefined(Integer)} in
---  c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Set{null} in
         c -> any(true) = c -> asSequence() -> any(true)
   ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> any(true) = c ->
--- asSequence() -> any(true)) .
--- rewrites: 41 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- 20
--- ?let c = Set{oclUndefined(Integer)} in
---  c -> any(true) = c -> asSequence() -> first()
red eval( let c = Set{null} in
         c -> any(true) = c -> asSequence() -> first()
   ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> any(true) = c ->
--- asSequence() -> first()) .
--- rewrites: 34 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- 21
--- ?let c = Set{oclUndefined(Integer)} in
---  c -> any(true) = c -> asSequence() -> last()
red eval( let c = Set{null} in
         c -> any(true) = c -> asSequence() -> last()
   ) .
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> any(true) = c ->
--- asSequence() -> last()).
--- rewrites: 32 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
---
--- 22
--- ?let c = Set{oclUndefined(Integer)} in
---  c -> any(true) = c -> asBag() -> asSequence() -> first()
red eval( let c = Set{null} in
       c -> any(true) = c -> asBag() -> asSequence() -> first() ) .
---
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> any(true) = c ->
--- asBag() -> asSequence() -> first()).
--- rewrites: 36 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
--- 23
--- ?let c = Set{oclUndefined(Integer)} in
---  c -> any(true) = c -> asBag() -> asSequence() -> last()
red eval( let c = Set{null} in
       c -> any(true) = c -> asBag() -> asSequence() -> last() ) .
---
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> any(true) = c ->
--- asBag() -> asSequence() -> last()).
--- rewrites: 34 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean
--- 24
--- ?let c = Set{oclUndefined(Integer)} in
---  c -> any(true) =
---  c -> iterate(elem; r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
---      r -> including(elem)) -> first()
red eval( let c = Set{null} in
       c -> any(true) =
       c -> iterate(elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem))
       -> first() ) .
---
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in c -> any(true) = c ->
--- iterate (elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem))
--- -> first()).
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 25
--- ?let c = Set{oclUndefined(Integer)} in
--- Sequence{c} -> flatten() = c -> asSequence()
red eval( let c = Set{null} in
    Sequence{c} -> flatten() = c -> asSequence()
)
--- =========================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in Sequence{c} -> flatten(
--- ) = c -> asSequence()) .
--- rewrites: 28 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 26
--- ?let c = Set{oclUndefined(Integer)} in
--- Sequence{c} -> flatten() = c -> asBag() -> asSequence()
red eval( let c = Set{null} in
    Sequence{c} -> flatten() = c -> asBag() -> asSequence()
)
--- ==================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null} in Sequence{c} -> flatten(
--- ) = c -> asBag() -> asSequence()) .
--- rewrites: 30 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- --- --- --- --- --- Bag
--- 27
--- ?let c = Bag{oclUndefined(Integer)} in
--- c -> asSequence() =
--- c -> iterate(elem;
--- r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r -> including(elem))
red eval( let c = Bag{null} in
    c -> asSequence() =
    c -> iterate(elem ;
    r : Sequence(Integer) = Sequence{} | r -> including(elem))
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null} in c -> asSequence() = c
--- -> iterate (elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem))) .
--- rewrites: 42 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 28
--- ?let c = Bag{oclUndefined(Integer)} in
--- c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Bag{null} in

301
c -> any(true) = c -> asSequence() -> any(true)

--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null} in c -> any(true) = c ->
--- asSequence() -> any(true)) .
--- rewrites: 39 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- 29
--- ?let c = Bag{oclUndefined(Integer)} in
---   c -> any(true) = c -> asSequence() -> first()
red eval( let c = Bag{null} in
   c -> any(true) = c -> asSequence() -> first()
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null} in c -> any(true) = c ->
--- asSequence() -> first()) .
--- rewrites: 34 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- 30
--- ?let c = Bag{oclUndefined(Integer)} in
---   c -> any(true) = c -> asSequence() -> last()
red eval( let c = Bag{null} in
   c -> any(true) = c -> asSequence() -> last()
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null} in c -> any(true) = c ->
--- asSequence() -> last()) .
--- rewrites: 32 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- 31
--- ?let c = Bag{oclUndefined(Integer)} in
---   c -> any(true) =
---     c -> iterate(elem;
---       r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
---       r -> including(elem)) -> first()
red eval( let c = Bag{null} in
   c -> any(true) =
     c -> iterate(elem;
       r : Sequence(Integer) = Sequence{} | r -> including(elem)) -> first()
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null} in c -> any(true) = c ->
--- iterate (elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem))
--- -> first()) .
--- rewrites: 54 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true

302
--- Expected Results:
--- true : Boolean

--- 32
--- ?let c = Bag{oclUndefined(Integer)} in
--- Sequence{c} -> flatten() = c -> asSequence()
red eval( let c = Bag{null} in
          Sequence{c} -> flatten() = c -> asSequence()
    ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null} in Sequence{c} -> flatten(
--- ) = c -> asSequence()) .
--- rewrites: 28 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- --- Sequence
--- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- Set -----------------------------------------------
--- 33
--- ?let c = Set{1} in
--- c -> any(true) = c -> asBag() -> any(true)
red eval( let c = Set{1} in
         c -> any(true) = c -> asBag() -> any(true)
    ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> any(true) = c ->
--- asBag() -> any(true)) .
--- rewrites: 38 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 34
--- ?let c = Set{1} in
--- c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Set{1} in
         c -> any(true) = c -> asSequence() -> any(true)
    ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> any(true) = c ->
--- asSequence() -> any(true)) .
--- rewrites: 39 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 35
--- ?let c = Set{1} in
--- c -> any(true) = c -> asBag() -> asSequence() -> any(true)
red eval( let c = Set{1} in
         c -> any(true) = c -> asBag() -> asSequence() -> any(true)
    ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> any(true) = c ->
--- asBag() -> asSequence() -> any(true)) .
--- rewrites: 41 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 36
--- ?let c = Set{1} in
--- c -> any(true) = c -> asSequence() -> first()
red eval( let c = Set{1} in
  c -> any(true) = c -> asSequence() -> first()
) .
---
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> any(true) = c ->
--- asSequence() -> first()) .
--- rewrites: 34 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 37
--- ?let c = Set{1} in
--- c -> any(true) = c -> asSequence() -> last()
red eval( let c = Set{1} in
  c -> any(true) = c -> asSequence() -> last()
) .
---
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> any(true) = c ->
--- asSequence() -> last()) .
--- rewrites: 32 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 38
--- ?let c = Set{1} in
--- c -> any(true) = c -> asBag() -> asSequence() -> first()
red eval( let c = Set{1} in
  c -> any(true) = c -> asBag() -> asSequence() -> first()
) .
---
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> any(true) = c ->
--- asBag() -> asSequence() -> first()) .
--- rewrites: 36 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 39
--- ?let c = Set{1} in
--- c -> any(true) = c -> asBag() -> asSequence() -> last()
red eval( let c = Set{1} in
---
c -> any(true) = c -> asBag() -> asSequence() -> last()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> any(true) = c ->
--- asBag() -> asSequence() -> last()) .
--- rewrites: 34 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 40
--- ?let c = Set{1} in
--- c -> any(true) =
--- c -> iterate(elem;
--- r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r -> including(elem)) -> first()
red eval( let c = Set{1} in
c -> any(true) =
c -> iterate(elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem))
-> first() ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in c -> any(true) = c ->
--- iterate (elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem))
--- -> first()) .
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 41
--- ?let c = Set{1} in
--- Sequence(c) -> flatten() = c -> asSequence()
red eval( let c = Set{1} in
Sequence(c) -> flatten() = c -> asSequence() ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in Sequence(c) -> flatten() =
--- c -> asSequence()) .
--- rewrites: 28 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 42
--- ?let c = Set{1} in
--- Sequence(c) -> flatten() = c -> asBag() -> asSequence()
red eval( let c = Set{1} in
Sequence(c) -> flatten() = c -> asBag() -> asSequence() ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1} in Sequence(c) -> flatten() =
--- c -> asBag() -> asSequence()) .
--- rewrites: 30 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- 305
---
--- Expected Results:
--- true : Boolean

