Rascal: Meta-Programming for Program Analysis

Mark Hills, Paul Klint, & Jurgen J. Vinju

9th International Workshop on Rewriting Logic and its Applications
March 25, 2012
Tallinn, Estonia

http://www.rascal-mpl.org

Overview

- Rascal: Introduction and Motivations
- Options for Program Analysis in Rascal
- Upgrade Analysis for PHP Programs

Overview

- Rascal: Introduction and Motivations
- Options for Program Analysis in Rascal
- Upgrade Analysis for PHP Programs

What is Rascal?

Rascal is a powerful domain-specific programming language that can scale up to handle challenging problems in the domains of:

- Software analysis
- Software transformation
- DSL Design and Implementation

Why Rascal?

~~Why Rascal?~~ Why not ASF+SDF?

"RASCAL is not an algebraic specification formalism with programming language features, but rather a programming language with algebraic specification features"

- *Rascal: From Algebraic Specification to Meta-Programming*, Jeroen van den Bos, Mark Hills, Paul Klint, Tijs van der Storm, and Jurgen J. Vinju, AMMSE 2011

Answer: The Intended Users of Rascal

VS

Lessons Learned: ASF, the Benefits

- "Match and Apply": equational logic and term rewriting, with conditional and default equations

- Powerful list matching features (especially in conjunction with SDF -- matching over lists of concrete terms)

- Reuse and extensibility: parameterized modules, renaming on import, can add new constructors and equations (but problematic under configuration changes)

Lessons Learned: SDF, the Benefits

- Syntax definitions are algebraic signatures
- Scannerless generalized parsing, handles complexity of real-life languages where whitespace, etc may matter
- Generalized parsing allows modularity -- unions of context free grammars are still context free
- With ASF, equations can perform complex transformations of source code

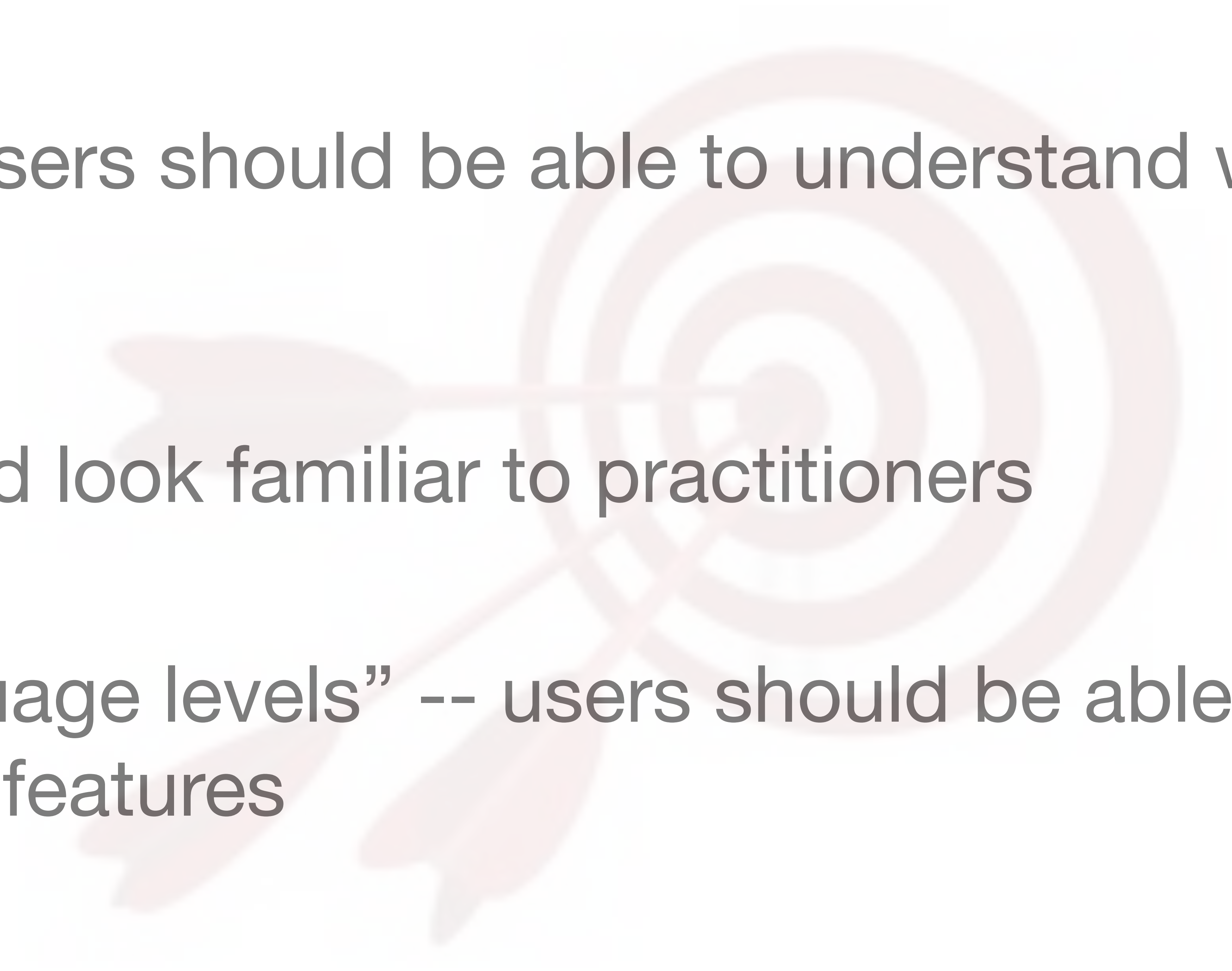
Lessons Learned: Some Challenges, Too

- Need a grammar for entities being reasoned about (e.g., dot files, XML configuration files, etc); not always trivial to create one
- Similarly, not everything is context free: requires pre-processing using other tools
- Ability to combine grammars does not preclude ambiguity
- Challenging to debug: type errors manifest as parse errors, programming bugs as matching failures

Lessons Learned: Some Challenges, Too

- For standard functional-style programs, “apply-anywhere” rules can provide too much freedom, requires program to constrain application
- Information stored as graphs, sets, etc has to be encoded into a tree (set matching in Maude alleviates this somewhat, context transformers in K even more; Rascal includes set matching now too!)
- Rule-based programming not familiar to normal programmers/software engineers that may want to use our tools

Rascal Goals



- Cover entire domain of meta-programming
- “No Magic” -- users should be able to understand what is going on from looking at the code
- Programs should look familiar to practitioners
- Unofficial “language levels” -- users should be able to start simple, build up to more advanced features

Rascal fixes these...

- Need a grammar for entities being reasoned about, plus not everything is context free: ***URI-based I/O operations, regexp matching, typed resources***
- Ambiguous grammars: ***ambiguity-detection and diagnostic tools help ameliorate (still undecidable)***
- Debugging challenges: ***static type system with local inference, developing tools to help detect cases where not all patterns are given, adding a code debugger, etc***

...and these, too!

