introduction to runtime verification

CS 119

beyond `assert` and `print`

action

standing order: sell when price drops more than 2% within 1 day.

property

events from Wall Street

definition

runtime verification is the study of how to design artifacts for monitoring and analyzing system executions. Such artifacts can be used for a variety of purposes, including testing/program understanding and fault protection.

the informal version

runtime verification is the study of how to get as much out of your runs possible.

purposes

- correctness checking
- debugging and testing
- dynamic documentation/understanding
- performance evaluation
- fault protection/dependability
- security/intrusion detection
- functionality modification
- aspect oriented programming

field still has many names

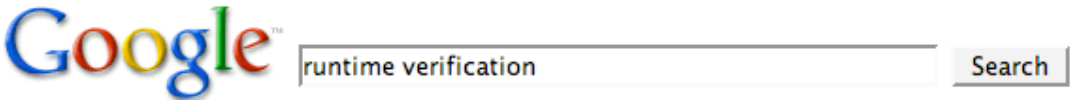
- runtime verification
- runtime monitoring
- runtime checking
- runtime result checking
- runtime reflection
- monitoring oriented programming
- design by contract
- runtime analysis
- dynamic analysis
- trace analysis
- fault protection

RV'08 – Eighth Workshop on *Runtime Verification*

Satellite workshop of ETAPS'08.

March 30, 2008
Budapest, Hungary

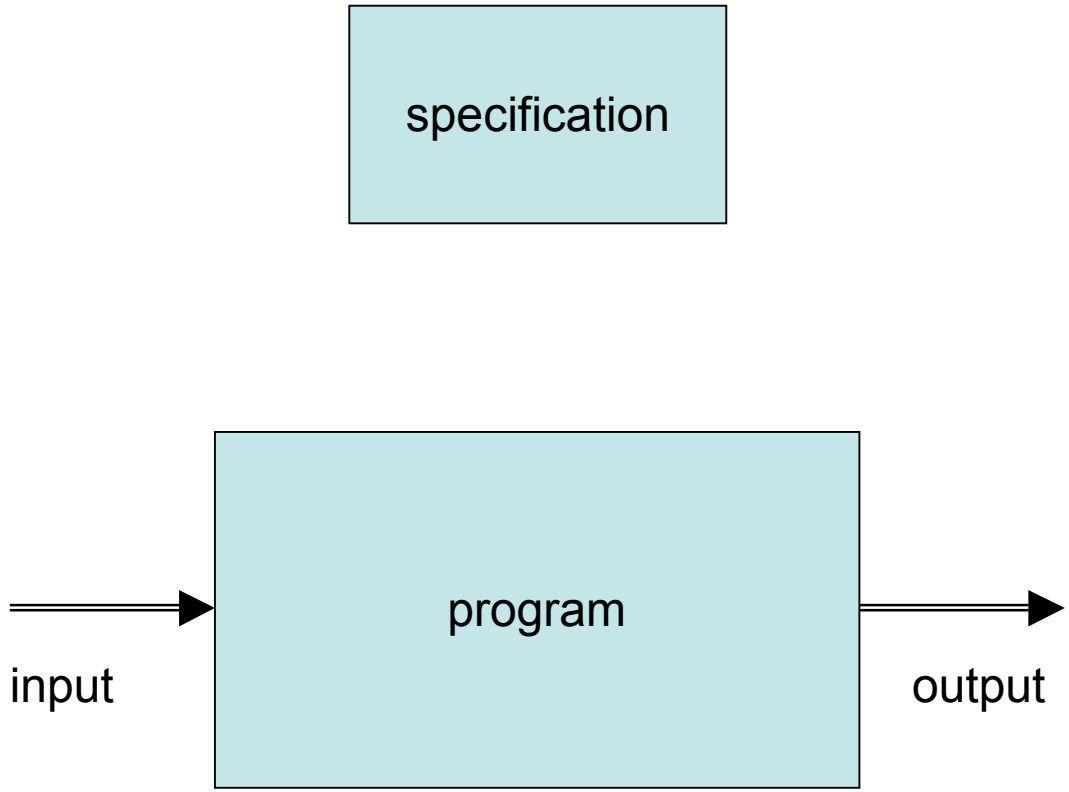
http://www.runtime-verification.org



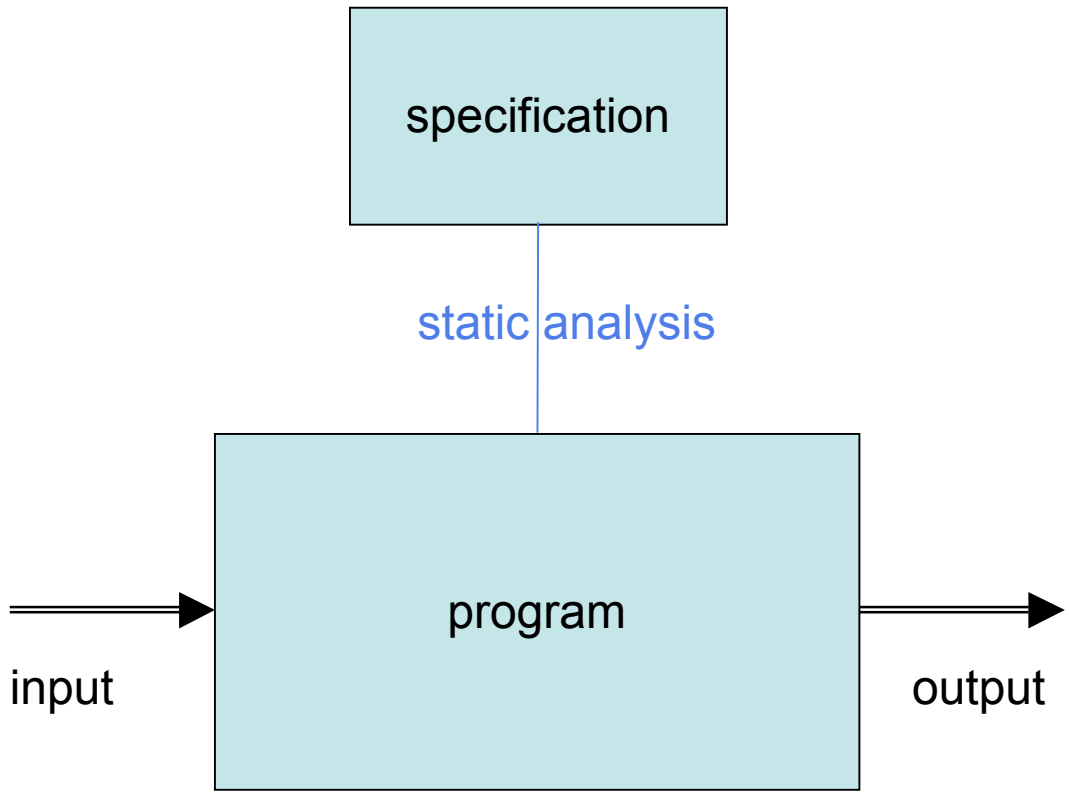
Sixth International Workshop on Dynamic Analysis (WODA 2008)

