



Maple: An Efficient Compiler

Jinning Li
Shanghai Jiao Tong University

摘要

- 共计10034行代码
- 清晰明了的编译过程：Source -> CST -> AST -> IR -> CFG -> Asm
- 简单高效的优化：基于图染色的寄存器分配、字符串优化
- 快速的内建函数库
- 通宵四天



困到昏厥

前端

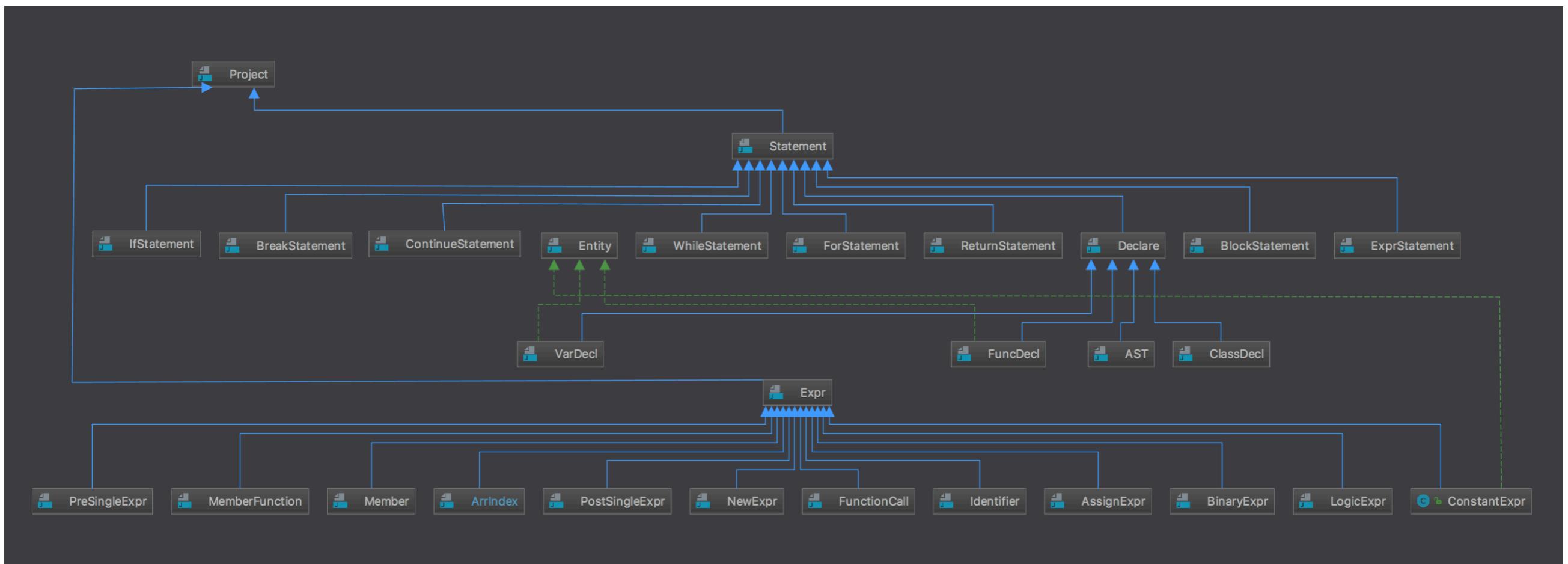


解析源代码

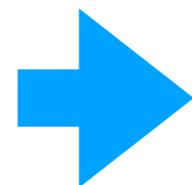
- 使用ANTLR的方法建立了一套适用于Maple语言的语法规则
- 一开始使用的是Listener方法来生成AST，但是发现这样要多扫好多遍
- 之后改用了Visitor的方法对ANTLR树遍历，来生成AST节点，只需要遍历一遍，并且把全局变量、全局函数、类、类成员、类函数都加到了符号表中

AST

- 包括Expr, Statement, Declare三个种类
- 共有27个类, 例如VarDecl, IfStatement, AssignExpr



ASTBuilder

- 继承MapleBaseVisitor
 - 规定了每个ANTLR Visitor节点建立AST的方法
 - 对最基本的语法进行简单检查
 - 初始化符号表的根节点部分。
对类成员函数进行转换
- ```
Class A{
 int a;
 int func(){
 return a;
 }
}
```
- 
- ```
Class A{  
    int a;  
}  
int func(A this){  
    return this.a;  
}
```