- Need to constraint program: ***programs now structured as functions with familiar control flow constructs; visits allow structure-shy traversal***

- Information must be encoded as trees: ***Rascal now includes lists, sets, maps, tuples, and relations, with comprehensions and matching***

- Unfamiliar programming style: ***see above; mainly-functional programs, with elements from rewriting, but with a Java-like syntax***

Rascal Features

- Scannerless GLL parsing
- Flexible pattern matching, lexical backtracking, and matching on concrete syntax
- Functions with parameter-based dispatch, default functions, and higher-order functions
- Traversal and fixpoint computation operations
- Immutable data, rich built-in data types, user-defined types

Example: 101Companies

```
start syntax S_Companies = S_Company+ companies;

syntax S_Company
    = @Foldable "company" S_StringLiteral name "{" S_Department* departments "}";

syntax S_Department
    = @Foldable "department" S_StringLiteral name "{" S_DepartmentElement* elements "}";

keyword S_Keywords
    = "company"
    | "department"
    | "manager"
    | "employee"
    ;

lexical Layout
    = [\t-\n\r\ ]
    | Comment
    ;

layout Layouts
    = Layout* !>> [\t-\n \r \ ]
    ;
```

Example: 101Companies

```
data Companies
    = companies(list[Company] comps);

data Company
    = company(str name, list[Department] deps);

data Department
    = department(str name, list[Department] deps, list[Employee] empls);

data Employee
    = employee(str name, list[EmployeeProperty] props);

data Employee
    = manager(Employee emp);

data EmployeeProperty
    = intProp(str name, int intVal)
    | strProp(str name, str strVal);
```

Example: 101Companies

```
Department toAST(S_Department d) {
    if (`department <S_StringLiteral name> { <S_DepartmentElement* elements> }` := d) {
        list[Department] dl = [ ];
        list[Employee] el = [ ];
        for (e <- elements) {
            switch(e) {
                case (S_DepartmentElement) `<S_Department ded>` : dl = dl + toAST(ded);
                case (S_DepartmentElement) `<S_Manager dem>` : el = el + toAST(dem);
                case (S_DepartmentElement) `<S_Employee dee>` : el = el + toAST(dee);
                default : throw "Unrecognized S_DepartmentElement syntax: <e>";
            }
        }
        return department(toASTString("<name>"), dl, el)[@at=d@\loc][@nameAt=name@\loc];
    }
    throw "Unrecognized S_Department syntax: <d>";
}
```

Example: 101Companies

```
@doc{Total the salaries of all employees}
public int total(Company c) {
    return (0 | it + salary | /employee(name, [*ep,ip:intProp("salary",salary),*ep2]) <- c);
}

@doc{Print the current salary assignments, useful for debugging}
public void printCurrent(Company c) {
    visit (c) {
        case employee(name, [*ep,ip:intProp("salary",salary),*ep2]) :
            println("<name>: $<salary>");
    }
}
```

Example: Rascal Type System

```
public Symbol \var-func(Symbol ret, list[Symbol] parameters, Symbol varArg) =
              \func(ret, parameters + \list(varArg));

public bool subtype(Symbol s, s) = true;
public default bool subtype(Symbol s, Symbol t) = false;
public bool subtype(\int(), \num()) = true;
public bool subtype(\rat(), \num()) = true;
public bool subtype(\real(), \num()) = true;
public bool subtype(\tuple(list[Symbol] l), \tuple(list[Symbol] r)) = subtype(l, r);
public bool subtype(\rel(list[Symbol] l), \rel(list[Symbol] r)) = subtype(l, r);
public bool subtype(\list(Symbol s), \list(Symbol t)) = subtype(s, t);
```

Example: Rascal V2I Transformation

```
return { f | <f,e> <- r@extends,
             entity([ifPrefix,class(cn,_)]) := e,
             (/^<cnp:[^\<]+>.*$/ := cn || /^<cnp:[^\<]+>$/ := cn), cName == cnp }
     + { f | <f,e> <- r@extends,
             entity([ifPrefix,class(cn)]) := e,
             (/^<cnp:[^\<]+>.*$/ := cn || /^<cnp:[^\<]+>$/ := cn), cName == cnp };

alias MethodInfoWDef = rel[str mname, loc mloc, Entity owner,
                           Entity method, Entity def];

MethodInfoWDef miImp = { <mi.mname,mi.mloc,mi.owner,mi.method,def> |
   e <- implementers,
   tuple[str mname, loc mloc, Entity owner, Entity method] mi <-
     getVisitorsInClassOrInterface(rascal,e),
    entity([_*,method(mn,_,_)]) := mi.method, mn in miBaseNames,
   def <- (miBase[mn]<2>) };
```

Overview

- Rascal: Introduction and Motivations
- Options for Program Analysis in Rascal
- Upgrade Analysis for PHP Programs

What is Rascal?

Rascal is a powerful domain-specific programming language that can scale up to handle challenging problems in the domains of:

- **Software analysis**
- Software transformation
- DSL Design and Implementation

