软件分析

机器学习与软件分析

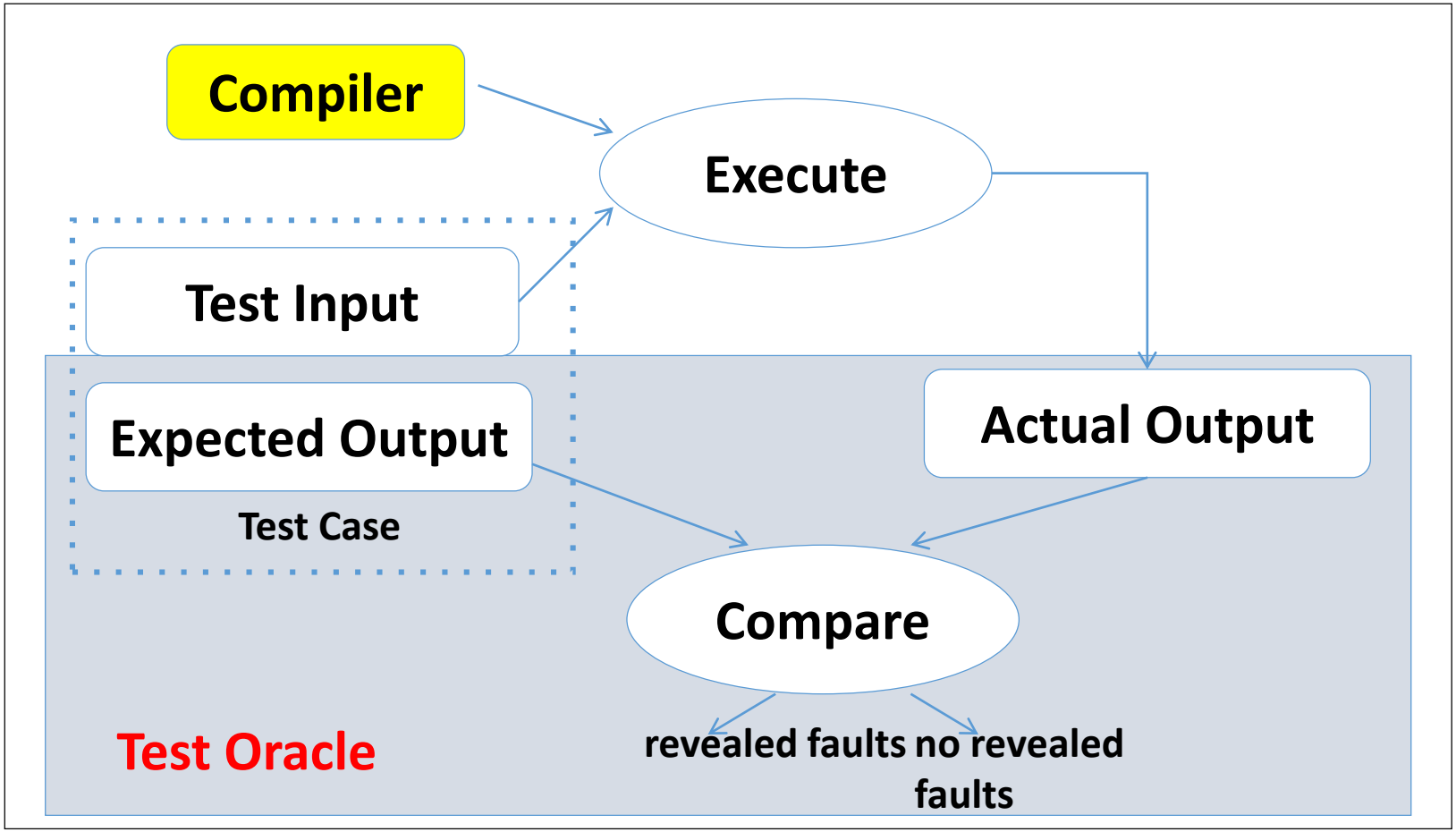
熊英飞
北京大学
2016

采用机器学习分析软件

- 主要挑战：
  - 程序并非简单的向量结构
  - 同一程序可以有多种不同写法
  - 软件制品中包含大量自定义标识符
- 主要解决方案：
  - 将问题转换成不需要考虑程序结构和自定义标识符的形式，同时尽可能减少同一程序的不同写法
  - 设计特定的算法来对程序结构进行处理