--- --- --- --- --- Bag
--- 43
--- ?let c = Bag{1} in
---  c -> asSequence() =
---  c -> iterate(elem;
---    r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
---    r -> including(elem))
red eval( let c = Bag{1} in
  c -> asSequence() =
  c -> iterate(elem;
    r : Sequence(Integer) = Sequence{} |
    r -> including(elem))
).
--- rewrites: 42 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 44
--- ?let c = Bag{1} in
---  c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Bag{1} in
  c -> any(true) = c -> asSequence() -> any(true))
).
--- rewrites: 39 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 45
--- ?let c = Bag{1} in
---  c -> any(true) = c -> asSequence() -> first()
red eval( let c = Bag{1} in
  c -> any(true) = c -> asSequence() -> first())
).
--- rewrites: 34 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 46
--- ?let c = Bag{1} in
--- c -> any(true) = c -> asSequence() -> last()
red eval( let c = Bag{1} in
c -> any(true) = c -> asSequence() -> last()
) .
--- ______________________________________________________________
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1} in c -> any(true) = c ->
--- asSequence() -> last()) .
--- rewrites: 32 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 47
--- ?let c = Bag{1} in
--- c -> any(true) =
--- c -> iterate(elem;
--- r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r -> including(elem)) -> first()
red eval( let c = Bag{1} in
c -> any(true) =
c -> iterate(elem ;
   r : Sequence(Integer) = Sequence{} | r -> including(elem)) -> first()
) .
--- ______________________________________________________________
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1} in c -> any(true) = c ->
--- iterate (elem ; r : Sequence(Integer) = Sequence{} | r -> including(elem))
--- -> first()) .
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 48
--- ?let c = Bag{1} in
--- Sequence(c) -> flatten() = c -> asSequence()
red eval( let c = Bag{1} in
   Sequence(c) -> flatten() = c -> asSequence()
) .
--- ______________________________________________________________
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1} in Sequence(c) -> flatten() =
--- c -> asSequence()) .
--- rewrites: 28 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- Sequence
--- --- --- --- --- Many Elements
--- --- --- --- --- Includes Undefined
--- --- --- --- --- Set -------------------------------
--- 49
--- ?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in
--- c -> any(true) = c -> asBag() -> any(true)
red eval( let c = Set{null, 1, -1, 3, 2, 0} in
       c -> any(true) = c -> asBag() -> any(true)
   ) .
--- ==========================================  
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 2, 0} in c ->
--- any(true) = c -> asBag() -> any(true)) .
--- rewrites: 63 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    false : Boolean
---
--- 50
--- ?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in
--- c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Set{null, 1, -1, 3, 2, 0} in
       c -> any(true) = c -> asSequence() -> any(true)
   ) .
--- ==========================================  
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 2, 0} in c ->
--- any(true) = c -> asSequence() -> any(true)) .
--- rewrites: 64 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- 51
--- ?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in
--- c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Set{null, 1, -1, 3, 2, 0} in
       c -> any(true) = c -> asSequence() -> any(true)
   ) .
--- ==========================================  
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 2, 0} in c ->
--- any(true) = c -> asSequence() -> any(true)) .
--- rewrites: 66 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    false : Boolean
---
--- 52
--- ?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in
--- c -> any(true) = c -> asSequence() -> first()
red eval( let c = Set{null, 1, -1, 3, 2, 0} in
       c -> any(true) = c -> asSequence() -> first()
   ) .
--- ==========================================  
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 2, 0} in c ->
--- any(true) = c -> asSequence() -> first()) .
--- rewrites: 59 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- 53
--- ?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in
--- c -> any(true) = c -> asSequence() -> first()
red eval( let c = Set{null, 1, -1, 3, 2, 0} in
       c -> any(true) = c -> asSequence() -> first()
   ) .
--- ==========================================  
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 2, 0} in c ->
--- any(true) = c -> asSequence() -> first()) .
--- rewrites: 60 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    false : Boolean
\[
\text{let } c = \text{Set\{oclUndefined(Integer), 1, -1, 3, 2, 0\} in}
\text{c \rightarrow any(true) = c \rightarrow asSequence() \rightarrow last()}
\]

\[
\text{reduce in BENCHMARK-TEST-B6 : eval(let c = Set\{null, 1, -1, 3, 2, 0\} in c \rightarrow}
\text{any(true) = c \rightarrow asSequence() \rightarrow last()) .}
\]

\[
\text{rewrites: 57 in 4ms cpu (1ms real) (14250 rewrites/second)}
\]

\[
\text{result Bool: false}
\]

---

\[
\text{Expected Results:}
\]

\[
\text{false : Boolean}
\]

