ELTE | FACULTY OF INFORMATICS

PROGRAMMING

Lecture 7

Zsuzsa Pluhár

pluharzs@inf.elte.hu

Matrix

Basics:

sequence that

- has elements from the same type (same typed data elements);
- referring to the elements need to use 2 indexes

Example$_1$: Chess game

Basic set of elements:

{♔, ♕, ♖, ♗, ♘, ♙,
♚, ♛, ♜, ♝, ♞, ♟, empty}

Matrix

Example$_2$:

We measure in M different place (city) the temperature on N days. The basic set of elements: $\mathbb{N}$ (the temperatures – depends on the precision)

day \ place	1	…	M
1	the temperature of the 1st place on the 1st day		the temperature of the Mth place on the 1st day
…			
N	the temperature of the 1st place on the Nth day		the temperature of the Mth place on the Nth day

Matrix- specification

Sequence: homogeneous = finite length sequence, from the ***same typed*** data elements; we can refer to the elements with two indexes.

Elements: we refer to the (i,j)th element of the sequence: **`S[i,j]`**

Index: `i∈1..rowLength,j∈1..colLength`

Matrix- specification

Task: We store a position of a chess game. Let's count how many black and white chess pieces can be found on the board.

Representation idea:

- Storing the chess peaces: pawn=1, knight=2, bishop=3, castle=4, queen=5, king=6;
- White pieces with a positive, black pieces with a negative value;
- The empty field=0

Specification:

Input: `Position[1..8,1..8]`$\in\mathbb{N}^{8\times 8}$

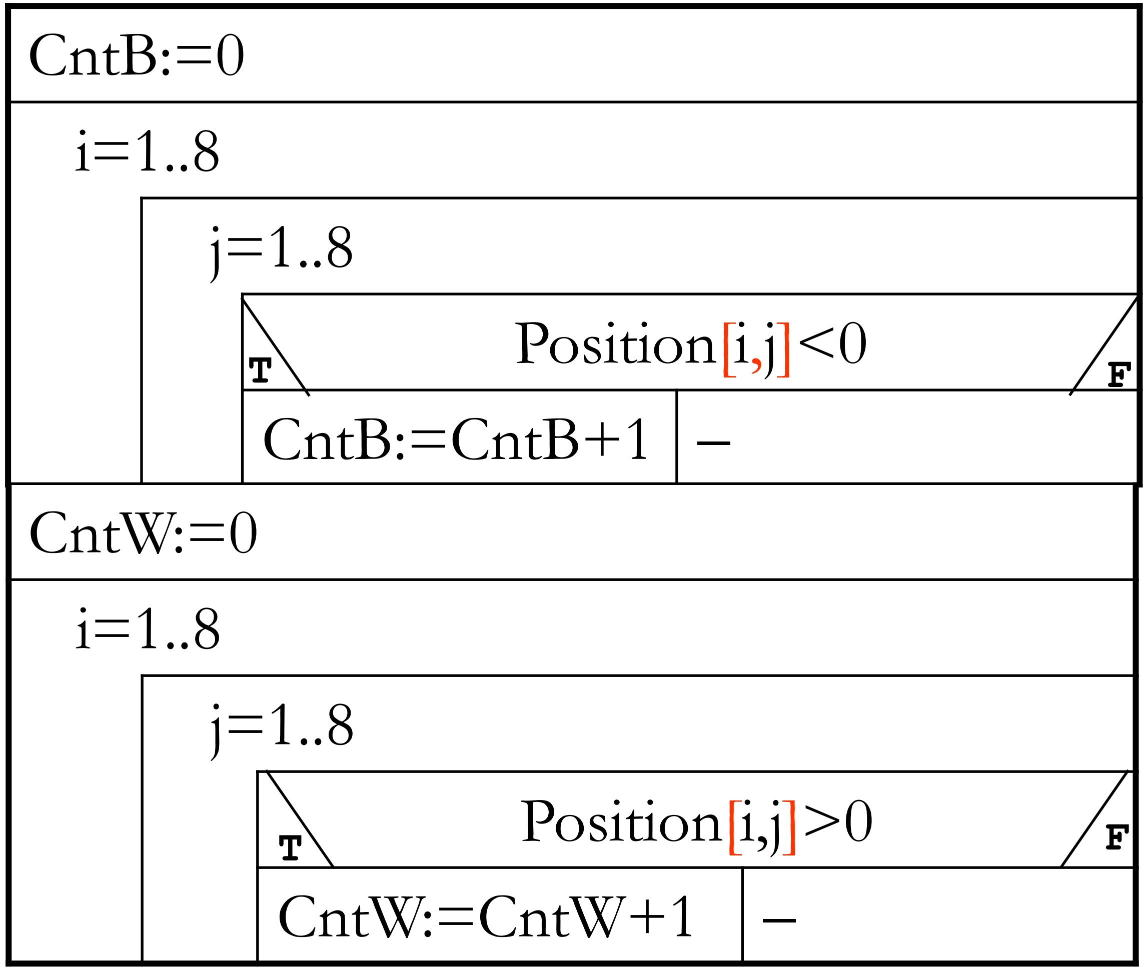
Output: `CntB,CntW`$\in\mathbb{N}$

Precondition: $\forall$`i,j(1≤i≤8):|Position[i,j]|`$\in$`[0..6]`

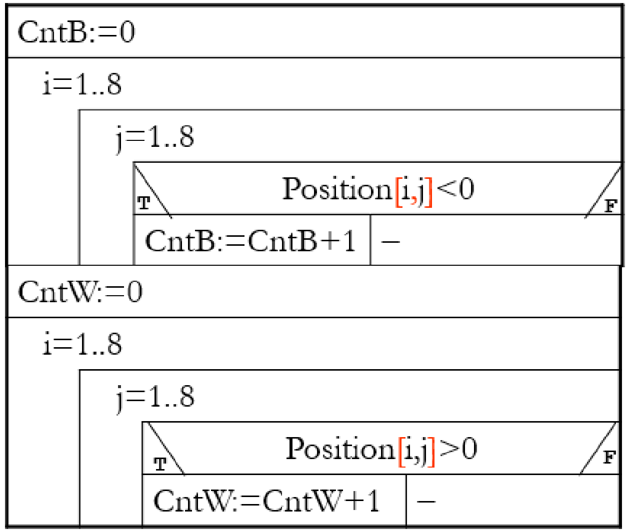
Postcondition:

$$CntB = \sum_{i=1}^{8}\sum_{j=1}^{8} 1 \quad , \quad CntW = \sum_{i=1}^{8}\sum_{j=1}^{8} 1$$

Matrix- algorithm



Matrix- code




```
int[,] Position=new int[8,8];
int CntW,CntB;

CntB=0;
for(int i=0;i<8;i++){
  for(int j=0;j<8;j++){
    if (Position[i,j]<0){
        CntB++;}
  }
}
CntW=0;
for(int i=0;i<8;i++){
  for(int j=0;j<8;j++){
    if (Position[i,j]>0){
        CntW++;}
  }
}
```

Matrix- summation PoA

Task: The sum of elements of a matrix.

