Building Cyclic Data in a Functional-Like Language Extended with Monotonic Objects

Alexei I. Adamovich[1], **Andrei V. Klimov**[2]

[1] Ailamazyan Program Systems Institute of RAS, Pereslavl-Zalessky

[2] Keldysh Institute of Applied Mathematics of RAS, Moscow

X Workshop “Program Semantics, Specification and Verification: Theory and Applications” (PSSV-2019)
July 1–2, 2019, Novosibirsk Akademgorodok, Russia

Part of General Problem

**Building Cyclic Data
in a Functional-Like Language
Extended with Monotonic Objects**

is a particular task within the following general problem

Construction of an intermediate model of computation between the functional and object-oriented programming paradigms, preserving the majority of nice properties of the functional one and extending the domain of its efficient applications

Hence the task

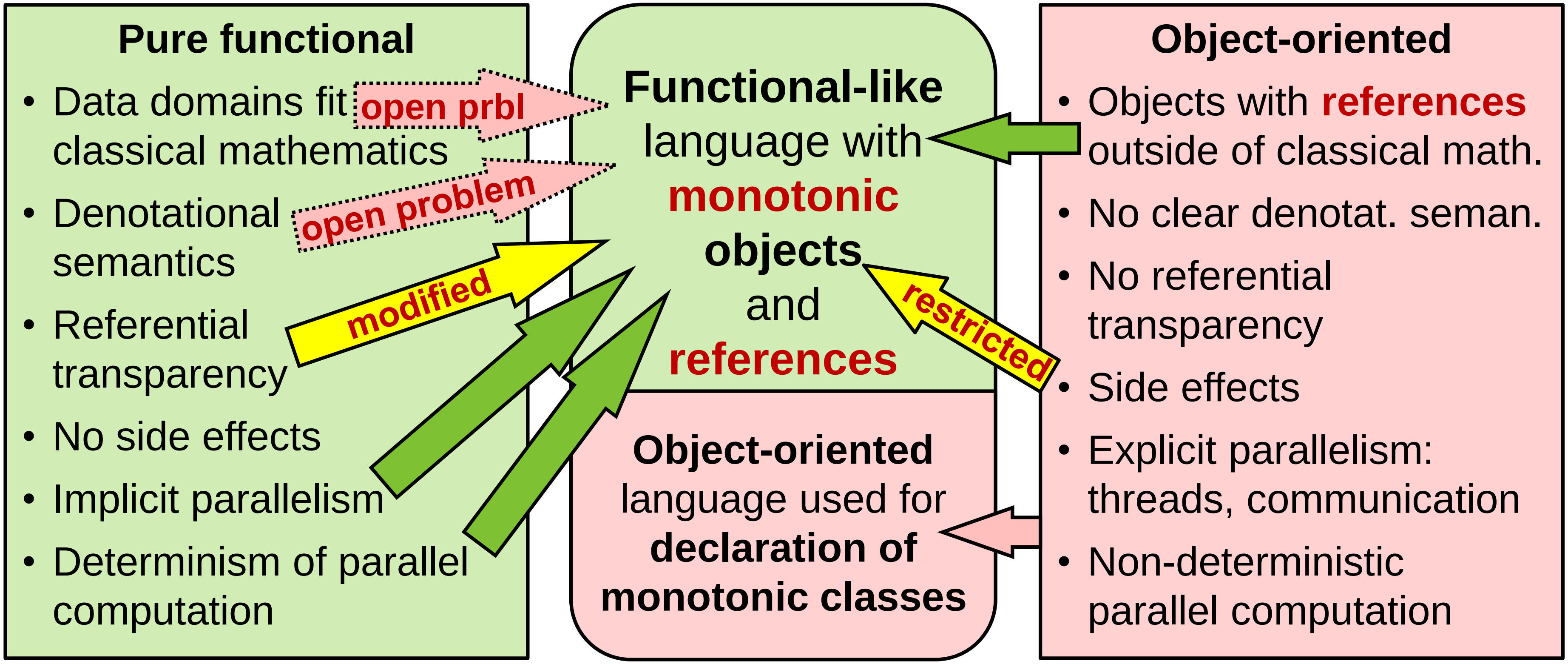
What is **the main limitation of functional** programing?

