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.
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 → 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

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
Linear analysis

\[ y = x \times x; \]

\[ _d_x += _d_y \times x + x \times _d_y; \]
\[ _d_y = 0; \]

\[ y = 2 \times x + 3 \times z; \]

\[ _d_x += 2 \times _d_y; \]
\[ _d_z += 3 \times _d_y; \]
\[ _d_y = 0; \]
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 no constant

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

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

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

myStruct x;

x.a \rightarrow VarData
x.b \rightarrow VarData
type x[n];

ArrType VarData

<table>
<thead>
<tr>
<th>x[0]</th>
<th>VarData</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[7]</td>
<td>VarData</td>
</tr>
<tr>
<td>x[i]</td>
<td>VarData</td>
</tr>
</tbody>
</table>
RefType VarData

double& x = y;  \rightarrow  \text{clang::Expr* Y}  
(corresponds to y)
Control-flow

analyzed with clang:CFG

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

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

We have to assume both \(x\) and \(y\) were used.
while (cond) {
    ///some code
}

pre-while branch
↓
loop body
branch
↓
post-while branch

Control-flow. Loops
Thank you