TOPL

Radu Grigore, Rasmus Lerchedahl Petersen, Dino Distefano

October 23, 2011

```
1  import java.util.*;
2
3  public class A {
4    public static void main(String[] args) {
5      Collection<Integer> c = new ArrayList<Integer>();
6      c.add(1);
7      c.add(2);
8      Iterator<Integer> i = c.iterator();
9      Iterator<Integer> j = c.iterator();
10     i.next();
11     i.remove();
12     System.out.println(j.next());
13   }
14 }
```

Line 12 throws `ConcurrentModificationException`.

```
1  E next() {
2    if (expectedModCount != modCount)
3      throw new ConcurrentModificationException();
4    int i = lastRet + 1;
5    if (i >= size) throw new NoSuchElementException();
6    Object[] elementData = ArrayList.this.elementData;
7    if (i >= elementData.length)
8      throw new ConcurrentModificationException();
9    return (E) elementData[lastRet = i];
10 }
```

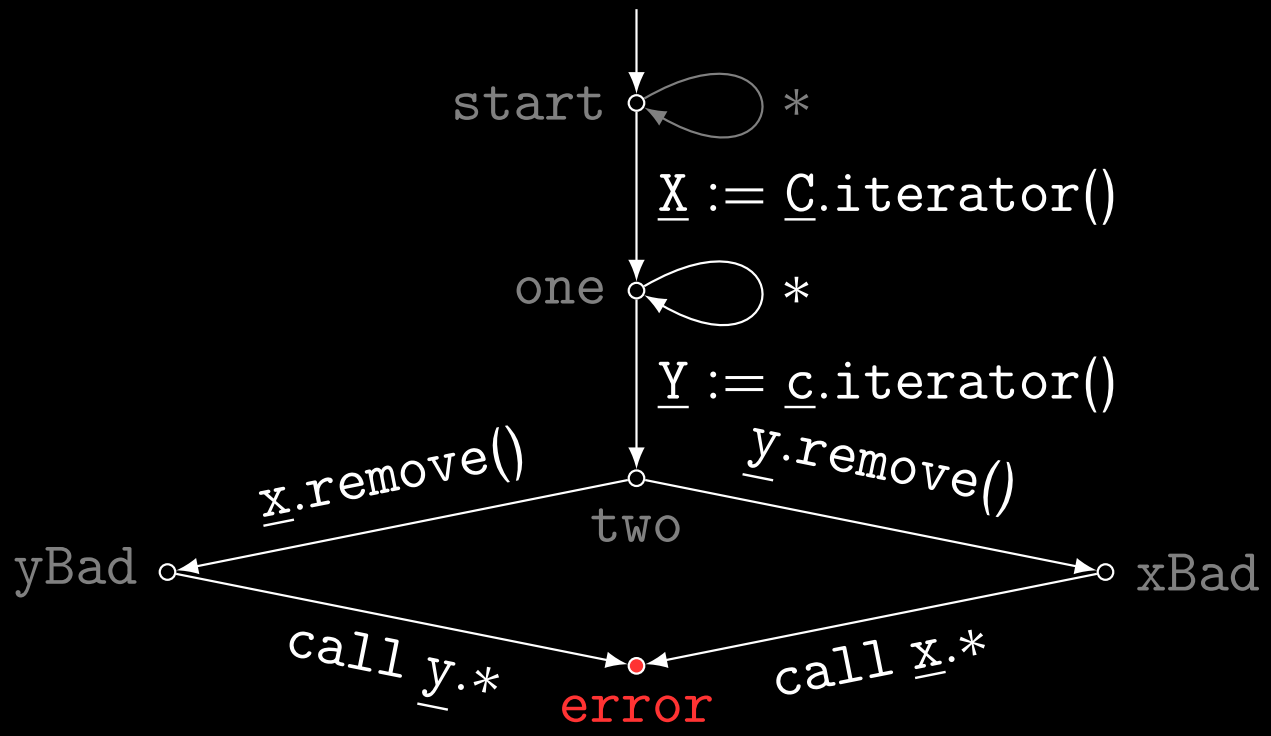
Goal

A high-level language to describe such properties.
Possible uses:

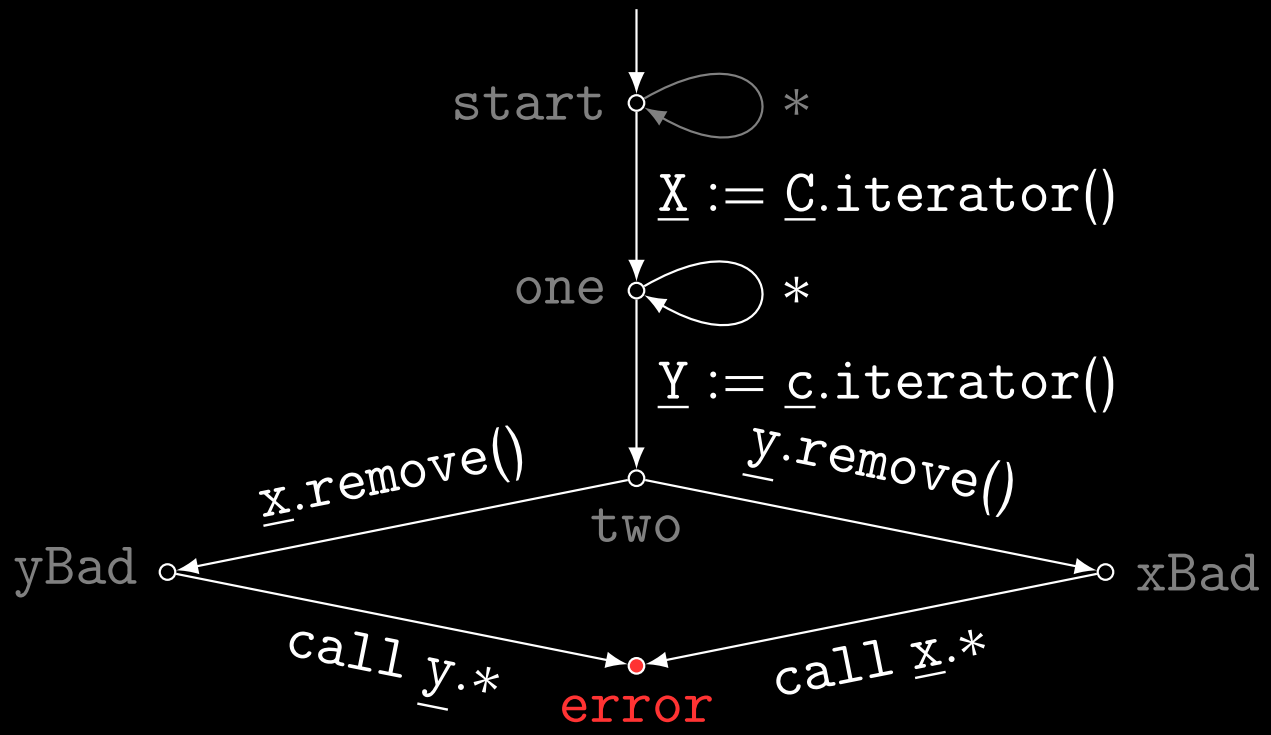
- dynamic analysis: synthesize the (run-time) checks
- static analysis:
  - ensure that programmer's run-time checks are enough
  - ensure that no dynamic check is needed

A Bit More Specific . . .

