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

Viktor Cseh

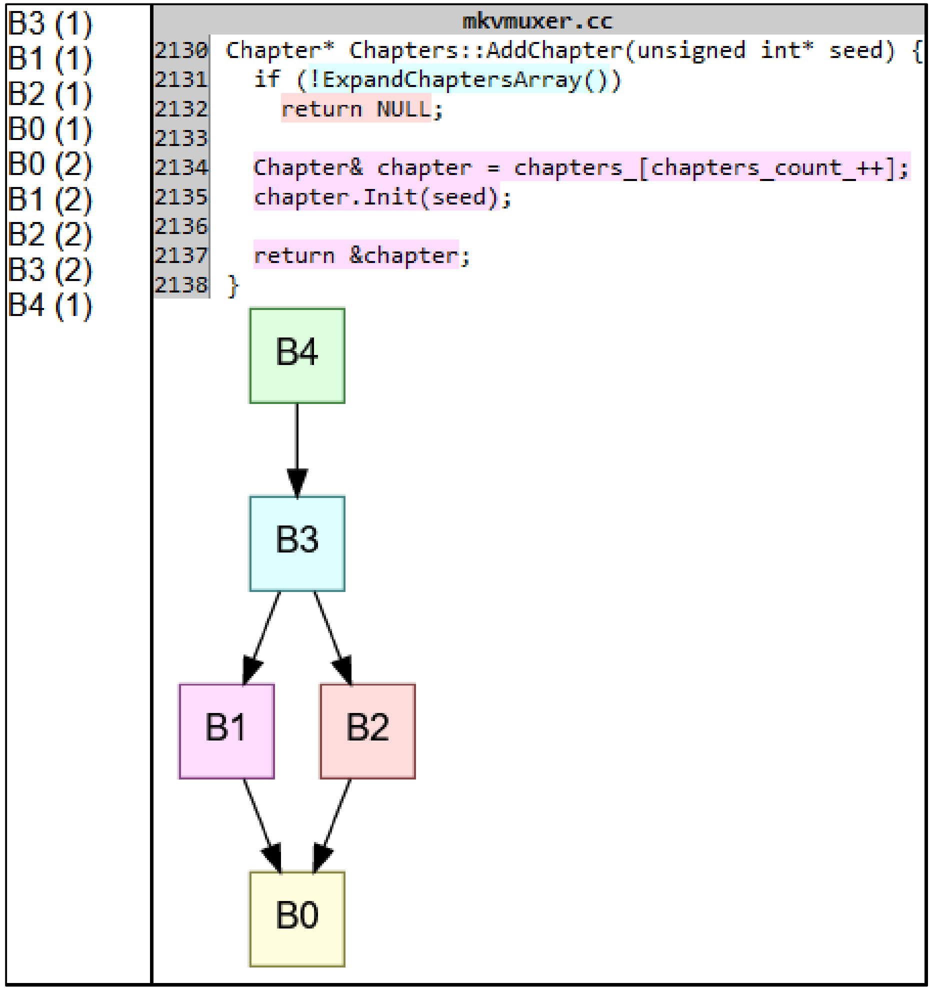
cseh.viktor@gmail.com, Github: @Discookie

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

```
int *ptr = (int *) malloc(sizeof(int));
  3 < 'ptr' initialized here >

// ...

if (ptr) {
    4 < Assuming 'ptr' is null >
    *ptr = 20;
}

// ...

*ptr += 10;
  5 < Dereference of null pointer (loaded
      from variable 'ptr')
      For more information see the
      checker documentation.
```

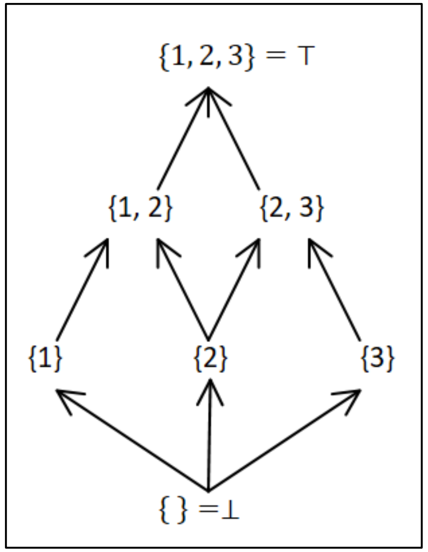
Null-pointer analysis

- Clang Static Analyzer is already good at detecting null-pointer dereferences
- Our goal: Reverse null checker
  - Pointer is checked after it's already dereferenced

```
    int *ptr = (int *) malloc(sizeof(int));

    // ...

    *ptr = 10;

    // ...

    if (ptr) {
        *ptr = 20;
    }

```

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



$$\left(\varphi_1 \Rightarrow \left(\varphi_{\text{merged}} = \{1\}\right)\right) \text{ and } \left(\varphi_2 \Rightarrow \left(\varphi_{\text{merged}} = \{1, 2\}\right)\right)$$

