程序依赖图——以SVF为例

@canliture

大纲

- 1. SVF简介
- 2. Program Assignment Graph (PAG / SVFIR)
- 3. Constraint Graph (CG)
- 4. ICFG (Inter-procedural Control Flow Graph)
- 5. Call Graph
- 6. (Sparse) Value Flow Graph (SVFG)
- 7. Control Dependence Graph (CDG)
- 8. Program Dependence Graph (PDG)
- 9. SVF二次开发
- 10. 参考资料

1. SVF简介

- SVF: Static Value Flow (Data Dependence, Def-Use)
- 提供一套层次化，全程序的指针分析框架
  - 层次化: flow-context insensitive whole program pointer analysis -> SVFG Construction -> client analysis / (sparse) flow-sensitive analysis
  - 全程序: 相对于Modular Analysis, Demand-Driven Analysis
- 基于这套指针分析框架 (flow insensitive)
  - 构造全程序数据依赖图
  - 作流敏感（稀疏）指针分析

1. SVF简介

- 基础数据结构
  - 图: template<class NodeTy, class EdgeTy> class GenericGraph;
    - map<u32_t, NodeTy*>::iterator begin/end(); // foreach 迭代图
    - NodeType* getGNode(u32_t id); // 通过节点ID获取节点
  - 节点: template<class NodeTy, class EdgeTy> class GenericNode;
    - u32_t getId(); // 节点ID
    - u64_t getNodeKind(); // 节点类型
    - set<EdgeTy*> getOutEdges(); // 出边
    - set<EdgeTy*> getInEdges(); // 入边
  - 边: template<class NodeTy> class GenericEdge;
    - u64_t getEdgeKind(); // 边类型
    - NodeTy* getSrcNode(); // source node
    - NodeTy* getDstNode(); // destination node
  - 工作队列：template<class Data> class FIFOWorkList; // 去重队列
    - bool empty(); // 判空
    - bool push(const Data &data); // 入队
    - Data pop(); // 出队

- SCC-DAG
  - Template<class GraphType> class SCCDetection;
    - SCCDetection(GraphType &GT);
    - void find([set<u32_t> nodeSet]);
    - stack<u32_t> topoNodeStack();

2. Program Assignment Graph

- 程序赋值图
  - 紧凑地表示变量间的赋值关系 (load, store, gep, copy, ...)
  - 为Andersen式指针分析提供初始的指针约束关系图
    - Andersen式: flow-context insensitive
    - 指针约束关系图：集合包含关系可以表示成图上指针间边的关系

边的类型	指令	连边
Address 边	p = alloca$_O$	alloca$_O$ $\xrightarrow{Addr}$ p
Copy 边	p = q	q $\xrightarrow{Copy}$ p
Load 边	p = *q	q $\xrightarrow{Load}$ p
Store 边	*p = q	q $\xrightarrow{Store}$ p
Field 边	p = q gep f	q $\xrightarrow{.f}$ p

-

2. Program Assignment Graph

- 举个例子

```
clang -S -c -Xclang -disable-O0-optnone -fno-discard-value-names -emit-llvm -o svfir.ll svfir.c
opt -S -mem2reg -o  svfir.ll svfir.ll
../cmake-build-debug/bin/wpa -nander -dump-pag svfir.ll -extapi=/Users/liture/CLionProjects/SVF-Dev-Env/github/SVF/svf-llvm/lib/extapi.ll
dot -Tsvg -o svfir_initial.svg svfir_initial.dot
```

```c
typedef struct st { char f1; char f2; } ST;

void swap(char **p, char **q);

int main () {
    char a1;
    ST st;
    char *a = &a1;
    char *b = &(st.f2);
    swap(&a, &b);
}

void swap(char **p, char **q) {
    char *t = *p;
    *p = *q;
    *q = t;
}
```

```llvm
%struct.st = type { i8, i8 }

; Function Attrs: noinline nounwind ssp uwtable
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
  %st = alloca %struct.st, align 1
  %a = alloca i8*, align 8
  %b = alloca i8*, align 8
  store i8* %a1, i8** %a, align 8
  %f2 = getelementptr inbounds %struct.st, %struct.st* %st, i32 0, i32 1
  store i8* %f2, i8** %b, align 8
  call void @swap(i8** %a, i8** %b)
  ret i32 0
}

; Function Attrs: noinline nounwind ssp uwtable
define void @swap(i8** %p, i8** %q) #0 {
entry:
  %0 = load i8*, i8** %p, align 8
  %1 = load i8*, i8** %q, align 8
  store i8* %1, i8** %p, align 8
  store i8* %0, i8** %q, align 8
  ret void
}
```

-

2. Program A Graph

```c
typedef struct st { char f1; char f2; } ST;

void swap(char **p, char **q);

int main () {
    char a1;
    ST st;
    char *a = &a1;
    char *b = &(st.f2);
    swap(&a, &b);
}

void swap(char **p, char **q) {
    char *t = *p;
    *p = *q;
    *q = t;
}
```

```llvm
%struct.st = type { i8, i8 }

; Function Attrs: noinline nounwind ssp uwtable
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
  %st = alloca %struct.st, align 1
  %a = alloca i8*, align 8
  %b = alloca i8*, align 8
  store i8* %a1, i8** %a, align 8
  %f2 = getelementptr inbounds %struct.st, %struct.st* %st, i32 0, i32 1
  store i8* %f2, i8** %b, align 8
  call void @swap(i8** %a, i8** %b)
  ret i32 0
}

; Function Attrs: noinline nounwind ssp uwtable
define void @swap(i8** %p, i8** %q) #0 {
entry:
  %0 = load i8*, i8** %p, align 8
  %1 = load i8*, i8** %q, align 8
  store i8* %1, i8** %p, align 8
  store i8* %0, i8** %q, align 8
  ret void
}
```

SVFIR

2. Program Assignment Graph

- class SVFIR : IRGraph : GenericGraph<SVFVar, SVFStmt>. // typedef SVFIR PAG;
    - class SVFVar : GenericNode<SVFVar, SVFStmt> // typedef SVFVar PAGNode;
    - class SVFStmt : GenericEdge<SVFVar> // typedef SVFStmt PAGEdge;
-

2. Program Assignment Graph

- class SVFIR : IRGraph : GenericGraph<SVFVar, SVFStmt>.   // typedef SVFIR PAG;

- class SVFVar : GenericNode<SVFVar, SVFStmt>   // typedef SVFVar PAGNode;
  - ObjVar
    - GepObjVar: obj->field
    - FIObjVar: arr[i], arr*, alloca
    - DymmyObjVar
  - ValVar
    - GepValVar: var->field
    - RetPN
    - VarArgPN:
    - DymmyValVar

- class SVFStmt : GenericEdge<SVFVar>   // typedef SVFStmt PAGEdge;
  - AssignStmt
    - AddrStmt: p = alloc(…)
    - CopyStmt: p = q
    - StoreStmt: *p = q
    - LoadStmt: p = *q
    - CallPE : formal_param <- actual_arg
      - TDForkPE
    - RetPE : actual_ret <- formal_ret
      - TDJoinPE
    - GepStmt: p = t->f
  - MultiOpndStmt
    - PhiStmt: p = phi(a1, a2, …, an)
    - SelectStmt: q = p ? q : r
    - CmpStmt: cmp = a cmp_op b
    - BinaryOpStmt: p = a op b
  - UnaryOpStmt: p = op q
  - BranchStmt: switch / if-then-else

3.Constraint Graph

- Andersen式指针分析中，指针约束图，集合包含关系约束图
  - 在开始进行Andersen之前，ConstraintGraph为PAG的一份Copy
  - 迭代进行Andersen指针分析，在ConstraintGraph上添加边，直到不动点
    - 不动点
      - 不能再向ConstraintGraph上添加更多的边 或者 节点

3.Constraint Graph

- 举个例子

- $$\frac{x = \&o}{pt(x) \supseteq \{o\}} \qquad \frac{x = y}{pt(x) \supseteq pt(y)}$$

  $$\frac{x = {}^*y \quad o \in pt(y)}{pt(x) \supseteq pt(o)} \qquad \frac{{}^*x = y \quad o \in pt(x)}{pt(o) \supseteq pt(y)}$$

-

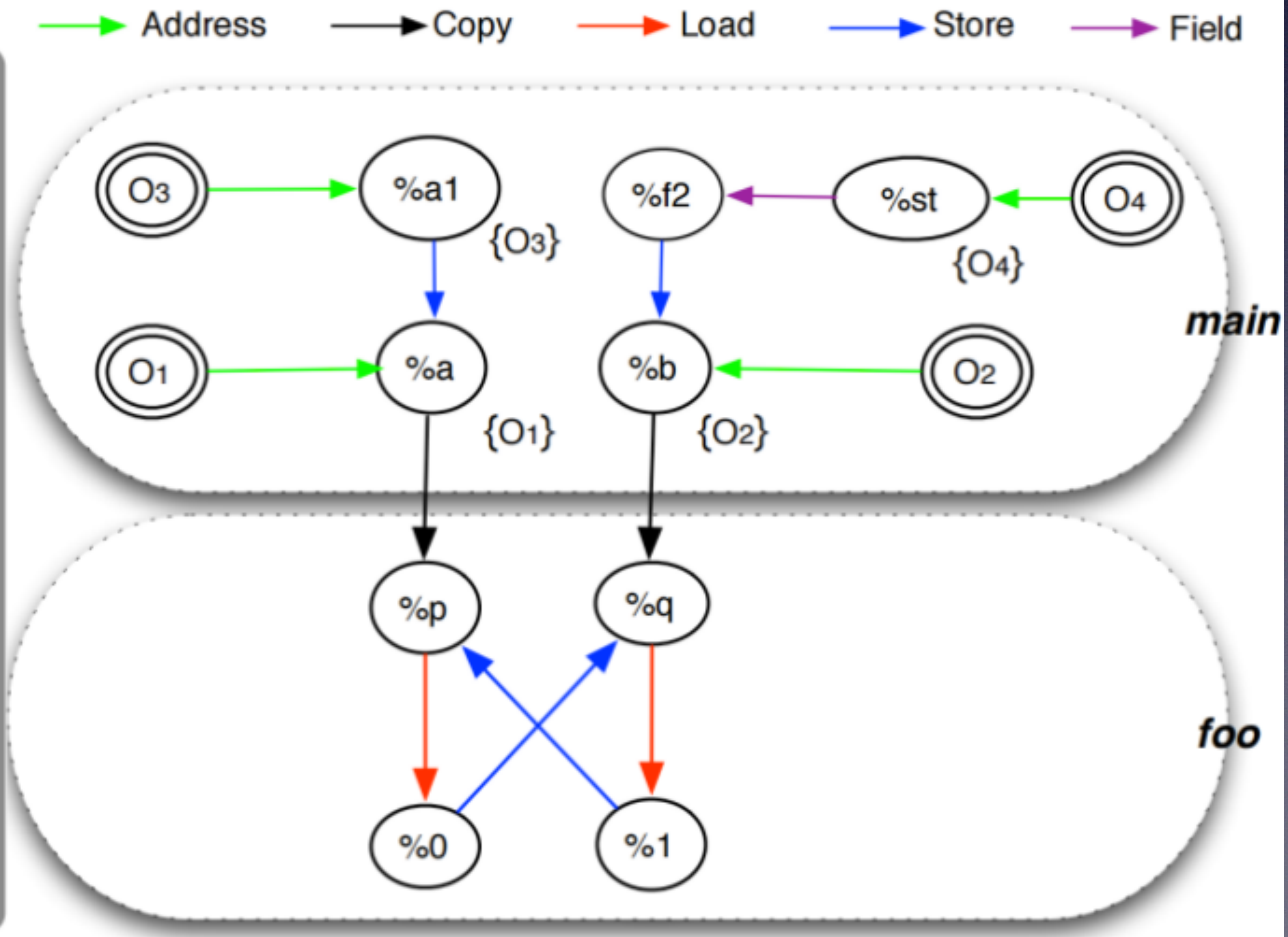
3.Constraint Graph

```c
struct st{
      char f1;
      char f2;
};
typedef struct st ST;

int main(){
      char a1; ST st;
      char *a = &a1;
      char *b = &(st.f2);
      swap(&a,&b);
}
void swap(char **p, char **q){
      char* t = *p;
      *p = *q;
      *q = t;
}
```

```llvm
define i32 @main() {
entry:
      %a = alloca i8*, align 8                // O1
      %b = alloca i8*, align 8                // O2
      %a1 = alloca i8, align 1                // O3
      %st = alloca %struct.st, align 1        // O4
      store i8* %a1, i8** %a, align 8
      %f2 = getelementptr inbounds %struct.st, %struct.st* %st, i32 0, i32 1
      store i8* %f2, i8** %b, align 8
      call void @swap(i8** %a, i8** %b)
      ret i32 0
}
define void @swap(i8** %p, i8** %q) {
entry:
      %0 = load i8** %p, align 8
      %1 = load i8** %q, align 8
      store i8* %1, i8** %p, align 8
      store i8* %0, i8** %q, align 8
      ret void
}
```