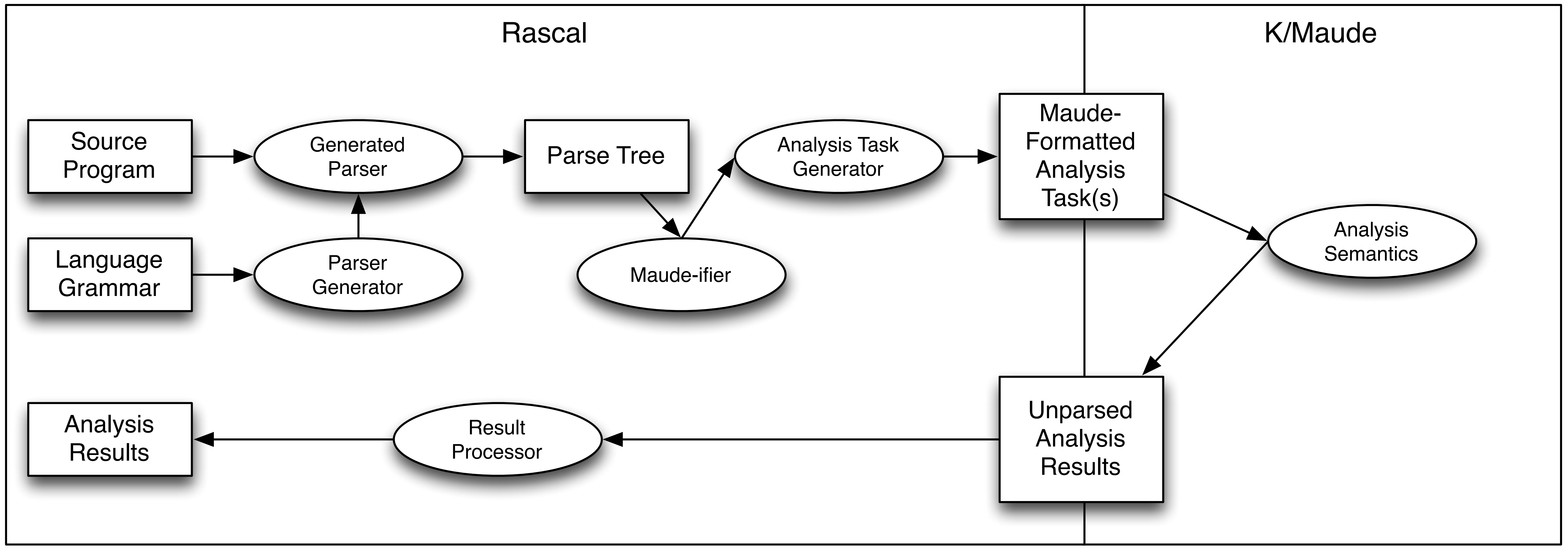
Options for Program Analysis in Rascal

- Reuse
- Collaboration
- From-scratch implementation (all in Rascal)

Reuse: Linking with Rewriting Logic Semantics and K

- Syntax, development environment for language defined in Rascal
- Semantics (execution, analysis, etc) defined in K or directly in Maude
- Rascal generates K or Maude terms decorated with location information
- Rascal displays results of execution: text, graphical annotations, etc

Linking Rascal with Rewriting Logic Semantics and K

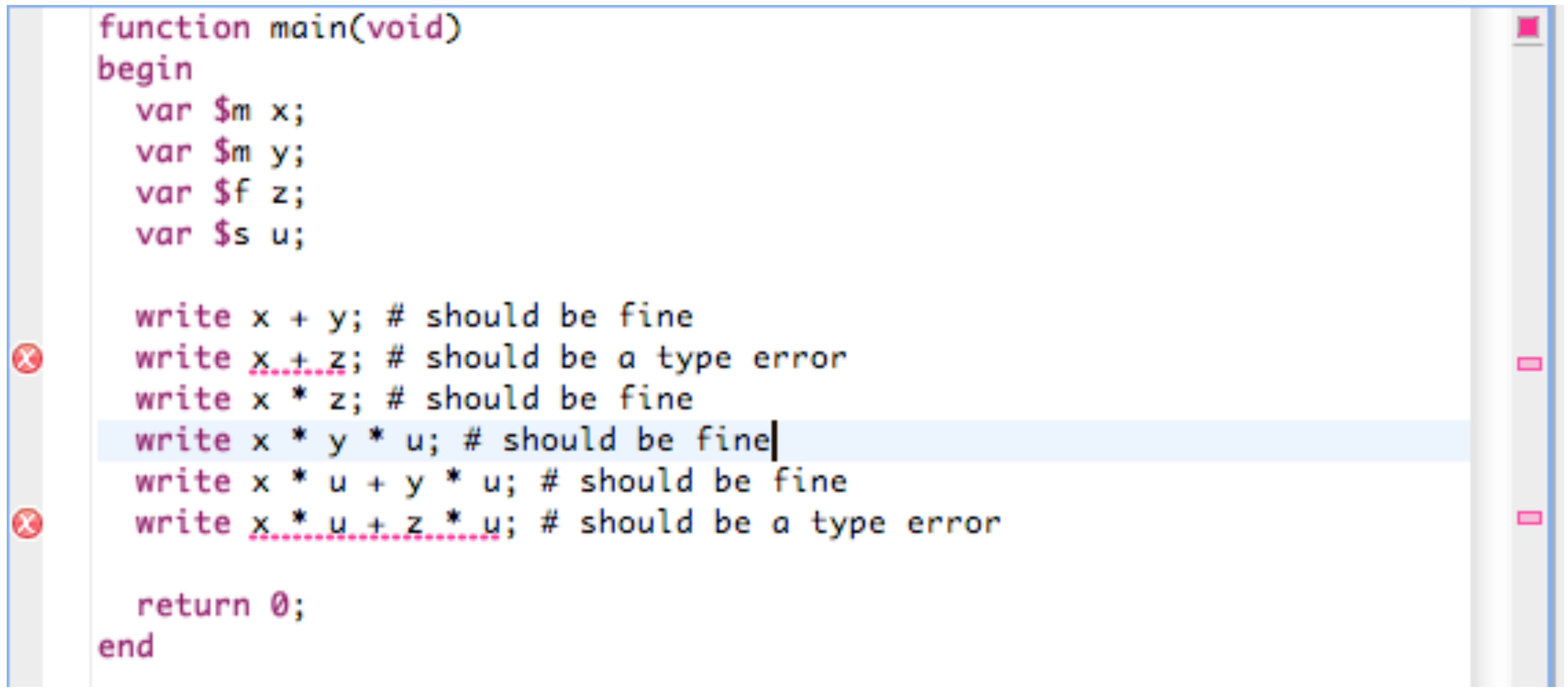


Representing Locations in Maude

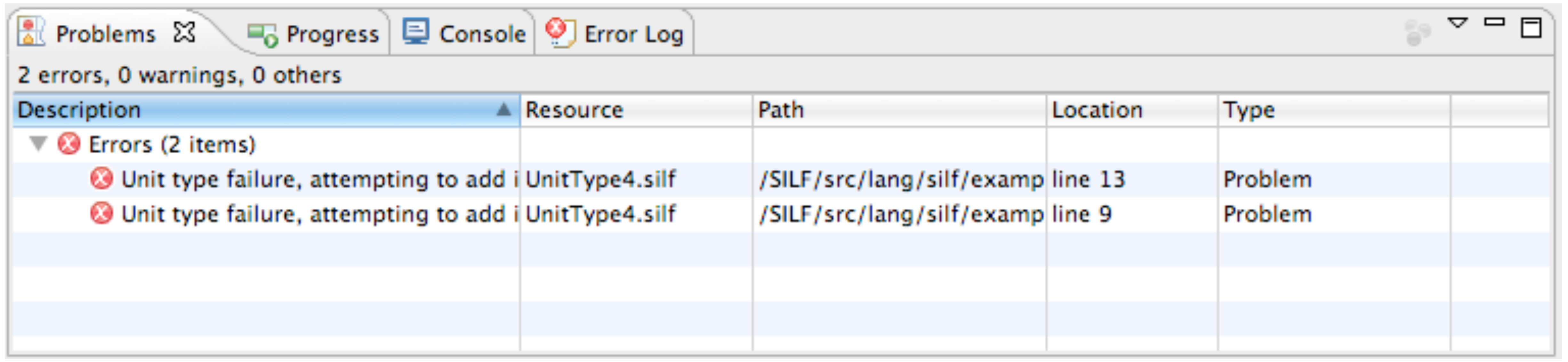
```
fmod RASCAL-LOCATION is
  including STRING .
  including INT .
  sort RLocation .
  op sl : String Int Int Int Int Int Int -> RLocation .
endfm
```

```
op currLoc : RLocation -> State [format (r! o)] .

op rloc : RLocation -> ComputationItem .

eq k(rloc(RL) -> K) currLoc(RL') = k(K) currLoc(RL) .

eq k(exp(locatedExp(E, RL)) -> K) currLoc(RL') =
     k(exp(E) -> rloc(RL') -> K) currLoc(RL) .
```