Co-located with ISSTA 2008
July 21, 2008
Seattle, Washington

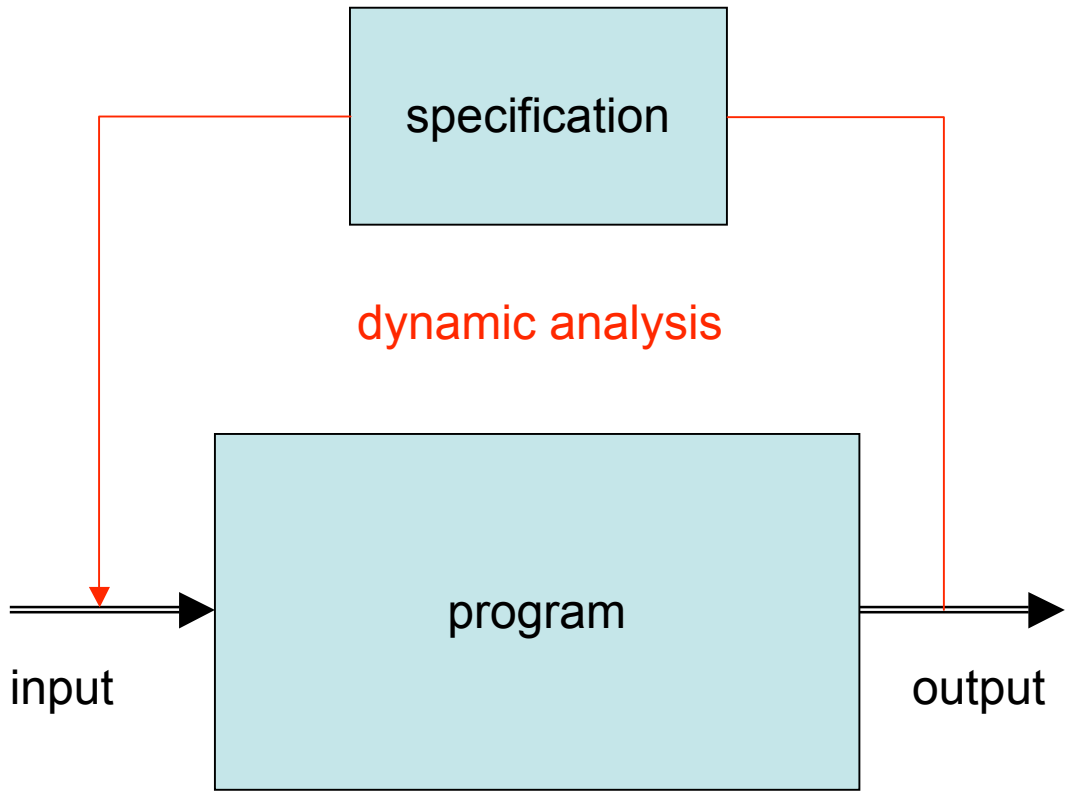
specification and programming



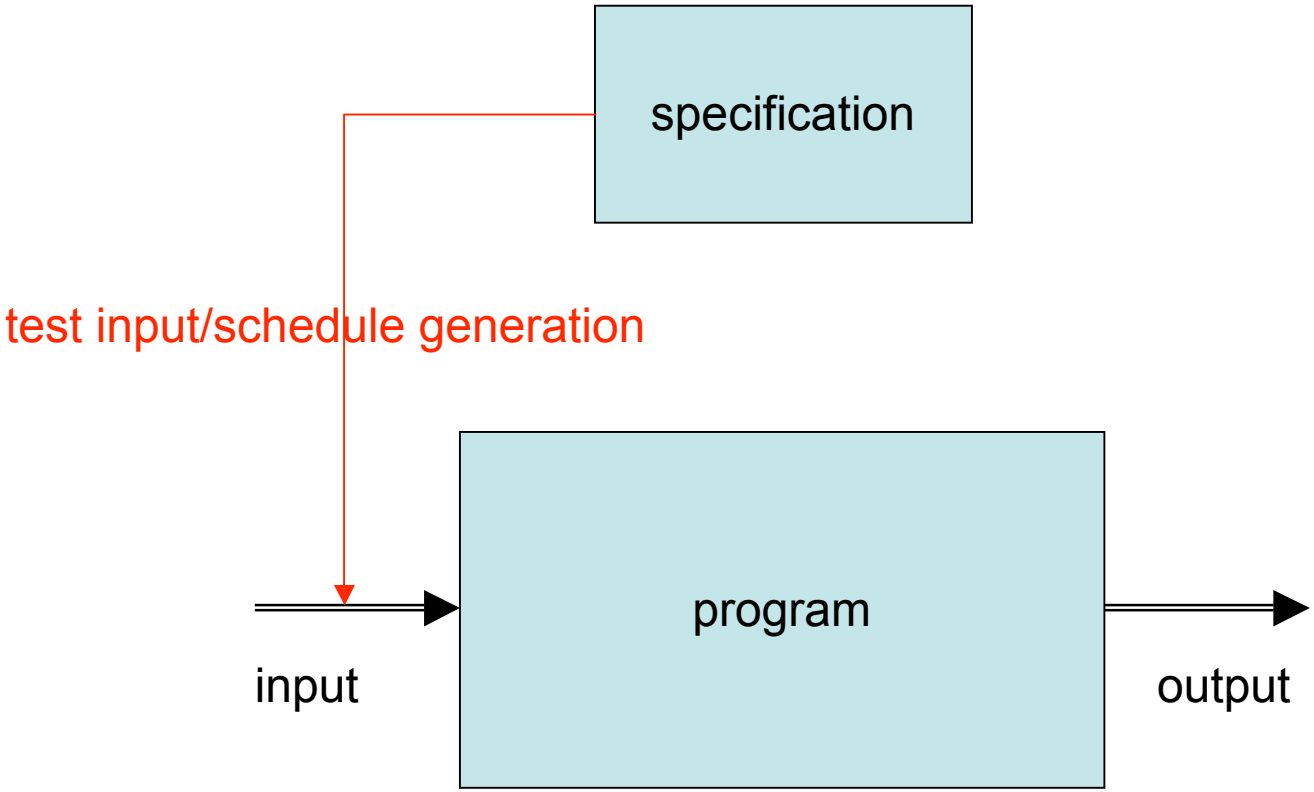
specification and programming



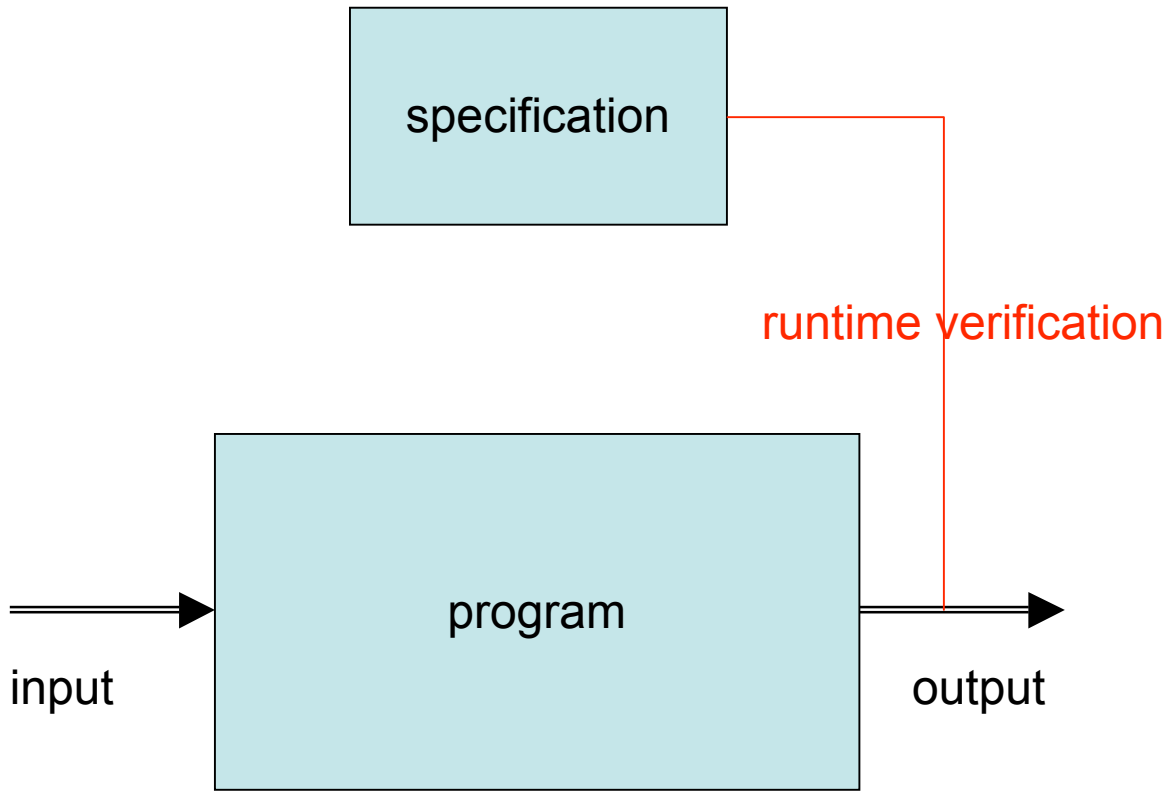
specification and programming



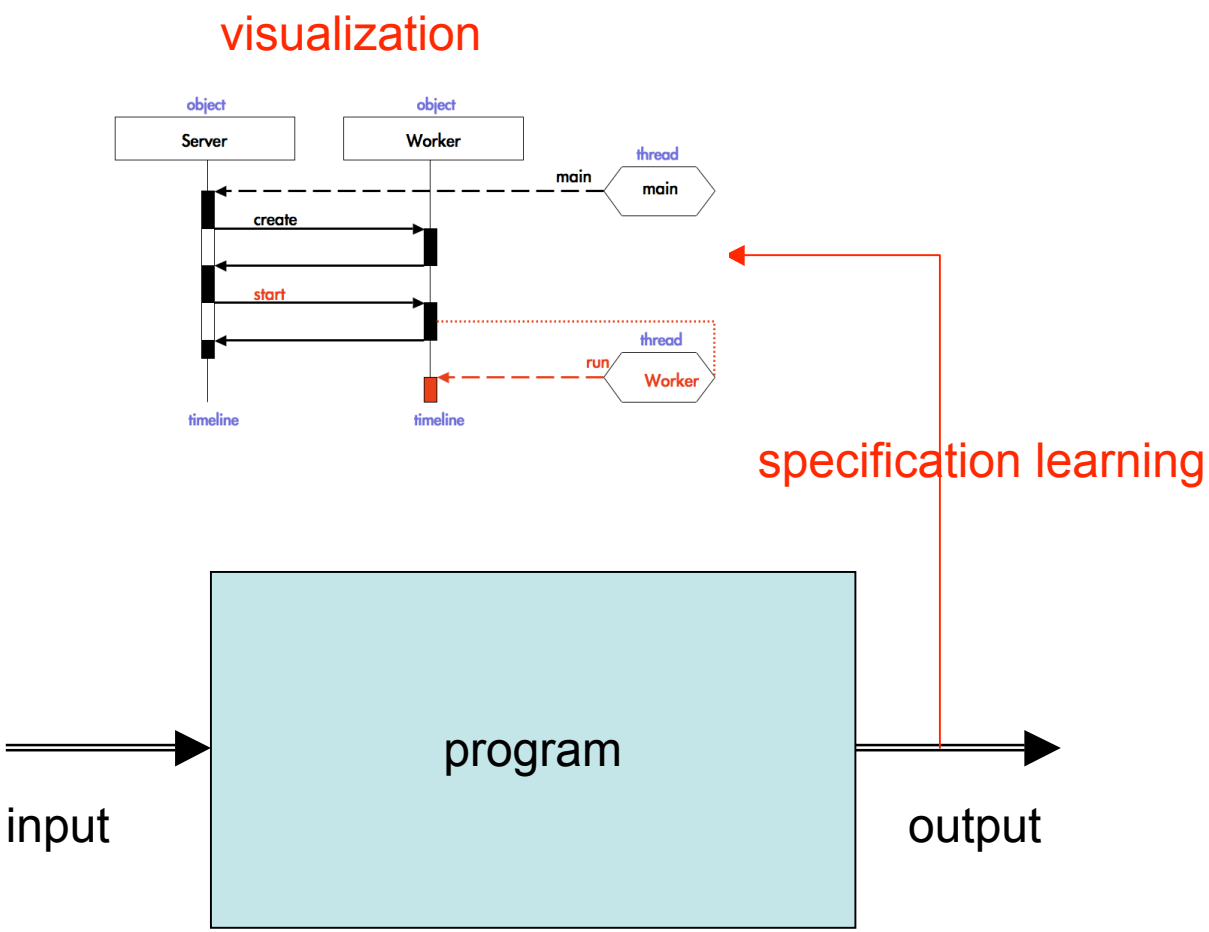
specification and programming



specification and programming

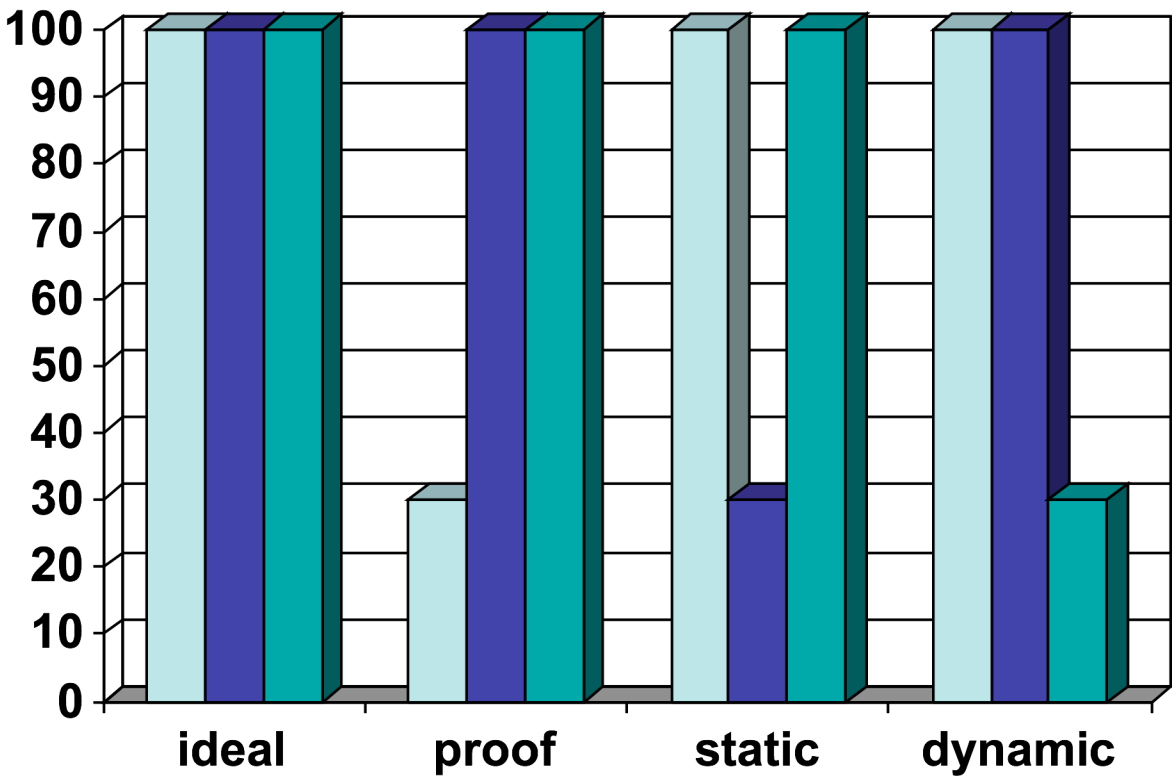


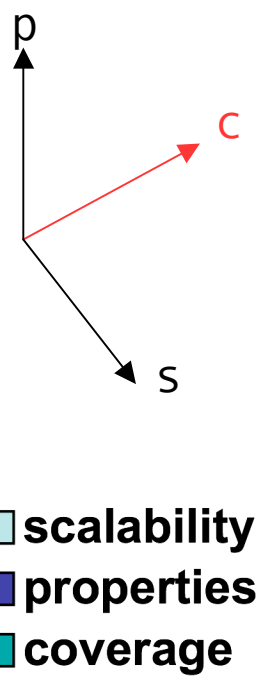
specification and programming



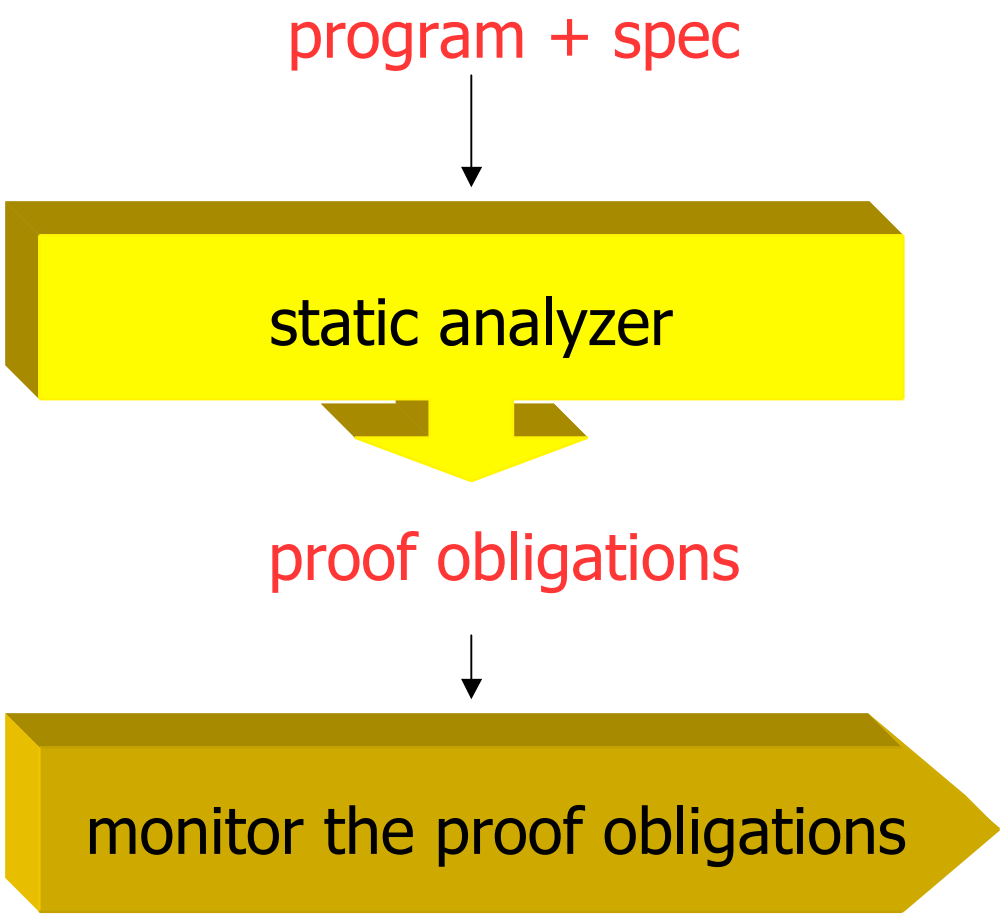
comparison of techniques

we are giving up on coverage
to write better specifications
and scale



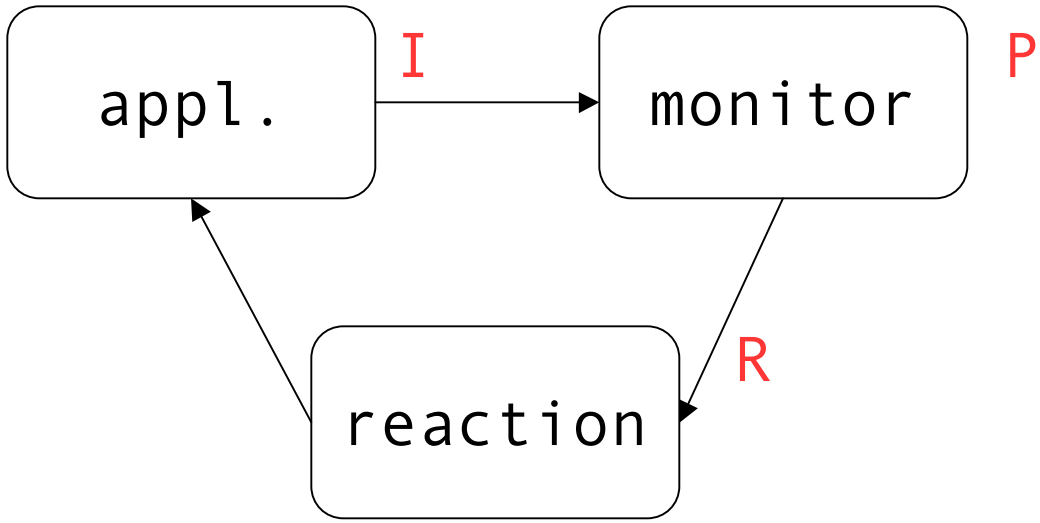


combining static and dynamic analysis



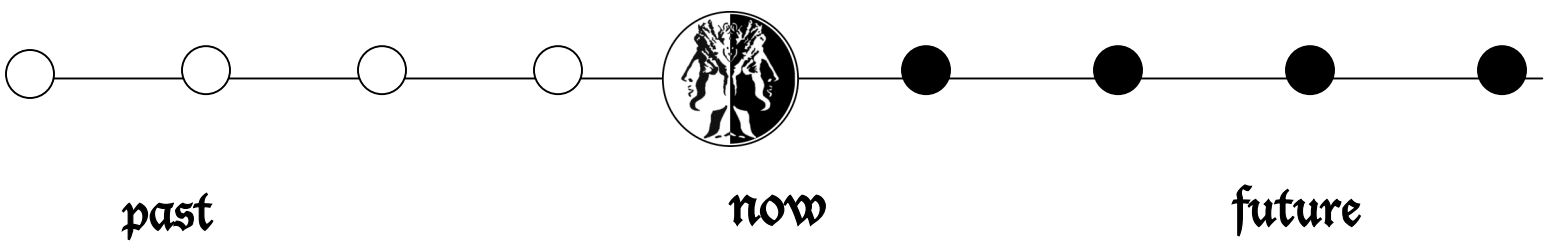
components

- instrumentation language (I)
- property specification language (P)
- reaction language (R)



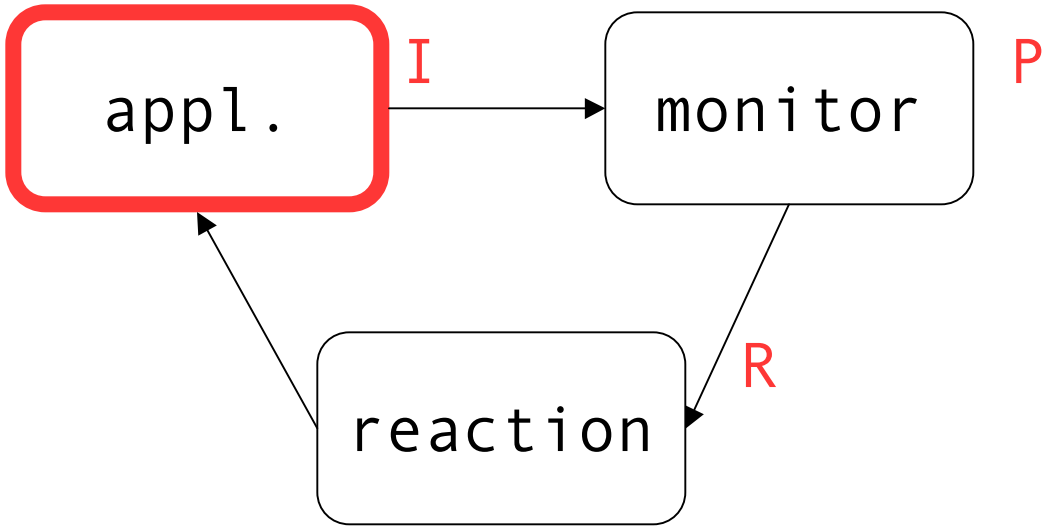
a trace view of execution

- a formal view of an execution is to consider it as a sequence σ of program states: $\sigma = s_1\ s_2\ s_3\ \ldots\ s_n$
- during program execution we are at any point in time in the present moment now where the past is known and the future is unknown.



instrumentation

- instrumentation language (I)
- property specification language (P)
- reaction language (R)



trace generation

```
while(true){
  if(x>0)lock(L);
  x = shared;
  shared = f(x);
  unlock(L);
}
```

trace generation

```
while(true){
  if(x>10){
    lock(L);
    logLock(t,L);
  }
  x = shared;
  shared = f(x);
  release(L);
  logRelease(t,L);
}
```

Instrument program.
For example using
Aspect Oriented Programming

execute

Trace:

```
,L)
release(t2,L)
release(t2,L)
lock(t2,L)
release(t2,L)
lock(t2,L)
```

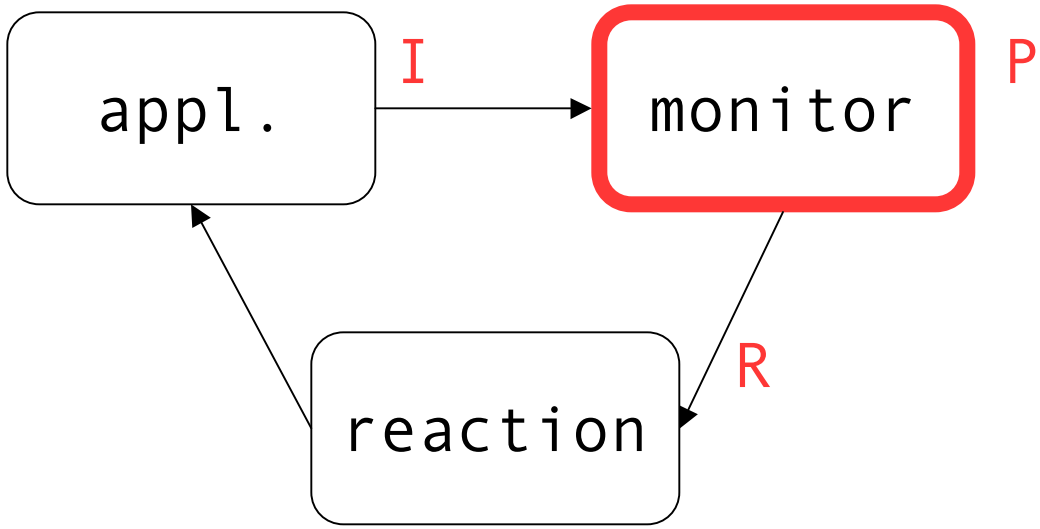
