Program Analysis and Verification with Rewriting Logic and Rascal

Mark Hills (Appalachian State University, NC, USA)

Invited Talk at Kristu Jayanti College (Autonomous)
February 22, 2024

NOTE: Based on material from WRLA 2012, ISSTA 2013, SANER 2017, SCAM 2017, SERPL 2019, and SCAM 2023

The big picture

- Program Analysis and Verification
- Rewriting Logic and Rascal
- PHP AiR for PHP Program Analysis
- Go AiR for Go Program Analysis and Verification
- Other Work on Program Analysis and Rascal
- Q&A

Program Analysis and Verification

What is static analysis?

In computer science, static program analysis (or static analysis) is the analysis of computer programs performed without executing them, in contrast with dynamic program analysis, which is performed on programs during their execution. … In most cases the analysis is performed on some version of a program's source code, and, in other cases, on some form of its object code.

https://en.wikipedia.org/wiki/Static_program_analysis

What is static analysis?

In computer science, static program analysis (or static analysis) is the analysis of computer programs **performed without executing them**, in contrast with dynamic program analysis, which is performed on programs during their execution. … In most cases the **analysis is performed on some version of a program's source code**, and, in other cases, on some form of **its object code**.

https://en.wikipedia.org/wiki/Static_program_analysis

Why static analysis?

- You can analyze all possible program behaviors, not just those you encounter while running the program

- Running a program may be expensive

- Triggering some conditions to test the resulting behavior could be problematic or even dangerous (e.g., code that run when a collision is detected)

- Many scenarios (e.g., refactoring, IDE support, code understanding) are inherently static

What is program verification?

[F]ormal verification is the act of proving or disproving the correctness of a system with respect to a certain formal specification or property, using formal methods of mathematics

https://en.wikipedia.org/wiki/Formal_verification

Why program verification?

- Needed for high-assurance systems (e.g., systems used in safety-critical hardware devices like medical devices)

- Techniques can show that programs meet desired requirements, such as reachability or avoidance of certain program states (e.g., deadlocks or starvation in concurrent programs)

- Static analysis is often used for program verification

- Related: same techniques can be used to better understand program behaviors, such as possible concurrent behaviors

Rewriting Logic and Rascal

What is rewriting logic?

- Rewriting logic is an extension of equational logic with support for concurrency

- Language semantics provides formal definitions of language features

- Rewriting logic semantics joins these two: formal language definitions using rewriting logic

- Definitions are executable with rewriting logic engines, like Maude, and can be reasoned about with existing tools

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

https://www.rascal-mpl.org/

What are the design goals for Rascal?

- 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 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