> *From specifying APIs we notice several recurring patterns including the importance of*
>
> 1. *dynamic state tests,*
> 2. *method cases, and*
> 3. *the dependency of API objects on each other.*
>
> — Bierhoff, Beckman, Aldrich, 2009



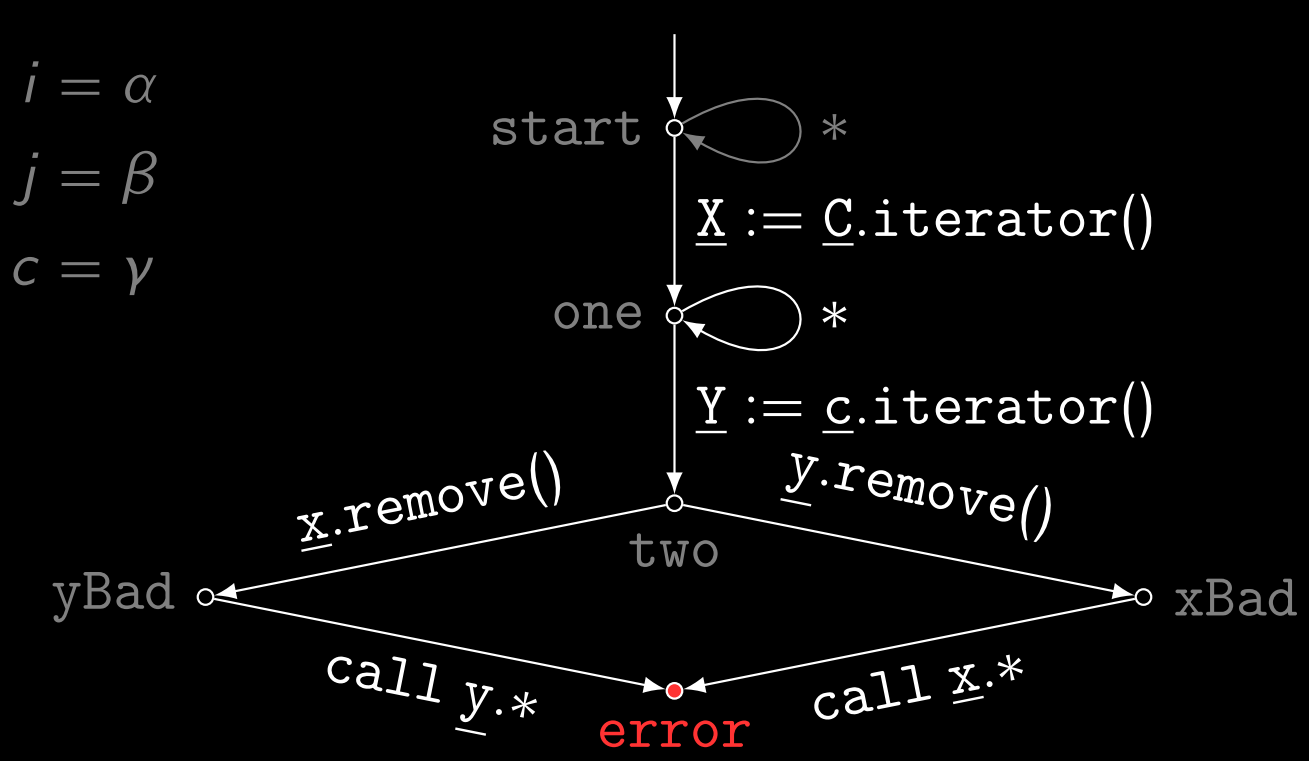
Given two iterators x and y for the same collection c, if one of them removes the element it points to then the other becomes invalid.



$\{(\texttt{start}, [])\}$ (1)

```
i = c.iterator()
```
(2)

