**目录：**

[**1** **编译器和静态分析的关系** 1](#_Toc52858842)

[2 AST vs IR 1](#_Toc52858843)

[3 IR:3-地址代码（3AC） 2](#_Toc52858844)

[4 实际静态分析器的3AC—Soot（Java） 3](#_Toc52858845)

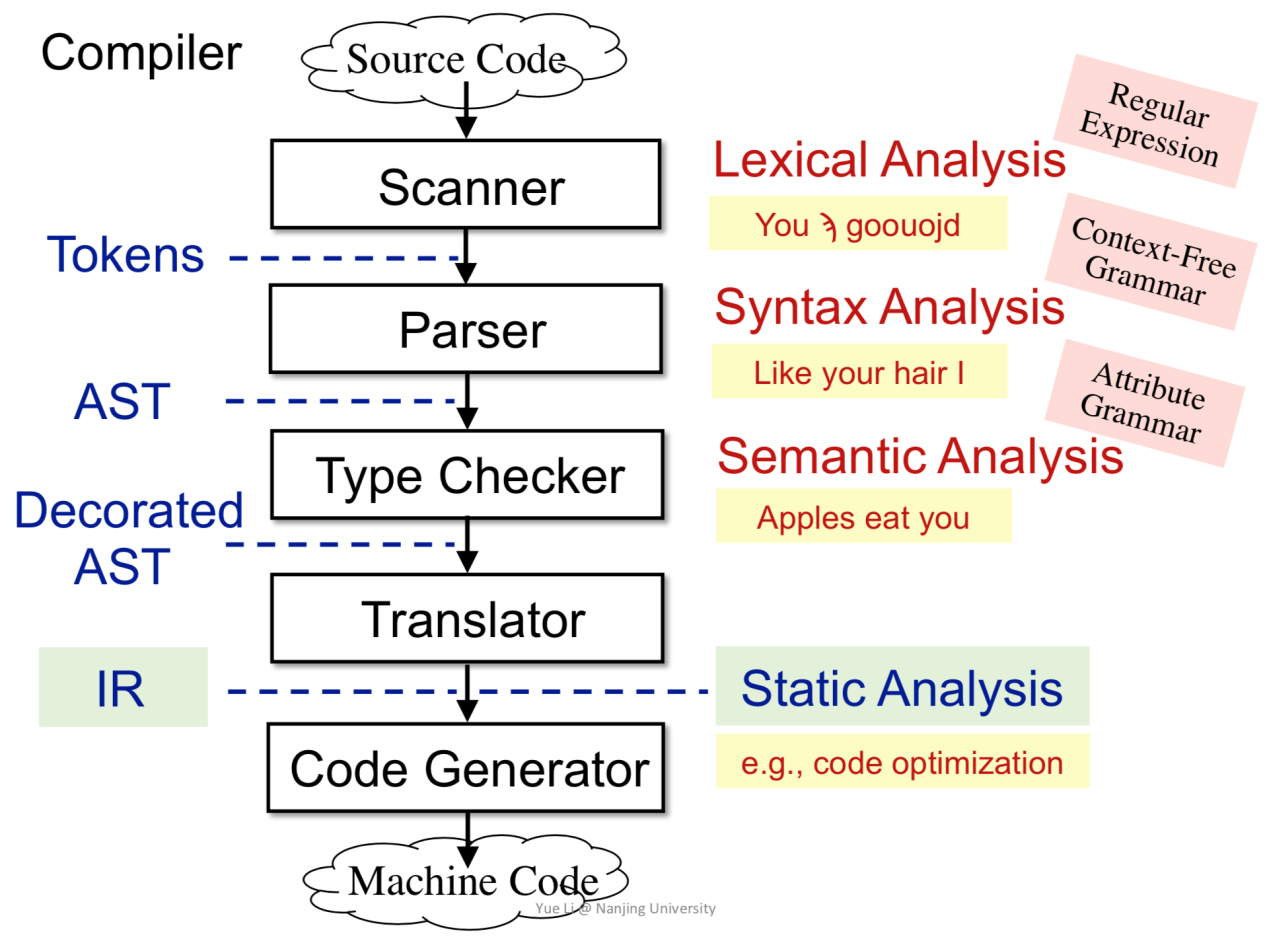
[5 SSA-静态单赋值 3](#_Toc52858846)

