

Beijing Forest Studio
北京理工大学信息系统及安全对抗实验中心



基于神经网络的源代码表示方法

Graph neural networks for source code representation

陈传涛 硕士研究生

2020年07月19日

- 背景简介
- 基本概念
- 算法原理
- 优劣分析
- 应用总结
- 参考文献

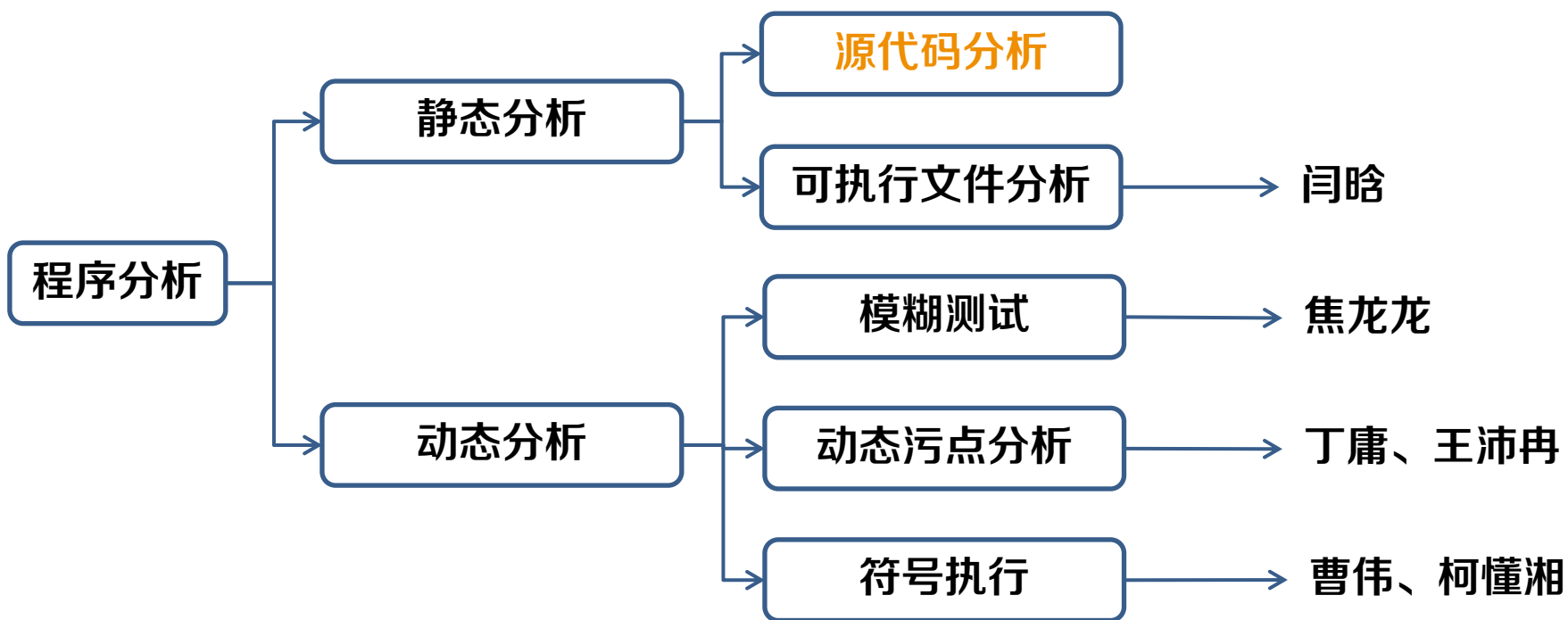
- 预期收获
 - 1. 了解源代码表示的基本概念和常见方法
 - 2. 理解基于神经网络的源代码表示算法的基本思想
 - 3. 了解源代码表示的实际应用