Specification:

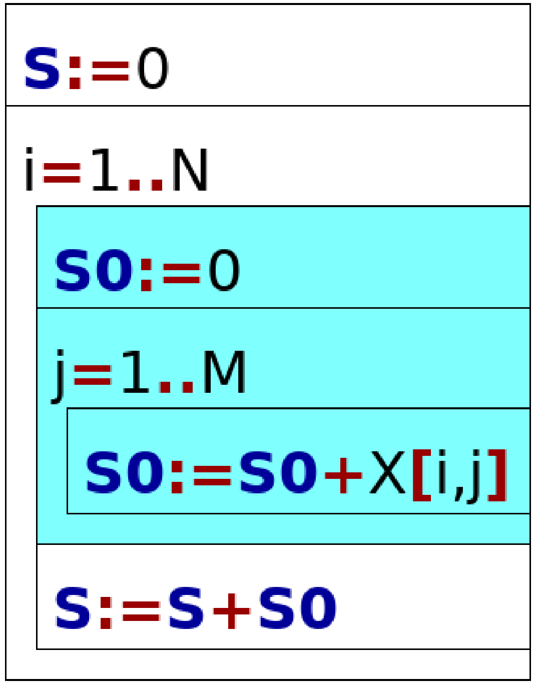
Input: n,m∈ℕ, X[1..n,1..m]∈$\mathbb{Z}^{n\times m}$

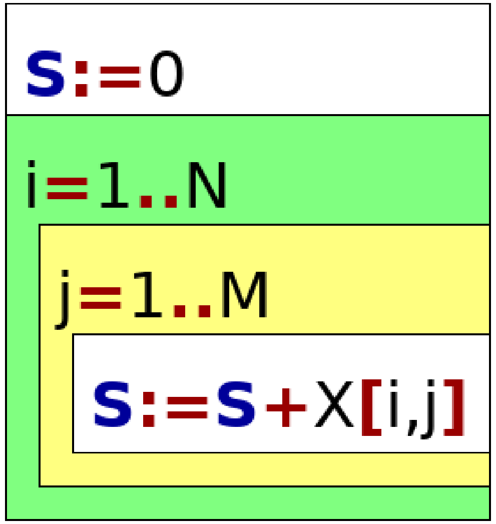
Output: s∈ℤ

Precondition: –

Postcondition: $$s = \sum_{i=1}^{n}\left(\sum_{j=1}^{m} X[i,j]\right)$$

Even smaller difference from basic pattern: just two embedded loops





Comment: copy, count and maximum selection can be done similarly for matrices

Matrix- decision

Task: Is there an element with a given attribute in the matrix?

Specification:

Input: $n,m\in\mathbb{N},\ X[1..n,1..m]\in\mathbb{S}^{n\times m},\ A:\mathbb{S}\to\mathbb{L}$

Output: $Exists\in\mathbb{L}$

Precondition: –

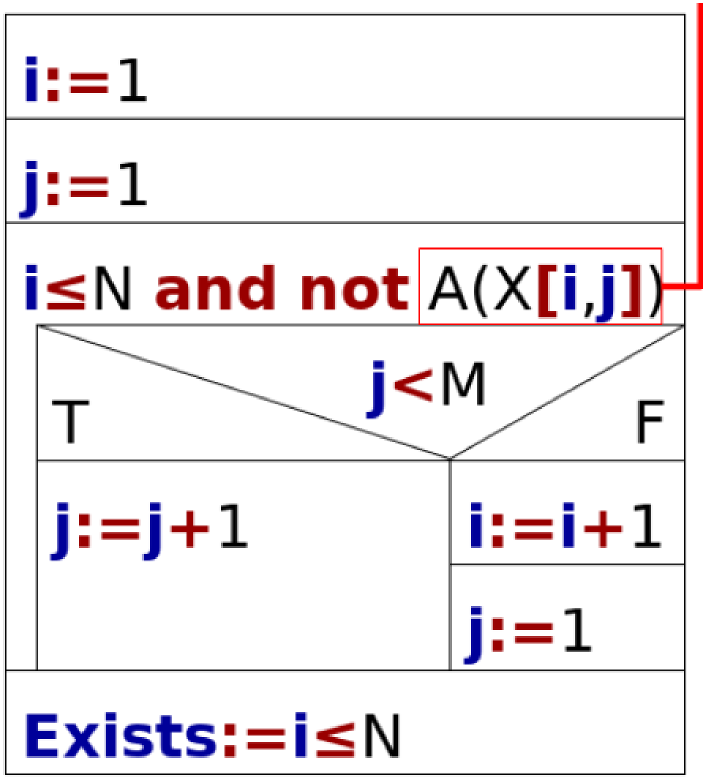
Postcondition:

$Exists=\exists i(1\leq i\leq n):\ \exists j(1\leq j\leq m):\ A(X[i,j])$

Matrix- decision

Algorithm:

You have to define the way of going through the **elements** of the matrix, but not necessarily through all the elements – **line by line**, from left to right



Comment: selection and search can be done similarly

Type definition - set

Value set:

- The iteration of the base set (that is defined by the element type) – „which items can be in the set?”
- The element type is usually a finite, discrete type, sometimes even the count of elements is limited (<256)
- If it does not exist in the language, then the implementation might allow more elements

Type definition - set

Operations (implementation)

- ToSet (puts an element into the set): $S:=S\cup\{e\}$
- FromSet (discards an element from the set): $S:=S\setminus\{e\}$
- Read (reading in the set)
- Write (writing out the set)
- Empty (creating an empty set) or Empty'SetType pre-defined constant
- Empty? (boolean function)

Type definition - set

Operations (mathematical)

- Intersection: $A \cap B$
- Union: $A \cup B$
- Difference: $A \setminus B$
- Complement: A' (not always implementable)
- Element of set: $a \in A$
- Subset: $A \subseteq B$, $A \subset B$