---

\[
\text{let } c = \text{Set\{oclUndefined(Integer), 1, -1, 3, 2, 0\} in}
\text{c \rightarrow any(true) = c \rightarrow asBag() \rightarrow asSequence() \rightarrow first()}
\]

\[
\text{reduce in BENCHMARK-TEST-B6 : eval(let c = Set\{null, 1, -1, 3, 2, 0\} in c \rightarrow}
\text{any(true) = c \rightarrow asBag() \rightarrow asSequence() \rightarrow first()) .}
\]

\[
\text{rewrites: 61 in 0ms cpu (0ms real) (~ rewrites/second)}
\]

\[
\text{result Bool: true}
\]

---

\[
\text{Expected Results:}
\]

\[
\text{false : Boolean}
\]

---

\[
\text{let } c = \text{Set\{oclUndefined(Integer), 1, -1, 3, 2, 0\} in}
\text{c \rightarrow any(true) = c \rightarrow asBag() \rightarrow asSequence() \rightarrow last()}
\]

\[
\text{reduce in BENCHMARK-TEST-B6 : eval(let c = Set\{null, 1, -1, 3, 2, 0\} in c \rightarrow}
\text{any(true) = c \rightarrow asBag() \rightarrow asSequence() \rightarrow last()) .}
\]

\[
\text{rewrites: 59 in 0ms cpu (0ms real) (~ rewrites/second)}
\]

\[
\text{result Bool: false}
\]

---

\[
\text{Expected Results:}
\]

\[
\text{false : Boolean}
\]

---

\[
\text{let } c = \text{Set\{oclUndefined(Integer), 1, -1, 3, 2, 0\} in}
\text{c \rightarrow any(true) = c \rightarrow iterate(elem;}
\text{r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |}
\text{r \rightarrow including(elem)) \rightarrow first())}
\]

\[
\text{reduce in BENCHMARK-TEST-B6 : eval(let c = Set\{null, 1, -1, 3, 2, 0\} in c \rightarrow}
\text{any(true) = c \rightarrow iterate(elem ;}
\text{r : Sequence(Integer) = Sequence{} |}
\text{r \rightarrow including(elem)) \rightarrow first()) .}
\]

---

\[
\text{Expected Results:}
\]

\[
\text{true : Boolean}
\]

---

\[
\text{let } c = \text{Set\{oclUndefined(Integer), 1, -1, 3, 2, 0\} in}
\text{c \rightarrow any(true) = c \rightarrow iterate(elem;}
\text{r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |}
\text{r \rightarrow including(elem)) \rightarrow first())}
\]

\[
\text{reduce in BENCHMARK-TEST-B6 : eval(let c = Set\{null, 1, -1, 3, 2, 0\} in c \rightarrow}
\text{any(true) = c \rightarrow iterate(elem ;}
\text{r : Sequence(Integer) = Sequence{} |}
\text{r \rightarrow including(elem)) \rightarrow first()) .}
\]
--- any(true) = c -> iterate (elem ; r : Sequence(Integer) = Sequence{} | r ->
--- including(elem)) -> first()) .
--- rewrites: 139 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
--- 57
--- ?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in
---   Sequence(c) -> flatten() = c -> asSequence()
red eval( let c = Set{null, 1, -1, 3, 2, 0} in
---   Sequence(c) -> flatten() = c -> asSequence()
)
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 2, 0} in
---   Sequence(c) -> flatten() = c -> asSequence()) .
--- rewrites: 53 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
--- 58
--- ?let c = Set{oclUndefined(Integer), 1, -1, 3, 2, 0} in
---   Sequence(c) -> flatten() = c -> asBag() -> asSequence()
red eval( let c = Set{null, 1, -1, 3, 2, 0} in
---   Sequence(c) -> flatten() = c -> asBag() -> asSequence()
)
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{null, 1, -1, 3, 2, 0} in
---   Sequence(c) -> flatten() = c -> asBag() -> asSequence()) .
--- rewrites: 55 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   false : Boolean
--- --- --- --- --- Bag
--- --- --- --- --- --- Equal Values
--- 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)
---   )
red eval( let c = Bag{null, 1, -1, 3, 2, 0,
---   null} in
---   c -> asSequence() =
---   c -> iterate(elem ;
---     r : Sequence(Integer) = Sequence{} |
---     r -> including(elem)
--- )
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0, null} in c
---   -> asSequence() = c -> iterate (elem ; r : Sequence(Integer) = Sequence{} |
--- r -> including(elem)) .
--- rewrites: 144 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean

--- 60
--- ?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0,
---         oclUndefined(Integer)} in
---     c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Bag{null, 1, -1, 3, 2, 0,
---                 null} in
---     c -> any(true) = c -> asSequence() -> any(true)
--- ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0, null} in c
---     -> any(true) = c -> asSequence() -> any(true)) .
--- rewrites: 69 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean

--- 61
--- ?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0,
---         oclUndefined(Integer)} in
---     c -> any(true) = c -> asSequence() -> first()
red eval( let c = Bag{null, 1, -1, 3, 2, 0,
---                 null} in
---     c -> any(true) = c -> asSequence() -> first()
--- ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0, null} in c
---     -> any(true) = c -> asSequence() -> first()) .
--- rewrites: 64 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---  true : Boolean

--- 62
--- ?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0,
---         oclUndefined(Integer)} in
---     c -> any(true) = c -> asSequence() -> last()
red eval( let c = Bag{null, 1, -1, 3, 2, 0,
---                 null} in
---     c -> any(true) = c -> asSequence() -> last()
--- ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0, null} in c
---     -> any(true) = c -> asSequence() -> last()) .
--- rewrites: 62 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---     false : Boolean

--- 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()
red eval( let c = Bag{null, 1, -1, 3, 2, 0, null} in
c → any(true) =
c → iterate(elem ;
    r : Sequence(Integer) = Sequence{} |
    r → including(elem)) → first()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0, null} in c
--- → any(true) = c → iterate (elem ; r : Sequence(Integer) = Sequence{} | r
--- → including(elem)) → first() ) .
--- rewrites: 156 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 64
--- ?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0, oclUndefined(Integer)} in
--- Sequence(c) → flatten() = c → asSequence()
red eval( let c = Bag{null, 1, -1, 3, 2, 0, null} in
Sequence(c) → flatten() = c → asSequence()
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0, null} in
--- Sequence(c) → flatten() = c → asSequence()) .
--- rewrites: 58 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 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))
red eval( let c = Bag{null, 1, -1, 3, 2, 0} in
c → asSequence() =
c → iterate(elem ;
    r : Sequence(Integer) = Sequence{} |
    r → including(elem))
).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0} in c →
--- asSequence() = c → iterate (elem ; r : Sequence(Integer) = Sequence{} | r
--- → including(elem)) ) .
--- rewrites: 127 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- 312
--- Expected Results:  
--- true : Boolean  