Displaying Detected Errors using Rascal



```
function main(void)
begin
  var $m x;
  var $m y;
  var $f z;
  var $s u;

  write x + y; # should be fine
  write x + z; # should be a type error
  write x * z; # should be fine
  write x * y * u; # should be fine
  write x * u + y * u; # should be fine
  write x * u + z * u; # should be a type error

  return 0;
end
```



2 errors, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (2 items)				
Unit type failure, attempting to add i	UnitType4.silf	/SILF/src/lang/silf/examp	line 13	Problem
Unit type failure, attempting to add i	UnitType4.silf	/SILF/src/lang/silf/examp	line 9	Problem

Collaboration: Using the Eclipse JDT

- JDT Library uses Eclipse to extract facts about Java files hos
  Eclipse project

- Examples: locations of method declarations, uses of class fields, types of variable names

- Facts presented as relations over Java entities

- An example use: find all implementations of methods defined in a specific interface, as well as all non-public fields and methods accessed in the method bodies

Overview

- Rascal: Introduction and Motivations
- Options for Program Analysis in Rascal
- Upgrade Analysis for PHP Programs

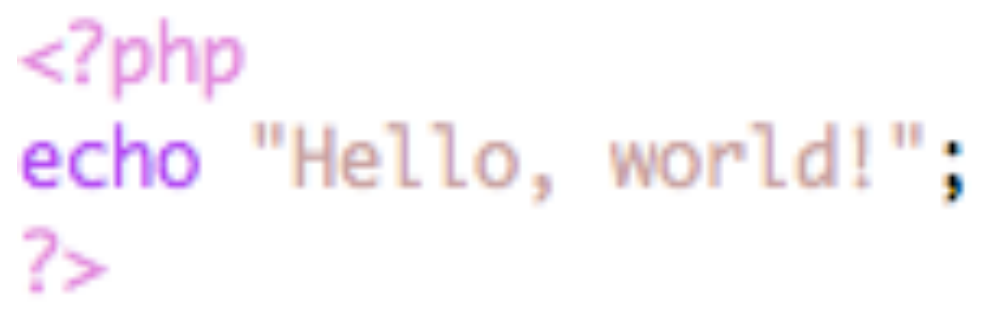
PHP: An Overview

- Created by Rasmus Lerdorf in 1994 so he could maintain his own homepage
- Originally written in Perl, now in C
- Dynamic programming language with static scopir
- Constantly extended with new features: Java-like class model (v5), goto statements (v5.3), and now traits (v5.4)

PHP Programs

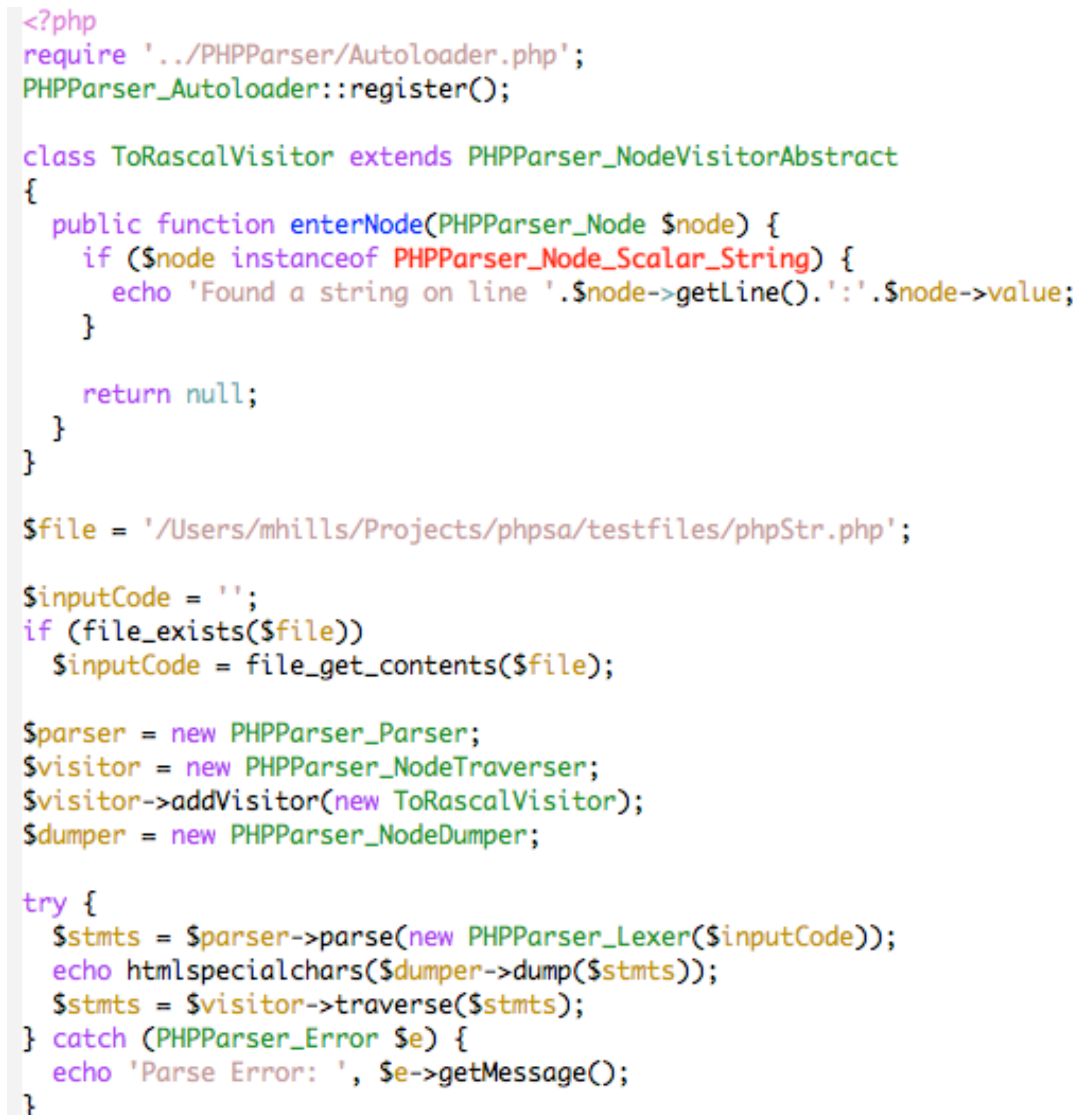
- Scripts are HTML with embedded fragments of PHP
- Can also be just PHP (special case)
- Executed on the server, client-side content just HTML, JavaScript, etc

