Source Code Analysis

Dave Binkley
Loyola College in Maryland

Special thanks to Mark Harman without whom this paper would not exist in its present form. He is also credited with the idea described in Section 4 at the end of the confluence of static and dynamic analysis. Special thanks also to Rainer Koschke for his insights and information and to Alex Aiken, Jeri Hanly, Mike Hinchee, Dawn Lawrie, Phil Pfeiffer, Tom Reps, and Josep Silva for their detailed comments on drafts of the paper. Thanks also to the following for their comments and suggestions on the future or source-code analysis: Saswat Anand, Paul Anderson, V. J. Benokraitis, Darren Curry, Tom Dean, Florian Deissenböck, Massimiliano Di Penta, Matt Dwyer, Michael Ernst, David Etzioni, Keith Gallagher, Tibor Gyimothy, Mary Jean Harrold, Gorel Hedin, Michael Hind, Kevin Hunter, Gaspar Justin, Jens Krinke, Arun Lakhotia, Spirou Mancoridis, Tom McKenzie, Phil McMinn, Tom Mens, David Notkin, Alex Orso, Mauro Pезzo, Ross Raszelewski, Daniel Raitu, Barbara Ryder, Kamran Sartipi, Silvije Schupp, Harry Sneed, Eleni Stoulia, Paolo Tonella, Jason Tyler, Tao Xie, and Xiaofang Zhang. This work is supported by National Science Foundation grant CCR-0305330.

Art ?

is defined as
Source Code?

is defined as

```c
main()
{
    printf("Hello World\n");
}
```

Source Code

Working Definition

any static, textual, human readable, fully executable description of a computer program that can be compiled automatically into an executable form
Source Code Analysis

Extracting information from source code (or artifacts generated from the source code) using automatic tools

Output is **not** source code (no transformation)

**But, ...** to support dynamic analysis include as “source code” documents needed to execute the program, such as program inputs.

---

Wanta Take This Outside?

- Discussions of such definitions offline
- Preferably over a pint
Perspective

Means to an end

If you build it they will come

Outline

- Source code analysis components
  1. Parsing
  2. Internal representation
  3. Analysis
- Current techniques
- Future techniques
Parsing the Source

Spectrum
- Full
  ✓ match “the” language
- Regular Expression
  ✓ deals with dialects, syntax errors

<table>
<thead>
<tr>
<th></th>
<th>accurate</th>
<th>complete</th>
<th>flexible</th>
<th>robust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RE</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Best of both Worlds
Island Grammars
- Two parts
  – *detailed* productions for constructs of interest
  – *liberal* productions that catch remainder

Parse tree from “standard” grammar
- has structure everywhere

Parse tree from island grammar
- only has structure for islands
Component 2
Internal Representation

• More convenient
• Faster to work with
• Examples
  – Graphs (CFG, PDG, SDG, SSA, VDG)
  – XML

Component 3
Analysis

• 6 dimensions to consider
  – Static vs Dynamic
  – Sound vs Unsound (another “take it outside” discussion)
  – Safe vs Unsafe
  – Flow – Sensitive vs Insensitive
  – Context – Sensitive vs Insensitive
  – Complexity – Precision vs Effort

Examples follow
Applications

- architecture recovery [8, 105, 21]
- assertion discovery [37]
- automotive software engineering† [94]
- clone detection [80, 71, 21]
- comprehension [19, 28, 101, 21]
- debugging [108, 90, 45, 24]
- empirical software engineering research† [110]
- fault location [83, 64]
- middleware† [61]
- model checking in formal analysis† [34]
- model-driven development† [41]
- optimization techniques in software engineering† [48]
- performance analysis† [117]
- program evolution [5, 118]
- quality assessment [107, 66]
- reverse engineering† [21]
- safety critical† [54]
- software maintenance [52, 30]
- software reliability engineering† [78]
- software versioning [112, 43]
- specification semantics [113]
- symbolic execution [2, 67]
- testing† [10, 53]
- tools and environments† [119]
- validation (conformity checking)†
- verification, sound formal [18]
- verification, unsound syntactic [17, 29, 65]
- visualizations of analysis results [14, 93, 99, 4, 16]
- web application development† [93]

Details and Examples

- Other FoSE sessions
- FoSE papers
- Cited works

Looking Forward

- Guiding Principals
- Current Source Code Analysis Challenges
- Future Source Code Analysis Challenges
Guiding Principal 1

• Power Hungary is Good
  – Moore’s Law continues
    • especially for the parallelizable tasks

Guiding Principal 2

• Less is more
  – Simple language with good tool support viable?
  – Pascal? Ada? Ruby??
  – C++? Java/5?
    • A Feature of DMS –
      type accurate C++ Src Browsing
Guiding Principal 3

• Symbiotic Relationships
  – Exploit the best of man and machine

A sampling of Current Challenges

• Pointer Analysis
• Concurrent Analysis
• Dynamic Approximation
• Executable Analysis
• Information Retrieval
• Data Mining
• Confluence of static and dynamic analysis
• Self Healing Code
• Real Time Analysis
Dynamic Approximation of Static Information