机器学习分析示例：编译器测试排序

Compiler Testing
--- guaranteeing compiler quality



Software Testing Process

随机编译器测试

- 测试输入：随机程序生成工具Csmith
  - 大型公司往往开发内部的随机程序生成工具
- 测试预言：
  - RDT: 用两个不同的编译器执行测试输入，看结果是否一致
  - DOL: 用编译器的不同优化级别，看结果是否一致
  - EMI: 把原程序变异成等价程序，看结果是否一致

随机编译器测试的问题

- 随机测试的效率很低
- RDT花了3年在GCC和LLVM上检查出来325个Bug
- EMI花了11个月在GCC和LLVM上检查出来147个Bug

如何加速编译器测试？

- 先跑能发现Bug的测试程序
- 先跑时间短的测试程序

- 假设程序发现错误的能力为C，运行时间为t，则C/t越大的程序越先执行

- 如何快速知道C和t？

用机器学习预测C和t

- 特征选择
  - 基本思路：复杂的程序编译时容易出错，且运行时间可能长
  - 存在特征
    - 是否存在某个类型的语句
    - 是否存在某个类型的变量
    - ……
  - 使用特征
    - 程序中变量被读取的次数
    - 程序中指针被引用的次数
    - 程序中指针被比较的次数
    - ……

用机器学习预测C和t

- 训练集
  - 在旧版本的GCC和LLVM上发现Bug的测试程序和没有发现Bug的测试程序
  - 各自1000个，保持数量平衡
- 预测C
  - 采用线性SVM模型
- 预测t
  - 采用高斯过程回归——一种回归模型
- 特征选择：预先采用特征选择来排除无效特征
- 运行结果：绝大多数时候加速30%-60%

编译器测试排序小结

- 从程序中提取数值型特征
  - 特征的大小表示某种可能和结果相关的量的大小
  - 特征中不包含程序结构
  - 特征中不包含自定义标识符

机器学习分析示例：API调用序列查询

Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, Sunghun Kim, Deep API Learning, FSE 2016

通过设定问题只考虑固定自定义标识符集合

API调用序列查询问题

$DocumentBuilderFactory.newInstance$

↓

$DocumentBuilderFactory.newDocumentBuilder$

↓

$DocumentBuilder.parse$

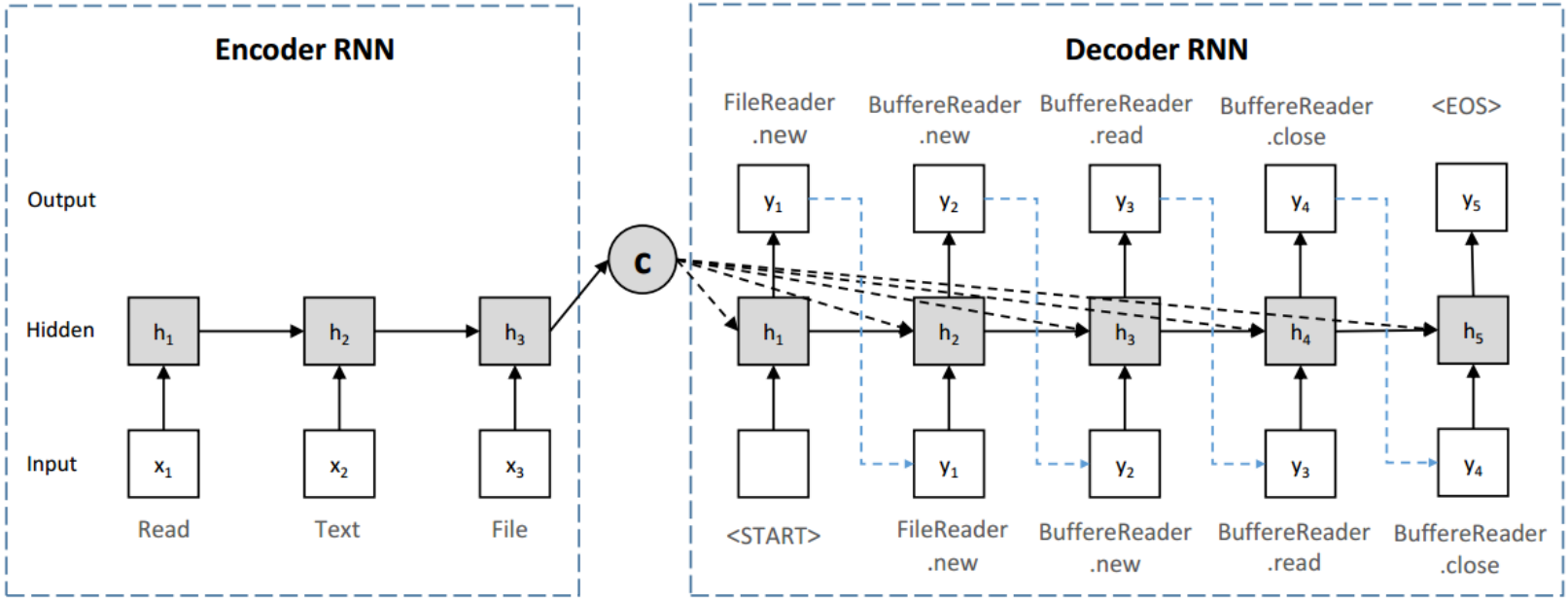
训练集收集

```java
/***
 * 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
 *  . . .
 * @since 2.2
 */
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;
}
```