Sequence → set transformation

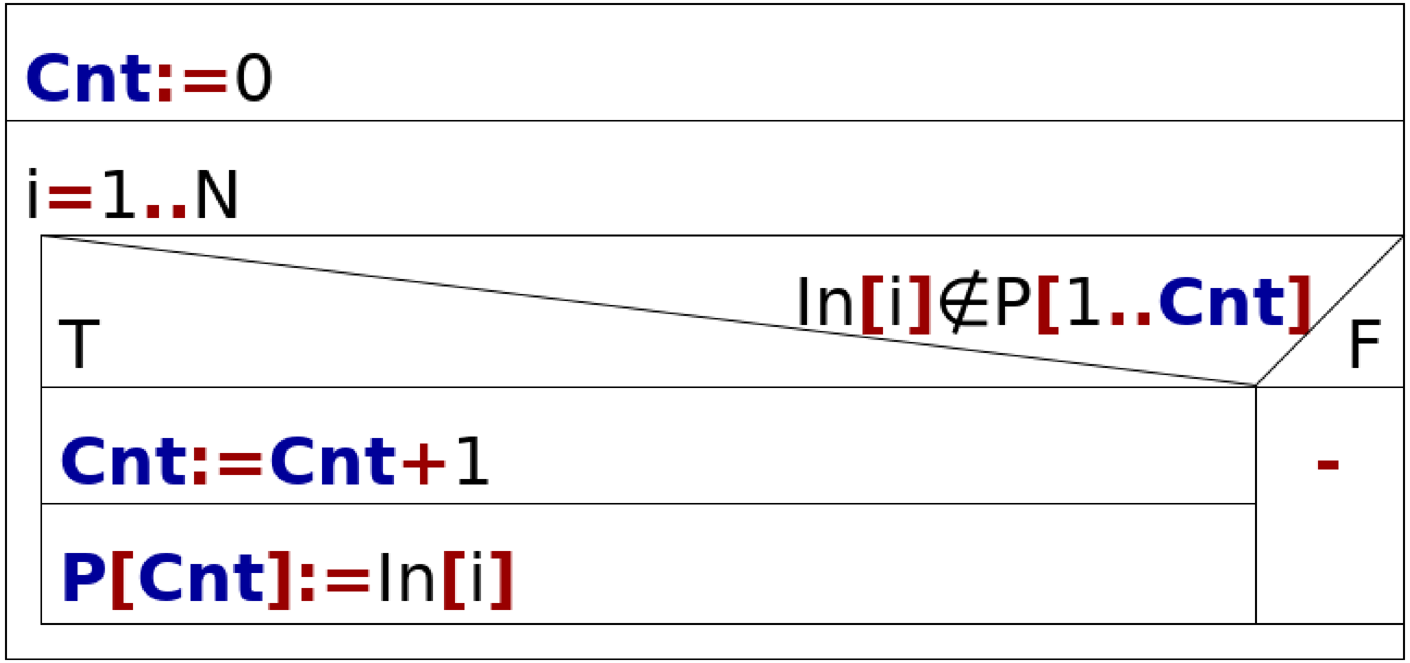
In some tasks, like the ones using the intersection and union pattern of algorithm, the starting data are in a **set**. If we get a sequence as input, we might need to make a set from it.

Example: We know about N shoppings which products customers bought (`In[1..N]`). List the products bought by customers (`P[1..Cnt]`).

Solution: multiple item selection programming pattern – select all the items from the input that have not been put in the result set of the selection yet (decision).

Sequence → set transformation

Algorithm



Type definition - set

Representation

- Listing the elements
- A boolean vector - bitmap

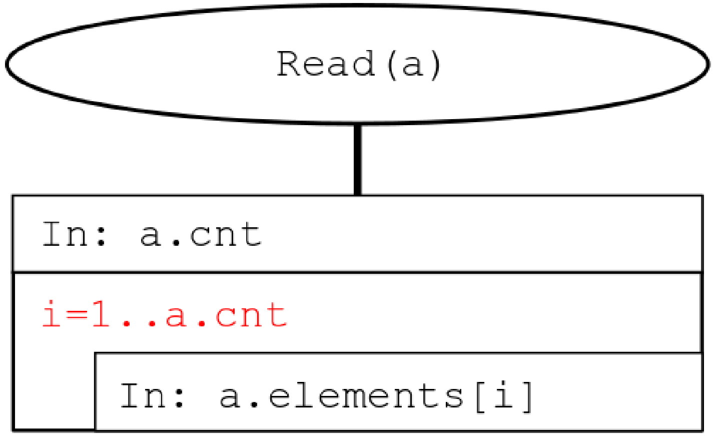
Set – by listing elements

Representation: by listing the elements

```
Set(ElementType)=
  Record(cnt:integer,
  elements:Array(1..MaxCnt:ElementType))
```

We give the set by listing its elements in an array whose length is the same as the count of elements of the set (more precisely, in the first Cnt elements).

Set – by listing elements - READ

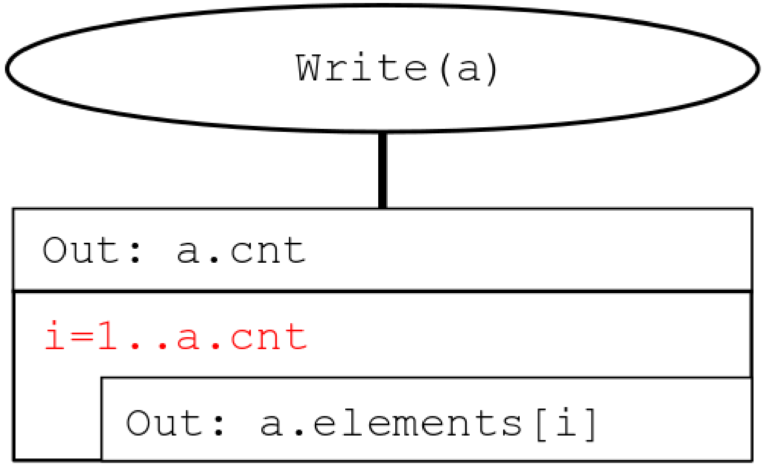


We assume that we read in a **set**.

Calculation of the operation need:

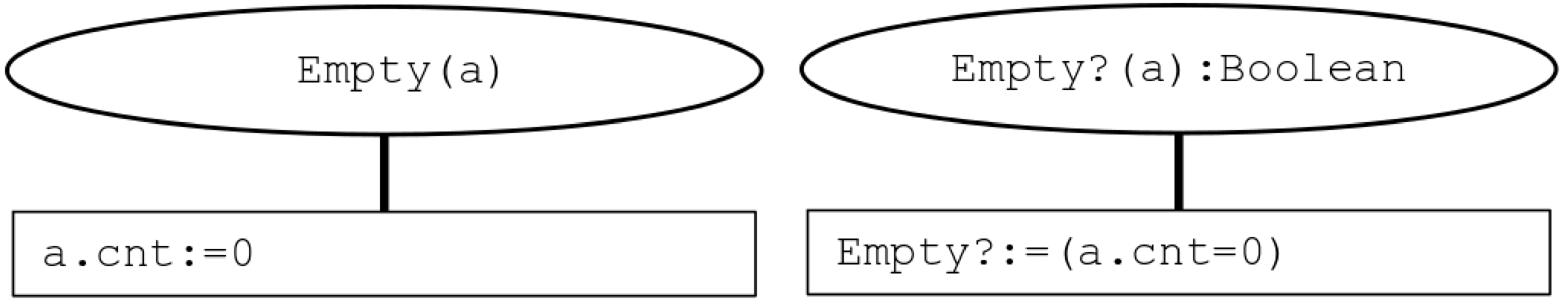
The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set.

Set – by listing elements - WRITE



Calculation of the operation need:
The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set.

Set – by listing elements – EMPTY, EMPTY?