monitor

instrumentation

- manual
  - assertions
  - pre/post conditions in design by contract solutions
- automated
  - instrumentation of source code:
    - CIL (C) http://manju.cs.berkeley.edu/cil/
  - instrumentation of byte/object code
    - BCEL (Java) http://jakarta.apache.org/bcel/
    - Valgrind (C) http://valgrind.org
  - High level bytecode APIs
    - Sofya http://sofya.unl.edu
  - aspect orinted programming: this course
    - AspectJ (Java) http://www.eclipse.org/aspectj
    - AspectC (C) http://research.msrg.utoronto.ca/ACC
    - AspectC++ (C++) http://www.aspectc.org

monitor

- instrumentation language (I)
- property specification language (P)
- reaction language (R)



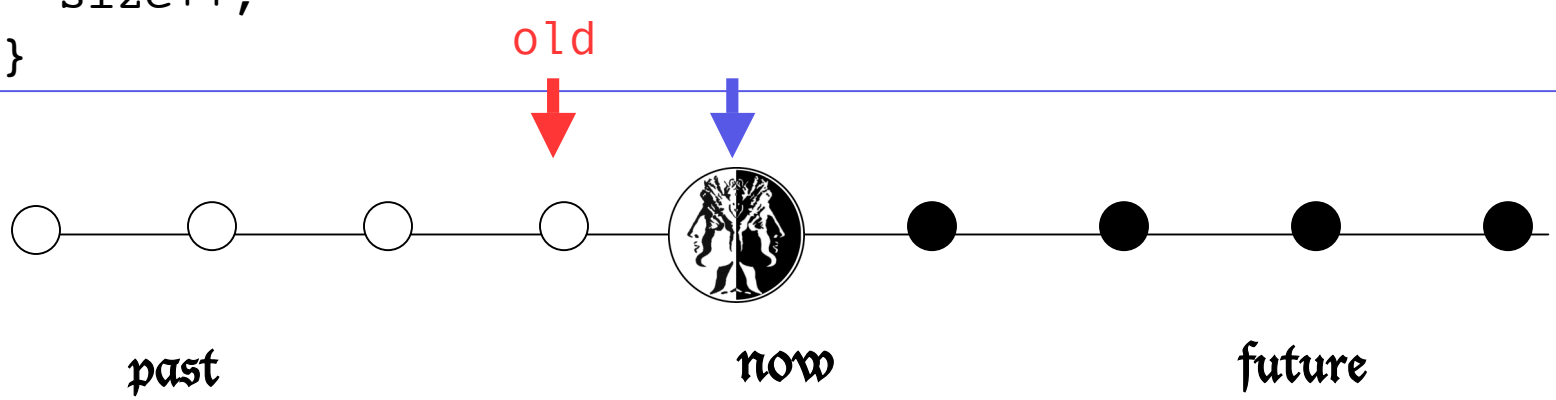
property language

- programming language
- program (built-in algo. focused on specific problem)
  - data race detection
  - atomicity violation
  - deadlock detection
- logic (formal system)
  - design by contract (pre/post conditions), JML for example
  - state machines
  - regular expressions
  - grammars (context free languages)
  - temporal logic (past time, future time)
  - csp/ccs (process algebra)
  - “super languages” combining several of these
  - full fledged formal specification languages (Z, VDM, …)
  - graphical languages (UML, …)

design by contract (JML)

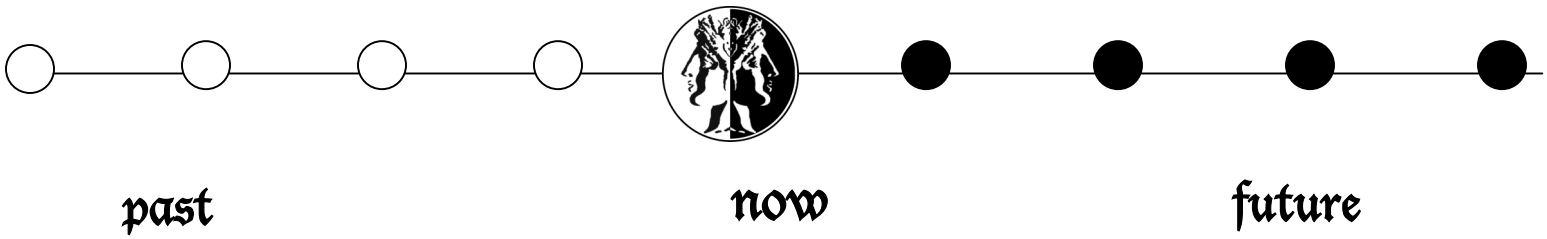
```
//@ public invariant 0 <= size;
/*@ requires size < elems.length-1;
  @ assignable elems[size], size;
  @ ensures size == \old(size+1);
  @ ensures elems[size-1] == x;
  @ ensures_redundantly
  @   (\forall int i; 0 <= i && i < size-1;
  @       elems[i] == \old(elems[i]));
  @*/

public void push(Object x) {
  elems[size] = x;
  size++;
}
```



what about the rest of the trace?