- **Functional** languages allow us to declare and efficiently manipulate **tree data only**, by composing terms of constructors
- **Object-oriented** languages allow us to efficiently represent and traverse **arbitrary graphs**, denoting relations between vertices and/or vertices and edges by **references** to objects

General Problem Statement

Building Cyclic Data
in a Functional-Like Language
Extended with Monotonic Objects

is a particular task within the following general problem



Pure functional

- Data domains fit classical mathematics
- Denotational semantics
- Referential transparency
- No side effects
- Implicit parallelism
- Determinism of parallel computation

open prbl

open problem

modified

Functional-like language with **monotonic objects** and **references**

restricted

Object-oriented language used for **declaration of monotonic classes**

Object-oriented

- Objects with **references** outside of classical math.
- No clear denotat. seman.
- No referential transparency
- Side effects
- Explicit parallelism: threads, communication
- Non-deterministic parallel computation

Monotonic Objects and Classes

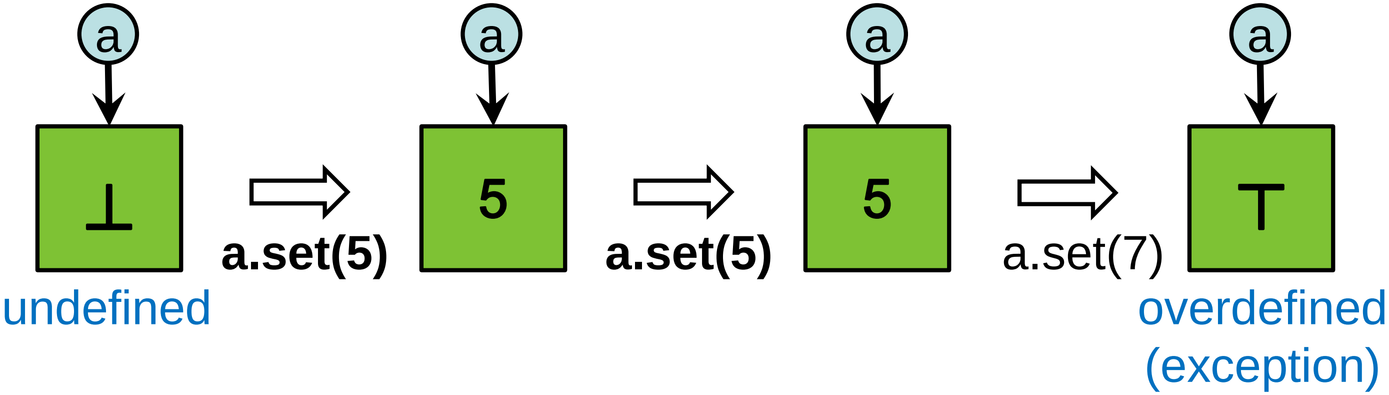
Definition. **Monotonic classes and objects** are such that **functional-like programs invoking methods on them** satisfy the following properties

- **Operational properties** (formal)
  - **Determinism** of parallel computation:
    - results obtained in different order of computation are **equivalent** (their difference is unobservable)
  - **Idempotency**:
    - recomputation of an expression produces an **equivalent result** and **side effect difference is unobservable**
- **Target properties** (still informal)
  - Existence of **semilattices** in which objects change **monotonically**
    - derived from declarations of monotonic classes
  - Existence of **denotational semantics**
    - generalizing that of pure functional languages
    - not using the order of computation (parallel, in essence)