[6 基本块（BB） 4](#_Toc52858847)

[7 控制流图（CFG） 5](#_Toc52858848)

1. **编译器和静态分析的关系**

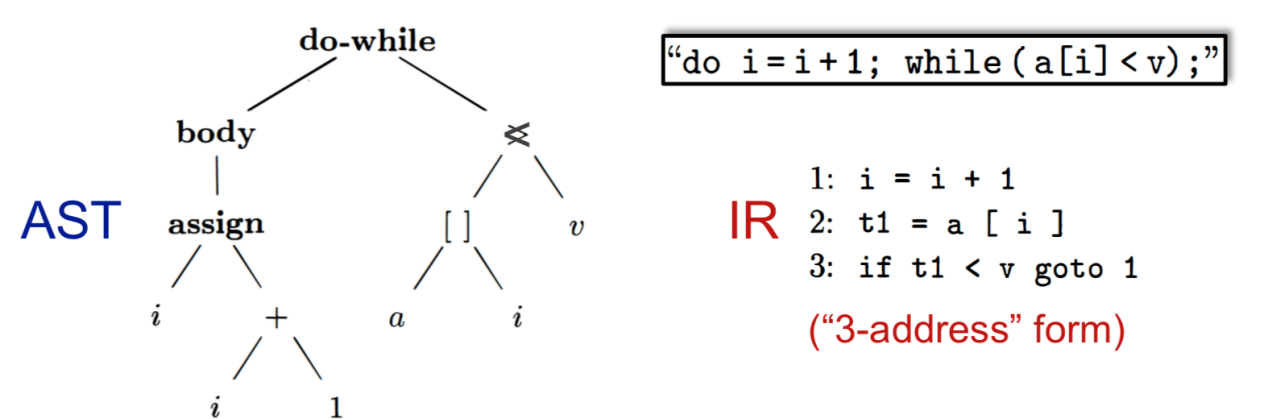
源码->（Scanner - 词法Lexical分析-Regular Expression）->（Parser- 语法Syntax分析-Context-Free Grammar）， 生成AST ->（Type Checker - 语义Semantic分析 - Attribute Grammar），生成 Decorated AST -> Translator，生成IR，进行静态分析 -> Code Generator



