

---

monitoring with  
state machines &  
regular expressions

CS 119

the propositional case

---

recall this property

Enumeration (Java 2 Platform SE 5.0)

http://java.sun.com/j2se/1.5.0/docs/api/

Java™ 2 Platform Standard Ed. 5.0

Overview Package **Class** Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.util

## Interface Enumeration<E>

All Known Subinterfaces:  
[NamingEnumeration<T>](#)

All Known Implementing Classes:  
[StringTokenizer](#)

**R<sub>2</sub>: An enumeration should not be propagated after the underlying vector has been changed .**

### Method Summary

boolean	<a href="#">hasMoreElements()</a>	Tests if this enumeration contains more elements.
E	<a href="#">nextElement()</a>	Returns the next element of this enumeration if this enumeration object has at least one more element to provide.

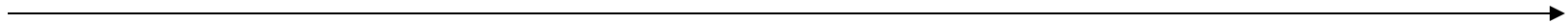
Find: 97 Next Previous Highlight all Match case

Downloads

---

example  
monitored  
run

```
...  
v.add(3);  
Enumeration en = v.elements();  
while(en.hasMoreElements()) {  
    Integer i = (Integer)en.nextElement();  
    if (i == 2)  
        v.add(4);  
    else  
        System.out.println(i);  
}  
}
```



---

example  
monitored  
run

→

```
...  
v.add(3);  
Enumeration en = v.elements();  
while(en.hasMoreElements()) {  
    Integer i = (Integer)en.nextElement();  
    if (i == 2)  
        v.add(4);  
    else  
        System.out.println(i);  
}  
}
```

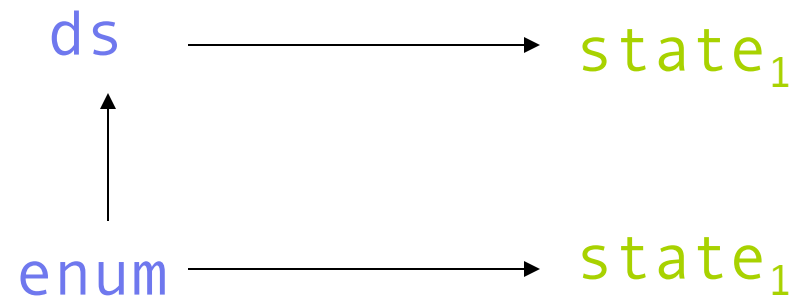
ds → state<sub>1</sub>

→

update

example  
monitored  
run

```
...  
v.add(3);  
Enumeration en = v.elements();  
while(en.hasMoreElements()) {  
    Integer i = (Integer)en.nextElement();  
    if (i == 2)  
        v.add(4);  
    else  
        System.out.println(i);  
}  
}
```



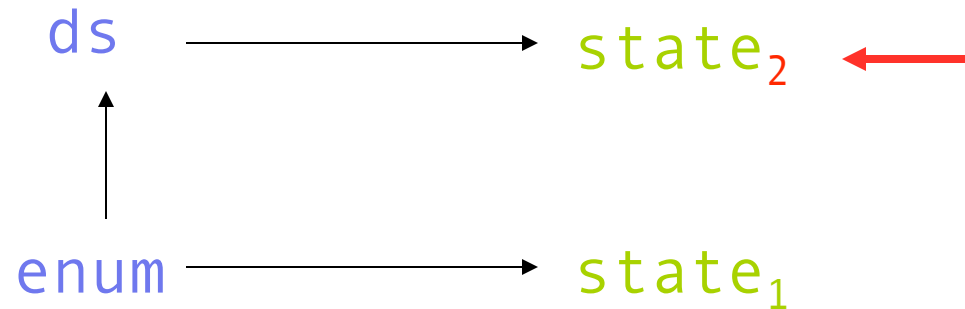
update

create

example  
monitored  
run



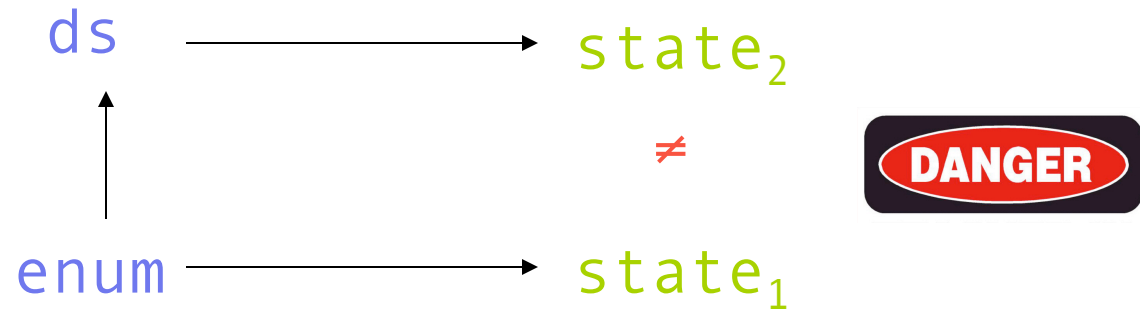
```
...  
v.add(3);  
Enumeration en = v.elements();  
while(en.hasMoreElements()) {  
    Integer i = (Integer)en.nextElement();  
    if (i == 2)  
        v.add(4);  
    else  
        System.out.println(i);  
}  
}
```



example  
monitored  
run



```
...  
v.add(3);  
Enumeration en = v.elements();  
while(en.hasMoreElements()) {  
    Integer i = (Integer)en.nextElement();  
    if (i == 2)  
        v.add(4);  
    else  
        System.out.println(i);  
}  
}
```



recall  
AspectJ  
solution

```
aspect SafeEnum {
    private Map ds_state = new WeakIdentityHashMap();
    private Map enum_state = new WeakIdentityHashMap();
    private Map enum_ds = new WeakIdentityHashMap();

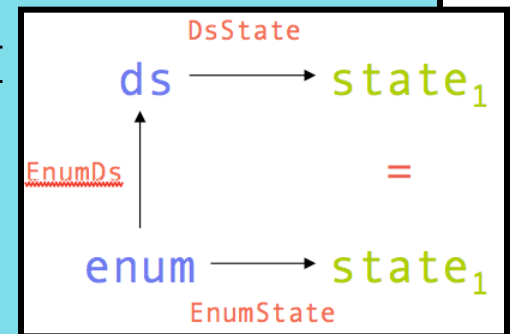
    private static class StateId {}

    pointcut vector_update() :
        call(* Vector.add*(..)) || call(* Vector.clear()) ||
        call(* Vector.insertElementAt(..)) || call(* Vector.remove*(..)) ||
        call(* Vector.retainAll(..)) || call(* Vector.set*(..) && scope());

    after(Vector ds) returning(Enumeration e) :
        call(Enumeration Vector.elements()) && target(ds) {
        enum_ds.put(e, ds);
        Object s = ds_state.get(ds);
        if (s != null) enum_state.put(e, s);
    }

    before(Enumeration e):
        call(Object Enumeration.nextElement()) && target(e) {
        if (ds_state.get(enum_ds.get(e)) != enum_state.get(e))
            error("nextElement called on enumerator after update");
        }

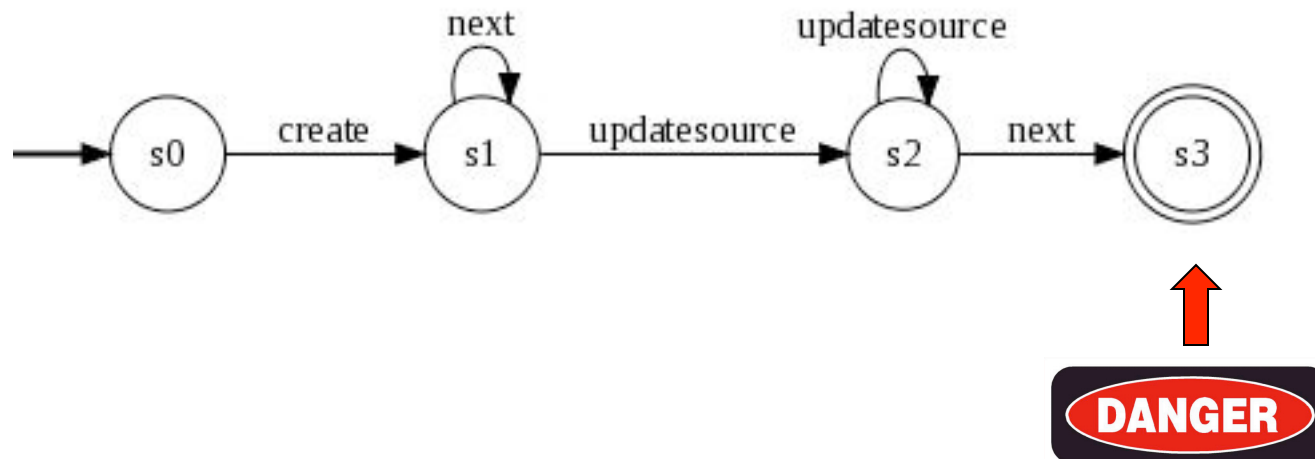
    after(Vector ds) : vector_update() && target(ds) {
        ds_state.put(ds, new StateId());
    }
}
```



some  
programming  
required!

---

# state machines



# systems



JavaMOP  
this course

MANCHESTER  
1824

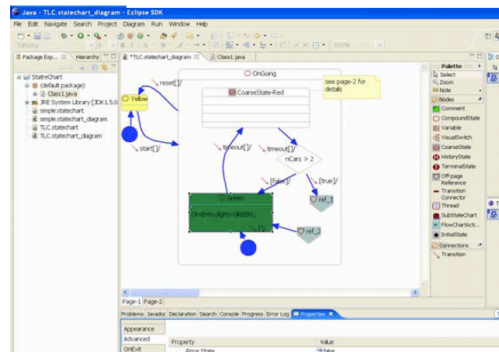
JPL

RuLeR  
this course

commercial



First in Temporal Solutions



*The StateRover graphical programming environment supports statecharts, flowcharts, and advanced software design constructs for representing, building, and verifying complex software modules and classes.*

Software module specification

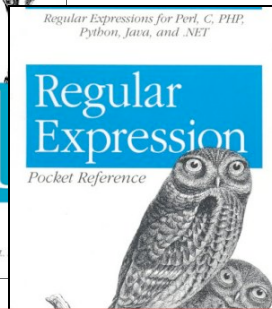
The StateRover provides a broad range of features that enable users to specify, generate, and verify the desired behavior of their software.

Features

- ❑ Eclipse graphical user interface for mixed UML statechart and flowchart models and specifications.
- ❑ Automatic C, C++, and Java code generation.
- ❑ Simultaneous diagram and source level debugging.
- ❑ Run-time verification of behavioral requirements.