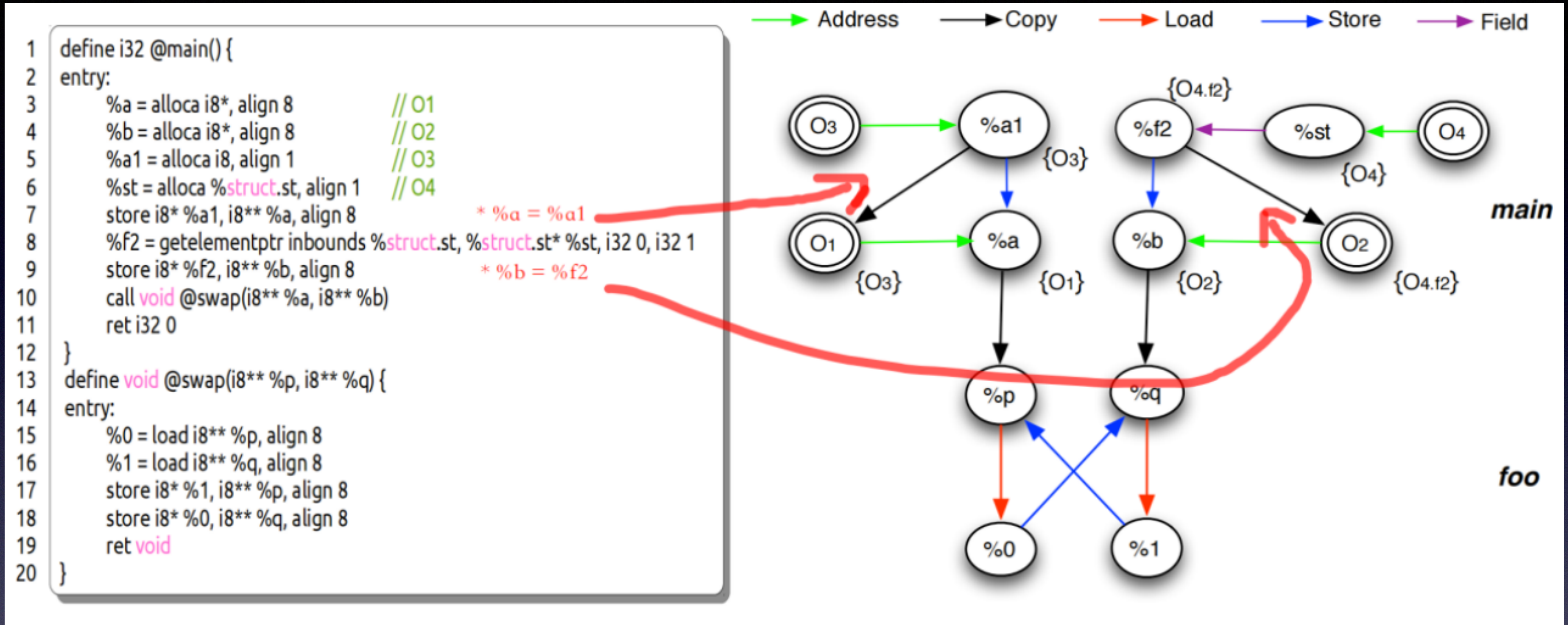

```llvm
define i32 @main() {
entry:
      %a = alloca i8*, align 8                // O1
      %b = alloca i8*, align 8                // O2
      %a1 = alloca i8, align 1                // O3
      %st = alloca %struct.st, align 1        // O4
      store i8* %a1, i8** %a, align 8
      %f2 = getelementptr inbounds %struct.st, %struct.st* %st, i32 0, i32 1
      store i8* %f2, i8** %b, align 8
      call void @swap(i8** %a, i8** %b)
      ret i32 0
}
define void @swap(i8** %p, i8** %q) {
entry:
      %0 = load i8** %p, align 8
      %1 = load i8** %q, align 8
      store i8* %1, i8** %p, align 8
      store i8* %0, i8** %q, align 8
      ret void
}
```


LLVM IR



Program Assignment Graph: ConstraintGraph初始状态

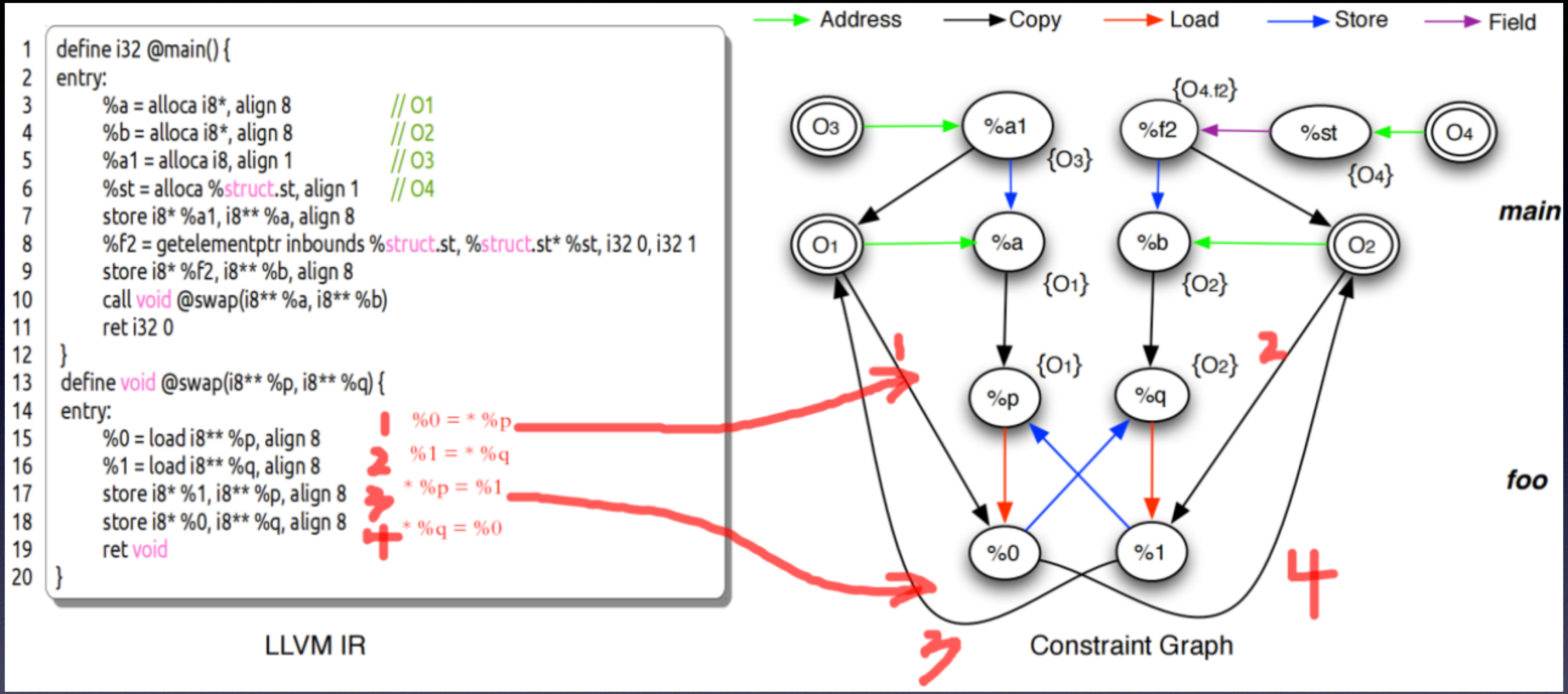
3.Constraint Graph



LLVM IR　　Constraint Graph

ConstraintGraph: 迭代，传播指向集，添加新边 (1)

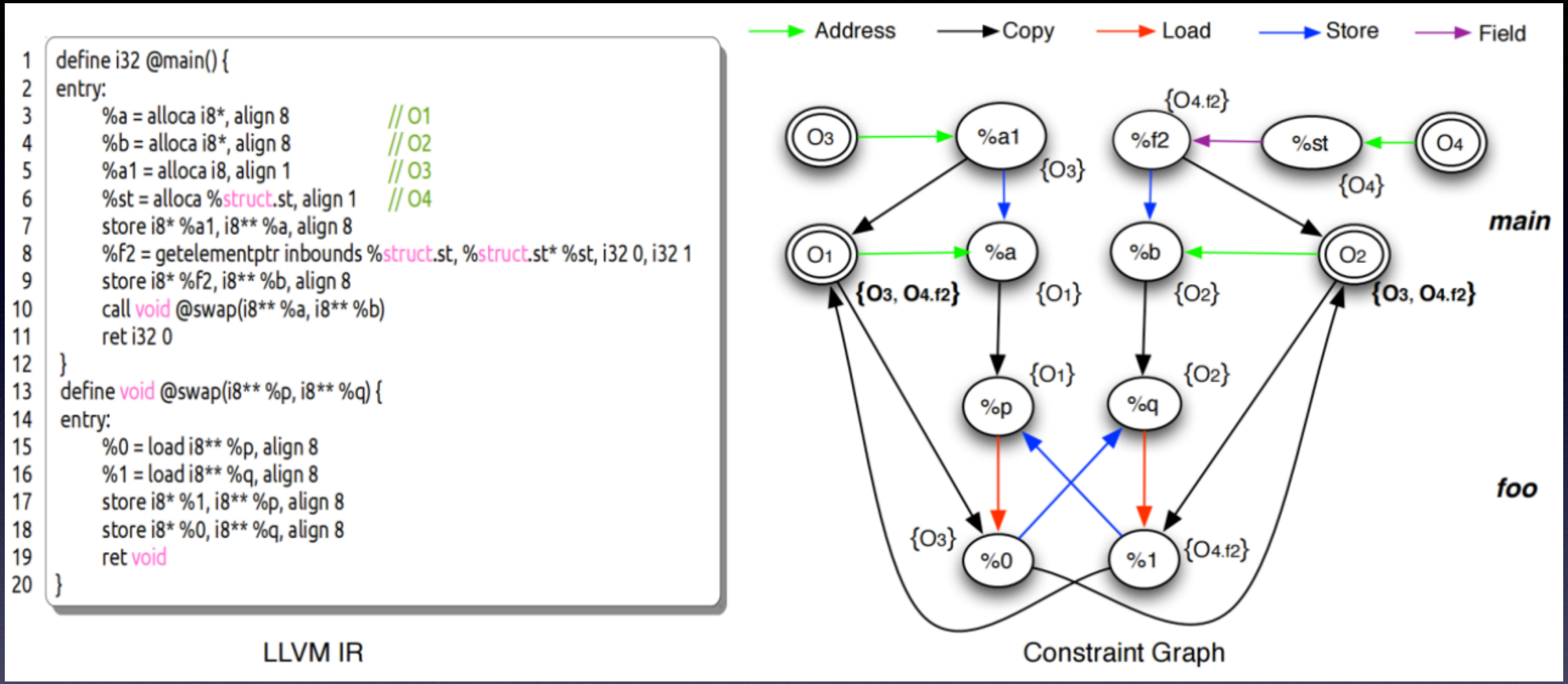
3.Constraint Graph



ConstraintGraph: 迭代，传播指向集，添加新边 (2)

•

3.Constraint Graph



ConstraintGraph: 迭代稳定，达到不动点 (3)

•

3.Constraint Graph

- class ConstraintGraph : GenericGraph<ConstraintNode, ConstraintEdge>
- class ConstraintNode : GenericGraph<ConstraintNode, ConstraintEdge>
    - get/remove: AddrIn/OutEdges, CopyIn/OutEdges, GepIn/OutEdges, LoadIn/OutEdges, StoreIn/OutEdges,
- class ConstraintEdge : GenericGraph<ConstraintEdge>
    - AddrCGEdge, CopyCGEdge, LoadCGEdge, StoreCGEdge
    - GepCGEdge
        - NormalGepEdge: fixed offset size
        - VariantGepCGEdge: variant offset size, 未知指针算数式的访问, 如p = (q + i)

3.Constraint Graph

AndersenBase::analyze()

- TOPLAS 2007 Efficient Field-Sensitive Pointer Analysis of C

4.Inter-Procedural Control Flow Graph

- class ICFG : GenericGraph<ICFGNode, ICFGEdge>
- class ICFGNode : GenericNode<ICFGNode, ICFGEdge>
    - GlobalICFGNode
    - IntraICFGNode
    - InterICFGNode
        - CallICFGNode
            - SVFInstruction *getCallSite();
            - RetICFGNode *getRetICFGNode();
            - Vector<SVFVar *> getFormalParams();
        - RetICFGNode
            - SVFInstruction *getCallSite();
            - CallICFGNode *getCallICFGNode();
            - SVFVar *getActualRet();
        - FunEntryICFGNode
            - vector<SVFVar *> getFormalParams();
        - FunExitICFGNode
            - SVFVar *getFormalRet();