## AST vs IR

**AST** ：高级，更接近于语法结构，依赖于语言种类，适用于快速类型检查，缺少控制流信息

**IR**：低级，更接近于机器码，不依赖语言种类，压缩且简洁，包含控制流信息。是静态分析的基础



## IR:3-地址代码（3AC）

// 最多1个操作符

a+b+3 -> t1 = a+b

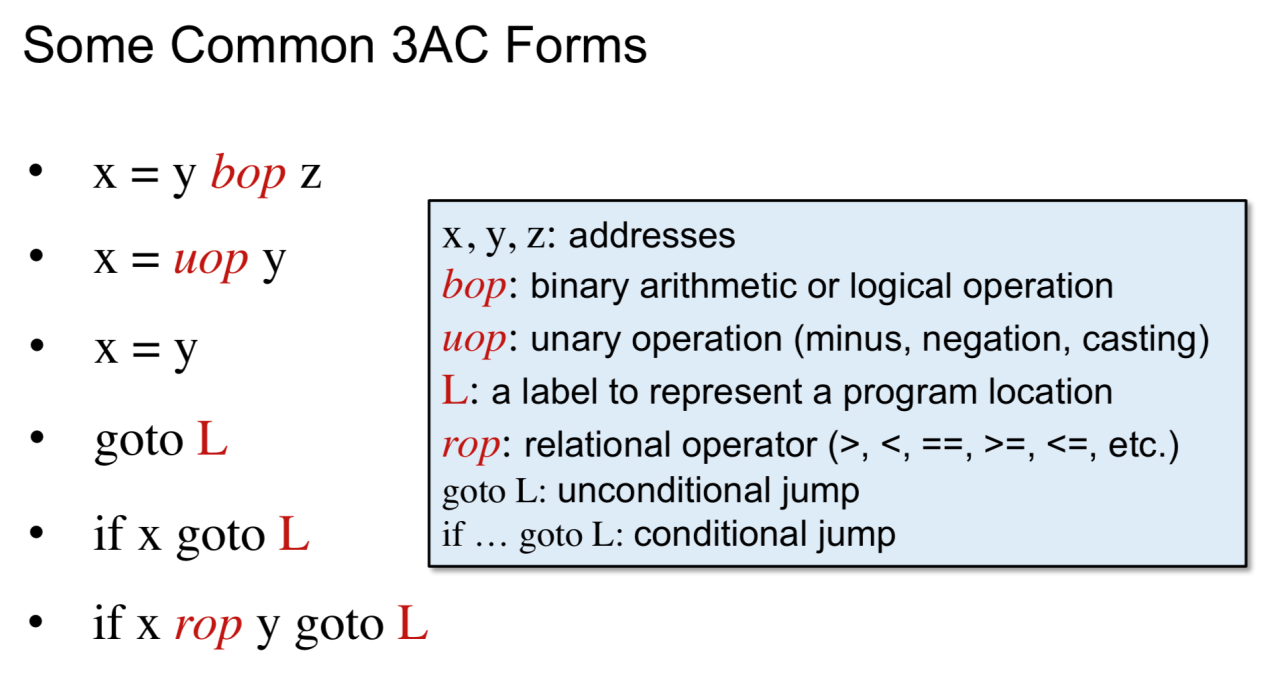
t2 = t1+3

Address：

Name:a、b

Constant: 3

编译器的临时变量：t1、t2



## 实际静态分析器的3AC—Soot（Java）

Soot-常用的Java静态分析框架

// java IR（Jimple）基本知识

invokespecial：call constructor, call superclass methods, call private methods

invokevirtual: instance methods call (virtual dispatch)

invokeinterface: cannot optimization, checking interface implementation

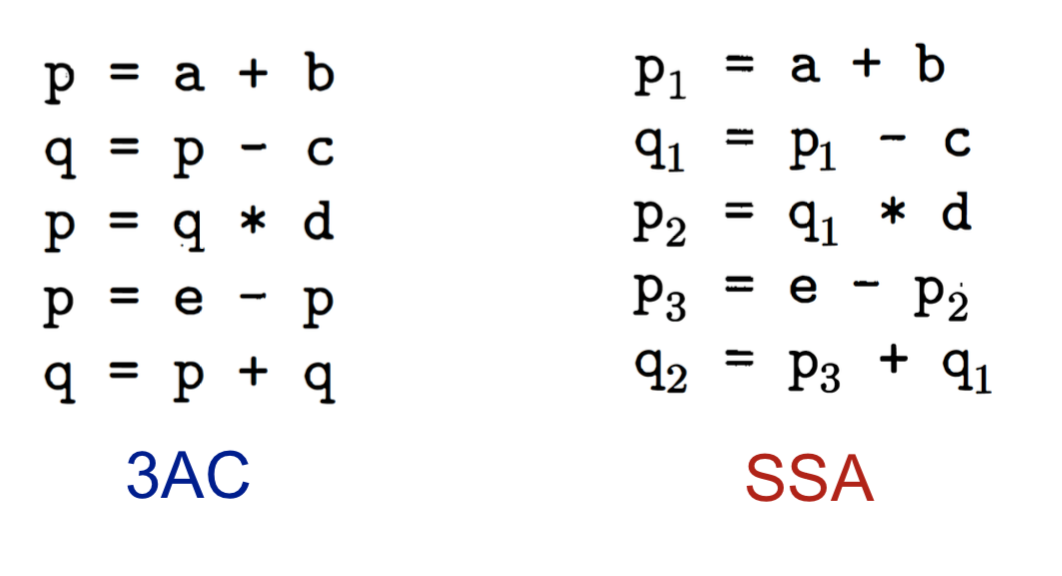
invokestation:call static methods

Java 7: invokedynamic -> Java static typing, dynamic language runs on JVM

method signature: class name, return type, method name(parameter1 type, parameter2 type)