Simplest Monotonic Class: Arvind’s I-Structure

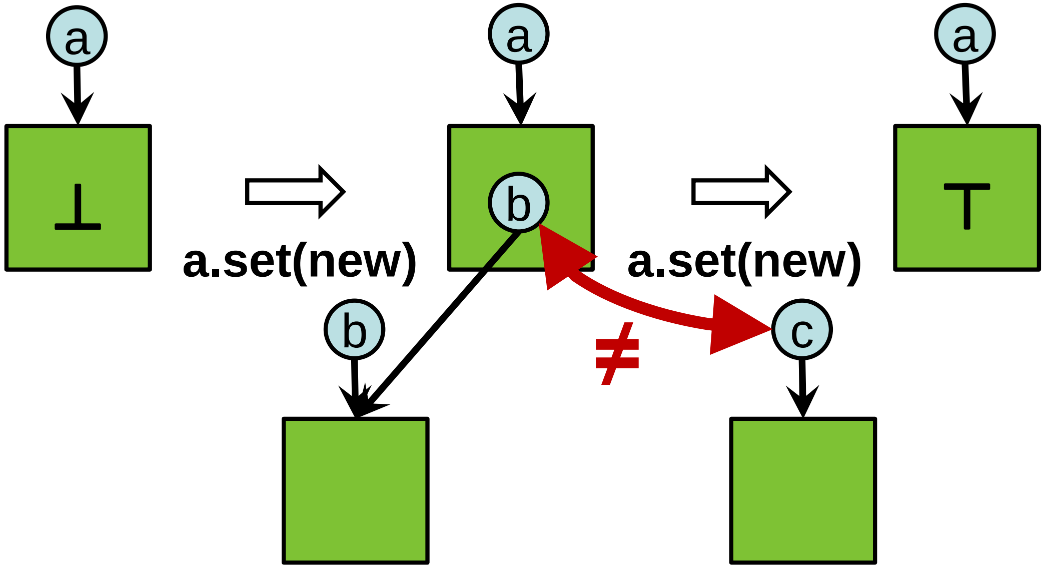
Monotonic with a primitive value

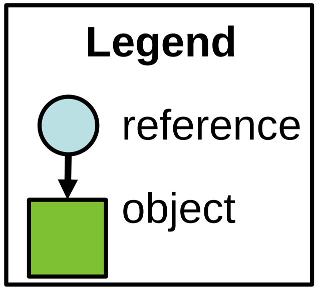
a = new IntVar



Non-monotonic with a reference value

a = new ObjectVar





Notice
repeated
a.set(5)
a.set(new)

Simplest Monotonic Class: Java Code

```
public class IntVar {

  boolean defined = false;
  int value;

  public synchronized int get() {
    if (!defined) wait();
    return value;
  }

  public synchronized void set(int x) {
    if (!defined) {
      value = x;
      defined = true;
      notifyAll();
    }
    else if (value != x)
      throw new RuntimeException();
  }
}
```