The Mandatory Hello, World Example



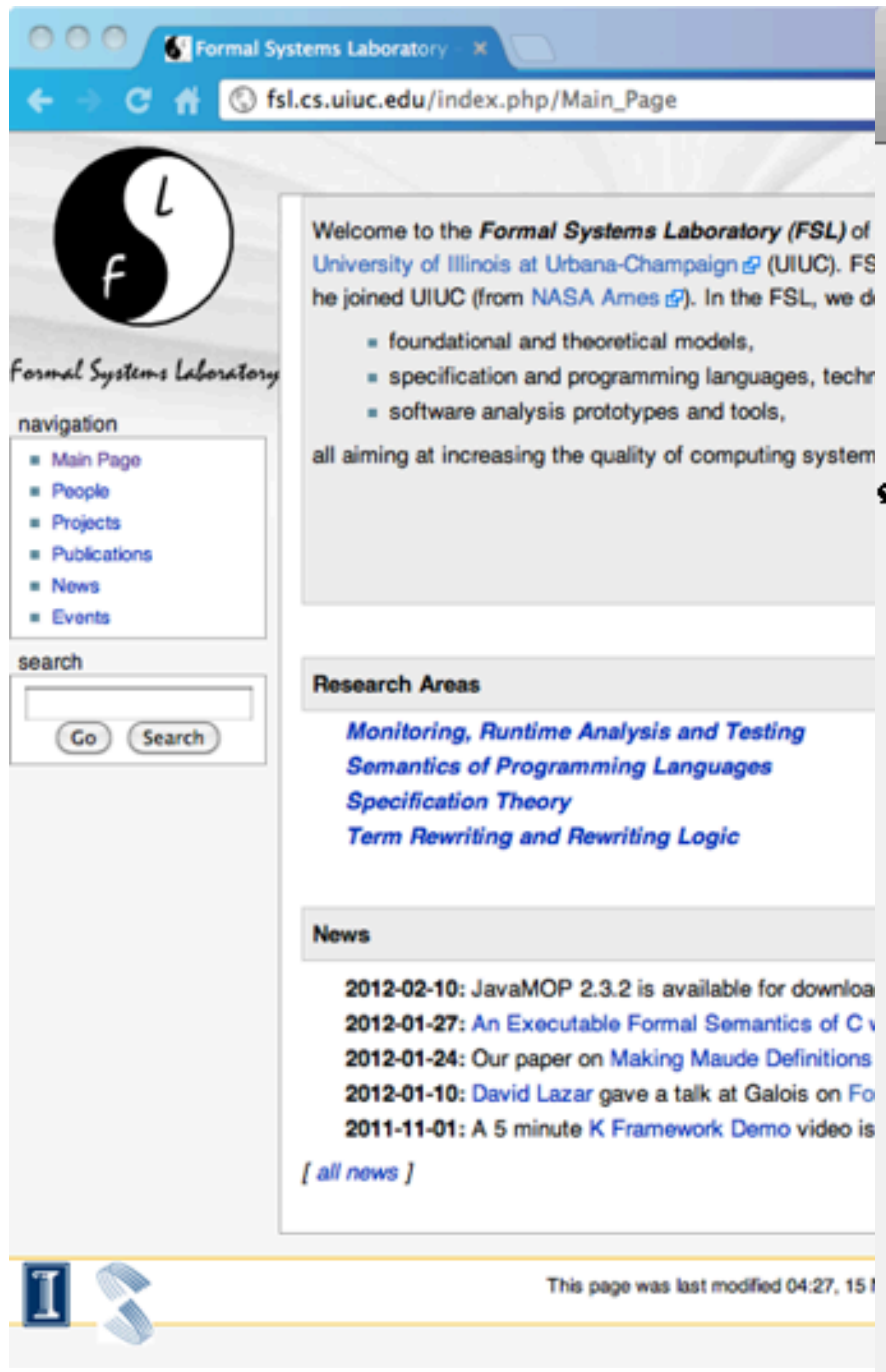
```php
<?php
echo "Hello, world!";
?>
```

Parsing PHP Programs in PHP



```php
<?php
require '../PHPParser/Autoloader.php';
PHPParser_Autoloader::register();

class ToRascalVisitor extends PHPParser_NodeVisitorAbstract
{
  public function enterNode(PHPParser_Node $node) {
    if ($node instanceof PHPParser_Node_Scalar_String) {
      echo 'Found a string on line '.$node->getLine().':'.$node->value;
    }

    return null;
  }
}

$file = '/Users/mhills/Projects/phpsa/testfiles/phpStr.php';

$inputCode = '';
if (file_exists($file))
  $inputCode = file_get_contents($file);

$parser = new PHPParser_Parser;
$visitor = new PHPParser_NodeTraverser;
$visitor->addVisitor(new ToRascalVisitor);
$dumper = new PHPParser_NodeDumper;

try {
  $stmts = $parser->parse(new PHPParser_Lexer($inputCode));
  echo htmlspecialchars($dumper->dump($stmts));
  $stmts = $visitor->traverse($stmts);
} catch (PHPParser_Error $e) {
  echo 'Parse Error: ', $e->getMessage();
}
```

Web Example: The FSL Wiki (Mediawiki)



Formal Systems Laboratory

fsl.cs.uiuc.edu/index.php/Main_Page

Formal Systems Laboratory

navigation

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

search

Go   Search

Welcome to the ***Formal Systems Laboratory (FSL)*** of
University of Illinois at Urbana-Champaign (UIUC). FS
he joined UIUC (from NASA Ames). In the FSL, we d

