Advanced optimizations for source transformation based automatic differentiation

Maksym Andriichuk, Petro Zarytskyi, Vassil Vassilev

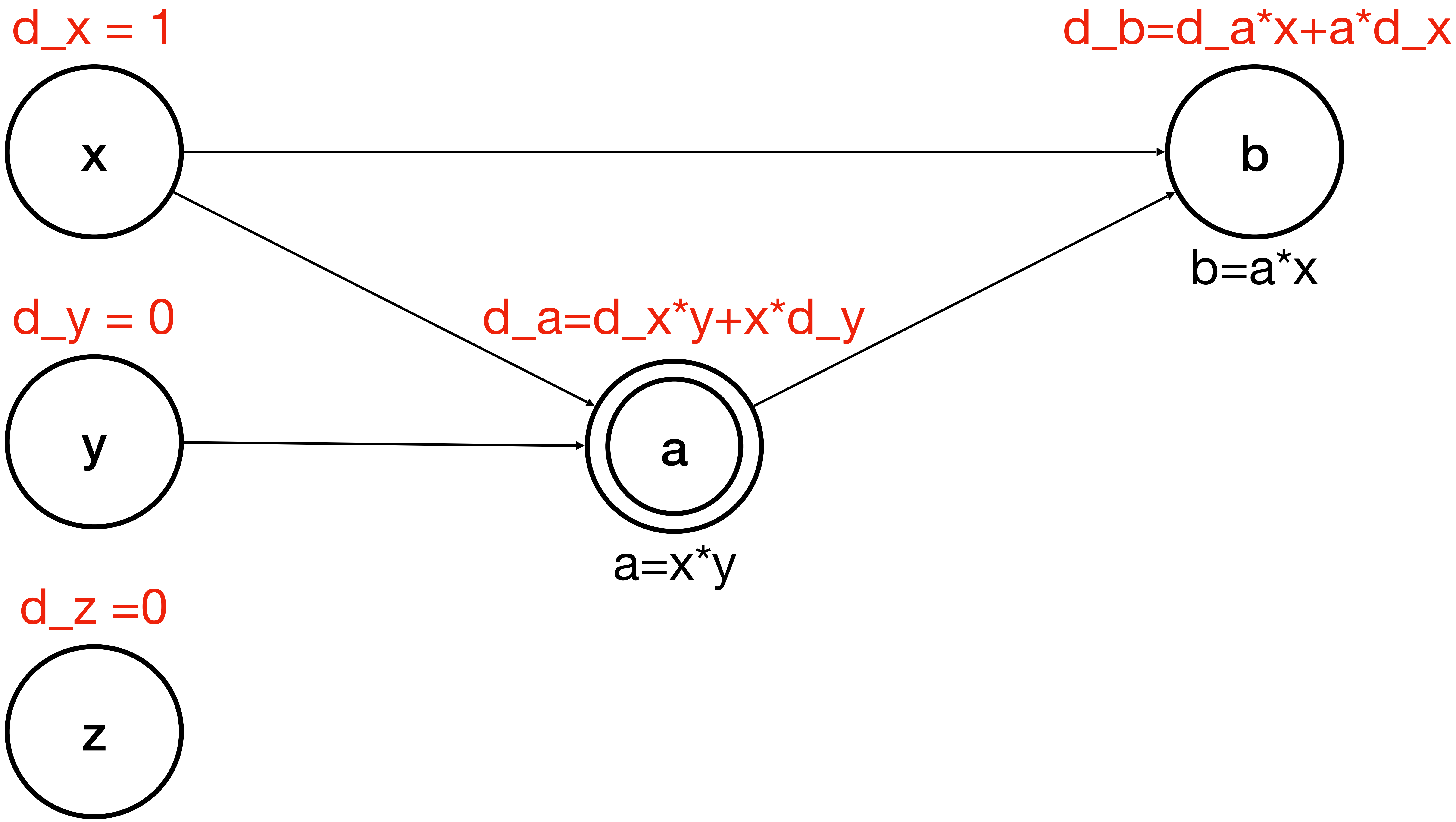
Motivation

Part 1: Automatic Differentiation

AD Forward Accumulation Mode

```
mode(x, y, z):
  a = x*y
  b = a*x
  return a
```