API sequence: InputStream.read → OutputStream.write
Annotation: copies bytes from a large inputstream to an outputstream.

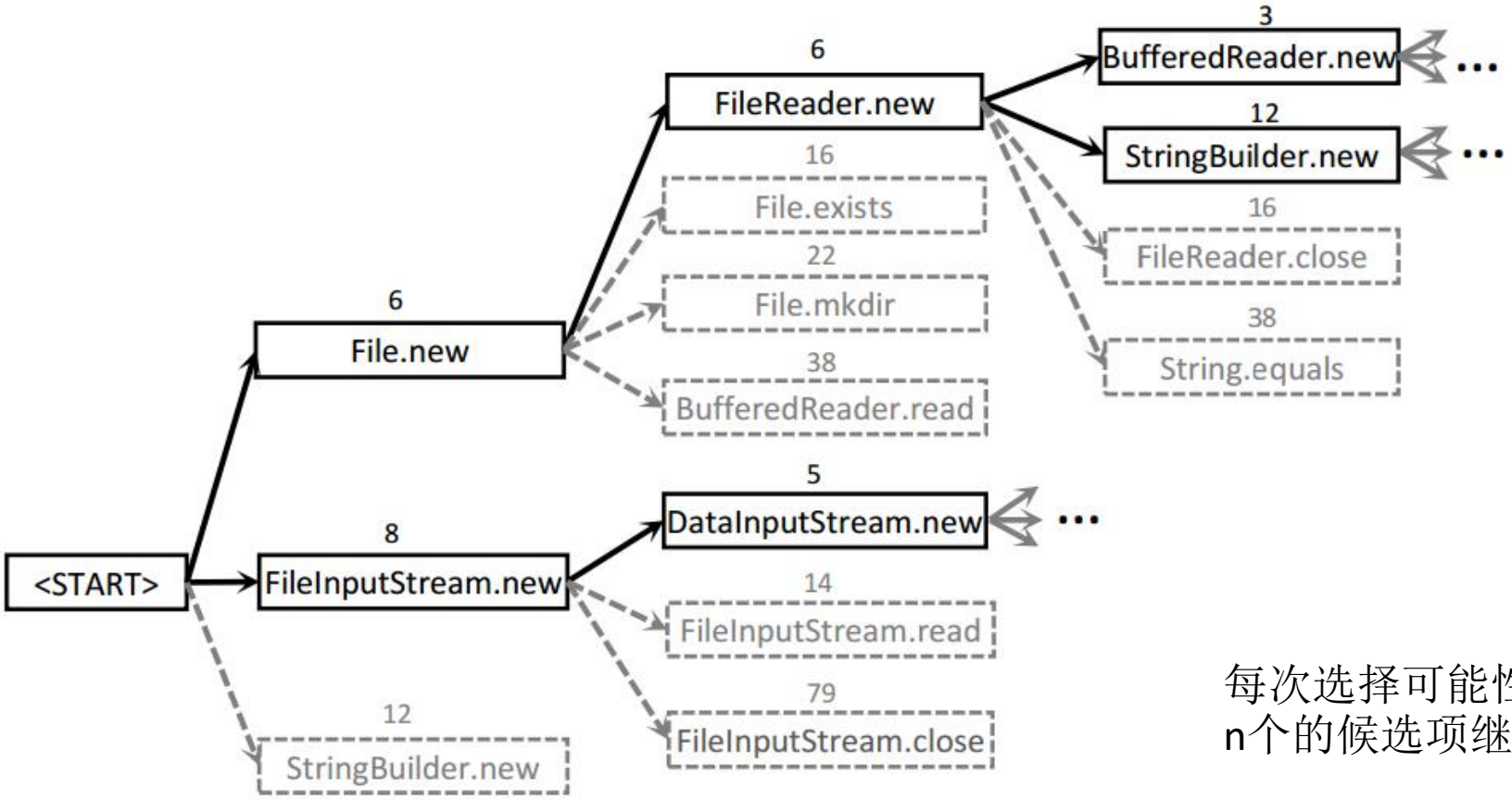
- 从控制流提取API调用序列
- 循环中的语句只考虑一次
- 注释第一句作为查询
- API方法为一个预先定义好的大小固定的集合