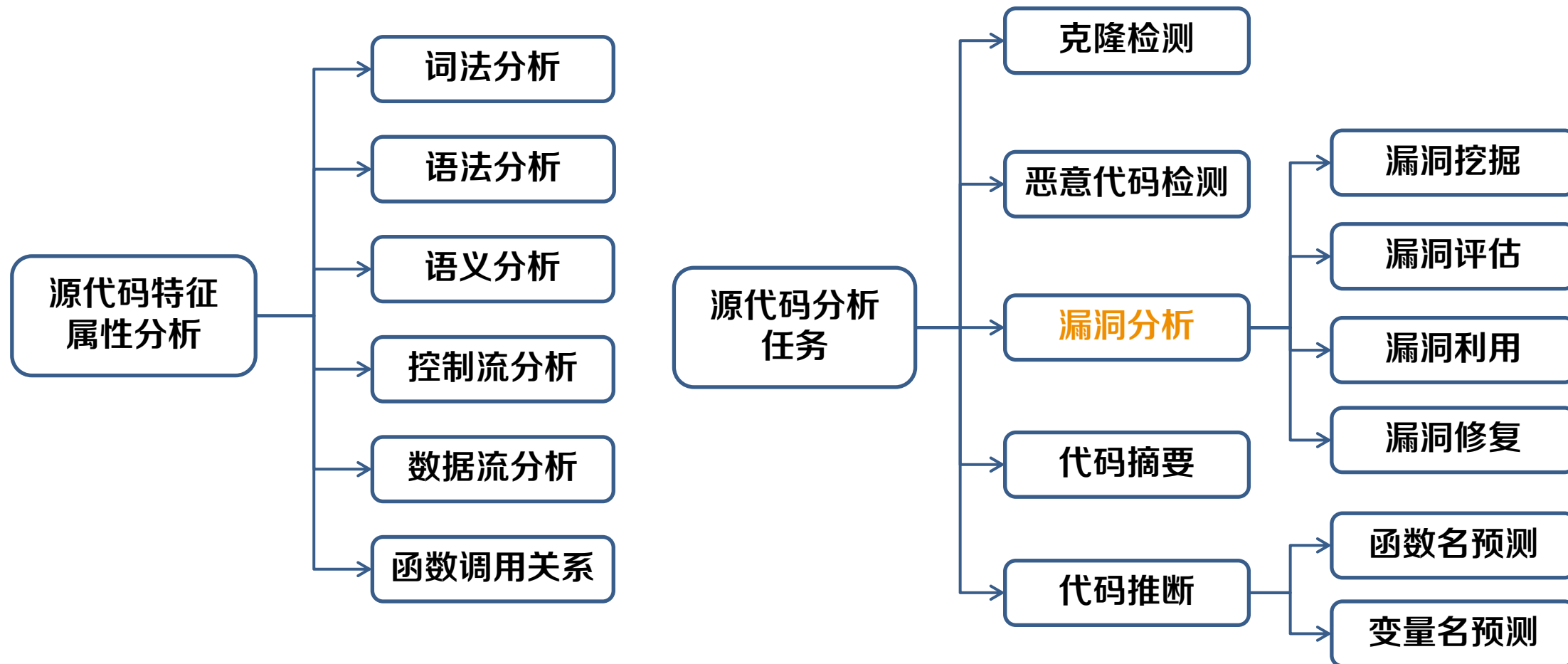
背景简介

- 程序分析

- 程序分析指的是对计算机程序进行人工或自动化的分析，以确认或发现其特性，如性能、正确性、安全性等。



- 源代码分析

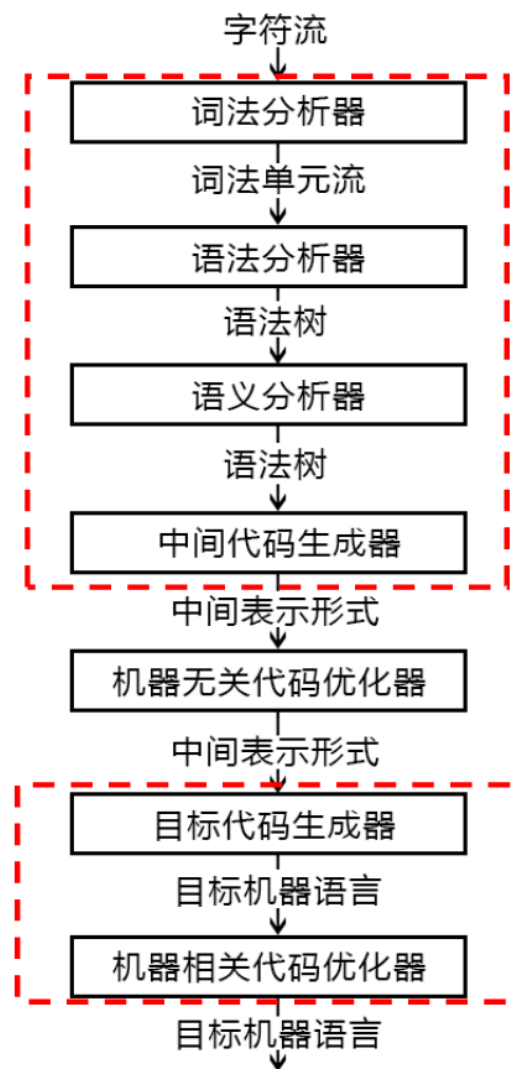


- 编译过程的前端分析部分:

- 词法分析 (lexical analysis) : 顺序扫描字符流, 识别令牌 (tokens)。
- 语法分析 (syntax analysis) : 将令牌序列组合成语法短语, 生成语法树。
- 语义分析 (semantic analysis) : 上下文相关性审查。