d_a := da/dx



d_x = 1

d_y = 0

d_z =0

d_a=d_x*y+x*d_y

a=x*y

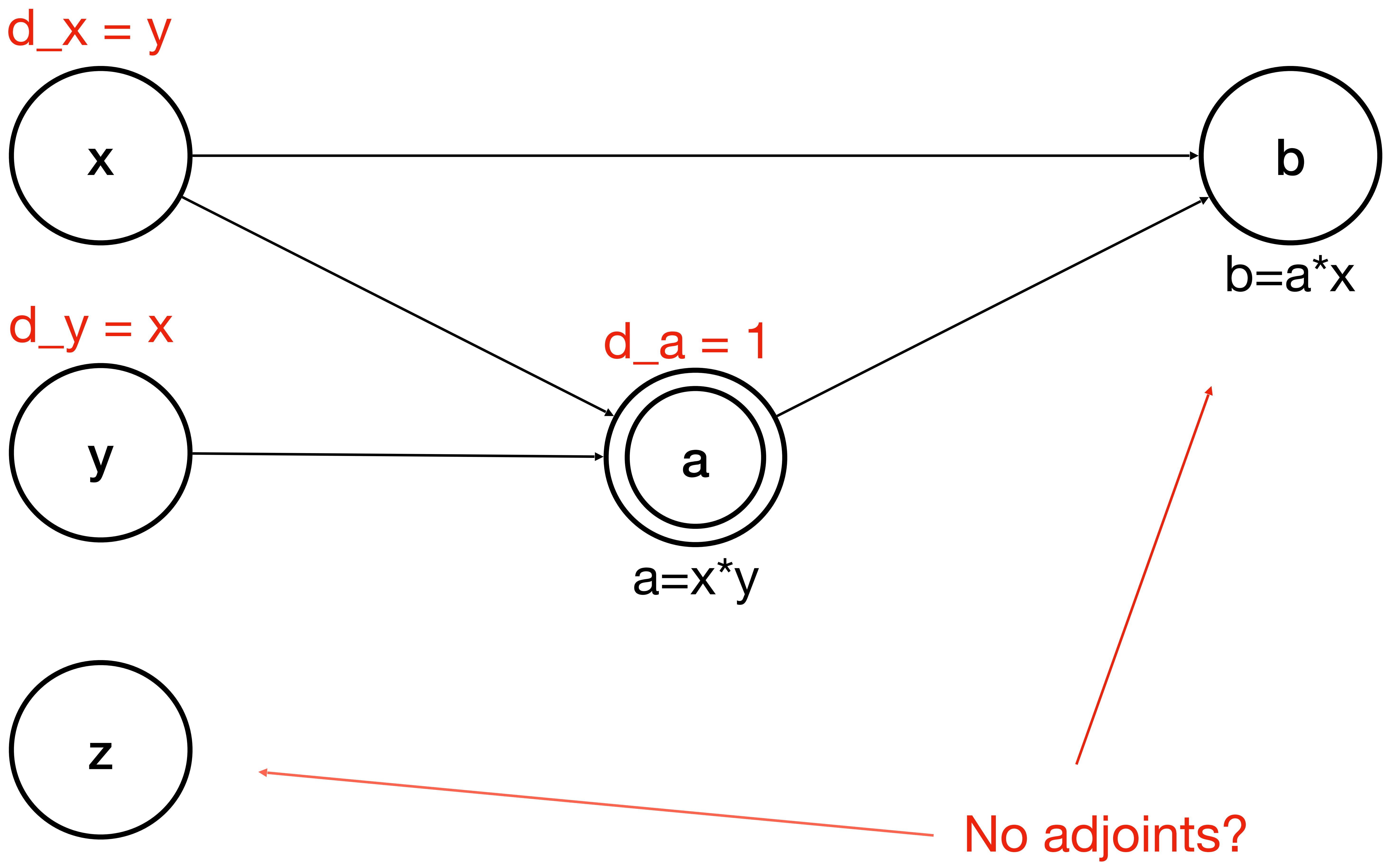
d_b=d_a*x+a*d_x

b=a*x

AD Reverse Accumulation Mode

```
mode(x, y, z):
  a = x*y
  b = a*x
  return a
```