--- 66  
--- `let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0} in  
--- c -> any(true) = c -> asSequence() -> any(true)  
red eval(let c = Bag{null, 1, -1, 3, 2, 0} in  
  c -> any(true) = c -> asSequence() -> any(true)  
) .  
--- ==============================================================  
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0} in c ->  
--- any(true) = c -> asSequence() -> any(true)) .  
--- rewrites: 64 in 0ms cpu (0ms real) (~ rewrites/second)  
--- result Bool: true  
--- 
--- Expected Results:  
--- true : Boolean  

--- 67  
--- `let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0} in  
--- c -> any(true) = c -> asSequence() -> first()  
red eval(let c = Bag{null, 1, -1, 3, 2, 0} in  
  c -> any(true) = c -> asSequence() -> first()  
) .  
--- ==============================================================  
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0} in c ->  
--- any(true) = c -> asSequence() -> first()) .  
--- rewrites: 59 in 0ms cpu (0ms real) (~ rewrites/second)  
--- result Bool: true  
--- 
--- Expected Results:  
--- true : Boolean  

--- 68  
--- `let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0} in  
--- c -> any(true) = c -> asSequence() -> last()  
red eval(let c = Bag{null, 1, -1, 3, 2, 0} in  
  c -> any(true) = c -> asSequence() -> last()  
) .  
--- ==============================================================  
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0} in c ->  
--- any(true) = c -> asSequence() -> last()) .  
--- rewrites: 57 in 0ms cpu (0ms real) (~ rewrites/second)  
--- result Bool: false  
--- 
--- Expected Results:  
--- false : Boolean  

--- 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()  
red eval(let c = Bag{null, 1, -1, 3, 2, 0} in  
  c -> any(true) =  
  c -> iterate(elem ;  
    r : Sequence(Integer) = Sequence{} |  
  ) .  
--- ==============================================================  
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0} in c ->  
--- any(true) = c -> asSequence() -> last()) .  
--- rewrites: 57 in 0ms cpu (0ms real) (~ rewrites/second)  
--- result Bool: false  
--- 
--- Expected Results:  
--- false : Boolean
r -> including(elem)) -> first()

--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0} in c ->
---     any(true) = c -> iterate (elem ; r : Sequence(Integer) = Sequence{} | r ->
---     including(elem)) -> first()) .
--- rewrites: 139 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- 70
--- ?let c = Bag{oclUndefined(Integer), 1, -1, 3, 2, 0} in
---     Sequence{c} -> flatten() = c -> asSequence()
red eval( let c = Bag{null, 1, -1, 3, 2, 0} in
     Sequence{c} -> flatten() = c -> asSequence() ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{null, 1, -1, 3, 2, 0} in
---     Sequence{c} -> flatten() = c -> asSequence()) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   true : Boolean
---
--- --- --- --- --- --- --- Sequence
--- --- --- --- --- --- --- Equal Values
--- --- --- --- --- --- --- No Equal Values
--- --- --- --- --- --- --- Excludes Undefined
--- --- --- --- --- --- --- Set
--- --- --- --- --- --- --- -----------------------------------------------
--- 71
--- ?let c = Set{1, -1, 3, 2, 0} in
---     c -> any(true) = c -> asBag() -> any(true)
red eval( let c = Set{1, -1, 3, 2, 0} in
     c -> any(true) = c -> asBag() -> any(true) ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1, -1, 3, 2, 0} in c -> any(
---     true) = c -> asBag() -> any(true)) .
--- rewrites: 58 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---   false : Boolean
---
--- 72
--- ?let c = Set{1, -1, 3, 2, 0} in
---     c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Set{1, -1, 3, 2, 0} in
     c -> any(true) = c -> asSequence() -> any(true) ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1, -1, 3, 2, 0} in c -> any(
   true) = c -> asSequence() -> any(true)) .
--- rewrites: 59 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 73
--- ?let c = Set{1, -1, 3, 2, 0} in
---   c -> any(true) = c -> asBag() -> asSequence() -> any(true)
red eval(let c = Set{1, -1, 3, 2, 0} in
   c -> any(true) = c -> asBag() -> asSequence() -> any(true)
).
--- ========================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1, -1, 3, 2, 0} in c -> any(
---   true) = c -> asSequence() -> any(true)) .
--- rewrites: 61 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- false : Boolean

--- 74
--- ?let c = Set{1, -1, 3, 2, 0} in
---   c -> any(true) = c -> asSequence() -> first()
red eval(let c = Set{1, -1, 3, 2, 0} in
   c -> any(true) = c -> asSequence() -> first()
).
--- ========================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1, -1, 3, 2, 0} in c -> any(
---   true) = c -> asSequence() -> first()) .
--- rewrites: 54 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 75
--- ?let c = Set{1, -1, 3, 2, 0} in
---   c -> any(true) = c -> asSequence() -> last()
red eval(let c = Set{1, -1, 3, 2, 0} in
   c -> any(true) = c -> asSequence() -> last()
).
--- ========================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1, -1, 3, 2, 0} in c -> any(
---   true) = c -> asSequence() -> last()) .
--- rewrites: 52 in 0ms cpu (Oms real) (~ rewrites/second)
--- result Bool: false
---
--- Expected Results:
--- false : Boolean

--- 76
--- ?let c = Set{1, -1, 3, 2, 0} in
---   c -> any(true) = c -> asBag() -> asSequence() -> first()
red eval(let c = Set{1, -1, 3, 2, 0} in
   c -> any(true) = c -> asBag() -> asSequence() -> first()
).
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set{1, -1, 3, 2, 0} in c -> any(
    true) = c -> asBag() -> asSequence() -> first()) .
--- rewrites: 56 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 77
--- ?let c = Set{1, -1, 3, 2, 0} in
--- c -> any(true) = c -> asBag() -> asSequence() -> last()
red eval( let c = Set{1, -1, 3, 2, 0} in
    c -> any(true) = c -> asBag() -> asSequence() -> last()
)
--- rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: false
---
--- Expected Results:
--- false : Boolean
---
--- 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()
red eval( let c = Set{1, -1, 3, 2, 0} in
    c -> any(true) =
    c -> iterate(elem ;
    r = Sequence{} |
    r -> including(elem)) -> first()
)
--- rewrites: 121 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 79
--- ?let c = Set{1, -1, 3, 2, 0} in
--- Sequence{c} -> flatten() = c -> asSequence()
red eval( let c = Set{1, -1, 3, 2, 0} in
    Sequence{c} -> flatten() = c -> asSequence()
)
--- rewrites: 48 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 80
--- ?let c = Set\{1, -1, 3, 2, 0\} in
--- Sequence(c) -> flatten() = c -> asBag() -> asSequence()
red eval( let c = Set\{1, -1, 3, 2, 0\} in
  Sequence(c) -> flatten() = c -> asBag() -> asSequence()
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Set\{1, -1, 3, 2, 0\} in Sequence(c)
--- -> flatten() = c -> asBag() -> asSequence()).
--- rewrites: 50 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- false : Boolean

