Clang 学习

lgbo

December 27, 2013

Contents

1 内存模型

1.1 参考文献

1. A Memory Model for Static Analysis of C Programs。Apple Inc.

2 ConstrainManager

2.1 变量的约束

以下为“include/clang/StaticAnalyzer/Core/PathSensitive/ProgramState.h”中的原文。

Each ProgramState records constraints on symbolic values. These constraints are managed using the ConstraintManager associated with a ProgramStateManager. As constraints gradually accrue on symbolic values, added constraints may conflict and indicate that a state is infeasible (as no real values could satisfy all the constraints). This is the principal mechanism for modeling path-sensitivity in ExprEngine/ProgramState.

Various ”assume” methods form the interface for adding constraints to symbolic values. A call to ’assume’ indicates an assumption being placed on one or symbolic values. ’assume’ methods take the following inputs:

(1) A ProgramState object representing the current state.

(2) The assumed constraint (which is specific to a given ”assume” method).

(3) A binary value ”Assumption” that indicates whether the constraint is assumed to be true or false.

The output of ”assume*” is a new ProgramState object with the added constraints. If no new state is feasible, NULL is returned.

2.2 实现了的 constraint manager 类

目前在 clang 中找到的关于 constraint manager 的类的定义有。

1. ConstraintManager

2. SimpleConstraintManager
3. RangeConstraintManager

该三个类的继承关系为:
ConstraintManager $\longleftarrow$ SimpleConstraintManager $\longleftarrow$ RangeConstraintManager.

2.2.1 ConstraintManager

ConstraintManager 为所有的 constrain manager 提供一致对外的接口。

比较重要的接口有：assum() and assumDual().

2.2.2 SimpleConstraintManager

2.2.3 RangeConstraitManager

原理？

2.2.4 clang 默认的 constraint manager

ProgramStateManager 中包含了一个 ConstraintManager 成员 ConstraintMgr. 在 ProgramStateManager 的构造函数中，调用了一个 ConstraintManagerCreator 类型的函数指针创建了 ConstraintMgr。

```
/*
include/clang/StaticAnalyzer/Core/PathSensitive/ProgramState.h
*/
typedef ConstraintManager* (*ConstraintManagerCreator)(
    ProgramStateManager&,SubEngine*);
```

```
/*
lib/StaticAnalyzer/Core/ProgramState.cpp
*/
ProgramStateManager::ProgramStateManager(ASTContext &Ctx,
                         StoreManagerCreator CreateSMgr,
                         ConstraintManagerCreator CreateCMgr,
                         llvm::BumpPtrAllocator &alloc,
                         SubEngine *SubEng)
  : Eng(SubEng), EnvMgr(alloc), GDMFactory(alloc),
    svalBuilder(createSimpleSValBuilder(alloc, Ctx, *this)),
    CallEventMgr(new CallEventManager(alloc)), Alloc(alloc) {
```

```
  StoreMgr.reset((*CreateSMgr)(*this));
  ConstraintMgr.reset((*CreateCMgr)(*this, SubEng));
}
```

在”lib/StaticAnalyzer/Frontend/AnalysisComsumer.cpp”, 我们发现了一个 ConstraintManagerCreator 类型的变量：CreateConstraintMgr。

```
/*
lib/StaticAnalyzer/Frontend/AnalysisComsumer.cpp
*/
/*
  \brief Stores the declarations from the local translation
     unit.
  Note, we pre-compute the local declarations at parse time as
     an
  optimization to make sure we do not deserialize everything
     from disk.
  The local declaration to all declarations ratio might be very
      small when
  working with a PCH file.
  SetOfDecls LocalTUDecls;
*/
  /*
   Set of PathDiagnosticConsumers.  Owned by AnalysisManager.
  */
  PathDiagnosticConsumers PathConsumers;

  StoreManagerCreator CreateStoreMgr;
  ConstraintManagerCreator CreateConstraintMgr;
```

以下代码出现了对 CreateConstraintMgr 的初始化。

```
/*
lib/StaticAnalyzer/Frontend/AnalysisConsumer.cpp
*/
switch (Opts->AnalysisConstraintsOpt) {
default:
  llvm_unreachable("Unknown␣constraint␣manager.");
#define ANALYSIS_CONSTRAINTS(NAME, CMDFLAG, DESC, CREATEFN)
        \
  case NAME##Model: CreateConstraintMgr = CREATEFN; break;
#include "clang/StaticAnalyzer/Core/Analyses.def"
}
```

