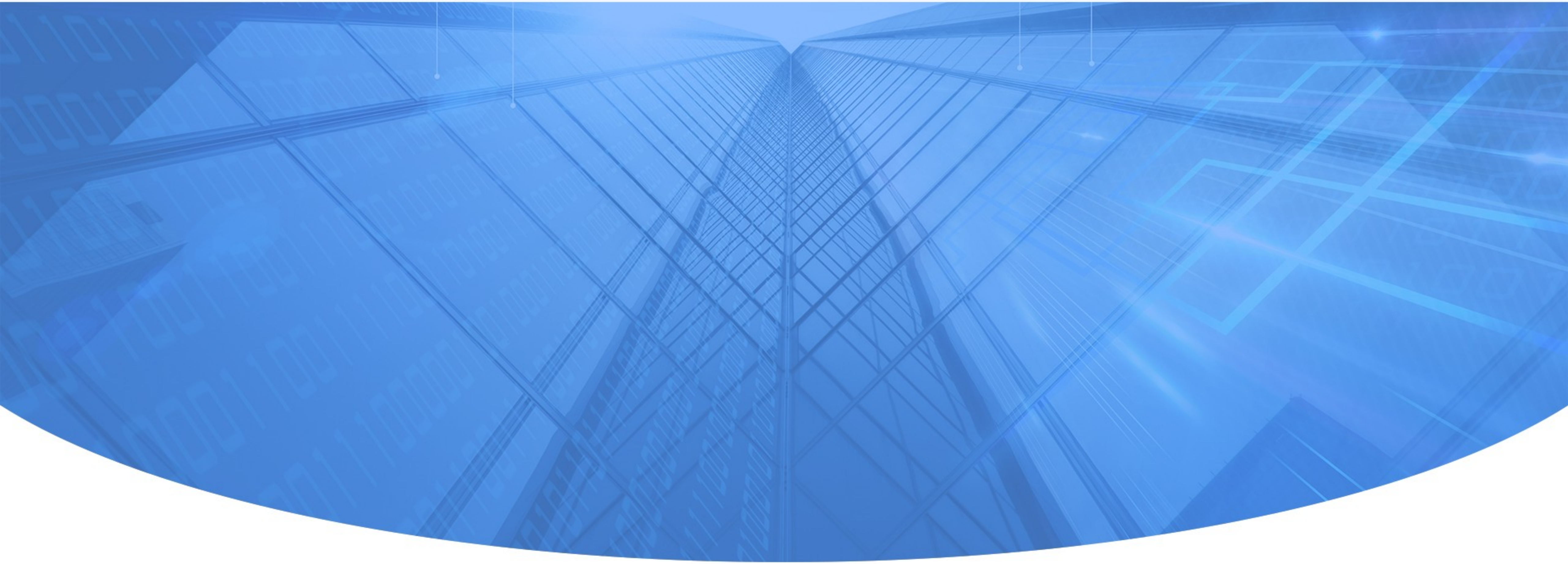
ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences

智能软件研究中心 Intelligent Software Research Center



上手 OLLVM： Porting to LLVM 10

程序语言与编译技术实验室（PLCT） 李威威

2019/11/23

CONTENT

目录

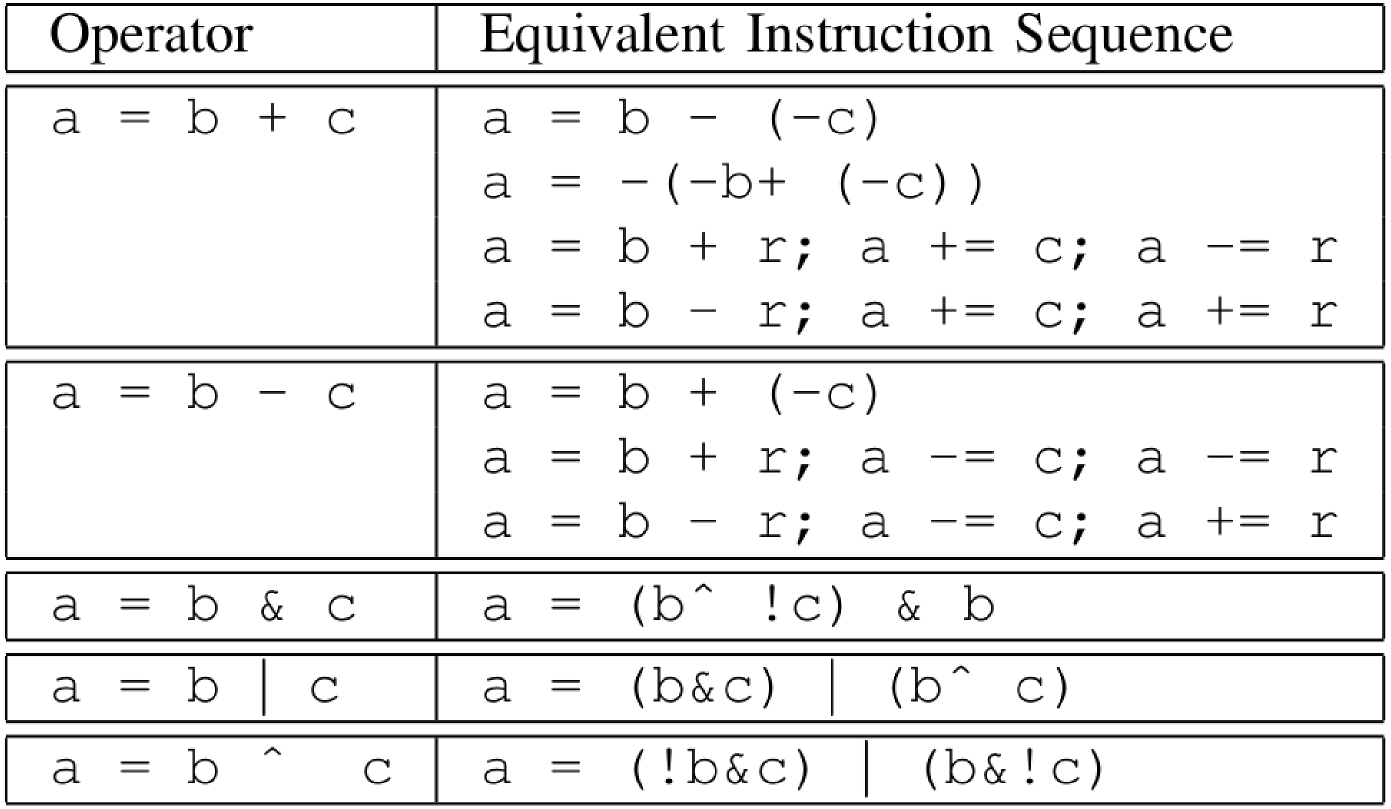
- OLLVM 介绍
- 移植过程中的问题
- 下一步工作

OLLVM 介绍

- OLLVM （Obfuscator-LLVM）是瑞士西北应用科技大学安全实验室于 2010 年 6 月份发起的一个项目
  - 通过随机化的代码混淆以及防篡改，增加对逆向工程的难度，提供更高的软件安全性
    - 目前， OLLVM 仅支持到 LLVM-4.0.1 版本（2017.6.30）
    - github 地址： https://github.com/obfuscator-llvm/obfuscator
      - lib\Transforms\Obfuscation
  - OLLVM 的混淆操作主要针对中间表示 IR 层进行，通过编写 Pass 来混淆 IR
    - Substitution
    - SplitBasicBlock
    - Flattening
    - BogusControlFlow
    - StringObfuscation(孤挺花 , Armariris)

OLLVM 混淆技术介绍——指令替换

- 指令替换 Pass 针对加、减、或、与、异或这五种操作进行替换



Operator	Equivalent Instruction Sequence

a = b + c	a = b - (-c)
	a = -(-b+ (-c))
	a = b + r; a += c; a -= r
	a = b - r; a += c; a += r

a = b - c	a = b + (-c)
	a = b + r; a -= c; a -= r
	a = b - r; a -= c; a += r

a = b & c	a = (b^ !c) & b
a = b | c	a = (b&c) | (b^ c)
a = b ^ c	a = (!b&c) | (b&!c)