符号表

- 树形结构，ScopeTree
- 树的每个节点代表一个Scope
- 每个节点存储了一组从名字到实例的映射Map
- 提供了很多功能，比如查找变量名对应的实例，就从变量所在的scope查到根。还可以询问scope里的变量，所有局部变量全局变量等。

语法检查

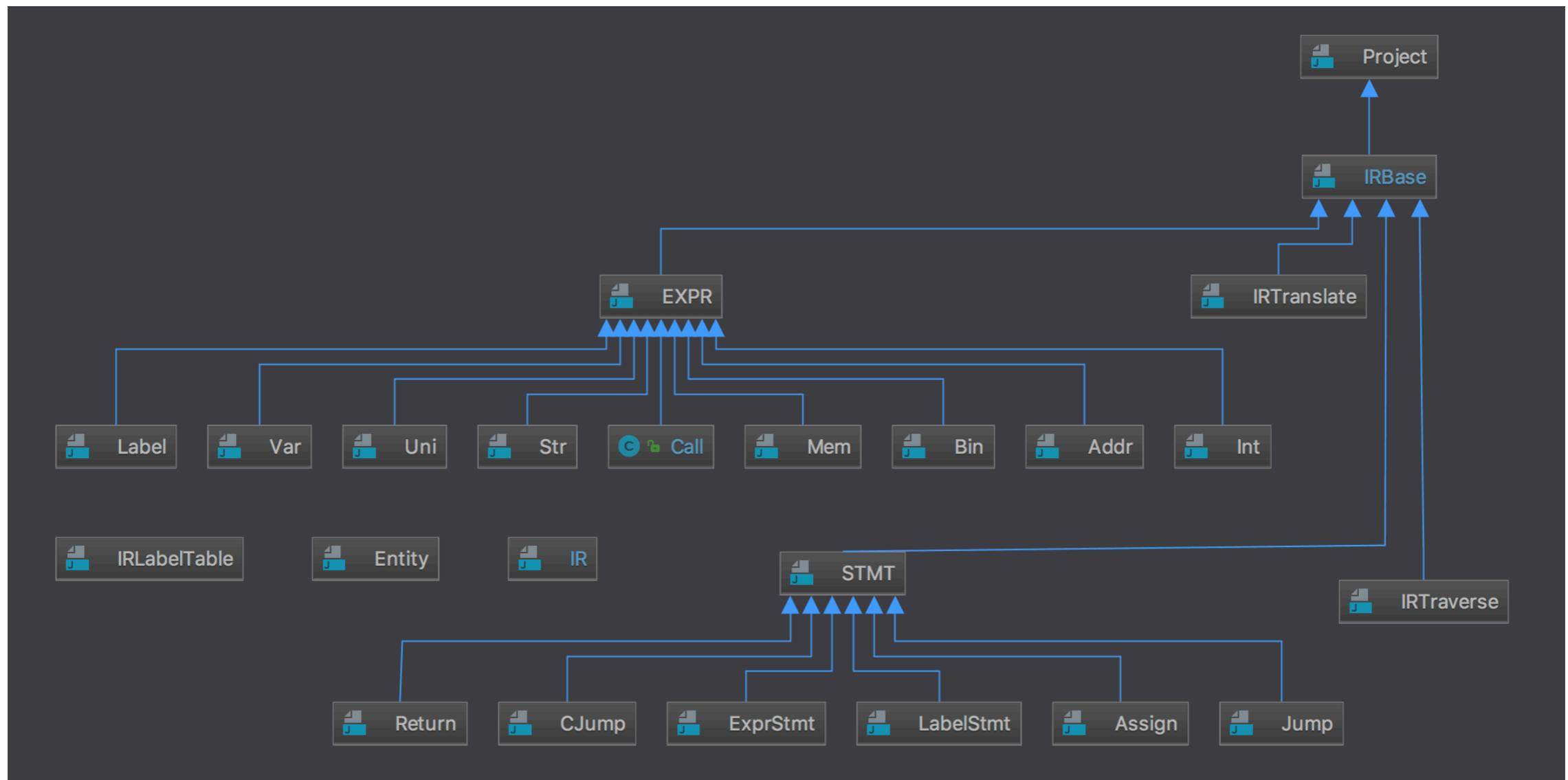
- 类似于visitor的递归的方法遍历AST
- 从下往上，对每个节点进行检查
- 不符合语法要求则抛出异常，否则对AST节点做进一步初始化，例如符号表的完善
- 只需要遍历一次