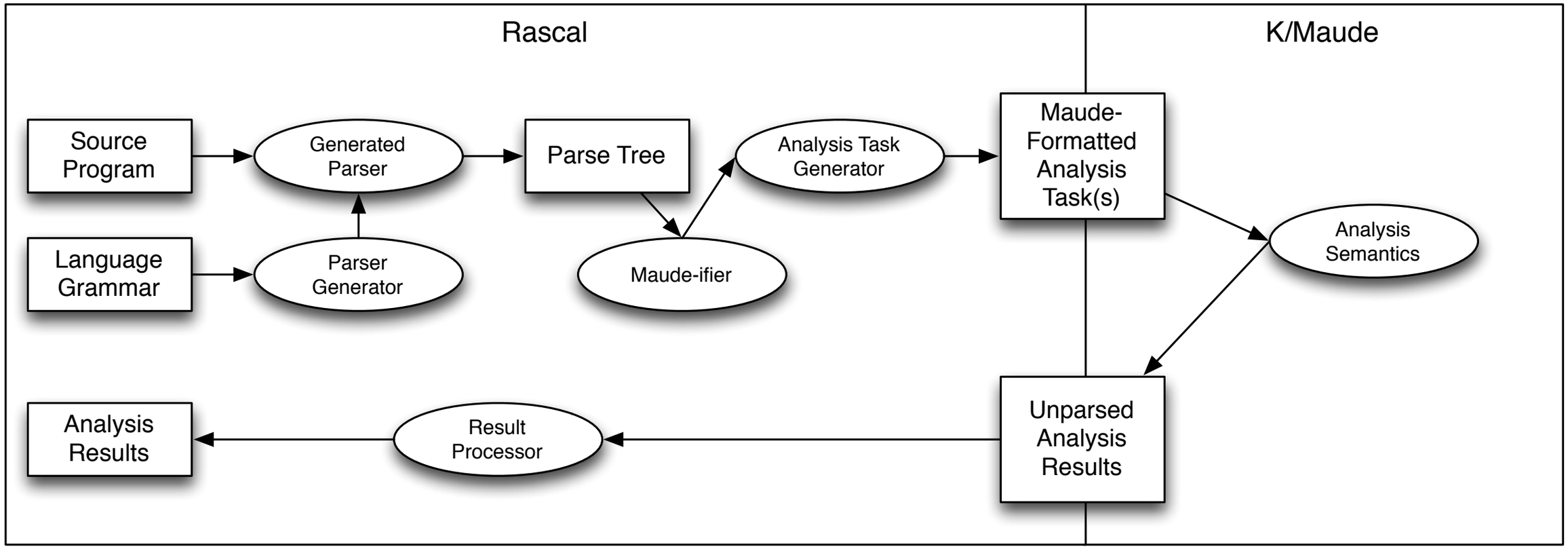
Options for Program Analysis in Rascal

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

Reuse: Linking with Rewriting Logic Semantics

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

Linking Rascal with Rewriting Logic Semantics and K



Collaboration: Using the Eclipse JDT

- JDT Library uses Eclipse to extract facts about Java files hosted inside an 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

- Note: Other tools could be used to interface with other languages