d_b := da/db



Part 2: Clad

Clad: Source-Transformation AD Tool

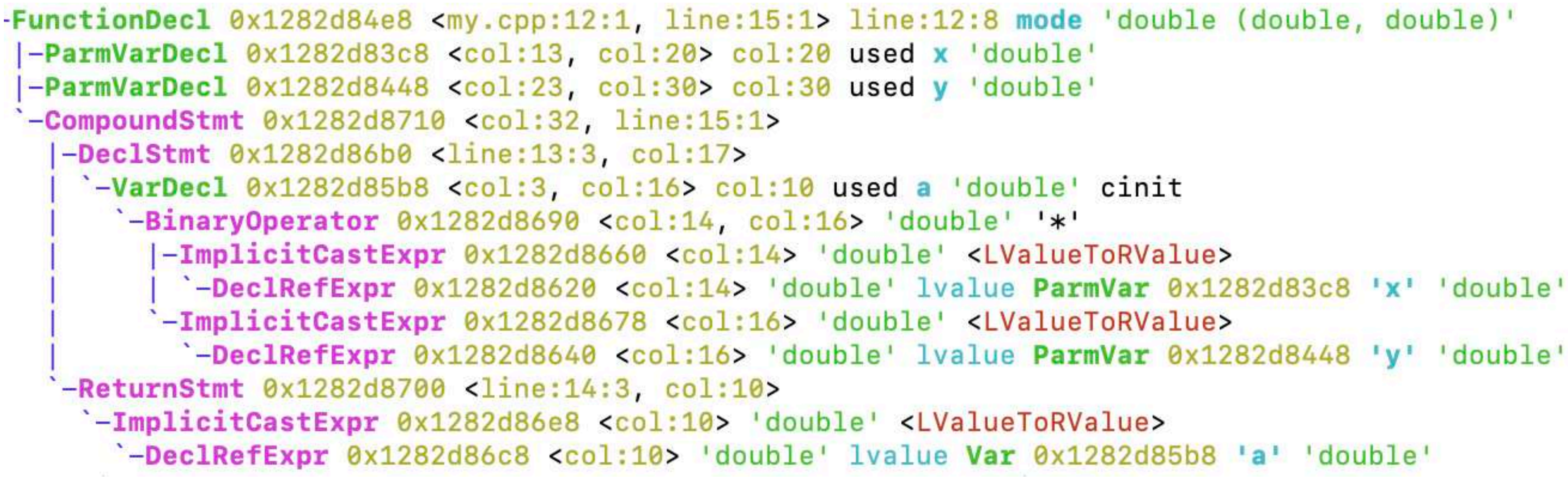
```
double mode(double x, double y){
  double a = x*y;
  return a;
}
```

clad::differentiate(f, “x”) →

```
double mode_darg0(double x, double y) {
  double _d_x = 1;
  double _d_y = 0;
  double _d_a = _d_x * y + x * _d_y;
  double a = x * y;
  return _d_a;
}
```

