Rascal: A One-Stop-Shop for Program Analysis and Transformation

Mark Hills

SET Seminar
November 20, 2012
Eindhoven, The Netherlands

http://www.rascal-mpl.org
Rascal: A Meta-Programming One-Stop-Shop

• Context: wide variety of programming languages (including dialects) and meta-programming tasks

• Typical solution: many different tools, lots of glue code

• Instead, we want this to all be in one language, i.e., the “one-stop-shop”

• Rascal: domain specific language for program analysis, program transformation, DSL creation
Usage Scenarios

• Parsing (briefly!)

• DSLs

• Software Repository Mining

• Program Analysis

• Visualization

• Many others...
Parsing
Parsing

- Rascal grammar definition language, GLL parsing
- Filtering rules written in Rascal provide disambiguation
  - Example: C’s famous T *b, need a symbol table
- Other features: implode to AST, track source locations
- Parsing integrates with IDE support: provides parse trees needed by IDE functionality, annotations on tree trigger IDE functionality
Domain-Specific Languages
Domain-Specific Languages

• DSLs support domain-level concepts, syntax familiar to practitioners

• Many familiar examples from tech space: SQL for database access, HTML for web pages, ATL for model transformations

• Some not so familiar: S3QL in Bioinformatics, Cg for graphics programming
Another Domain: Digital Forensics (Jeroen van den Bos)

• From Wikipedia: “Digital forensics is a branch of forensic science encompassing the recovery and investigation of material found in digital devices, often in relation to computer crime.”

• Challenges: need custom software, engineered to specific requirements (including for legal reasons), that performs well

• Research Question: can model-driven techniques be used to create fast, maintainable digital forensics software?
File carving

• File carving is the process of recovering files without the help of (file system) storage metadata.

• A file carver typically consists of two parts:
  • The carver itself, which selects and/or combines blocks of data from the input as candidate files.
  • A set of format validators that determine whether a candidate file is of any of the formats they validate.

Slide from Jeroen van den Bos
Describing File Formats in Derric

1. Header
   Name and encoding/type defaults

   format PNG
   strings ascii
   size 1
   unit byte
   sign false
   type integer
   order lsb0
   endian little

2. Sequence
   Data structure ordering

   sequence

3. Structures
   Layout of individual data structures

   structures

   IHDR {
     l: lengthOf(d)
     size 4;
     n: “IHDR”;
     d: { ... }
     c: checksum
     (...) size 4; }

Slide from Jeroen van den Bos
Describing File Formats in Derric

structures

Chunk {
    length: lengthOf(chunkdata) size 4;
    chunktype: type string size 4;
    chunkdata: size length;
    crc: checksum(algorithm="crc32-ieee",
        fields=chunktype+chunkdata) size 4;
    end: 0xFF3F;
}
IHDR = Chunk {
    chunktype: "IHDR";
    chunkdata: {
        width: !0 size 4;
        height: !0 size 4;
        bitdepth: 1|2|4|8|16;
        imagesize: (width*height*bitdepth)/8 size 4;
        interlace: 0|1;
    }
}
Applying Derric

• Each format has one/several descriptions.

• Code generator uses descriptions:
  • Applies (domain-specific) optimizations/transformations.
  • Runs quickly, so easy to rerun after changes.
  • May output for multiple targets.
Applying Derric

- Runtime system uses generated validators:
  - Recognizes, extracts or ignores files.
  - Implements algorithms and common optimizations.
Excavating Architecture

Slide from Jeroen van den Bos
Comparing to Existing Tools on a Set of Benchmarks

files Recovered (count)

Processing speed (MB/second)

Excavator
ReviveIt
PhotoRec
Scalpel

“Bringing Domain-Specific Languages to Digital Forensics”, van den Bos/van der Storm, ICSE’11.

Slide from Jeroen van den Bos
Software Repository Mining
Repository Mining

• “The Mining Software Repositories (MSR) field analyzes the rich data available in software repositories to uncover interesting and actionable information about software systems and projects.” -- MSR 2013

Homepage

• Repositories: version control, defect tracking, communications between team members

• Uses: support maintenance, improve design, facilitate reuse, empirical validation, prediction and planning
Example: What Features are Used in PHP?

• Goal: determine which features are used in PHP programs, what usage patterns are visible

• Special focus: which features are hard to analyze?