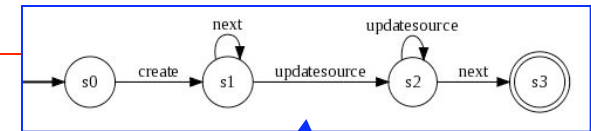
Benefits

- ❑ Intuitive graphical programming tool with automatic modular and portable code generation



- . Matches any single character (many applications exclude newlines, and exactly which characters are considered newlines is flavor, character encoding, and platform specific, but it is safe to assume that the line feed character is included). Within POSIX bracket expressions, the dot character matches a literal dot. For example, `a.c` matches `"abc"`, etc., but `[a.c]` matches only `"a"`, `"."`, or `"c"`.
- [ ] A bracket expression. Matches a single character that is contained within the brackets. For example, `[abc]` matches `"a"`, `"b"`, or `"c"`. `[a-z]` specifies a range which matches any lowercase letter from `"a"` to `"z"`. These forms can be mixed: `[abcx-z]` matches `"a"`, `"b"`, `"c"`, `"x"`, `"y"`, and `"z"`, as does `[a-cx-z]`.  
The `-` character is treated as a literal character if it is the last or the first character within the brackets, or if it is escaped with a backslash: `[abc-]`, `[-abc]`, or `[a\-bc]`.
- [^ ] Matches a single character that is not contained within the brackets. For example, `[^abc]` matches any character other than `"a"`, `"b"`, or `"c"`. `[^a-z]` matches any single character that is not a lowercase letter from `"a"` to `"z"`. As above, literal characters and ranges can be mixed.
- ^ Matches the starting position within the string. In line-based tools, it matches the starting position of any line.
- \$ Matches the ending position of the string or the position just before a string-ending newline. In line-based tools, it matches the ending position of any line.
- \( \) Defines a marked subexpression. The string matched within the parentheses can be recalled later (see the next entry, `\n`). A marked subexpression is also called a block or capturing group.
- \n Matches what the `n`th marked subexpression matched, where `n` is a digit from 1 to 9. This construct is theoretically **irregular** and was not adopted in the POSIX ERE syntax. Some tools allow referencing more than nine capturing groups.
- \* Matches the preceding element zero or more times. For example, `ab*c` matches `"ac"`, `"abc"`, `"abbbc"`, etc. `[xyz]*` matches `"`, `"x"`, `"y"`, `"z"`, `"zx"`, `"zyx"`, `"xyzz"`, and so on. `\(ab\)*` matches `"`, `"ab"`, `"abab"`, `"ababab"`, and so on.
- \{m,n\} Matches the preceding element at least `m` and not more than `n` times. For example, `a\{3,5\}` matches only `"aaa"`, `"aaaa"`, and `"aaaaa"`. This is not found in a few, older instances of regular expressions.

# systems



```
Safeliterator(Collection c, Iterator i) {  
  
    event create after(Collection c) returning(Iterator i) :  
        call(Iterator Collection+.iterator()) && target(c) && BaseAspect.notwithin(){}  
  
    event updatesource after(Collection c) :  
        (call(* Collection+.remove*(..)) || call(* Collection+.add*(..)) ) && target(c) && BaseAspect.notwithin(){}  
  
    event next before(Iterator i) :  
        call(* Iterator.next()) && target(i) && BaseAspect.notwithin(){}  
  
    are : create next* updatesource updatesource* next  
  
    @validation {  
        System.out.println("improper iterator usage");  
    }  
}
```

this course  
using AspectJ

JavaMOP :

<http://fsl.cs.uiuc.edu/index.php/Special:JavaMOP2.0>

```
public aspect FailSafeEnum {  
  
    pointcut vector_update() :  
        call(* Vector.add*(..)) ||  
        call(* Vector.clear()) ||  
        call(* Vector.insertElementAt(..)) ||  
        call(* Vector.remove*(..)) ||  
        call(* Vector.retainAll(..)) ||  
        call(* Vector.set*(..));  
  
    tracematch(Vector ds, Enumeration e) {  
        sym create_enum after returning(e) : ((call(* Vector+.elements()) && target(ds)) || (call(Enumeration+.new(..)) && args(ds)));  
        sym call_next before : call(Object Enumeration.nextElement()) && target(e);  
        sym update_source before : vector_update() && target(ds);  
  
        create_enum call_next* update_source+ call_next  
    {  
        if(System.getProperty("TMTEST_ACTIVE")!=null) {  
            System.err.println("##### Tracematch "+getClass().getName()+" matched!");  
        }  
    }  
}
```

extending AspectJ

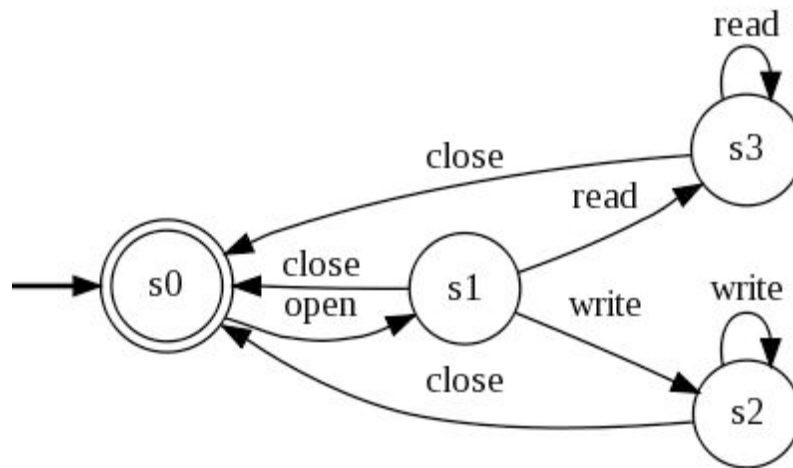
Tracematches :

<http://abc.comlab.ox.ac.uk/papers>

# regular expression

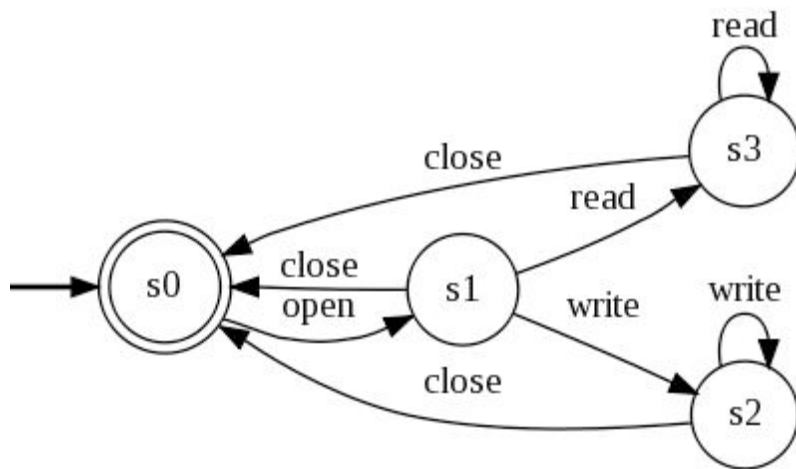
- **file access property**: a file should be opened, uniformly accessed (either only read from or only written to) zero or more times, and then closed:

$(\text{open } (\text{read}^* + \text{write}^*) \text{close})^*$



only propositional specifications in this lecture: no data

# state machine



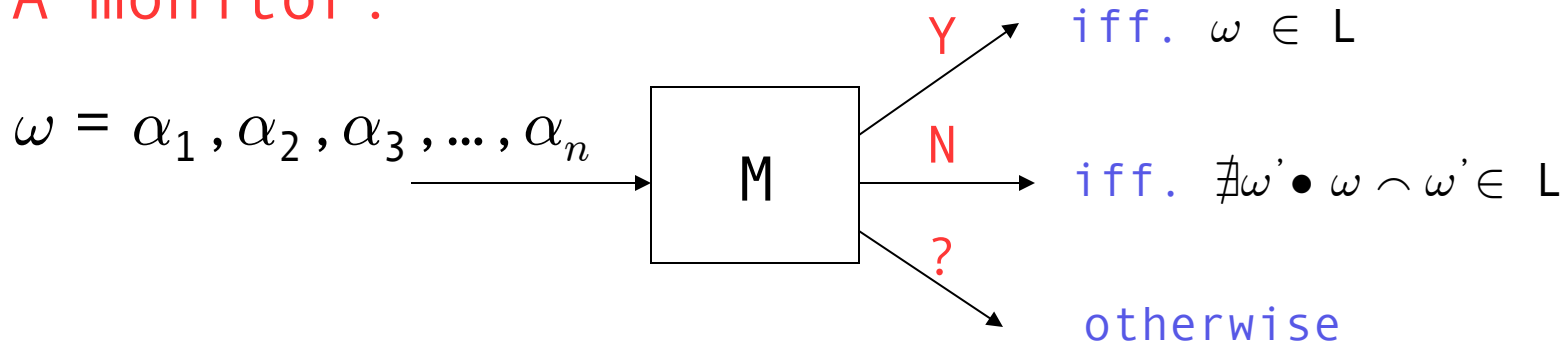
```
fsm:  
!s0[  
  open -> s1  
]  
S1[  
  read -> s3  
  write -> s2  
  close -> s0  
]  
S2[  
  write -> s2  
  close -> s0  
]  
s3[  
  read -> s3  
  close -> s0  
]
```

# recalling formal setting

## Property:

A property  $P$  over  $A$  is a language:  $P \subseteq A^*$

## A monitor:



but languages  
are usually infinite

## The monitoring problem:

Given logic  $\mathcal{L}$ , and formula  $\varphi \in \text{Formulas}(\mathcal{L})$ ,  
construct monitor for the language of  $\varphi$ :  $M_{\mathcal{L}(\varphi)}$

---

# from regular expressions to monitors via state machines

regular expression (RE)

→ step 1

deterministic finite state automaton (DFA)

→ step 2

monitor (M)

---

## Step 2 : DFA $\rightarrow$ M

Given an automaton (DFA):

$$\mathcal{A} = \{Q, i \in Q, F \subseteq Q, \sigma: Q \times A \rightarrow Q_{\perp}\}$$

$Q$  : set of states

$i$  : the initial state

$F$  : set of final states

$\sigma$  : partial transition function

---

## Step 2 : DFA $\rightarrow$ M

Given an automaton (DFA):

$$\mathcal{A} = \{Q, i \in Q, F \subseteq Q, \sigma: Q \times A \rightarrow Q_{\perp}\}$$

Extend  $\sigma$  to  $A^*$  as follows:

$$\sigma^* : Q \times A^* \rightarrow Q_{\perp}$$

$$\sigma^*(q, \epsilon) = q$$

$$\sigma^*(q, \alpha) = \sigma(\alpha)$$

$$\sigma^*(q, \omega\alpha) =$$

$$\text{if } \sigma^*(q, \omega) \neq \perp \text{ then } \sigma(\sigma^*(q, \omega), \alpha) \text{ else } \perp$$