Look for patterns in computed values
- Instrument
- Run
- Extract potential invariants from data traces
- Check them

Dimension 2: Unsound (but useful) vs. Sound

Example
(taken from “The Science of Programming”)

\[
i = 0 \\
s = 0 \\
do \ i \neq n \rightarrow \ i = i + 1 \\
s = s + b[i]
\]

Precondition
\[ n \geq 0 \]
Postcondition
\[ s = (\Sigma j : 0 \leq j < n : b[j]) \]
Loop Invariant
\[ 0 \leq i \leq n \text{ and } \]
\[ s = (\Sigma j : 0 \leq j < i : b[j]) \]
Daikon results
(on 100 randomly generated input arrays of length 7-13)

• ENTER
  – N = size(B)
  – N in [7 ... 13]
  – B - All elements ≥ -100

• EXIT
  – N = I = orig(N) = size(B)
  – B = orig(B)
  – S = sum(B)
  – N in [7 ... 13]
  – B - All elements ≥ -100

• LOOP
  – N = size(B)
  – S = sum(B[0 ... I -1])
  – N in [7 ... 13]
  – I in [0 ... 13]
  – I ≤ N
  – sum(B) in [-556.539]
  – B[0] nonzero in [-99.96]

Information Retrieval in Source Code Analysis

• Impact of origin in complier analysis
• Look outside this box

Never, ever, think outside the box
Information Retrieval
The Three Layers of a Program

<table>
<thead>
<tr>
<th>Concepts Layer</th>
<th>Purity</th>
<th>Fatal Polysemy</th>
<th>Logical Redundancy</th>
<th>Polysemy</th>
<th>Synonymy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a)</td>
<td>b)</td>
<td>c)</td>
<td>d)</td>
<td>e)</td>
</tr>
</tbody>
</table>

Less is more? Cost of identifier naming freedom?

D. Ralu, F. Deissenboeck, How Programs Represent Reality (and how they don’t), WCRE, 2006
Source Code Analysis FoSE © 2007 D. Binkley

Self Healing Code

Example

for each person p on list
if p->name == looking-for
return data [p->id]

Person is
ID
Name

42 Dave
-15 Joan
29 Judy

Source Code Analysis FoSE © 2007 D. Binkley
Self Healing Code

Original

42
Dave

-15
Joan

29
Judy

for each person p on list
if p->name == looking-for
return data [p->id]

(inferred) Invariant – \( ID > 0 \)

Repaired

42
Dave

-15
Joan

29
Judy

Perhaps not “correct”, but no core dump in the field!

A sampling of Future Challenges
10 years hence

- Analysis at link and run time
- Formal vs. informal methods the debate continues
- **Better metrics**
- Increase natural language analysis of source code
Better Metrics 😊

How much will that cost?  

Dimension 1: static vs dynamic

Ripple effect measures

Bad Smells

Long Parameter List

LTRACoeffSetup(double *coefflist, int listszize, double T,  
double firstvalue, double *valuelist,  
double curtime, double *timelist,  
int timeindex, int*auxindexptr)

Semantics Based Metrics

• Slice based metrics

• (Natural) Language based metrics

Query

Source Code Analysis FoSE © 2007 D. Binkley
A sampling of Future Challenges
the year 2025

• Everything you think, do, and say is tracked by *data mining* techniques that can predict the behavior of individual programmers and teams

A sampling of Future Challenges
the year 2025

• *Program size continues to grow*
• Moore’s Law: 8K *processors* on your desktop
• *Paradigm shift – next generation programming language*
Large Code based Requires Continued Work on Tool Support

- Design Maintenance System

Scale, Scale, Scale!
- Ameratize costs across
  - Projects
  - Languages

http://www.semdesigns.com/Company/Publications/

Source Code Analysis FuSE © 2007 D. Binkley

Large Code based Requires Tool Support

"Reducing Visualization Complexity Using Decomposition Slices ", Keith Gallagher, Liam O'Brien
Source Code Analysis FuSE © 2007 D. Binkley
Paradigm Shift?
Next Level Programming Language
For example “true” model-based languages
(Not model driven development)

Exist in some domains:
  RMPL – Reactive Model-based Programming Language
  Robotic Space Explorers

A sampling of
Future Challenges
50 years hence

• Software engineering is twice present age!
• Higher level languages
  (e.g., model based)
  – Look for ICSE 2057 “Model Analysis: A Road Map”
• DNA computing
• Quantum computing
Quantum Computing

- Consider a 3 bit register
- The bits are in a definite state, e.g., 101
- In a quantum computer, qubits are in a superposition of all the classical states
- Register is described by the wavefunction

$$|\psi\rangle = a|000\rangle + b|001\rangle + c|010\rangle + d|011\rangle + e|100\rangle + f|101\rangle + g|110\rangle + h|111\rangle$$

Quantum Computing

- Consider factoring the product of two 300-digit primes
  – computationally difficult
    (with an ordinary computer)
- A quantum computer could solve this problem with just over $2n$ qubits using Shor’s O((log N)^3) algorithm
My Thanks

• To those that
  – Sent papers and ideas
  – Sent comments
  – Sent slides
  – Enjoyed the presentation