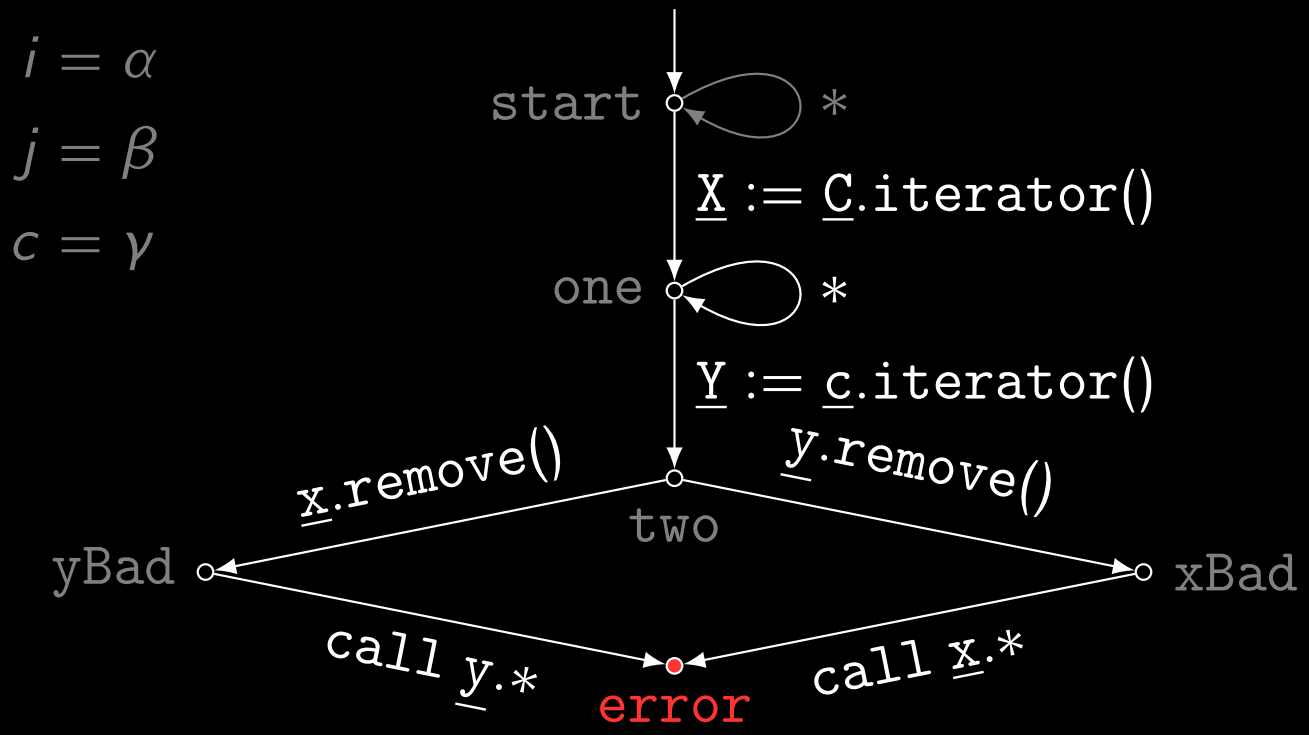
$\{(\texttt{start}, []), (\texttt{one}, [c : \gamma, x : \alpha])\}$ (3)



$$\{(\texttt{start}, []),\ (\texttt{one}, [c : \gamma,\ x : \alpha])\} \tag{4}$$

```
j = c.iterator()
```
(5)

$$\left\{ \begin{array}{l} (\texttt{start}, []),\ (\texttt{one}, [c : \gamma,\ x : \alpha]), \\ (\texttt{one}, [c : \gamma,\ x : \beta]),\ (\texttt{two}, [c : \gamma,\ x : \alpha,\ y : \beta]) \end{array} \right\} \tag{6}$$

