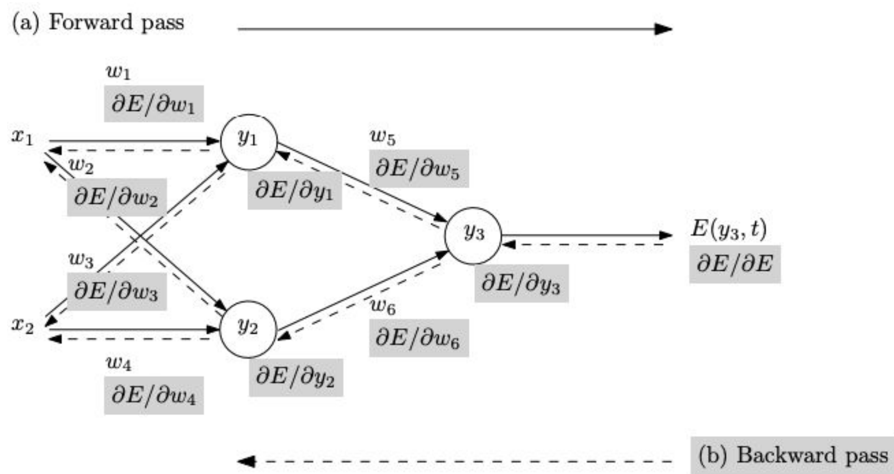
To-Be-Recorded Analysis In Clad. Summary

Petro Zarytskyi

Mentors: Vassil Vassilev, David Lange

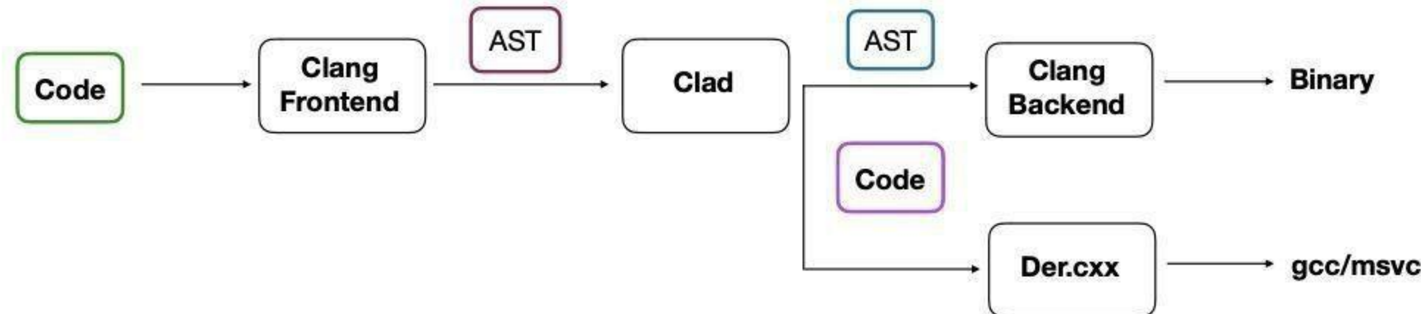
Introduction: Automatic Differentiation

Automatic differentiation is a method of differentiation of functions expressed as procedures. It involves breaking up the function into simple operations and applying chain rule to each one of them. This can be done both ways: from the input to the output (forward mode) and vice versa (reverse mode). This project focuses on the second approach which is more efficient for computing gradients. In reverse mode, we need two passes: a forward pass to store the intermediate values of all the variables and a backward pass to compute derivatives.



Introduction: Clad

Clad is an automatic differentiation Clang plugin for C++. It automatically generates code that computes derivatives of functions given by the user.




```
double f(double x) {
  return x * x;
}
```

```
FunctionDecl f 'double (double)'
|-ParmVarDecl x 'double'
`-CompoundStmt
  `-ReturnStmt
    `-BinaryOperator 'double' '*'
      |-ImplicitCastExpr 'double' <LValueToRValue>
      | `-DeclRefExpr 'double' lvalue ParmVar 'x' 'double'
      `-ImplicitCastExpr 'double' <LValueToRValue>
        `-DeclRefExpr 'double' lvalue ParmVar 'x' 'double'
```

```
double f_darg0(double x) {
  double _d_x = 1;
  return _d_x * x + x * _d_x;
}
```

COMPILER
C | R
RESEARCH

iris hep

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

A quick reminder of how TBR analysis works

History of usage of a variable x

Overview

Modes	used for analysing expressions and finding used variables (data-flow)
VarData	stores the information about one variable
CFG	used to handle control-flow

Modes

marking mode

y;

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

y = x * x;

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

COMPILER C | R RESEARCH

iris hep

Linear analysis

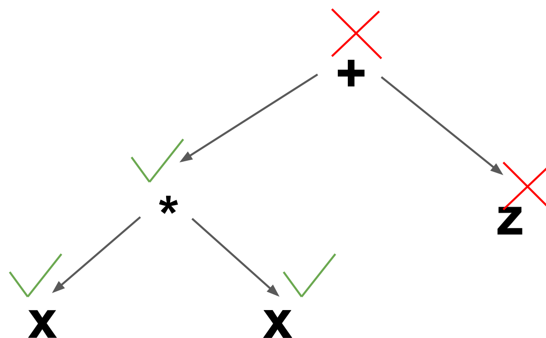
```
y = x * x;          →   _d_x += _d_y * x + x * _d_y;
                        _d_y = 0;
```

```
y = 2 * x + 3 * z;  →   _d_x += 2 * _d_y;
                        _d_z += 3 * _d_y;
                        _d_y = 0;
```