分析部分/
前端(front end):
与源语言相关

综合部分/
后端(back end):
与目标语言相关



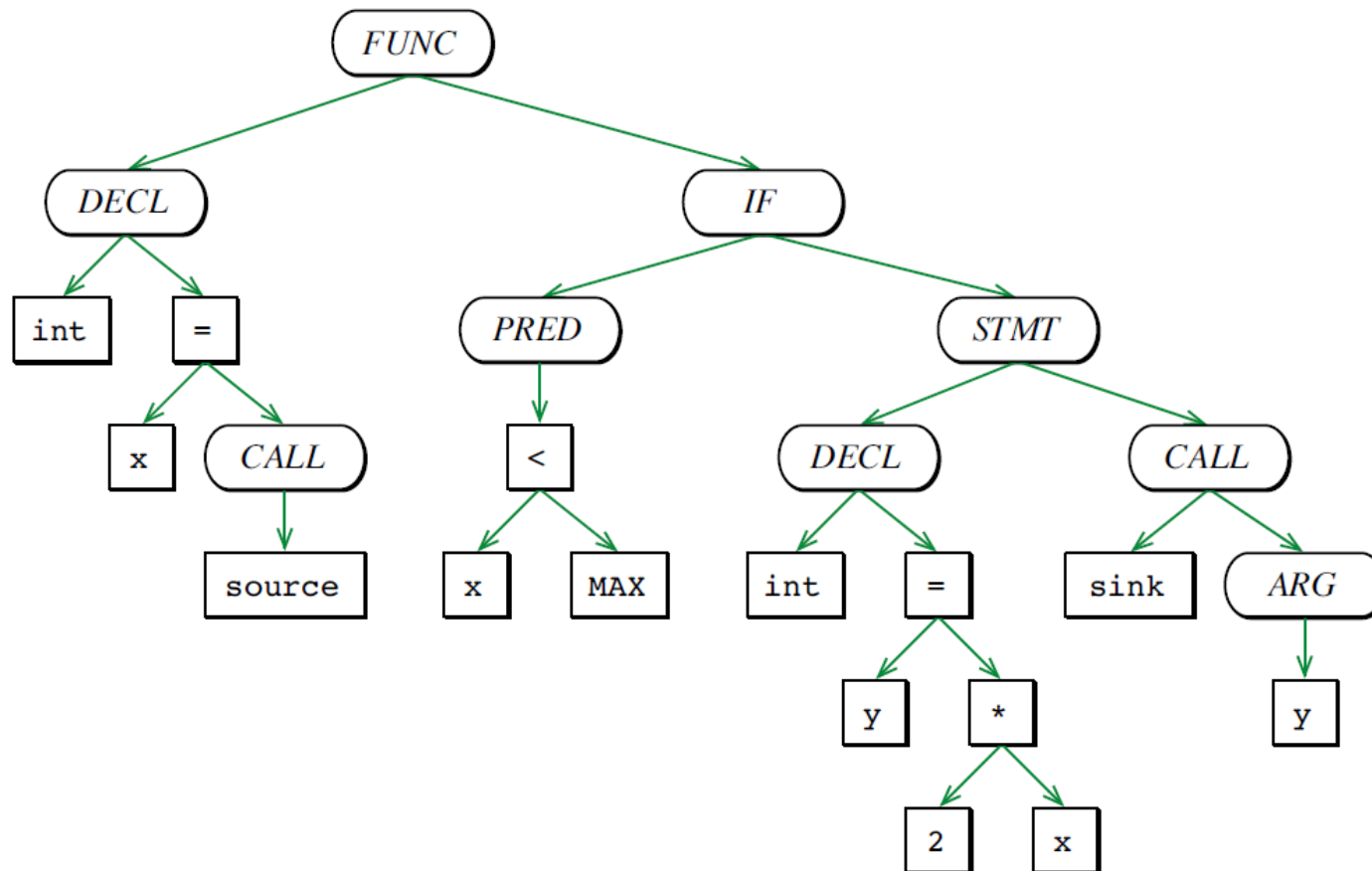


基本概念

- 抽象语法树 (Abstract syntax tree)

```
void foo()  
{  
  int x = source();  
  if (x < MAX)  
  {  
    int y = 2 * x;  
    sink(y);  
  }  
}
```

Source code

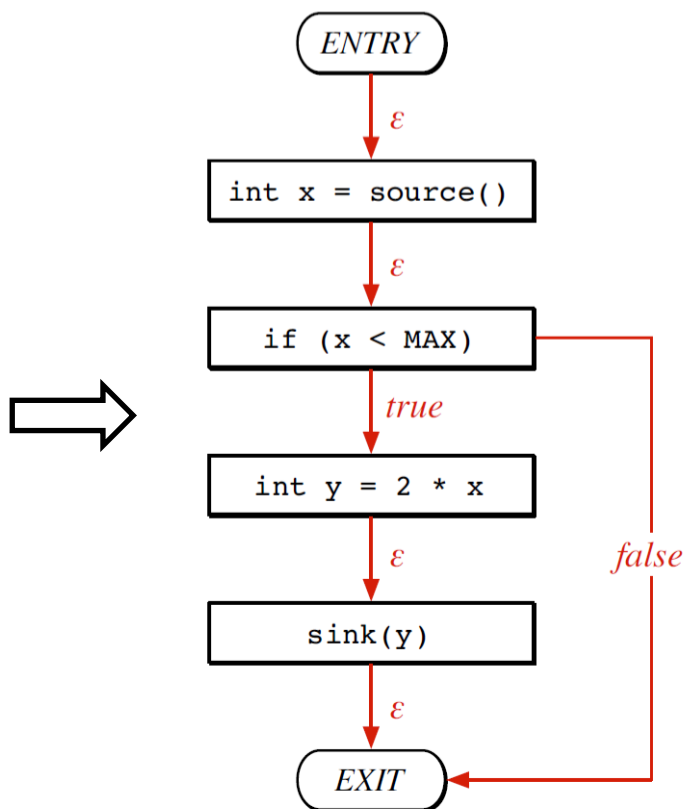


Abstract syntax tree (AST)

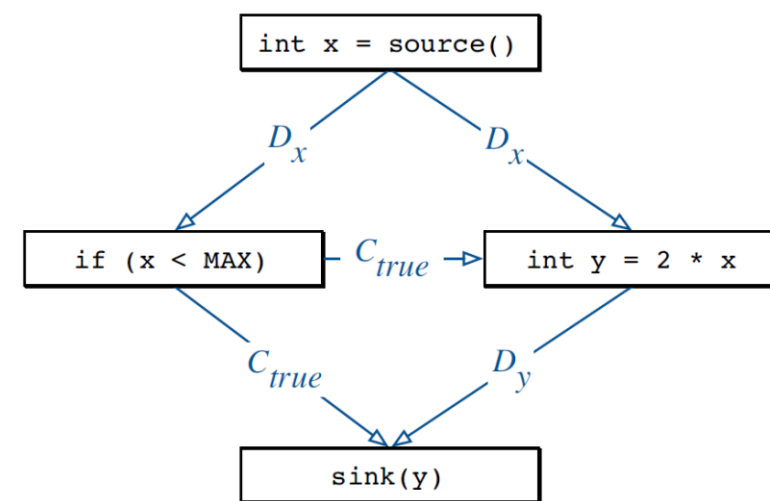
- 控制流图 (CFG) 与程序依赖图 (PDG, 也叫数据依赖图)

```
void foo()  
{  
    int x = source();  
    if (x < MAX)  
    {  
        int y = 2 * x;  
        sink(y);  
    }  
}
```

