Automating Dependability Analysis for Software Systems

John Heaps
Computer Science Department
Dependability

● Dependability in Software Engineering
  ○ Availability
  ○ Reliability
  ○ Safety
  ○ Security

● Dependability defined for Software Systems
  ○ Privacy Policies
  ○ Software Bugs

● Analysis of Dependability
  ○ Privacy policy compliance
  ○ Software bug detection
The Cost of Poor Dependability

● Financial Loss
  ○ In 2018 alone, software bugs cost the world economy over $1.7 trillion and impacted over 3.7 billion people

● Loss of Privacy
  ○ Facebook scandals have leaked private information for about 500 million users

● Loss of Life
  ○ The U.S. Patriot missile defense system did not detect an incoming missile due to inaccurate tracking calculation causing the loss of 28 U.S. military troops

● By verifying policy requirements and detecting bugs in code, many of the costs and damages caused by poor software Dependability can be eliminated
The Need For Automation

- The manual process to verify policy compliance and perform bug detection of code takes too long and requires too much information
  - Any system could have dozens or hundreds of privacy constraints
    - Medical systems must be in compliance with HIPAA
  - Any system could exhibit any number of different types of bugs
    - Currently, CWE defines 808 different types of bugs
    - A system could be tens of thousands to millions of lines of code in size
State of the Art: Static Analysis and Model Checking Techniques

- Model Checking
  - Construct specifications (usually in temporal logic) that define the constraints of the system
  - If any state of the system does not satisfy the specifications, it is reported as a bug

- Many of the most popular static analysis bug detection tools and techniques use Bug Patterns
  - Each pattern defines a bug or policy violation
  - Usually defined in the form of if-then
Model Checking Problems

- Model checking is not scalable
  - Subject to the state explosion problem

- Still need to manually generate specifications for policy and system or manually create a model of the system to be checked
Static Analysis Problems

● Static analysis requires patterns/specifications to be provided for it
  ○ Patterns and specifications must be manually defined for each bug that wants to be detected
  ○ Cannot detect bugs that do not have a pattern or specification defined for it

● Static analysis is conservative
  ○ Most static analysis techniques have a high false positive rate

● Static analysis is not always scalable
  ○ Some static analysis techniques require an exploration of all possible paths/states/etc. which is subject to the state explosion problem
Why Perform Deep Learning

- Many of the limitations of static analysis can likely be mitigated by deep learning:
  - Will learn features of code and bugs, so no patterns or specifications need to be defined
  - There is a good chance that new or different versions of learned bugs can still be detected
  - Will not be as conservative
  - Is not subject to the state explosion problem

- We should be able to utilize many existing deep learning techniques from natural language processing (NLP)
Background: Feed-Forward Neural Networks (FFNNs)
Background: Vector Representations

\[ \text{king} = \{2,0,3\} \]
\[ \text{queen} = \{2,2,2\} \]
\[ \text{man} = \{0,1,4\} \]
\[ \text{woman} = \{0,3,3\} \]

How do we learn these vector representations?
Source Text

The quick brown fox jumps over the lazy dog.

Training Samples

(the, quick)
(the, brown)
(quick, the)
(quick, brown)
(quick, fox)
(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)
(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)
Major Obstacles for Deep Learning on Software Code

- Syntactic structures are more complex
- The data sparsity problem is more severe
- The number of new code elements encountered is usually far greater than normal NLP
Toward a Semantic-Oriented Model

- Previous word embeddings based on definition of a language model was based on statistical frequencies of co-occurrences of code elements (a contextual-oriented model).

- To learn a new semantic-oriented model, a new type of relationship between words must be defined:
  - An equivalence relation can be defined between one word and a sequence of words by using the dictionary.

- By using this semantic model we can solve many previously mentioned limitations:
  - No longer affected by data sparsity (only need a word’s definition)
  - New token can be handled easily (only a definition needs to be provided for it).
Toward a Semantic-Oriented Model

```java
public static int max(int x, int y) {
    if(x > y) {
        return x;
    }
    return y;
}
```
Conclusion

- Automation of dependability analysis for software systems is important to prevent loss and feasibly perform analysis at large scale.

- Model checking and static analysis are state of the art, but have many limitations.

- Many of these limitations may be solved through the use of deep learning.
Questions
Overview

- Dependability Using Model Checking and Static Analysis
- Background: Deep Learning
- Toward Deep Learning on Software Code
- Toward a Semantic-Oriented Model
Policy Verification Obstacles

- **Policy**
  - Written in natural language
  - Natural language is ambiguous

- **Verification of Compliance**
  - Want a similar representation for comparison/analysis
  - Want formalized specifications