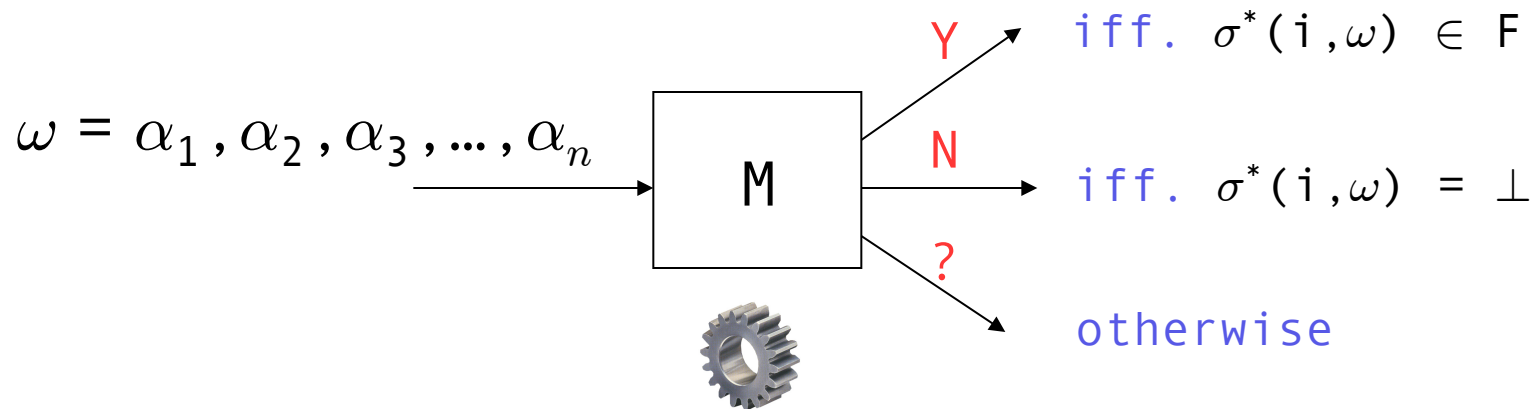
---

## Step 2 : DFA $\rightarrow$ M

Given an automaton (DFA):

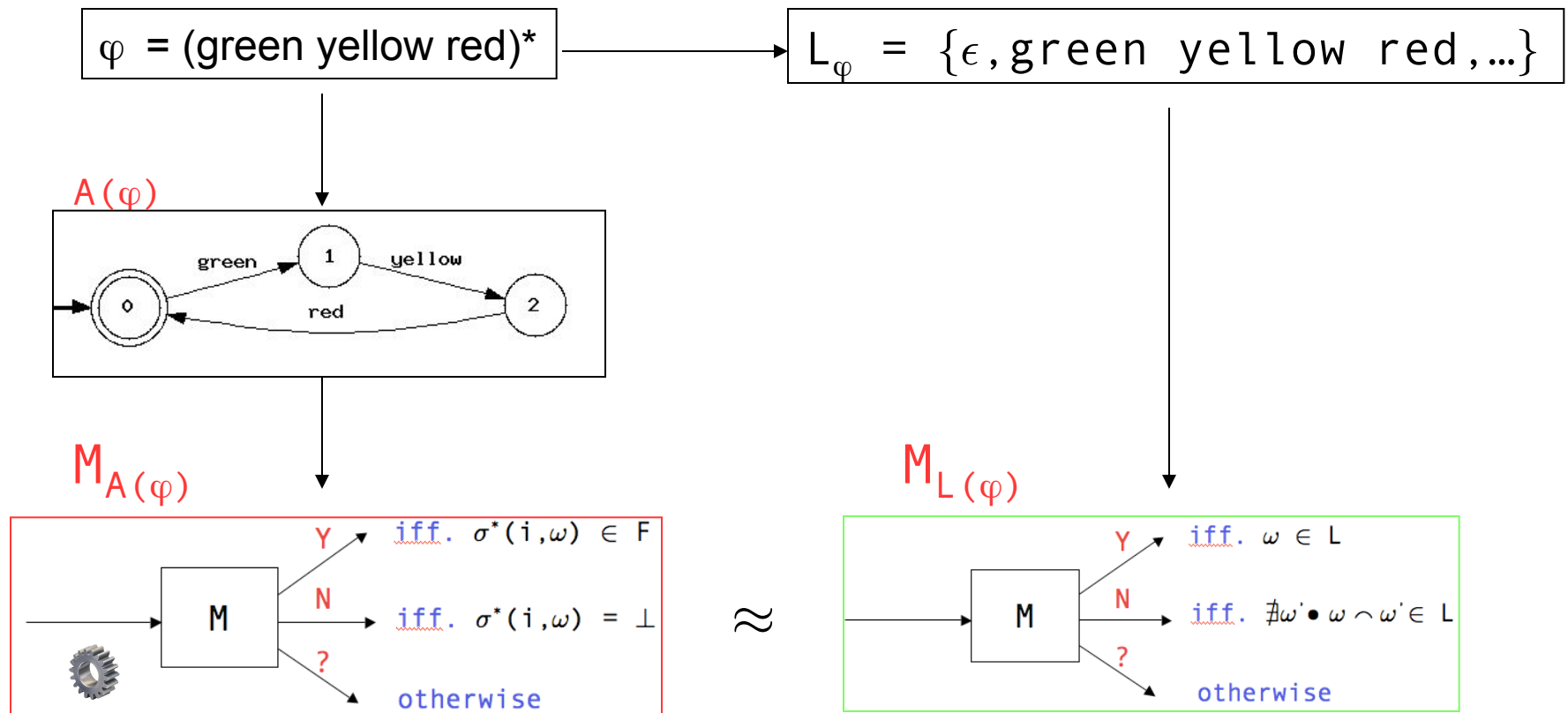
$$\mathcal{A} = \{Q, i \in Q, F \subseteq Q, \sigma: Q \times A \rightarrow Q_{\perp}\}$$

Define monitor  $M_{\mathcal{A}}$ :



# theorem

$M_{A(\varphi)}$  is equivalent to  $M_{L(\varphi)}$



---

# syntax for extended regular expressions

$E ::= \emptyset \mid \epsilon \mid A \mid E E \mid E^* \mid E+E \mid E\&E \mid \neg E$



examples  $A = \{a, b, c\}$ : extended with negation

$aab$	$\{aab\}$
$(ab)^*$	$\{\epsilon, ab, ababab, \dots\}$
$(a+b)^* \& \neg(ab)^*$	words of randomly interleaved a's and b's but not any words of form: ababab ... $\{a, aa, abba, bbbb, \dots\}$

# semantics

$L(\emptyset)$	$=$	$\{\}$
$L(\epsilon)$	$=$	$\{\epsilon\}$
$L(\alpha)$	$=$	$\{\alpha\}$
$L(E_1 E_2)$	$=$	$\{\omega_1\omega_2 \mid \omega_1 \in L(E_1) \wedge \omega_2 \in L(E_2)\}$
$L(E^*)$	$=$	$\{\omega_1\omega_2 \dots \omega_i \dots \omega_n \mid \omega_i \in L(E)\}$
$L(E_1 + E_2)$	$=$	$L(E_1) \cup L(E_2)$
$L(E_1 \& E_2)$	$=$	$L(E_1) \cap L(E_2)$
$L(\neg E)$	$=$	$A^* \setminus L(E)$

---

# from regular expressions to monitors via state machines

recall:

regular expression (RE)

→ step 1 ←

deterministic finite state automaton (DFA)

→ step 2

monitor (M)

---

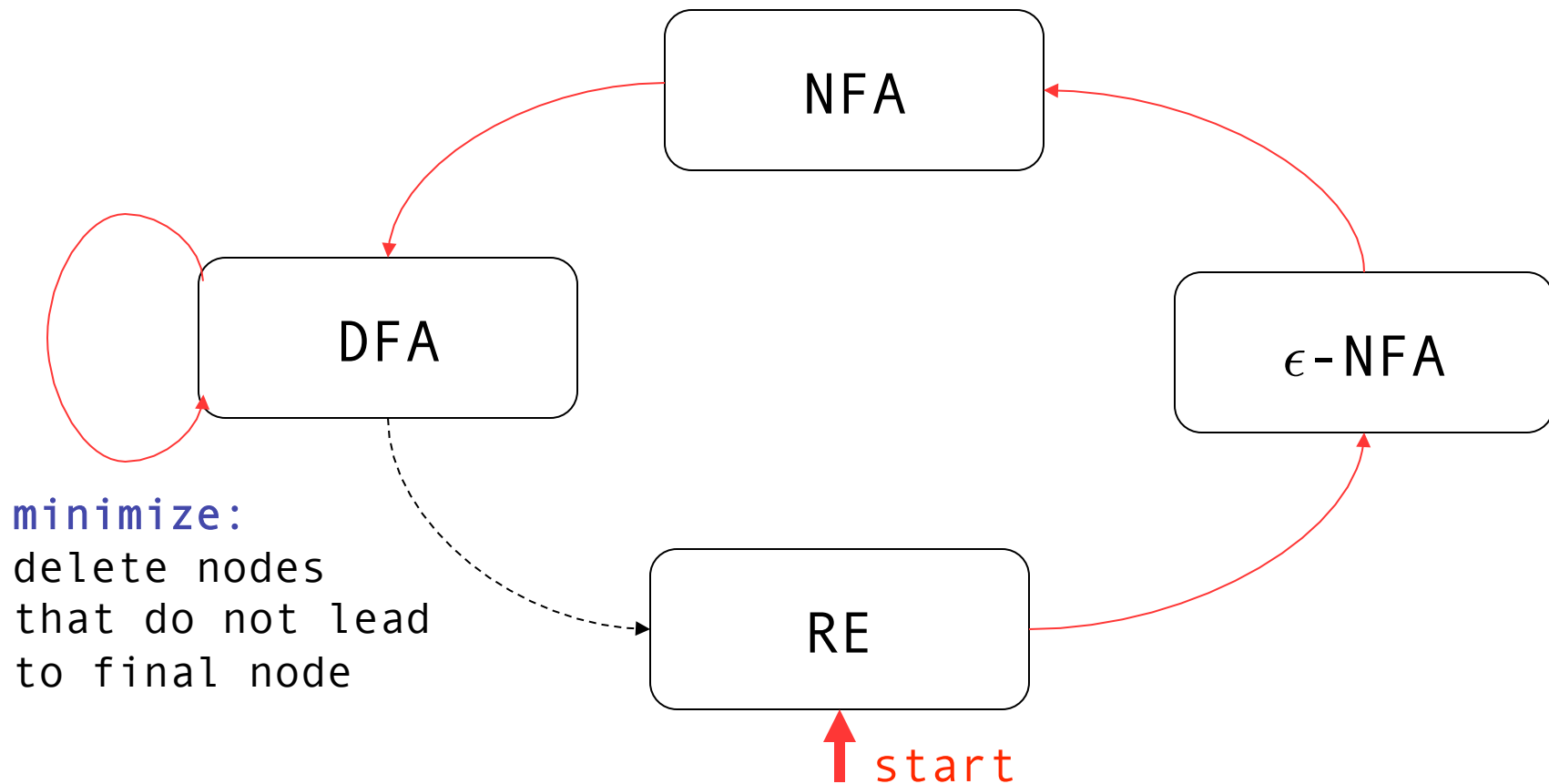
# four representations of regular languages