- foundational and theoretical models,
- specification and programming languages, techr
- software analysis prototypes and tools,

all aiming at increasing the quality of computing system

Research Areas

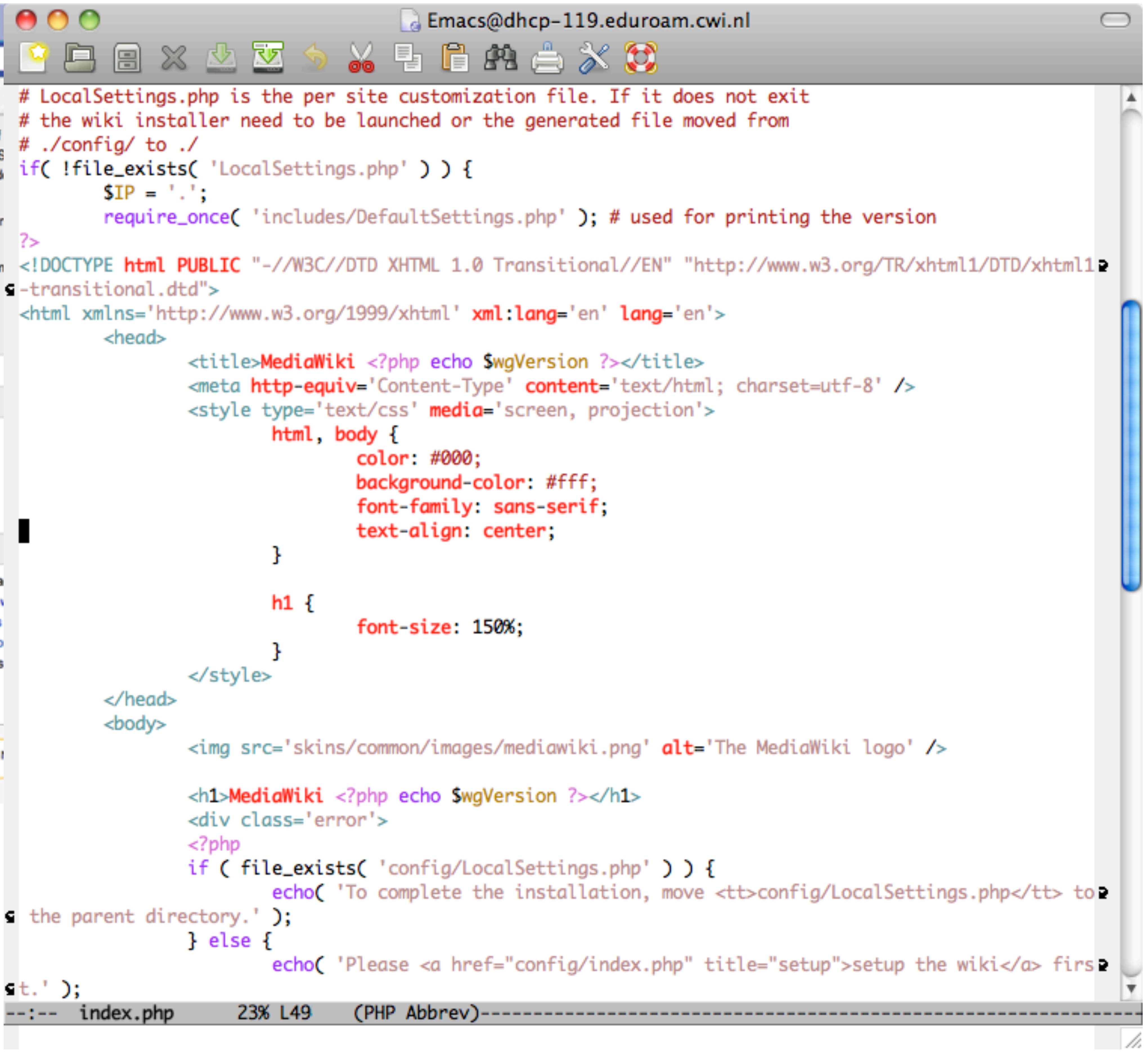
Monitoring, Runtime Analysis and Testing
Semantics of Programming Languages
Specification Theory
Term Rewriting and Rewriting Logic

News

2012-02-10: JavaMOP 2.3.2 is available for downloa
2012-01-27: An Executable Formal Semantics of C
2012-01-24: Our paper on Making Maude Definitions
2012-01-10: David Lazar gave a talk at Galois on Fo
2011-11-01: A 5 minute K Framework Demo video is

[all news]

This page was last modified 04:27, 15



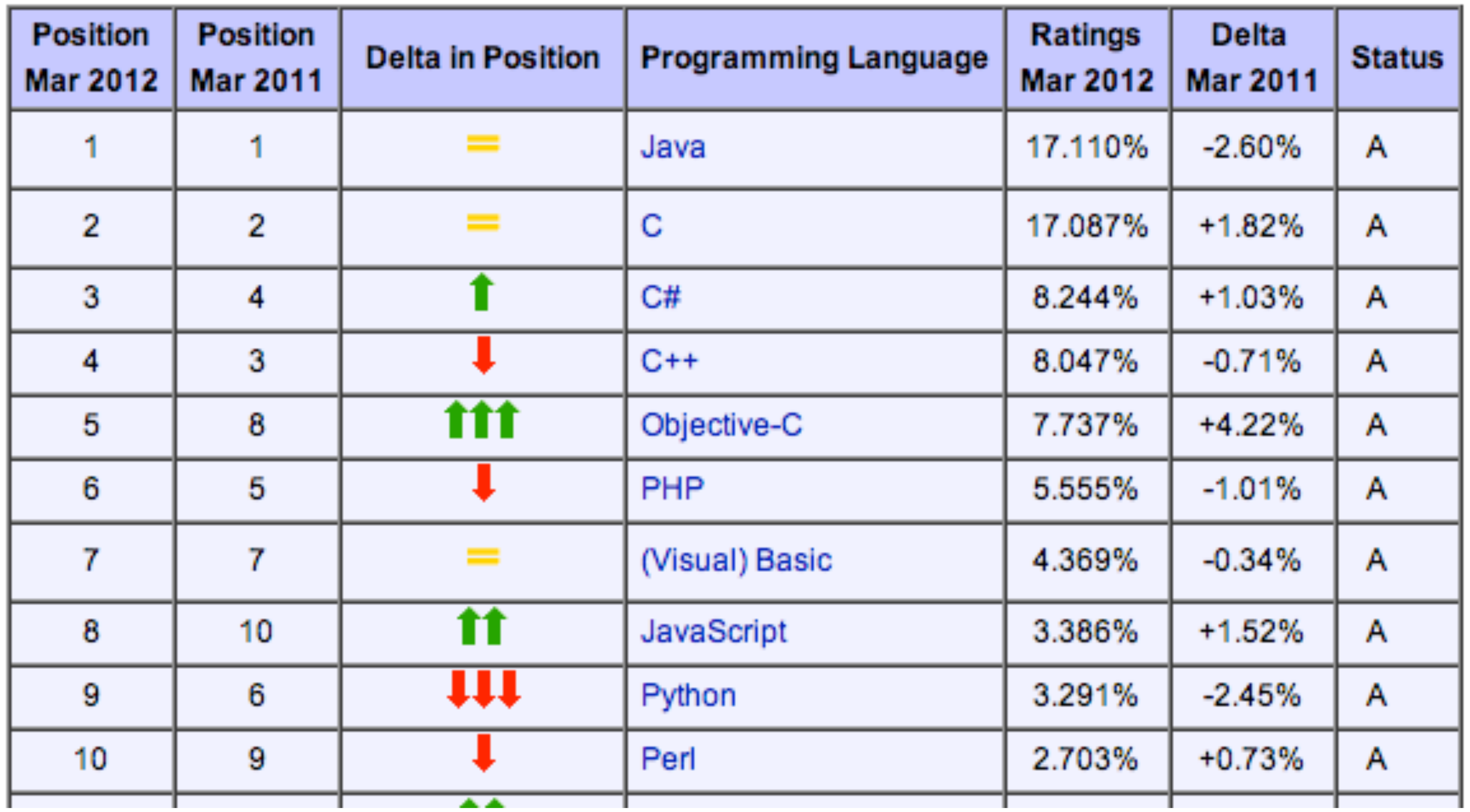
Emacs@dhcp-119.eduroam.cwi.nl

```
# LocalSettings.php is the per site customization file. If it does not exit
# the wiki installer need to be launched or the generated file moved from
# ./config/ to ./
if( !file_exists( 'LocalSettings.php' ) ) {
        $IP = '.';
        require_once( 'includes/DefaultSettings.php' ); # used for printing the version
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1
-transitional.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'>
        <head>
                <title>MediaWiki <?php echo $wgVersion ?></title>
                <meta http-equiv='Content-Type' content='text/html; charset=utf-8' />
                <style type='text/css' media='screen, projection'>
                        html, body {
                                color: #000;
                                background-color: #fff;
                                font-family: sans-serif;
                                text-align: center;
                        }

                        h1 {
                                font-size: 150%;
                        }
                </style>
        </head>
        <body>
                <img src='skins/common/images/mediawiki.png' alt='The MediaWiki logo' />

                <h1>MediaWiki <?php echo $wgVersion ?></h1>
                <div class='error'>
                <?php
                if ( file_exists( 'config/LocalSettings.php' ) ) {
                        echo( 'To complete the installation, move <tt>config/LocalSettings.php</tt> to
 the parent directory.' );
                } else {
                        echo( 'Please <a href="config/index.php" title="setup">setup the wiki</a> firs
t.' );
--:--  index.php      23% L49     (PHP Abbrev)--------------------------------------------------------------
```