- pre/post conditions are not designed for total trace view, only “now” and “previous” (old) state.
- one can of course encode a total trace view using history variables.
- but that can be rather inconvenient.

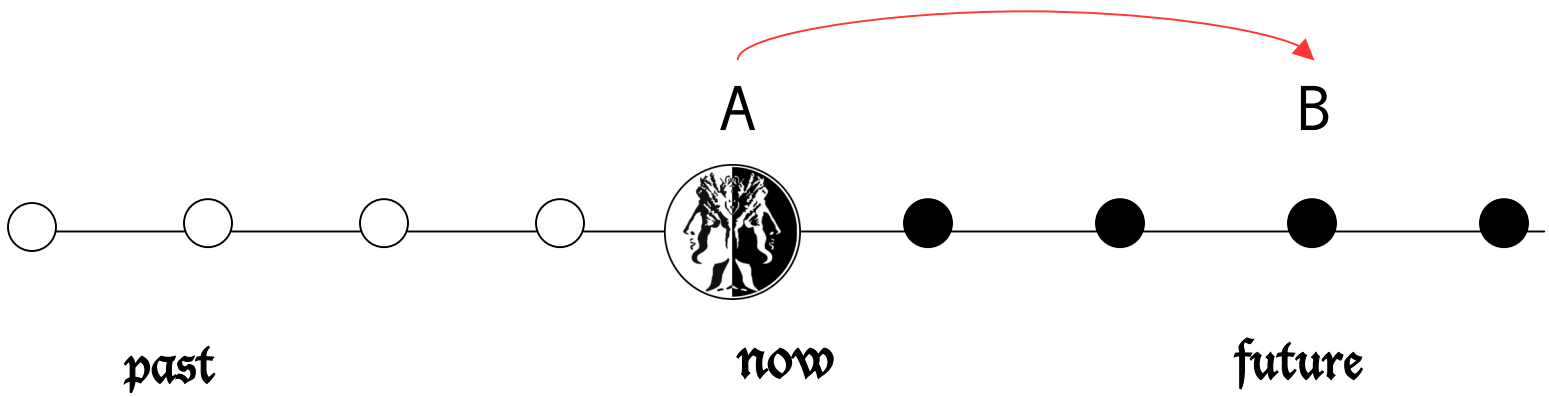


future time properties

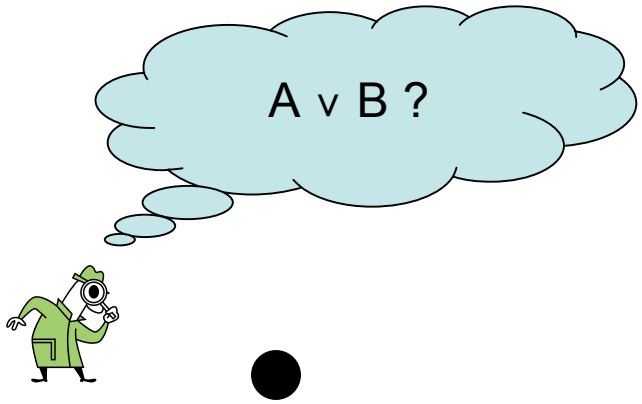
- If A happens now

  B must happen

$$\Box(A \rightarrow \Diamond B)$$



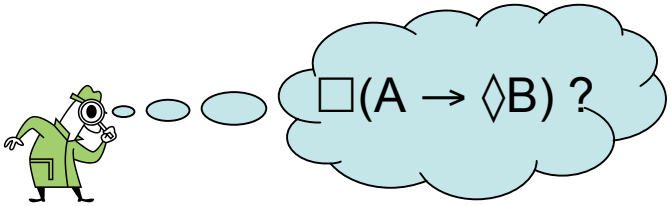
logic

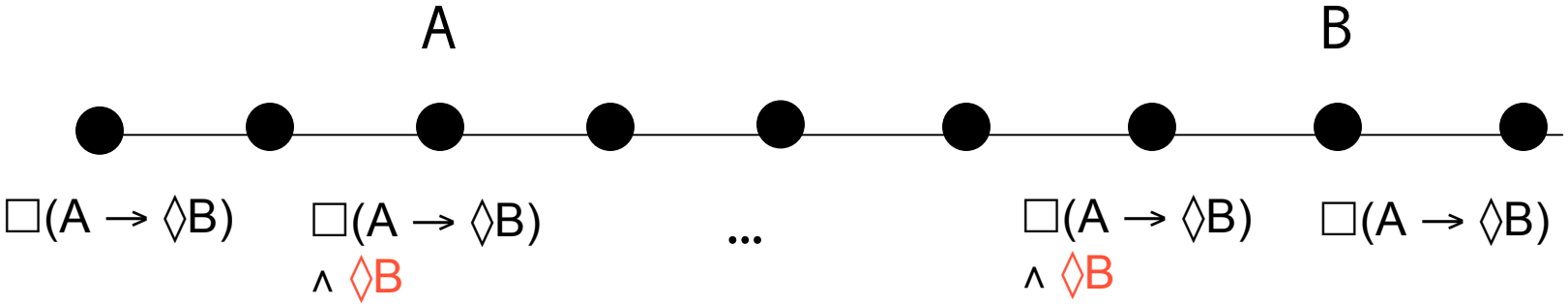


Boolean logic allows us to formulate statements about a single world/situation : now

$p \wedge q$	“p and q”
$p \vee q$	“p or q”
$p \rightarrow q$	“p implies q”
$\neg p$	“not p”

future time temporal logic





$p \wedge q$	"p and q"
$p \vee q$	"p or q"
$p \rightarrow q$	"p implies q"
$\neg p$	"not p"

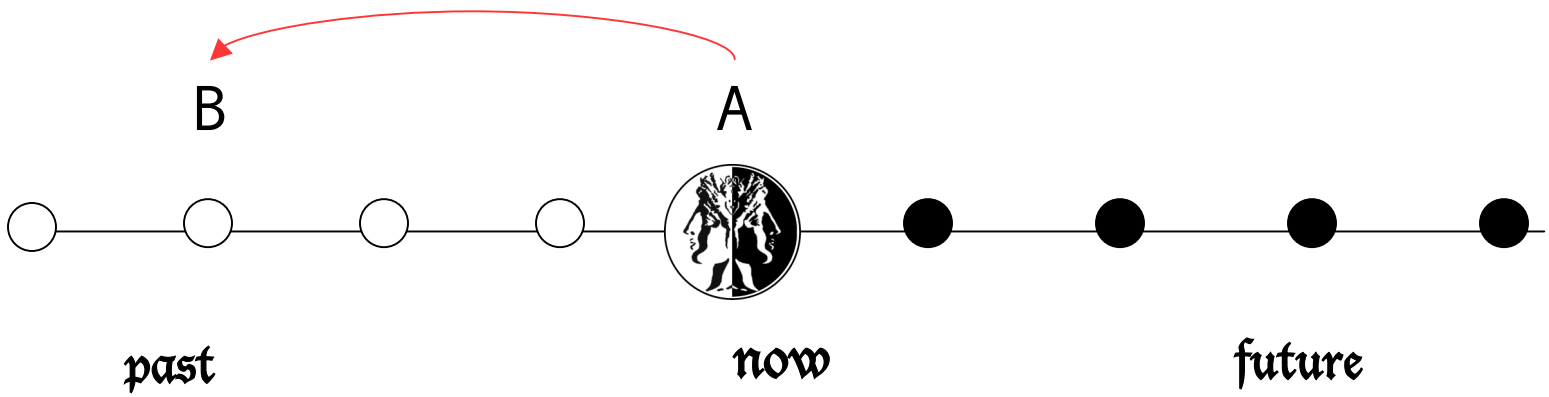
$\Box p$	"always p"
$\Diamond p$	"eventually p"
$p \mathrm{U} q$	"p until q"

past time properties

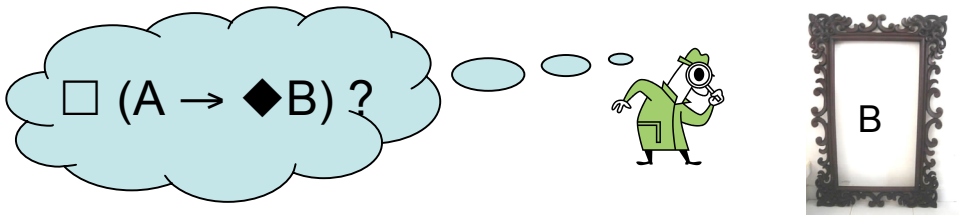
- If A happens now

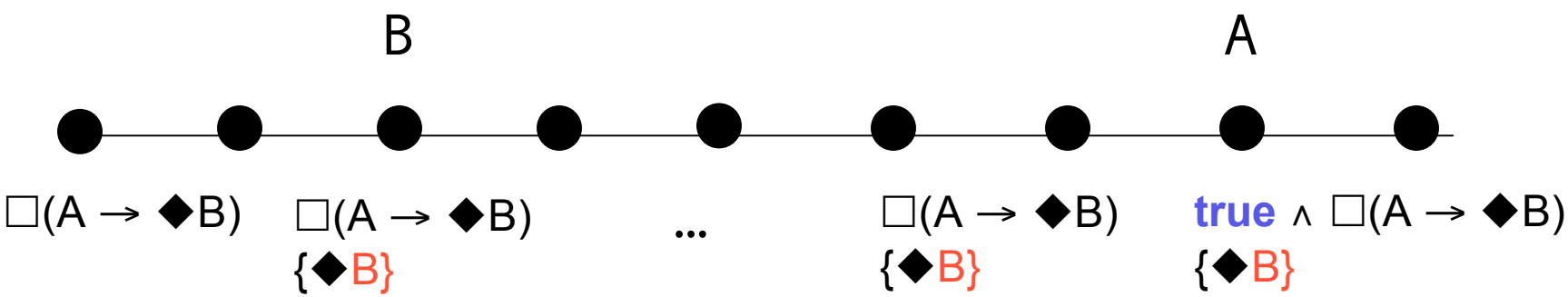
  B must have happened

$$\Box(A \rightarrow \blacklozenge B)$$



past time temporal logic



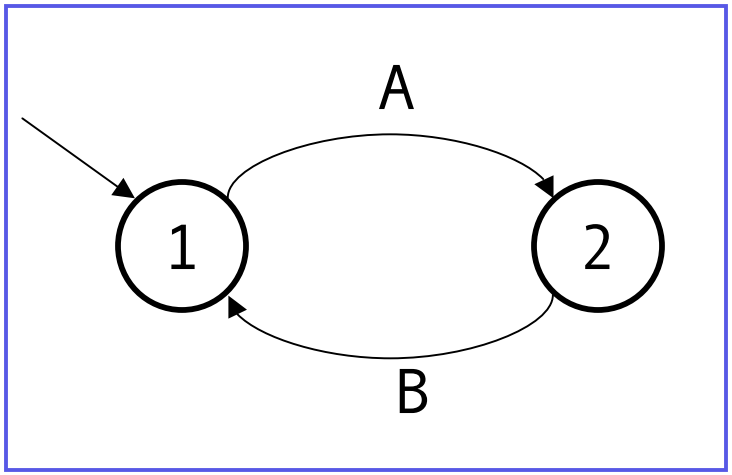


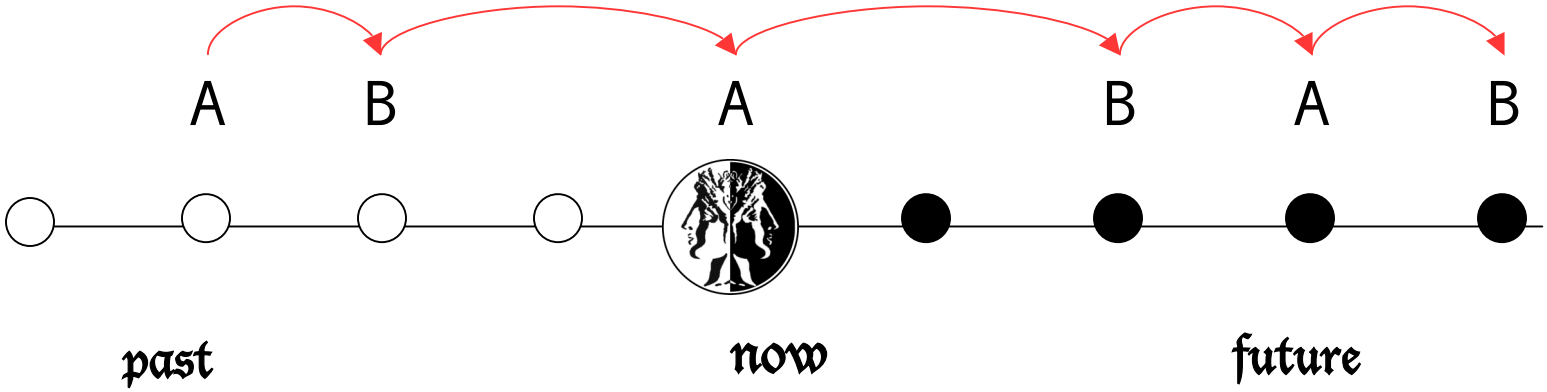
p ∧ q	"p and q"
p ∨ q	"p or q"
p → q	"p implies q"
¬p	"not p"