From-scratch: PHP AiR and Go AiR

- We do use external parsers for these languages, which makes it easier to stay in sync as languages evolve
- All analysis code currently written in Rascal
- Go AiR is moving towards collaboration, with a rewriting logic semantics of Go under development…

PHP AiR

PHP Analysis in Rascal (PHP AiR)

- PHP AiR: a framework for PHP source code analysis
- Domains:
  - Static program analysis
  - Empirical software engineering
  - Software metrics

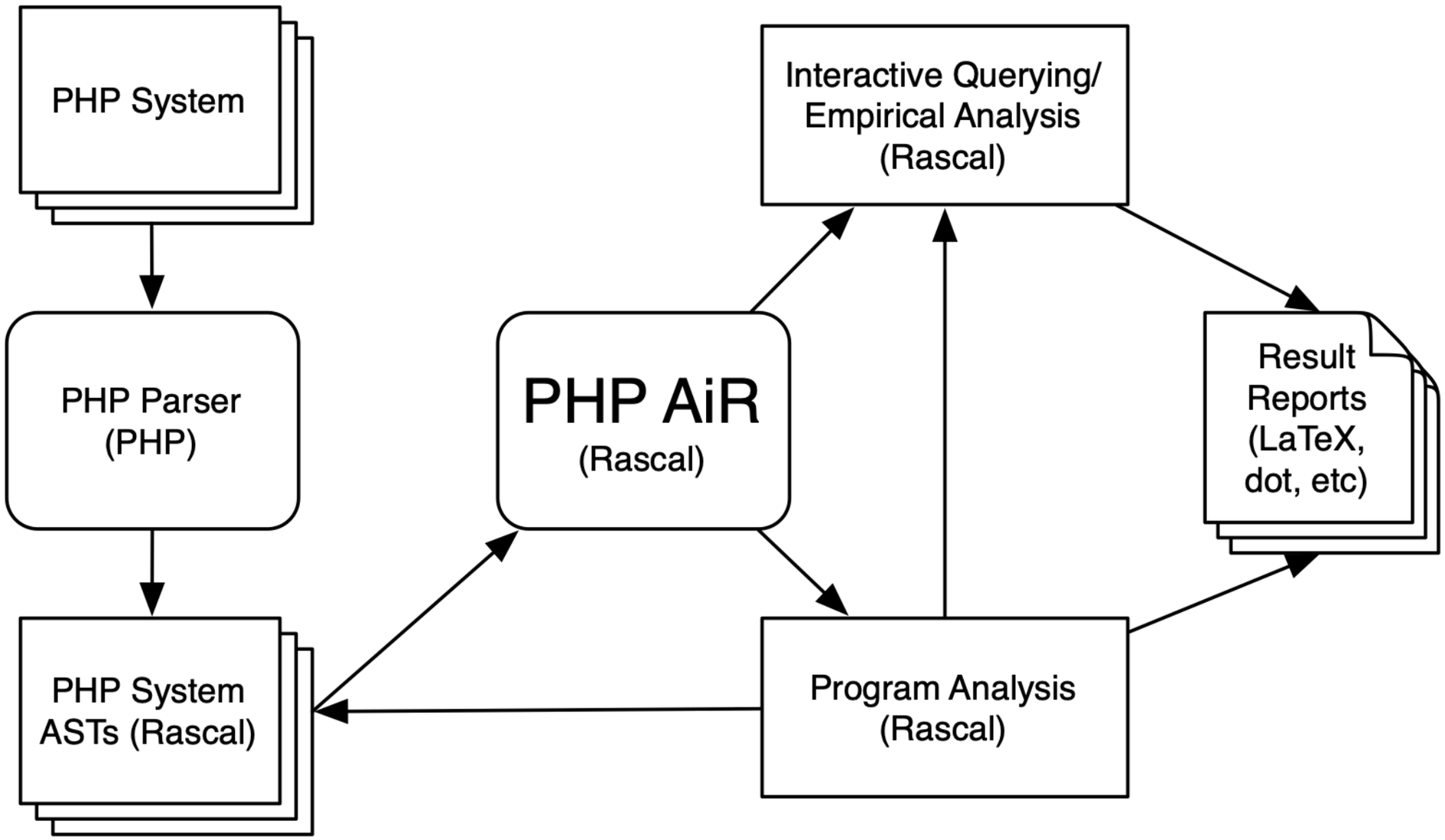
Why look at PHP applications?