- class ICFGEdge : GenericEdge<ICFGNode>
    - IntraCFGEdge
    - CallCFGEdge
    - RetCFGEdge

4.Inter-Procedural Control Flow Graph

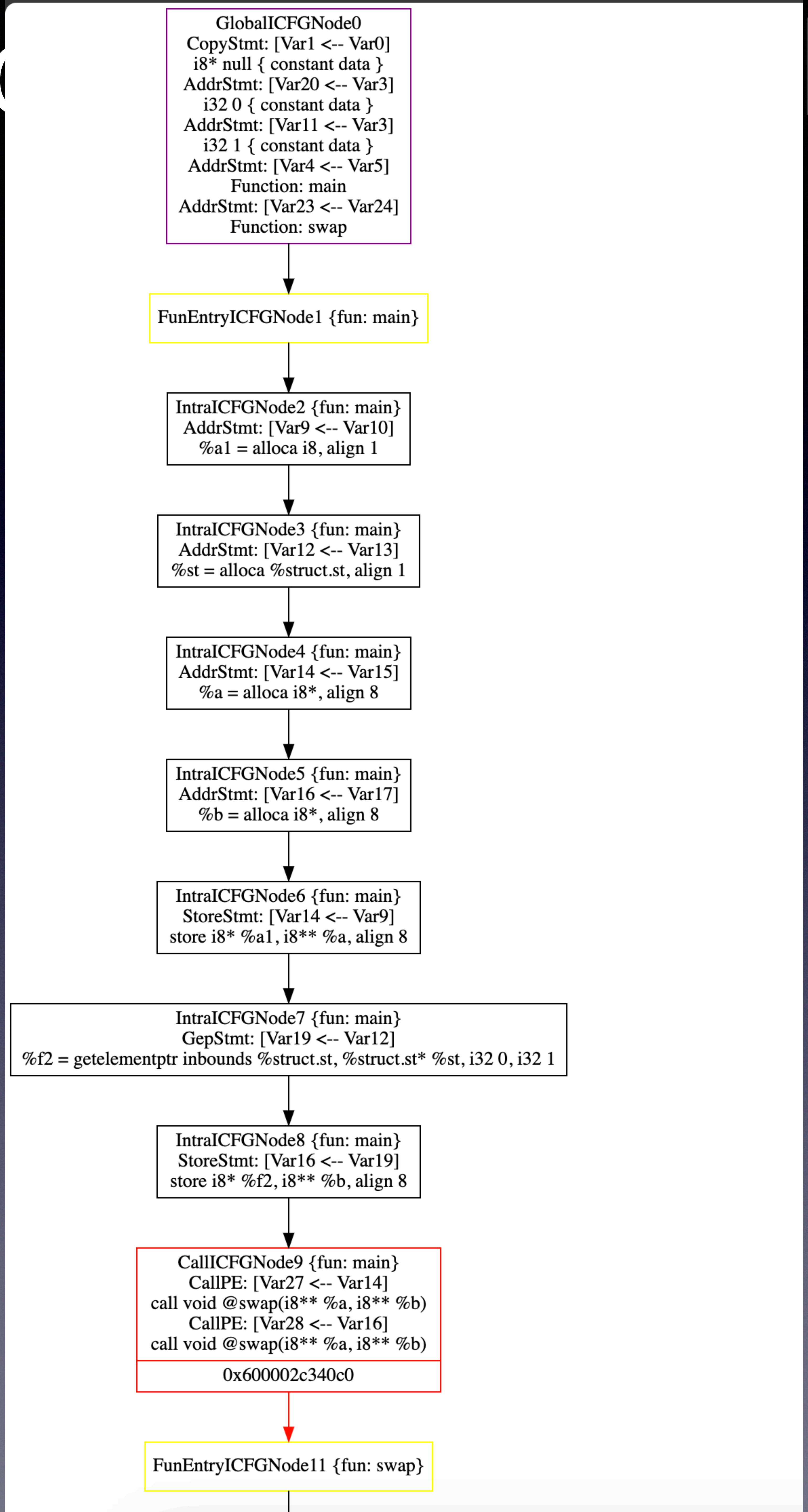
- 举个例子

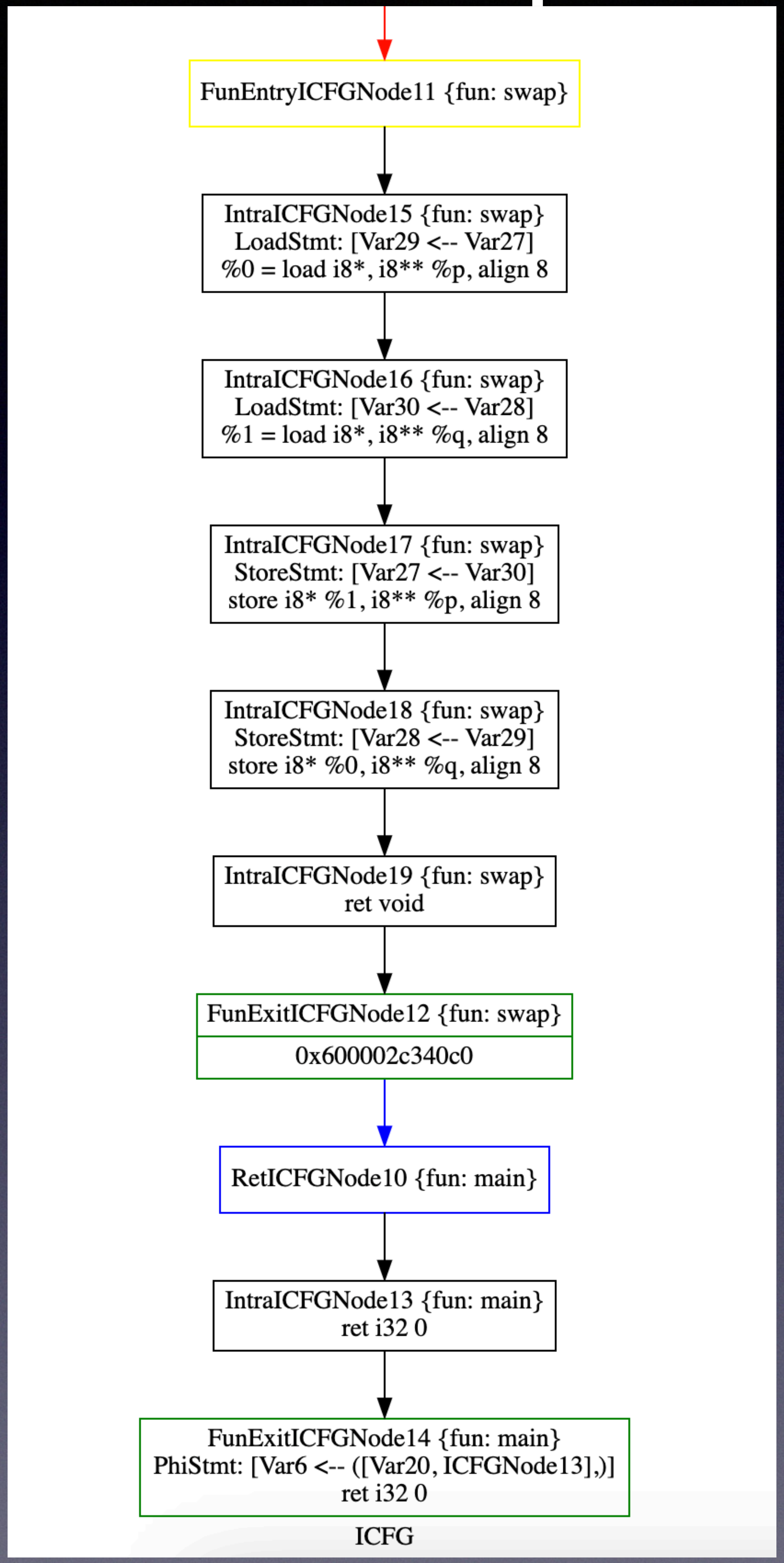
```
../cmake-build-debug/bin/wpa -ander -dump-icfg svfir.ll -extapi=/Users/liture/CLionProjects/SVF-Dev-Env/github/SVF/svf-llvm/lib/extapi.ll
dot -Tsvg -o icfg_initial.svg icfg_initial.dot
```

-

4.Inter-Proced… Flow Graph

```
%struct.st = type { i8, i8 }

; Function Attrs: noinline nounwind ssp uwtable
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
  %st = alloca %struct.st, align 1
  %a = alloca i8*, align 8
  %b = alloca i8*, align 8
  store i8* %a1, i8** %a, align 8
  %f2 = getelementptr inbounds %struct.st, %struct.st* %st, i32 0, i32 1
  store i8* %f2, i8** %b, align 8
  call void @swap(i8** %a, i8** %b)
  ret i32 0
}

; Function Attrs: noinline nounwind ssp uwtable
define void @swap(i8** %p, i8** %q) #0 {
entry:
  %0 = load i8*, i8** %p, align 8
  %1 = load i8*, i8** %q, align 8
  store i8* %1, i8** %p, align 8
  store i8* %0, i8** %q, align 8
  ret void
}
```





5.Call Graph

- class PTACallGraph : GenericGraph<PTACallGraphNode, PTACallGraphEdge>
- class ThreadCallGraph : PTACallGraph
- 获取调用图的拓扑序列
    - PointerAnalysis *pta = new AndersenWaveDiff(pag); // WPAPass::runPointerAnalysis
    - PTACallGraph *cg = pta->getPTACallGraph();
    - SCCDetection *callGraphSCC = new SCCDetection(cg); // MRGenerator::MRGenerator

5.Call Graph

- Call Graph应用举例：MOD/REF

MemRegion.cpp

```cpp
/*!
 * Get the reverse topo order of scc call graph
 */
void MRGenerator::getCallGraphSCCRevTopoOrder(WorkList& worklist)
{

    NodeStack& topoOrder = callGraphSCC->topoNodeStack();
    while(!topoOrder.empty())
    {
        NodeID callgraphNodeID = topoOrder.top();
        topoOrder.pop();
        worklist.push(callgraphNodeID);
    }
```

MemRegion.cpp

```cpp
    WorkList worklist;
    getCallGraphSCCRevTopoOrder(worklist);

    while(!worklist.empty())
    {
        NodeID callGraphNodeID = worklist.pop();
        /// handle all sub scc nodes of this rep node
        const NodeBS& subNodes = callGraphSCC->subNodes(callGraphNodeID);
        for(NodeBS::iterator it = subNodes.begin(), eit = subNodes.end(); it!=eit; ++it)
        {
            PTACallGraphNode* subCallGraphNode = callGraph->getCallGraphNode(*it);
            /// Get mod-ref of all callsites calling callGraphNode
            modRefAnalysis(subCallGraphNode,worklist);
        }
    }
```

5.Call Graph

MemRegion.cpp

```cpp
/*!
 * Get the reverse topo order of scc call graph
 */
void MRGenerator::getCallGraphSCCRevTopoOrder(WorkList& worklist)
{

    NodeStack& topoOrder = callGraphSCC->topoNodeStack();
    while(!topoOrder.empty())
    {
        NodeID callgraphNodeID = topoOrder.top();
        topoOrder.pop();
        worklist.push(callgraphNodeID);
    }
}
```

MemRegion.cpp

```cpp
    WorkList worklist;
    getCallGraphSCCRevTopoOrder(worklist);

    while(!worklist.empty())
    {
        NodeID callGraphNodeID = worklist.pop();
        /// handle all sub scc nodes of this rep node
        const NodeBS& subNodes = callGraphSCC->subNodes(callGraphNodeID);
        for(NodeBS::iterator it = subNodes.begin(), eit = subNodes.end(); it!=eit; ++it)
        {
            PTACallGraphNode* subCallGraphNode = callGraph->getCallGraphNode(*it);
            /// Get mod-ref of all callsites calling callGraphNode
            modRefAnalysis(subCallGraphNode,worklist);
        }
    }
```

```cpp
/*!
 * Call site mod-ref analysis
 * Compute mod-ref of all callsites invoking this call graph node
 */
void MRGenerator::modRefAnalysis(PTACallGraphNode* callGraphNode, WorkList& worklist)
{

    /// add ref/mod set of callee to its invocation callsites at caller
    for(PTACallGraphNode::iterator it = callGraphNode->InEdgeBegin(), eit = callGraphNode->InEdgeEn
            it!=eit; ++it)
    {
        PTACallGraphEdge* edge = *it;

        /// handle direct callsites
        for(PTACallGraphEdge::CallInstSet::iterator cit = edge->getDirectCalls().begin(),
                ecit = edge->getDirectCalls().end(); cit!=ecit; ++cit)
        {
            NodeBS mod, ref;
            const CallICFGNode* cs = (*cit);
            bool modrefchanged = handleCallsiteModRef(mod, ref, cs, callGraphNode->getFunction());
            if(modrefchanged)
                worklist.push(edge->getSrcID());
        }
        /// handle indirect callsites
        for(PTACallGraphEdge::CallInstSet::iterator cit = edge->getIndirectCalls().begin(),
                ecit = edge->getIndirectCalls().end(); cit!=ecit; ++cit)
        {
            NodeBS mod, ref;
            const CallICFGNode* cs = (*cit);
            bool modrefchanged = handleCallsiteModRef(mod, ref, cs, callGraphNode->getFunction());
            if(modrefchanged)
                worklist.push(edge->getSrcID());
        }
    }
}
```