Lattice vs. boolean constraints/2

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

```
    int *ptr;

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



    ptr;
```

```
Flow condition: ϕ
  Ptr is-null: unknown

{ Flow condition ϕtrue - ϕ and (condition == true)
    Ptr is-null: true
}
{ Flow condition ϕfalse - ϕ and (condition == false)
    Ptr is-null: false
}
Flow condition: (ϕtrue) or (ϕfalse)
    - same as ϕ

Ptr is-null: ((ϕtrue) => true) or ((ϕfalse) => false)
```

Architecture and implementation notes

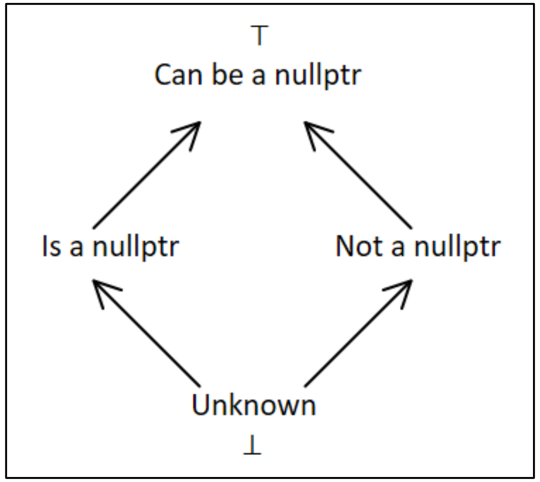
- DataflowAnalysis class
  - Implements transfer, *branchTransfer*, merge
  - operator*, operator->, and comparisons
- 2 boolean constraints: is-null, is-nonnull
  - Unknown state stored as 'uncertain'

```
    *ptr = {};
    ptr->member = 0;

    ptr2 = ptr;

    if (ptr) { /***/ }

```

```

  if (Env.flowConditionImplies(IsNull)) {
    // Can be null
  } else if (Env.flowConditionImplies(Env.makeNot(IsNull))) {
    // Is definitely not null
  } else {
    // Unknown value, could be either
  }

```

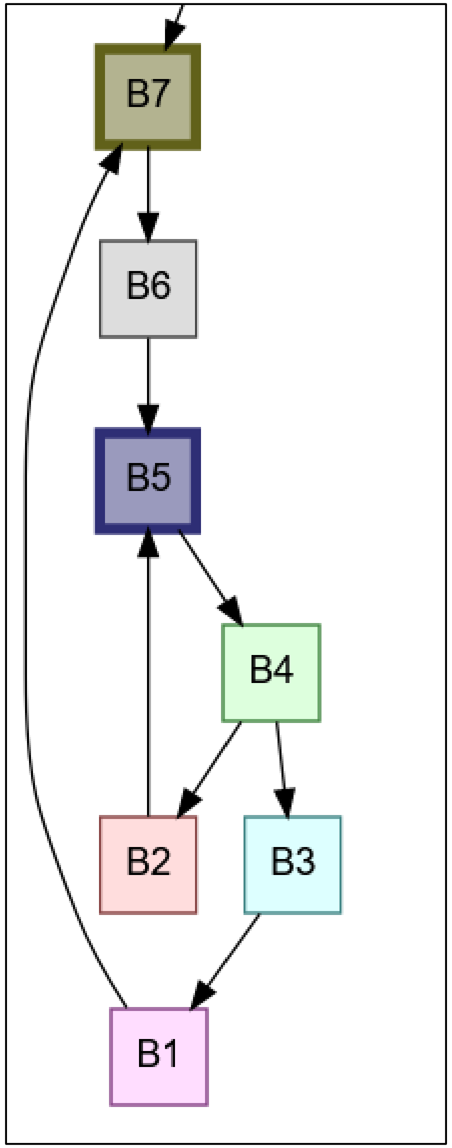
Constraint information and performance

- 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 appproach:
  - (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%)

```
    // 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[Data] = *chars;
        }
        return out;
    }

    bool headerdone = state && state->internalState & HeaderDone;
```

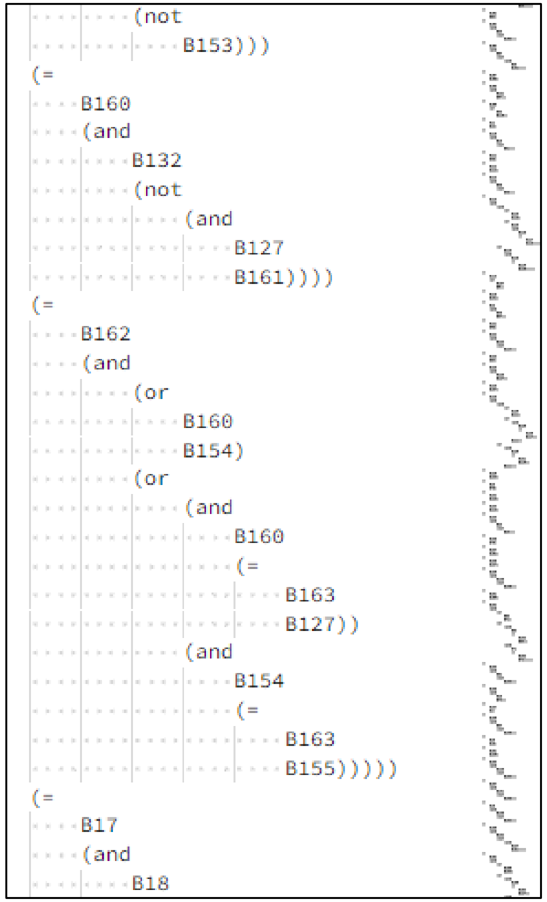
1 pointer value is checked despite dereferencing it earlier

