Using the clang data-flow framework for null-pointer analysis

Viktor Cseh
cseh.viktor@gmail.com, Github: @Discookie

Eötvös Loránd University, Budapest
Ericsson Hungary
Data-flow primer

• Approximation of the program state at various points
• Basic principles: transfer, merge
• Iterative method - needs to reach a fixpoint to be useful
  • Transfer function needs to be monotone
Clang data-flow framework

- Analysis classes: MAY/MUST
- Clang Static Analyzer is good at MAY-analyses
  - Not suited for MUST-analysis
  - Few standalone data-flow analyses
- New data-flow framework in early 2022
Null-pointer analysis

• Clang Static Analyzer is already good at detecting null-pointer dereferences

• Our goal: Reverse null checker
  • Pointer is checked after it's already dereferenced

```
11 int *ptr = (int*) malloc(sizeof(int));
12 // ...
13 14 +ptr = 10;
15 // ...
16 if (ptr) {
17     *ptr = -20;
18 }
19```
Lattice vs. boolean constraints

• Lattices
  • Operations are fast and well-defined, but stores less information

• Boolean constraints
  • Can store context, but requires a SAT-solver - can be expensive!
    • true, false, 'uncertain' values

• Data-flow framework supports both approaches

\[(\phi_1 \Rightarrow (\phi_{\text{merged}} = \{1\})) \land (\phi_2 \Rightarrow (\phi_{\text{merged}} = \{1, 2\}))\]
Lattice vs. boolean constraints/2

• Flow condition token
  • Precondition to the program’s current state

```c
int *ptr;

if (condition) {
  ptr = nullptr;
}
else {
  ptr = &reference;
}

Flow condition: \( \phi \)
Ptr is null: unknown

{ Flow condition: false \( \land \) (condition == true)
  Ptr is null: true
}
{ Flow condition: false \( \land \) (condition == false)
  Ptr is null: false
}

Flow condition: (\( \phi \) true) \( \lor \) (\( \phi \) false)

Ptr is null: (\( \phi \) true) \( \land \) (\( \phi \) false)
```
Architecture and implementation notes

- **DataflowAnalysis class**
  - Implements transfer, `branchTransfer`, merge
  - `operator*`, `operator->`, and comparisons

- 2 boolean constraints: `is-null`, `is-nonnull`
  - Unknown state stored as 'uncertain'

```c
7    ptr = {};                         
8    ptr->member = 0;                
9    ptr2 = ptr;                     
10   if (ptr){//***}                 
```

```c
11   if (Env.flowConditionImplies(IsNull)){
12      // Can be null
13   }else if (Env.flowConditionImplies(Env.makeNot(IsNull))){
14      // Is definitely not null
15   }else{
16      // Unknown value, could be either
17     }
18 }                                         
19                                           
```
Constraint information and performance

• Various amounts of stored information and performance
• Only emulate lattices, true/false
  • Main bottleneck is number of boolean values
• Encode conditional data
  • The constraint grows very quickly - slows down the solver
• No way to get size of constraint-expression (yet)
Widening

• Ran every time a head node executes twice
• Default: if different, forget all information
  • Loses information, but analysis terminates faster
• First approach: check if the expressions are the same using the SAT solver
  • Involves multiple calls to the solver - each call is slow
  • Can lead to very long analysis times
Widening/2

• Current approach: Check trivial true/false cases, lose information otherwise
  • Terminates slower, but stores more constraint information
• Optimize known true/false constraints

Default approach:
- (a == b) by address
- or delete

First approach:
- (a == b) by address
- or (a == b) by satisfiability
- or delete

Current approach:
- (a == b) by address
- or (a == true) and (b == true)
- or (a == false) and (b == false)
- or delete
Results (on C++ projects)

• **1 report** (% of files analyzed):
  - libwebm (95%)
  - Qtbase (11%)

• **0 reports:**
  - tinyxml2 (100%)
  - xerces (98%)
  - bitcoin (98%)
  - protobuf (45%)
  - contour (99%)

```cpp
// make sure we can decode at least one char
if (state->remainingChars + len < 2) {
  if (len) {
    Q_ASSERT(state->remainingChars == 0 && len == 1);
    state->remainingChars = 1;
    state->state_data[0] = "chars;
  } return out;

bool headerdone = state & state->internalState & HeaderDone;
// pointer value is checked despite dereferencing it earlier
// for more information see the checker documentation.
if (state->flags & QObjectConverter::Flag::ConvertInitialDom)
  headerdone = true;
```
Framework limitations: Solver timeouts

• There is a timeout on data-flow iterations
  • No timeout on SAT solver runtime
  • Creating the constraints is fast, querying the solver is slow

• Constraints get large very quickly
  • Widening and reset on merge helps, but not always
  • Flow condition is kept across all states
Framework limitations: Type modeling

• No C support due to boolean datatype issues
  • Analysis crashes on any condition

• Quick fix: value tracking for integers
• Long-term solution: SMT solver

```c
2    int i = 1;
3    if (i) {
4        // IfStmt
5        // - ImplicitCastExpr 'int' <LValueToRValue>
6        // | - DeclRefExpr 'int' 'i'
7        // missing <IntegralToBoolean> cast in C!
8    }
9
10
11
```
Debug using the framework

• Environment is logged nicely, each value is visible
  • `dataflow-log` and HTML page, good for visualization

• Constraints are difficult to debug
  • No information attached to boolean variables
Future work

• General-purpose pointer nullability checker
• Different types of values - integers, smart pointers, etc.
• Detect and handle assertions

Framework:
• Interprocedural analysis – function summaries
• Z3 solver
• Support for more data types
Thank you!

• Acknowledgements
  • The static analysis team at Ericsson

• Bibliography