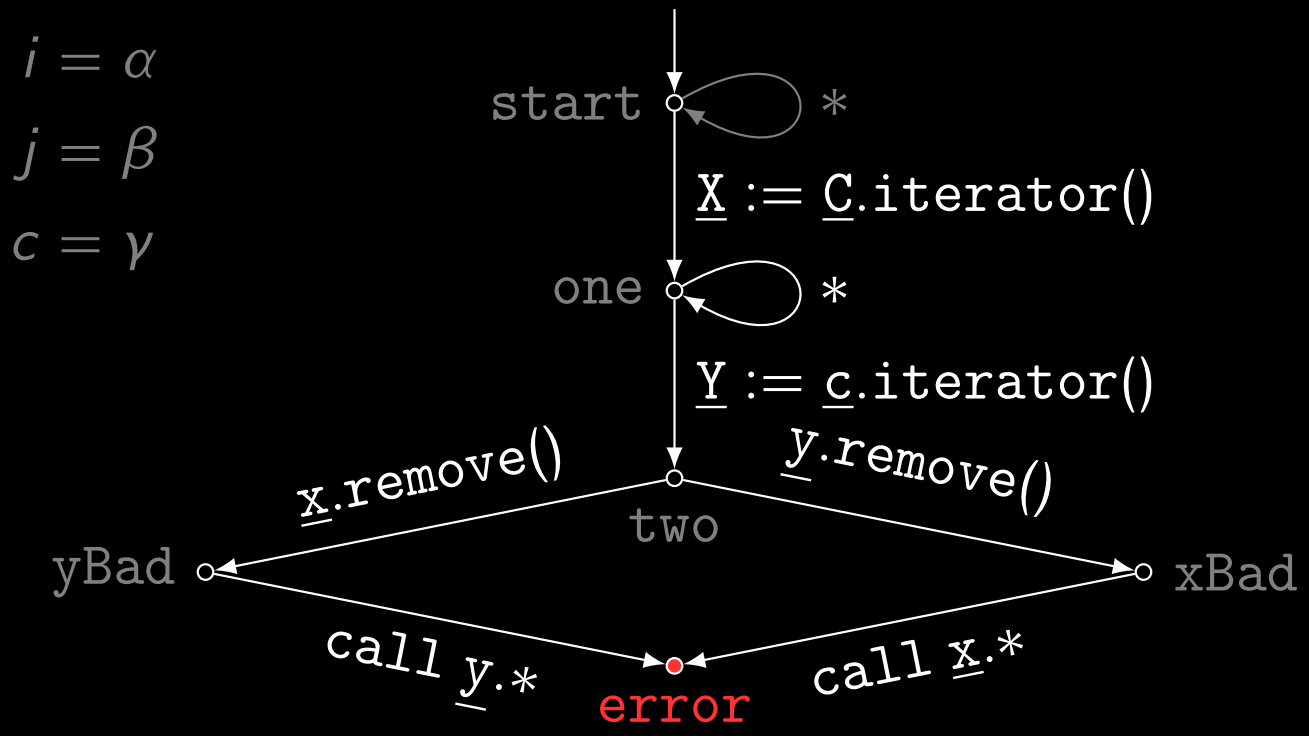


$$\left\{ \begin{array}{l} (\texttt{start}, [\,]),\ (\texttt{one}, [c : \gamma,\ x : \alpha]), \\ (\texttt{one}, [c : \gamma,\ x : \beta]),\ (\texttt{two}, [c : \gamma,\ x : \alpha,\ y : \beta]) \end{array} \right\} \tag{7}$$

```
i.next()
```

(8)

$$\left\{ \begin{array}{l} (\texttt{start}, [\,]),\ (\texttt{one}, [c : \gamma,\ x : \alpha]), \\ (\texttt{one}, [c : \gamma,\ x : \beta]),\ (\texttt{two}, [c : \gamma,\ x : \alpha,\ y : \beta]) \end{array} \right\} \tag{9}$$