OLLVM 混淆技术介绍——基本块分割

- 基本块分割 Pass 通过分割基本块增加控制流的复杂度
  - 仅针对指令数大于 1 且不包含 PHI 节点的基本块进行切割
  - 切割数由 splitNum 参数来指定 [2, 10]

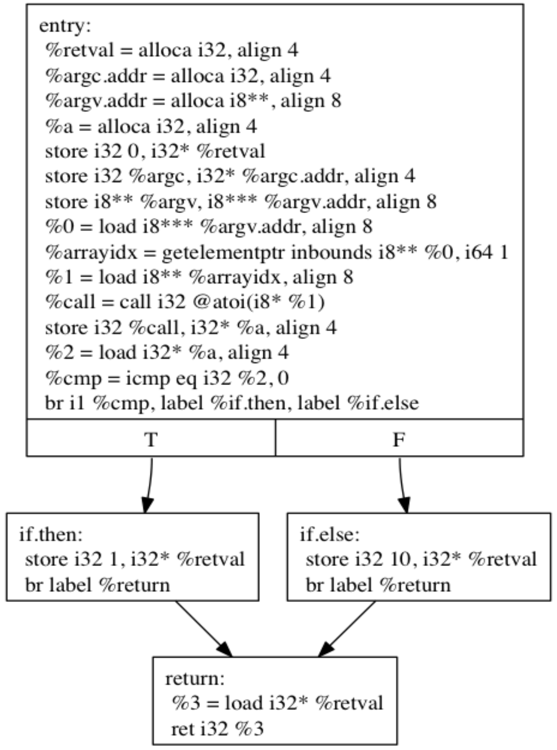
```
entry:
  %m.addr = alloca i32, align 4
  %n.addr = alloca i32, align 4
  store i32 %m, i32* %m.addr, align 4
  store i32 %n, i32* %n.addr, align 4
  %0 = load i32, i32* %m.addr, align 4
  %cmp = icmp eq i32 %0, 0
  br i1 %cmp, label %cond.true, label
%cond.false
```

```
entry:
  %m.addr = alloca i32, align 4
  br label %entry.split

entry.split:                                   ; preds = %entry
  %n.addr = alloca i32, align 4
  store i32 %m, i32* %m.addr, align 4
  br label %entry.split.split

entry.split.split:                             ; preds = %entry.split
  store i32 %n, i32* %n.addr, align 4
  %0 = load i32, i32* %m.addr, align 4
  %cmp = icmp eq i32 %0, 0
  br i1 %cmp, label %cond.true, label %cond.false
```

OLLVM 混淆技术介绍——控制流扁平化

- 控制流扁平化 Pass 实现了控制流的完全扁平化
  - 通过 Loop+Switch 结构来衔接原来的基本块

```
#include <stdlib.h>
int main(int argc, char** argv) {
  int a = atoi(argv[1]);
  if(a == 0)
    return 1;
  else
    return 10;
  return 0;
}
```