后端



IR

- 有点树形的IR。即Statement之间是线形的，部分Expression之间是树形的。相对AST做了很大的简化。

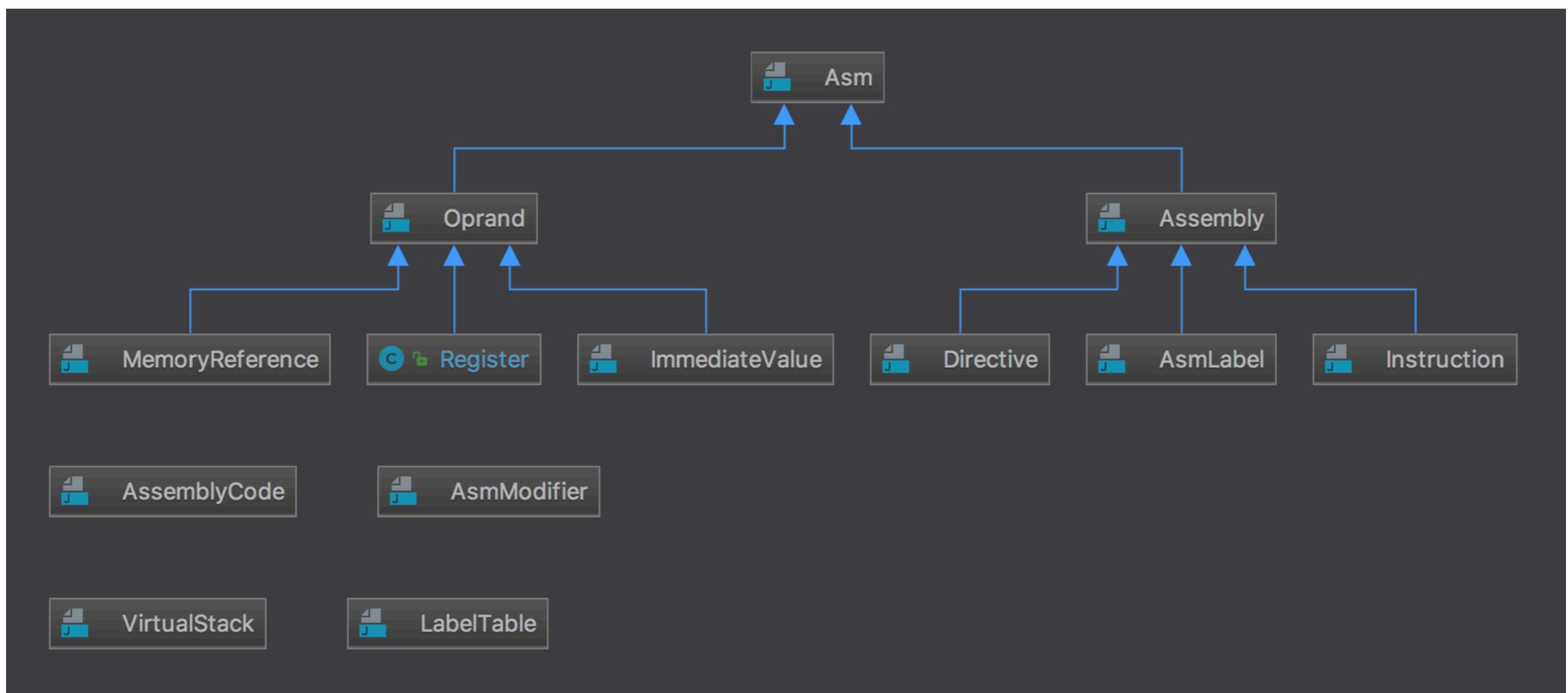


IRGenerator

- 简化AST并生成IR。
- 完成变量名的消解等工作
- 其中最麻烦的部分是New数组时的转换。解决的办法是将其递归地转化成多层循环并调用malloc
- 对每个函数生成CFG，为优化作准备

Asm 相关的类

- 包括操作数Oprand和用来表示一条汇编代码的Assembly两个基类。
- 例如MemoryReference类能够表示寄存器+偏移和Label+偏移等多种内存模式； AsmLabel来表示汇编代码里的标签Label。