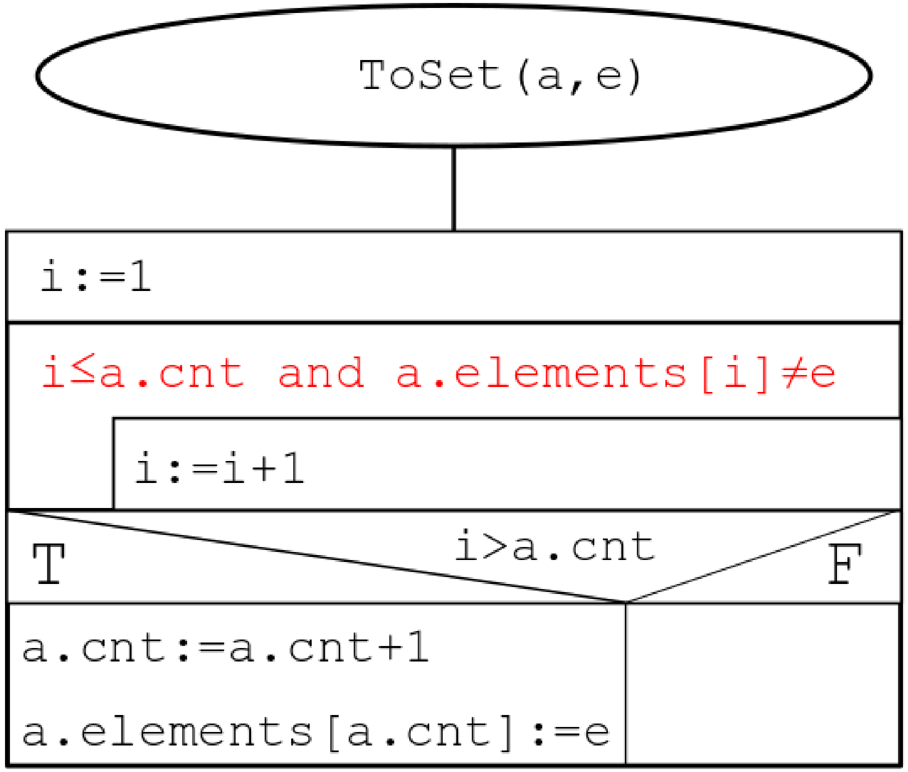


Calculation of the operation need:
It does not depend on the count of elements of the set.

Set – by listing elements – ToSet

Applying the **Decision** PoA

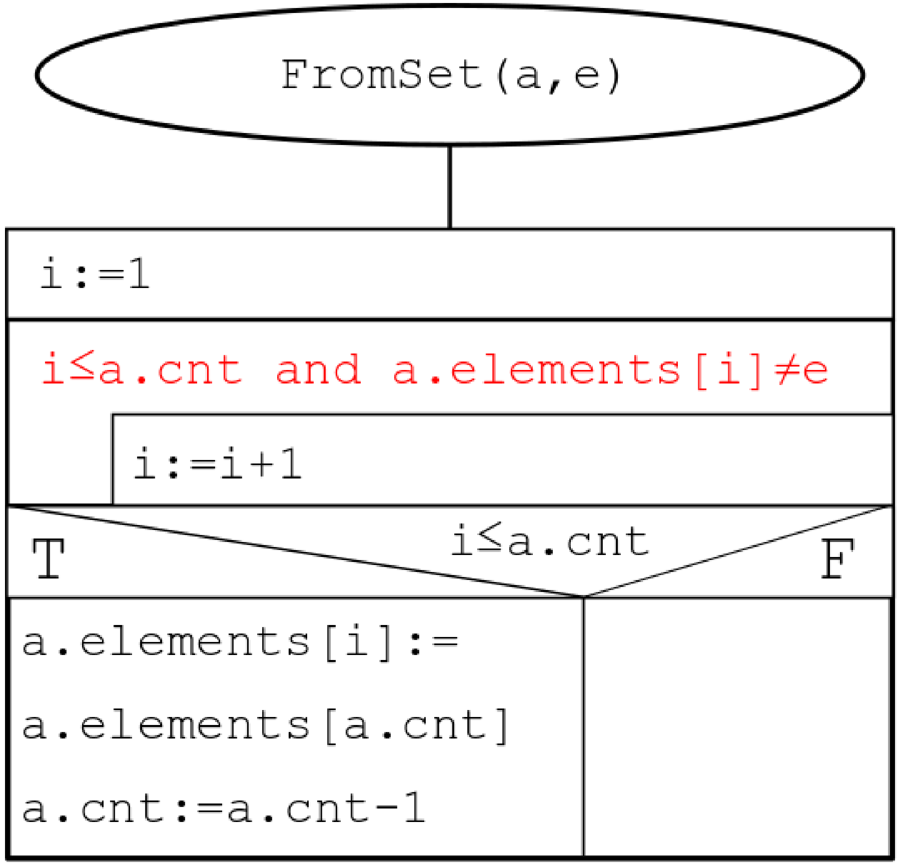
Calculation of the operation need:
The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set.



Set – by listing elements – FromSet

Applying the **Search** PoA

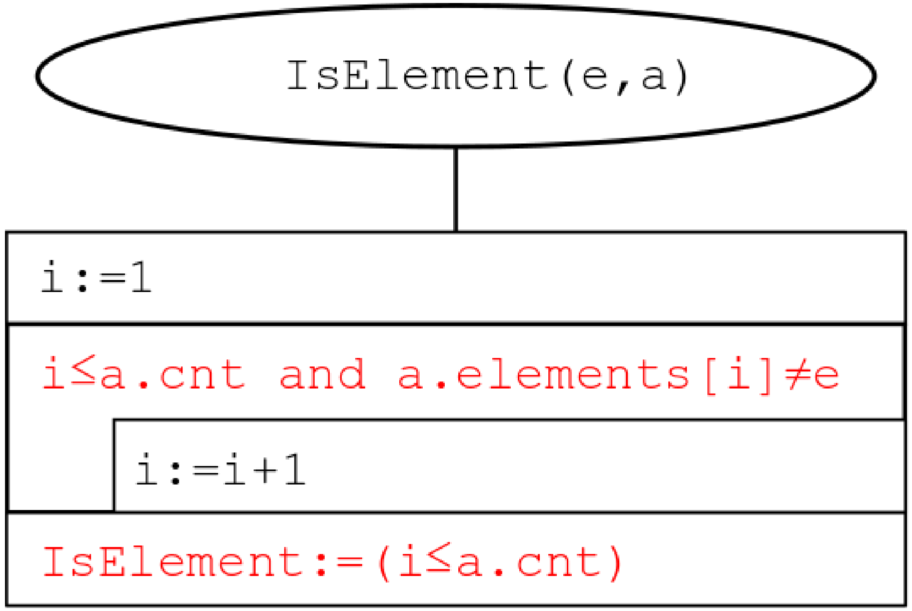
Calculation of the operation need:
The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set.