Source code



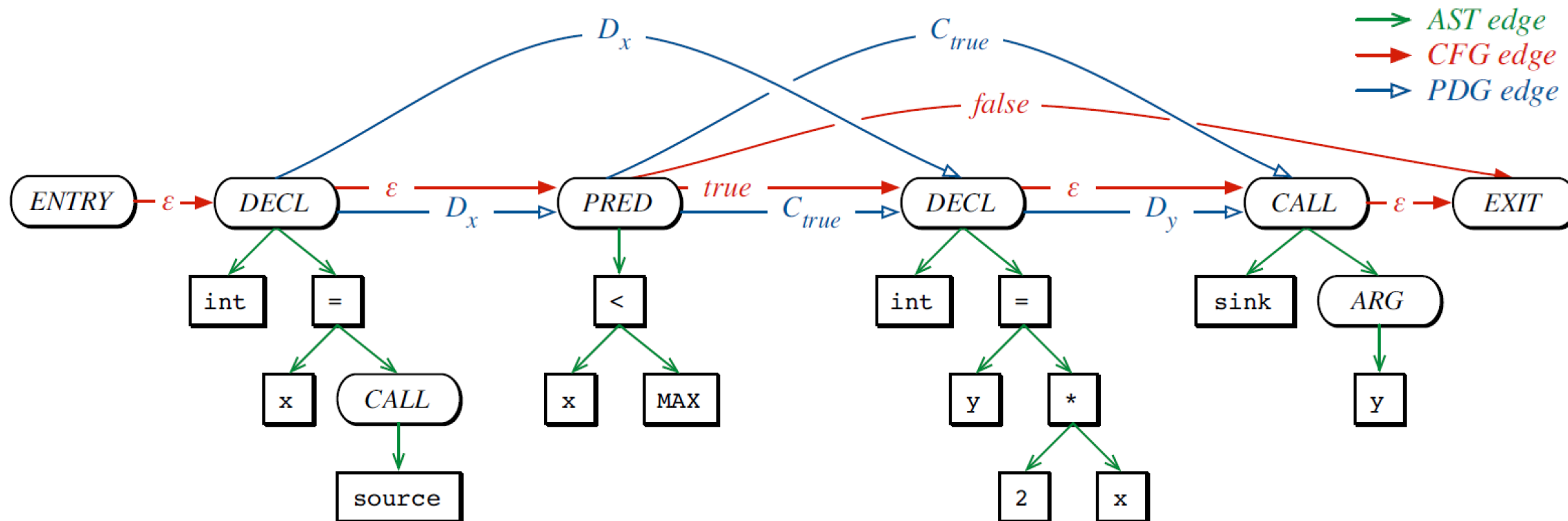
Control flow graph (CFG)



Program dependence graph (PDG)

- 代码属性图 (code property graph, CPG)

- 代码属性图=抽象语法树+控制流图+数据依赖图



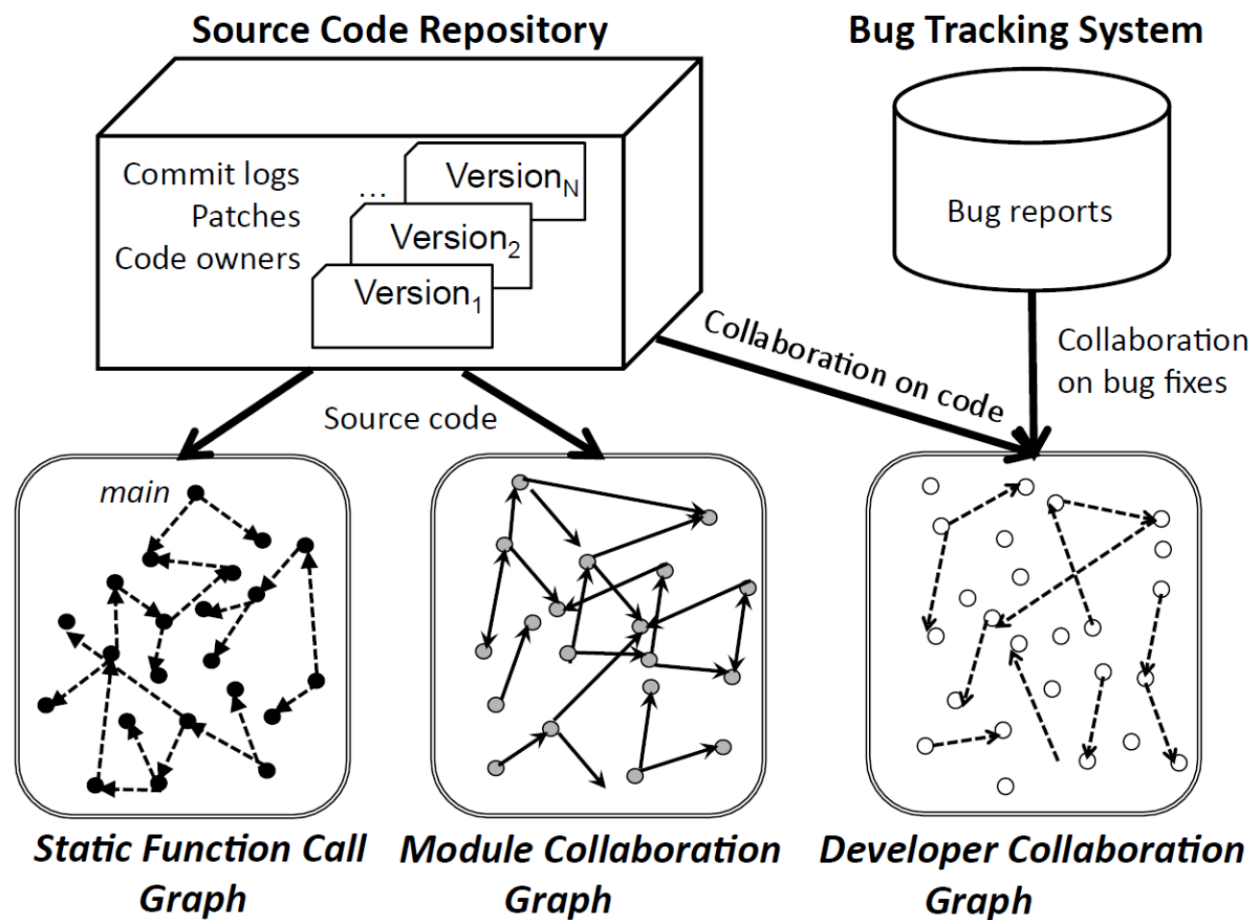
Code property graph

- 调用关系图

- 静态函数调用图：用来捕捉函数调用关系。
- 模块协作图：用来捕捉模块之间的通信特征。

- 调用关系图的意义

- 目标函数或模块对于程序功能完整的重要性影响



- 代码表示（code representation）：
 - 一种**适当的**程序中间表示方法，能够将源代码字符表示成数值向量。也称程序表示（program representation）
- 如何找到适当的中间表示？
 - 源代码的原始特征有哪些？
 - 词法、语法和语义信息；
 - 抽象语法树、数据依赖图和控制流图；
 - 函数或模块间的调用关系。
 - 应用问题是什么？
 - 代码克隆检测，漏洞挖掘，程序安全性评估等等。