CodeGenerator

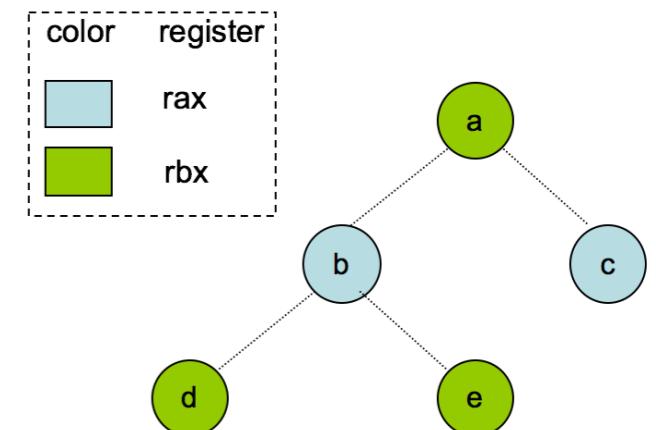
- `AssemblyCode`类：线性地保存`Assembly`的实例
- 初始化工作：首先是生成必须的汇编代码部分；然后初始化内建函数，并把全局变量的声明放到`.data`区，把字符串常量也存到`.data`区
- 随后对每个函数对应的IR以visitor的方式来访问（因为IR并不是完全线性的）。对每个函数生成一个`AssemblyCode`的实例，把这个过程生成的每条汇编代码都存到该实例中。

Translate

- 用来把各部分的AssemblyCode转化为汇编代码的字符串并结合起来，生成最后结果
- 每个Instruction都有对应的toSource()方法， 把它们按照对应的格式组合起来并做好对齐、美化、加注释的工作即可。

优化：寄存器分配

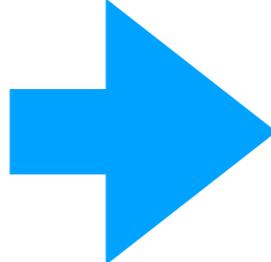
- 图染色的方法来分配寄存器
- 在IR生成之后，对每个函数的IR建立一个CFG，并进行活性分析
- 使用Kempe算法来染色，暂未考虑把寄存器拆成低位使用，所以NASM的寄存器共有16个（预留了两个寄存器来解决spill问题，并且不用rbp, rsp等有特殊作用的寄存器。虽然效率低了但是比较稳定）
- 寄存器不够时，将变量存到内存中



其他优化

- 字符串优化：把输出时的字符串加法变成多次输出。

```
print("hello" + " " + "world");
```



```
print("hello");
print(" ");
print("world");
```

- 如果IfStatement和WhileStatement的条件可以在程序运行前直接计算得到固定的结果的，就把结果计算出来判断会不会跳转，并删除掉不必要的汇编代码部分。

设计的数据

- 数据算法解决的问题是一个网络流问题
- 类比较多，其中内置了一个生成伪随机数的类，其中有计算gcd的内置的递归函数。还有用来解决网络流的类。
- 很多的while, for和if嵌套，并且有缺省的条件
- 加了很多while, if条件是定值的情况，比如while(true), if(false)
- 用文法糖new了比较大的数组
- 局部变量较多

在数据集上的表现

- 在大部分数据集上跑得很快，尤其在T362上比一般的同学快了两倍

```
int main(){
    int T = 0;
    for ( ; T < 1000000; ++T){
        A = T+1;
        B = T+1;
        C = T+1;
        ...
        if (T % 1000 == 0)
            println(toString(A) + " " + toString(B) + " " + toString(C));
    }
}
```

- 我认为主要原因在于做了字符串优化以及之前的IR的树形的结构做的比较好
- 在有些涉及逻辑判断的点跑得比较慢，本编译器在这个方面需要更多优化

感悟与收获

- 第一次写这种大工程，感觉收获还是很大的
- 学习了JAVA语言，真的很棒！
- 很多地方还可以做优化，如果时间允许的话使用SSA会更快
- 代码长，编译时间长，这大概是由于我的代码写的比较冗余，特判比较多
- 熟悉了git的用法
- 对计算机语言的了解更深了，以及了解了许多编译器的工作原理
- 熟悉了汇编代码的写法
- 最后代码交的很迟感到很抱歉，右手严重的腱鞘炎所以打代码只能用一根手指，然后代码又比较冗长写了一万多行。。
- 感谢郑怜悯、张哲恺、陈欣昊同学的帮助，感谢马融老师和助教组的付出

A photograph of a lush green cornfield in the foreground, with large, billowing white clouds against a clear blue sky in the background. A large, bold, white sans-serif font overlays the center of the image, reading "Thanks!"

Thanks!