Set – by listing elements – IsElement

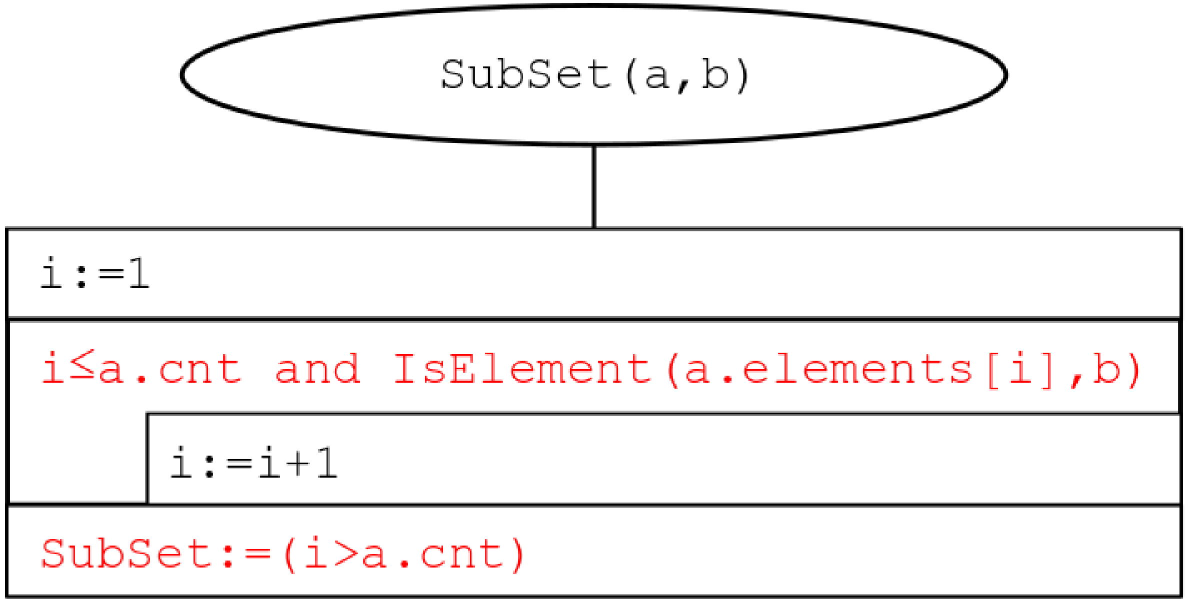
Applying the **Decision** PoA

Calculation of the operation need:
The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set.



Set – by listing elements – SubSet

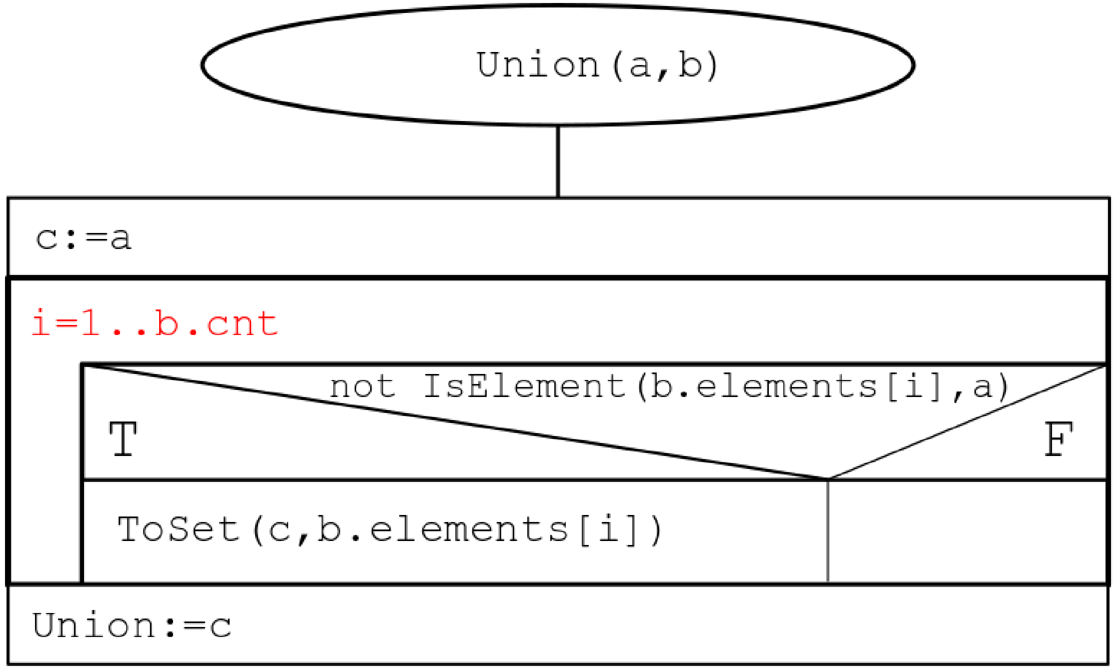
Applying the **Decision** PoA with decision in the conditional statement



Calculation of the operation need:
The loop will run as many times as many elements there are in set A, the IsElement function as many times as many elements there are in set B, thus, the runtime is proportional to the product of the count of elements in the two sets.

Set – by listing elements – SubSet

Applying the **Copy, Multiple item selection, Decision** PoA's

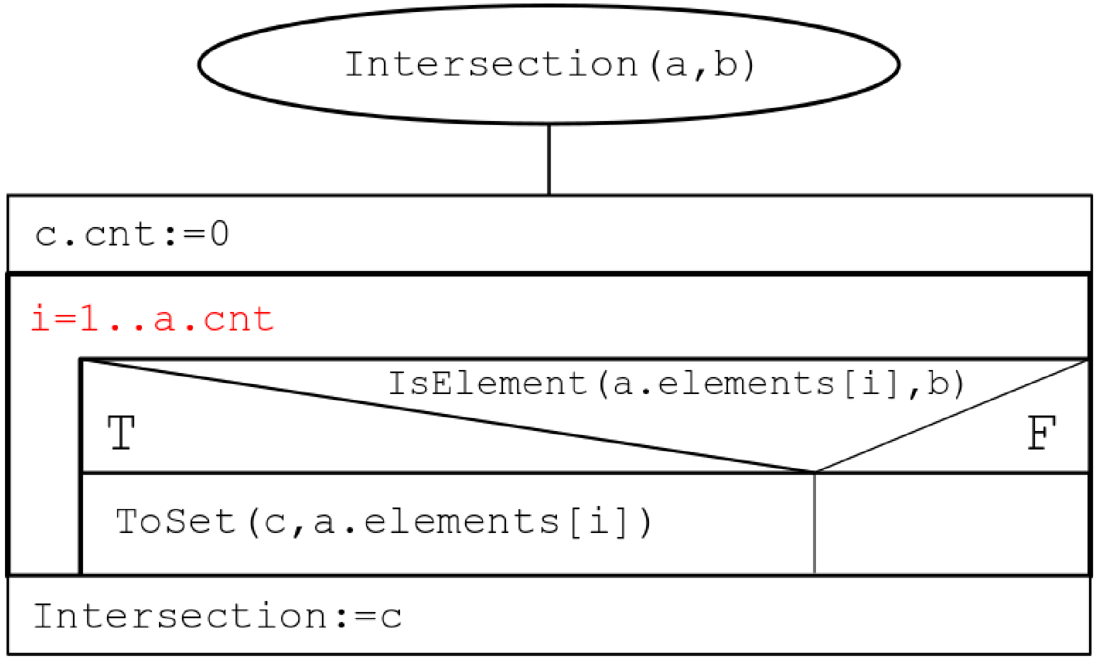


Calculation of the operation need:
The loop will run as many times as many elements there are in set B, the IsElement function as many times as many elements there are in set A, thus, the runtime is proportional to the product of the count of elements in the two sets.

The set type – by listing elements – Intersection

Applying the **Multiple item selection, Decision**

PoA's



Calculation of the operation need:
The loop will run as many times as many elements there are in set A, the IsElement function as many times as many elements there are in set B, thus, the runtime is proportional to the product of the count of elements in the two sets.

The set type – by listing elements

Notes:

- Problem: it is not checked if only elements that should be in the set are actually in the set.
- No limit on the type of elements stored in the set, as we can store **anything** in an vector
- **No limit** on the **count of elements** of the base set that the elements of the set derive from. We only limit the count of elements of the specific sets.