1. regular expressions (RE)
  2. non-deterministic  $\epsilon$ -automata ( $\epsilon$ -NFA)
  3. non-deterministic finite automata (NFA)
  4. deterministic finite automata (DFA)
- a DFA denotes a so-called regular language (set of words accepted by the automaton).
  - for any specification in any of the above 4 representations, there exist specifications in the other representations denoting the same language. Especially: RE  $\rightarrow$  DFA

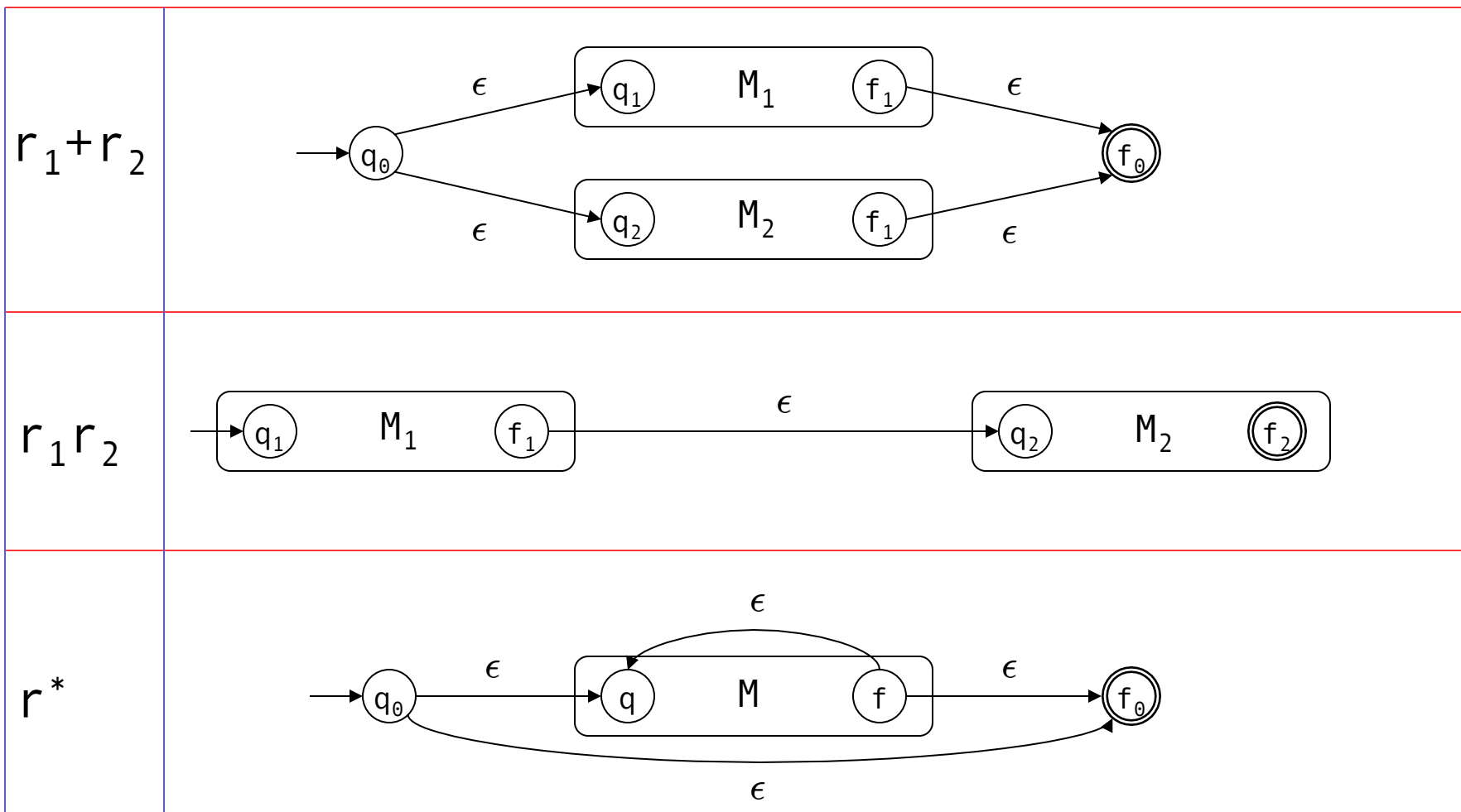
---

# Step 1 : RE $\rightarrow$ DFA

DFA  $\rightarrow$  minimized DFA



# RE $\rightarrow$ $\epsilon$ -NFA

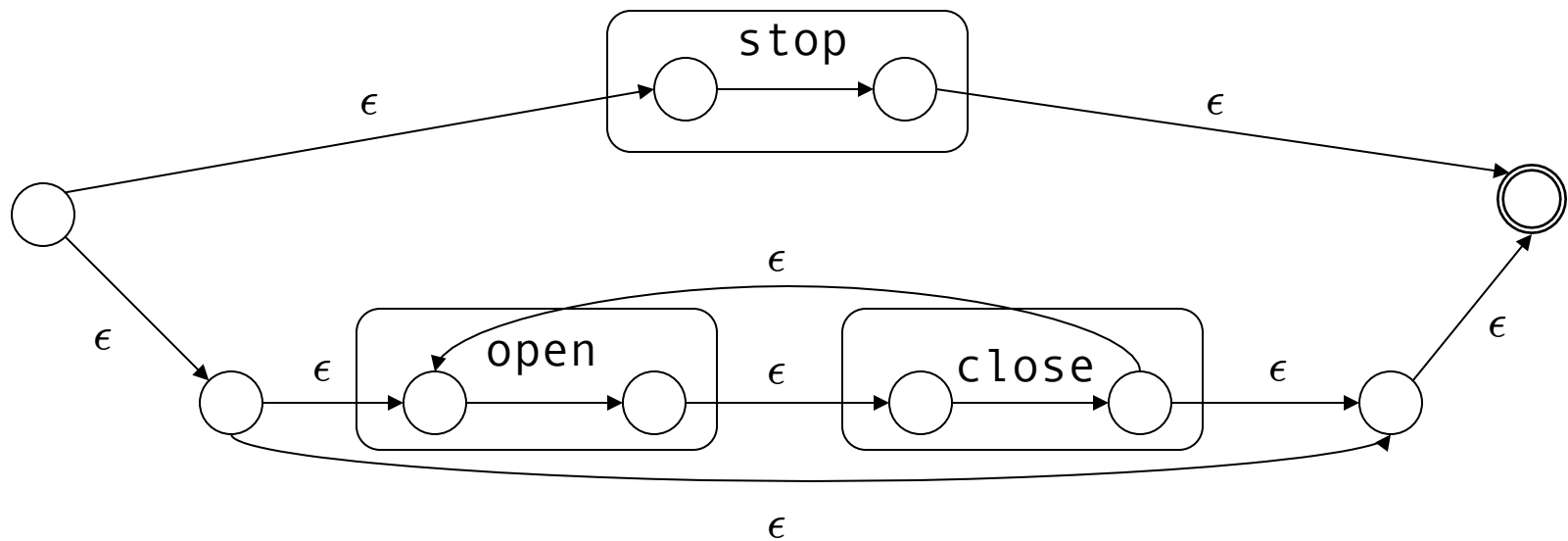


---

# example derivation

## RE $\rightarrow$ $\epsilon$ -NFA

$(\text{open close})^* + \text{stop}$

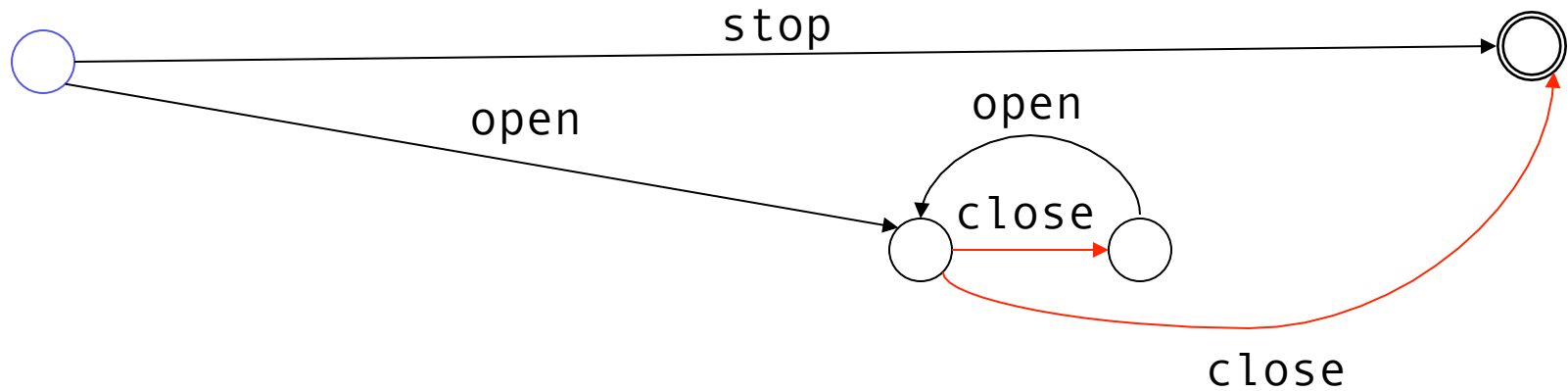


---

# example derivation

$\epsilon$ -NFA  $\rightarrow$  NFA

(open close)\* + stop

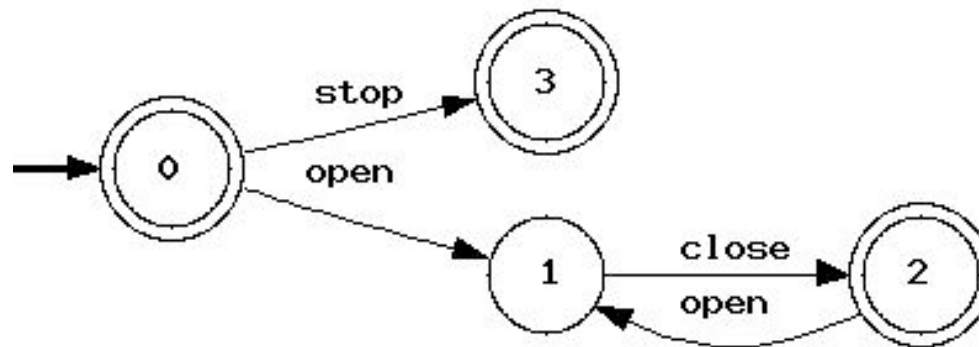


---

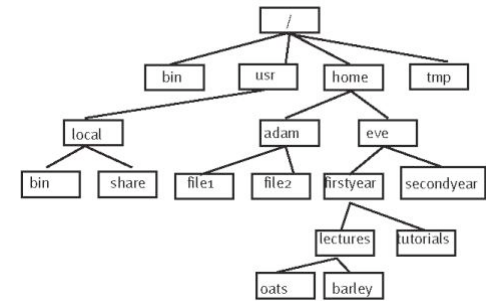
# example derivation

## NFA $\rightarrow$ DFA

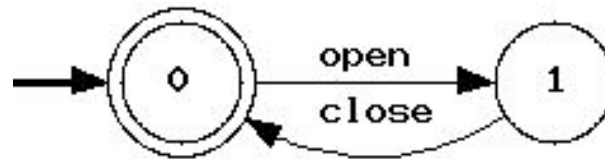
(open close)\* + stop



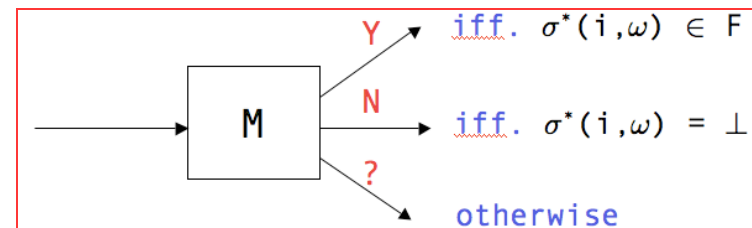
# file example revisited



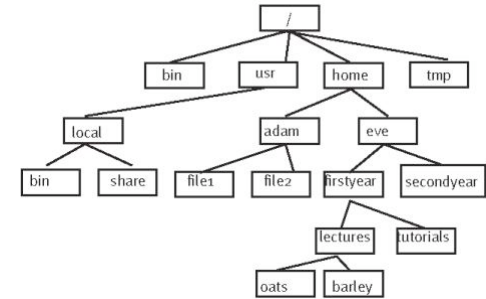
(open close)\*