Clang Abstract Syntax Tree (AST)

```
double mode(double x, double y){
  double a = x*y;
  return a;
}
```



```
-FunctionDecl 0x1282d84e8 <my.cpp:12:1, line:15:1> line:12:8 mode 'double (double, double)'
 |-ParmVarDecl 0x1282d83c8 <col:13, col:20> col:20 used x 'double'
 |-ParmVarDecl 0x1282d8448 <col:23, col:30> col:30 used y 'double'
 `-CompoundStmt 0x1282d8710 <col:32, line:15:1>
   |-DeclStmt 0x1282d86b0 <line:13:3, col:17>
   | `-VarDecl 0x1282d85b8 <col:3, col:16> col:10 used a 'double' cinit
   |   `-BinaryOperator 0x1282d8690 <col:14, col:16> 'double' '*'
   |     |-ImplicitCastExpr 0x1282d8660 <col:14> 'double' <LValueToRValue>
   |     | `-DeclRefExpr 0x1282d8620 <col:14> 'double' lvalue ParmVar 0x1282d83c8 'x' 'double'
   |     `-ImplicitCastExpr 0x1282d8678 <col:16> 'double' <LValueToRValue>
   |       `-DeclRefExpr 0x1282d8640 <col:16> 'double' lvalue ParmVar 0x1282d8448 'y' 'double'
   `-ReturnStmt 0x1282d8700 <line:14:3, col:10>
     `-ImplicitCastExpr 0x1282d86e8 <col:10> 'double' <LValueToRValue>
       `-DeclRefExpr 0x1282d86c8 <col:10> 'double' lvalue Var 0x1282d85b8 'a' 'double'
```

```
double mode(double x, double y){
  double a = x*y;
  return a;
}
```

clad::differentiate(f, “x”)

```
double mode_darg0(double x, double y) {
  double _d_x = 1;
  double _d_y = 0;
  double _d_a = _d_x * y + x * _d_y;
  double a = x * y;
  return _d_a;
}
```

Part 3: Activity Analysis

```
mode(x, y, z):
  a = x*y
  b = a*x
  return a
```

Do we need ‘em?

```
mode_darg0(x, y, z):
  d_x = 1
  d_y = 0
  d_z = 0

  d_a = d_x*y + x*d_y
  a = x*y

  d_b = d_a*x + a*d_x
  b = a*x

  return d_a
```

A variable is called ***varied*** if it depends on some independent input and ***useful*** if some dependent output depends on it.

The claim is that if a variable isn’t varied in the reverse mode or isn’t useful in the forward mode the adjoint could be omitted.

```
mode(x, y, z):
  a = x*y
  b = a*x
  return a
```

```
mode(x, y, z):
  a = x*y
  b = g(x)
  return a
```

Part 4: To-Be-Recorded Analysis

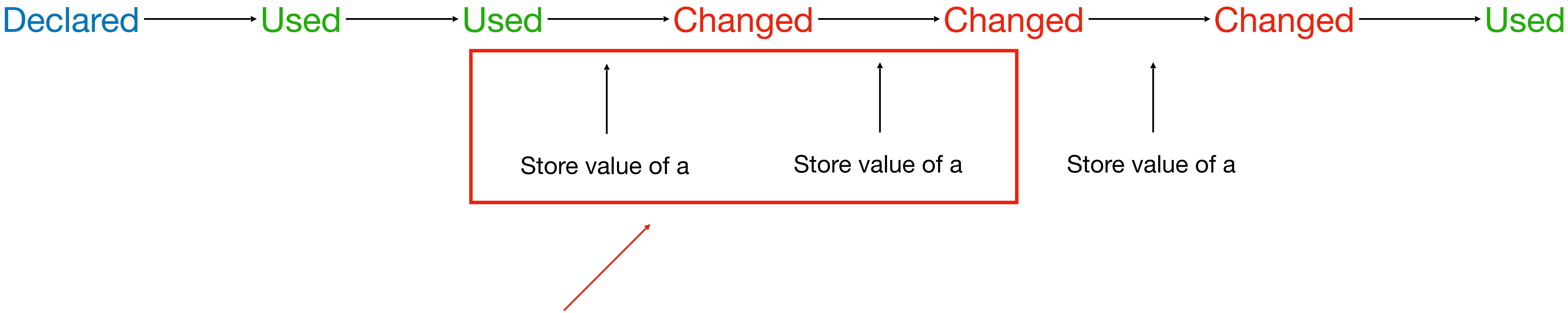
```
mode(x, y, z):
  a = 0
  for i in 1 to 5:
    a += x;
```

```
mode_grad(x):
  d_x, d_i, d_a = 0
  a = 0
  mem_set = {}
  for i in 1 to 5:
    mem_set.push(a)
    a+=x

  d_a += 1

  for i in 5 to 1:
    a = mem_set.pop()
    d_x += d_a
```

