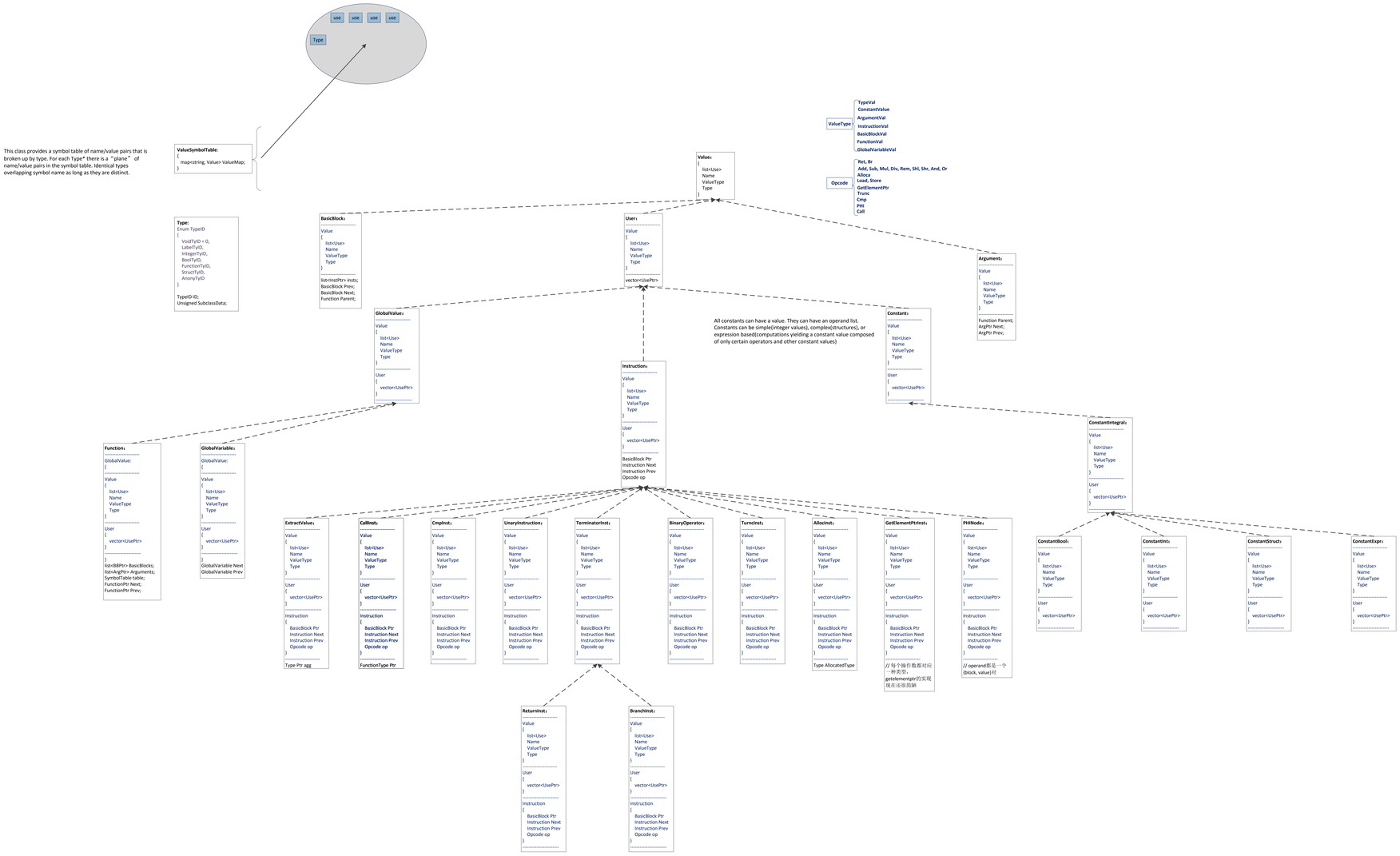
use
use
use
use
Type
This class provides a symbol table of name/value pairs that is broken up by type. For each Type* there is a "plane" of name/value pairs in the symbol table. Identical types overlapping symbol name as long as they are distinct.
ValueSymbolTable:
{
map<string, Value> ValueMap;
}
ValueType:
TypeVal
ConstantValue
ArgumentVal
InstructionVal
BasicBlockVal
FunctionVal
GlobalVariableVal
Opcode:
Ret, Br
Add, Sub, Mul, Div, Rem, Shl, Shr, And, Or
Alloca
Load, Store
GetElementPtr
Trunc
Cmp
Phi
Call
Type:
Enum TypeID
{
VoidTyID = 0,
LabelTyID,
IntegerTyID,
BoolTyID,
FunctionTyID,
StructTyID,
AnonyTyID
}
TypeID ID;
Unsigned SubclassData;
Value:
{
list<Use>
Name
ValueType
Type
}
BasicBlock:
Value
{
list<Use>
Name
ValueType
Type
}
list<InstPtr> insts;
BasicBlock Prev;
BasicBlock Next;
Function Parent;
User:
Value
{
list<Use>
Name
ValueType
Type
}
vector<UsePtr>
Argument:
Value
{
list<Use>
Name
ValueType
Type
}
Function Parent;
ArgPtr Next;
ArgPtr Prev;
GlobalValue:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
All constants can have a value. They can have an operand list. Constants can be simple(integer values), complex(structures), or expression based(computations yielding a constant value composed of only certain operators and other constant values)
Constant:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
Function:
GlobalValue:
{
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
list<BBPtr> BasicBlocks;
list<ArgPtr> Arguments;
SymbolTable table;
FunctionPtr Next;
FunctionPtr Prev;
GlobalVariable:
GlobalValue:
{
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
GlobalVariable Next
GlobalVariable Prev
ConstantIntegral:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
ExtractValue:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
Type Ptr agg
CallInst:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
FunctionType Ptr
CmpInst:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
UnaryInstruction:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
TerminatorInst:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
BinaryOperator:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
TurncInst:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
AllocInst:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
Type AllocatedType
GetElementPtrInst:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
// 每个操作数都对应一种类型，
getelementptr的实现现在还很简陋
PHINode:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
// operand都是一个{block, value}对
ConstantBool:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
ConstantInt:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
ConstantStruct:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
ConstantExpr:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
ReturnInst:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}
BranchInst:
Value
{
list<Use>
Name
ValueType
Type
}
User
{
vector<UsePtr>
}
Instruction
{
BasicBlock Ptr
Instruction Next
Instruction Prev
Opcode op
}