The set type – as boolean vector

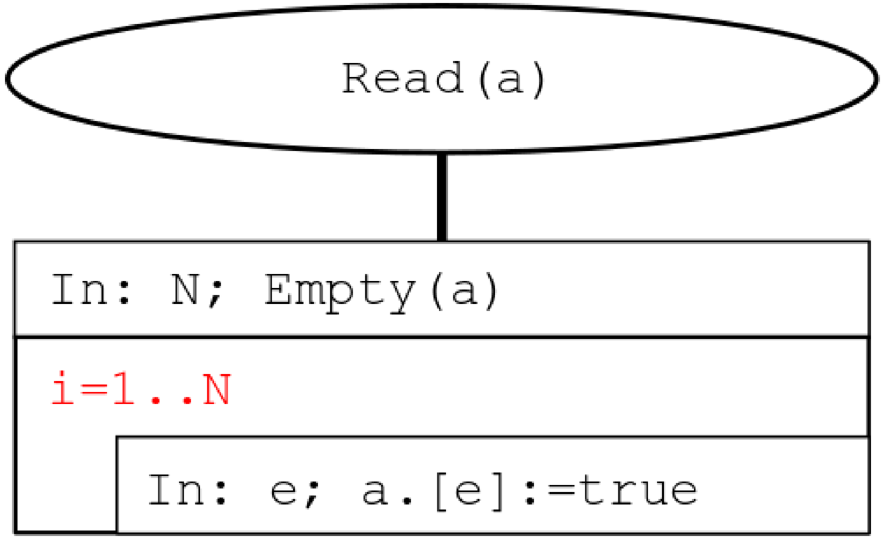
Bit map – boolean vector:

`Set(ElementType)=Array(Min'ElementType..Max'ElementType: Boolean)`

We interpret the set as a vector of `{true,false}` elements, where we use the **value** of the element as **index**.

Such a set is always **sorted**.

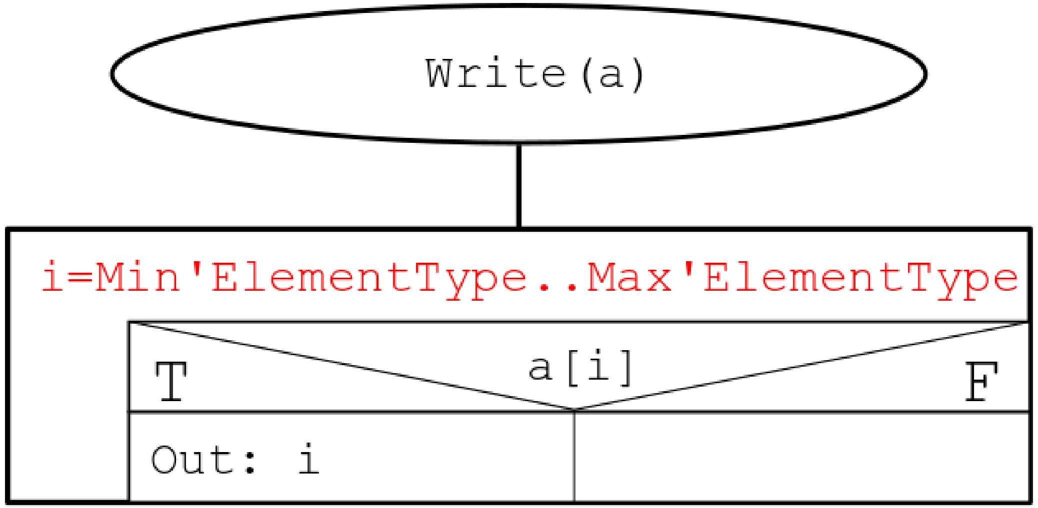
The set type – as boolean vector - READ



Calculation of the operation need:

The operation need of Empty(a) and the loop. The loop will run as many times as many elements there are in the set – thus, the runtime is proportional to the count of elements of the set

The set type – as boolean vector - WRITE

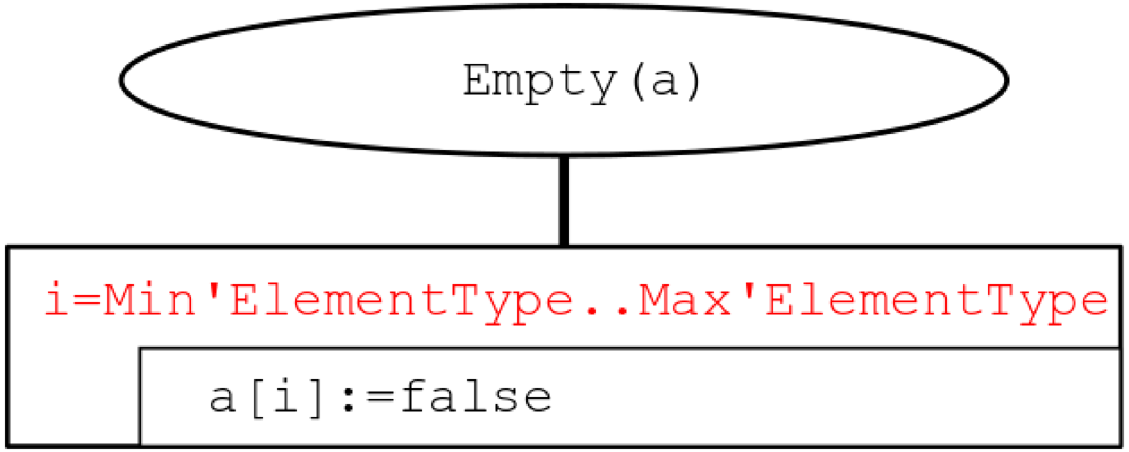


Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

What if we stored the maximum and minimum element of the set?

The set type – as boolean vector - EMPTY

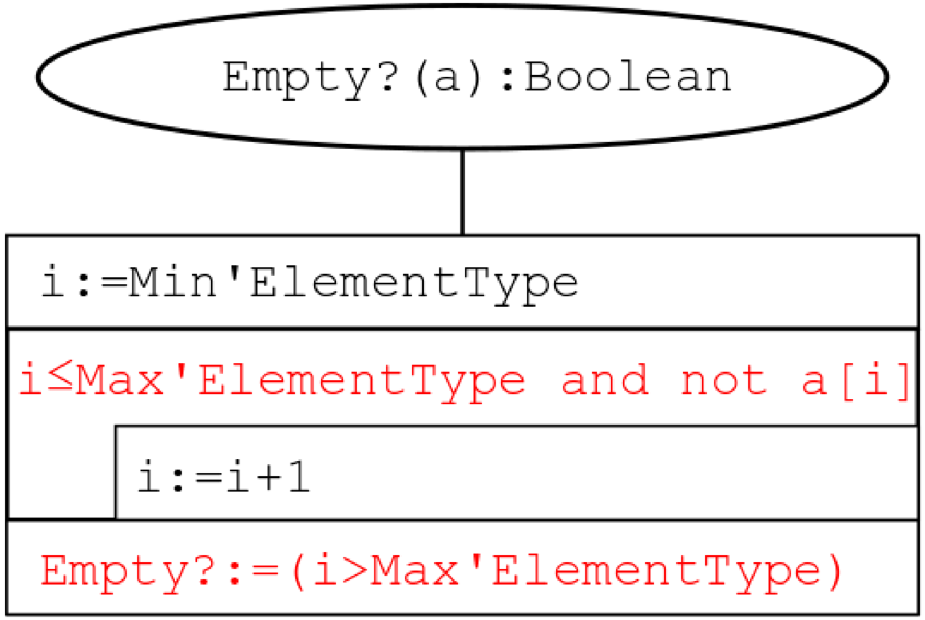


Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

The set type – as boolean vector – EMPTY?