RNN翻译模型



- 利用一个单独的神经网络将单词映射成向量（称作Word Embedding）

基于Beam搜索的生成



每次选择可能性最大的n个的候选项继续展开

实验效果

- 串的匹配用BLEU指标来衡量
  - $BLEU = BP \prod_{n=1}^{N} \frac{\text{出现在参考输出中的}n-gram\text{的数量}}{\text{输出中}n-gram\text{的数量}}$
  - N为所考虑的n-gram的最大值
  - BP为较短序列的惩罚，令c为输出序列的长度，r为参考序列的长度，则 $BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$
  - 易见BLEU为0-1之间的数，越大越好
- 实验结果
  - BLEU=54.42%，之前的方法都不到20%

代码补全的统计模型

Probabilistic CFG

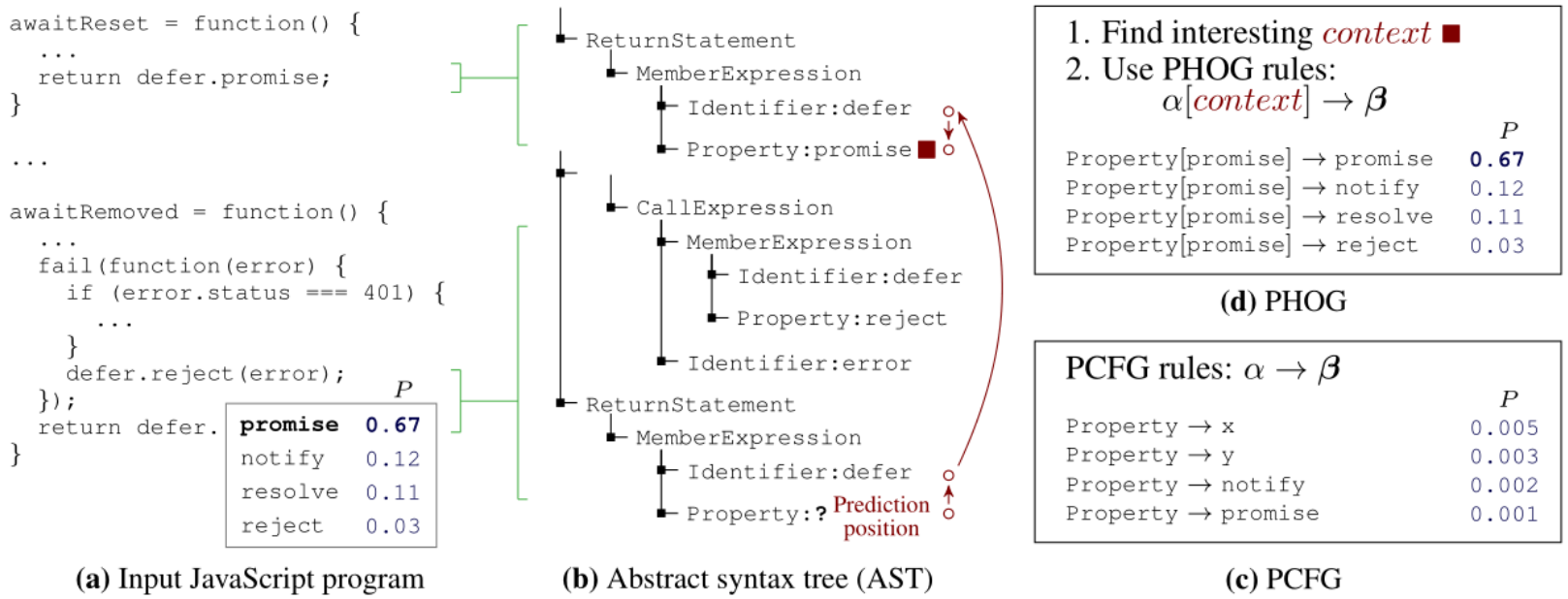
- 给每个推导式附一个概率
  - A ->B C　　0.8
  - -> C　　0.2
  - B -> X “,” B　　0.5
  - -> X　　0.5
- 同一个非终结符出发的推导式概率和为1
- 训练方法：
  - $\frac{\text{所有}AST\text{树上该推导式出现次数}}{\text{该推导式左边终结符出现次数}}$
- 可以用作代码补全