• Technique: extract system source from PHP repositories, perform statistical analysis over code bases of systems, use Rascal to identify interesting parts of code that we can look at more closely

• Corpus: 19 systems, close to 3.4 million lines of PHP
Results

• Of 109 language features, 7 are never used in the corpus, while 35 are not used in 80% of the files

• Most PHP files are below 1300 lines of code

• Most uses of variable-variables can be resolved statically (75% with systems that support PHP5)

• And more! (ask for the paper if you are interested...)
Program Analysis
Program Analysis

• Program analysis is an umbrella for a large number of techniques to programmatically discover information about programs

• Two camps: static and dynamic (with some mixing at the borders)

• Many techniques: abstract interpretation, control-flow analysis, data-flow analysis, augmented type systems (including type and effect systems), constraints

• Many uses: bug finding, optimization, verification
Example: Analysis of Lua Code (Reimer van Rozen)

• Lua is a popular scripting language, including for games

• Standard dynamic language challenges: no types, checks at runtime, high flexibility can lead to unexpected results

• Solution: static analysis of Lua in Rascal, with IDE tooling
Lua AiR Framework

Contextual Analysis Example

1. `function f(c)`          -- assign function to f
Lua AiR Framework

Contextual Analysis Example

1. `function f(c)` -- assign function to `f`
2. `a = 1` -- creates global `a`

Slide from Riemer van Rozen
Lua AiR Framework

Contextual Analysis Example

1. `function f(c)` -- assign function to f
2. `a = 1` -- creates global a
3. `local b = true` -- creates local b

Slide from Riemer van Rozen
Lua AiR Framework

Contextual Analysis Example

1. \texttt{function f(c)} \quad -- assign function to \texttt{f}
2. \texttt{a = 1} \quad -- creates global \texttt{a}
3. \texttt{local b = true} \quad -- creates local \texttt{b}
4. \texttt{a, b = b, a} \quad -- swap \texttt{a} and \texttt{b}
Lua AiR Framework

Contextual Analysis Example

1. `function f(c)` -- assign function to f
2. `a = 1` -- creates global a
3. `local b = true` -- creates local b
4. `a, b = b, a` -- swap a and b
5. `a, b = 1,2,3` -- discards 3
Lua AiR Framework

Contextual Analysis Example

1. function \( f(c) \)  -- assign function to \( f \)
2. \( a = 1 \)  -- creates global \( a \)
3. \texttt{local} \( b = \texttt{true} \)  -- creates local \( b \)
4. \( a, b = b, a \)  -- swap \( a \) and \( b \)
5. \( a, b = 1,2,3 \)  -- discards 3
6. \( a, b = c \)  -- implicitly deletes \( b \)
Lua AiR Framework

Contextual Analysis Example

1. `function f(c)` -- assign function to f
2. `a = 1` -- creates global a
3. `local b = true` -- creates local b
4. `a, b = b, a` -- swap a and b
5. `a, b = 1, 2, 3` -- discards 3
6. `a, b = c` -- implicitly deletes b
7. `print(b)` -- nil, undeclared b
1. `function f(c)`  -- assign function to f
2. `a = 1`  -- creates global a
3. `local b = true`  -- creates local b
4. `a, b = b, a`  -- swap a and b
5. `a, b = 1,2,3`  -- discards 3
6. `a, b = c`  -- implicitly deletes b
7. `print(b)`  -- nil, undeclared b
8. `end`  -- close scope
Lua AiR Framework

Contextual Analysis Example

1. `function f(c)` -- assign function to f
2. `a = 1` -- creates global a
3. `local b = true` -- creates local b
4. `a, b = b, a` -- swap a and b
5. `a, b = 1, 2, 3` -- discards 3
6. `a, b = c` -- implicitly deletes b
7. `print(b)` -- nil, undeclared b
8. `end` -- close scope
9. `f("4")` -- call f, bind c to "4"

Slide from Riemer van Rozen
Lua AiR Framework

Contextual Analysis Example

1.   function f(c)       -- assign function to f
2.     a = 1            -- creates global a
3.   local b = true     -- creates local b
4.   a, b = b, a        -- swap a and b
5.   a, b = 1, 2, 3     -- discards 3
6.   a, b = c           -- implicitly deletes b
7.   print(b)          -- nil, undeclared b
8.   end               -- close scope
9.   f("4")           -- call f, bind c to "4"
10.  print(a)          -- 4, read global a
1. function \( f(c) \) -- assign function to \( f \)
2. \( a = 1 \) -- creates global \( a \)
3. local \( b = true \) -- creates local \( b \)
4. \( a, b = b, a \) -- swap \( a \) and \( b \)
5. \( a, b = 1,2,3 \) -- discards \( 3 \)
6. \( a, b = c \) -- implicitly deletes \( b \)
7. print(b) -- nil, undeclared \( b \)
8. end -- close scope
9. \( f("4") \) -- call \( f \), bind \( c \) to "4"
10. print(a) -- 4, read global \( a \)
11. \( d = 2 .. a \) -- coerces 2 to string
Lua AiR Framework