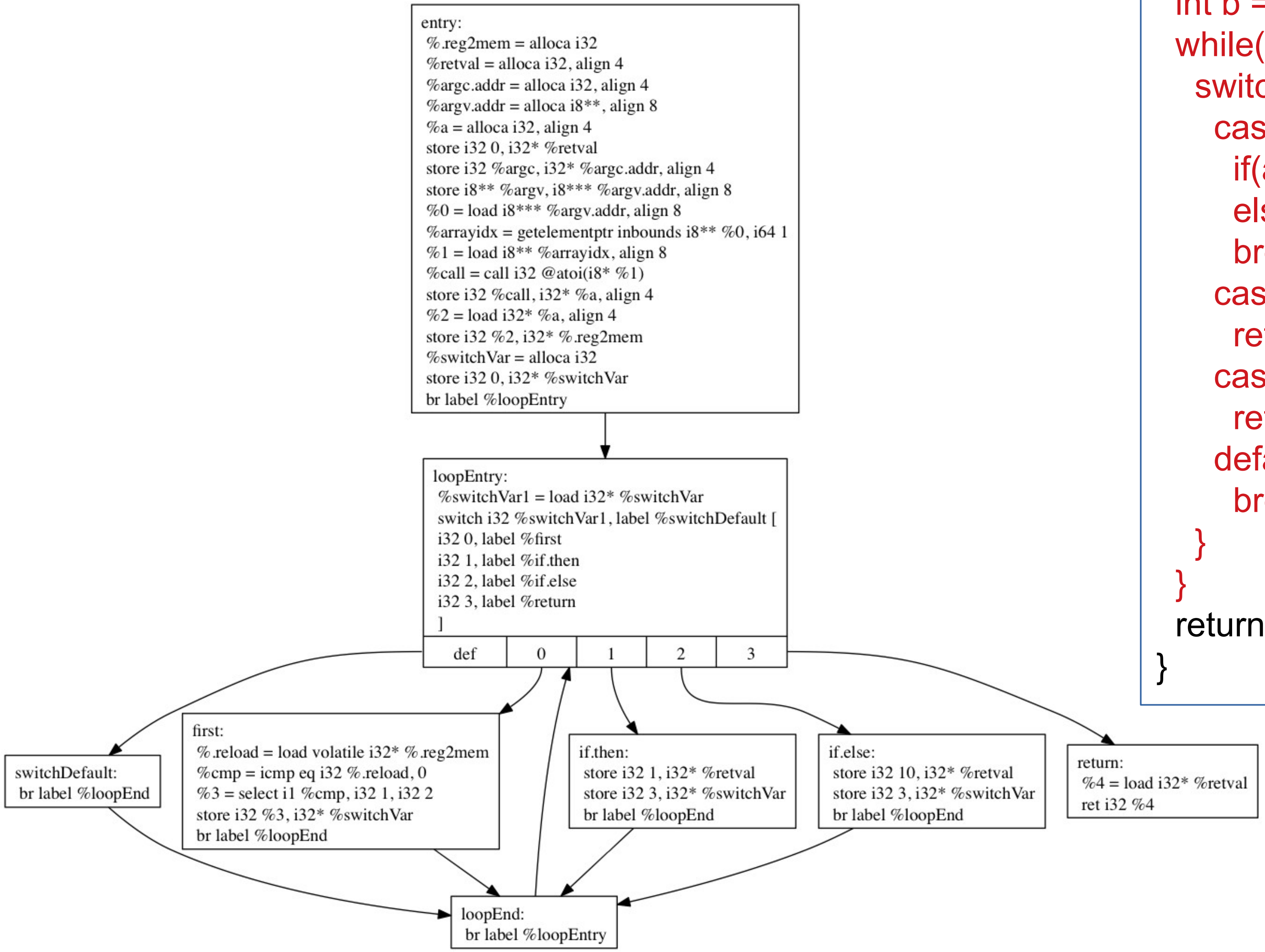
```
entry:
 %retval = alloca i32, align 4
 %argc.addr = alloca i32, align 4
 %argv.addr = alloca i8**, align 8
 %a = alloca i32, align 4
 store i32 0, i32* %retval
 store i32 %argc, i32* %argc.addr, align 4
 store i8** %argv, i8*** %argv.addr, align 8
 %0 = load i8*** %argv.addr, align 8
 %arrayidx = getelementptr inbounds i8** %0, i64 1
 %1 = load i8** %arrayidx, align 8
 %call = call i32 @atoi(i8* %1)
 store i32 %call, i32* %a, align 4
 %2 = load i32* %a, align 4
 %cmp = icmp eq i32 %2, 0
 br i1 %cmp, label %if.then, label %if.else
```

```
if.then:
 store i32 1, i32* %retval
 br label %return
```

```
if.else:
 store i32 10, i32* %retval
 br label %return
```

```
return:
 %3 = load i32* %retval
 ret i32 %3
```