- Popular with programmers: highly ranked on TIOBE Programming Community Index, on of the most popular languages on GitHub
- Used by over 75% of all websites whose server-side language can be determined, including in WordPress and sites like Wikipedia
- Big projects (MediaWiki 1.19.1 > 846k lines of PHP), wide range of programming skills, very limited tool support
- Hostile environments: most PHP code runs on the web, security is critical

PHP AiR design points

- Support for multiple PHP parsers
- Limited integration with Eclipse (can use the Eclipse PHP parser, can use LTK for transforming PHP files, but note that this needs to be brought up to date at this moment in time)
- Perform tasks by writing Rascal code (not focused on push-button solutions, goes with Rascal “no magic” principle)
- Must work with real PHP code (WordPress, MediaWiki, etc), not just toy samples
- Open source!

PHP AiR architecture



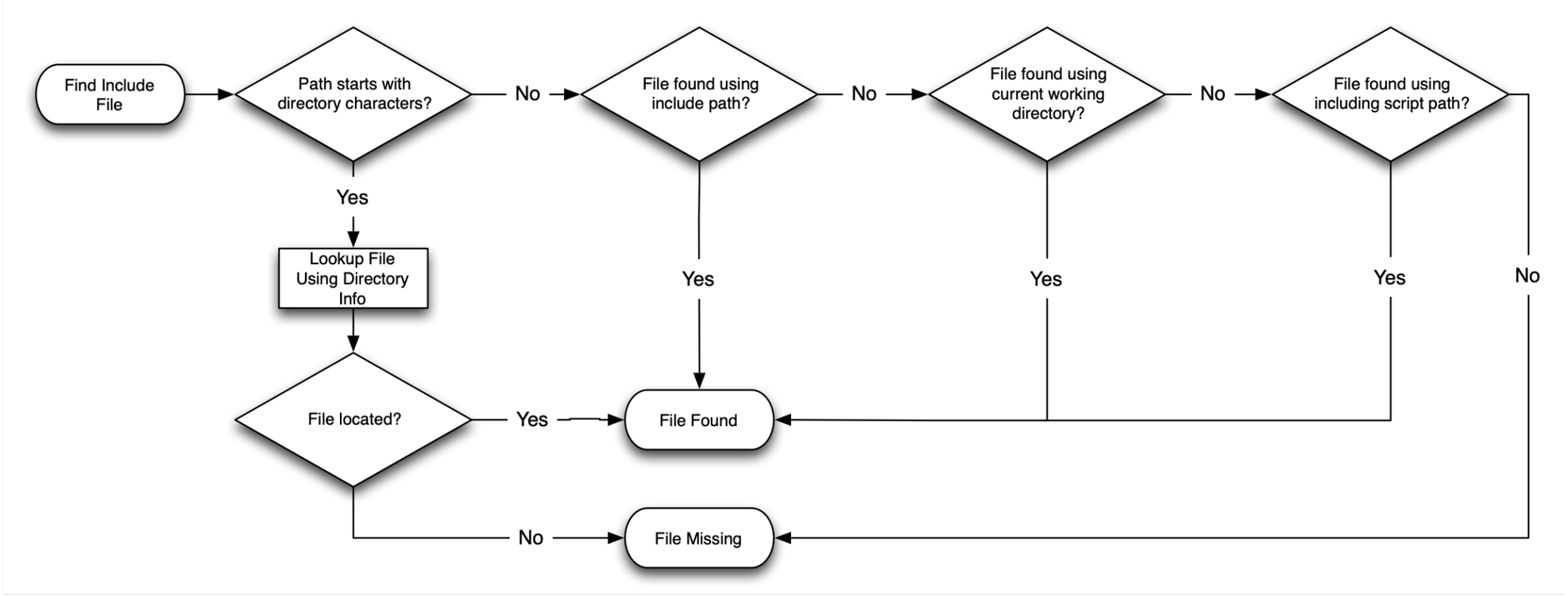
Example: PHP feature usage

- Perspective: Creators of program analysis tools
- What does a typical PHP program look like?
- What features of PHP do people really use?
- How often are dynamic features, which are hard for static analysis to handle, used in real programs?
- When dynamic features appear, are they really dynamic? Or are they used in static ways?

"An Empirical Study of PHP Feature Usage: A Static Analysis Perspective", Hills, Klint, and Vinju, ISSTA 2013.

Example: Resolving PHP includes

- Resolving PHP includes is messy (process is shown below)
- How many can we resolve at a per-file level? Per-program?



"Static, Lightweight Includes Resolution for PHP", Hills, Klint, and Vinju, ASE 2014.

Example: Resolving PHP Variable Features

- Variable features in PHP defer selection of an identifier until runtime
- Patterns in the code can help us to identify the identifiers that will be used at runtime, if we can detect them

```
// MediaWiki, /includes/Sanitizer.php, lines 424-428
$vars = array( 'htmlpairsStatic', 'htmlsingle',
  'htmlsingleonly', 'htmlnest',
  'tabletags', 'htmllist', 'listtags',
  'htmlsingleallowed', 'htmlelementsStatic' );
foreach ( $vars as $var ) {
  $$var = array_flip( $$var );
}
```


"Variable Feature Usage Patterns in PHP", Hills, ASE 2015.

Example: Understanding WordPress Plugins

- How do developers use WordPress plugin features?
- How can we help developers to find the right extension points?
- How can we help developers to find high-quality examples of handlers for these extension points?

```
// WordPress 4.2.4, wp-includes/meta.php, line 480
apply_filters( "get_{$meta_type}_metadata", null, $object_id, $meta_key, $single )

// Responsive Nagivation plugin, metabox/helpers/cmb_Meta_Box_Ajax.php, line 112
add_filter( 'get_post_metadata', array( 'cmb_Meta_Box_ajax', 'hijack_oembed_cache_get' ), 10, 3 )
```


"Navigating the WordPress Plugin Landscape", Hills, ICPC 2016.

Example: Query Construction Patterns

- How are queries typically built in PHP scripts?
- What parts of a query tend to be dynamic?
- What features are used to build these dynamic query parts?

“Query Construction Patterns in PHP”, Anderson and Hills, SANER 2017 (ERA Track).

Context: PHP and MySQL

- MySQL API uses query functions (originally mysql_query, that is the one we focus on here) to execute queries

- Queries given as strings, formed using string building operations

- Queries often have a mixture of static and dynamic pieces

```
$query = mysql_query("
SELECT title
FROM semesters
WHERE semesterid = $_POST[semester]
");
```

Note: Real but horrible query, this has a major security vulnerability…

Results of SANER'17 paper

- Queries appear to be built in predictable patterns

- Dynamic parts are mainly in the "right" places, making a transformation to prepared statements possible

- We need a more extensive analysis with more systems and more precise and sound analysis algorithms

- We need better models of the queries themselves (current work) for more precise pattern identification

- We need to build the transformation!

Query Models

- R1: How can we model how queries are built?

- R2: Using these models, how can we generate the queries or query templates (with placeholders for dynamic bits) that could actually be executed?

- R3: What parts of the query language (here, SQL) are used in these queries? Which of these parts are static, and which are dynamic?