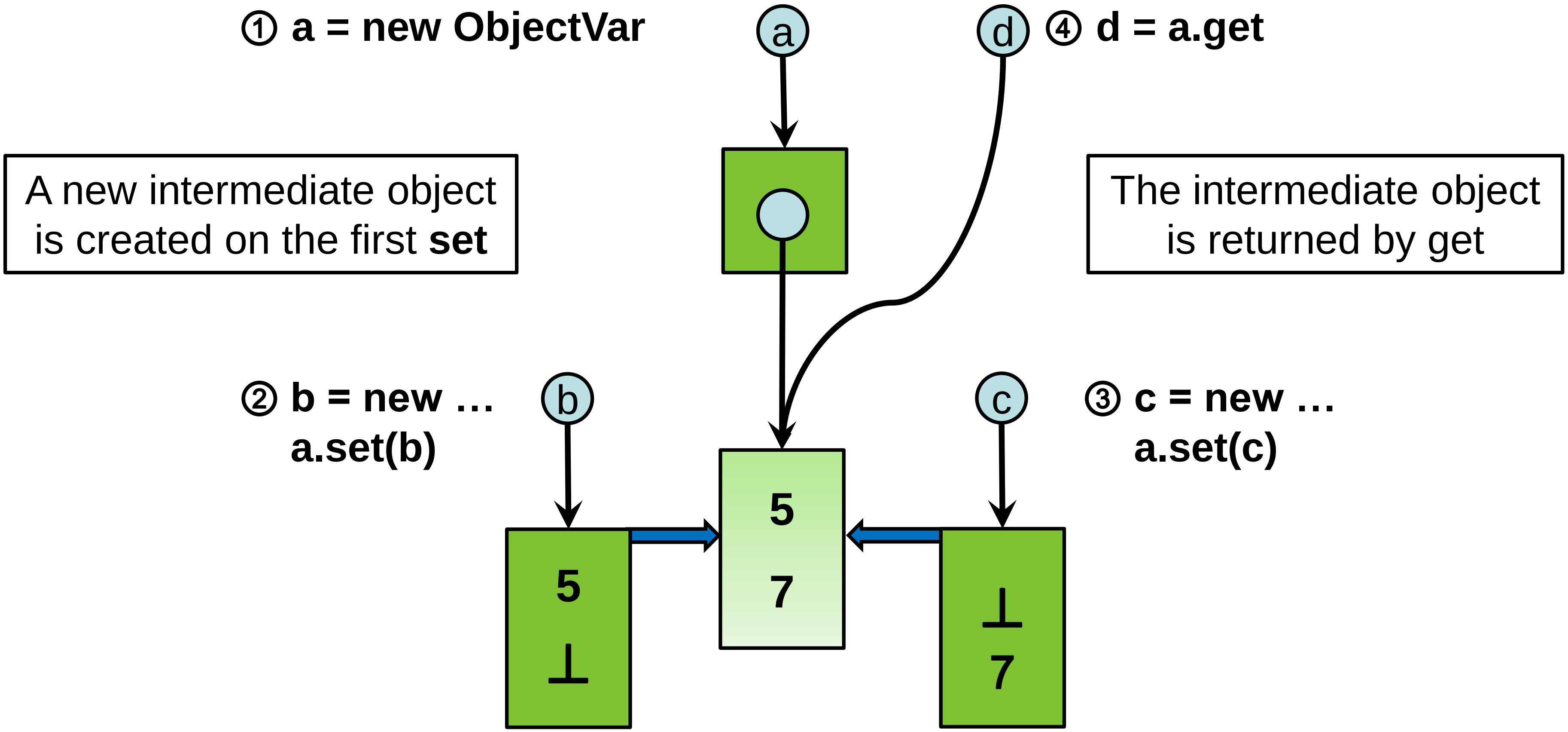
Monotonic class with a value of **primitive type int**

```
public class ObjectVar {

  boolean defined = false;
  Object value;

  public synchronized Object get() {
    if (!defined) wait();
    return value;
  }

  public synchronized void set(Object x) {
    if (!defined) {
      value = x;
      defined = true;
      notifyAll();
    }
    else if (value != x)
      throw new RuntimeException();
  }
}
```

Non-monotonic class with a value of **reference type Object**

Monotonic I-Structure with a Reference Value

Monotonic with a reference value



① **a = new ObjectVar**

④ **d = a.get**

A new intermediate object is created on the first **set**

The intermediate object is returned by get

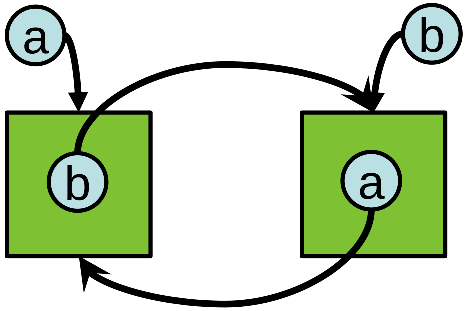
② **b = new …**
a.set(b)

③ **c = new …**
a.set(c)

Unification of the **set** objects

How to Monotonically Create a Cycle (1)