6.Sparse Value Flow Graph

- class VFG : GenericGraph<VFGNode, VFGEdge>
- class VFGNode : GenericNode<VFGNode, VFGEdge>
    - StmtVFGNode
        - LoadVFGNode
        - StoreVFGNode
        - CopyVFGNode
        - GepVFGNode
        - AddrVFGNode
    - PHIVFGNode
        - IntraPHIVFGNode
        - InterPHIVFGNode
    - BinaryOPVFGNode
    - UnaryOPVFGNode
    - CmpVFGNode
    - BranchVFGNode
    - ArgumentVFGNode
        - ActualParamVFGNode
        - ActualRetVFGNode
        - FormalParamVFGNode
        - FormalRetVFGNode

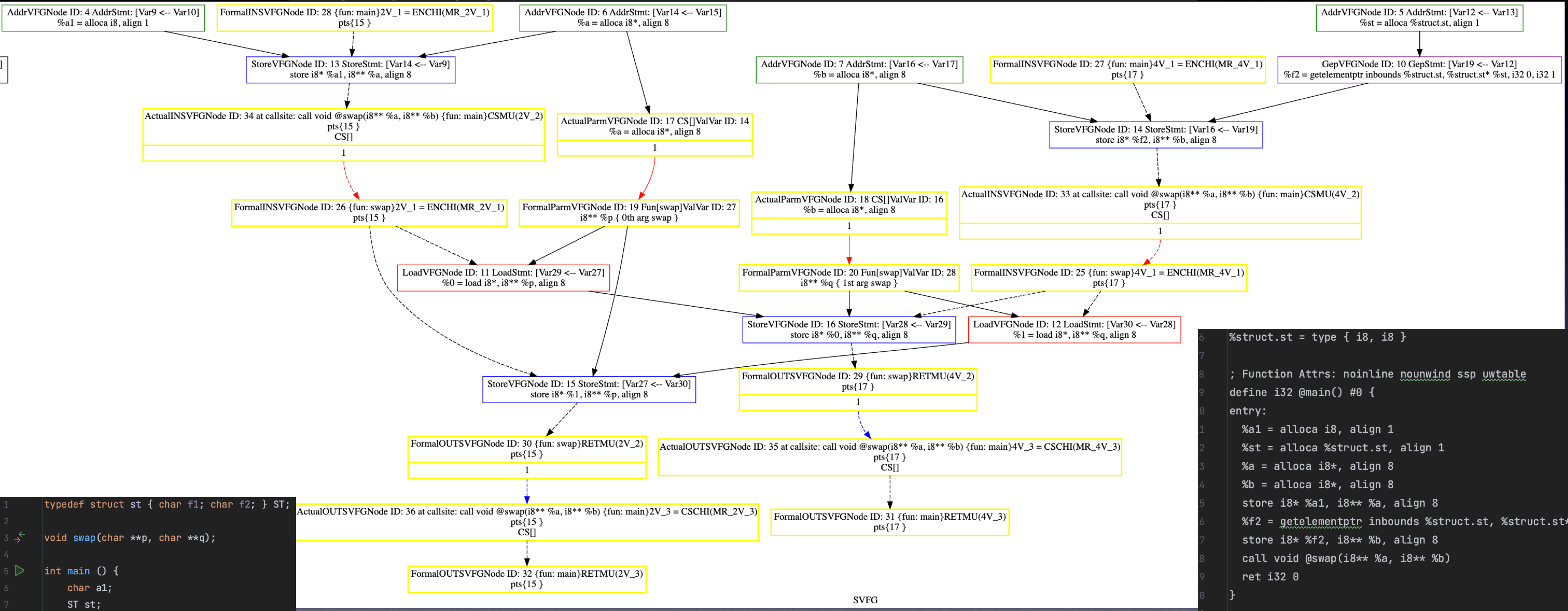
- typedef VFGNode SVFGNode
- MRSVFGNode
    - FormalINSVFGNode
    - FormalOUTSVFGNode
    - ActualINSVFGNode
    - ActualOUTSVFGNode
    - MSSAPHISVFGNode
        - IntraMSSAPHISVFGNode
        - InterMSSAPHISVFGNode

- typedef VFGEdge SVFGEdge
- class VFGEdge : GenericEdge<VFGNode>
    - DirectSVFGEdge
        - IntraDirSVFGEdge
        - CallDirSVFGEdge
        - RetDirSVFGEdge
    - IndirectSVFGEdge
        - IntraIndSVFGEdge
        - CallIndSVFGEdge
        - RetIndSVFGEdge
        - ThreadMHPIndSVFEdge

6.Sparse Value Flow Graph

- 举个例子

```
../cmake-build-debug/bin/wpa -ander -svfg -dump-vfg svfir.ll -extapi=/Users/liture/CLionProjects/SVF-Dev-Env/github/SVF/svf-llvm/lib/extapi.ll
dot -Tsvg -o svfg_final.svg svfg_final.dot
```



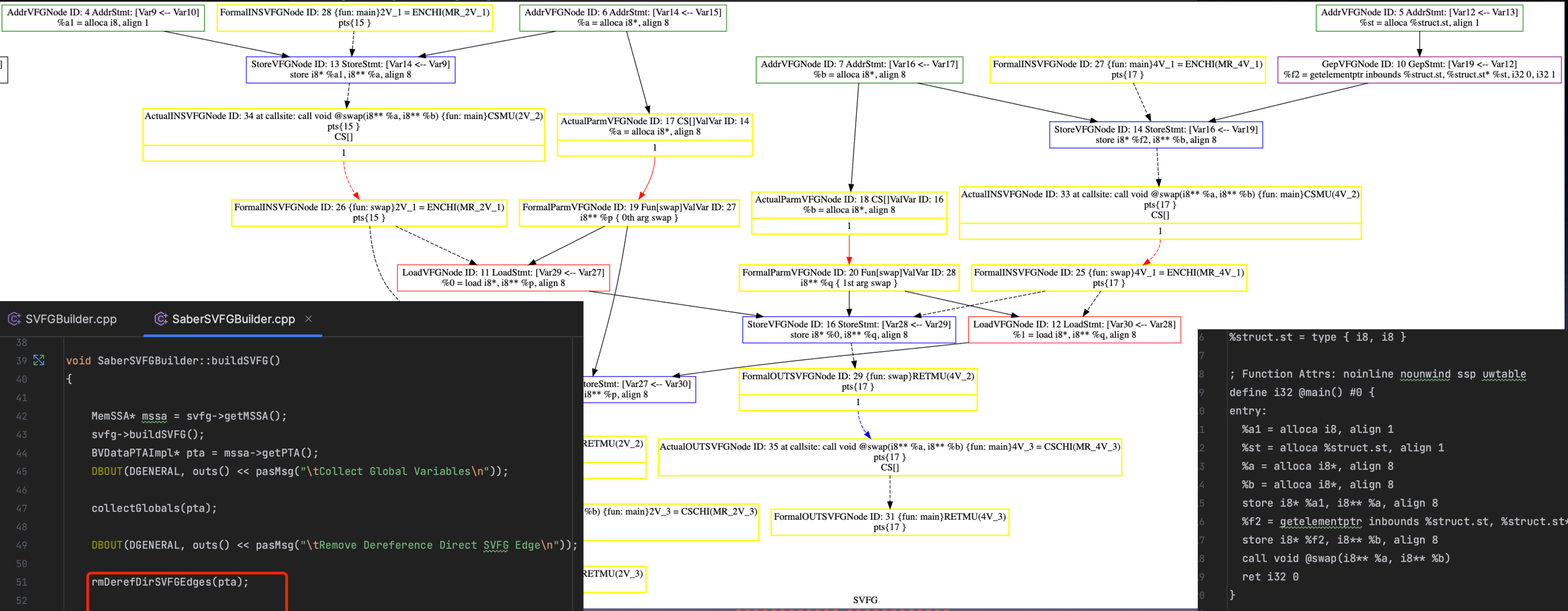

```c
typedef struct st { char f1; char f2; } ST;

void swap(char **p, char **q);

int main () {
    char a1;
    ST st;
    char *a = &a1;
    char *b = &(st.f2);
    swap(&a, &b);
}

void swap(char **p, char **q) {
    char *t = *p;
    *p = *q;
    *q = t;
}
```

```llvm
%struct.st = type { i8, i8 }

; Function Attrs: noinline nounwind ssp uwtable
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
  %st = alloca %struct.st, align 1
  %a = alloca i8*, align 8
  %b = alloca i8*, align 8
  store i8* %a1, i8** %a, align 8
  %f2 = getelementptr inbounds %struct.st, %struct.st* %st, i32 0, i32 1
  store i8* %f2, i8** %b, align 8
  call void @swap(i8** %a, i8** %b)
  ret i32 0
}

; Function Attrs: noinline nounwind ssp uwtable
define void @swap(i8** %p, i8** %q) #0 {
entry:
  %0 = load i8*, i8** %p, align 8
  %1 = load i8*, i8** %q, align 8
  store i8* %1, i8** %p, align 8
  store i8* %0, i8** %q, align 8
  ret void
}
```

6.Sparse Value Flow Graph

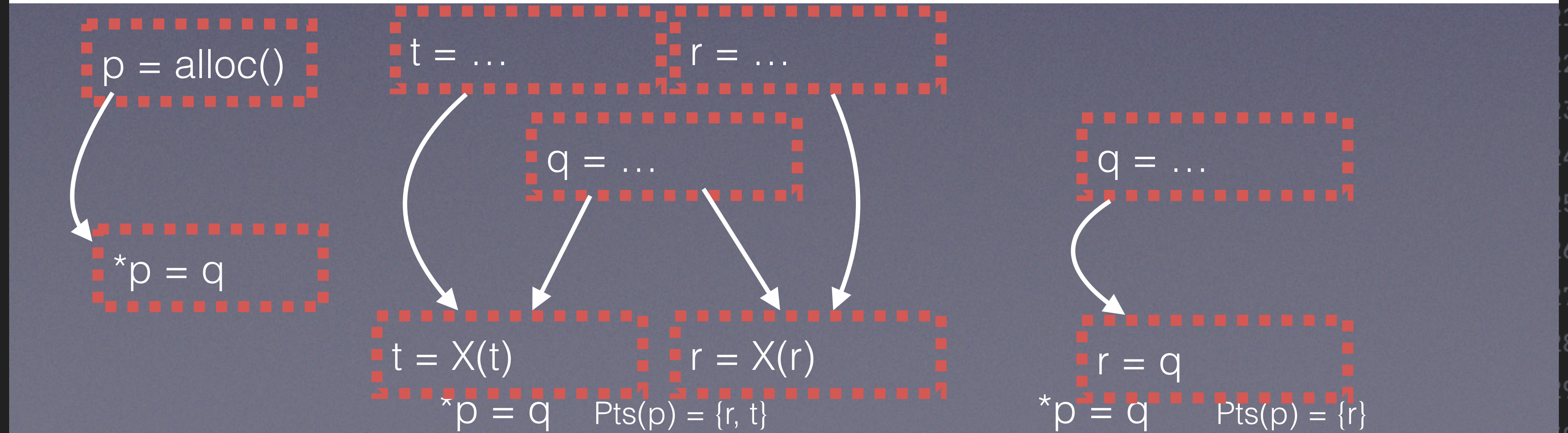
- 举个例子

```
../cmake-build-debug/bin/wpa -ander -svfg -dump-vfg svfir.ll -extapi=/Users/liture/CLionProjects/SVF-Dev-Env/github/SVF/svf-llvm/lib/extapi.ll
dot -Tsvg -o svfg_final.svg svfg_final.dot
```




```cpp
void SaberSVFGBuilder::buildSVFG()
{

    MemSSA* mssa = svfg->getMSSA();
    svfg->buildSVFG();
    BVDataPTAImpl* pta = mssa->getPTA();
    DBOUT(DGENERAL, outs() << pasMsg("\tCollect Global Variables\n"));

    collectGlobals(pta);

    DBOUT(DGENERAL, outs() << pasMsg("\tRemove Dereference Direct SVFG Edge\n"));

    rmDerefDirSVFGEdges(pta);

    rmIncomingEdgeForSUStore(pta);

    DBOUT(DGENERAL, outs() << pasMsg("\tAdd Sink SVFG Nodes\n"));

    AddExtActualParmSVFGNodes(pta->getPTACallGraph());

    if(pta->printStat())
        svfg->performStat();
}
```