## SSA-静态单赋值

**定义**：给每一个定义变量一个新的名字，传递到接下来的使用当中，每个变量有1个定义（赋值的目标变量）。



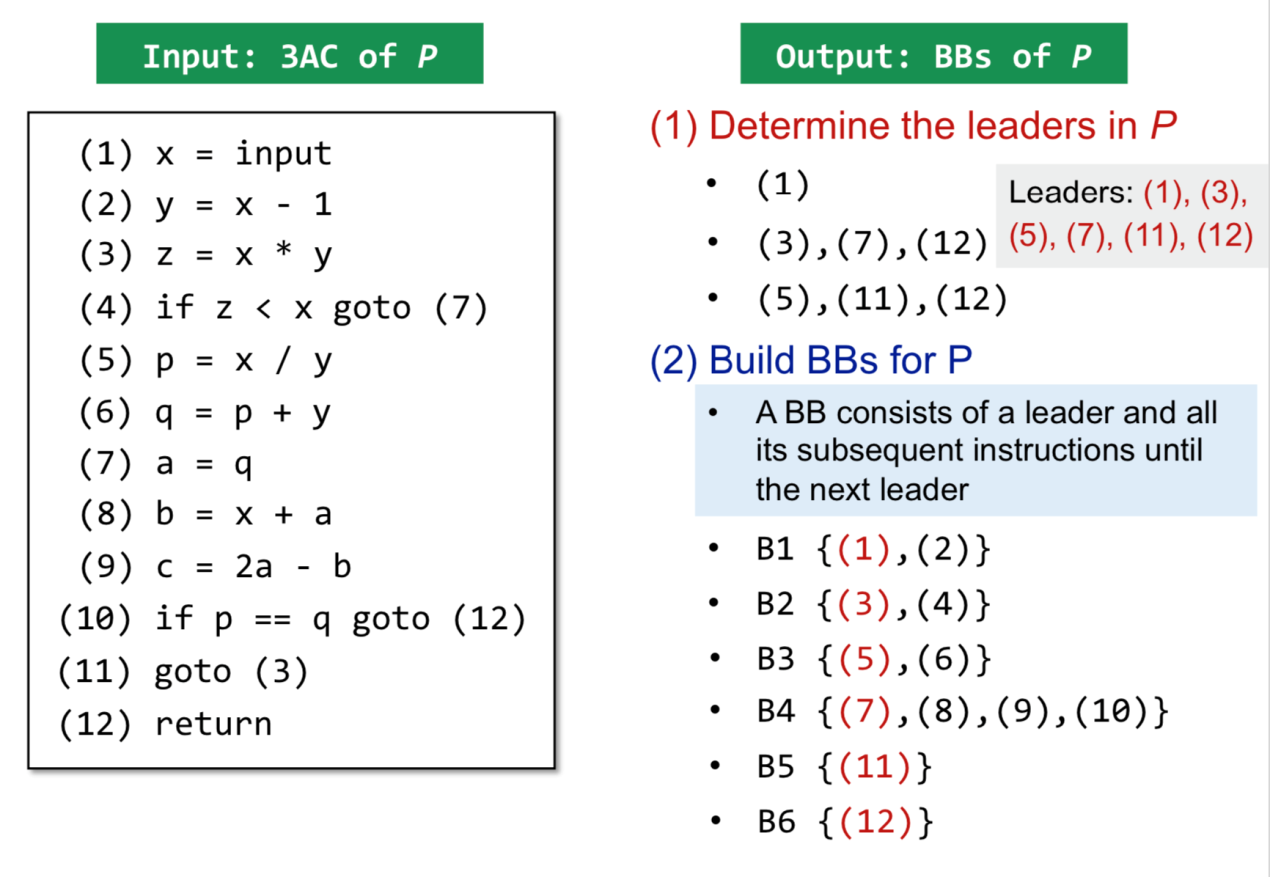
**优点**：唯一的变量名可以间接体现程序流信息，简化分析过程；清楚的Define-Use信息。

**缺点**：引入很多变量和phi-function；转换为机器码时效率变低（引入很多拷贝操作）。

## 基本块（BB）

**定义**：只有1个开头入口和1个结尾出口的最长3-地址指令序列。

**识别基本块的算法**：首先确定入口指令，第一条指令是入口；任何跳转指令的目标地址是入口；任何跟在跳转指令之后的指令是入口。然后构造基本块，任何基本块包含1个入口指令和其接下来的指令。



## 控制流图（CFG）

**控制流边**：基本块A的结尾有跳转指令跳转到基本块B；原始指令序列中，B紧跟着A，且A的结尾不是无条件跳转。