Modes

$y = x * x + z;$

non-linear mode



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 no constant

VarData

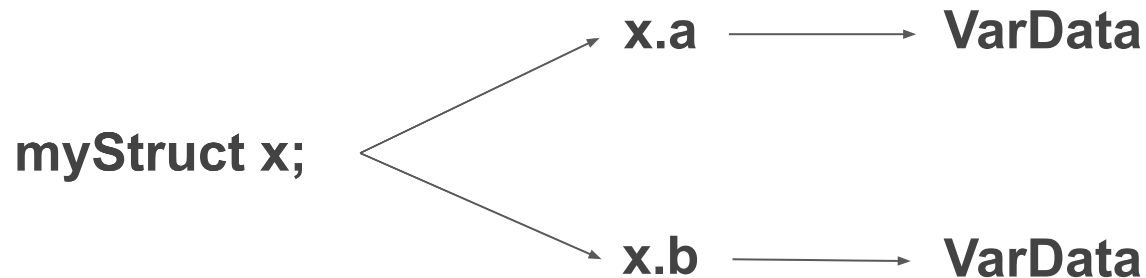
Stores all the necessary information about one variable (in trivial cases, it is represented with bool)

FundType VarData

double x; → **bool**

ObjType VarData

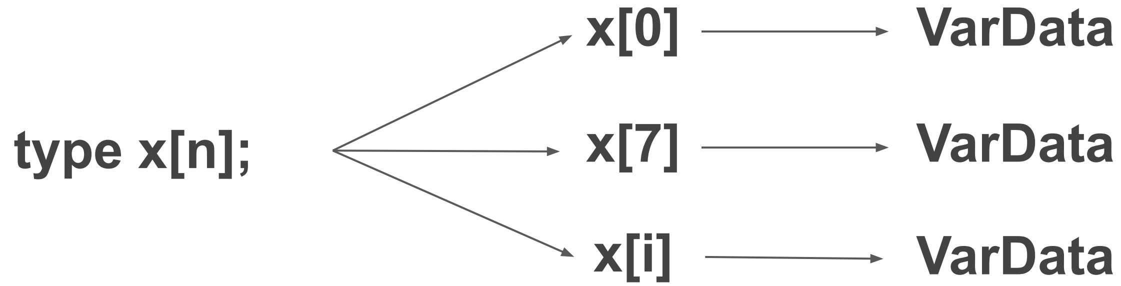
```
struct myStruct {
   type1 a;
   type2 b;
};
```



COMPILER C | R RESEARCH

iris hep

ArrType VarData



COMPILER C | R RESEARCH

iris hep

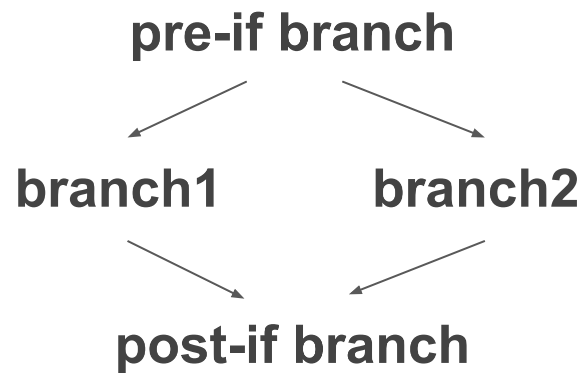
RefType VarData

double& x = y; → clang::Expr* Y (corresponds to y)

Control-flow

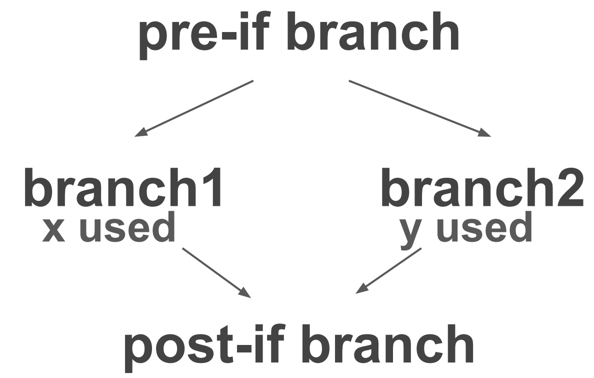
analyzed with clang:CFG

```
if (cond1) {
  ///part 1
} else {
  ///part 2
}
```



Merging

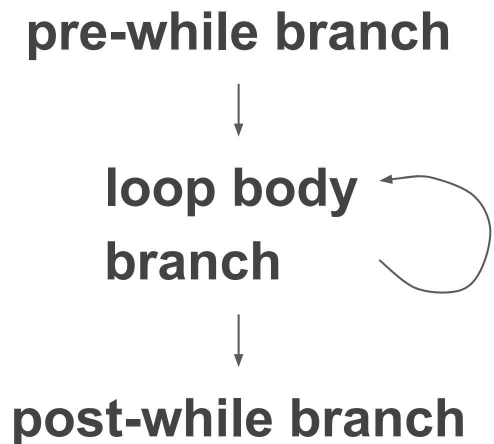
```
if (cond1) {
   ///x used
} else {
   ///y used
}
```



We have to assume both x and y were used.

Control-flow. Loops

```
while (cond) {
  ///some code
}
```



Thank you