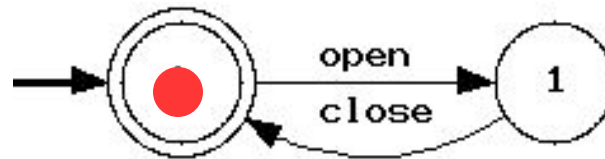
consider the trace:  
open close open



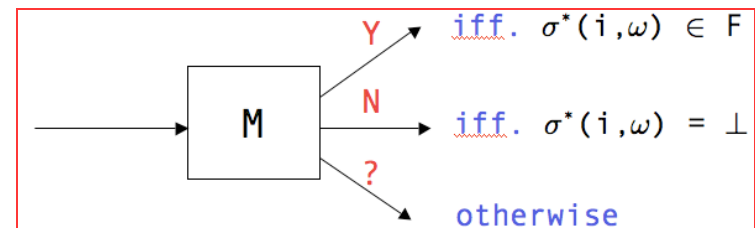
# file example revisited



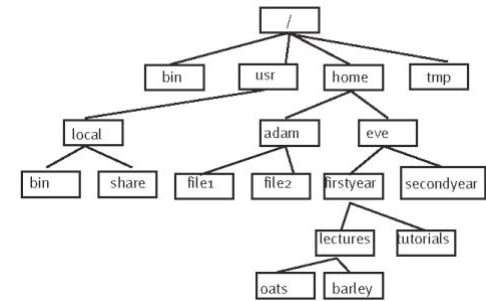
(open close)\*



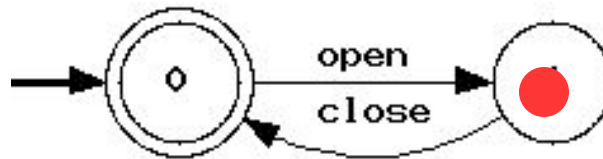
consider the trace:  
open close open



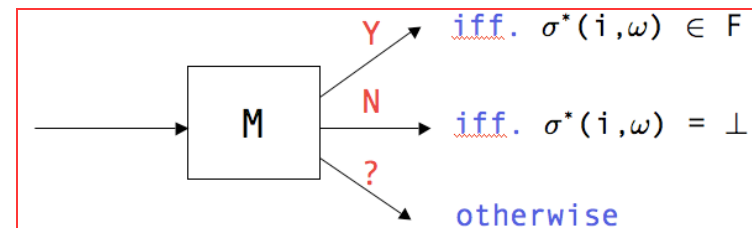
# file example revisited



(open close)\*

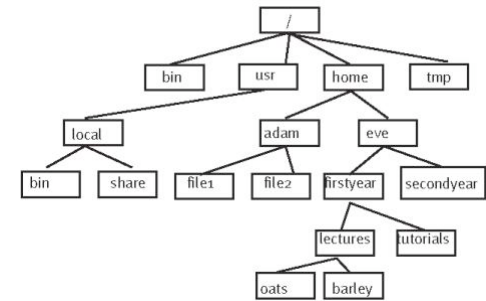


consider the trace:  
open close open

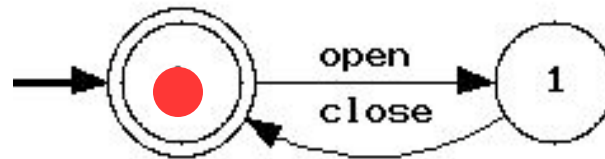


?

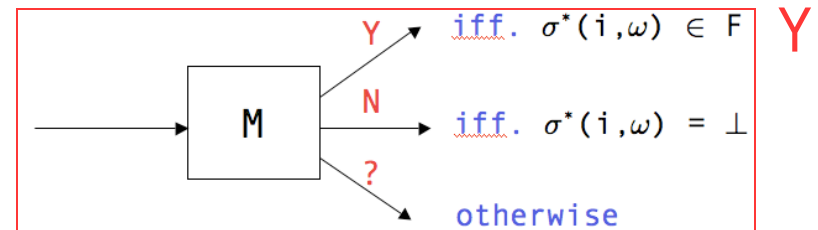
# file example revisited



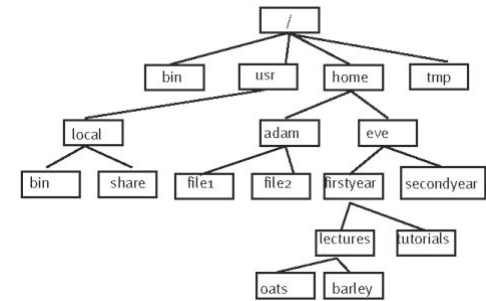
(open close)\*



consider the trace:  
open close open

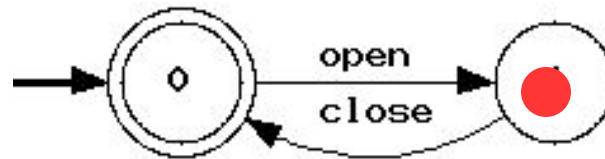


# file example revisited

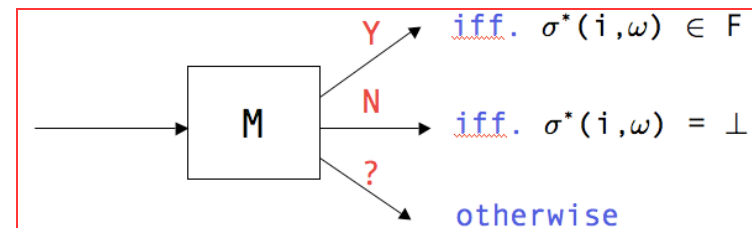


(open close)\*

monitored as safety property  $\Rightarrow$  no error

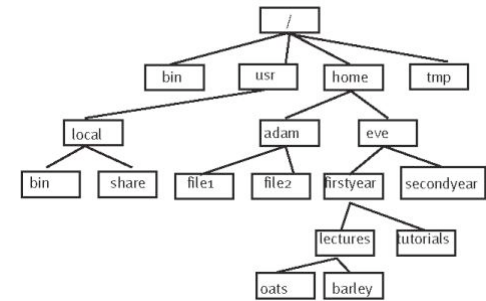


consider the trace:  
open close open

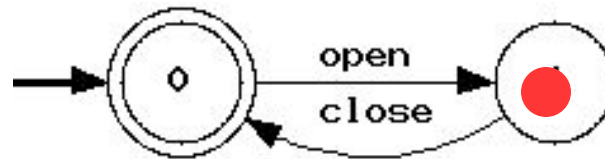


?

# file example revisited

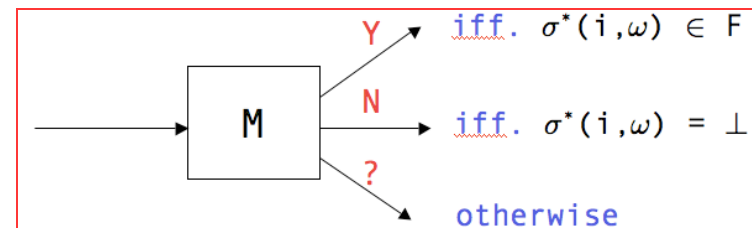


(open close)\*



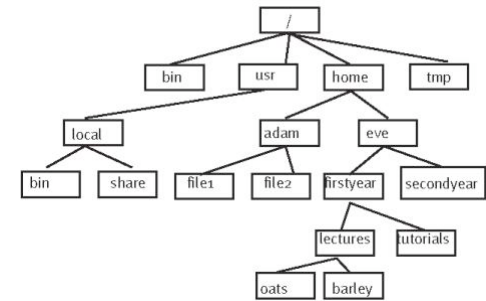
now, assume extra event

consider the trace:  
open close open open



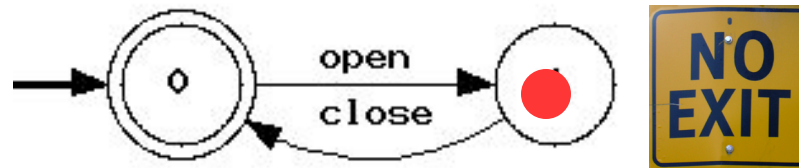
?