PHOG

- PHOG=Probabilistic Higher Order Grammar
- PCFG相当于用于预测的特征只有当前非终结符
- 成功的预测需要提取更多特征，称为Context

- 相关论文：
  - PHOG: Probabilistic Model for Code. Pavol Bielik, Veselin Raychev, Martin Vechev. ACM ICML 2016
  - Learning Programs from Noisy Data. Veselin Raychev, Pavol Bielik, Martin Vechev, Andreas Krause. ACM POPL 2016
  - Code Completion with Statistical Language Models. Veselin Raychev, Martin Vechev, Eran Yahav. ACM PLDI 2014

PHOG示例：javascript方法补全



提取同一个对象上的上一次操作作为Context

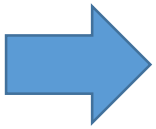
PHOG定义

- 非终结符[Context] -> 符号1 符号2 … 符号n　概率
- 同时存在一个函数，从已有的代码中提取Context
- 同一个非终结符和同一个Context的推导式概率和为1
- 训练方法：
  - $$\frac{\text{所有}AST\text{树上该推导式在该}context\text{下的出现次数}}{\text{该推导式左边终结符在该}context\text{下的出现次数}}$$

更复杂的Context示例

- 基于变量之前的n次使用情况预测下一个API调用

```
Stream s = new FileStream(“c:\book.txt”);
Reader r = new TextReader(s);
r.read();
s.?();
r.?();
```



s的context：
<new FileStream, ret>
<new TextReader, 1>

r的context：
<new TextReader, ret>
<read, 0>

变量使用位置编号：ret = 0.method(1, 2, 3, …)

Context定义语言

- 把Context获取过程定义为从当前结点出发的AST树的遍历

```
   Ops ::=  ϵ | Op Ops
    Op ::= WriteOp | MoveOp
WriteOp ::= WriteValue | WritePos | WriteAction
MoveOp ::= Up | Left | DownFirst | DownLast | PrevDFS |
           PrevLeaf | PrevNodeType | PrevActor
```

- 之前的示例可以写作

```
Left   PrevActor WriteAction WritePos
       PrevActor WriteAction WritePos
```

Left：左边的兄弟结点（即补全的目标对象）
PrevActor：前一次该变量的使用
WriteAction：使用时调用的方法
WritePos：该Actor在AST树中的位置