- **Software System**
  - Written in code
  - Code can have bugs
  - Code can be written in different programming languages
Improper Authorization Code Example

```java
public ResultSet runEmployeeQuery(Connection conn, String name){
    PreparedStatement stmt = conn.prepareStatement("SELECT * " +
    "FROM employees WHERE name = ?");
    stmt.setString(1, name);
    ResultSet ts = stmt.executeQuery()
    return rs;
}
...

// "canQueryEmployee()" returns true if current user is authorized to
// query the employees table.
if(AuthCheck.canQueryEmployee()){
    ResultSet employeeRecord = runEmployeeQuery(dbConn, employeeName);
}
```
Improper Authorization Pattern Detection Example

```java
public void sawOpcode(int seen) {
    if ("AuthCheck".equals(classConstant) &&
        seen == INVOKESTATIC &&
        "canQueryEmployee".equals(nameConstant) && "()Z".equals(sigConstant)) {
        seenGuardClauseAt = PC;
        return;
    }
    if (seen == IFEQ && (PC >= seenGuardClauseAt + 3 && PC < seenGuardClauseAt + 7)) {
        logBlockStart = branchFallThrough;
        logBlockEnd = branchTarget;
    }
    if (seen == INVOKEVIRTUAL && "runEmployeeQuery".equals(nameConstant)) {
        if (PC < logBlockStart || PC >= logBlockEnd) {
            bugReporter.reportBug(
                new BugInstance("IMPROPER_AUTHORIZATION", HIGH_PRIORITY)
                .addClassAndMethod(this).addSourceLine(this);
            }
        }
    }
}
Privacy Policy Verification Using Model Checking
Background: Artificial Neuron

\[ x_0 \rightarrow w_0 \rightarrow w_0 x_0 \rightarrow \text{axon from a neuron} \]

\[ w_1 x_1 \rightarrow \sum_{i} w_i x_i + b \rightarrow f (\sum_{i} w_i x_i + b) \rightarrow \text{output axon} \]

Cell body

Activation function

Output axon
Background: Feed-Forward Neural Networks (FFNNs)
Background: Feed-Forward Neural Networks (FFNNs)
Background: Vector Representations

- Neural networks and deep learning work great for numerical data, but can’t perform calculations on code because it is text
- Need to convert code to some numerical representation
- A vector representation is an $m$-dimensional real-valued vector representing the relative meaning of a word (compared to other words in the vocabulary)
  - $king = \{2,0,3\}$
  - $queen = \{2,2,2\}$
  - $man = \{0,1,4\}$
  - $woman = \{0,3,3\}$
  - Learns vector representations based on the language model
Background: Vector Representations

\[\text{king} = \{2,0,3\}\]
\[\text{queen} = \{2,2,2\}\]
\[\text{man} = \{0,1,4\}\]
\[\text{woman} = \{0,3,3\}\]

How do we learn these vector representations?
Background: Language Model

- A language model is a probability distribution of occurrence of a sentence (or sequence of words) or the next word in a sequence

\[ P(W) = P(w_1, w_2, w_3, \ldots, w_n) \]

\[ P(“I ran to the store for groceries”) \]

- Vector representations in deep learning attempt to model the meaning of words
Background: Language Model

- Predict the next word given a previous sequence of words
  
  “I” -> “ran”
  “I ran” -> “to”
  ...
  “I ran to the store for” -> “groceries”

- Limitations of the language model:
  - The data sparsity problem (need a huge corpus to learn on)
  - If new words are encountered after training they cannot be handled
On the Naturalness of Software (Hindel et al.)
- Software exhibits behavior like a natural language, and can therefore be treated like a natural language
- Can have natural language processing techniques and language model be applied to software
- Can learn software using deep learning similar to natural language

Previous deep learning on code can be split into two main categories:
- Models that apply word prediction exactly like NLP
- Models that use an abstract syntax tree (AST) and perform prediction using AST nodes
  - Perform better on average
Background: Recurrent Neural Networks (RNNs)
Need to Learn Word Embeddings Differently

- Previous word embeddings based on definition of a language model were based on statistical frequencies of co-occurrences of code elements (a *contextual-oriented model*)
<table>
<thead>
<tr>
<th>Node Types</th>
<th>Identifier</th>
<th>Code Elements Used As Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Creation Expression</td>
<td>array data type</td>
<td>&quot;new&quot; array data type</td>
</tr>
<tr>
<td>Array Type</td>
<td>&quot;[&quot;</td>
<td>array data type</td>
</tr>
<tr>
<td>Assignment Expression</td>
<td>assign operator</td>
<td>target and value expressions</td>
</tr>
<tr>
<td>Binary Expression</td>
<td>binary operator</td>
<td>left and right term expressions</td>
</tr>
<tr>
<td>Boolean Literal</td>