--- --- --- --- --- --- --- Bag

--- --- --- --- --- --- --- Equal Values
--- 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))
red eval( let c = Bag\{1, -1, 3, 2, 0, 1\} in
  c -> asSequence() =
  c -> iterate(elem ;
  r : Sequence(Integer) = Sequence{} | r
  -> including(elem))
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag\{1, -1, 3, 2, 0, 1\} in c ->
--- asSequence() = c -> iterate (elem ; r : Sequence(Integer) = Sequence{} | r
--- -> including(elem))).
--- rewrites: 127 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 82
--- ?let c = Bag\{1, -1, 3, 2, 0, 1\} in
--- c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Bag\{1, -1, 3, 2, 0, 1\} in
  c -> any(true) = c -> asSequence() -> any(true)
) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag\{1, -1, 3, 2, 0, 1\} in c -> any(
--- true) = c -> asSequence() -> any(true)).
--- rewrites: 64 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean

--- 83
--- ?let c = Bag\{1, -1, 3, 2, 0, 1\} in
--- c -> any(true) = c -> asSequence() -> first()
red eval( let c = Bag{1, -1, 3, 2, 0, 1} in
c -> any(true) = c -> asSequence() -> first()
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1, -1, 3, 2, 0, 1} in c -> any(
---   true) = c -> asSequence() -> first()) .
--- rewrites: 59 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 84
--- ?let c = Bag{1, -1, 3, 2, 0, 1} in
c -> any(true) = c -> asSequence() -> last()
red eval( let c = Bag{1, -1, 3, 2, 0, 1} in
c -> any(true) = c -> asSequence() -> last()
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1, -1, 3, 2, 0, 1} in c -> any(
---   true) = c -> asSequence() -> last()) .
--- rewrites: 57 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- false : Boolean
--- 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()
red eval( let c = Bag{1, -1, 3, 2, 0, 1} in
c -> any(true) =
c -> iterate(elem;
r : Sequence(Integer) = Sequence{} |
r -> including(elem)) -> first()
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1, -1, 3, 2, 0, 1} in c -> any(
---   true) = c -> iterate (elem ; r : Sequence(Integer) = Sequence{} | r ->
---   including(elem)) -> first()) .
--- rewrites: 139 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 86
--- ?let c = Bag{1, -1, 3, 2, 0, 1} in
Sequence{c} -> flatten() = c -> asSequence()
red eval( let c = Bag{1, -1, 3, 2, 0, 1} in
  Sequence{c} -> flatten() = c -> asSequence()
)
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1, -1, 3, 2, 0, 1} in Sequence{
c} -> flatten() = c -> asSequence()) .
--- rewrites: 53 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- --- --- --- --- --- --- No Equal Values
--- 87
--- ?let c = Bag{1, -1, 3, 2, 0} in
--- c -> asSequence() =
--- c -> iterate(elem;
--- r:Sequence(Integer) = oclEmpty(Sequence(Integer)) |
--- r -> including(elem))
red eval( let c = Bag{1, -1, 3, 2, 0} in
c -> asSequence() = c -> iterate(elem;
    r : Sequence(Integer) = Sequence{} |
    r -> including(elem)) ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1, -1, 3, 2, 0} in c ->
--- asSequence() = c -> iterate (elem ; r : Sequence(Integer) = Sequence{} | r
--- -> including(elem))) .
--- rewrites: 110 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 88
--- ?let c = Bag{1, -1, 3, 2, 0} in
--- c -> any(true) = c -> asSequence() -> any(true)
red eval( let c = Bag{1, -1, 3, 2, 0} in
c -> any(true) = c -> asSequence() -> any(true) ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1, -1, 3, 2, 0} in c -> any(
--- true) = c -> asSequence() -> any(true)) .
--- rewrites: 59 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
---
--- 89
--- ?let c = Bag{1, -1, 3, 2, 0} in
--- c -> any(true) = c -> asSequence() -> first()
red eval( let c = Bag{1, -1, 3, 2, 0} in
c -> any(true) = c -> asSequence() -> first() ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1, -1, 3, 2, 0} in c -> any(
--- true) = c -> asSequence() -> first()) .
--- rewrites: 54 in Oms cpu (Oms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
--- true : Boolean
--- 90
--- ?let c = Bag{1, -1, 3, 2, 0} in
--- c -> any(true) = c -> asSequence() -> last()
red eval( let c = Bag{1, -1, 3, 2, 0} in
        c -> any(true) = c -> asSequence() -> last() ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1, -1, 3, 2, 0} in c -> any(
---     true) = c -> asSequence() -> last()) .
--- rewrites: 52 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: false
---
--- Expected Results:
---    false : Boolean
---
--- 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()
red eval( let c = Bag{1, -1, 3, 2, 0} in
        c -> any(true) =
        c -> iterate(elem;
            r : Sequence(Integer) = Sequence{} | r ->
            including(elem)) -> first() ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1, -1, 3, 2, 0} in c -> any(
---     true) = c -> iterate (elem ; r : Sequence(Integer) = Sequence{} | r ->
---     including(elem)) -> first()) .
--- rewrites: 122 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- 92
--- ?let c = Bag{1, -1, 3, 2, 0} in
--- Sequence(c) -> flatten() = c -> asSequence()
red eval( let c = Bag{1, -1, 3, 2, 0} in
        Sequence(c) -> flatten() = c -> asSequence() ) .
--- ==============================================================
--- reduce in BENCHMARK-TEST-B6 : eval(let c = Bag{1, -1, 3, 2, 0} in Sequence(c)
---     -> flatten() = c -> asSequence()) .
--- rewrites: 48 in 0ms cpu (0ms real) (~ rewrites/second)
--- result Bool: true
---
--- Expected Results:
---    true : Boolean
---
--- --- --- --- --- --- --- Sequence
---
--- --- --- --- --- --- --- Equal Values
---
--- --- --- --- --- --- --- No Equal Values
--- class MathLib
--- operations
--- root(arg:Real,precision:Real):Real=
--- if arg>0 then
--- rootAux(arg,
--- precision,
--- if arg>=1 then 1.0 else arg endif,
--- if arg>=1 then arg else 1.0 endif)
--- else 0.0 endif
--- rootAux(arg:Real,precision:Real,low:Real,high:Real):Real=
--- let new:Real=(low+high)/2 in
--- if (arg-new*new).abs<=precision
--- then new
--- else rootAux(arg,
--- precision,
--- if new*new<arg then new else low endif,
--- if new*new<arg then high else new endif)
--- endif
--- end
--- isPrime(arg:Integer):Boolean=
--- if arg<=1 then false else
--- if arg=2 then true else isPrimeAux(arg,2,arg div 2) endif endif
--- isPrimeAux(arg:Integer,cur:Integer,top:Integer):Boolean=
--- if arg.mod(cur)=0 then false else
--- if cur+1<=top then isPrimeAux(arg,cur+1,top) else true endif
--- end
--- end
--- mod CLASSES-BENCHMARK-B7-A is
--- pr mOdCL .
--- op new : -> Vid .
op m1 : -> Oid .
vars a precision low high : Float .
eq root(a, precision) = if a > 0 then
--- rootAux(a,
--- precision,
--- if a >= 1 then 1.0 else a endif,
--- if a >= 1 then arg else 1.0 endif)
--- else 0.0 endif .
eq rootAux(a, precision, low, high)
--- = (let new = (low + high) / 2 in
--- if (a - new * new).abs() <= precision
--- then new
--- else rootAux(a,
precision,
if new * new < a then new else low endif,
if new * new < a then high else new endif)
endif).