- R4 (suggested by reviewers): Can we use models to find potential security vulnerabilities in PHP code? We haven't looked at this yet, but it sounds promising, especially with interprocedural support…

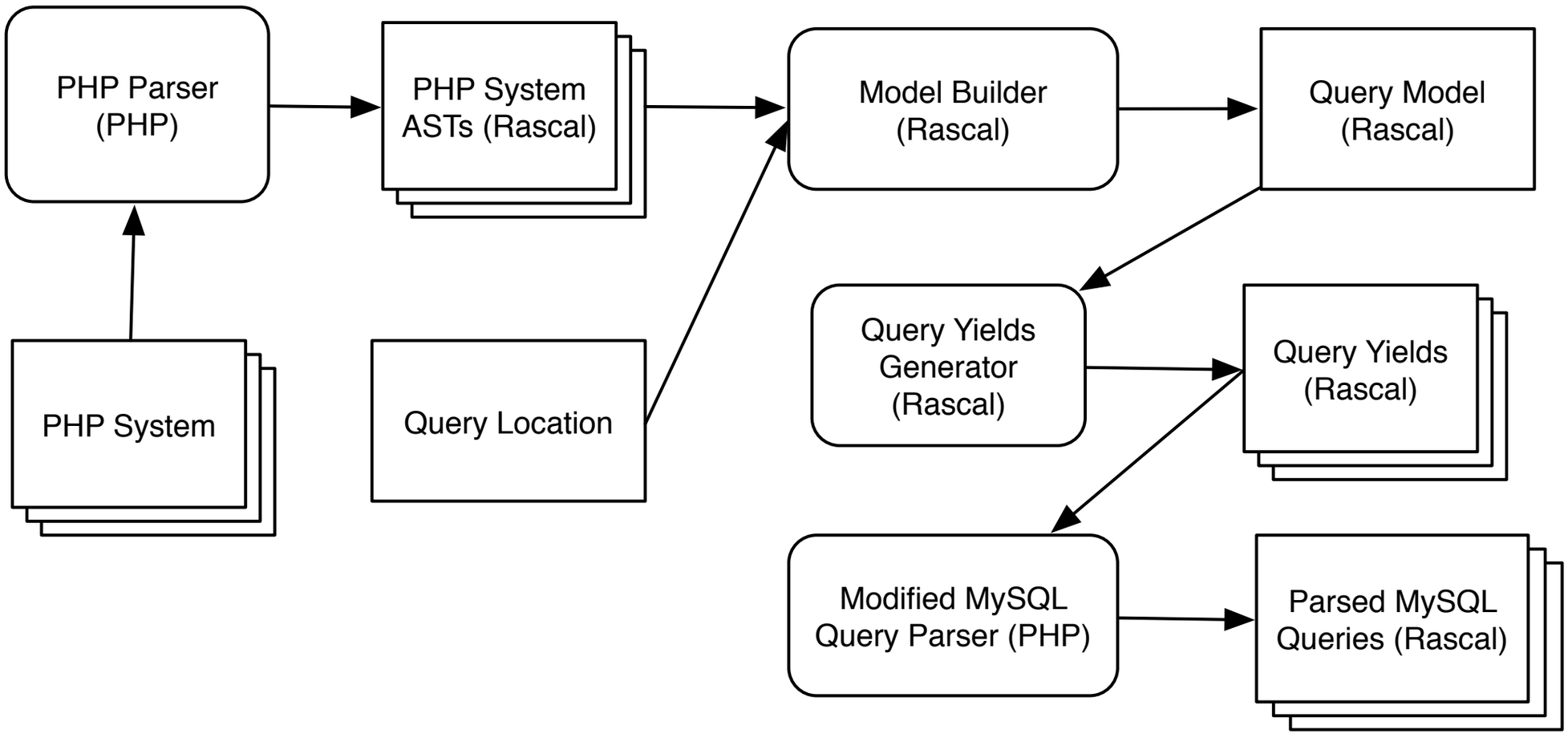
"Supporting Analysis of SQL Queries in PHP AiR", Anderson and Hills, SCAM 2017 (Engineering Track).

Analyzing and parsing queries: methodology

- All analysis code is written using Rascal
  - https://github.com/PLSE-Lab/mysql-query-construction-analysis
- PHP is parsed using a PHP parser written in PHP, parser yields Rascal terms
- MySQL queries are parsed using a fork of the parser found in phpMyAdmin, a web frontend for administering MySQL

"Supporting Analysis of SQL Queries in PHP AiR", Anderson and Hills, SCAM 2017 (Engineering Track).

Analyzing and parsing queries: “The Big Picture”



“Supporting Analysis of SQL Queries in PHP AiR”, Anderson and Hills, SCAM 2017 (Engineering Track).