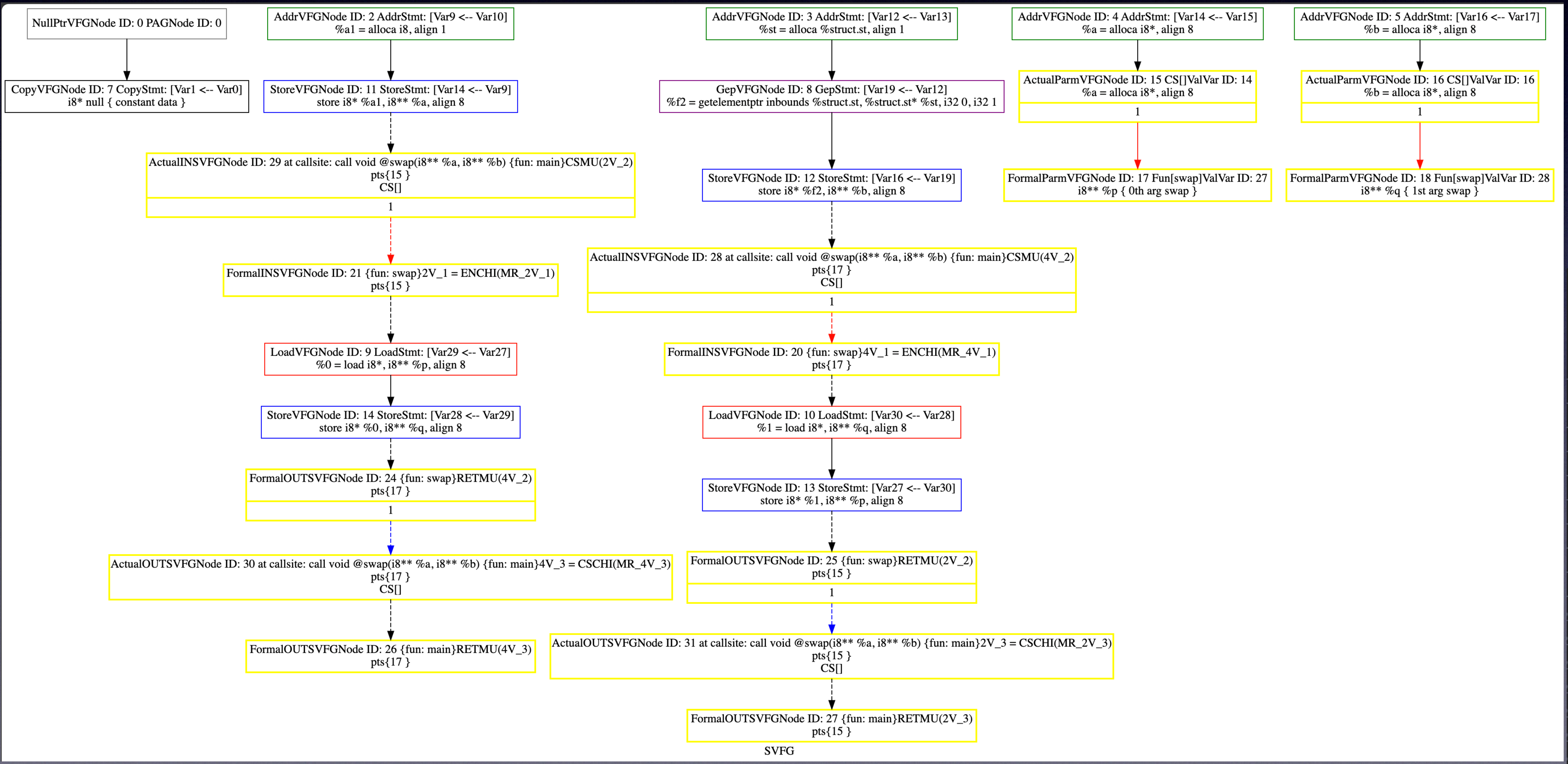
```llvm
%struct.st = type { i8, i8 }

; Function Attrs: noinline nounwind ssp uwtable
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
  %st = alloca %struct.st, align 1
  %a = alloca i8*, align 8
  %b = alloca i8*, align 8
  store i8* %a1, i8** %a, align 8
  %f2 = getelementptr inbounds %struct.st, %struct.st* %st, i32 0, i32 1
  store i8* %f2, i8** %b, align 8
  call void @swap(i8** %a, i8** %b)
  ret i32 0
}

; Function Attrs: noinline nounwind ssp uwtable
define void @swap(i8** %p, i8** %q) #0 {
entry:
  %0 = load i8*, i8** %p, align 8
  %1 = load i8*, i8** %q, align 8
  store i8* %1, i8** %p, align 8
  store i8* %0, i8** %q, align 8
  ret void
```



6.Sparse Value Flow Graph

- 举个例子

```
../cmake-build-debug/bin/saber -dump-vfg svfir.ll -extapi=/Users/liture/CLionProjects/SVF-Dev-Env/github/SVF/svf-llvm/lib/extapi.ll
dot -Tsvg -o svfg_final_1.svg svfg_final.dot
```




```c
typedef struct st { char f1; char f2; } ST;

void swap(char **p, char **q);

int main () {
    char a1;
    ST st;
    char *a = &a1;
    char *b = &(st.f2);
    swap(&a, &b);
}

void swap(char **p, char **q) {
    char *t = *p;
    *p = *q;
    *q = t;
}
```

```llvm
%struct.st = type { i8, i8 }

; Function Attrs: noinline nounwind ssp uwtable
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
  %st = alloca %struct.st, align 1
  %a = alloca i8*, align 8
  %b = alloca i8*, align 8
  store i8* %a1, i8** %a, align 8
  %f2 = getelementptr inbounds %struct.st, %struct.st* %st, i32 0, i32 1
  store i8* %f2, i8** %b, align 8
  call void @swap(i8** %a, i8** %b)
  ret i32 0
}

; Function Attrs: noinline nounwind ssp uwtable
define void @swap(i8** %p, i8** %q) #0 {
entry:
  %0 = load i8*, i8** %p, align 8
  %1 = load i8*, i8** %q, align 8
  store i8* %1, i8** %p, align 8
  store i8* %0, i8** %q, align 8
  ret void
}
```

6.Sparse Value Flow Graph

- 构造算法
  - 1. PreAnalysis (Auxiliary Analysis)
    - flow-context Insensitive pointer analysis
  - 2. MemRegionPatition
    - MRGenerator::generateMRs
      - collectModRefForLoadStore // 对每个函数，收集每个函数Load/Store指令对应指针所指向的内存
      - collectModRefForCall // 沿着调用图的拓扑序列, 自底向上传播非局部变量的MOD/REF (修改/引用)
      - partitionMRs // 划分内存
    - MRGenerator
      - DistinctMRG
      - IntraDisjointMRG
      - InterDisjointMRG
  - 3. buildMemSSA for each function
    - createMUCHI
    - insertPHI
    - rename
  - 4. 连接Def-Use形成SVFG

- CGO’2011 Flow-Sensitive Pointer Analysis for Millions of Lines of Code

6.Sparse Value Flow Graph

```
void foo(){
  ...

 r = *p

  ...
 *q = s



}
```

POINTS-TO::

$p \rightarrow \{ \ x, y \ ,v \ \}$

$q \rightarrow \{ \ x, z \ ,w \ \}$

x, y, z: nonlocal in foo
v, w: locally declared in foo

-

ISSTA2012's talk Static Memory Leak Detection Using Full Sparse Value-Flow Analysis

6.Sparse Value Flow Graph

void foo(){

...

$\mu(v)\ \mu(\boxed{R(x, y)})$

r = *p

...

*q = s

$w = \chi(w)$

$\boxed{R(x, z)} = \chi(\boxed{R(x, z)})$

}

REGION PARTITIONING:

POINTS-TO::

$p \rightarrow \{ \boxed{R(x, y)}, v \}$

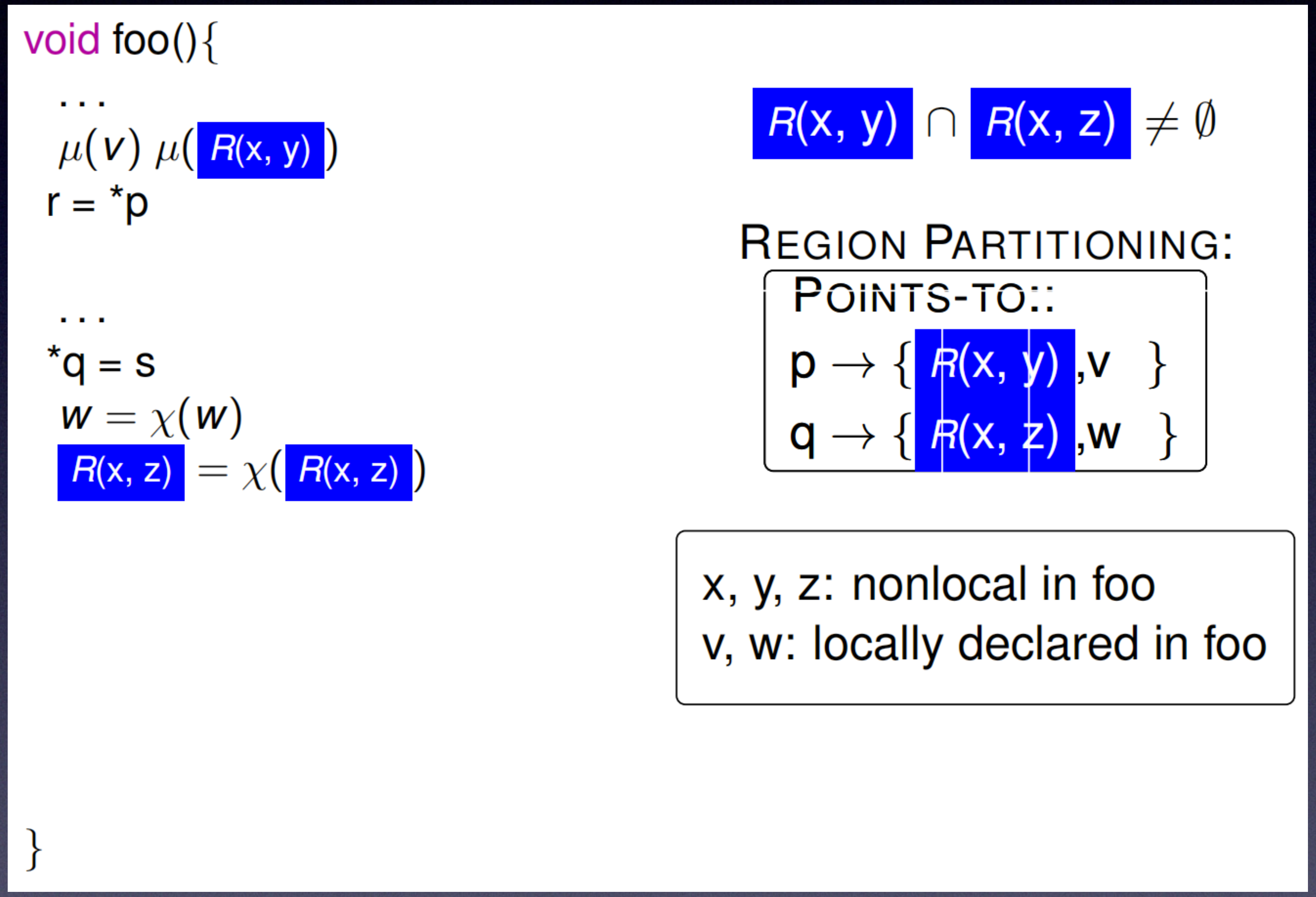
$q \rightarrow \{ \boxed{R(x, z)}, w \}$

x, y, z: nonlocal in foo

v, w: locally declared in foo

-

6.Sparse Value Flow Graph



•

6.Sparse Value Flow Graph

```
void foo(){
  ...
  μ(v) μ(R(x, y))  μ(R(x, z))
 r = *p

  ...
 *q = s
  w = χ(w)
  R(x, z) = χ(R(x, z))
  R(x, y) = χ(R(x, y))

}
```