# file example revisited



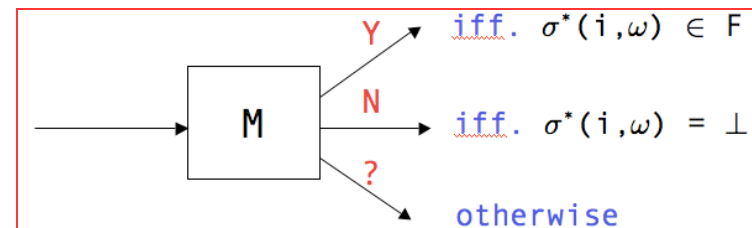
(open close)\*

monitored as safety property  $\Rightarrow$  error



**DANGER**

consider the trace:  
open close open open



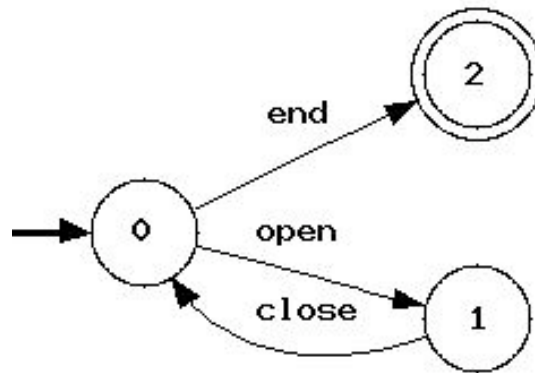
**N**

*R1: A file should eventually be closed once opened.*

## can write a response property?

introduce a new **end** event  
that **we know** will get emitted

$(\text{open close})^* \text{end}$



# properties of Java library APIs

The screenshot shows the Java API documentation for the `Iterator` interface in the `java.util` package. The page is titled "Iterator (Java 2 Platform SE 5.0)". The left sidebar lists various Java classes and packages. The main content area shows the interface definition and a list of methods. A red box highlights a property  $R_1$  with the text: "There should be no two calls to `next()` without a call to `hasNext()` in between, on the same iterator."

**Overview Package Class Use Tree Deprecated Index Help**

PREV CLASS NEXT CLASS  
SUMMARY: NESTED | FIELD | CONSTR | METHOD  
DETAIL: FIELD | CONSTR | METHOD

java.util  
**Interface Iterator<E>**

All Known Subinterfaces:  
[ListIterator<E>](#)

All Known Implementing Classes:  
[BeanContextSupport.BCSIterator](#), [Scanner](#)

public interface **Iterator**<E>

An iterator over a collection. Iterator takes the place of Enumeration in the Java collections framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

This interface is a member of the [Java Collections Framework](#).

Since:  
1.2

See Also:  
[Collection](#), [ListIterator](#), [Enumeration](#)

**Method Summary**

boolean	<a href="#">hasNext()</a>	Returns true if the iteration has more elements.
E	<a href="#">next()</a>	Returns the next element in the iteration.
void	<a href="#">remove()</a>	Removes from the underlying collection the last element returned by the iterator (optional operation).

**Method Detail**

[hasNext](#)

```
java.util
Interface Iterator<E>

All Known Subinterfaces:
  ListIterator<E>

All Known Implementing Classes:
  BeanContextSupport.BCSIterator, Scanner
```

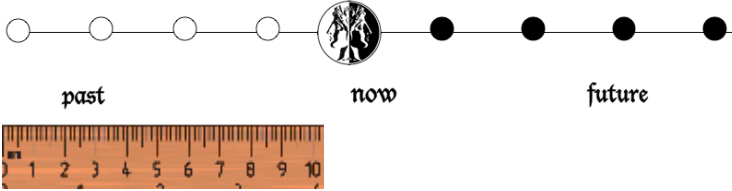
$R_1$ : There should be no two calls to `next()` without a call to `hasNext()` in between, on the same iterator.

$(hasNext\ next)^*$

reg exp is too strong  
should allow repeated hasNext calls



- i) total trace semantics
- ii) looking for violation



```

java.util
Interface Iterator<E>

All Known Subinterfaces:
  ListIterator<E>

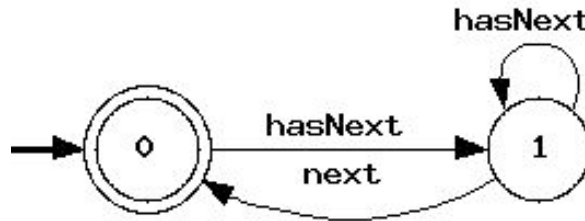
All Known Implementing Classes:
  BeanContextSupport.BCSIterator, Scanner

```

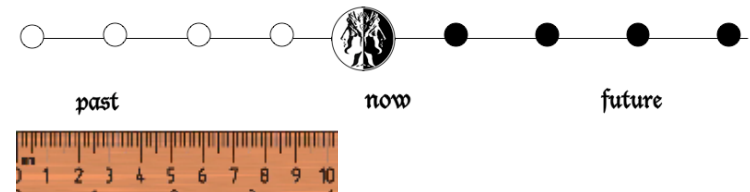
$R_1$ : There should be no two calls to `next()` without a call to `hasNext()` in between, on the same iterator.

$(hasNext\ hasNext^*\ next)^*$

reg exp  
still too  
strong  
should allow  
optional  
initial next  
call



- i) total trace semantics
- ii) looking for violation



```

java.util
Interface Iterator<E>

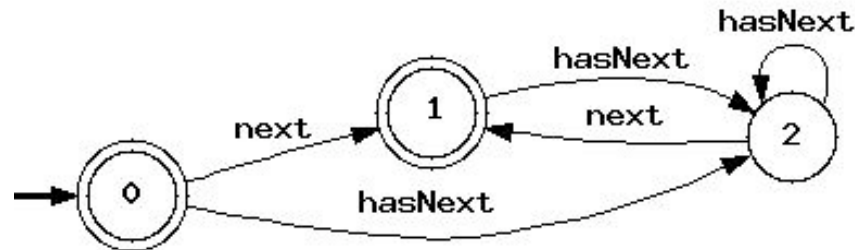
All Known Subinterfaces:
  ListIterator<E>

All Known Implementing Classes:
  BeanContextSupport.BCSIterator, Scanner

```

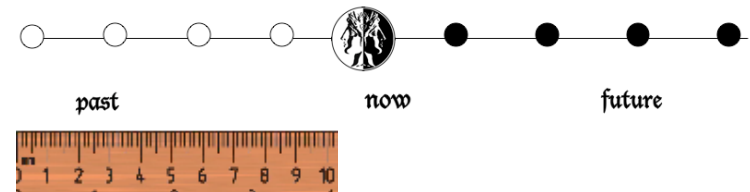
$R_1$ : There should be no two calls to `next()` without a call to `hasNext()` in between, on the same iterator.

$(next + \epsilon)(hasNext\ hasNext^*\ next)^*$



reg exp is too strong should allow next not to be called

- i) total trace semantics
- ii) looking for violation



```

java.util
Interface Iterator<E>

All Known Subinterfaces:
  ListIterator<E>

All Known Implementing Classes:
  BeanContextSupport.BCSIterator, Scanner

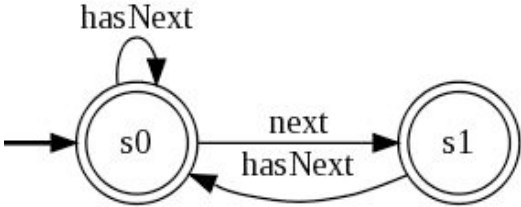
```

$R_1$ : There should be no two calls to `next()` without a call to `hasNext()` in between, on the same iterator.

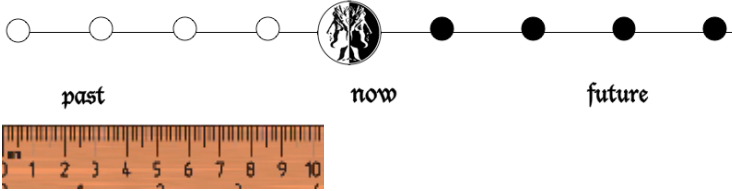
complex!?

$(\text{next} + \text{epsilon})(\text{hasNext} \text{ hasNext}^* (\text{next} + \text{epsilon}))^*$

ok



- i) total trace semantics
- ii) looking for violation



```

java.util
Interface Iterator<E>

All Known Subinterfaces:
  ListIterator<E>

All Known Implementing Classes:
  BeanContextSupport.BCSIterator, Scanner

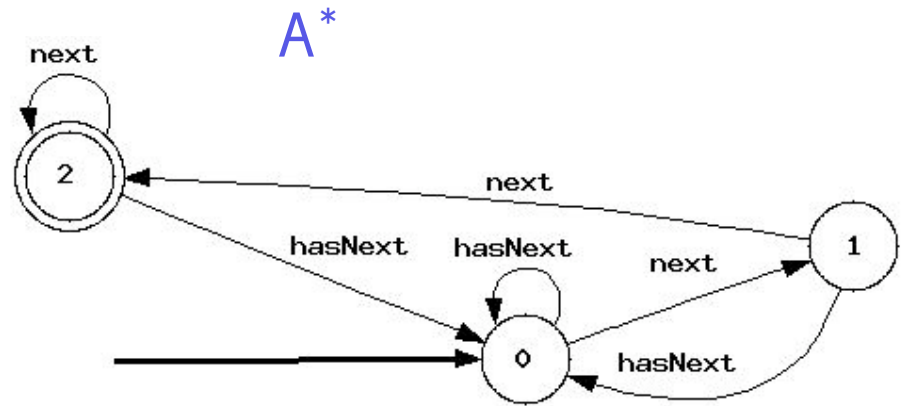
```

$R_1$ : There should be no two calls to `next()` without a call to `hasNext()` in between, on the same iterator.

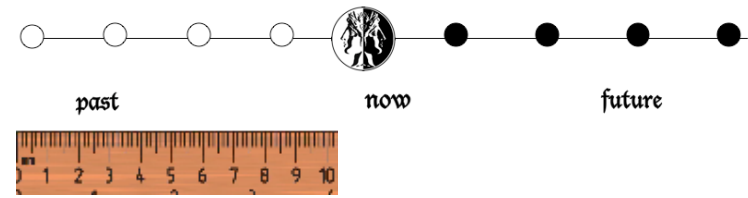
can we simplify further!?