R1: Building models

Input : *sys*, a PHP system, mapping from file locations to abstract syntax trees
Input : *callLoc*, a location indicating the query call to be analyzed
Output : *res*, a query model

```
1  inputCFG ← buildCFG4Loc (callLoc)
2  inputNode ← the CFG node from inputCFG representing the call at callLoc
3  d ← definitions (inputCFG)
4  u ← uses (inputCFG, d)
5  slicedCFG ← basicSlice (inputCFG, inputNode, usedNames (u, inputNode), d, u)
6  startingFragment ← expr2qf (inputNode)
7  fragmentRel ← {}
8  while fragmentRel continues to change do
9  |   nodesToExpand ← (inputNode.l × startingFragment) ∪ fragmentRel⟨3, 4⟩
10 |   foreach (nodeLabel × nodeFragment) ∈ nodesToExpand do
11 |   |   add expandFragment (nodeLabel, nodeFragment, d, u) to fragmentRel
12 |   end
13 end
14 fragmentRel ← addEdgeInfo (fragmentRel, slicedCFG)
15 res ← the model, including fragmentRel, startingFragment, and callLoc
```

Algorithm 1: Extracting a Model of a SQL Query.

See the SCAM 2017 paper for all the details, here comes a summary…

“Supporting Analysis of SQL Queries in PHP AiR”, Anderson and Hills, SCAM 2017 (Engineering Track).

R1: Building models

- Models are (possibly cyclic) graphs of *query fragments* (with a bit of bookkeeping info, like the location of the call)

- A query fragment is a static or dynamic piece of the query

- Intraprocedural, backwards slice throws away code that does not impact query

- CFG and def/use info link names in fragments to defs of those names

- Reachability conditions used to decorate graph edges

"Supporting Analysis of SQL Queries in PHP AiR", Anderson and Hills, SCAM 2017 (Engineering Track).

R2: Extract yields

- Yields are lists of “pieces”:
  - Static pieces for query text
  - Dynamic pieces for arbitrary expressions
    - Name pieces for names (useful to track separately)
- Generated by traversing the graph, currently cuts off when cycles detected
- Can use edge labels to filter infeasible yields, improving to work in more situations (loops are problematic)

“Supporting Analysis of SQL Queries in PHP AiR”, Anderson and Hills, SCAM 2017 (Engineering Track).

R3: Parsing partial queries

- First, yields are converted to strings: static pieces yield strings directly, dynamic and name pieces are turned into query holes (e.g., ?1, ?2, generally ?n)

- Second, string is parsed by our modified MySQL parser, yielding PHP objects representing MySQL AST (limitation: we assume holes are expressions and do not cross clause boundaries, supporting this is ongoing work, a dedicated parser would be quite helpful here since we depend on the MySQL parser but don't control this at all!)

- Third, AST pretty-printed to a Rascal term representing AST, similarly to current PHP parser

"Supporting Analysis of SQL Queries in PHP AiR", Anderson and Hills, SCAM 2017 (Engineering Track).

Go AiR

Why look at Go?

- Go is a widely used language with an interesting channel-based concurrency model plus traditional concurrency features

- Origin of this work was a student MS thesis

  - Earlier work had studied how channel-based concurrency was used in Go programs (see “An Empirical Study of Message Passing Concurrency in Go Projects” by Dilley and Lange from SANER 2019)

  - Student’s Focus: How do people use traditional concurrency features, like mutex and condition variables? Do they?

“Enabling Go Program Analysis in Rascal”, Swearngan and Hills, SCAM 2023 (Engineering Track).

First Idea: Just write this in Go!

- Go includes several libraries for working with Go programs, so it's fairly easy to get started
  - The `go/ast` library defines all the interfaces (e.g., `Expr`) and structures (e.g., `SelectorExpr`) for Abstract Syntax Tree nodes
  - The `go/parser` library lets you parse Go code and get back an AST
  - The `go/token` library defines all the lexical tokens in the language
- So, just create a Visitor, walk the AST, and collect the info — done!

"Enabling Go Program Analysis in Rascal", Swearngan and Hills, SCAM 2023 (Engineering Track).