在”include/clang/StaticAnalyzer/Core/Analyses.def” 中，有以下定义：

```
#ifndef ANALYSIS_CONSTRAINTS
#define ANALYSIS_CONSTRAINTS(NAME, CMDFLAG, DESC, CREATFN)
#endif

```

```
ANALYSIS_CONSTRAINTS(RangeConstraints, "range", "Use␣constraint
    ␣tracking␣of␣concrete␣value␣ranges",
    CreateRangeConstraintManager)
```

CreateRangeConstraintManager 函数在 RangeConstraintManager.cpp 中定义了。

到此，我们似乎可以确定 clang 目前使用的 constraint manager 是 RangeConstraintManager。且数组越界检查使用的也是默认的 constraint manager。

2.2.5 如何往 constraint manager 中添加约束条件

如前所述，Clang 通过 assume()/asumeDual() 为入口添加新的约束条件。实际上约束条件就是一个与符号绑定的状态。

状态注册。

```
REGISTER_TRAIT_WITH_PROGRAMSTATE(ConstraintRange,
    CLANG_ENTO_PROGRAMSTATE_MAP(SymbolRef,RangeSet))
```

状态读取。

```
/*RangeConstraintManager.cpp*/
ProgramStateRef
RangeConstraintManager::assumeSymEQ(ProgramStateRef St,
    SymbolRef Sym,const llvm::APSInt &Int,const llvm::APSInt &
    Adjustment) {
  /*......*/
  RangeSet New = GetRange(St, Sym).Intersect(getBasicVals(), F,
      AdjInt, AdjInt);
  /*......*/
}


RangeSet
RangeConstraintManager::GetRange(ProgramStateRef state,
    SymbolRef sym) {
  if (ConstraintRangeTy::data_type* V = state->get<
      ConstraintRange>(sym))
    return *V;
  /*......*/
}
```

状态写入。

```
ProgramStateRef
```

```
RangeConstraintManager::assumeSymNE(ProgramStateRef St, SymbolRef Sym,const llvm::APSInt &Int,const llvm::APSInt & Adjustment) {
  /*......*/
  RangeSet New = GetRange(St, Sym).Intersect(getBasicVals(), F, Upper, Lower);
  return New.isEmpty() ? NULL : St->set<ConstraintRange>(Sym, New);
}
```

2.3 接口

```
//ConstraintManger.h
virtual ProgramStateRef assume(ProgramStateRef state, DefinedSVal Cond,bool Assumption) = 0;


/// Returns a pair of states (StTrue, StFalse) where the given condition is
/// assumed to be true or false, respectively.
ProgramStatePair assumeDual ( ProgramStateRef State , DefinedSVal Cond) {
  ProgramStateRef StTrue = assume(State, Cond, true);

  // If StTrue is infeasible, asserting the falseness of Cond is unnecessary
  // because the existing constraints already establish this.
  if (!StTrue) {
#ifndef __OPTIMIZE__
    // This check is expensive and should be disabled even in Release+Asserts
    // builds.
    // FIXME: __OPTIMIZE__ is a GNU extension that Clang implements but MSVC
    // does not. Is there a good equivalent there?
    assert(assume(State, Cond, false) && "System␣is␣over␣constrained.");
#endif
    return ProgramStatePair((ProgramStateRef)NULL, State);
  }

  ProgramStateRef StFalse = assume(State, Cond, false);
  if (!StFalse) {
    // We are careful to return the original state, /not/ StTrue,
    // because we want to avoid having callers generate a new node
    // in the ExplodedGraph.
    return ProgramStatePair(State, (ProgramStateRef)NULL);
  }

  return ProgramStatePair(StTrue, StFalse);
}
```

```
//SimpleConstraintManger.h
ProgramStateRef assume(ProgramStateRef state, DefinedSVal Cond,
    bool Assumption);
ProgramStateRef assume(ProgramStateRef state, NonLoc Cond, bool
     Assumption);
```

3 Checkers

3.1 Divide Zero Checker

方法：约束求解。