vars arg cur top : Nat.
op isPrime : -> OpName [ctor].
eq isPrime(arg) = if arg <= 1 then false else
  if arg = 2 then true else isPrimeAux(arg, 2, arg . div (2)) endif endif.

eq isPrimeAux : -> OpName [ctor].
eq isPrimeAux(arg, cur, top) = if arg . mod(cur) = 0 then false else
  if cur + 1 <= top then isPrimeAux(arg, cur + 1, top) else true endif
endif.
endm

--- model MapWorld
---
--- class Town
--- attributes
--- name:String -- key attribute
--- operations
--- connect():Set(Town)=
---  fst->union(snd)
--- connectPlus():Set(Town)=
---  connectPlusAux(connect())
--- connectPlusAux(aSet:Set(Town)):Set(Town)=
---  let oneStep:Set(Town)=aSet->collect(t|t.connect())->flatten()->asSet() in
---  if oneStep->exists(t|aSet->excludes(t)) then
---    connectPlusAux(aSet->union(oneStep))
---  else aSet endif
--- end
---
--- association Road between
---  Town[0..*] role fst
---  Town[0..*] role snd
--- end
---
--- constraints
---
--- context Town inv uniqueName: Town.allInstances->isUnique(t|t.name)
---
mod CLASSES-BENCHMARK-B7-B is
  pr mOdCL.

    ---- class Town
    sort Town.
    subsort Town < Cid.
    op Town : -> Town [ctor].

    ---- Attributes for Town
    op name : -> AttributeName [ctor].
    op fst : -> AttributeName [ctor].
    op snd : -> AttributeName [ctor].
    op connect : -> OpName [ctor].
eq connect()
    = fst -> union(snd) .

op connectPlus : -> OpName [ctor] .
eq connectPlus()
    = connectPlusAux(connect()) .

ops t oneStep : -> Vid .
var aSet : Set .

eq connectPlusAux(aSet)
    = let oneStep = aSet -> collect(t | t . connect()) -> flatten() -> asSet() in
        if oneStep -> exists(t | aSet -> excludes(t)) then
            connectPlusAux(aSet -> union(oneStep))
        else aSet endif .

endm

--- ------------------------------------------------------------------------------------
mod BENCHMARK-TEST-B7-A is
    inc CLASSES-BENCHMARK-B7-A .
    op i : -> Vid .
    op x : -> Vid .
    op m : -> Oid .
    op base : -> Vid .
    op res : -> Vid .
    op longstr : -> Vid .
    op count : -> Vid .
endm

mod BENCHMARK-TEST-B7-B is
    inc CLASSES-BENCHMARK-B7-B .
    op res : -> Vid .
    ops i s : -> Vid .
    ops Town1 Town2 Town3 Town4 Town5 Town6 Town7 Town8 Town9 Town10 Town11 Town12 Town13 Town14 Town15 Town16 Town17 Town18 Town19 Town20 Town21 Town22 Town23 Town24 Town25 Town26 Town27 Town28 Town29 Town30 Town31 Town32 Town33 Town34 Town35 Town36 Town37 Town38 Town39 Town40 Town41 Town42 : -> Oid .
    op state : -> Configuration .
eq state =
        < Town1 : Town | fst : Set{Town29}, snd : Set{Towns3, Town38, Town39},
            name : "Mild Goosemont north of the Ford" >
        < Town2 : Town | fst : Set{Towns38, Town38}, snd : Set{Town18},
            name : "Foggy Horseport close to the Lake" >
        < Town3 : Town | fst : Set{Town1}, snd : Set{},
            name : "Great Gooseport behind the Castle" >
        < Town4 : Town | fst : Set{}, snd : Set{Town29, Town33},
            name : "Mild Bearport under the River" >
        < Town5 : Town | fst : Set{}, snd : Set{Town13},
            name : "Windy Bearville west of the Hill" >
        < Town6 : Town | fst : Set{Town17}, snd : Set{Town35},
            name : "Hot Dogport above the Cave" >
        < Town7 : Town | fst : Set{Town8, Town24, Town32}, snd : Set{},
            name : "Mild Cowmont east of the Mountain" >
        < Town8 : Town | fst : Set{Town31}, snd : Set{Town7, Town42},
            name : "" >