~empty next next

ok



- i) total trace semantics
- ii) looking for validation

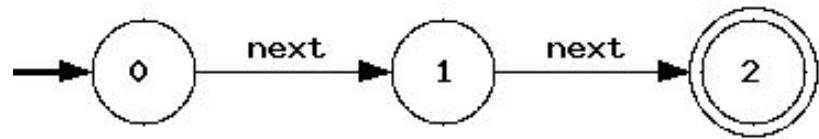


```
java.util
Interface Iterator<E>
All Known Subinterfaces:
  ListIterator<E>
All Known Implementing Classes:
  BeanContextSupport.BCSIterator, Scanner
```

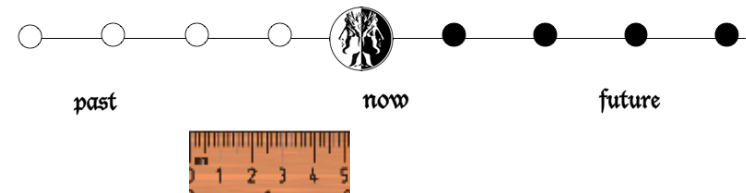
$R_1$ : There should be no two calls to `next()` without a call to `hasNext()` in between, on the same iterator.

next next

final



- i) suffix trace semantics
- ii) looking for validation



---

# next examples

- suffix trace semantics
- validation (meaning an error)
- these interpretations appear the most convenient for regular expressions
- they yield the most succinct specifications
- most (if not all) existing systems for regular expressions support this interpretation
- this does not apply necessarily to other logic systems, such as for example temporal logic.

# properties of Java library APIs

The screenshot shows the Java API documentation for the `Enumeration` interface in the `java.util` package. The page is titled "Enumeration (Java 2 Platform SE 5.0)". The navigation tabs include Overview, Package, Class (selected), Use, Tree, Deprecated, Index, and Help. The page content includes a summary of the interface, a list of subinterfaces, and a list of implementing classes. A red box highlights a note about propagation. A blue box highlights the Method Summary section.

**Java™ 2 Platform Standard Ed. 5.0**  
[All Classes](#)

Packages  
[java.applet](#)  
[java.awt](#)  
[Entity](#)  
[EntityReference](#)  
[EntityResolver](#)  
[EntityResolver2](#)  
[Enum](#)  
[EnumConstantNotPresent](#)  
[EnumControl](#)  
[EnumControl.Type](#)  
[Enumeration](#)  
[EnumMap](#)  
[EnumSet](#)  
[EnumSyntax](#)  
[Environment](#)  
[EOFException](#)  
[Error](#)  
[ErrorHandler](#)  
[ErrorListener](#)  
[ErrorManager](#)  
[EtchedBorder](#)  
[Event](#)  
[Event](#)

**Overview Package Class Use Tree Deprecated Index Help** **Java™ 2 Platform Standard Ed. 5.0**

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#) DETAIL: FIELD | CONSTR | [METHOD](#)

`java.util`  
**Interface Enumeration<E>**

**All Known Subinterfaces:**  
[NamingEnumeration<T>](#)

**All Known Implementing Classes:**  
[StringTokenizer](#)

**R<sub>2</sub>: An enumeration should not be propagated after the underlying vector has been changed .**

**Method Summary**

boolean	<a href="#">hasMoreElements</a> () Tests if this enumeration contains more elements.
E	<a href="#">nextElement</a> () Returns the next element of this enumeration if this enumeration object has at least one more element to provide.

Find: 97    Next    Previous    Highlight all    Match case

Downloads

```
java.util
Interface Enumeration<E>
```

```
All Known Subinterfaces:
  NamingEnumeration<T>
```

```
All Known Implementing Classes:
  StringTokenizer
```

$R_2$ : An enumeration should not be propagated after the underlying vector has been changed .

create next\* updatesource updatesource\* next



# properties of Java library APIs

The image shows two overlapping browser windows displaying Java API documentation. The top window shows the `HashSet` class page, and the bottom window shows the `Collection` interface page. A red box highlights a specific property of `HashSet`.

**HashSet (Java 2 Platform SE 5.0)**

Overview Package **Class** Use Tree Deprecated Index Help Java™ 2 Platform Standard Ed. 5.0

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.util  
**Class HashSet<E>**

java.lang.Object  
↳ java.util.AbstractCollection<E>

**R<sub>3</sub>**: A collection should not be modified while it is a member of a hashset (don't change the hashcode).

**Collection (Java 2 Platform SE 5.0)**

Overview Package **Class** Use Tree Deprecated Index Help Java™ 2 Platform Standard Ed. 5.0

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.util  
**Interface Collection<E>**

All Superinterfaces:  
↳ Iterable<E>

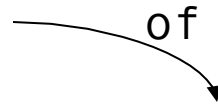
All Known Subinterfaces:  
↳ BeanContext, BeanContextServices, BlockingQueue<E>, List<E>, Queue<E>, Set<E>, SortedSet<E>

All Known Implementing Classes:  
↳ AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentLinkedQueue, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingQueue, LinkedHashSet, LinkedList, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

..., backed by a hash table  
no guarantees as to the iteration  
... not guarantee that the order

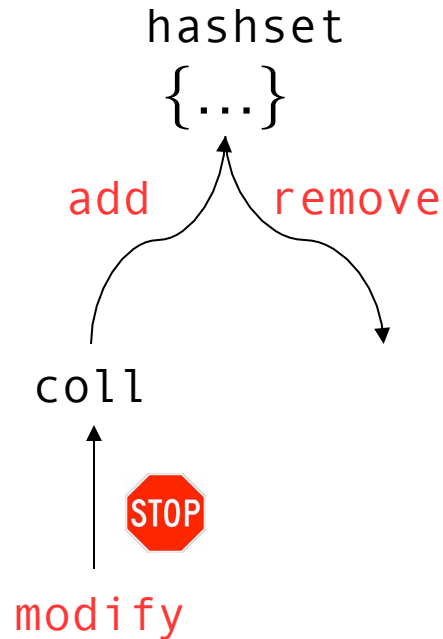
java.util  
**Class HashSet<E>**

java.lang.Object  
└ java.util.AbstractCollection<E>  
 └ java.util.AbstractSet<E>  
 └ java.util.HashSet<E>



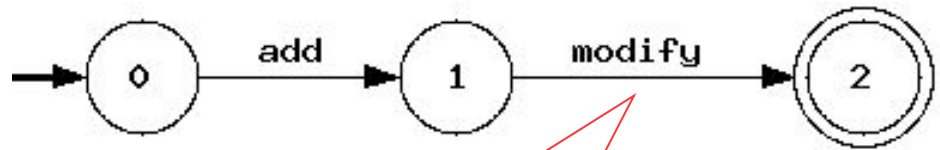
java.util  
**Interface Collection<E>**

**R<sub>3</sub>**: A collection should not be modified while it is a member of a hashset (don't change the hashCode).



add	collection to hashset
remove	collection from hashset
modify	collection

add modify



note: without having seen a remove

suffix validation

---

# limitations of regular expressions for specification

- regular expressions convenient for “brief” properties.
- less convenient on very state-full problems, where all good or bad behaviors must be formulated (state machines, see next slides).
- can only express regular properties. They cannot count an apriori unknown number of times, as required for the following property:

$R_4$ : locks can be taken in a nested manner, but should be released in reverse order.

lock<sup>n</sup> release<sup>n</sup>

```
lock
  lock
    lock
      release
    release
  release
```

# properties of Java library APIs

The screenshot shows the Java 2 Platform Standard Ed. 5.0 API documentation for the `Thread` class. The browser address bar shows `http://java.sun.com/j2se/1.5.0/docs/api/`. The page title is "Thread (Java 2 Platform SE 5.0)". The navigation bar includes "Overview", "Package", "Class", "Use Tree", "Deprecated", "Index", and "Help". The "Class" tab is selected. The page content includes:

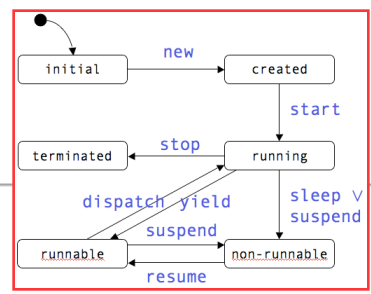
- Navigation links: [PREV CLASS](#), [NEXT CLASS](#), [FRAMES](#), [NO FRAMES](#)
- SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)    DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)
- Class hierarchy: `java.lang.Object` | `java.lang.Thread`
- All Implemented Interfaces: [Runnable](#)
- Code snippet:

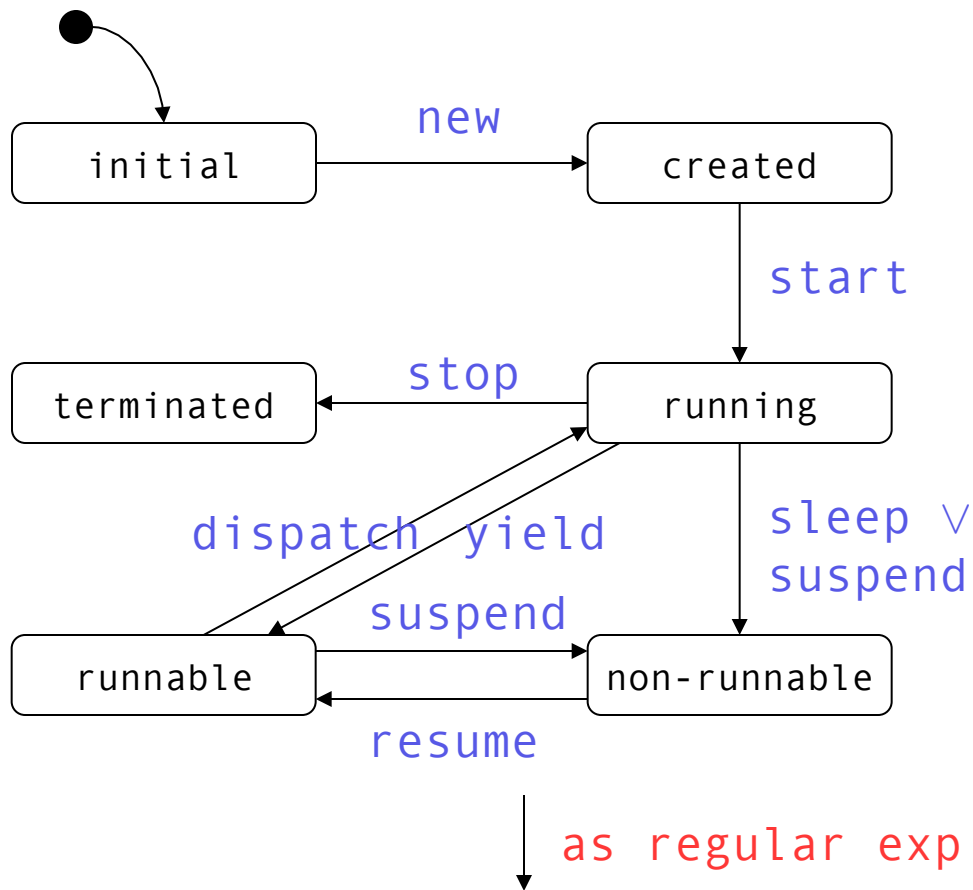
```
public class Thread
extends Object
implements Runnable
```
- Text: "A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently."
- Text: "Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread"