For more information see the checker documentation.

```
    if (state->flags & QStringConverter::Flag::ConvertInitialBom)
        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



```
        (not
            B153)))
(=
    B160
    (and
        B132
        (not
            (and
                B127
                B161))))
(=
    B162
    (and
        (or
            B160
            B154)
        (or
            (and
                B160
                (=
                    B163
                    B127))
            (and
                B154
                (=
                    B163
                    B155)))))
(=
    B17
    (and
        B18
```

Framework limitations: Type modeling

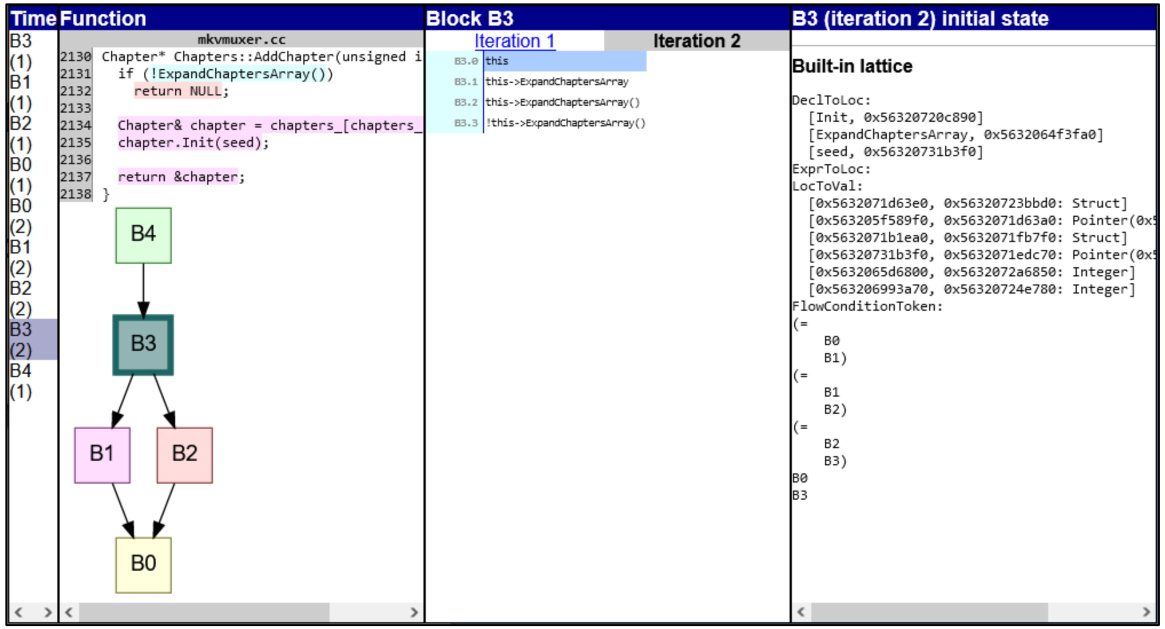
- No C support due to boolean datatype issues
  - Analysis crashes on any condition

```
  int i = 1;

  if (i) {
    // IfStmt
    // |-ImplicitCastExpr 'int' <LValueToRValue>
    // | `-DeclRefExpr 'int' 'i'

    // missing <IntegralToBoolean> cast in C!
  }

```

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

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

  - LLVM. Data-flow analysis – an informal introduction. 2022. https://clang.llvm.org/docs/DataFlowAnalysisIntro.html
  - Uday Khedker, Amitabha Sanyal, and Bageshri Karkare. 2009. Data Flow Analysis: Theory and Practice (1st. ed.). CRC Press, Inc., USA.
  - Keith D. Cooper, Linda Torczon. Chapter 9 - Data-Flow Analysis. Engineering a Compiler (Third Edition). 2023. Morgan Kaufmann.
  - David Hovemeyer, Jaime Spacco, and William Pugh. 2005. Evaluating and tuning a static analysis to find null pointer bugs. SIGSOFT Softw. Eng. Notes 31, 1 (January 2006), 13–19.
  - Collavizza, Hélene, and Michel Rueher. "Exploration of the capabilities of constraint programming for software verification." TACAS 2006, Held as Part of ETAPS 2006, Vienna, Austria, March 25-April 2, 2006. Proceedings 12. Springer Berlin Heidelberg, 2006.