CFG for 'main' function



```
entry:
 %.reg2mem = alloca i32
 %retval = alloca i32, align 4
 %argc.addr = alloca i32, align 4
 %argv.addr = alloca i8**, align 8
 %a = alloca i32, align 4
 store i32 0, i32* %retval
 store i32 %argc, i32* %argc.addr, align 4
 store i8** %argv, i8*** %argv.addr, align 8
 %0 = load i8*** %argv.addr, align 8
 %arrayidx = getelementptr inbounds i8** %0, i64 1
 %1 = load i8** %arrayidx, align 8
 %call = call i32 @atoi(i8* %1)
 store i32 %call, i32* %a, align 4
 %2 = load i32* %a, align 4
 store i32 %2, i32* %.reg2mem
 %switchVar = alloca i32
 store i32 0, i32* %switchVar
 br label %loopEntry
```

```
loopEntry:
 %switchVar1 = load i32* %switchVar
 switch i32 %switchVar1, label %switchDefault [
 i32 0, label %first
 i32 1, label %if.then
 i32 2, label %if.else
 i32 3, label %return
 ]
```

```
switchDefault:
 br label %loopEnd
```

```
first:
 %.reload = load volatile i32* %.reg2mem
 %cmp = icmp eq i32 %.reload, 0
 %3 = select i1 %cmp, i32 1, i32 2
 store i32 %3, i32* %switchVar
 br label %loopEnd
```

```
if.then:
 store i32 1, i32* %retval
 store i32 3, i32* %switchVar
 br label %loopEnd
```

```
if.else:
 store i32 10, i32* %retval
 store i32 3, i32* %switchVar
 br label %loopEnd
```

```
return:
 %4 = load i32* %retval
 ret i32 %4
```

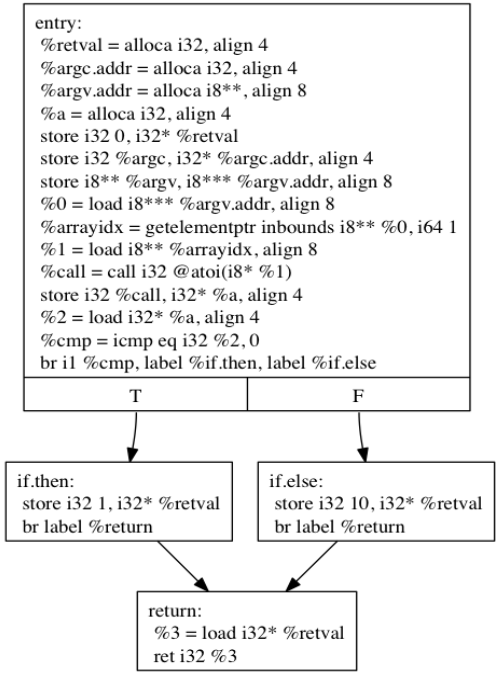
```
loopEnd:
 br label %loopEntry
```

CFG for 'main' function

```
int main(int argc, char** argv) {
  int a = atoi(argv[1]);
  int b = 0;
  while(1) {
    switch(b) {
      case 0:
        if(a == 0)  b = 1;
        else b = 2;
        break;
      case 1:
        return 1;
      case 2:
        return 10;
      default:
        break;
    }
  }
  return 0;
}
```

OLLVM 混淆技术介绍—虚假控制流

- 虚假控制流 Pass 会在原控制流图基础上添加虚假控制流
    - 通过遍历基本块，随机决定是否添加虚假控制流
        - 将原基本块通过不透明谓词（opaque predicate）来模拟假循环