■p	"sofar p"
◆p	"previously p"
p S q	"p since q"

state machines

- A and B should alternate

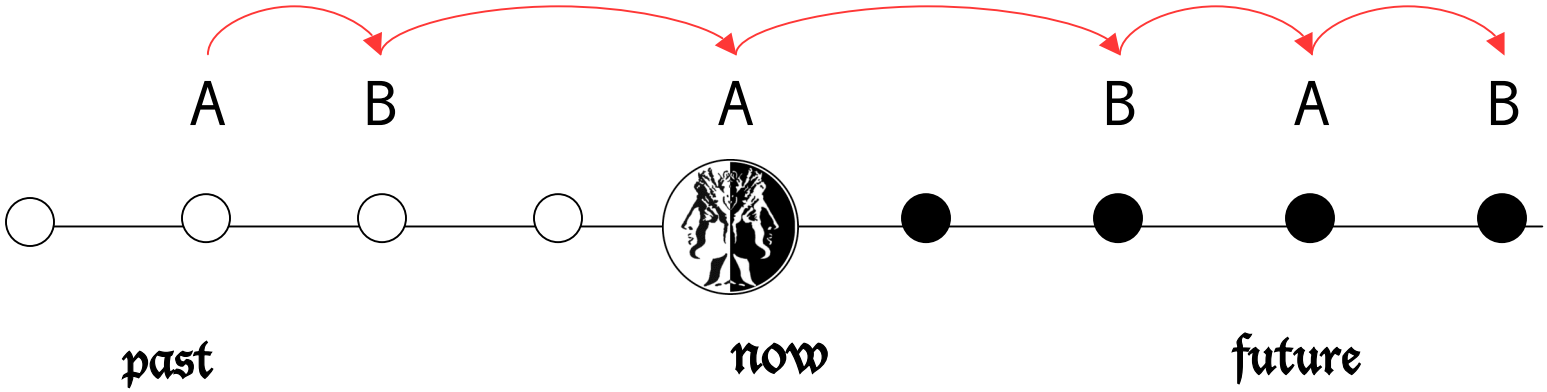




regular expressions

- A and B should alternate


```
(A B)*
```

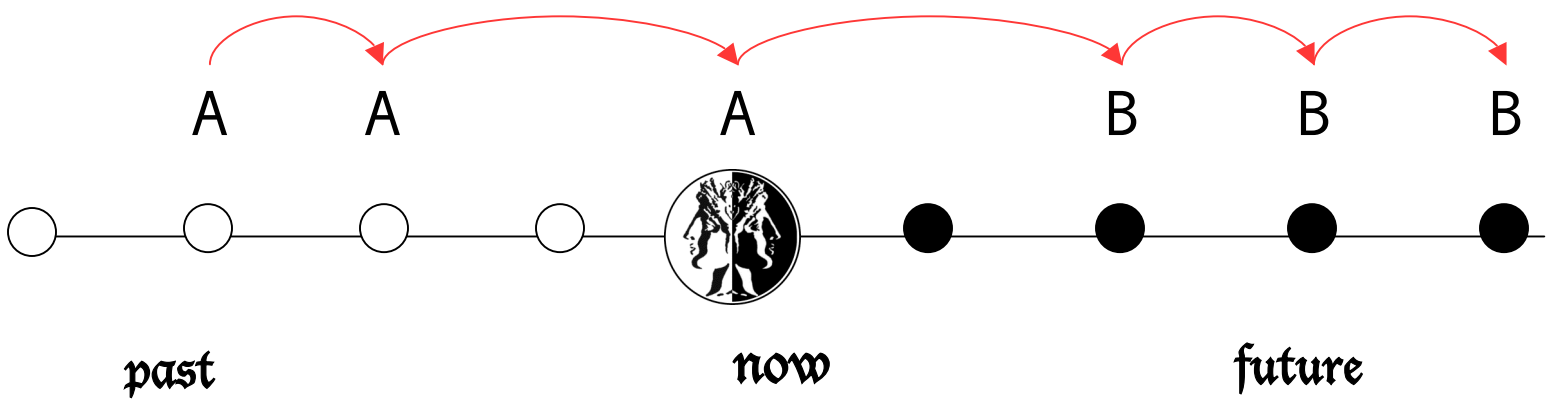


context free grammars

- $A^n B^n$

```
unary_expression
        : postfix_expression
        | INC_OP unary_expression
        | DEC_OP unary_expression
        | unary_operator cast_expression
        | SIZEOF unary_expression
        | SIZEOF '(' type_name ')'
        ;
```

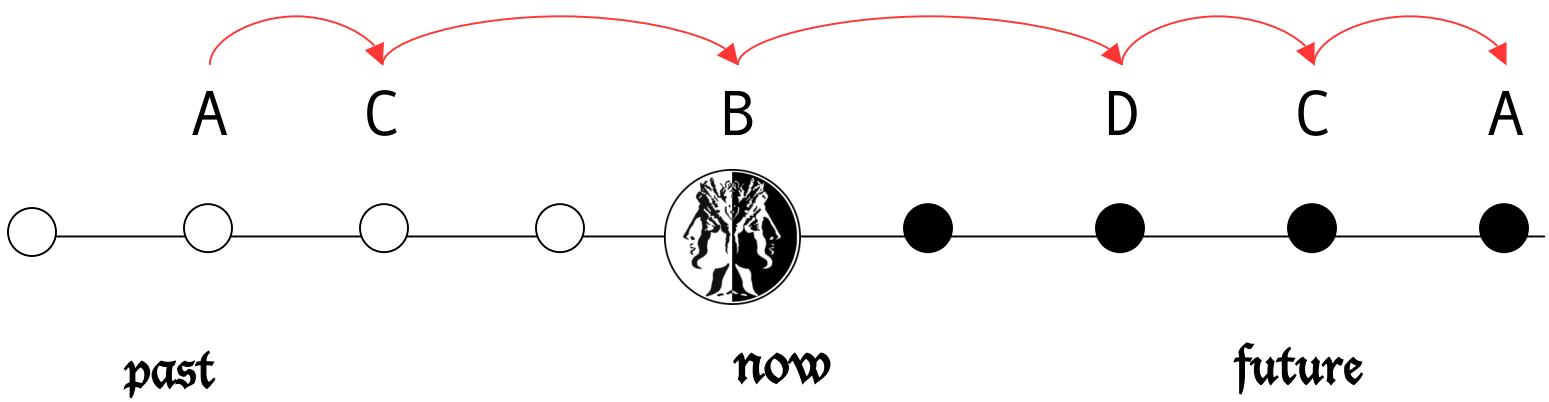
$$S \rightarrow A\ S\ B \mid \epsilon$$



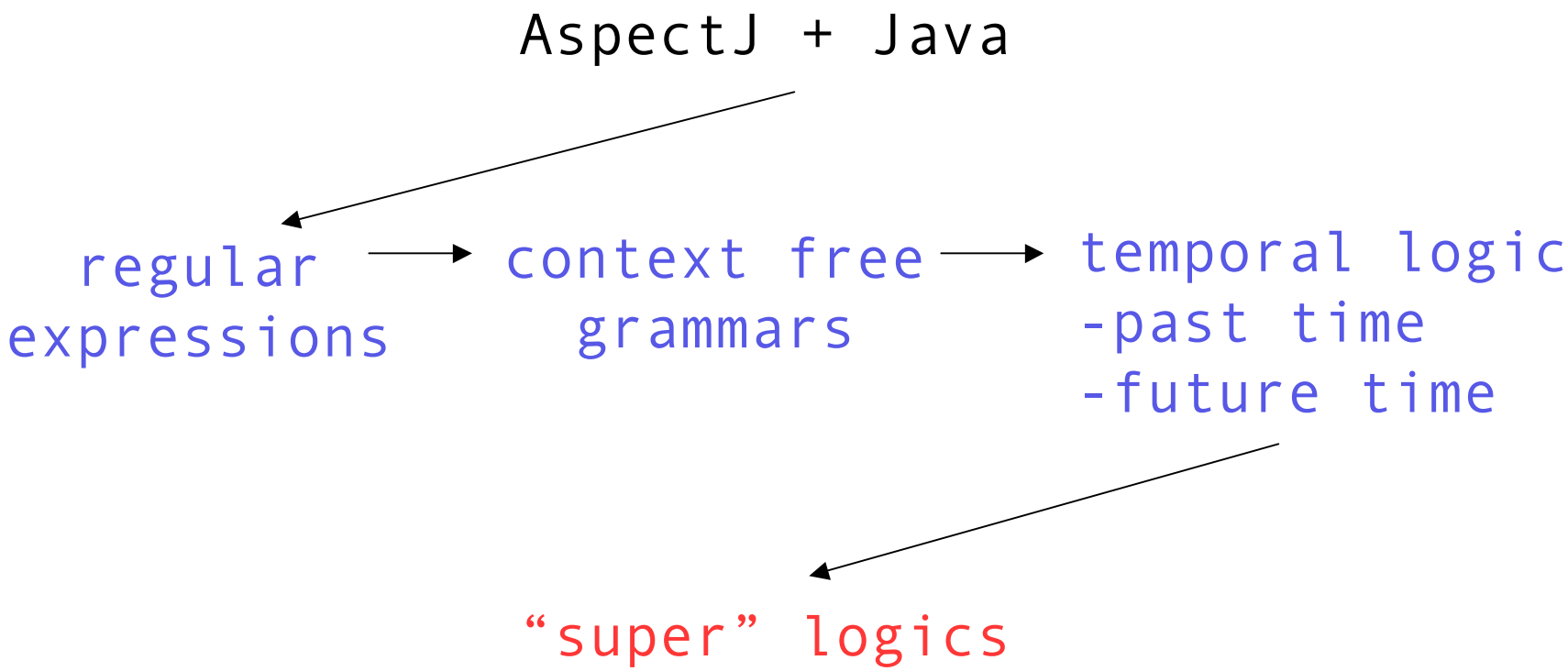
process algebra

- AB randomly interleaved with CD

$$
\begin{aligned}
p &= q \parallel r \\
q &= A \rightarrow B \rightarrow q \\
r &= C \rightarrow D \rightarrow r
\end{aligned}
$$



this course


AspectJ + Java
regular expressions
context free grammars
temporal logic
-past time
-future time
“super” logics

expressiveness

Java

“super” logics

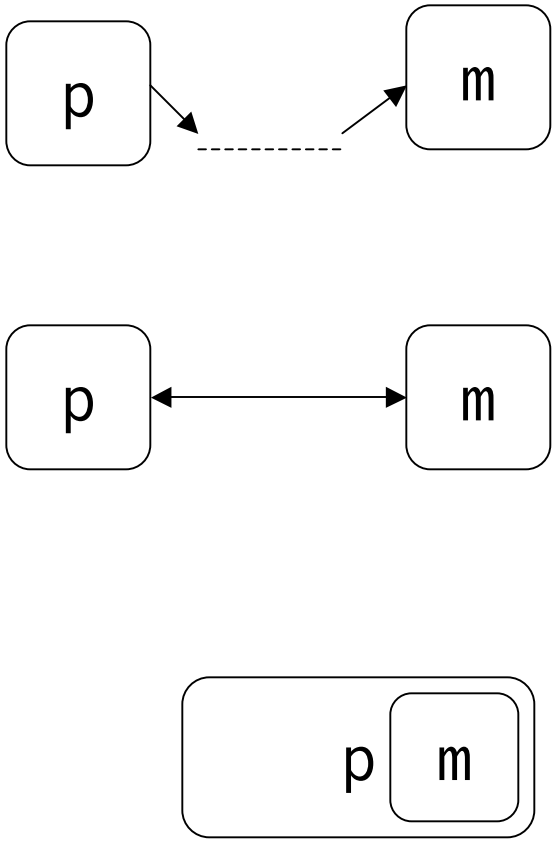
context free grammars

regular expressions

temporal logic
-past time
-future time

monitor integration

- offline
  - analyzing log file / trace dump
- online
  - outline
    - monitor runs in parallel with application.
    - communication for ex. over socket.
      - synchronous (appl. waits for response)
      - asynchronous (buffered communication)
  - inline
    - monitoring code is embedded into the application