图统计模型

- PHOG的Context仍然是序列结构
- 代码的本质是图结构，能否在统计时把图结构利用起来？

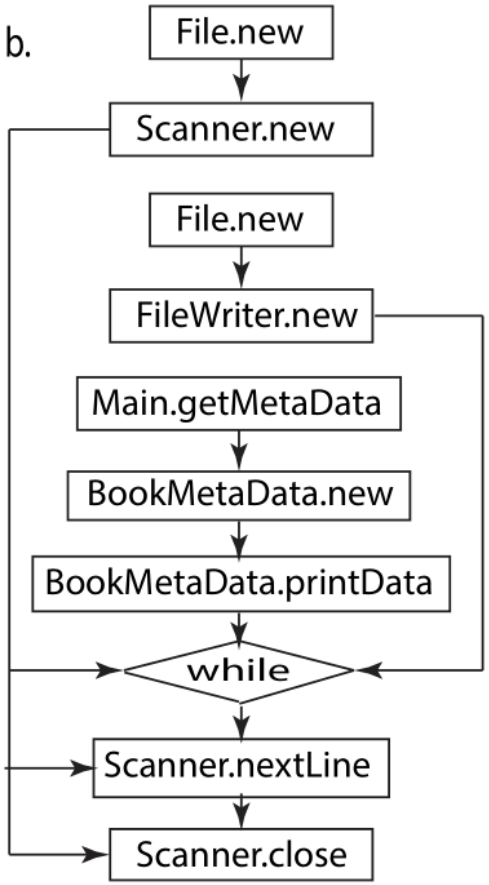
- 相关论文：
  - Nguyen, Anh Tuan, and Tien N. Nguyen. "Graph-based statistical language model for code." ICSE 2015.
  - Nguyen, Tung Thanh, Hoan Anh Nguyen, Nam H. Pham, Jafar M. Al-Kofahi, and Tien N. Nguyen. "Graph-based mining of multiple object usage patterns." FSE 2009

API使用图Groum

a.

```
1  File bookFile = new File("books.txt");
2  Scanner bookSc = new Scanner(bookFile);
3
4  File authorFile = new File("authors.txt");
5  FileWriter authorFW= new FileWriter
6                                  (authorFile);
7  BookMetaData metaData =
8           getMetaData("bookMetaData.txt");
9  metaData.printData();
10
11 while ( ) {
12     bookSc.nextLine();
13 }
14
15 bookSc.close();
```

b.



- 结点是API调用或者if, while
- 边为数据依赖或者控制依赖

图上的API调用补全问题

- 在图上加上一个新节点，问该结点为哪个API调用的时候概率最大
  - 即寻找G’，最大化P(G’|G)，其中G’比G多一个API调用节点
- 经典机器学习问题：样本不足以支持取样
- 解决方案：
  - 朴素贝叶斯+n-gram