Applying the **Decision**
PoA



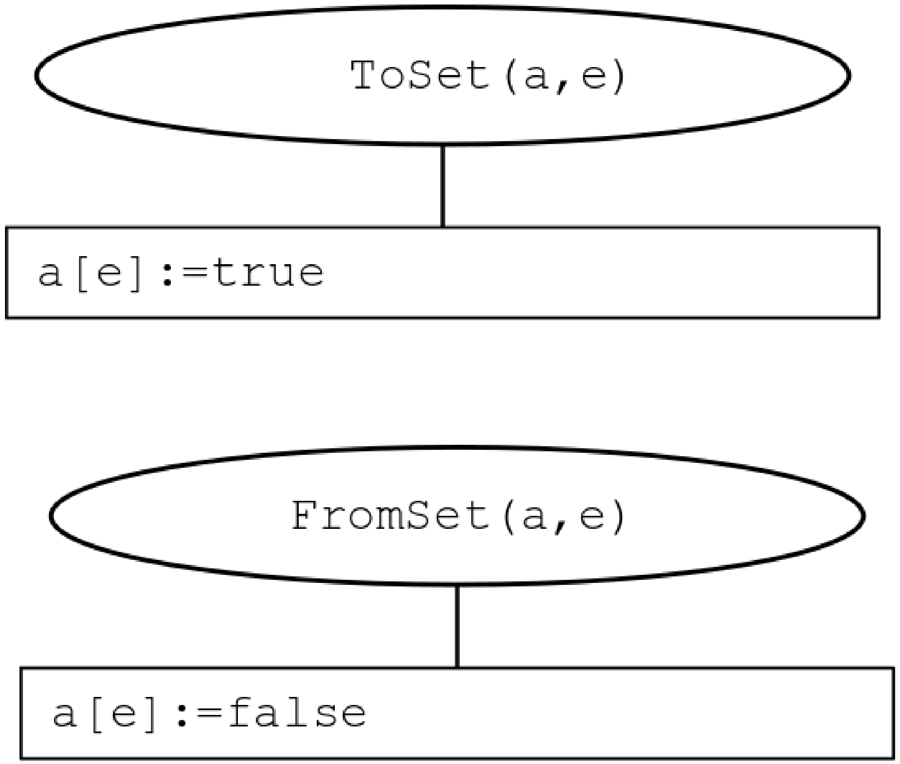
Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

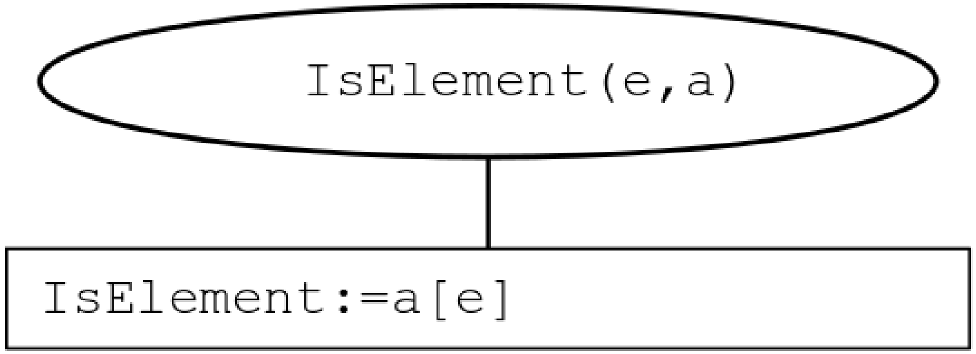
The set type – as boolean vector – ToSet, FromSet

Calculation of the operation need:

It does not depend on the count of elements of the set.



The set type – as boolean vector – IsElement



Calculation of the operation need:

It does not depend on the count of elements of the set.

The set type – as boolean vector – SubSet

Applying the **Decision** PoA

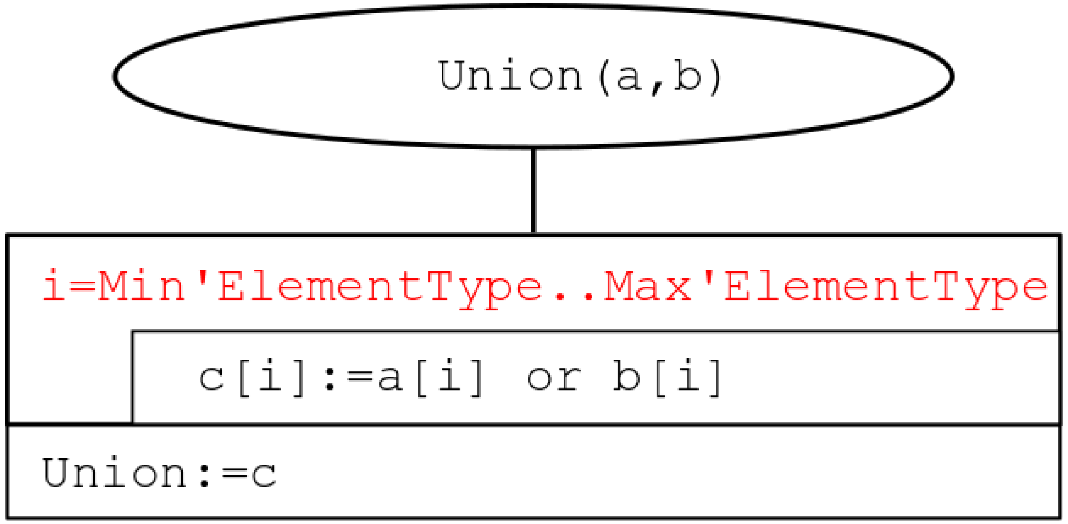
```
SubSet(a,b)

i:=Min'ElementType
i≤Max'ElementType and not (a[i] and not b[i])
    i:=i+1
SubSet:=(i>Max'ElementType)
```

Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

The set type – as boolean vector – Union

Applying the
Copy PoA

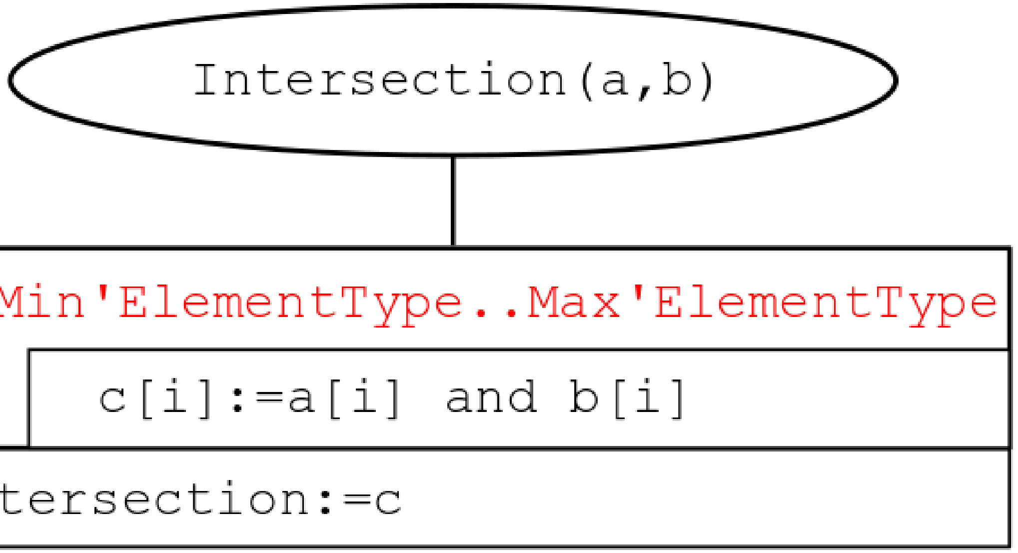


Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

The set type – as boolean vector – Intersection

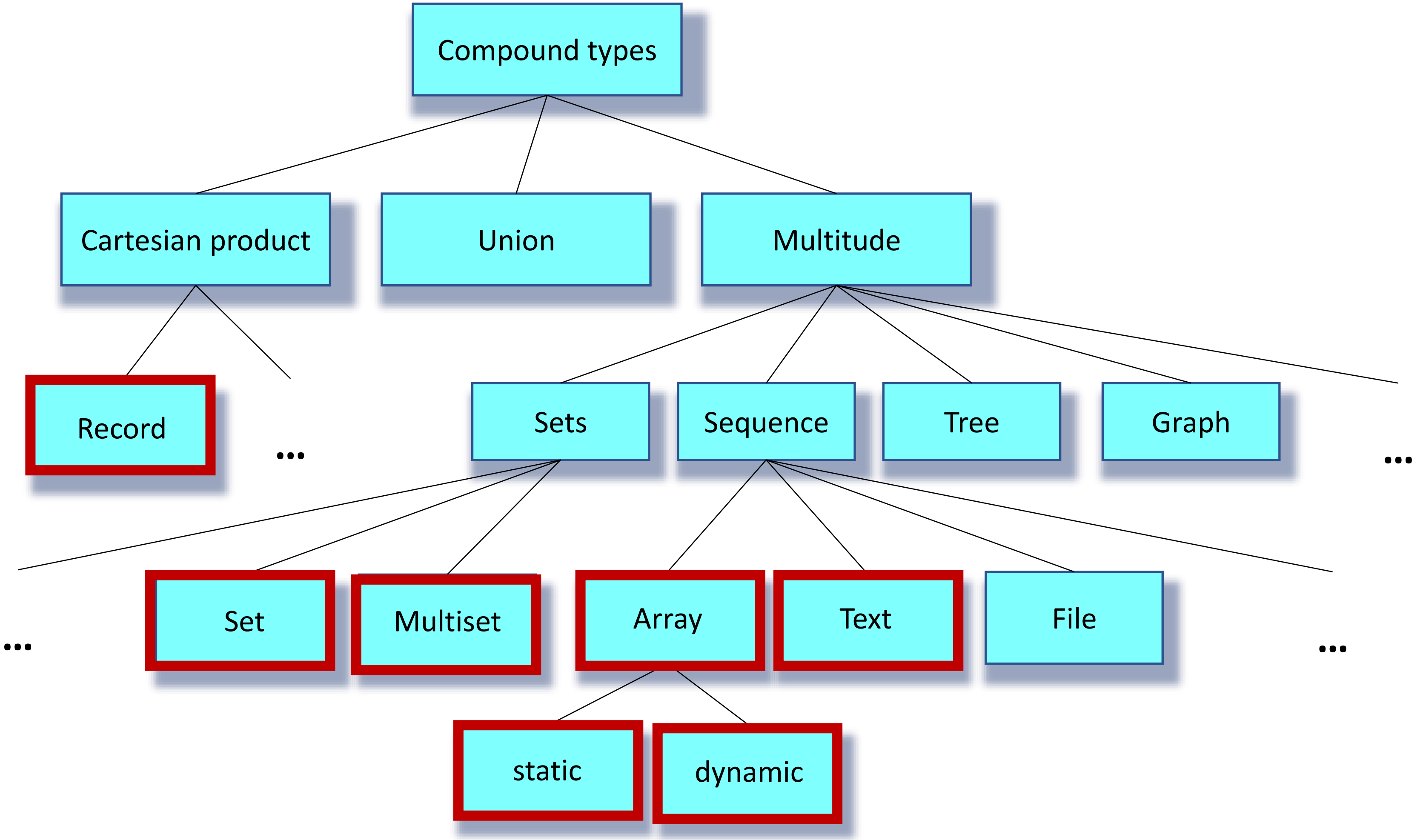
Applying the **Copy** PoA



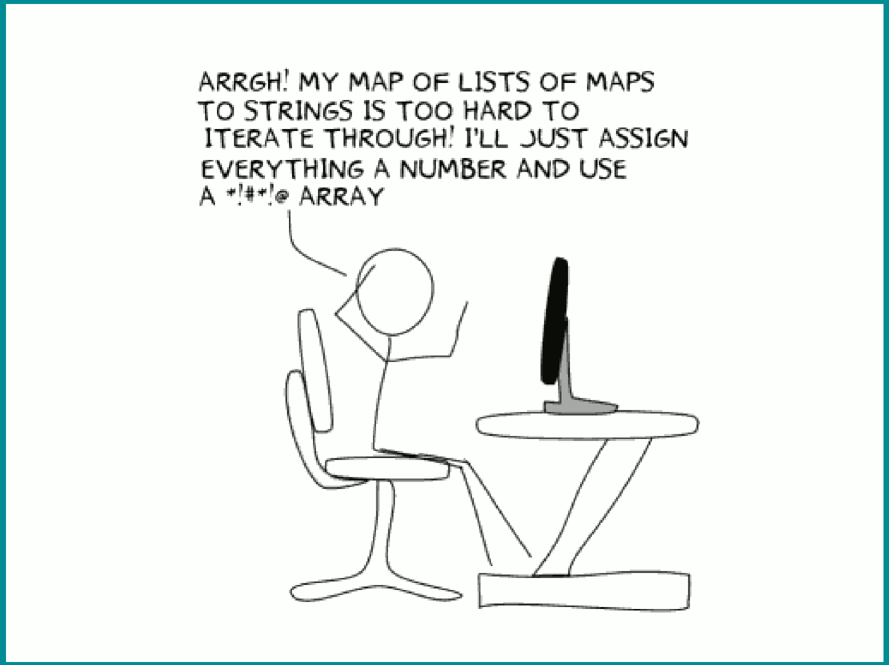
Calculation of the operation need:

The loop will run as many times as many elements there might be in the set – thus, the runtime is proportional to the cardinality of element type of the set.

Compound types



ELTE | FACULTY OF INFORMATICS



Thank you for your attention!