$R(x, y) \cap R(x, z) \neq \emptyset$

REGION PARTITIONING:

POINTS-TO::

$p \rightarrow \{ R(x, y), v \}$

$q \rightarrow \{ R(x, z), w \}$

x, y, z: nonlocal in foo
v, w: locally declared in foo

•

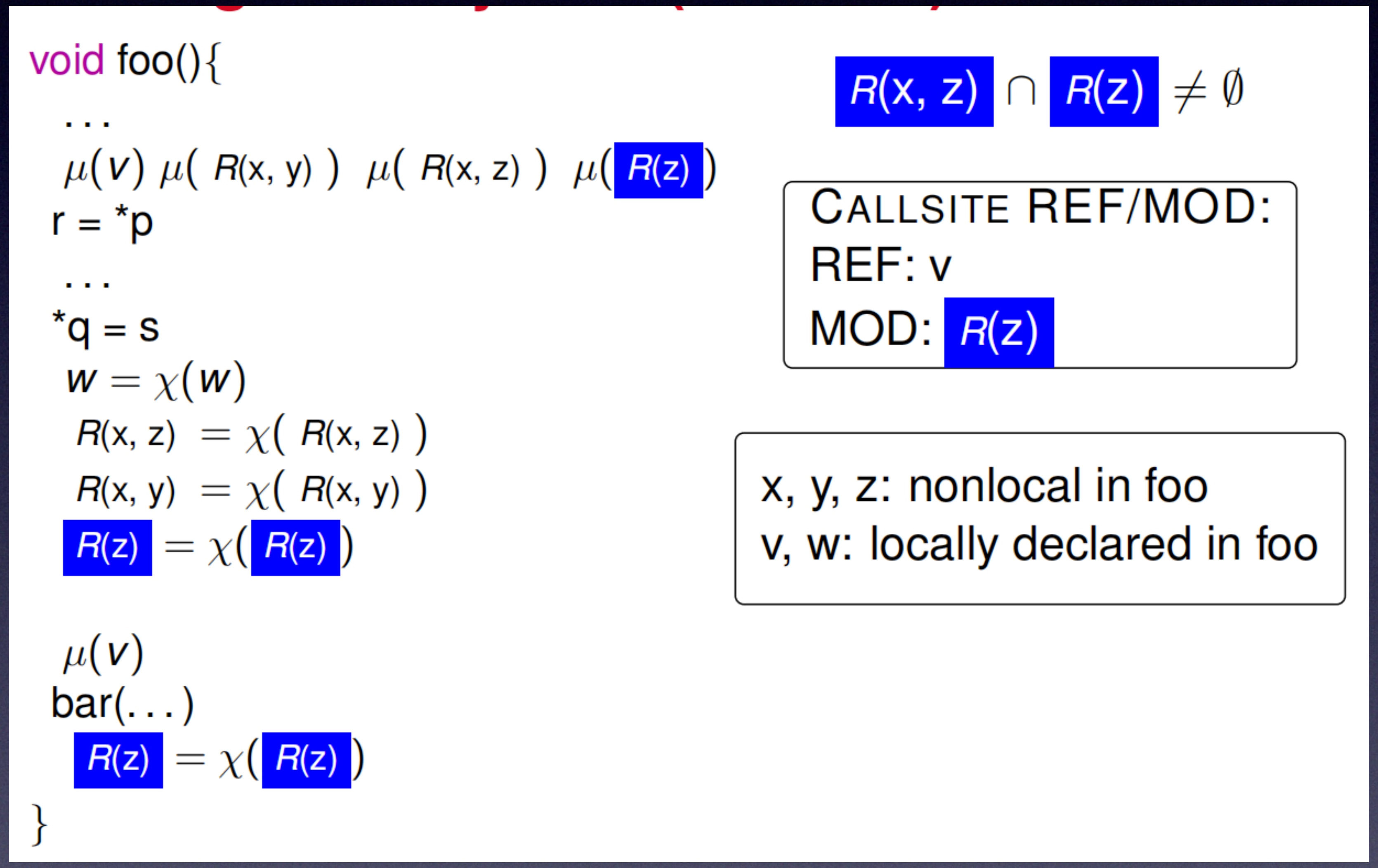
6.Sparse Value Flow Graph

```
void foo(){
  ...
  μ(v) μ( R(x, y) )  μ( R(x, z) )
  r = *p
  ...
  *q = s
  w = χ(w)
  R(x, z) = χ( R(x, z) )
  R(x, y) = χ( R(x, y) )


  μ(v)
  bar(...)
  R(z) = χ( R(z) )
}
```

CALLSITE REF/MOD:
REF: v
MOD: R(z)

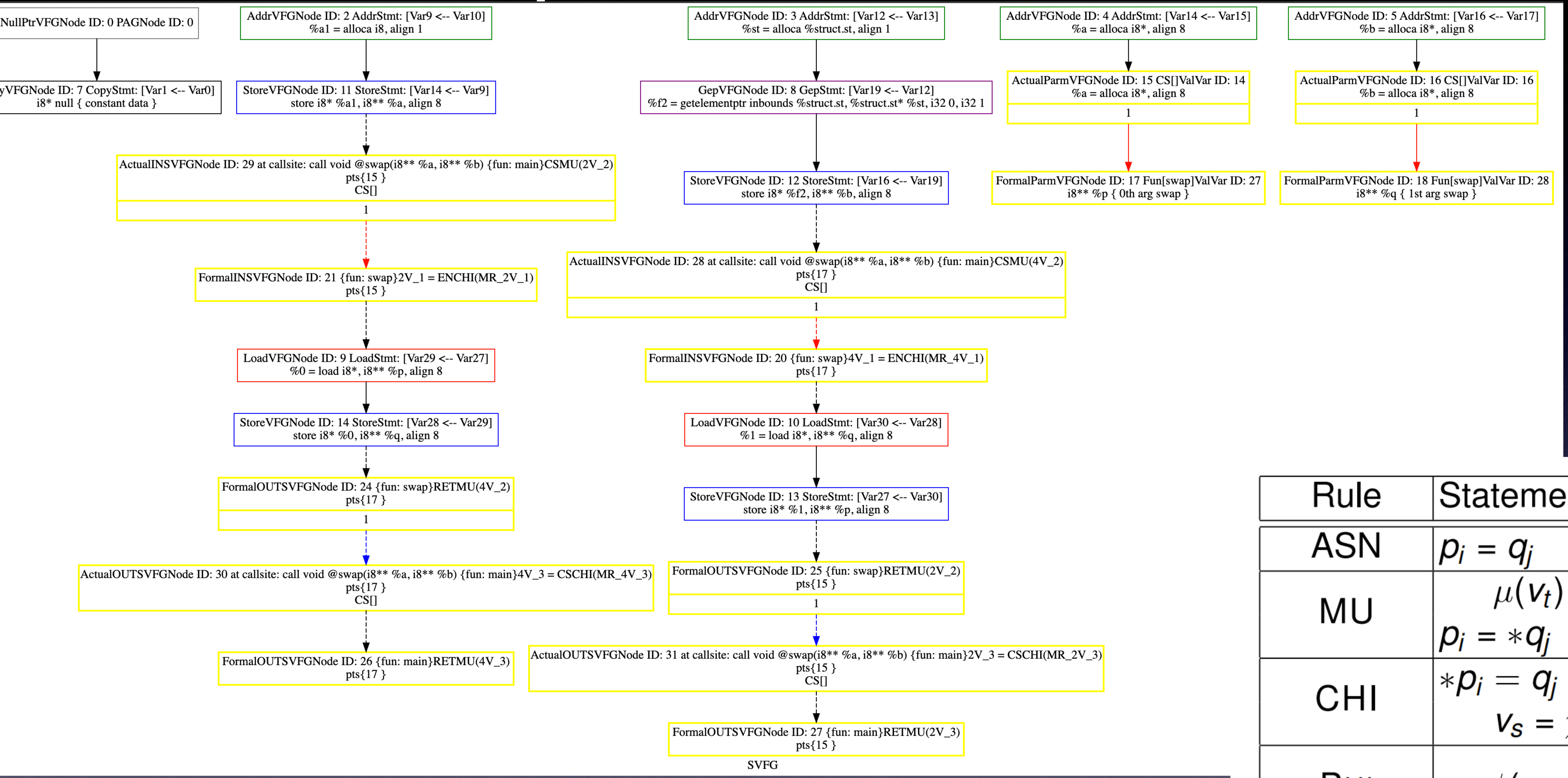
x, y, z: nonlocal in foo
v, w: locally declared in foo

6.Sparse Value Flow Graph



•

6.Sparse Value Flow Graph



Rule	Statement (SSA)	Value-Flow Edges
ASN	$p_i = q_j$	$\widehat{p_i} \leftarrow \widehat{q_j}$
MU	$\mu(v_t)$ $p_i = *q_j$	$\widehat{p_i} \leftarrow \widehat{v_t}$
CHI	$*p_i = q_j$ $v_s = \chi(v_t)$	$\widehat{v_s} \leftarrow \widehat{q_j}$, $\widehat{v_s} \leftarrow \widehat{v_t}$
PHI	$p_i=\phi(q_j,q_k)$	$\widehat{p_i} \leftarrow \widehat{q_j}$, $\widehat{p_i} \leftarrow \widehat{q_k}$
CALL (at a callsite *c* for a callee *g*)	$\mu(v_m)$ r_i=g($\dots, a_k, \dots$) $v_s = \chi(v_t)$	(1)$U_c^g(v_m) \leftarrow \widehat{\mu(v_m)}$, (2) $\widehat{\mu(v_m)} \xleftarrow{(^g_c} \widehat{v_m}$ (3)$FP(a_k) \xleftarrow{(^g_c} \widehat{a_k}$, (4) $\widehat{r_i} \xleftarrow{)^g_c} RET(r_i)$ (5) $\widehat{v_s} \xleftarrow{)^g_c} D_c^g(v_s)$, (6) $\widehat{v_s} \leftarrow \widehat{v_t}$

6.Sparse Value Flow Graph

SVFG.cpp VFG.cpp

```cpp
/*!
 * Constructor
 *  * Build VFG
 * 1) build VFG nodes
 *    statements for top level pointers (PAGEdges)
 * 2) connect VFG edges
 *    between two statements (PAGEdges)
 */
VFG::VFG(PTACallGraph* cg, VFGK k): totalVFGNode(0), callgraph(cg), pag(SVFIR::getPAG()), kind(k)
{

    DBOUT(DGENERAL, outs() << pasMsg("\tCreate VFG Top Level Node\n"));
    addVFGNodes();

    DBOUT(DGENERAL, outs() << pasMsg("\tCreate SVFG Direct Edge\n"));
    connectDirectVFGEdges();
}
```

SVFG.cpp VFG.cpp

```cpp
/*!
 * Build SVFG
 * 1) build SVFG nodes
 *    a) statements for top level pointers (PAGEdges)
 *    b) operators of address-taken variables (MSSAPHI and MSSACHI)
 * 2) connect SVFG edges
 *    a) between two statements (PAGEdges)
 *    b) between two memory SSA operators (MSSAPHI MSSAMU and MSSACHI)
 */
void SVFG::buildSVFG()
{
    DBOUT(DGENERAL, outs() << pasMsg("Build Sparse Value-Flow Graph \n"));

    stat->startClk();
    if (!Options::ReadSVFG().empty())
    {
        readFile(Options::ReadSVFG());
    }
    else
    {
        DBOUT(DGENERAL, outs() << pasMsg("\tCreate SVFG Addr-taken Node\n"));
        stat->ATVFNodeStart();
        addSVFGNodesForAddrTakenVars();
        stat->ATVFNodeEnd();
        DBOUT(DGENERAL, outs() << pasMsg("\tCreate SVFG Indirect Edge\n"));
        stat->indVFEdgeStart();
        connectIndirectSVFGEdges();
        stat->indVFEdgeEnd();
        if (!Options::WriteSVFG().empty())
            writeToFile(Options::WriteSVFG());
    }
}
```

7. Control Dependence Graph

- class CDG : GenericGraph<CDGNode, CDGEdge>
    - CDGNode *getCDGNode(u32_t cfgNodeId)
- class CDGNode : GenericNode<CDGNode, CDGEdge>
- class CDGEdge : GenericEdge<CDGNode>
    - set<pair<SVFVaValue*, u32_t>> getBranchConditions()