Why Analyze PHP?

- Widespread usage: PHP is ranked 6th in current Tiobe rankings (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html)

Tiobe Rankings, March 2012



Position Mar 2012	Position Mar 2011	Delta in Position	Programming Language	Ratings Mar 2012	Delta Mar 2011	Status
1	1	=	Java	17.110%	-2.60%	A
2	2	=	C	17.087%	+1.82%	A
3	4	↑	C#	8.244%	+1.03%	A
4	3	↓	C++	8.047%	-0.71%	A
5	8	↑↑↑	Objective-C	7.737%	+4.22%	A
6	5	↓	PHP	5.555%	-1.01%	A
7	7	=	(Visual) Basic	4.369%	-0.34%	A
8	10	↑↑	JavaScript	3.386%	+1.52%	A
9	6	↓↓↓	Python	3.291%	-2.45%	A
10	9	↓	Perl	2.703%	+0.73%	A

"The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. The popular search engines Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings.", from http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Why Analyze PHP?

- Widespread usage: PHP is ranked 6th in current Tiobe rankings (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html)
- Combination of dynamic types and odd features makes analysis important for program understanding, program correctness

Variable variables: the poor man’s pointer

```php
<?php
class foo {
    var $bar = 'I am bar.';
}

$foo = new foo();
$bar = 'bar';
$baz = array('foo', 'bar', 'baz', 'quux');
echo $foo->$bar . "\n";
echo $foo->$baz[1] . "\n";
?>
```

Variable variables: the poor man’s pointer

```php
<?php
$instance = new SimpleClass();

// This can also be done with a variable:
$className = 'Foo';
$instance = new $className(); // Foo()
?>
```

Coercions are sometimes unexpected...

```php
<?php
$foo = 1 + "10.5";                // $foo is float (11.5)
$foo = 1 + "-1.3e3";              // $foo is float (-1299)
$foo = 1 + "bob-1.3e3";           // $foo is integer (1)
$foo = 1 + "bob3";                // $foo is integer (1)
$foo = 1 + "10 Small Pigs";       // $foo is integer (11)
$foo = 4 + "10.2 Little Piggies"; // $foo is float (14.2)
$foo = "10.0 pigs " + 1;          // $foo is float (11)
$foo = "10.0 pigs " + 1.0;        // $foo is float (11)
?>
```

Figuring out what is included can be hard...

```
<?php

function foo()
{
    global $color;

    include 'vars.php';

    echo "A $color $fruit";
}

/* vars.php is in the scope of foo() so     *
* $fruit is NOT available outside of this  *
* scope.  $color is because we declared it *
* as global.                               */

foo();                        // A green apple
echo "A $color $fruit";   // A green

?>
```

