可变变量

Kaleidoscope让我们可以直接用SSA形式构建LLVM IR，而前端不需要构建SSA表单，因为LLVM已经提供了经过高度优化和测试的支持。

首先，可变变量会导致SSA结构的复杂性。在所给示例中，返回指令之前X有两个不同的可能值，所以插入一个PHI节点来合并这两个值，在LLVM IR中需要明确的是，当降低可变变量的赋值时谁将放置phi节点。这里存在着一个问题，LLVM要求它的IR是SSA形式，它没有“非SSA”模式，但是SSA的构建需要非平凡的算法和数据结构，因此每个前端再现这种逻辑都是复杂且浪费的。

LLVM内存

解决方法为：虽然LLVM要求所有寄存器值都是SSA形式，但没有要求内存对象必须为SSA形式。因此可以考虑为函数中的每个可变对象创建一个堆栈变量，而此时需要明确LLVM如何表示堆栈变量。通过实例，发现了一种不需要创建Phi节点而处理任意可变变量的方法：①每个可变变量成为一个堆栈分配；②变量的每次读取都成为堆栈中的负载；③变量的每次更新都成为堆栈的存储；④获取变量的地址只是直接使用堆栈地址。但这样又出现了一个问题：引入了大量堆栈流量，这是一个主要的性能问题。但LLVM optimizer有一个名为“mem2reg”的高度优化通道来处理这种情况，将这种分配提升到SSA寄存器中，并根据需要插入Phi节点，它实现了用于构建SSA表单的标准“迭代优势边界”算法，并具有许多加速常见的退化情况的优化。

调整现有变量来进行变化

Kaleidoscope中的符号表在代码生成时由“NamedValues”进行管理，而为了支持变量变化，使得NamedValues可以保存相关变量的存储位置，这种改变是一种重构：改变了代码的结构，但本身并不改变编译器的行为。Kaleidoscope仅支持两种变量的改变：函数的传入参数和“for”循环的归纳变量。为了一致性，除了其他用户定义的变量之外，需要允许这些变量的变化，而因此这两者都需要内存位置。完成的一个功能改变属于变量引用，变量位于堆栈上，因此生成对它们的引用的代码需要从堆栈槽中产生负载。为支持可变参数变量，还需要进行分配，而在实例中，没有修改if / then / else表达式，所以仍然插入了一个PHI节点。在其他优化器运行之后，simplifycfg pass决定将返回指令克隆到“else”块的末尾，这可以消除一些分支和PHI节点。而由于符号表引用都被更新为使用堆栈变量，所以将添加赋值运算符。

新赋值运算符

在当前框架下，添加一个新的赋值运算符非常简单。可以在内部处理它，而非允许用户定义它。首先是设置优先级：当前解析器明确了二进制运算符的优先级，可以负责所有解析和AST生成；在有了变量后，赋值就比较简单，发出赋值的RHS创建一个存储，并返回一个链式赋值。而在拥有了赋值运算符后，也可改变循环变量和参数。

用户定义的局部变量

为了真正有用，可以定义自己的局部变量，因此需要添加用户定义的局部变量。添加var / in和对Kaleidoscope做的其他扩展一样，首先是扩展lexer；然后定义将构建的AST节点，var / in允许一次定义所有名称的列表，并且每个名称可以可选地具有初始值设定项值；接下来定义解析器片段，在解析完所有变量后，解析主体并创建AST节点。基本思想是先发出初始值设定项，创建alloca，然后更新符号表指向它，一旦所有变量安装在符号表中，将评估var / in表达式的主体，最后在返回之前，恢复以前的变量绑定。