Analyzing Software using Deep Learning

Sequence-to-Sequence Networks and their Applications (Part 2)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2020

Overview

- **Sequence-to-sequence networks**
- **API usage sequences for natural language queries** 

  Based on ”Deep API learning” by Gu et al., 2016
- **Interpreting Python programs**

  Based on ”Learning to execute” by Zaremba and Sutskever, 2014

Motivation

APIs are difficult to use

- Which methods to call?
- In what order to call them?

Developers ask questions, e.g., on stackoverflow.com

- Human effort required to answer them

Goal: Automatically suggest API usages based on natural language query

Idea

Formulate the problem as a translation problem

- Input: Sequence of natural language words
- Output: Sequence of API method calls
- Train and query sequence-to-sequence neural network

Example

Natural language query:

”match regular expressions”

Sequence of API calls expected as (possible) answer:

Pattern.compile, Pattern.matcher, Matcher.group

Training Data

- Analyze 443.000 Java projects from GitHub
- Focus on JDK = APIs of Java standard library
- Extract pairs of annotation and call sequence
- About 7 million extracted pairs
- Use 10.000 for testing and others for training

Example

```
/***
* Copies bytes from a large (over 2GB) InputStream to an
* OutputStream. This method uses the provided buffer, so
* there is no need to use a BufferedInputStream.
* @param input the InputStream to read from
* . . .
*/
public static long copyLarge(final InputStream input,
final OutputStream output, final byte[] buffer)
      throws IOException {
  long count = 0;
  int n;
  while (EOF != (n = input.read(buffer))) {
    output.write(buffer, 0, n);
    count += n;
  }
  return count;
}
```

Example

```
/***
* Copies bytes from a large (over 2GB) InputStream to an
* OutputStream. This method uses the provided buffer, so
* there is no need
* @param input the
* . . .
*/
public static long
final OutputStream
      throws IOExce
  long count = 0;
  int n;
  while (EOF != (n = input.read(buffer))) {
    output.write(buffer, 0, n);
    count += n;
  }
  return count;
}
```

Annotation:
”copies bytes from a large inputstream to an outputstream”
Call sequence:
InputStream.read , OutputStream.write

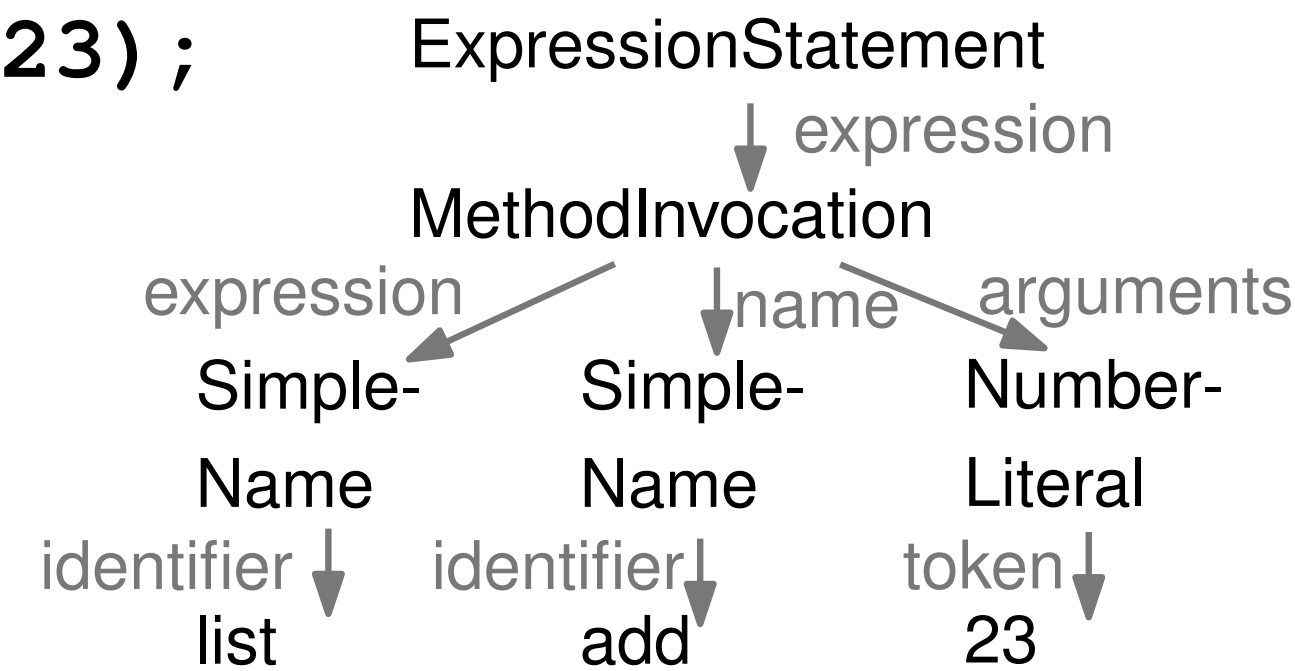
Extracting Annotations

- Extract JavaDoc of each method
- Extract first sentence
- Ignore methods without JavaDoc
- Ignore annotations with ”irregular” comments, e.g., `TODO: ...`