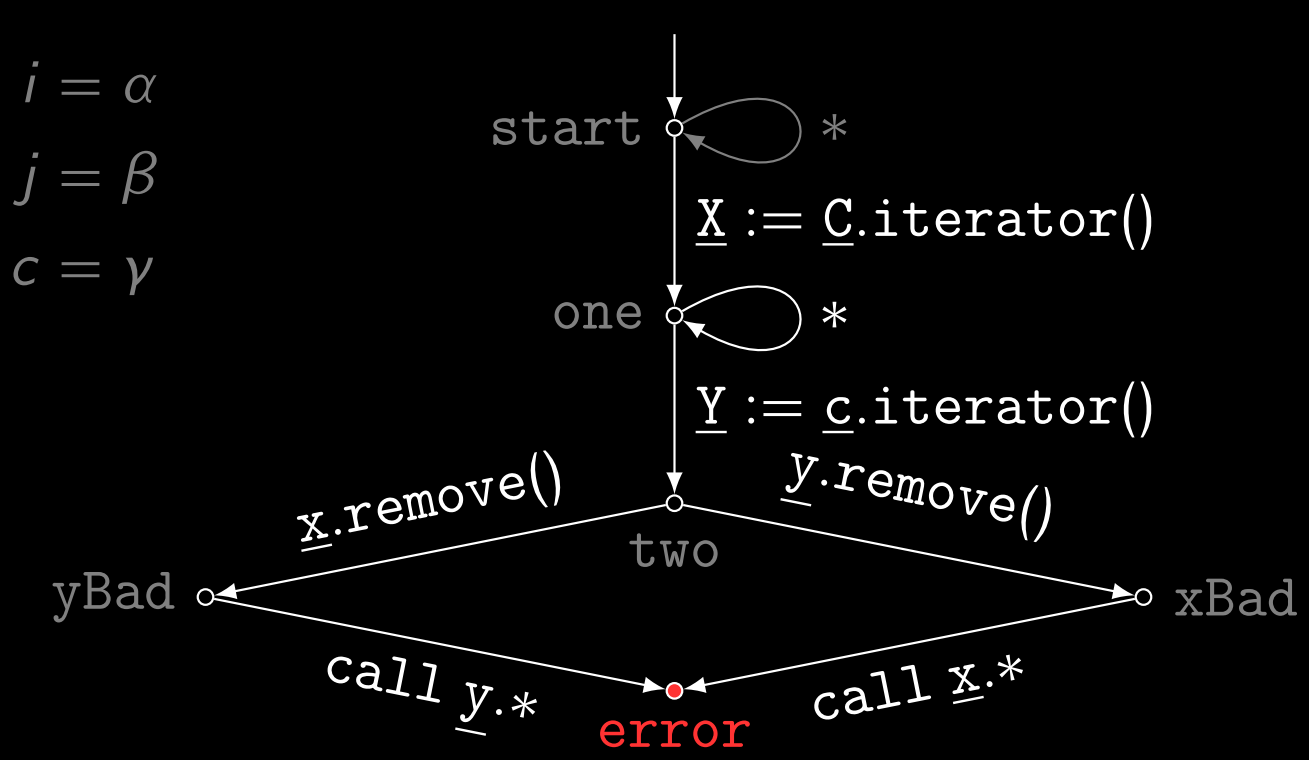


$$\left\{ \begin{array}{l} (\texttt{start}, [\,]),\ (\texttt{one}, [c : \gamma,\ x : \alpha]), \\ (\texttt{one}, [c : \gamma,\ x : \beta]),\ (\texttt{two}, [c : \gamma,\ x : \alpha,\ y : \beta]) \end{array} \right\} \tag{10}$$

```
i.remove()
```

(11)

$$\left\{ \begin{array}{l} (\texttt{start}, [\,]),\ (\texttt{one}, [c : \gamma,\ x : \alpha]), \\ (\texttt{one}, [c : \gamma,\ x : \beta]),\ (\texttt{yBad}, [c : \gamma,\ x : \alpha,\ y : \beta]) \end{array} \right\} \tag{12}$$