323
< Town38 : Town | fst : Set{Town1}, snd : Set{Town2, Town41},
   name : "Snowy Lionharbour east of the Sea" >
< Town39 : Town | fst : Set{Town19, Town1}, snd : Set{},
   name : "Hot Bearfield east of the Cave" >
< Town40 : Town | fst : Set{}, snd : Set{Town13},
   name : "Hot Bearview above the Desert" >
< Town41 : Town | fst : Set{Town38}, snd : Set{},
   name : "Small Horsegreen north of the Sea" >
< Town42 : Town | fst : Set{Town8}, snd : Set{Town17},
   name : "Hot Sparrowville south of the Desert" > .
concat("abcdefghijklmnopqrstuvwxyz") .
concat("ABCDEFGHIJKLMNOPQRSTUVWXYZ") in
let count : Integer = 16 in
let longstr : String : String = Set{1 .. count} -> iterate(
  i : Integer ; res : String = "" | res . concat(base)) in
Set{1 .. longstr . size()} -> iterate(i : Integer ; res : String = "" | res . concat(longstr .
  substring(longstr . size() + 1 - i, longstr . size() + 1 - i)))
).
--- =========================================
--- reduce in BENCHMARK-TEST-B7-A : eval(let base : String = "0123456789" . concat(
  "abcdefghijklmnopqrstuvwxyz") . concat("ABCDEFGHIJKLMNOPQRSTUVWXYZ") in let
  count : Integer = 16 in let longstr : String : String = Set{1 .. count} ->
  iterate (i : Integer ; res : String = "" | res . concat(base)) in Set{1 ..
  longstr . size()} -> iterate (i : Integer ; res : String = "" | res .
  concat(longstr . substring(longstr . size() + 1 - i,longstr . size() + 1 -
  i)))) .
--- rewrites: 44963 in 5760ms cpu (5940ms real) (7805 rewrites/second)
--- result String: "ZYX...210"
---
--- Expected Result:
--- 'ZYX ... 210' : String
---
--- Compute all prime numbers up to a given upper bound ------------
---
--- 3
--- ?Sequence{1..2048} -> iterate(i:Integer;res:Sequence(Integer)=
  oclEmpty(Sequence(Integer)) | if m.isPrime(i) then res -> including(i) else res endif)
red eval( Sequence{1 .. 2049} -> iterate(i : Integer ; res : Sequence(Integer) =
  Sequence{} | if m . isPrime(i) then res -> including(i) else res endif)
).
--- 326
---
--- Expected Result:
--- Sequence(2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,
--- 73,79,83,89,97, ..., 2003,2011,2017,2027,2029,2039) :
--- Sequence(Integer)
---
--- 4
--- ?Sequence{1..16} -> iterate(i;res:Sequence(Sequence(OclAny))=
  oclEmpty(Sequence(Sequence(OclAny))) | res -> including(Sequence(i,m.isPrime(i)))) -> select(last()=true)
red eval( Sequence{1 .. 16} -> iterate(i,m . isPrime(i))) -> select(x | x -> last() = true)
).
--- reduce in BENCHMARK-TEST-B7-A : eval(Sequence{1 .. 16} -> iterate (i ; res :
---  Sequence(Sequence(OclAny)) = Sequence{} | res -> including(Sequence(i, m .
---  isPrime(i)))) ) -> select (x | x -> last() = true)) .
--- rewrites: 1640 in 0ms cpu (1ms real) (~ rewrites/second)
--- result Sequence: Sequence{Sequence{2, true}, Sequence{3, true}, Sequence{5,
---  true}, Sequence{7, true}, Sequence{11, true}, Sequence{13, true}}
---
--- Expected Result:
---  Sequence{Sequence{2,true},Sequence{3,true},Sequence{5,true},
---     Sequence{7,true},Sequence{11,true},Sequence{13,true}} :
---     Sequence(Sequence(OclAny))
---
--- Compute the root of 10.89 with precision 1/10^-16 ---------------
--- 5
--- ?Set{1..16} -> iterate (i ; res : Real =
---  m . root(10.89,
---  1/(10*10*10*10*10*10*10*10*10*10*10*10*10*10*10*10) | res)
red eval( Set{1 .. 16} -> iterate (i ; res : Real =
---  m . root(10.89,
---=====================================================================
--- reduce in BENCHMARK-TEST-B7-A : eval(Set{1 .. 16} -> iterate (i ; res : Real =
---  m . root(1.0890000000000001e+1, 1 / (1.0e+1 * 1.0e+1 * 1.0e+1 * 1.0e+1 *
---  1.0e+1 * 1.0e+1 * 1.0e+1 * 1.0e+1 * 1.0e+1 * 1.0e+1 * 1.0e+1 * 1.0e+1 *
---  1.0e+1 * 1.0e+1) | res)) .
--- rewrites: 3641 in 8ms cpu (6ms real) (455068 rewrites/second)
--- --- result FiniteFloat: 3.300000000066939 : Real
--- --- Expected Result:
--- 3.300000000066939 : Real
---
--- File: world_0.2.3.13.23.maude
---
--- 10.2. Queries based on UML Model B
---
--- 1
--- use> ?Town.allInstances->collect(t|t.connectPlus()->size())->asSet()
red eval( Town . allInstances -> collect (t | t . connectPlus() ->
size()) -> asSet(), state) .
---=====================================================================
--- reduce in BENCHMARK-TEST-B7-B : eval(Town . allInstances -> collect (t | t .
---  connectPlus() -> size()) -> asSet(), state) .
--- rewrites: 252732 in 136ms cpu (152ms real) (1858214 rewrites/second)
--- result Set: Set{0, 2, 3, 13, 21, 23}
---
--- Expected result:
--- Set{0,2,3,13,23} : Set(Integer)
---
--- 2
--- use> ?Set{1..1024*1024}->iterate(i; res:Bag(Set(Town))=Town.allInstances->collect(t|t.connectPlus())|res)
red eval( Set{1 .. 1024} -> iterate(i ;
   res : Bag(Set(Town)) = Town . allInstances -> collectNested(t | t . connectPlus()) | res), state).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B7-B : eval(Set{1 .. 1024} -> iterate (i ; res : Bag(
   Set(Town)) = Town . allInstances -> collectNested (t | t . connectPlus()) |
   res), state).
--- rewrites: 381908 in 7200ms cpu (7590ms real) (53039 rewrites/second)
--- result Bag: Bag(Town42, Town8, Town17, Town31, Town7, Town6, Town12,
   Town34, Town24, Town32, Town35, Town37, Town27), Set(Town38, Town41, Town1,
   ... ... ...
   Town36, Town26, Town33, Town13, Town23, Town5, Town40, Town22), Set(Town1,
   Town29, Town3, Town38, Town39, Town4, Town16, Town2, Town41, Town19,
   Town33, Town36, Town9, Town13, Town18, Town23, Town11, Town26, Town5,
   Town40, Town22))
---
--- Expected result:
---   Bag(Sets{Town21, Town25}, ...)
---       Set(Town1,Town11,Town13,Town14,Town16,Town18,Town19,Town2,
---       Town22,Town23,Town26,Town28,Town29,Town3,Town33,Town36,
---       Town38,Town39,Town4,Town40,Town41,Town5,Town9)} : 
---       Bag(Set(Town))
---
--- 3
--- use> ?Town.allInstances->select(t|t.connectPlus()->size()=0)
red eval( Town . allInstances -> select(t | t . connectPlus() -> size () = 0), state).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B7-B : eval(Town . allInstances -> select (t | t .
---       connectPlus() -> size() = 0), state).
--- rewrites: 252781 in 136ms cpu (156ms real) (1858574 rewrites/second)
--- result Set: Set(Town20)
---
--- Expected result:
---   Set(Town20) : Set(Town)
---
--- 4
--- use> ?Town.allInstances->select(t|t.connectPlus()->size()=2)
red eval( Town . allInstances -> select(t | t . connectPlus() -> size () = 2), state).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B7-B : eval(Town . allInstances -> select (t | t .
---       connectPlus() -> size() = 0), state).
--- rewrites: 370973 in 976ms cpu (1178ms real) (380071 rewrites/second)
--- result Set: Set(Town20)
---
--- Expected result:
---   Set(Town21,Town25) : Set(Town)
---
--- 5
--- use> ?Town.allInstances->select(t|t.connectPlus()->size()=3)
red eval( Town . allInstances -> select(t | t . connectPlus() -> size () = 3), state).
--- ==============================================================
--- reduce in BENCHMARK-TEST-B7-B : eval(Town . allInstances -> select (t | t .
---       connectPlus() -> size() = 0), state).
--- rewrites: 370977 in 984ms cpu (1276ms real) (376985 rewrites/second)
--- result Set: Set(Town30, Town15, Town10)
--- Expected result:
--- Set(Town10, Town15, Town30) : Set(Town)