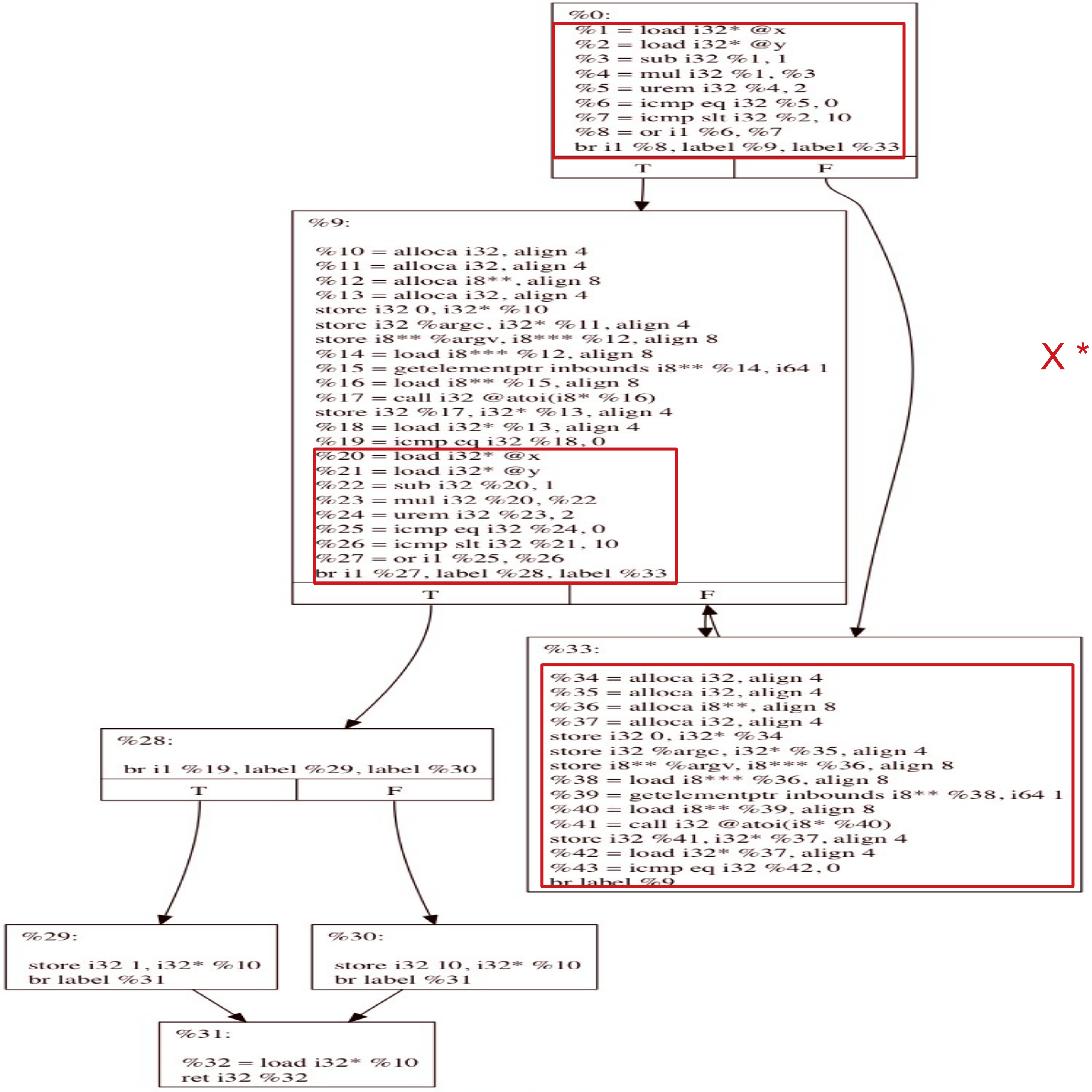
```
entry:
 %retval = alloca i32, align 4
 %argc.addr = alloca i32, align 4
 %argv.addr = alloca i8**, align 8
 %a = alloca i32, align 4
 store i32 0, i32* %retval
 store i32 %argc, i32* %argc.addr, align 4
 store i8** %argv, i8*** %argv.addr, align 8
 %0 = load i8*** %argv.addr, align 8
 %arrayidx = getelementptr inbounds i8** %0, i64 1
 %1 = load i8** %arrayidx, align 8
 %call = call i32 @atoi(i8* %1)
 store i32 %call, i32* %a, align 4
 %2 = load i32* %a, align 4
 %cmp = icmp eq i32 %2, 0
 br i1 %cmp, label %if.then, label %if.else
```

```
if.then:
 store i32 1, i32* %retval
 br label %return
```

```
if.else:
 store i32 10, i32* %retval
 br label %return
```

```
return:
 %3 = load i32* %retval
 ret i32 %3
```