The problem: Matching AST nodes

NOTE: We are looking for something like: var wg sync.WaitGroup

```
func matchWaitGroupDecl(x *ast.GenDecl, v *Visitor, n ast.Node) {
    for i := 0; i < len(x.Specs); i++ {
        if spec, ok := x.Specs[i].(*ast.ValueSpec); ok == true {
            if spec.Type != nil {
                if t, ok := spec.Type.(*ast.SelectorExpr); ok == true {
                    if tsel, ok := t.X.(*ast.Ident); ok == true {
                        if tsel.Name == "sync" && t.Sel.Name == "WaitGroup" {
                            for j := 0; j < len(spec.Names); j++ {
                                id := spec.Names[j]
                                v.addDef(createDecl(id.Name, WaitGroup))
                                v.state.addWaitGroupDecl()
                            }
                        }
                    }
                }
            }
        }
    } } } } // all on one line so this fits on a slide!
```


“Enabling Go Program Analysis in Rascal”, Swearngan and Hills, SCAM 2023 (Engineering Track).

The problem: Matching AST nodes

- Note: the code on the prior slide is not **bad**, it is just very verbose!

  - `spec, ok := x.Specs[I].(*ast.ValueSpec)` is a *type assertion*: we want to make sure that spec (which is just defined as being of interface type `Spec`) is of a certain concrete type (a `ValueSpec`) — this is essentially a downcast

  - We then check to see if `ok == true`, which means that the type assertion passed and `spec` can now be treated as a value of that type (which it must be if this worked) — if we just do the assertion without the `ok` check, this will panic (i.e., crash) if the assertion fails

“Enabling Go Program Analysis in Rascal”, Swearngan and Hills, SCAM 2023 (Engineering Track).

Is there a better way?

- Rascal is designed for these kinds of applications!

- The following is the Rascal version of what was inside the for loop in the example Go code:

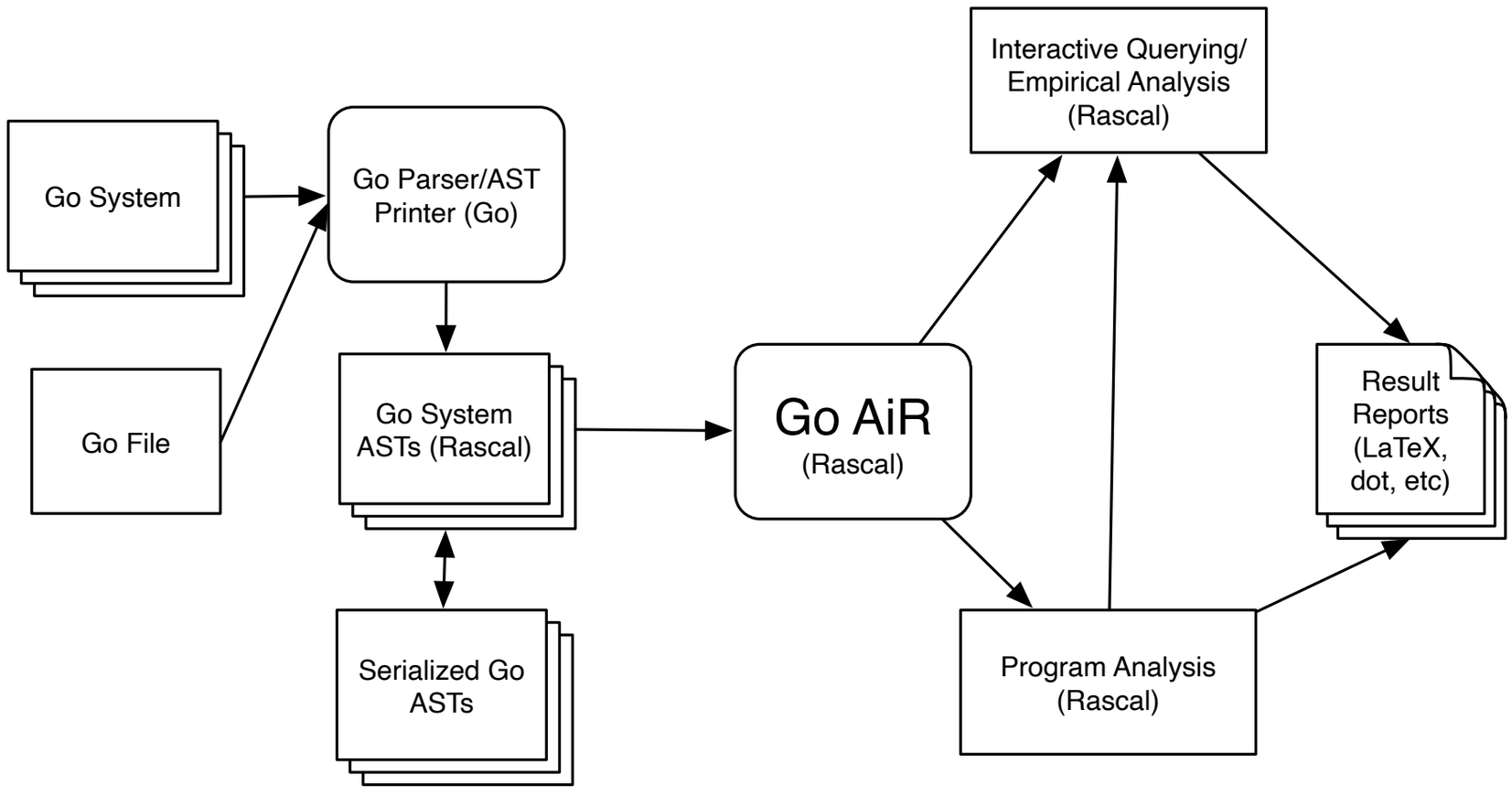
```
if (valueSpec(names,someExpr(selectorExpr(ident("sync"),"WaitGroup")),_) := d) {
  featureDecls = featureDecls
    + { < d.at, featureDecl(d.at, n, waitGroupDecl())> | n <- names };
  }
}
```