Our wish



```
a = new
b = new
a.set(b)
b.set(a)
```

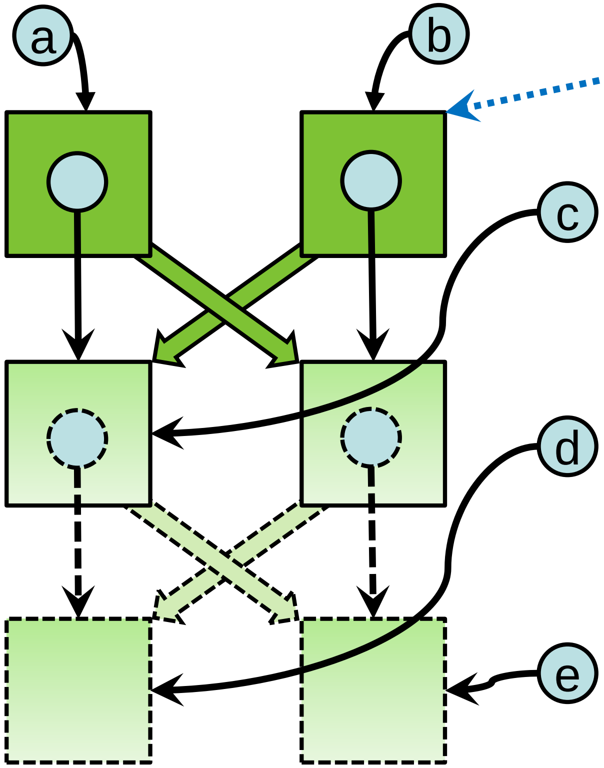
Easy in an object-oriented language

but it is non-monotonic

Monotonic 1st solution

(Example 4)

Solid-line objects and references are created by the above code



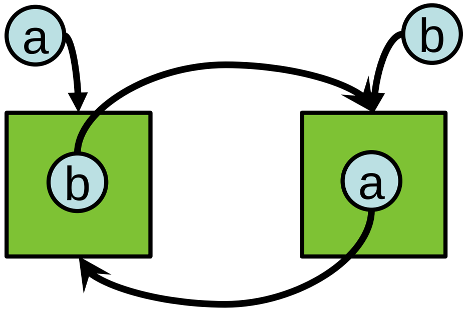
```
c = a.get
d = c.get
e = b.get.get
```

Undesirable unrolling of the cyclic graph

Can’t catch a cycle by comparing references while traversing

How to Monotonically Create a Cycle (2)

Our wish

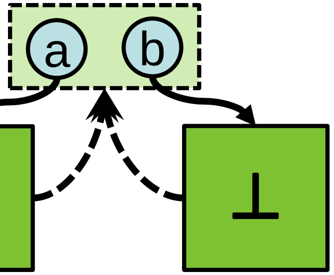


a = new
b = new
a.set(b)
b.set(a)

Easy in an object-oriented language
but it is non-monotonic

after creation

[a,b] =



Simultaneous creation of objects a **and** b

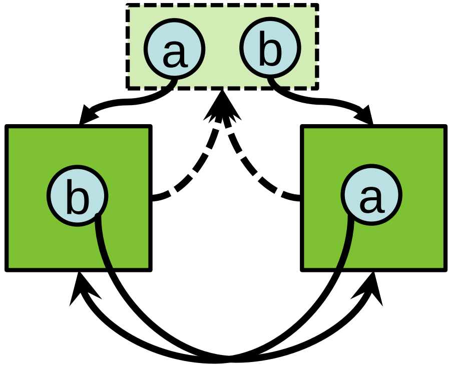
Monotonic
2nd solution

[a,b] = new[2]

Objects know references to each other from birth

(Example 5a)

after setting



a.set(b)
b.set(b)

In this case
plain setting as in OOL
does not violate
monotonicity

How to Monotonically Create a Cycle (3)

Monotonic 3rd solution

(Example 5b)

We know **one more solution*** to the **problem of building a cyclic data structure**, which is simpler to express as an algorithm than to draw a picture:

Algorithm

1. Consider building a cycling graph **as in solution 1**
2. Let us have **one object as the root**
3. Invoke on the root the **minimize** method that waits for all fields of the objects accessible from the root to **become defined** and then **merges the objects that are indistinguishable** by all operations
4. The operation **minimize** returns a **fresh reference to the root** such that all **objects accessible from it** get a fresh **unique reference** for each

In the 2nd and 3rd solutions catching cycles is possible like in a plain object-oriented language

*There may be other solutions which we don’t know yet

Conclusion

- We have made first steps to construction of a **new model of computation** intermediate **between functional and object-oriented paradigms**

- A programming language that implements the model is **two-level**:
  - The **higher** level is a **functional-like** language
  - The **lower** level is a common **object-oriented** language

- The **main idea** is to **restrict the methods** of the classes (called **monotonic**) that are **used in the functional language** in such a way that the following **properties of functional programs** are **preserved**:
  - **Determinism** of parallel computation
  - **Idempotency** of side effects and results

- The key initial problem to be solved is **overcoming the main limitation of functional languages** that **only tree-like structures** can be constructed

- We have demonstrated **examples of monotonic classes**, by using which a **functional program can build cyclic structures** where **each object has a programmatically accessible unique reference**