7. Control Dependence Graph

my_tool.cpp × cdg.c ExtAPI.cpp

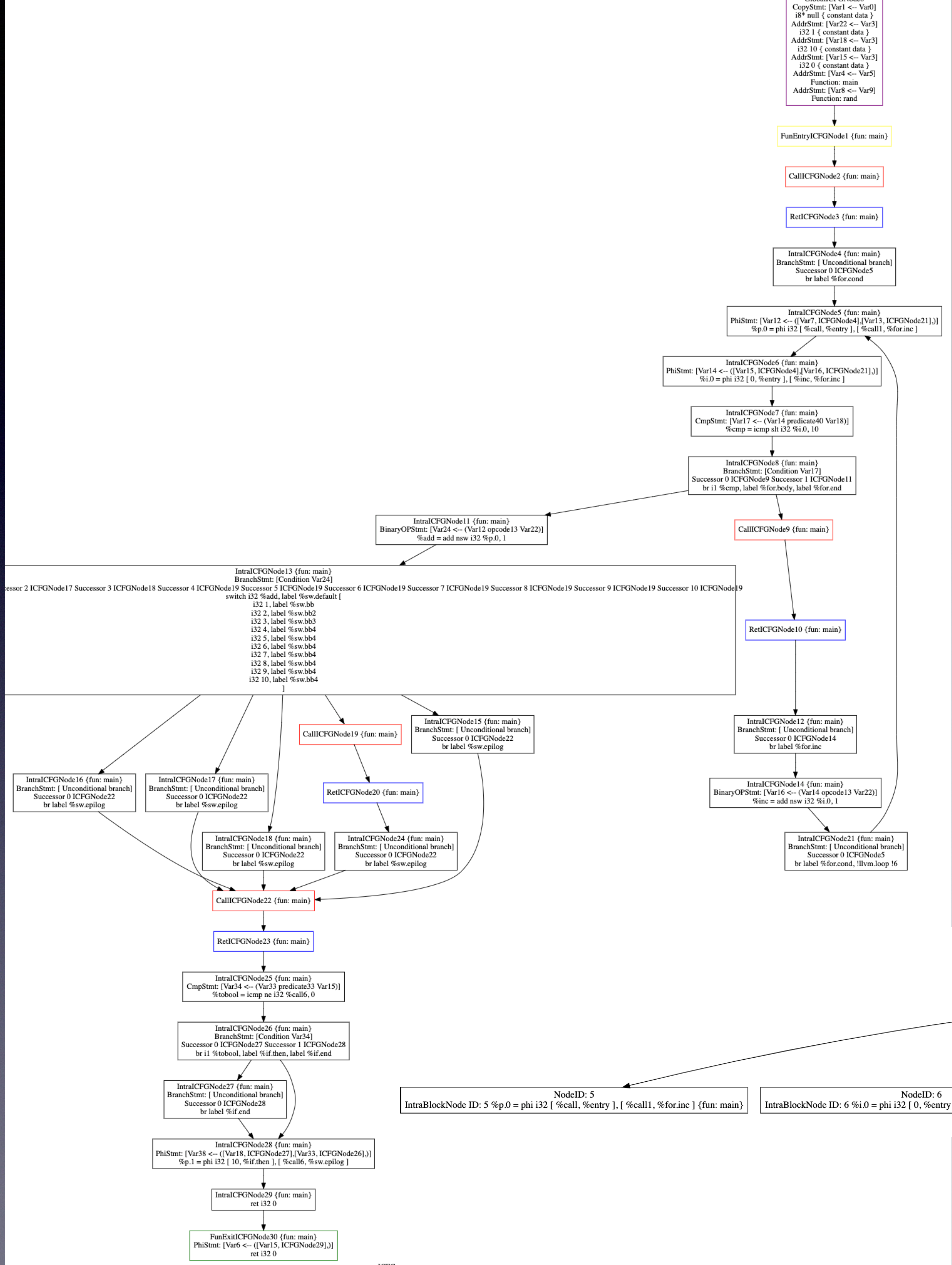
```cpp
//
// Created by Liture on 2023/10/10.
//

#include "SVF-LLVM/SVFIRBuilder.h"
#include "Util/CDGBuilder.h"
#include "Graphs/CDG.h"
#include "Util/ExtAPI.h"

using namespace std;
using namespace SVF;

int main(int argc, char ** argv)
{
    ExtAPI::setExtBcPath("/Users/liture/CLionProjects/SVF-Dev-Env/github/SVF/svf-llvm/lib/extapi.ll");
    SVFModule* svfModule = LLVMModuleSet::buildSVFModule({"/Users/liture/CLionProjects/SVF-Dev-Env/github/SVF/show-testcases/cdg.ll.ll"});

    SVFIRBuilder builder(svfModule);
    builder.build();

    CDGBuilder cdgBuilder;
    cdgBuilder.build();

    CDG *cdg = CDG::getCDG();
    cdg->dump("cdg");

    SVF::LLVMModuleSet::releaseLLVMModuleSet();
    llvm::llvm_shutdown();
    return 0;
}
```

7. Control Dependence Graph

```
int rand();

int main()
{
    // round 1
    int p = rand();
    for (int i = 0 ; i < 10 ; i++)
    {
        p = rand();
    }
    p = p + 1;

    // round 2
    switch (p) {
        case 1: p = 1; break;
        case 2: p = 2; break;
        case 3: p = 3; break;
        case 4:
        case 5:
        case 6:
        case 7:
        case 8:
        case 9:
        case 10:
                p = rand(); break;
        default:
                p = 11;
    }

    // round 3
    p = rand();
    if (p)
    {
        p = 10;
    }
    int k = p;
}
```

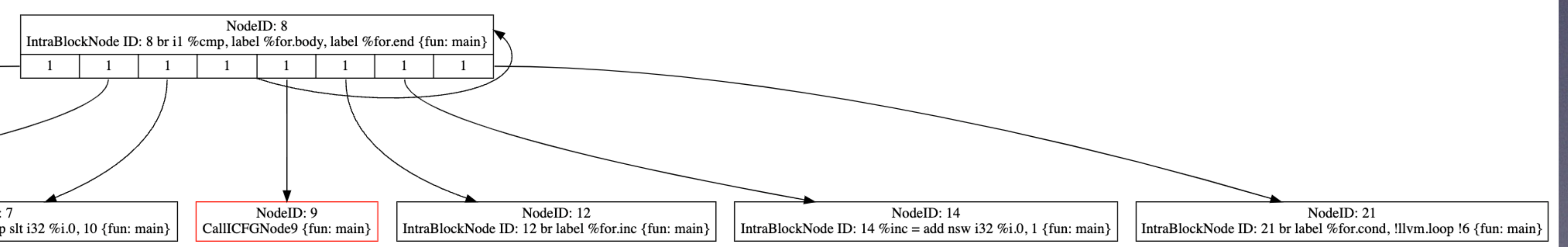
•

ence Graph



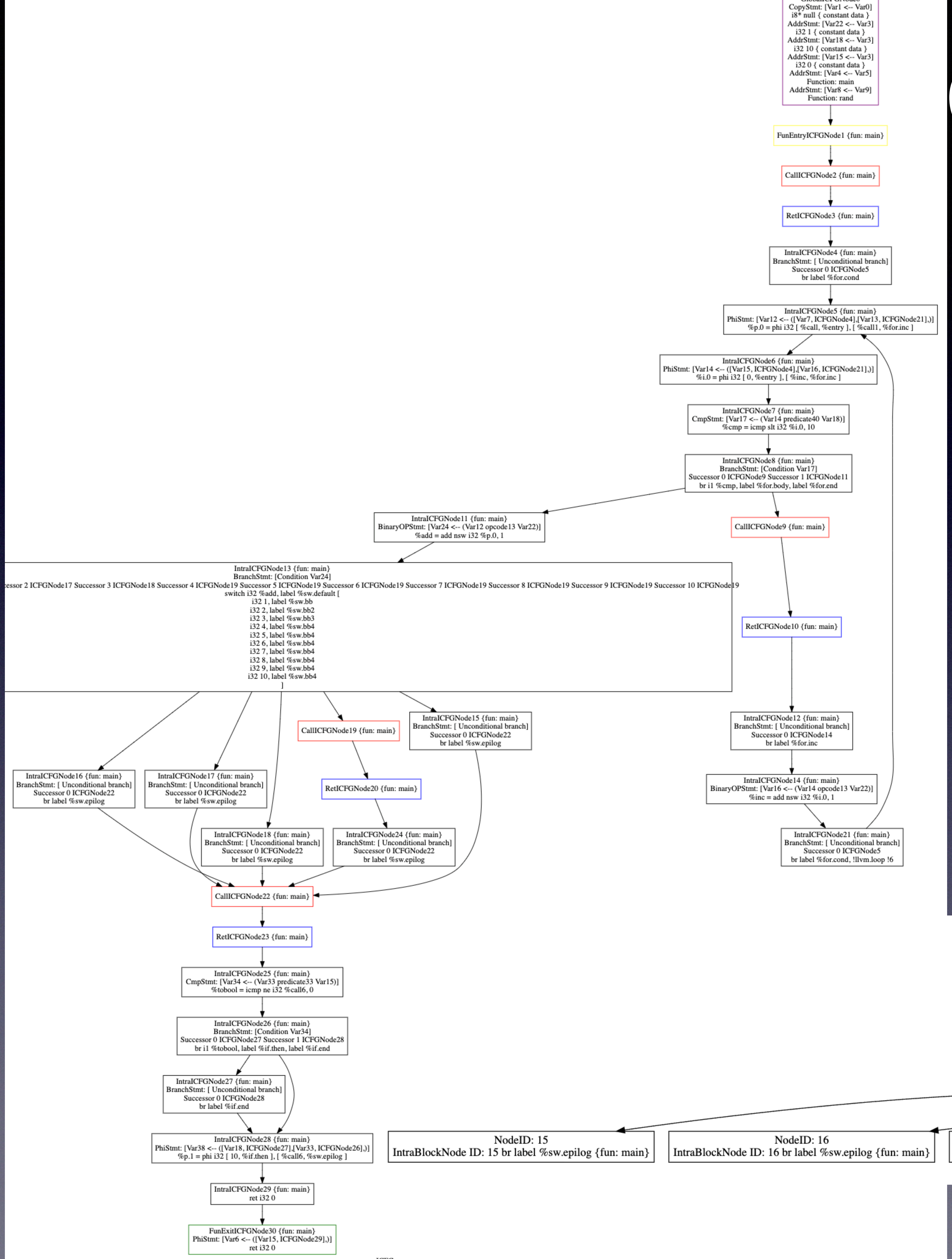
ICFG

```
int rand();

int main()
{
    // round 1
    int p = rand();
    for (int i = 0 ; i < 10 ; i++)
    {
        p = rand();
    }
    p = p + 1;

    // round 2
    switch (p) {
        case 1: p = 1; break;
        case 2: p = 2; break;
        case 3: p = 3; break;
        case 4:
        case 5:
        case 6:
        case 7:
        case 8:
        case 9:
        case 10:
            p = rand(); break;
        default:
            p = 11;
    }

    // round 3
    p = rand();
    if (p)
    {
        p = 10;
    }
    int k = p;
}
```



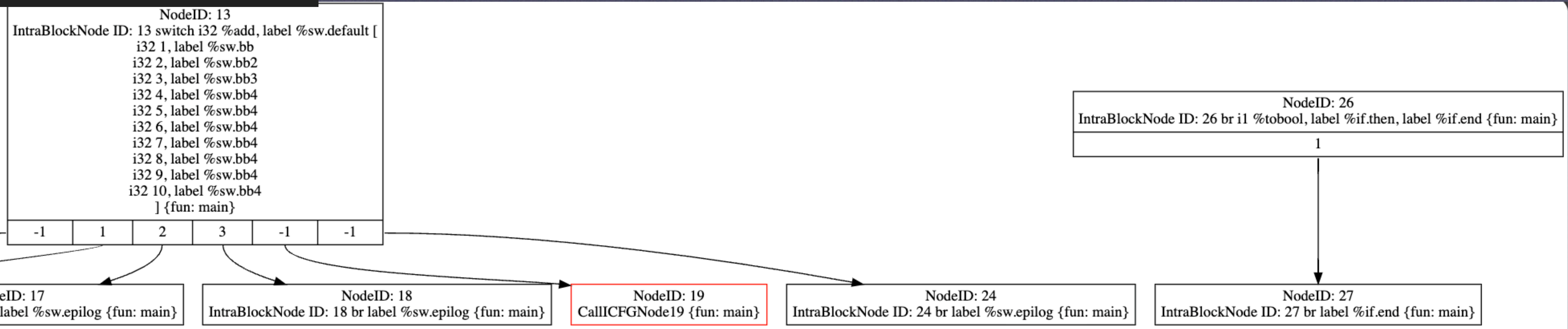
Control Dependence Graph

ence Graph



ICFG

```c
int rand();

int main()
{
    // round 1
    int p = rand();
    for (int i = 0 ; i < 10 ; i++)
    {
        p = rand();
    }
    p = p + 1;

    // round 2
    switch (p) {
        case 1: p = 1; break;
        case 2: p = 2; break;
        case 3: p = 3; break;
        case 4:
        case 5:
        case 6:
        case 7:
        case 8:
        case 9:
        case 10:
            p = rand(); break;
        default:
            p = 11;
    }

    // round 3
    p = rand();
    if (p)
    {
        p = 10;
    }
    int k = p;
}
```