--- 6
--- use> & Town.allInstances->select(t | t.connectPlus()->size() = 13)
red eval( Town . allInstances -> select(t | t . connectPlus() -> size() = 13) , state) .
--- ========
--- reduce in BENCHMARK-TEST-B7-B : eval(Town . allInstances -> select(t | t .
--- connectPlus() -> size() = 13) , state) .
--- rewrites: 370997 in 932ms cpu (1188ms real) (398040 rewrites/second)
--- result Set: Set(Town42, Town37, Town35, Town34, Town32, Town31, Town27, Town24,
--- Town17, Town12, Town8, Town7, Town6)
---
--- Expected result:
--- Set(Town12, Town17, Town24, Town27, Town31, Town32, Town34, Town35,
--- Town37, Town42, Town6, Town8) : Set(Town)

--- 7
--- use> & Town.allInstances->select(t | t.connectPlus()->size() = 23)
red eval( Town . allInstances -> select(t | t . connectPlus() -> size() = 23) , state) .
--- ========
--- reduce in BENCHMARK-TEST-B7-B : eval(Town . allInstances -> select(t | t .
--- connectPlus() -> size() = 23) , state) .
--- rewrites: 424823 in 1112ms cpu (1249ms real) (382011 rewrites/second)
--- result Set: Set(Town41, Town40, Town39, Town38, Town36, Town33, Town29, Town28,
--- Town26, Town23, Town22, Town19, Town18, Town16, Town14, Town13, Town11,
--- Town9, Town5, Town4, Town3, Town2, Town1)
---
--- Expected result:
--- Set(Town1, Town11, Town13, Town14, Town16, Town18, Town19, Town2, Town22,
--- Town23, Town26, Town28, Town29, Town3, Town33, Town36, Town38, Town39,
--- Town4, Town40, Town41, Town5, Town9) : Set(Town)

--- 8
--- use> & Town.allInstances->
--- collect(t | Seq{t, t.connect()->size()}) ->
--- sortedBy(s | s->last()->oclAsType(Integer))
red eval( Town . allInstances ->
--- collectNested(t | Sequence(t, t . connect() -> size()) ->
--- sortedBy(s | s -> last()))
, state) .
--- ========
--- reduce in BENCHMARK-TEST-B7-B : eval(Town . allInstances -> collectNested(t |
--- Sequence(t, t . connect() -> size()) -> sortedBy (s | s -> last()), state)
--- .
--- rewrites: 5009 in 12ms cpu (17ms real) (417381 rewrites/second)
--- result Sequence: Sequence{Sequence{Town20, 0}, Sequence{Town3, 1}, Sequence{
--- Town5, 1}, Sequence{Town10, 1}, Sequence{Town12, 1}, Sequence{Town21, 1},
--- Sequence{Town22, 1}, Sequence{Town23, 1}, Sequence{Town25, 1}, Sequence{
--- Town26, 1}, Sequence{Town27, 1}, Sequence{Town30, 1}, Sequence{Town33, 1},
--- Sequence{Town34, 1}, Sequence{Town37, 1}, Sequence{Town40, 1}, Sequence{
--- Town41, 1}, Sequence{Town2, 2}, Sequence{Town4, 2}, Sequence{Town6, 2},
--- Sequence{Town11, 2}, Sequence{Town14, 2}, Sequence{Town15, 2}, Sequence{
--- Town19, 2}, Sequence{Town24, 2}, Sequence{Town28, 2}, Sequence{Town32, 2},
--- Sequence{Town35, 2}, Sequence{Town36, 2}, Sequence{Town39, 2}, Sequence{
--- Town42, 2}, Sequence{Town7, 3}, Sequence{Town8, 3}, Sequence{Town18, 3},
---
--- 329
Sequence(Town29, 3), Sequence(Town31, 3), Sequence(Town38, 3), Sequence(Town1, 4), Sequence(Town9, 4), Sequence(Town13, 4), Sequence(Town16, 4),
Sequence(Town17, 5))

Expected result:
Seq(Seq(Town20, 0), Seq(Town10, 1), Seq(Town12, 1), Seq(Town21, 1),
Seq(Town22, 1), Seq(Town23, 1), Seq(Town25, 1), Seq(Town26, 1),
Seq(Town27, 1), Seq(Town3, 1), Seq(Town30, 1), Seq(Town33, 1),
Seq(Town34, 1), Seq(Town37, 1), Seq(Town40, 1), Seq(Town41, 1),
Seq(Town5, 1), Seq(Town11, 2), Seq(Town14, 2), Seq(Town15, 2),
Seq(Town19, 2), Seq(Town24, 2), Seq(Town28, 2), Seq(Town32, 2),
Seq(Town35, 2), Seq(Town36, 2), Seq(Town39, 2), Seq(Town4, 2),
Seq(Town42, 2), Seq(Town6, 2), Seq(Town18, 3), Seq(Town2, 3),
Seq(Town29, 3), Seq(Town31, 3), Seq(Town38, 3), Seq(Town7, 3),
Seq(Town8, 3), Seq(Town1, 4), Seq(Town13, 4), Seq(Town16, 4),
Seq(Town9, 4), Seq(Town17, 5)) : Seq(Seq(OclAny))