violation versus validation

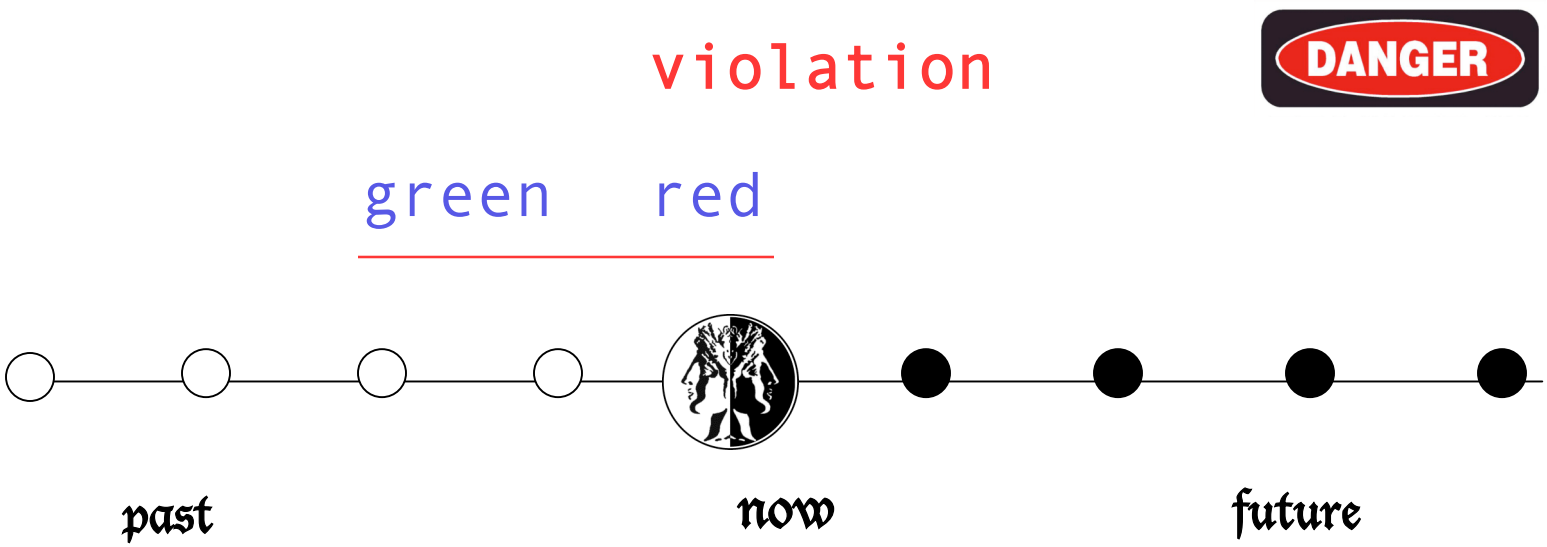
- violation : the normal case, checking that the system conforms to a property. A violation is reported when the property is violated.

- validation :
  - it can sometimes be easier to state property in negative form: what we do not want to happen. Hence we want reported when the bad property gets "validated".
  - or: it is a good property and we just want to log whenever something good happens

`most systems can only do one of the two forms`

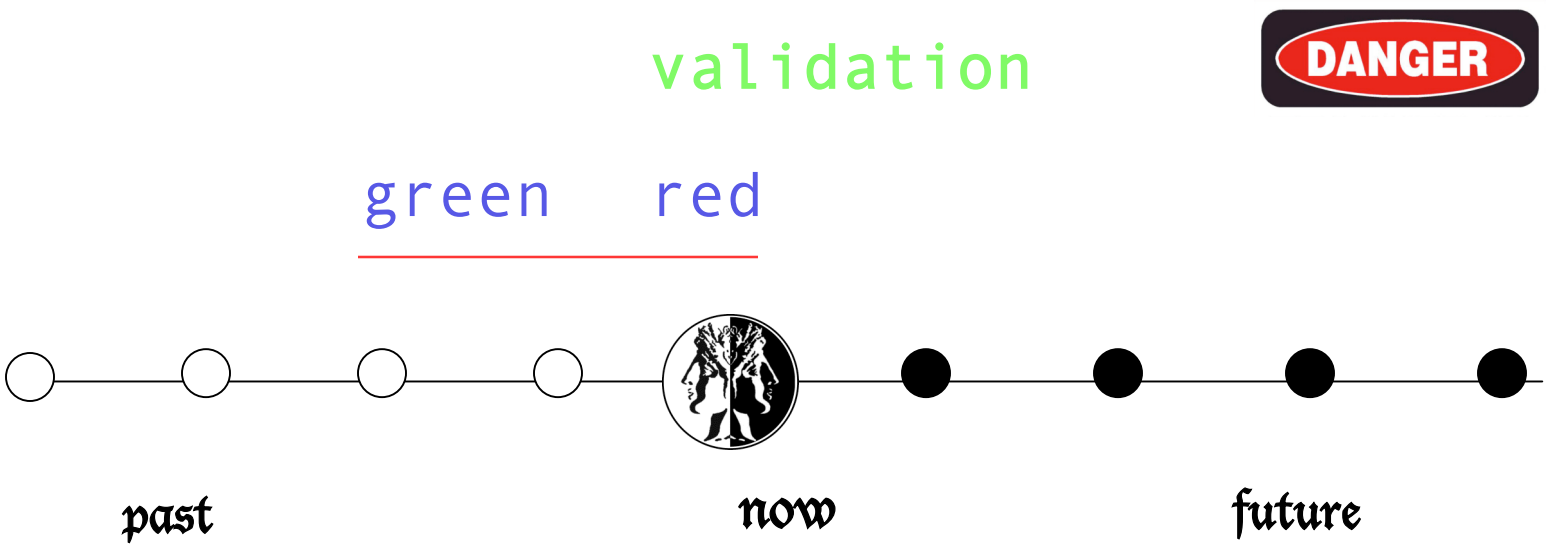
violation example

(green yellow red)*



validation example

green red

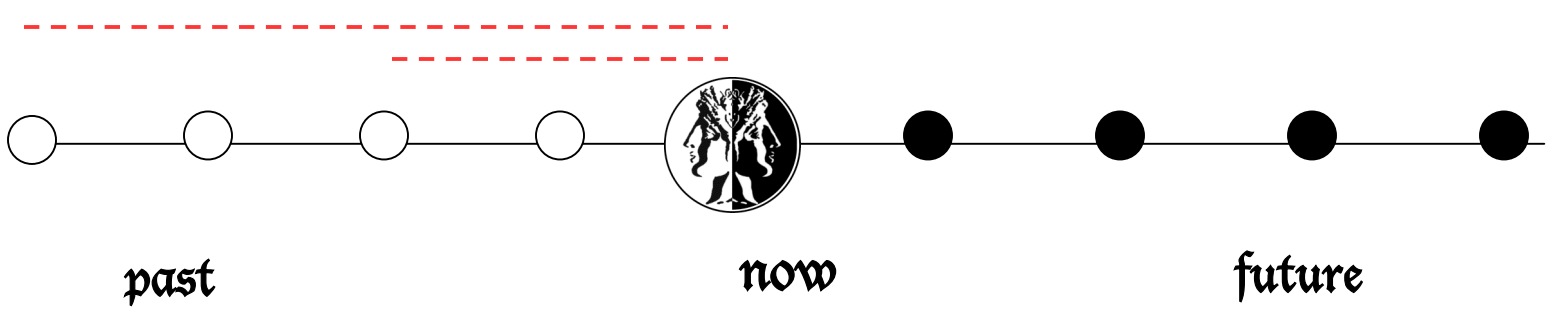


violation $\phi \neq$ validation $\neg \phi$

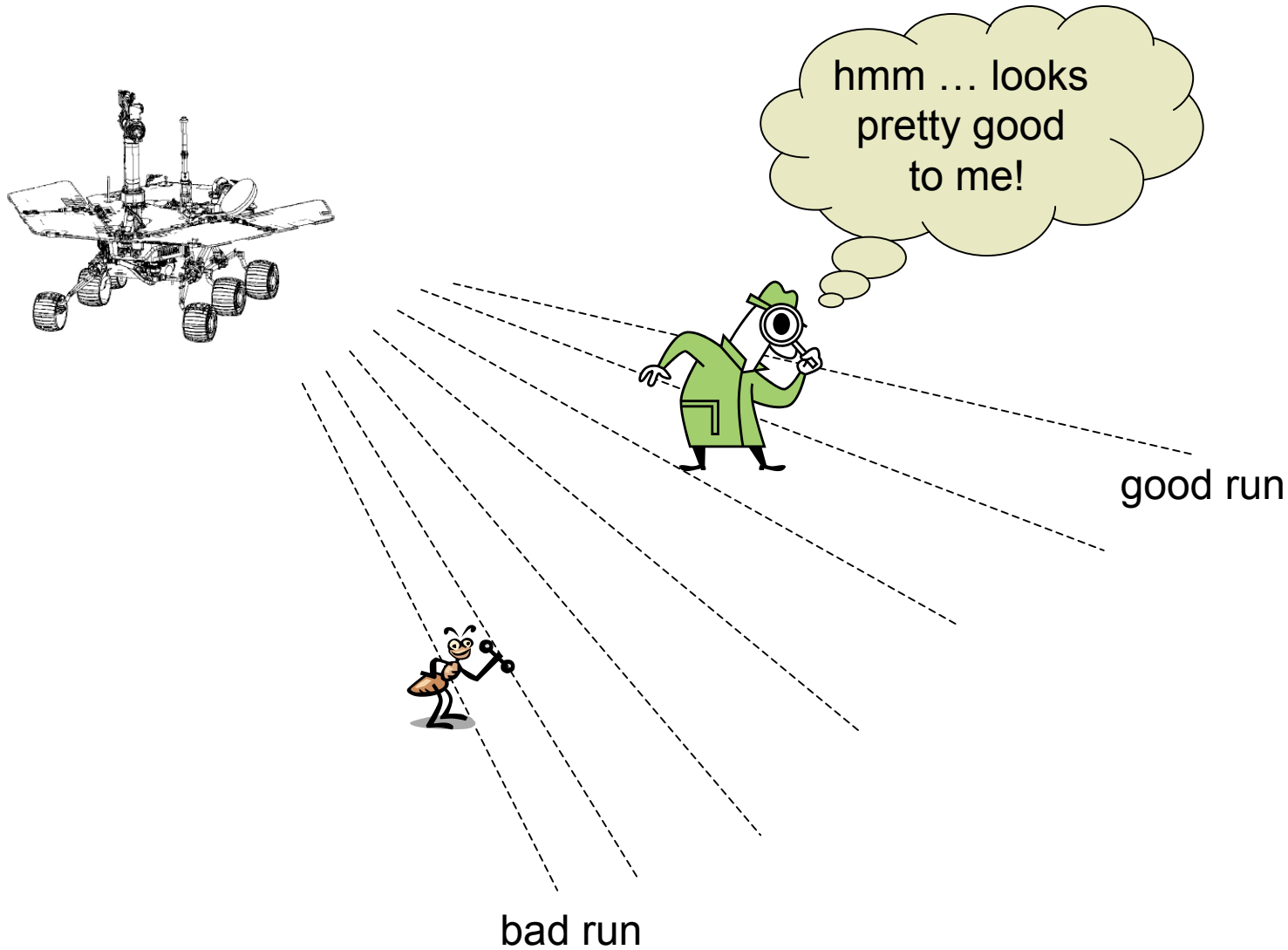
- some systems support both concepts
- one cannot achieve validation semantics just be negating a violation formula. example:
  - suppose our system can only check validation
  - violate ab $\neq$ validate ~(ab).
  - reason: observe the first ‘a’ in the trace ‘ab’.
    - it validates ~(ab)
    - it does not violate ab yet (and will not in this case).
- some logics do not have negation (context free grammars for example)