<td>&quot;true&quot; or &quot;false&quot;</td>
<td>&quot;boolean&quot;</td>
</tr>
<tr>
<td>Break Statement</td>
<td>&quot;break&quot;</td>
<td>associated loop or switch statement</td>
</tr>
<tr>
<td>Cast Expression</td>
<td>cast data type</td>
<td>expression being cast</td>
</tr>
<tr>
<td>Catch Clause</td>
<td>&quot;catch&quot;</td>
<td>exception types being caught</td>
</tr>
<tr>
<td>Char Literal</td>
<td>&quot;CHAR&quot;</td>
<td>&quot;char&quot;</td>
</tr>
<tr>
<td>Class Or Interface Declaration</td>
<td>name</td>
<td>&quot;class&quot; or &quot;interface&quot; and modifiers, interfaces, extension, and members</td>
</tr>
<tr>
<td>Conditional Expression</td>
<td>&quot;?&quot;</td>
<td>condition expression and &quot;else&quot;</td>
</tr>
<tr>
<td>Continue Statement</td>
<td>&quot;continue&quot;</td>
<td>associated loop statement</td>
</tr>
<tr>
<td>Do Statement</td>
<td>&quot;do&quot;</td>
<td>&quot;while&quot; and condition expression</td>
</tr>
<tr>
<td>&quot;double&quot;</td>
<td>&quot;DOUBLE&quot;</td>
<td></td>
</tr>
<tr>
<td>Enum Declaration</td>
<td>name</td>
<td>&quot;enum&quot; and all modifiers</td>
</tr>
<tr>
<td>Explicit Constructor Invocation</td>
<td>&quot;this&quot; or &quot;super&quot;</td>
<td>associated arguments and expressions</td>
</tr>
<tr>
<td>Field Declaration</td>
<td>field data type</td>
<td>initializations, assignments, and modifiers</td>
</tr>
<tr>
<td>For Each Statement</td>
<td>&quot;for&quot;</td>
<td>&quot;break&quot; and &quot;continue&quot; if present and variable and iterable types</td>
</tr>
<tr>
<td>For Statement</td>
<td>&quot;for&quot;</td>
<td>&quot;break&quot; and &quot;continue&quot; if present and initialization type, condition expression, and update expression</td>
</tr>
<tr>
<td>If Statement</td>
<td>&quot;if&quot;</td>
<td>condition expression and &quot;else&quot; if present</td>
</tr>
<tr>
<td>InstanceOf Expression</td>
<td>&quot;instanceof&quot;</td>
<td>expression and instance type</td>
</tr>
<tr>
<td>Integer Literal</td>
<td>&quot;INT&quot;</td>
<td>&quot;int&quot;</td>
</tr>
<tr>
<td>Long Literal</td>
<td>&quot;LONG&quot;</td>
<td>&quot;long&quot;</td>
</tr>
<tr>
<td>Method Call Expression</td>
<td>name</td>
<td>arguments</td>
</tr>
<tr>
<td>Method Declaration</td>
<td>name</td>
<td>modifiers, return type, parameters, thrown exceptions, and method body</td>
</tr>
<tr>
<td>Null Literal</td>
<td>&quot;null&quot;</td>
<td>&quot;void&quot;</td>
</tr>
<tr>
<td>Object Creation Expression</td>
<td>object data type</td>
<td>&quot;new&quot;, arguments, and anonymous class body if present</td>
</tr>
<tr>
<td>Primitive Type</td>
<td>primitive data type</td>
<td>associated expressions or declarations</td>
</tr>
<tr>
<td>Return Statement</td>
<td>&quot;return&quot;</td>
<td>associated expression</td>
</tr>
<tr>
<td>String Literal</td>
<td>&quot;String&quot;</td>
<td></td>
</tr>
<tr>
<td>Super Expression</td>
<td>&quot;super&quot;</td>
<td>associated expression</td>
</tr>
<tr>
<td>Switch Entry Statement</td>
<td>&quot;case&quot; or &quot;default&quot;</td>
<td>&quot;switch&quot; and associated label expression if present</td>
</tr>
<tr>
<td>Switch Statement</td>
<td>&quot;switch&quot;</td>
<td>selector expression and switch entry statements</td>
</tr>
<tr>
<td>Synchronized Statement</td>
<td>&quot;synchronized&quot;</td>
<td>associated expression</td>
</tr>
<tr>
<td>This Expression</td>
<td>&quot;this&quot;</td>
<td>associated expression</td>
</tr>
<tr>
<td>Throw Statement</td>
<td>&quot;throw&quot;</td>
<td>associated expression</td>
</tr>
<tr>
<td>Try Statement</td>
<td>&quot;try&quot;</td>
<td>resource expressions, catch clauses if present, and &quot;finally&quot; if present</td>
</tr>
<tr>
<td>Unary Expression</td>
<td>unary operator</td>
<td>associated expression</td>
</tr>
<tr>
<td>Variable Declaration Expression</td>
<td>variable data type</td>
<td>modifiers and initialization expression if present</td>
</tr>
<tr>
<td>Void Type</td>
<td>&quot;void&quot;</td>
<td>associated expressions or declarations</td>
</tr>
<tr>
<td>While Statement</td>
<td>&quot;while&quot;</td>
<td>&quot;break&quot; and &quot;continue&quot; if present and condition expression</td>
</tr>
</tbody>
</table>
If-Statement Example

Node Types | Identifier | Code Elements Used As Context
---|---|---
If Statement | "if" | condition expression and "else" if present
Word Embeddings

Word embeddings for Apache Shiro’s 99 most common code elements
Word Embeddings

Word embeddings for Apache Shiro's 99 most common code elements
Limitations of the Behavioral-Oriented Model

- Still subject to the data sparsity problem
- Still cannot handle new code elements
References