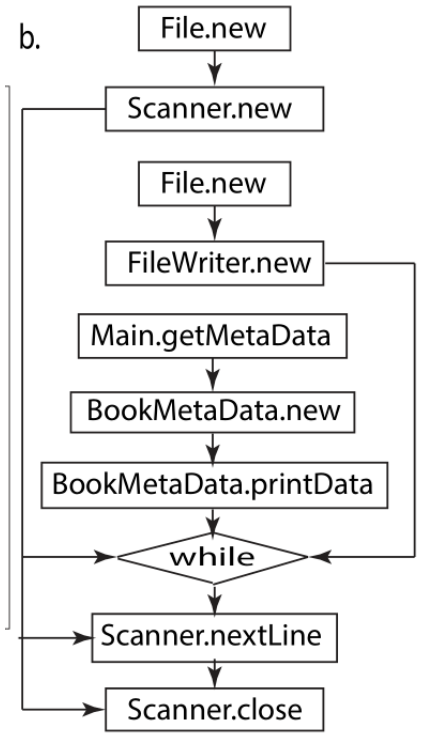
基于图的API调用补全

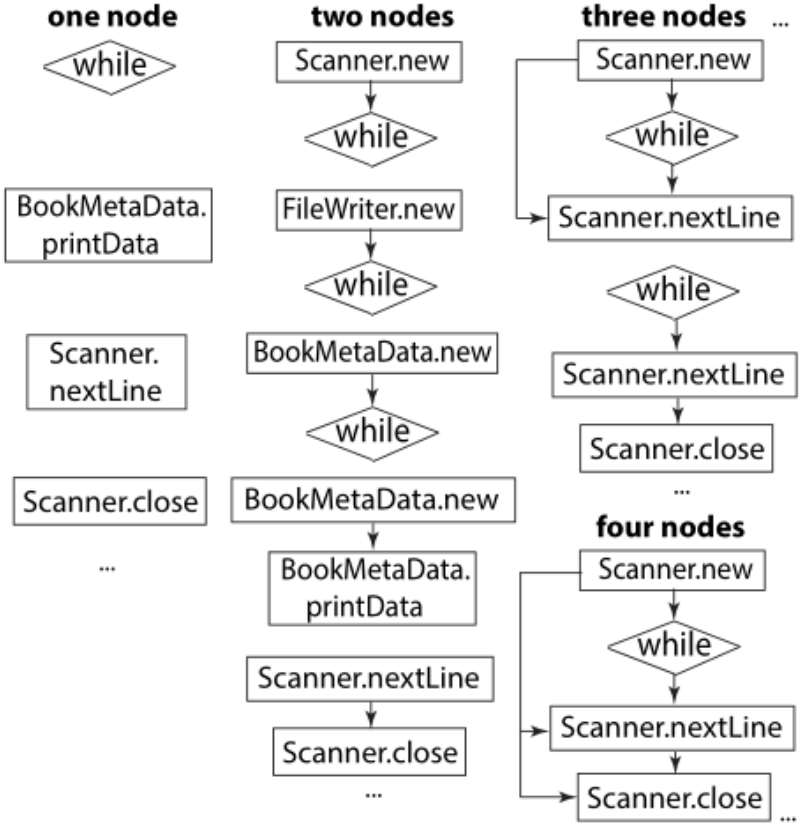
a.

```
1  File bookFile = new File("books.txt");
2  Scanner bookSc = new Scanner(bookFile);
3
4  File authorFile = new File("authors.txt");
5  FileWriter authorFW= new FileWriter
6                                  (authorFile);
7  BookMetaData metaData =
8           getMetaData("bookMetaData.txt");
9  metaData.printData();
10
11 while ( ) {
12     bookSc.nextLine();
13 }
14
15 bookSc.close();
```

1. 寻找m个距离补全点最近的节点，为上下文节点
   - 距离用文本上的字符数衡量
   - 令m=4，则包括
     - BookMetaData.printData
     - While
     - Scanner.nextLine
     - Scanner.close

基于图的API调用补全





2. 找到图上所有节点数<=n的子图（仅考虑包含节点间所有边的子图），选出仅包含上下文节点的子图，形成集合G

基于图的API调用补全

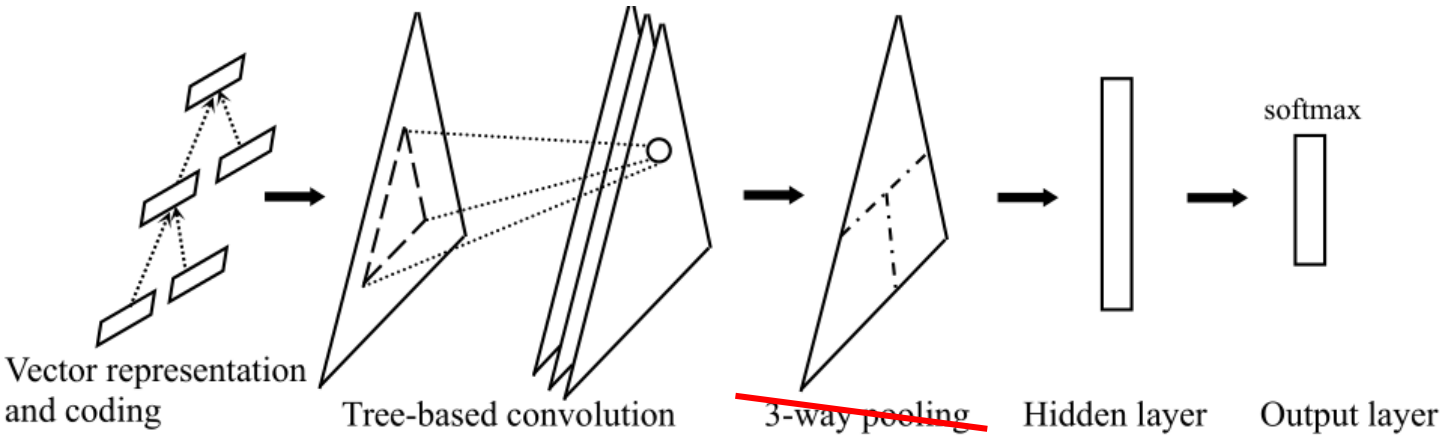
- 对于集合G中的每一个子图和每一个API，形成新的子图h
- 用朴素贝叶斯求概率
  - $P(h|G) = \frac{P(G|h)P(h)}{P(G)}$
  - $P(G|h) \approx P(g_1|h)P(g_2|h) \dots P(g_n|h)$
- 其中
  - $P(g_i|h) \approx \frac{\text{同时包含}g_i\text{和}h\text{的方法的数量}}{\text{包含}h\text{的方法的数量}}$
  - $P(h) \approx \frac{\text{包含}h\text{的方法的数量}}{\text{所有方法的数量}}$