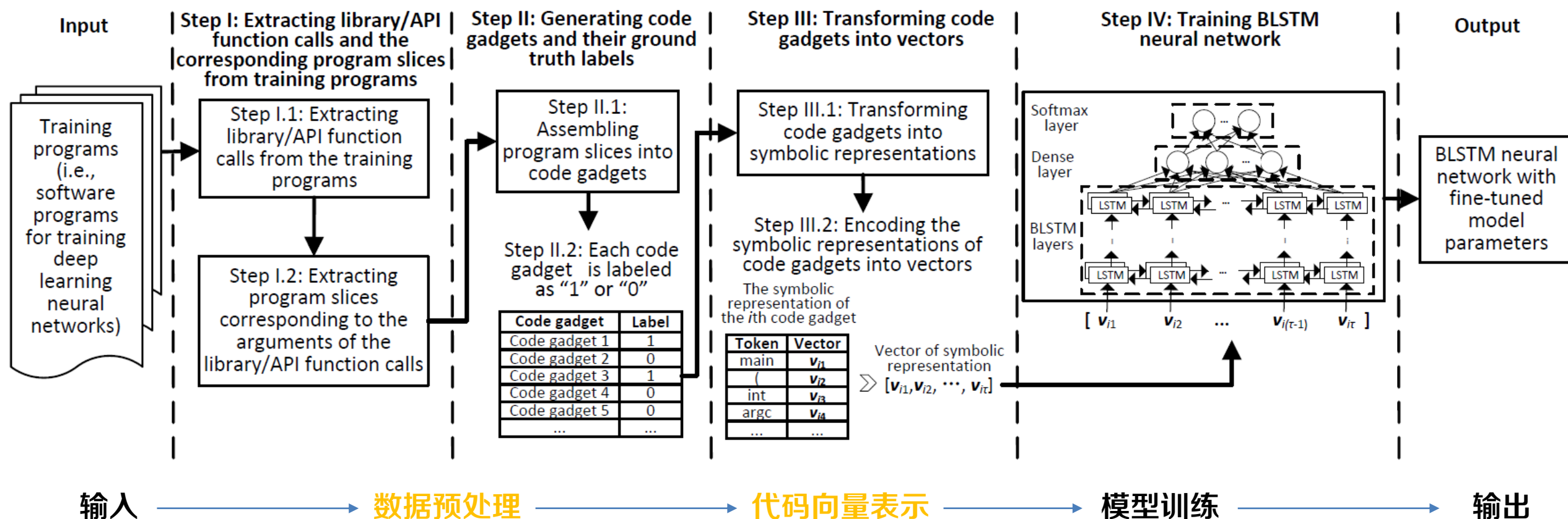
算法原理

T	提取程序源代码中的语法语义特征，生成代码表示向量
I	程序源代码
P	1. 代码预处理； 2. 特征预处理： 词法、语法和语义分析； 控制流、数据依赖分析； 3. 特征学习。
O	代码的向量表示

P	如何对程序源代码进行适当的向量表示
C	目标程序开源，仅针对特定编程语言
D	如何对程序源代码进行特征预处理，使得神经网络能够更好地学习和捕捉代码特征
L	CCF A类期刊和会议

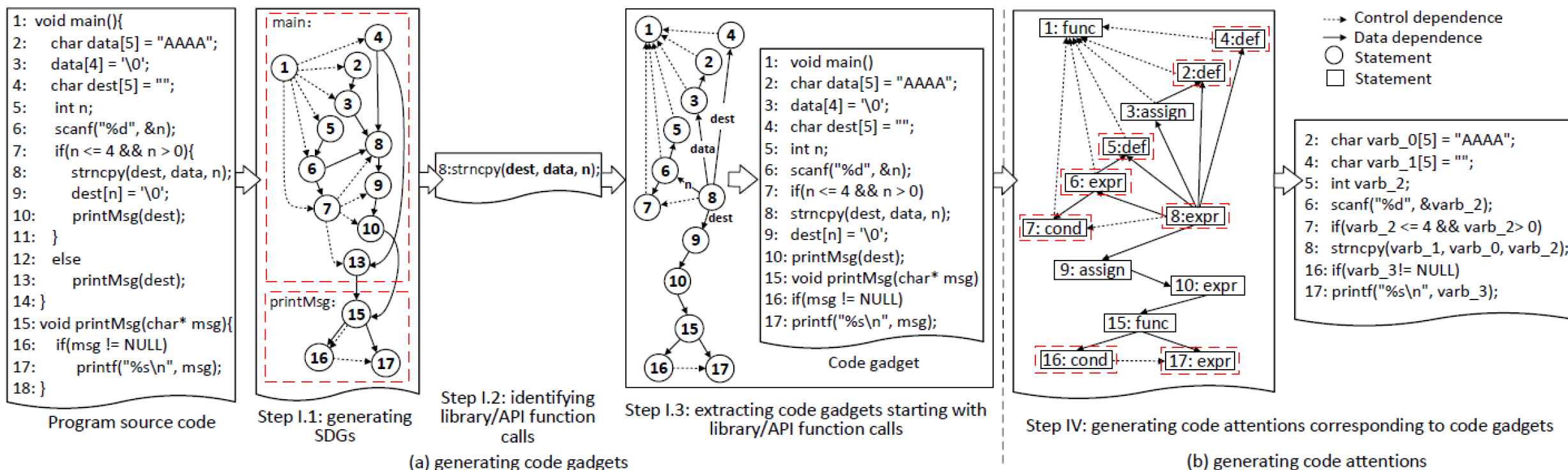
• 方法1, 利用LSTM算法提取源代码字符序列中的词法和语义信息

– 原理框架图:



• 方法1的代码预处理模块:

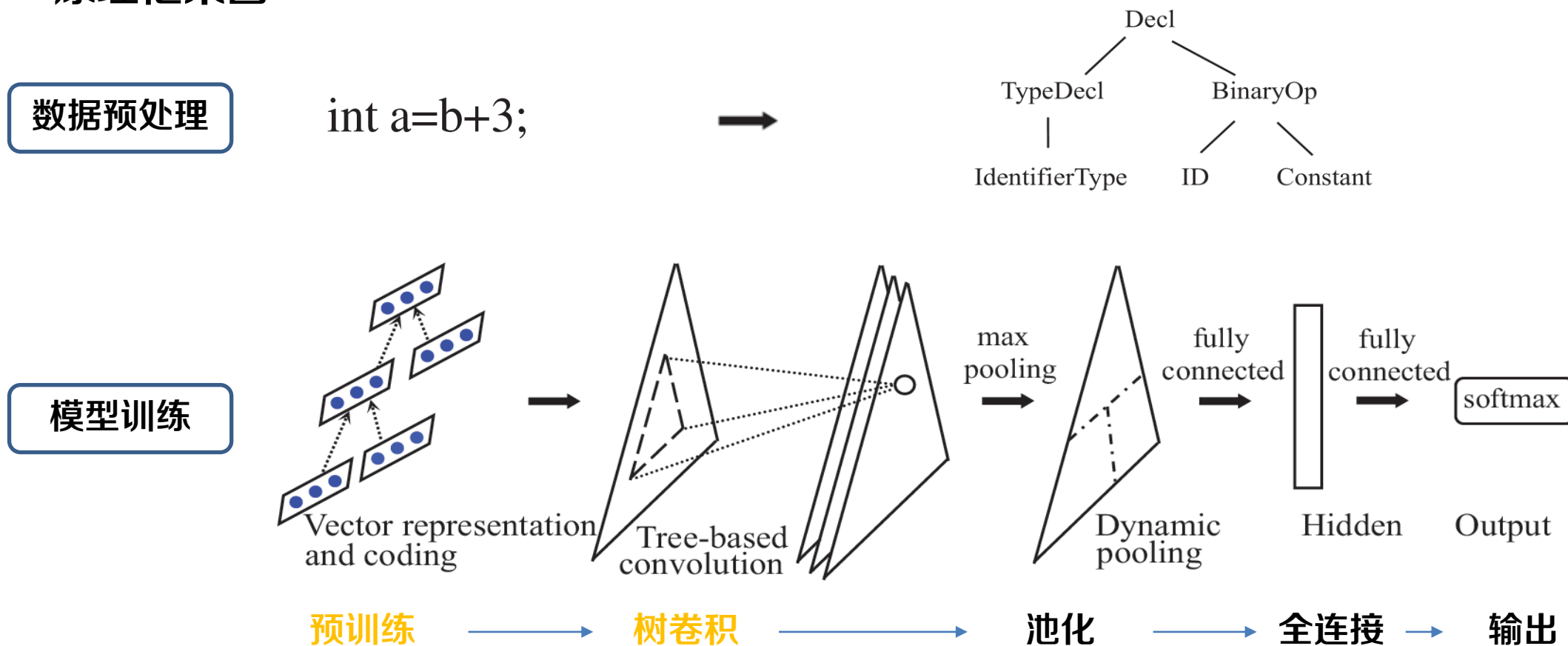
– 通过控制流和数据依赖分析, 提取敏感函数上下文相关的语句



- 方法1的基本思想：
 - 将程序源代码视为文本序列，利用自然语言处理中的文本表示方法生成代码的向量表示。
- 优点：
 - 利用深度学习算法自动提取代码中的词法和语义特征，提高了源代码分析效率。
- 缺点：
 - 难以捕捉代码中的语法结构信息；

- 方法2，利用基于树结构的卷积网络捕捉源代码的语法结构信息

- 原理框架图：



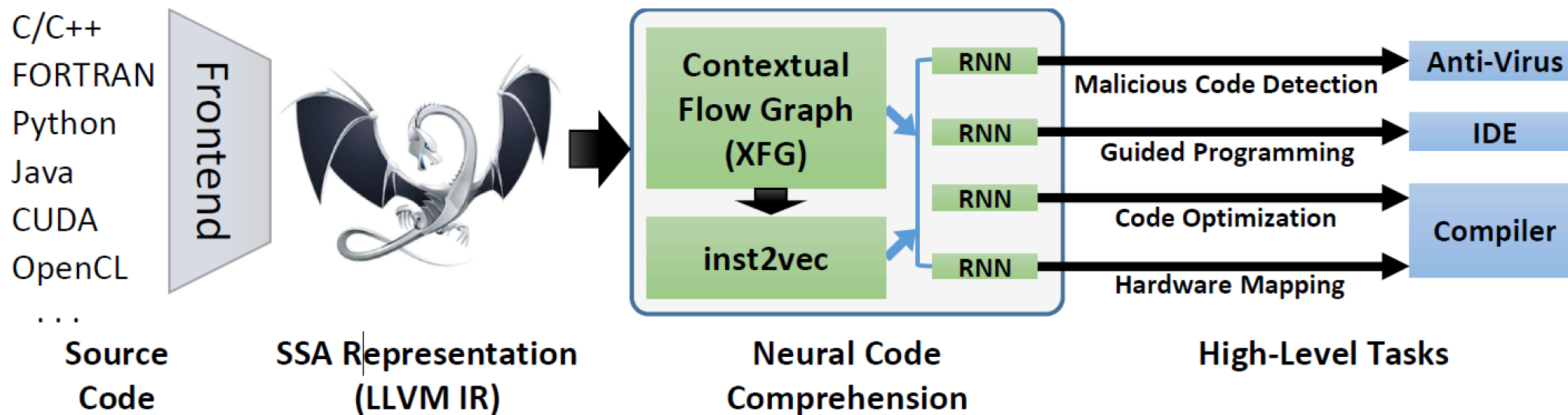
- 方法2的基本思想：
 - 将源代码的抽象语法树作为输入，利用基于树结构的卷积神经网络提取语法结构特征。
- 优点：
 - 能够有效提取源代码的语法结构特征。
- 缺点：
 - 只分析了源代码的抽象语法树，无法捕捉源代码中的数据依赖关系。

- 方法3，利用 LLVM 中间表示实现跨语言的代码表示

- 工具介绍：

- LLVM：提供了一套中立的中间代码和编译基础设施的编译器架构。
 - 中间表示（Intermediate Representation，IR），不同的前端语言最终都转换成同一种的IR。