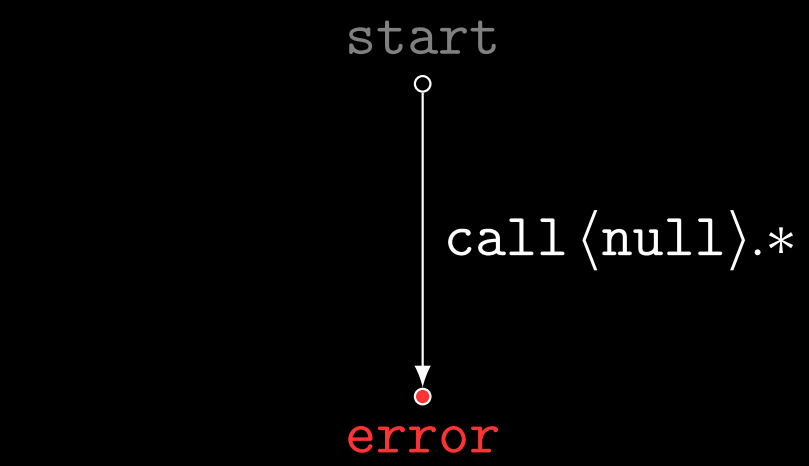


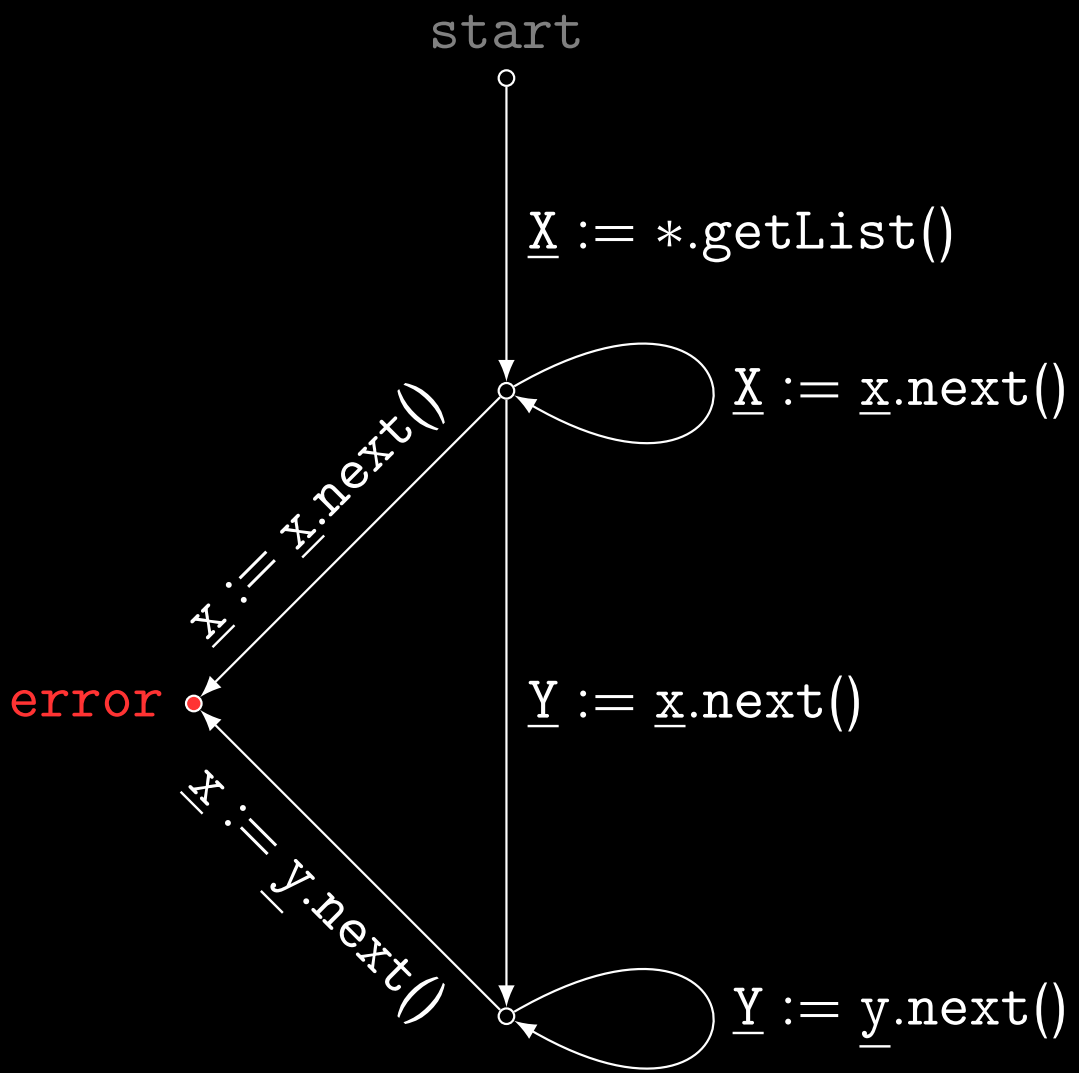
$$\left\{ \begin{array}{l} (\texttt{start}, [\,]),\ (\texttt{one}, [c : \gamma,\ x : \alpha]), \\ (\texttt{one}, [c : \gamma,\ x : \beta]),\ (\texttt{yBad}, [c : \gamma,\ x : \alpha,\ y : \beta]) \end{array} \right\} \tag{13}$$

```
j.next()
```