树卷积神经网络

Lili Mou, Ge Li, Lu Zhang, Tao Wang, Zhi Jin.
"Convolutional neural networks over tree structures for programming language processing." AAAI 2016.

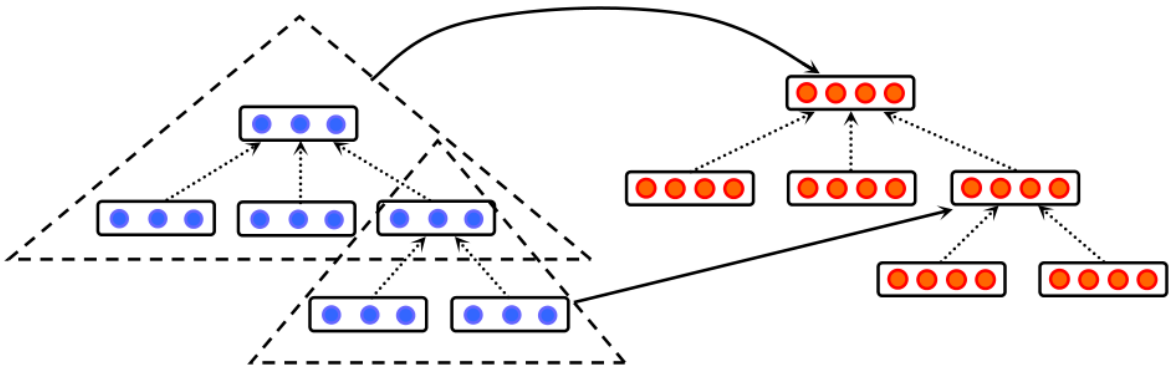
树卷积神经网络

- 在AST树上做卷积



1-way pooling

树卷积



$$\boldsymbol{y} = \tanh\left(\sum_{i=1}^{n} W_{\text{conv},i} \cdot \boldsymbol{x}_i + \boldsymbol{b}_{\text{conv}}\right)$$

- 主要问题：给定深度为d的窗口，n的大小不固定，怎么办？
- 解决方法：通过对固定参数线性加权来产生任意长度的参数列表
  - $\mathrm{W}_{conv,i} = \eta_i^t W^t + \eta_i^l W^l + \eta_i^r W^r$
  - $\eta_i^t = \frac{d_i - 1}{d - 1}$，其中$d_i$为节点i的深度
  - $\eta_i^r = (1 - \eta_i^t)\frac{p_i - 1}{n - 1}$，其中$n$为节点i所有兄弟（含节点i自己）的数量，$p_i$为节点i的在兄弟中位置
  - $\eta_i^l = (1 - \eta_i^t)\,(1 - \eta_i^r)$

树卷积神经网络效果

- 104道POJ编程题目，每道题500个答案
- 让神经网络自动分类每个答案应该属于哪个题目
- 准确率：94.0%