- 原理框架图：



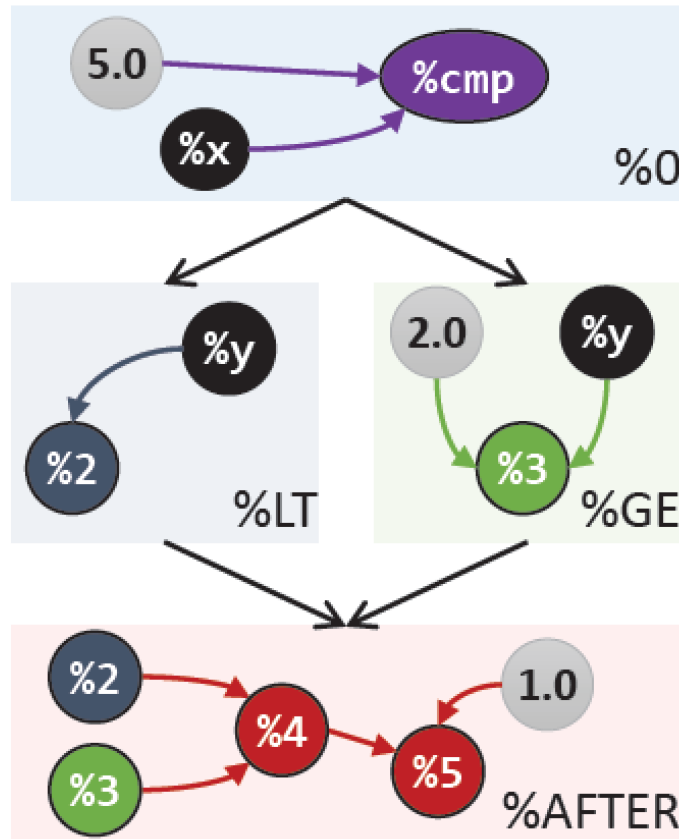
• 方法3的代码预处理：上下文流图（Contextual Flow Graph）

```
double thres = 5.0;
if (x < thres)
    x = y * y;
else
    x = 2.0 * y;
x += 1.0;
```

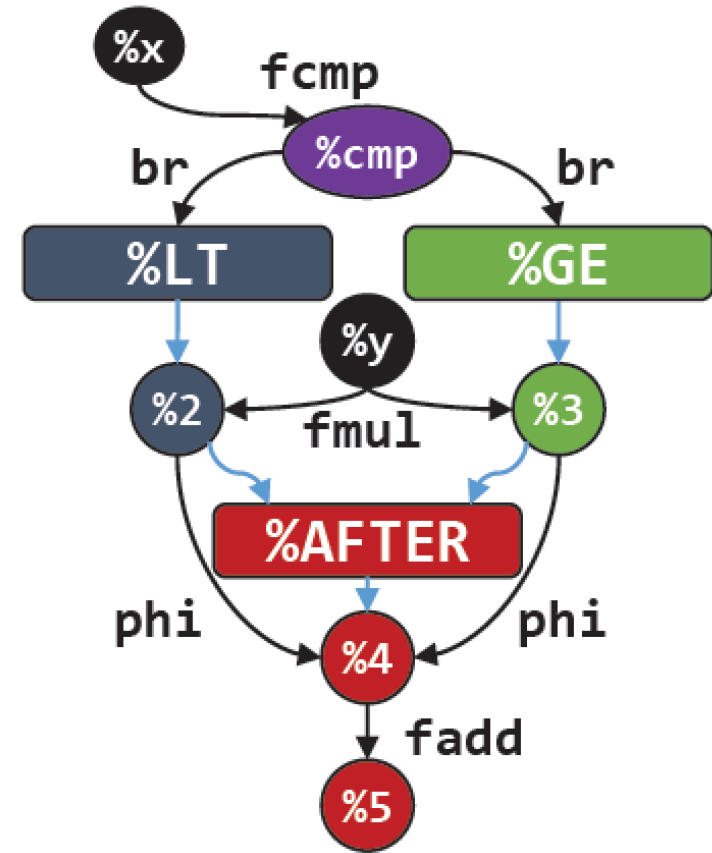
(a) Source code

```
%cmp = fcmp olt double %x, 5.0
br i1 %cmp, label %LT, label %GE
LT:
    %2 = fmul double %y, %y
GE:
    %3 = fmul double 2.0, %y
AFTER:
    %4 = phi double [%2,%LT], [%3,%GE]
    %5 = fadd double %4, 1.0
```

LLVM IR



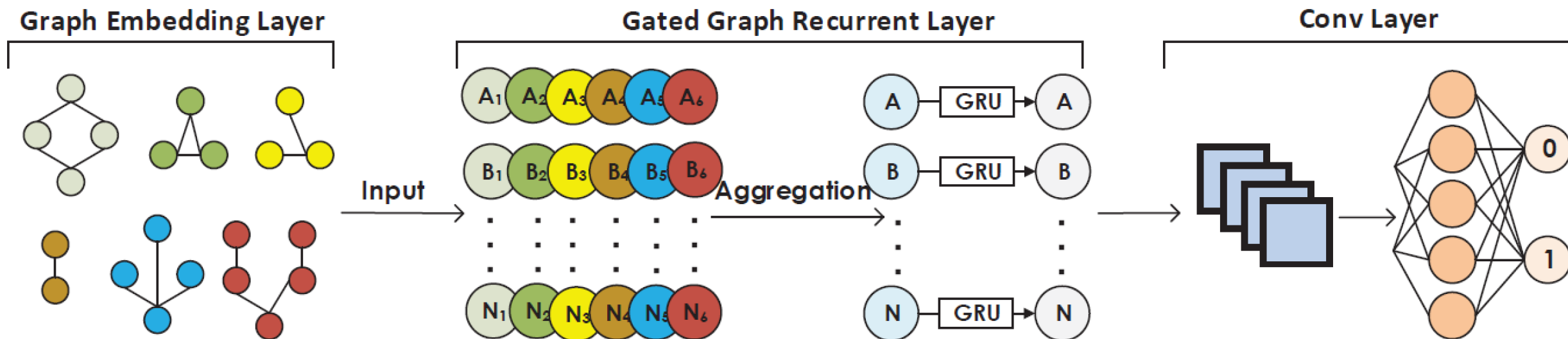
Dataflow basic blocks



Contextual Flow Graph

- 方法3的基本思想：
 - 使用LLVM IR 作为多语言代码的统一中间表示，定义了基于控制流和数据依赖的上下文流（contextual flow）特征。
- 优点：
 - 实现了跨语言的代码表示，能够用于跨语言克隆检测、代码分类等任务中。
- 缺点：
 - 需要源码完整，且LLVM运行效率低。