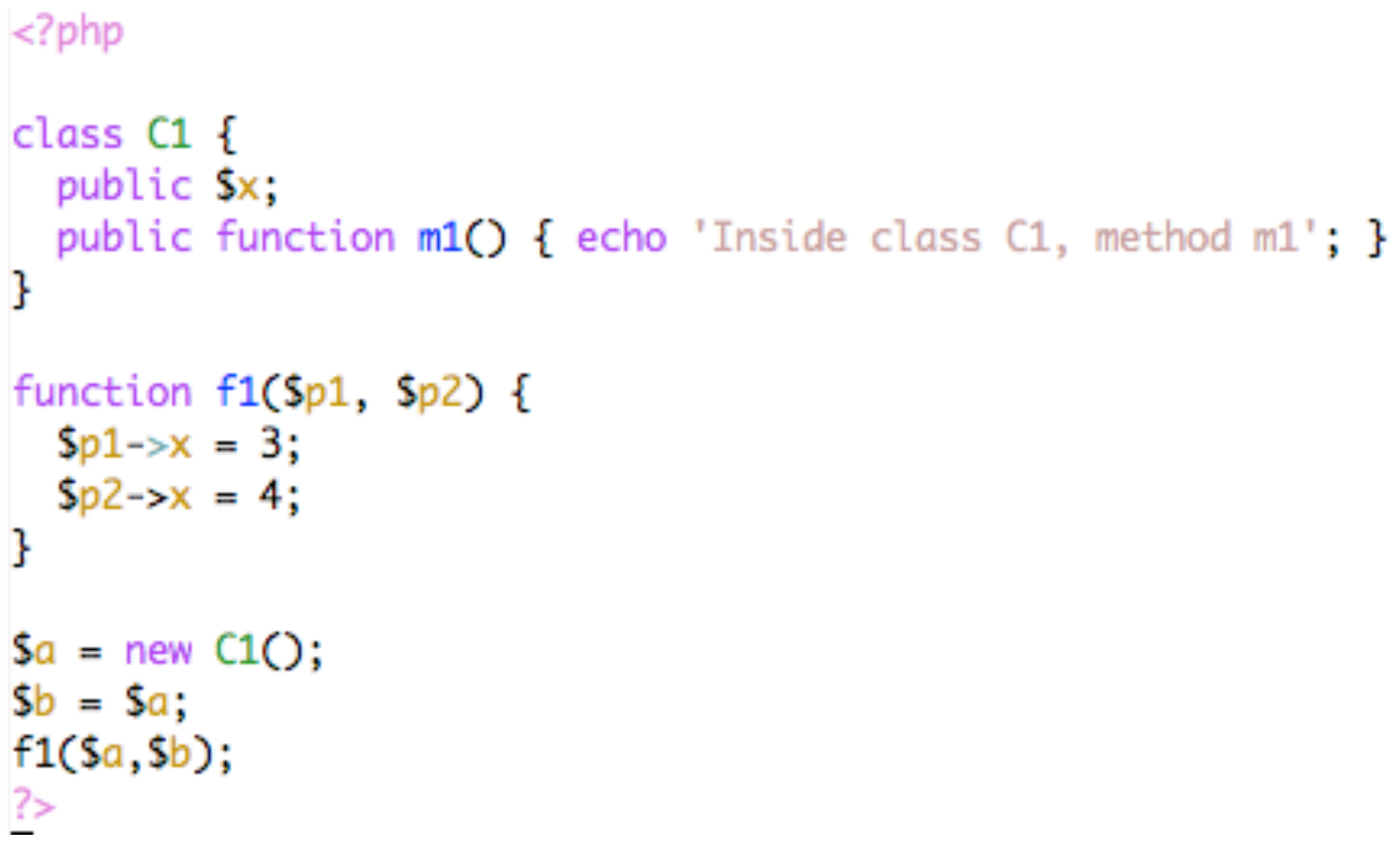
Upgrade Analysis for PHP Programs

- With introduction of new object model, default object representation changed: structures to references

- Potential to break existing code which relied on old behavior

- Analysis focused on finding potential problems statically, combination of type inference, alias analysis, intraprocedural dataflow analysis

Example Error Case



```php
<?php

class C1 {
  public $x;
  public function m1() { echo 'Inside class C1, method m1'; }
}

function f1($p1, $p2) {
  $p1->x = 3;
  $p2->x = 4;
}

$a = new C1();
$b = $a;
f1($a,$b);
?>
```

Analyzing PHP: A First Attempt

- Compile PHP scripts into intermediate tree representation using **phc**
- Perform analysis over tree: generate call graph, perform type inference, perform alias analysis
- Must iterate these analyses: type inference can detect new types, leading to new methods, leading to new aliases, etc
- Using generated information, find r/w or w/w pairs

Did this work? Sometimes...

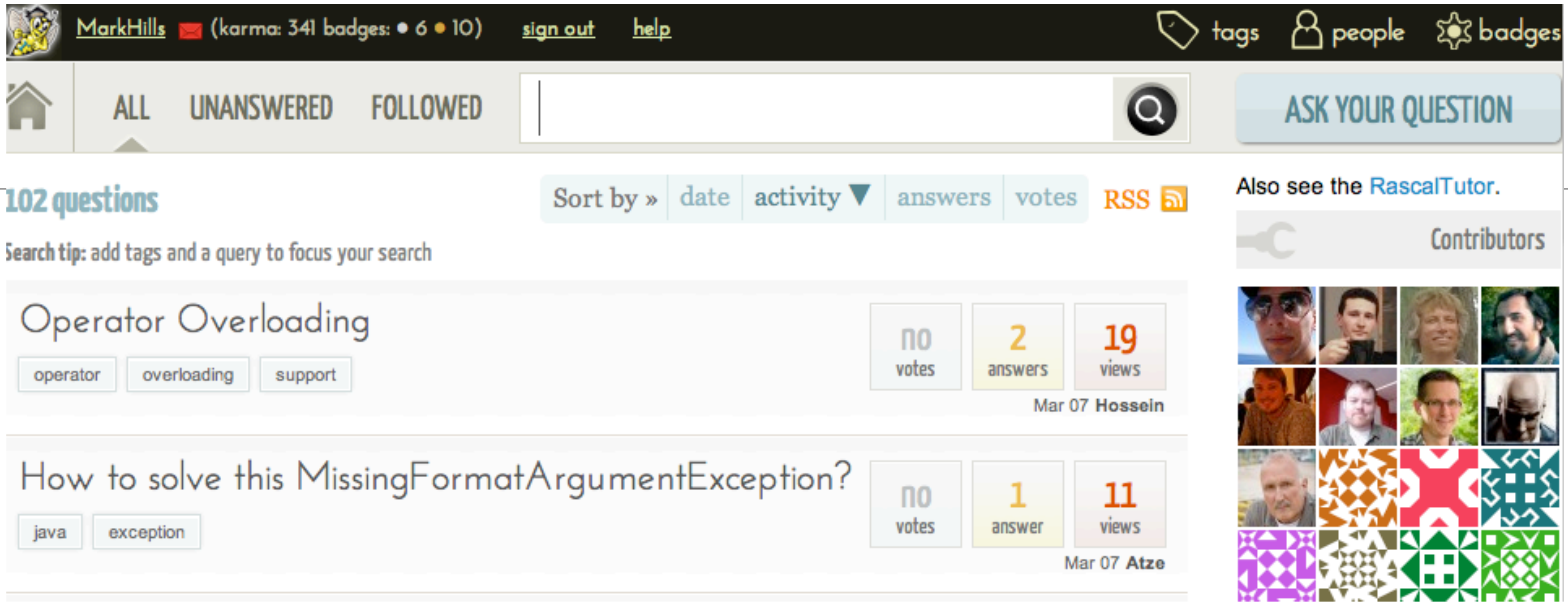
- Small examples, works great
- But large examples are too slow!
- Biggest problem: optimization of data structures, problems with both memory and CPU usage
- Fixed partially, implemented in Java, but then...
- Second biggest problem: no control over iteration, big examples take forever to stabilize

Analyzing PHP Rebooted

- Parse PHP with minimal transformations, preservation of location information
- Generate program representation using algebraic types
- Perform analysis as an abstract evaluation over the domain of interest

Current Status: Still Early Stage

- Signature (i.e., types and constructors) defined
- New parser working, generating Rascal terms
- Converting some old analysis code over: most of it is going away
- Rewriting analysis in style of Rascal type checker and CPF: abstract evaluation over an analysis domain



102 questions

Sort by » date activity ▼ answers votes RSS

Search tip: add tags and a query to focus your search

Also see the RascalTutor.

Operator Overloading
operator overloading support
no votes 2 answers 19 views
Mar 07 **Hossein**

How to solve this MissingFormatArgumentException?
java exception
no votes 1 answer 11 views
Mar 07 **Atze**

- Rascal: http://www.rascal-mpl.org
- SEN1: http://www.cwi.nl/sen1
- Me: http://www.cwi.nl/~hills