History of usage of variable a



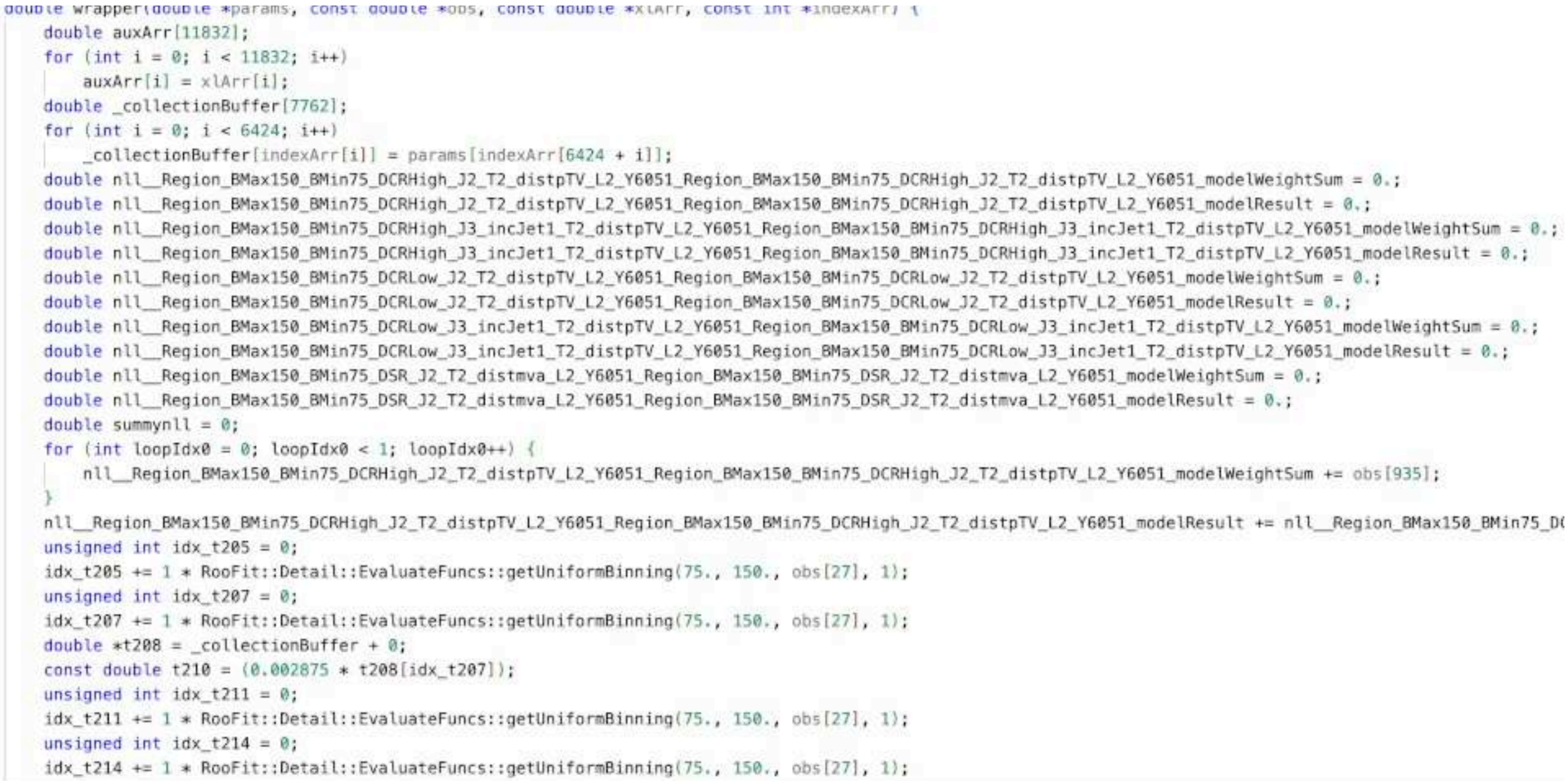
Declared → Used → Used → Changed → Changed → Changed → Used