(14)

$$\left\{ \begin{array}{l} (\texttt{start}, [\,]),\ (\texttt{one}, [c : \gamma,\ x : \alpha]), \\ (\texttt{one}, [c : \gamma,\ x : \beta]),\ (\texttt{error}, [c : \gamma,\ x : \alpha,\ y : \beta]) \end{array} \right\} \tag{15}$$


start
call ⟨null⟩.*
error


start
X := *.getList()
X := x.next()
x := x.next()
error
Y := x.next()
x := y.next()
Y := y.next()

$$\mathbf{Trace} = [\mathbf{Event}] \tag{16}$$

$$\mathbf{Event} = \mathbf{Tag} \times [\mathbf{Value}] \tag{17}$$

$$\mathit{property} = (\mathbf{Vertex}, \mathbf{Transition}) \tag{18}$$

$$\mathbf{Transition} \subseteq \mathbf{Vertex} \times \mathbf{Label} \times \mathbf{Vertex} \tag{19}$$

$$\mathbf{Label} = [\mathbf{Guard} \times \mathbf{Action}] \tag{20}$$

$$\mathbf{Guard} = \mathbf{Event} \times \mathbf{Store} \to \mathbb{B} \tag{21}$$

$$\mathbf{Action} = \mathbf{Event} \times \mathbf{Store} \to \mathbf{Store} \tag{22}$$

$$\mathbf{Store} = \mathbf{Variable} \to \mathbf{Value}_{\perp} \tag{23}$$

$$\mathbf{State} = \mathbf{Vertex} \times \mathbf{Store} \times \mathbf{Trace} \tag{24}$$

$$\rightsquigarrow \subseteq \mathbf{State} \times \mathbf{State} \tag{25}$$

$$exec \in \mathbf{Store} \times \mathbf{Trace} \times \mathbf{Label} \to \mathbf{Store}_{\perp} \tag{26}$$

$$exec(\sigma, [], []) = \sigma \tag{27}$$

$$exec(\sigma, e{:}es, (g, a){:}gas) = exec(a(e, \sigma), es, gas) \qquad \text{if } g(e, \sigma) \tag{28}$$

$$\frac{exec(\sigma, es_1, l) \neq \perp \qquad (x, l, y) \in \mathbf{Transition}}{(x, \sigma, es_1 \cdot es_2) \rightsquigarrow (y, exec(\sigma, es_1, l), es_2)} \tag{29}$$

$$\frac{\neg enabled(x, \sigma, e{:}es)}{(x, \sigma, e{:}es) \rightsquigarrow (x, \sigma, es)} \tag{30}$$

$$\frac{exec(\sigma, es_1, l) \neq \perp \qquad (x, l, y) \in \mathbf{Transition}}{enabled(x, \sigma, es_1 \cdot es_2)} \tag{31}$$

- done
  - interpreter for toy language (SOOL)
  - dynamic checking—Java bytecode instrumentation
- next
  - experiments with dynamic checking
  - abstraction for dynamic-checking
  - static checking (with jStar)
    - are checks enough?
    - is the client OK, so checks can be removed?