CFG for 'main' function



```
%0:
 %1 = load i32* @x
 %2 = load i32* @y
 %3 = sub i32 %1, 1
 %4 = mul i32 %1, %3
 %5 = urem i32 %4, 2
 %6 = icmp eq i32 %5, 0
 %7 = icmp slt i32 %2, 10
 %8 = or i1 %6, %7
 br i1 %8, label %9, label %33
```

```
%9:

 %10 = alloca i32, align 4
 %11 = alloca i32, align 4
 %12 = alloca i8**, align 8
 %13 = alloca i32, align 4
 store i32 0, i32* %10
 store i32 %argc, i32* %11, align 4
 store i8** %argv, i8*** %12, align 8
 %14 = load i8*** %12, align 8
 %15 = getelementptr inbounds i8** %14, i64 1
 %16 = load i8** %15, align 8
 %17 = call i32 @atoi(i8* %16)
 store i32 %17, i32* %13, align 4
 %18 = load i32* %13, align 4
 %19 = icmp eq i32 %18, 0
 %20 = load i32* @x
 %21 = load i32* @y
 %22 = sub i32 %20, 1
 %23 = mul i32 %20, %22
 %24 = urem i32 %23, 2
 %25 = icmp eq i32 %24, 0
 %26 = icmp slt i32 %21, 10
 %27 = or i1 %25, %26
 br i1 %27, label %28, label %33
```

```
%33:

 %34 = alloca i32, align 4
 %35 = alloca i32, align 4
 %36 = alloca i8**, align 8
 %37 = alloca i32, align 4
 store i32 0, i32* %34
 store i32 %argc, i32* %35, align 4
 store i8** %argv, i8*** %36, align 8
 %38 = load i8*** %36, align 8
 %39 = getelementptr inbounds i8** %38, i64 1
 %40 = load i8** %39, align 8
 %41 = call i32 @atoi(i8* %40)
 store i32 %41, i32* %37, align 4
 %42 = load i32* %37, align 4
 %43 = icmp eq i32 %42, 0
 br label %9
```

```
%28:

 br i1 %19, label %29, label %30
```

```
%29:

 store i32 1, i32* %10
 br label %31
```

```
%30:

 store i32 10, i32* %10
 br label %31
```

```
%31:

 %32 = load i32* %10
 ret i32 %32
```

CFG for 'main' function

X *(X-1) %2 == 0 | y < 10

OLLVM 混淆技术介绍—字符串混淆

- 字符串混淆 Pass 通过加密的方式来混淆静态字符串
  - 简单加密：异或

```
@.str.1 = private global [12 x i8] c"49003|+3.08\\"
@llvm.global_ctors = appending global [1 x { i32, void ()*, i8* }] [{ i32, void ()*, i8* } { i32 65535, void ()*
@.datadiv_decode9516702172628653374, i8* null }]

define void @.datadiv_decode9516702172628653374() {
entry:
  %cmp = icmp eq i32 12, 0
  br i1 %cmp, label %for.end, label %for.body

for.body:                                        ; preds = %for.body, %entry
 ...

for.end:                                        ; preds = %for.body, %entry
  ret void
}
```

移植过程中的问题

- 方法返回类型变换
  - BasicBlock::getTerminator: Instruction -> TerminatorInst
  - Module:: getOrInsertFunction: Constant * -> FunctionCallee

```
Constant* c = mod->getOrInsertFunction(".datadiv_decode" + random_str, FuncTy);
Function* fdecode = cast<Function>(c);
```

```
getOrInsertFunction(".datadiv_decode" + random_str, FuncTy);
Function* fdecode = mod->getFunction(".datadiv_decode" + random_str);
```