- Pattern matching gives us a natural way to work with AST terms, built-in relation types and comprehensions help us with fact extraction and analysis

"Enabling Go Program Analysis in Rascal", Swearngan and Hills, SCAM 2023 (Engineering Track).

Go AiR

- Go AiR (Analysis in Rascal) is a prototype analysis framework for Go



“Enabling Go Program Analysis in Rascal”, Swearngan and Hills, SCAM 2023 (Engineering Track).

What can we currently do?

- We can extract ASTs from Go source code (using a Go program to do this) and read them into Rascal, either for individual files or entire systems (tested across a large number of popular systems)

- We can serialize/deserialize these systems, along with additional extracted data

- We can explore Go code using Rascal's pattern matching features

- We can work with multiple releases of a system, based on Git version history

- We are moving earlier fact extraction code, written in Go, over to Rascal

"Enabling Go Program Analysis in Rascal", Swearngan and Hills, SCAM 2023 (Engineering Track).

What would we like to do?

- We want to redo our earlier work on traditional concurrency features and compare this to earlier work on message passing

- We want to integrate this with a rewriting logic semantics of Go, focused on concurrency, for concurrency analysis and verification (this is currently under construction)

- We want to extract models of concurrent behavior to help developers understand the possible behaviors of their code

"Enabling Go Program Analysis in Rascal", Swearngan and Hills, SCAM 2023 (Engineering Track).

Other Frameworks

Other Rascal program analysis frameworks

- Clair (C): https://github.com/usethesource/clair
- Python AiR (Python): https://github.com/cwi-swat/python-air
- JS-AiR (JavaScript): https://github.com/cwi-swat/js-air
- Ada AiR (Ada): https://github.com/cwi-swat/ada-air
- Other ongoing work includes frameworks for Lua (often used for game design and embeddable interpreters) and COBOL

Past and Current Collaborators (incomplete!)

- David Anderson (ECU)
- T. Baris Aktemur (Özyeğin University, Turkey)
- Marcelo d'Amorim (UFPE, Brazil)
- Jeroen van den Bos (CWI, NFO)
- Feng Chen (UIUC)
- Ben Givens (Hanover)
- Maurits Henneke (ippz)
- Paul Klint (CWI)
- Dimitris Kyritsis (UvA)
- Lindsey Lanier (ECU)
- Patrick Meredith (UIUC)
- Chris Mulder (UvA, Hyves)
- Grigore Rosu (UIUC)
- Ioana Rucareanu (UvA)
- Traian Serbanuta (UAIC, Romania)
- Tijs van der Storm (CWI)
- Apil Tamang (ECU)
- Frank Tip (Northeastern)
- Alex Vilkomir (ECU)
- Jurgen Vinju (CWI)

Discussion

Thank you!
Any Questions?

- PLSE Lab GitHub: https://github.com/PLSE-Lab
- Rascal: https://www.rascal-mpl.org/
- Me: https://cs.appstate.edu/hillsma/

Appalachian STATE UNIVERSITY