total trace versus trace suffix semantics

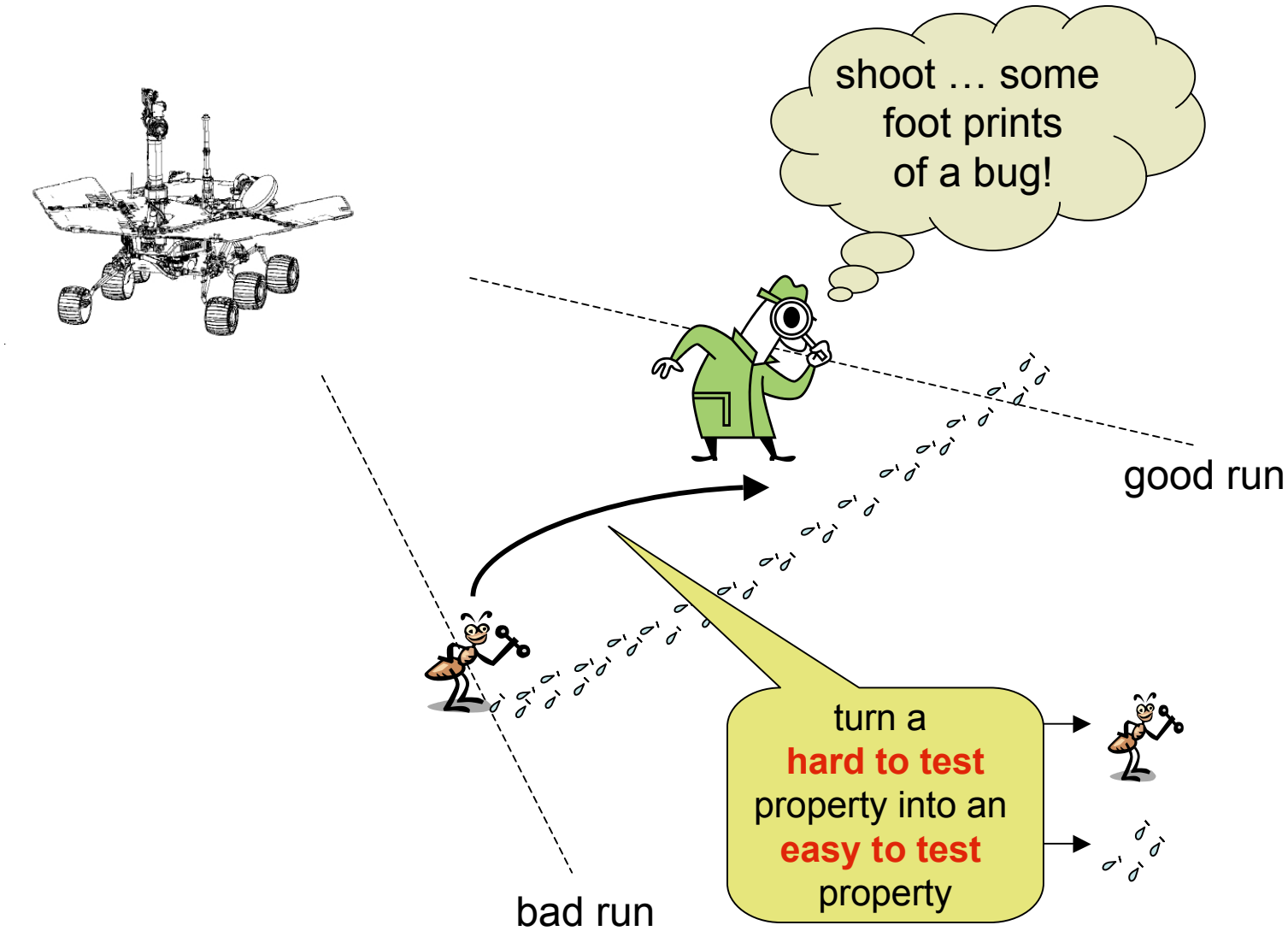
- total trace semantics: property must be violated/validated on the whole trace
- suffix trace semantics:
  - validation : there exists a rightmost suffix for which it holds
  - violation : hard to make sense of



predictive analysis



predictive analysis



factual versus predictive

- factual
  - the standard case: violation/validation is what it is:

    violation/validation(σ) $\Rightarrow$ error (σ)

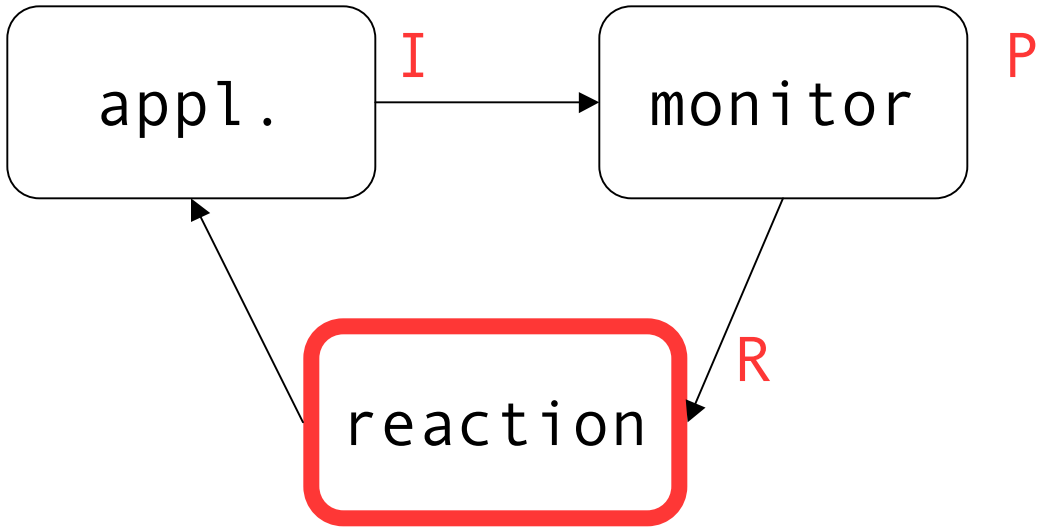
- predictive
  - violation/validation is only suggestive, indicating the potential for an error in some **other** trace of the monitored system:

    violation/validation(σ) $\rightarrow$

    $\exists\ \sigma'$: Traces $\bullet$ error(σ')

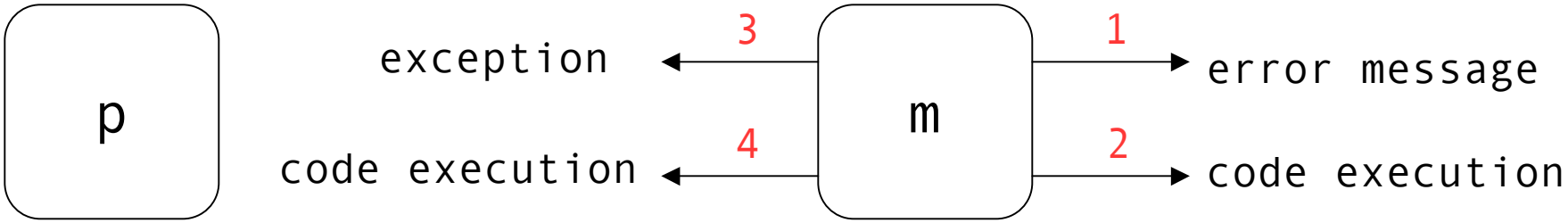
reaction

- instrumentation language (I)
- property specification language (P)
- reaction language (R)



interaction degree

1. error message
2. separate code execution (not interacting with monitored program, not same memory space)
3. exception in monitored program, and monitored program then deals with it
4. integrated code execution (access to memory). Here the monitor defines how to change the behavior of the monitored program



systems for Java

- Java's assert statements, Java expressions:
  - http://java.sun.com/j2se/1.5.0/docs/guide/language/assert.html
- JML, pre/post conditions:
  - http://www.eecs.ucf.edu/~leavens/JML/index.shtml
- jContractor, pre/post conditions:
  - http://jcontractor.sourceforge.net/
- Temporal Rover (commercial), metric future time temporal logic/UML:
  - http://www.dbrover.com/
- Java MaC, past time temporal logic:
  - http://rtg.cis.upenn.edu/mac/index.php3
- JLO, future time temporal logic:
  - http://www.sable.mcgill.ca/~ebodde/rv//JLO
- Tracematches, regular expression suffix-validation:
  - http://abc.comlab.ox.ac.uk/papers
- PQL, context free grammars, suffix-validation:
  - http://pql.sourceforge.net/
- PTQL, SQL, suffix-validation:
  - http://techreports.lib.berkeley.edu/accessPages/CSD-04-1315

systems for Java

- LOLA:
  - http://www-step.stanford.edu/papers/time05.html
- T-UPPAAL, timed automata:
  - http://www.cs.aau.dk/~marius/tuppaal
- Jass, CSP (process algebra, pre/post conditions):
  - http://csd.informatik.uni-oldenburg.de/~jass
- Eagle/Hawk, recursion-based “super logic”:
  - http://yangtze.cs.uiuc.edu/~ksen/eagle

- Java MOP, plugin for different logics:
  - http://fsl.cs.uiuc.edu/index.php/JavaMOP
- RuleR, rule-based “super logic” (second attempt after Eagle):
  - http://www.cs.man.ac.uk/~howard/Logics%20for%20Program%20Analysis.html

this course

systems for C

- AspectC:
  - http://research.msrg.utoronto.ca/ACC
- Arachne, simple subset of regular expressions:
  - http://www.emn.fr/x-info/arachne/index.html

predictive systems

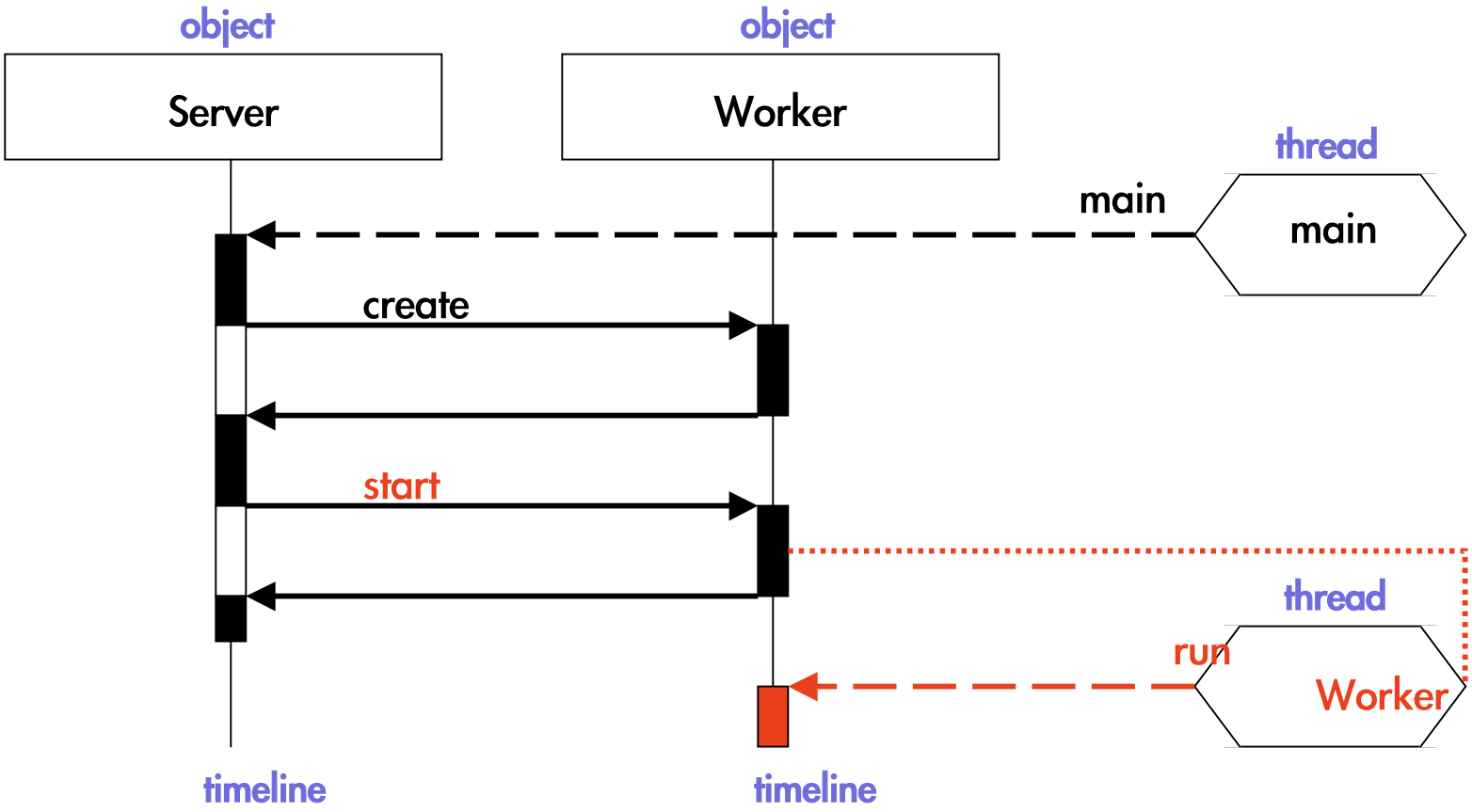
- Visual threads (Eraser)
  - http://h21007.www2.hp.com/portal/site/dspp/PAGE.template/page.document?ciid=e608bce06ee02110bce06ee02110275d6e10RCRD
- jPredictor
  - http://fsl.cs.uiuc.edu/index.php/JPredictor
- Atomicity

learning specs from runs

- DAIKON, learning JML:
  - http://groups.csail.mit.edu/pag/daikon/
- Perecotta, learning temporal formulas:
  - http://www.cs.virginia.edu/perracotta
- Redux : learning call graphs etc.
  - Allan Mycroft et. al.

program visualization

Thread main (polygon) executes method main in the Server Object (rectangle).
Method main creates a Worker object and calls start on it.