Store value of a

Store value of a

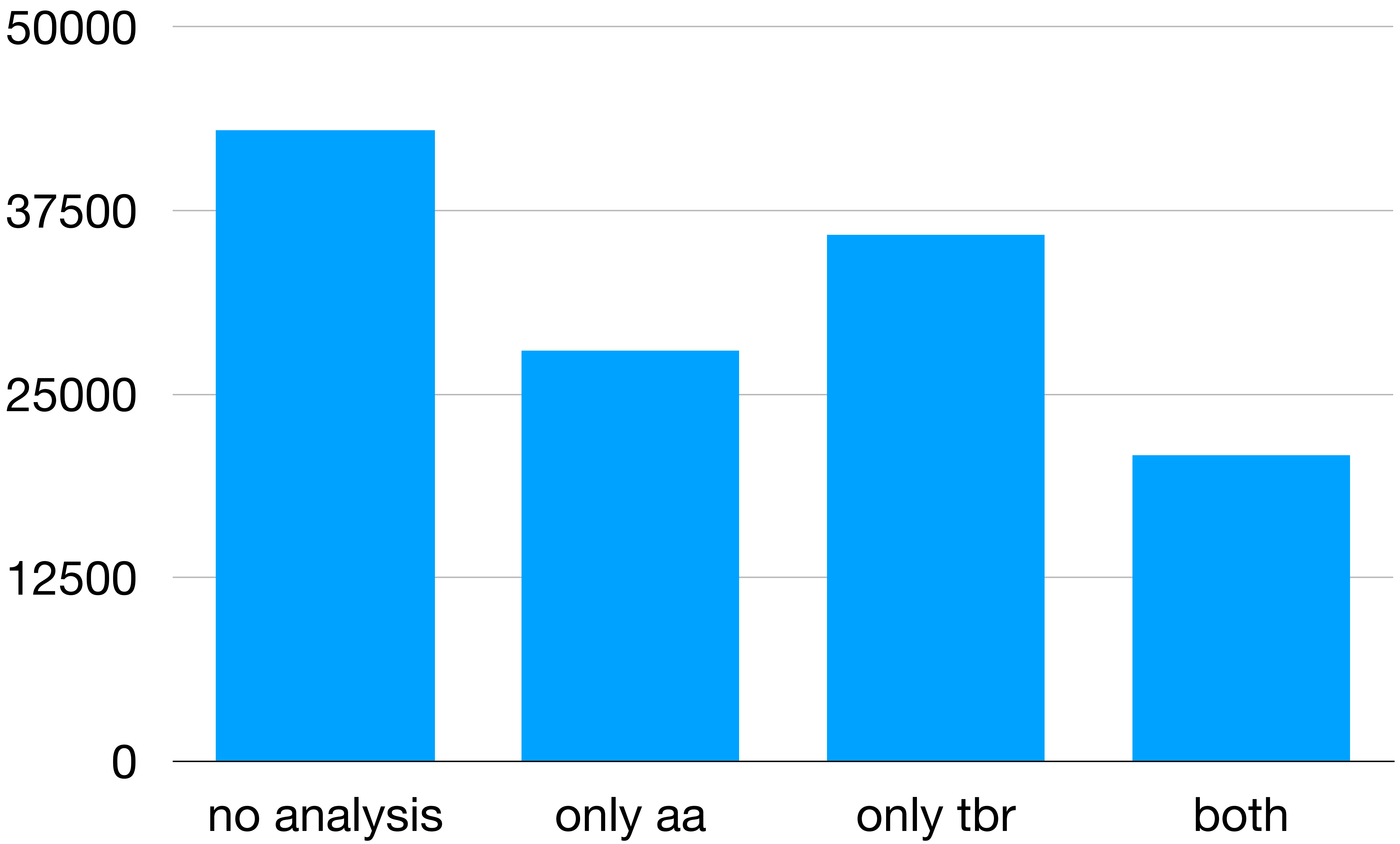
Store value of a

No need to store!



```
double wrapper(double *params, const double *obs, const double *xlArr, const int *indexArr) {
    double auxArr[11832];
    for (int i = 0; i < 11832; i++)
        auxArr[i] = xlArr[i];
    double _collectionBuffer[7762];
    for (int i = 0; i < 6424; i++)
        _collectionBuffer[indexArr[i]] = params[indexArr[6424 + i]];
    double nll__Region_BMax150_BMin75_DCRHigh_J2_T2_distpTV_L2_Y6051_Region_BMax150_BMin75_DCRHigh_J2_T2_distpTV_L2_Y6051_modelWeightSum = 0.;
    double nll__Region_BMax150_BMin75_DCRHigh_J2_T2_distpTV_L2_Y6051_Region_BMax150_BMin75_DCRHigh_J2_T2_distpTV_L2_Y6051_modelResult = 0.;
    double nll__Region_BMax150_BMin75_DCRHigh_J3_incJet1_T2_distpTV_L2_Y6051_Region_BMax150_BMin75_DCRHigh_J3_incJet1_T2_distpTV_L2_Y6051_modelWeightSum = 0.;
    double nll__Region_BMax150_BMin75_DCRHigh_J3_incJet1_T2_distpTV_L2_Y6051_Region_BMax150_BMin75_DCRHigh_J3_incJet1_T2_distpTV_L2_Y6051_modelResult = 0.;
    double nll__Region_BMax150_BMin75_DCRLow_J2_T2_distpTV_L2_Y6051_Region_BMax150_BMin75_DCRLow_J2_T2_distpTV_L2_Y6051_modelWeightSum = 0.;
    double nll__Region_BMax150_BMin75_DCRLow_J2_T2_distpTV_L2_Y6051_Region_BMax150_BMin75_DCRLow_J2_T2_distpTV_L2_Y6051_modelResult = 0.;
    double nll__Region_BMax150_BMin75_DCRLow_J3_incJet1_T2_distpTV_L2_Y6051_Region_BMax150_BMin75_DCRLow_J3_incJet1_T2_distpTV_L2_Y6051_modelWeightSum = 0.;