8. Program Dependence Graph

- 程序依赖图 = 数据依赖子图 + 控制依赖子图
  - 函数级别
    - VFG : 数据依赖图
    - CDG : 控制依赖图
- 系统依赖图 (SDG, System Dependence Graph)
  - 过程间数据依赖子图 + 过程间控制依赖子图
  - 跨函数数据依赖图 (SVFG)
    - ActualParam -> FormalParam
    - ActualIN -> FormalIN
    - FormalOut -> ActualOUT
    - FormalRet -> ActualRet
  - 跨函数控制依赖图
    - 每个函数引入一个EntryNode, 控制当前函数顶层没有控制依赖的节点
    - 每个CallNode控制入参数节点(ActualParam, ActualIN), 返回值节点(FormalParam, FormalRet)

8. Program Depend

- 程序依赖图 = 数据依赖子图 + 控制依赖子图
  - 函数级别
    - VFG : 数据依赖图
    - CDG : 控制依赖图
- 系统依赖图 (SDG, System Dependence Graph)
  - 过程间数据依赖子图 + 过程间控制依赖子图
  - 跨函数数据依赖图 (SVFG)
    - ActualParam -> FormalParam
    - ActualIN -> FormalIN
    - FormalOut -> ActualOUT
    - FormalRet -> ActualRet
  - 跨函数控制依赖图
    - 每个函数引入一个EntryNode, 控制当前函数顶层没有控制依赖的节点
    - 每个CallNode控制入参数节点(ActualParam, ActualIN), 返回值节点(FormalParam, FormalRet)

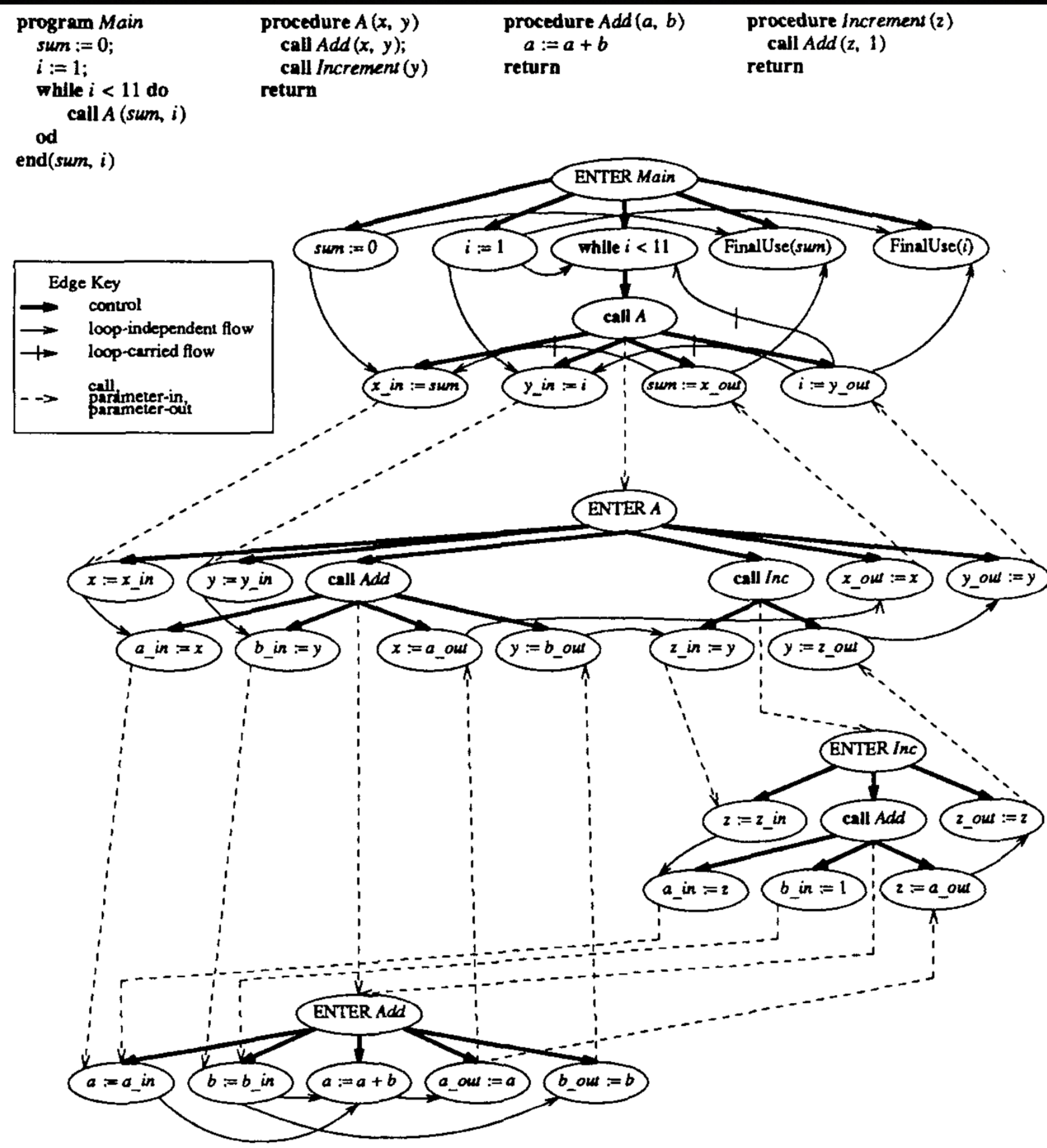


Fig. 4. Example system and corresponding program and procedure dependence graphs connected with parameter-in, parameter-out, and call edges. Edges representing control dependences are shown (unlabeled) in boldface; edges representing intraprocedural flow dependences are shown using arcs; parameter-in edges, parameter-out edges, and call edges are shown using dashed lines.

PLDI'1988 Interprocedural Slicing Using Dependence Graphs

9. SVF开发

- PLDI’2007 Practical Memory Leak Detection Using Guarded Value-Flow Analysis
  - FastCheck的Related Work
    - PLDI’1995 Efficient building and placing of gating functions
    - FSE’2003 Tracking pointers with path and context sensitivity for bug detection in C programs
    - TOSEM’2006 Efficient path conditions in dependence graphs for software safety analysis

```
p1 = 1
if (c) {
  p2 = 2
}
p3 = phi(p1, p2)
```

→

```
p1 = 1
if (c) {
  p2 = 2
}
p3 = γ(c, p2, p1)
```

The CQual system [8] is a system that infers type qualifiers in C programs using a set-constraint formulation. The system can be used to perform tainting analysis, i.e., to determine that a value from a tainted source (such as reading data from the network) never flows into a sink that must not be tainted (such as a critical kernel structure). Livshits and Lam [13] propose a similar tainting analysis, but using an augmented SSA form called IPSSA. Both analyses are context-sensitive; in addition, the latter analysis uses a guarded SSA form [19] that annotates value-flow edges at φ-nodes with branch conditions. The tainting analysis problem is a source-not-sink problem, aiming at checking that source values never flow into a sink. This is a fundamentally different than the source-sink problem discussed in this paper, and none of the above approaches can be used for the memory leak problem. This is because their value-flow representation is not powerful enough to determine that a source value flows into the sink on all paths. Although in the system of Livshits and Lam, value-flow edges of φ-nodes are tagged with branch conditions, the flows via assignments are not, therefore their analysis cannot solve source-sink problems. For instance, the examples from Section 3 do not contain φ-nodes from the source allocations to sink frees, hence none of these value-flows would be guarded.

Snelting et al. [17] present a tainting analysis based on value-flows using a program dependence graph (PDG) structure. They also annotate value flows in the dependence graph using path conditions. However, they use path conditions to improve the precision for a source-no-sink problem. In contrast, we use path conditions to solve source-sink problems such as memory leak detection.

8. Conclusions

We have presented a new analysis for detecting memory leaks. The analysis uses a sparse representation of the program in the form of a value-flow graph. The analysis reasons about program behavior on all paths by computing guards for the relevant value-flow edges. The approach makes the analysis efficient, and allows it to generate concise, easy-to-understand error messages.

References

Flow-insensitive type qualifiers. *ACM Transactions on Programming Languages and Systems*, 28(6):1035–1087, November 2006.

[9] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software — Practice and Experience*, 30(11):1203–1233, 2000.

[10] Brian Hackett and Radu Rugina. Shape analysis with tracked locations. In *Proceedings of the ACM Symposium on the Principles of Programming Languages*, Long Beach, CA, January 2005.

[11] David L. Heine and Monica S. Lam. A practical flow-sensitive and context-sensitive C and C++ memory leak detector. In *Proceedings of the ACM Conference on Program Language Design and Implementation*, San Diego, CA, June 2003.

[12] David L. Heine and Monica S. Lam. Static detection of leaks in polymorphic containers. In *Proceeding of the International Conference on Software Engineering*, Shanghai, China, May 2006.

[13] V. Benjamin Livshits and Monica S. Lam. Tracking pointers with path and context sensitivity for bug detection in C programs. In *ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Helsinki, Finland, September 2003.

[14] Maksim Orlovich and Radu Rugina. Memory leak analysis by contradition. In *Proceedings of the International Static Analysis Symposium*, Seoul, Korea, August 2006.

[15] Thomas Reps, Susan Horowitz, and Mooly Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of the ACM Symposium on the Principles of Programming Languages*, San Francisco, CA, January 1995.

[16] Radu Rugina, Maksim Orlovich, and Xin Zheng. Crystal: A program analysis system for C. URL: http://www.cs.cornell.edu/projects/crystal.

[17] Gregor Snelting, Torsten Robschink, and Jens Krinke. Efficient path conditions in dependence graphs for software safety analysis. *ACM Transactions on Software Engineering and Methodology*, 15(4):410–457, October 2006.

[18] Bjarne Steensgaard. Points-to analysis in almost linear time. In *Proceedings of the ACM Symposium on the Principles of Programming Languages*, St. Petersburg Beach, FL, January 1996.

[19] Peng Tu and David Padua. Efficient building and placing of gating functions. In *Proceedings of the ACM Conference on Program Language Design and Implementation*, La Jolla, CA, June 1995.

-

9. SVF开发

- PLDI’2007 Practical Memory Leak Detection Using Guarded Value-Flow Analysis
    - FastCheck的Related Work
        - PLDI’1995 Efficient building and placing of gating functions
        - FSE’2003 Tracking pointers with path and context sensitivity for bug detection in C programs
        - TOSEM’2006 Efficient path conditions in dependence graphs for software safety analysis
- SVF目前只能做类污点分析问题，对数值相关的问题，支持的不够好；如非外部输入导致的数组越界，除零
    - 相关工作 Sparrow: Sparse Abstract Interpretation
        - PLDI’2012 Design and Implementation of Sparse Global Analysis for C-like Languages
- 目前的分析全是全程序级别的，开销问题
    - PreAnalysis全程序Andersen指针分析
        - 预执行一个轻量级Unsound的MOD/REF分析，暴露出每个函数的副作用 (读/写 非局部内存)
    - 针对分析客户端的问题，懒构造依赖图，而非构造系统级别(全程序)的依赖图
        - 遍历依赖图的时候动态生成函数级别的PDG, CDG
        - 遍历路径时，只为相关的函数构造依赖图，生成的依赖图是函数级别的而非全程序级别的
- 路径敏感分析：目前SVF使用Z3进行SAT求解，考虑将SVFG转换为bitvector算数公式 (SMT公式)
- 求解任意状态机问题 (TypeState Problems)
    - Static Single Information, Static Single Use
- Systematic Program Analysis
    - 外部存储：磁盘, 数据库
    - 并行分析

10. 参考资料

- PLDI 1988 Interprocedural Slicing Using Dependence Graphs
- PLDI 1995 Efficient building and placing of gating functions
- FSE 2003 Tracking pointers with path and context sensitivity for bug detection in C programs
- TOSEM 2006 Efficient path conditions in dependence graphs for software safety analysis
- TOPLAS 2007 Efficient Field-Sensitive Pointer Analysis of C
- PLDI 2007 Practical Memory Leak Detection Using Guarded Value-Flow Analysis
- CGO 2011 Flow-Sensitive Pointer Analysis for Millions of Lines of Code
- ISSTA 2012 Static Memory Leak Detection Using Full-Sparse Value-Flow Analysis

下期预告

- ESP - Property Simulation
  - PLDI’2002 Path-Sensitive Program Verification in Polynomial Time
  - 路径敏感数据流分析

微信公众号：程序分析工具开发

+微信进技术交流群: trustxyz

Github: https://github.com/canliture/

Mail: canliture@outlook.com