A state machine diagram is shown in a red box, illustrating the lifecycle of a thread:

```
graph TD
    start(( )) --> initial
    initial -- new --> created
    created -- start --> running
    running -- stop --> terminated
    running -- sleep v suspend --> non-running
    non-running -- resume --> runnable
    runnable -- dispatch yield --> running
```

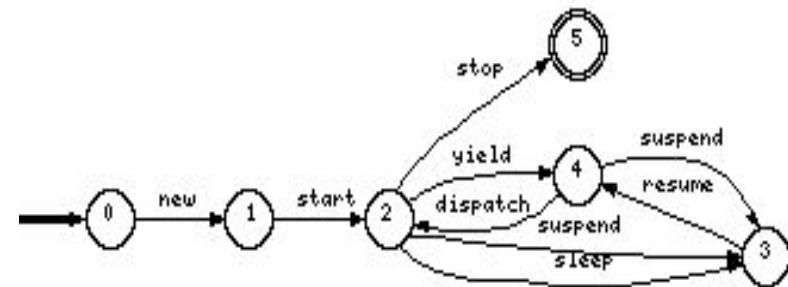
R<sub>5</sub>: A thread's life cycle should conform to the following state machine:





RE not suited for complex state

example: thread's life cycle



as regular exp

```

new start
((
  (yield (suspend resume)* dispatch) +
  ((sleep+suspend) (resume suspend)* resume dispatch)
))*
stop
  
```

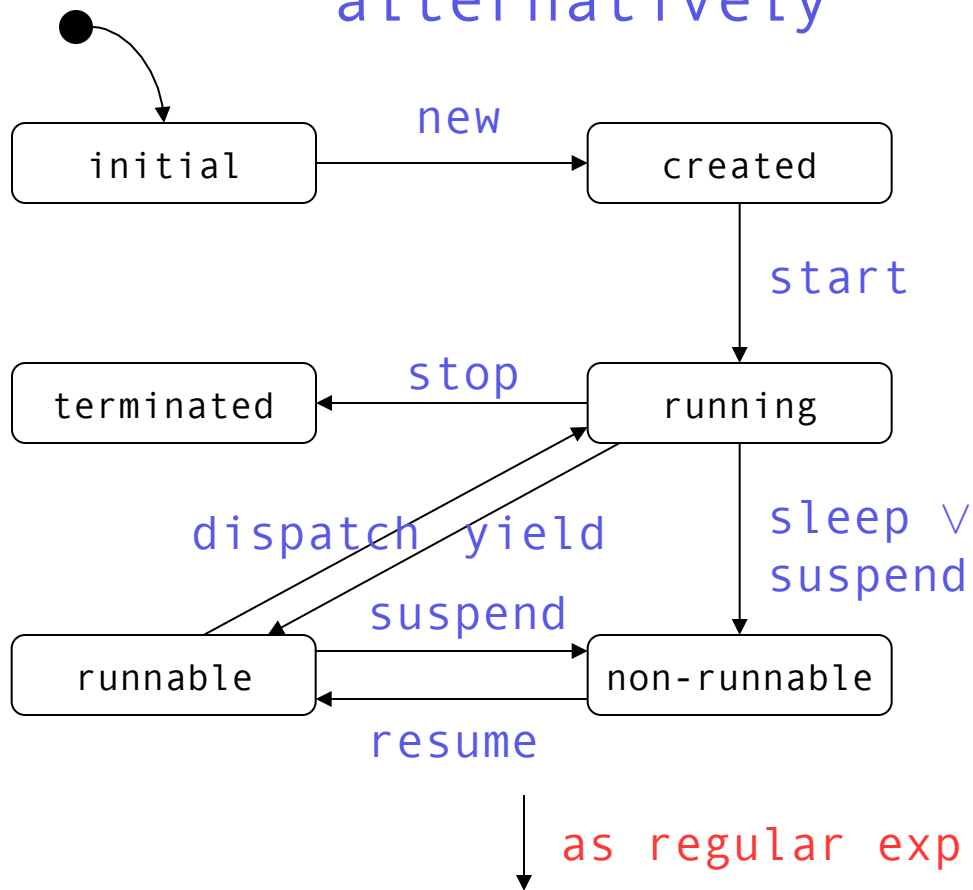
total trace  
track violation

tool

what if one can stop in every state?



alternatively



as regular exp

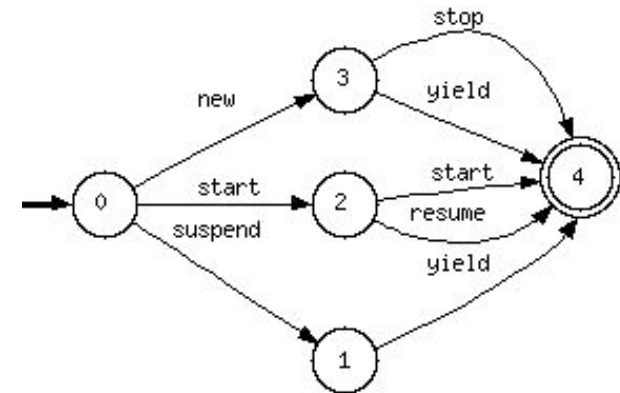
(new stop) +  
(new yield) +  
(start start) +  
(start resume) +  
(suspend yield)  
+ ...

suffix trace  
track validation

bad scenarios

RE not suited for  
complex state

example: thread's  
life cycle



---

# next time

- state machines and regular expressions in **JavaMOP**, a framework for writing logic-based monitors:

<http://fsl.cs.uiuc.edu/index.php/Special:JavaMOP2.0>

- data parameterization:

$\forall (f:\text{File}) \bullet (\text{open}(f) \text{ close}(f))^*$

JavaMOP 2.0 - FSL

http://fsl.cs.uiuc.edu/index.php/Special:JavaMOP2.0

Joergen Ingman Eventseer.net - Home Free Website Polls Programming with C/C++ Castle for Sale Costa Rica, ...Estate Blogs BASH Help - ...sh Tutorial DNS Alliance JPL Apple (194) News (1290) Shopping Amazon eBay Yahoo! Log in / create account

**JavaMOP 2.0**

Authors:  
Feng Chen  
Patrick Meredith  
Dongyun Jin  
Grigore Rosu

languages	MOP	FSM	ERE	CFG	PTLTL	FTLTL	PTCaRet	...
	JavaMOP	JavaFSM	JavaERE	JavaCFG	JavaPTLTL	JavaFTLTL	JavaPTCaRet	...
	BusMOP	BusFSM	BusERE	...	BusPTLTL	...	...	...
	...	...	...	...	...	...	...	...

MOP Matrix: a clickable map of MOP pages.

(This is the new JavaMOP2.0. The old [JavaMOP1.0](#) is now deprecated.)

### Download JavaMOP

Before downloading it, you can try JavaMOP online using the form below. Most users will find the online version good enough for their needs, so will never need to download and install JavaMOP on their machines. [HERE](#) are the instructions on how to download, install and experiment with JavaMOP.

### Run JavaMOP Online

Enter your specification or chose (and modify) one example from the menu - provided examples are also reachable from the individual JavaMOP plugin pages, via the MOP matrix above. Click **Run** to run JavaMOP. Generated monitor can be compiled using any AspectJ compiler. [HERE](#) are instructions on how to do it.

**Note:** if there are any technical difficulties please alert [pmeredit@cs.uiuc.edu](mailto:pmeredit@cs.uiuc.edu)

### Performance Evaluation

One critical issue in runtime verification is the monitoring overhead caused by monitors. We have evaluated the performance of the monitoring code generated by JavaMOP using the [Dacapo benchmark](#) and many monitoring-intensive properties (most of them are listed below in the online interface). Our evaluation shows that JavaMOP generates very efficient monitoring code, as can be seen in the [JavaMOP experiments page](#).

Choose an example:

- CFG
- ERE
  - HasNext
  - SafeFile
  - SafeIterator
  - SafeMapIterator
  - SafeSyncCollection
  - SafeSyncMap
- FSM
- FTLTL
- PTLTL
- ptCaRet

```

ERE/HasNext
package mop;

import java.io.*;
import java.util.*;

//decentralized HasNext(Iterator i)
HasNext(Iterator i) {
    int k = 0;
    event hasNext after(Iterator i) : call(* Iterator.hasNext()) && target(i) {}
    event next before(Iterator i) : call(* Iterator.next()) && target(i) {}

    ere : (hasnext hasNext* next)*

    @violation {
        //System.err.println("! hasNext() has not been called before calling next() for an iterator");
        //Thread.dumpStack();
        __RESET;
    }
}

```

Run Reset Specification Syntax Help (new window)

try writing  
FSMs & EREs

MOP Extended Regular Expression (ERE) Plugin - FSL

http://fsl.cs.uiuc.edu/index.php/Special:EREPlugin

Joergen Ingman Eventseer.net - Home Free Website Polls Programming with C/C++ Castle for Sale Costa Rica Estate Blogs BASH Help ... sh Tutorial DNS Alliance JPL Apple (194) News (1290) Shopping Amazon eBay Yahoo!

This page allows one to synthesize online monitors from extended regular expressions (ERE), using the ERE plugin for MOP available for download below. The generated monitors are automata-based; see links in the top-right box for details and syntax. These monitors for ERE specifications are language-independent and can be used in various language instances of MOP, as well as in other monitoring applications not necessarily based on MOP. Simply chose one of the examples below from the list on the left, or write your own in the text box, then click run.

**Download:** [MOP\\_ERE\\_Plugin.zip](#)

**Note:** if there are any technical difficulties please alert pmeredit@cs.uiuc.edu

Choose an example:

- ComplexPattern2
- FileOperations
- HasNext
- HashSet
- InterruptFix
- SafeConversionSpeed
- SafeCounterModify
- SafeFile
- SafeIterator
- SafeMapIterator
- SafeSyncCollection
- SafeSyncMap

HasNext

```
event open
event close
ere: (open close)*
```

Number of generated monitors (since 14 December 2008): 792 (MOP), 164 (ERE), 48 (Language-neutral ERE)

The textbox below shows the generated monitor; see [ERE Plugin Output Syntax](#) (opens new window) for its precise syntax and meaning. [Below](#) the textbox is a more easily read graph representation of it.

```
fsm:
!s0[
  open -> s1
]
s1[
  close -> s0
]
```

Below is an image representation of the monitor:

(open close)\*

```

graph LR
  start(( )) --> s0(((s0)))
  s0 -- open --> s1(((s1)))
  s1 -- close --> s0
  style start fill:none,stroke:none
  
```

Powered by MediaWiki

---

end