    double nll__Region_BMax150_BMin75_DCRLow_J3_incJet1_T2_distpTV_L2_Y6051_Region_BMax150_BMin75_DCRLow_J3_incJet1_T2_distpTV_L2_Y6051_modelResult = 0.;
    double nll__Region_BMax150_BMin75_DSR_J2_T2_distmva_L2_Y6051_Region_BMax150_BMin75_DSR_J2_T2_distmva_L2_Y6051_modelWeightSum = 0.;
    double nll__Region_BMax150_BMin75_DSR_J2_T2_distmva_L2_Y6051_Region_BMax150_BMin75_DSR_J2_T2_distmva_L2_Y6051_modelResult = 0.;
    double summynll = 0;
    for (int loopIdx0 = 0; loopIdx0 < 1; loopIdx0++) {
        nll__Region_BMax150_BMin75_DCRHigh_J2_T2_distpTV_L2_Y6051_Region_BMax150_BMin75_DCRHigh_J2_T2_distpTV_L2_Y6051_modelWeightSum += obs[935];
    }
    nll__Region_BMax150_BMin75_DCRHigh_J2_T2_distpTV_L2_Y6051_Region_BMax150_BMin75_DCRHigh_J2_T2_distpTV_L2_Y6051_modelResult += nll__Region_BMax150_BMin75_D
    unsigned int idx_t205 = 0;
    idx_t205 += 1 * RooFit::Detail::EvaluateFuncs::getUniformBinning(75., 150., obs[27], 1);
    unsigned int idx_t207 = 0;
    idx_t207 += 1 * RooFit::Detail::EvaluateFuncs::getUniformBinning(75., 150., obs[27], 1);
    double *t208 = _collectionBuffer + 0;
    const double t210 = (0.002875 * t208[idx_t207]);
    unsigned int idx_t211 = 0;
    idx_t211 += 1 * RooFit::Detail::EvaluateFuncs::getUniformBinning(75., 150., obs[27], 1);
    unsigned int idx_t214 = 0;
    idx_t214 += 1 * RooFit::Detail::EvaluateFuncs::getUniformBinning(75., 150., obs[27], 1);
```

Preliminary Results


no analysis
only aa
only tbr
both