Contextual Analysis Example

1. `function f(c)` -- assign function to f
2. `a = 1` -- creates global a
3. `local b = true` -- creates local b
4. `a, b = b, a` -- swap a and b
5. `a, b = 1, 2, 3` -- discards 3
6. `a, b = c` -- implicitly deletes b
7. `print(b)` -- nil, undeclared b
8. `end` -- close scope
9. `f("4")` -- call f, bind c to "4"
10. `print(a)` -- 4, read global a
11. `d = 2 .. a` -- coerces 2 to string
12. `d = d / "12"` -- coerces 12 to number

Slide from Riemer van Rozen
Lua AiR Framework

Contextual Analysis Example

1. `function f(c)` -- assign function to f
2. `a = 1` -- creates global a
3. `local b = true` -- creates local b
4. `a, b = b, a` -- swap a and b
5. `a, b = 1, 2, 3` -- discards 3
6. `a, b = c` -- implicitly deletes b
7. `print(b)` -- nil, undeclared b
8. `end` -- close scope
9. `f("4")` -- call f, bind c to "4"
10. `print(a)` -- 4, read global a
11. `d = 2 .. a` -- coerces 2 to string
12. `d = d / "12"` -- coerces 12 to number
13. `print(c, d)` -- nil 2, undeclared c

Slide from Riemer van Rozen
Example: Lua IDE

Slide from Riemer van Rozen
Example: Analysis of PHP (Ongoing Work)

• Eventual goal: full suite of PHP analysis tools

• Current work: the basics!
  
  • Analysis of file includes
  
  • Type inference
  
  • Alias analysis

• Some promising initial work on statically resolving includes, which are a dynamic property
Visualization
How to integrate Software Visualization in Rascal?
public IValue visitMap(IMap o) throws VisitorException {
    append('C');
    Iterator<IValue> mapIterator = o.iterate().keySet().iterator();
    if (mapIterator.hasNext()) {
        IVValue key = mapIterator.next();
        key.accept(this);
        append(':');
    }
    o.get(key).accept(this);
    append('C');
    return o;
}

while (mapIterator.hasNext()) {
    append(',');
    key = mapIterator.next();
    key.accept(this);
    append(':');
    o.get(key).accept(this);
}

append('C');

public IVValue visitNode(INode o) throws 
String name = o.getName();
    if (name.indexOf('-') != -1) {
        append('\\');
    }
    append(name);
Software Visualization: Execution Frequency

Credits: Steven Eick
Slide from Paul Klint
Software Visualization: Revision Histories

Credits: Alex Telea, RUG

Slide from Paul Klint
Software Visual Analytics

• Emerging field where data extracted from software artifacts are visualized in order to

  • **Understand** the software: Architecture? Component dependencies?

  • **Identify** parts with special properties: Most complex? Most revisions? Test coverage?

  • **What if** questions: What happens if we adapt this part?
Rascal Visualization Design Principles

• Automatic & Domain-Specific: reduce low-level issues (layout, size), automate mappings (e.g., axis, color scale, ...), automate interaction support

• Reuse: treat figures and visual attributes as ordinary values; can be parameters/result of functions, arbitrary nesting of figures, well-defined composition of visual attributes
Rascal Visualization Design Principles

• Compositionality: global visualization state (e.g. Pen color) hinders composition, self-contained, composable, visualizations

• Interactivity: enable Schneidermann's Mantra of Overview First, Zoom and Filter, then Details-on-demand, provide the GUI-elements (buttons, text fields, ...) to achieve this.
And, of course, the ultimate goal...
Ideas for Assignments
Assignment Ideas: Grab-bag, needs work...

• Parsing -- see Ali Afroozeh’s talk from November 13

• DSL construction -- challenge here is coming up with something novel and useful in the limited timeframe

• Data-rich programming: add support for new formats, like RDF -- challenge is you really need something useful to do with it

• IDE support: can we use information in IDEs for other languages to provide support similar to what we have with Java?
Assignment Idea #1: Taint Analysis in PHP

• Problem: user inputs in GET and POST should not be used directly in database queries


• Analysis: verify that, along all paths, steps are taken to sanitize strings before they are used in queries

http://xkcd.com/327/
Assignment Idea #2: MSR

• Context: major changes from PHP4 to PHP5, many upgraded systems

• Question 1: How have OO features been adopted?

• Question 2: Does this lead to differences in popular code quality metrics?

• Question 3: Can information in the repository be tied into support of new features and language changes?

• Question 4: Can we identify committers that are improving quality metrics?
Thank you!

Any Questions?

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