- 方法4，利用图神经网络算法挖掘源代码的图表示中的特征信息
 - 原理框架图：



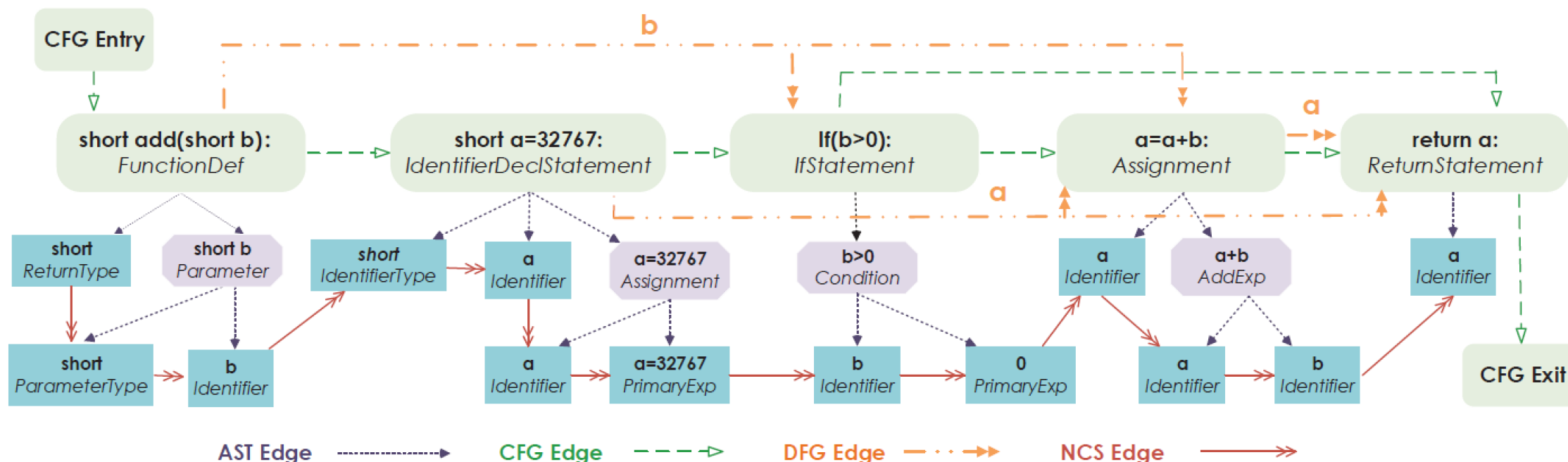
• 方法4的代码预处理

- 代码联合图：在代码属性图的基础上，添加自然语言序列（Natural Code Sequence, NCS）

$$\bullet = \text{CPG} + \text{NCS} = \text{AST} + \text{CFG} + \text{EDG} + \text{NCS}$$

```

1 short add (short b) {
2     short a = 32767;
3     if (b > 0) {
4         a = a + b;
5     }
6     return a;
7 }
    
```



- 方法4的基本思想：
 - 将源代码的图表示作为输入，利用图神经网络算法提取复合语义信息。
- 优点：
 - 构建源代码的多类特征图，能够学习丰富的代码语义表示。
- 缺点：
 - 依赖于 Joern 代码解析工具生成代码属性图，只能处理 C/C++ 代码。

- 横向对比
 - 论文1（代码序列+LSTM）：无法捕捉源代码语法结构信息；
 - 论文2（AST+树卷积网络）：无法提取控制流和数据依赖关系；
 - 论文3（LLVM IR +RNN）：可以实验跨语言代码表示，但需要源码完整。
 - 论文4（CPG+图神经网络）：构建源代码的多类特征图，能够学习丰富的代码语义信息。
- 纵向对比
 - 基于图神经网络的程序表示方法能够接受图结构数据作为输入，挖掘源代码的图表示中的潜在特征信息。

- 算法的应用领域
 - 克隆检测或相似性检测。如：代码查重系统。
 - 恶意代码检测。如：Android恶意软件检测，PC端恶意程序检测，入侵检测。
 - 漏洞分析。如：源代码漏洞检测，漏洞严重性评估。
- 未来的发展
 - 基于图神经网络和源代码表示方法

- 经典文献

- [1]. Yamaguchi F, Golde N, Arp D, et al. Modeling and Discovering Vulnerabilities with Code Property Graphs[C]. 2014 IEEE Symposium on Security and Privacy, 2014. 【提出代码属性图CPG】
- [2]. P. B, M. I, I. N, et al. Graph-based analysis and prediction for software evolution[C]. 2012 34th International Conference on Software Engineering (ICSE), 2012. 【提出代码调用图】

- 高水平文献

- [1]. Zou D, Wang S, Xu S, et al. μ VulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection[J]. IEEE Transactions on Dependable and Secure Computing, 2019.
- [2]. Mou, L., et al., Convolutional Neural Networks over Tree Structures for Programming Language Processing[C]. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona, 2016.
- [3]. Ben-Nun T, Jakobovits A S, Hoefler T. Neural code comprehension: A learnable representation of code semantics[C]. Advances in Neural Information Processing Systems, 2018.
- [4]. Zhou Y, Liu S, Siow J, et al. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks[C]. Advances in Neural Information Processing Systems, 2019.



谢谢！

大成若缺，其用不弊。大盈
若冲，其用不穷。大直若屈。
大巧若拙。大辩若讷。静胜
躁，寒胜热。清静为天下正。