Extracting Call Sequences

- Goal: Lightweight analysis that scales to millions of code files
- Static, AST-based analysis with type bindings
- Example:

```
list.add(23);
```



AST-based Extraction (1)

- Constructor call:

  `new C()` → C.new (if `C` is JDK class)

- Method call:

  `obj.m()` → C.m (if type of `obj` is JDK class)

- Call expressions as arguments:

  `o1.m1(o2.m2())` → C2.m2, C1.m1

AST-based Extraction (2)

- Sequence of statements:

  `o1.m1(); o2.m2();` → C1.m1, C2.m2

- Conditionals:

```
if(o1.m1()) {
    o2.m2();
} else {
    o3.m3();
}
```

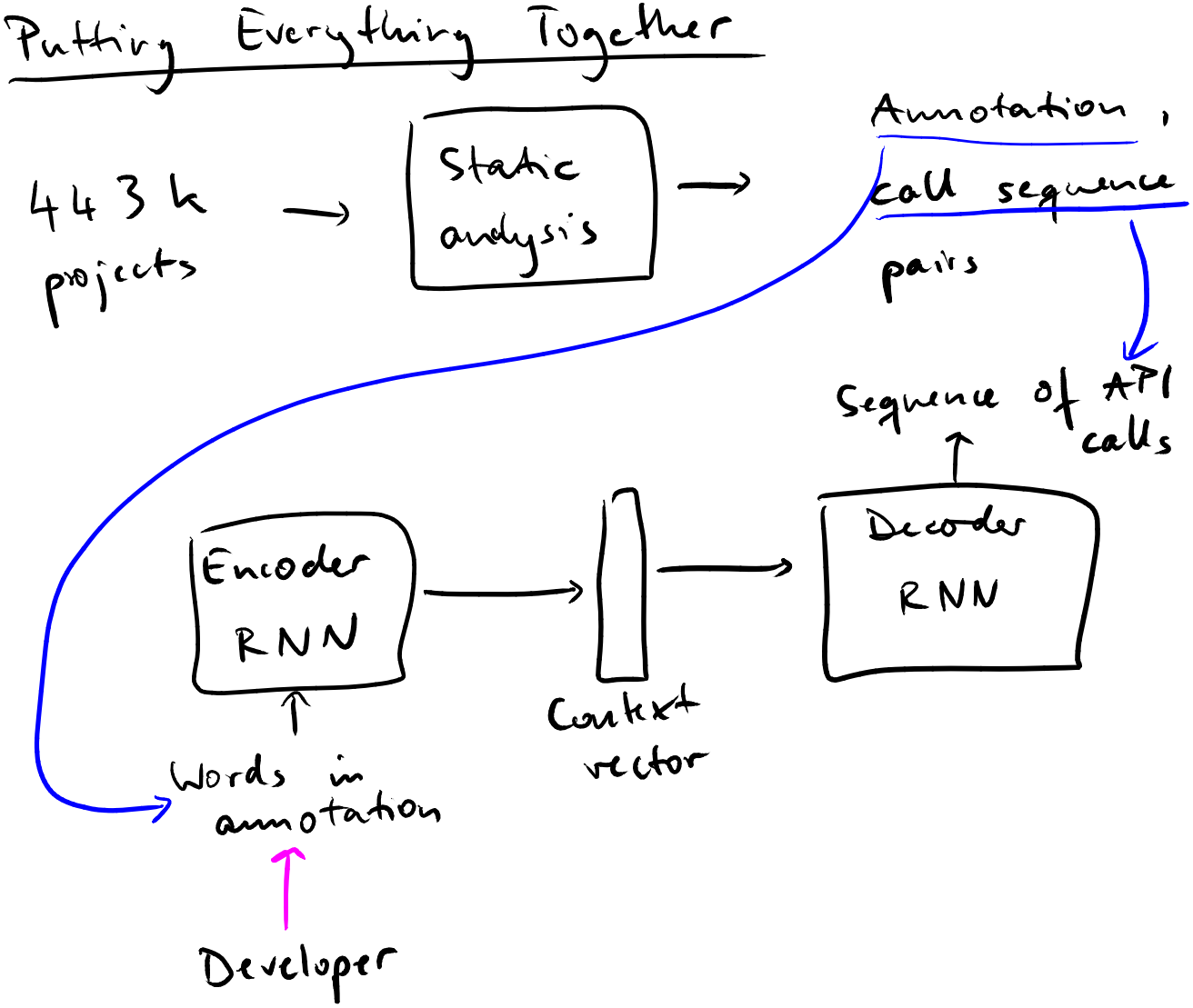
  → C1.m1, C2.m2, C3.m3

- Loops:

```
while(o1.m1()) { o2.m2(); }
```

  → C1.m1, C2.m2

Putting Everything Together



→ .. during training

→ .. API prediction

Examples

- ”generate md5 hash code”

  $\leadsto$ MessageDigest.getInstance, MessageDigest.update, MessageDigest.digest

- ”convert int to string”

  $\leadsto$ Integer.toString

- ”get files in folder”

  $\leadsto$ File.new, File.list, File.new, File.isDirectory