After the call of start on the Worker object the Worker thread can run and eventually it does. The run can happen anytime after start.

formalizing concept of monitor

Alphabet:
Given an alphabet (set) A of symbols

Language:
A language $L \subseteq A^*$ over A is a subset of A^*.

Property:
A property P over A is a language:

$$P \subseteq A^*$$

examples

formula φ	language $L(\varphi)$
$\Box(a \rightarrow \Diamond b)$	$\{\epsilon, cc, b, ab, aab, \ldots\}$ liveness
$\Box(a \rightarrow \blacklozenge b)$	$\{\epsilon, c, ba, baa, baba, \ldots\}$ safety
$(ab)^*$	$\{\epsilon, ab, abab, ababab, \ldots\}$
$S \rightarrow a\ S\ b \mid \epsilon$	$\{\epsilon, ab, aabb, aaabbb, \ldots\}$

a formula in a logic compacts an infinite language

safety properties

Safety property:
A property $P \subseteq A^*$ is a safety property if and only if P is prefix closed:

$$\text{prefix}(P) = P$$

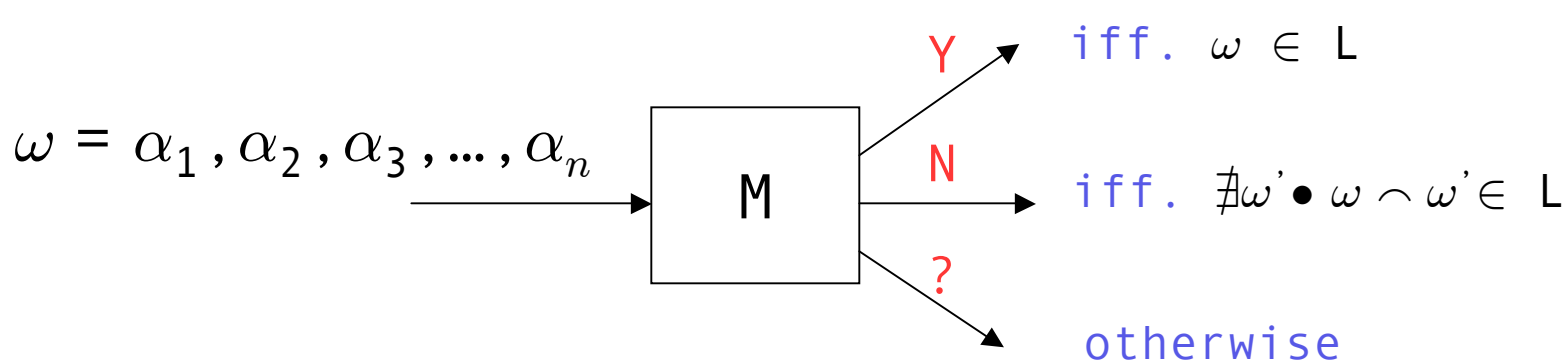
where $\text{prefix}(L) = \{\sigma \mid \exists\, \sigma' \bullet \sigma \frown \sigma' \in L\}$

intuitively: if a trace is safe then all prefixes of the trace are safe.
This is for example not true for 'aab' in the language P of $\Box(a \rightarrow \Diamond\, b)$.
Here 'aa' is for example not in P.

monitoring a language

A monitor:

A monitor M for a language $L \subseteq A^*$ is a "box" which as input takes a list of symbols $\alpha \in A$, one by one, and emits a value for the next symbol α_n as follows:



the monitoring problem for a logic $\mathcal{L}$

The monitoring problem:

Given a logic $\mathcal{L}$.
Given a formula $\varphi \in$ Formulas($\mathcal{L}$).

Construct a monitor for the language of φ:

$$M_{L(\varphi)}$$

For $\mathcal{L} \in$ {RE,CFG,PTTL,FTTL,…}

challenges

- specification-based monitoring
  - definition of specification languages
    - succinct
    - easy to adopt
    - expressive
  - creation of efficient monitors from specs
  - minimize impact on monitored system
  - integrate static and dynamic analysis
  - how to control application in case of violation/validation (fault protection)
- predictive analysis/learning: what can one conclude from good runs
- how to instrument code (languages for specifying relevant events, reactions)

overview of course

1. introduction
2. the AspectJ language (AspectJ)
3. using AspectJ for monitoring (AspectJ)
4. propositional regular expressions
5. parameterized regular expressions (JavaMOP)
6. context free grammars (JavaMOP)
7. temporal logic (JavaMOP)
8. rule-based systems (RuleR)
9. predictive analysis

properties of Java library APIs

Iterator (Java 2 Platform SE 5.0)

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

Joergen Ingman | TimeLife.com | Micro Site | Eventseer.net - Home | Free Website Polls | Richest Cou...n the World | Castle for Sale | Programming with C/C++ | Costa Rica, ...Estate Blogs | BASH Help - ...sh Tutorial

Try JavaMOP Online - FSL | Iterator (Java 2 Platform...

Java™ 2 Platform Standard Ed. 5.0

All Classes

Packages
- java.applet
- java.awt
- java.awt.color
- java.awt.datatransfer
- java.awt.dnd
- java.awt.event
- java.awt.font

IOException
IOR
IORHelper
IORHolder
IORInfo
IORInfoOperations
IORInterceptor
IORInterceptor_3_0
IORInterceptor_3_0Helper
IORInterceptor_3_0Holder
IORInterceptor_3_0Operations
IORInterceptorOperations
IRObject
IRObjectOperations
IstringHelper
ItemEvent
ItemListener
ItemSelectable
Iterable
Iterator
IvParameterSpec
JApplet
JarEntry
JarException
JarFile
JarInputStream
JarOutputStream
JarURLConnection
JButton
JCheckBox
JCheckBoxMenuItem
JColorChooser
JComboBox
JComboBox.KeySelectionManager

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™ 2 Platform Standard Ed. 5.0

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.

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	hasNext() — Returns true if the iteration has more elements.
E	next() — Returns the next element in the iteration.
void	remove() — Removes from the underlying collection the last element returned by the iterator (optional operation).

Method Detail

hasNext

http://www.eclipse.org/aspectj

The AspectJ Project

http://www.eclipse.org/aspectj/

eclipse

HOME | COMMUNITY | MEMBERSHIP | COMMITTERS | DOWNLOADS | RESOURCES | PROJECTS | ABOUT US

- AspectJ
  - About
  - Team
  - Plans
- Users
  - Downloads
  - Docs
  - Bugs
  - FAQ
  - Mailing lists
- Developers
  - CVS
  - Mailing list

aspectj *crosscutting objects for better modularity*

aspectj is

- a seamless aspect-oriented extension to the Java[tm] programming language
- Java platform compatible
- easy to learn and use

aspectj enables

- clean modularization of crosscutting concerns, such as error checking and handling, synchronization, context-sensitive behavior, performance optimizations, monitoring and logging, debugging support, and multi-object protocols

Quick Links

- For Eclipse development: **AJDT: The AspectJ Development Tools**
- Popular AspectJ downloads: **Latest development build** | **Latest stable release** | **More downloads...**
- Popular AspectJ docs: **AspectJ 5 Developer's Notebook** | **Programming Guide** | **More docs...**
- **Eclipse AspectJ:** the book, by some of the leading AspectJ committers
- **AOSD.net**
- **AspectJ PARC Page**

monitoring with AspectJ

"an access should be preceded by an authentication"

application

```
public class Resource {
  void authenticate() {…}
  void access() {…}
}
```

aspect

```
aspect MonitorAspect {
  boolean authenticated = false;

  before() : call(void Resource.authenticate()) {
    authenticated = true;
  }

  before() : call(void Resource.access()) {
    if (!authenticated)
      error("access without authentication");
  }
}
```

http://fsl.cs.uiuc.edu/index.php/JavaMOP

JavaMOP - FSL

http://fsl.cs.uiuc.edu/index.php/JavaMOP

Joergen Ingman | TimeLife.com | Micro Site | Eventseer.net - Home | Free Website Polls | Richest Cou...n the World | Castle for Sale | Programming with C/C++ | Costa Rica, ...Estate Blogs | BASH Help - ...sh Tutorial

JavaMOP - FSL

Log in / create account

Formal Systems Laboratory

navigation

- Main Page
- People
- Projects
- Publications
- News
- Events

search

Go Search

JavaMOP

JavaMOP is an MOP tool for Java. It has a client-server architecture for better portability: the client is a pure Java application and runs on any platform supporting Java and HTTP; the server prefers Linux. The client has two user interfaces: a command-line one good for batch processing and a more friendly one based on Eclipse. Before downloading and installing the tool, you can *Try it online!*

Related projects: Monitoring-Oriented Programming; MOP logic repository

Contents [hide]

News and Change Logs

[edit]

- **2007-8-2:** JavaMOP version 1.2.1 was released
  - ***Important***: JavaMOP users please update your packages, because the server URLs have been changed!
  - Fixed some bugs (thanks to Pavel Avgustinov)
- **2007-3-18:** JavaMOP version 1.2 was released
  - Support of ***parametric*** specifications provided -- see our online demo for examples
  - MOP specification syntax updated
  - Please download the latest version of JavaMOP
- **2006-11-16:** JavaMOP version 1.1.1 was released
  - Automatic indentation added to generated code
  - Bugs for generating monitors from ERE specifications fixed
  - Bugs for initialization of check point specifications fixed
  - Special thanks to Ylies Falcone for several bug reports and suggestions
- **2006-11-06:** JavaMOP version 1.1 was released with the following changes:
  - The JavaMOP specification syntax slightly changed
  - A new web-based interface provided
  - JavaMOP client applications improved. Note: old clients do ***not*** work any longer
- **2006-05-10:** JavaMOP version 1.0 was released

http://fsl.cs.uiuc.edu/index.php/JavaMOP

JavaMOP - FSL

http://fsl.cs.uiuc.edu/index.php/JavaMOP

Joergen Ingman | TimeLife.com | Micro Site | Eventseer.net - Home | Free Website Polls | Richest Cou...n the World | Castle for Sale | Programming with C/C++ | Costa Rica, ...Estate Blogs | BASH Help - ...sh Tutorial

JavaMOP - FSL

Log in / create account

Formal Systems Laboratory

navigation

- Main Page
- People
- Projects
- Publications
- News
- Events

search

Go Search

JavaMOP

JavaMOP is an MOP tool for Java. It has a client-server architecture for better portability: the client is a pure Java application and runs on any platform supporting Java and HTTP; the server prefers Linux. The client has two user interfaces: a command-line one good for batch processing and a more friendly one based on Eclipse. Before downloading and installing the tool, you can *Try it online!*

Related projects: Monitoring-Oriented Programming; MOP logic repository

Contents [hide]

News and Change Logs

[edit]

- **2007-8-2:** JavaMOP version 1.2.1 was released
  - ***Important***: JavaMOP users please update your packages, because the server URLs have been changed!
  - Fixed some bugs (thanks to Pavel Avgustinov)
- **2007-3-18:** JavaMOP version 1.2 was released
  - Support of ***parametric*** specifications provided -- see our online demo for examples
  - MOP specification syntax updated
  - Please download the latest version of JavaMOP
- **2006-11-16:** JavaMOP version 1.1.1 was released
  - Automatic indentation added to generated code
  - Bugs for generating monitors from ERE specifications fixed
  - Bugs for initialization of check point specifications fixed
  - Special thanks to Ylies Falcone for several bug reports and suggestions
- **2006-11-06:** JavaMOP version 1.1 was released with the following changes:
  - The JavaMOP specification syntax slightly changed
  - A new web-based interface provided
  - JavaMOP client applications improved. Note: old clients do ***not*** work any longer
- **2006-05-10:** JavaMOP version 1.0 was released

end