To-Be-Recorded Analysis
In Clad. Summary

Petro Zarytskyi
Mentors: Vassil Vassilev, David Lange
A quick reminder of how TBR analysis works

History of usage of a variable x

DECLARED → USED → USED → CHANGED → CHANGED → CHANGED → USED
A quick reminder of how TBR analysis works

History of usage of a variable x

DECLARED → USED → USED → CHANGED → CHANGED → CHANGED → USED
A quick reminder of how TBR analysis works

History of usage of a variable x

false → true → true → √ → false → × → false → × → false

DECLARED → USED → USED → CHANGED → CHANGED → CHANGED → USED
Overview

**Modes**
used for analysing expressions and finding used variables (data-flow)

**VarData**
stores the information about one variable

**VarDatas graph**
used to handle control-flow
Modes

marking mode

\[ y; \]

no variables are changed, therefore, the marking mode is off

\[ y = x \times x; \]

because of assignment, the marking mode is turned on for RHS
Linear analysis

\[ y = x \cdot x; \quad \Rightarrow \quad _d_x += _d_y \cdot x + x \cdot _d_y; \quad \_d_y = 0; \]

\[ y = 2 \cdot x + 3 \cdot z; \quad \Rightarrow \quad _d_x += 2 \cdot _d_y; \quad _d_z += 3 \cdot _d_y; \quad \_d_y = 0; \]
Modes

by default, the RHS of the assignment operator is in linear mode

addition is not able to affect linearity itself

a product becomes non-linear when both terms are not constant

\[ y = x \times x + z; \]

non-linear mode
Data types

**VarData**
stores all the necessary information about one variable (in trivial cases, it is represented with bool)

**VarsData**
stores information about all the variables (this is a map from VarDecl* to VarData)
FundType VarData

double x;  →  bool
struct myStruct {
    type1 a;
    type2 b;
};

myStruct x;

ObjType VarData

x.a \rightarrow VarData

x.b \rightarrow VarData
RefType VarData

double& x = y;  VarData
(corresponds to y)
RefType VarData

double& x = y;  \quad \rightarrow \quad \text{VarData}
\quad \text{(corresponds to y)}

double& x = (\text{cond} ? y : z);

double& t = \text{arr}[k];
type x[n];

ArrType VarData

x[0] → VarData

x[7] → VarData

x[i] → VarData
Non-constant indices

\[
x = y \times x[k];
\]

\[
x[0] = 1;
\]

this could be any element of \(x\)

here, we have to be conservative and save \(x[0]\)
reqStack

std::vector<std::vector<VarsData>>
```cpp
if (cond1) {
  ///part 1
} else {
  if (cond2) {
    ///part 2
  } else {
    ///part 3
  }
}
```
How are branches merged?

```
std::vector<std::vector<VarsData>>
if (cond1) {
    ///part 1
} else {
    ///part 2
}
```

```
mergedBranch[VD] = branch1[VD] || branch2[VD]
```
What about loops?

```
while (cond)  
///A
```

- Initial branch
  - Branch
    - Branch
      - Branch
        - ///for break and continue statements
  - Branch
    - ///for continue statements
      - ///pass A
What about loops?

```plaintext
while (cond)
  ///A
```

- Initial branch
  - Branch for break and continue statements
    - Continue branch
      - Branch for continue statements
        - Branch for pass A
What about loops?

while (cond)
  ///A

initial branch

branch
  ///for break and
  ///continue statements

branch
  ///pass A once again
What should be implemented in future

- Calling functions should make the analysis proceed to analysing the function
- Add reliable support for references
- Add support for pointers