移植过程中的问题（续）

- @llvm.global_ctors IR 变换

“The 2-field form of global variables @llvm.global_ctors and @llvm.global_dtors has been deleted. The third field of their element type is now mandatory. Specify i8* null to migrate from the obsoleted 2-field form.”

https://github.com/llvm/llvm-project/blob/release/9.x/llvm/docs/ReleaseNotes.rst

移植过程中的问题（续）

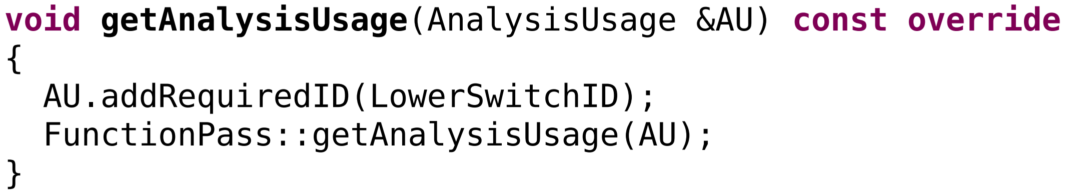
- Pass 间依赖问题
  - Flattening Pass 进行扁平化之前需要将之前的 switch 结构转换成 if 结构（LowerSwitch）

```
FunctionPass *lower = createLowerSwitchPass();
lower->runOnFunction(*f);
```

```
llvm/include/llvm/PassAnalysisSupport.h:221: AnalysisType&
llvm::Pass::getAnalysis() const [with AnalysisType =
llvm::LazyValueInfoWrapperPass]: Assertion `Resolver && "Pass has not
been inserted into a PassManager object!"' failed.
```

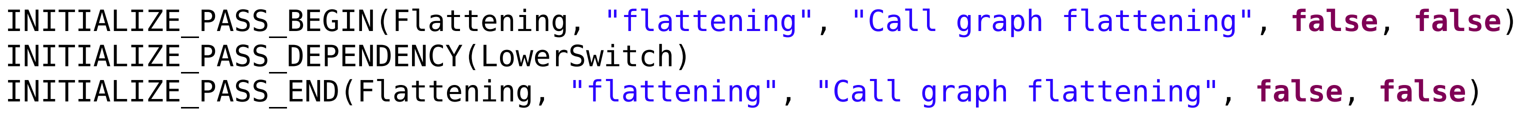
移植过程中的问题（续）

- 当前解决方案
  - 添加 getAnalysisUsage 函数描述依赖关系

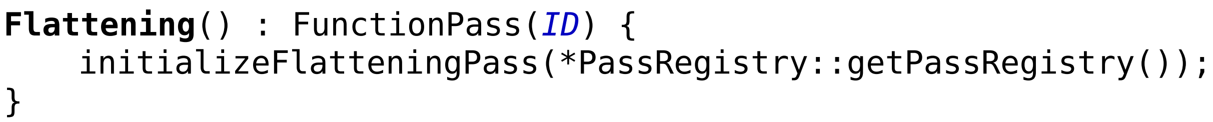


```
void getAnalysisUsage(AnalysisUsage &AU) const override
{
  AU.addRequiredID(LowerSwitchID);
  FunctionPass::getAnalysisUsage(AU);
}
```

  - 删除 create*pass + runOnfunction 代码
  - 初始化注册 pass



```
INITIALIZE_PASS_BEGIN(Flattening, "flattening", "Call graph flattening", false, false)
INITIALIZE_PASS_DEPENDENCY(LowerSwitch)
INITIALIZE_PASS_END(Flattening, "flattening", "Call graph flattening", false, false)
```



```
Flattening() : FunctionPass(ID) {
    initializeFlatteningPass(*PassRegistry::getPassRegistry());
}
```

下一步工作

- 代码防篡改（Code Tamper-Proofing）
  - 结合控制流扁平化技术插入 check() 检查代码完整性（随机选取一段代码求其 CRC 值），并动态更新影响控制流的变量

- 过程合并（Procedures Merging）
  - 将编译单元内的所有函数合并为一个统一的函数

谢谢

欢迎交流合作

2019/11/23