TBR (To-Be-Recorded) Analysis Implementation Strategy for Clad

Petro Zarytskyi
Mentors: Vassil Vassilev, David Lange

General TBR Analysis Strategy

History of usage of a variable x

History of usage of a variable y

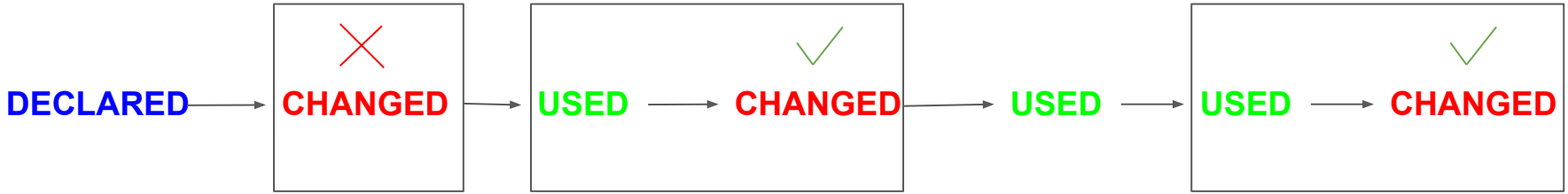
General TBR Analysis Strategy

History of usage of a variable x

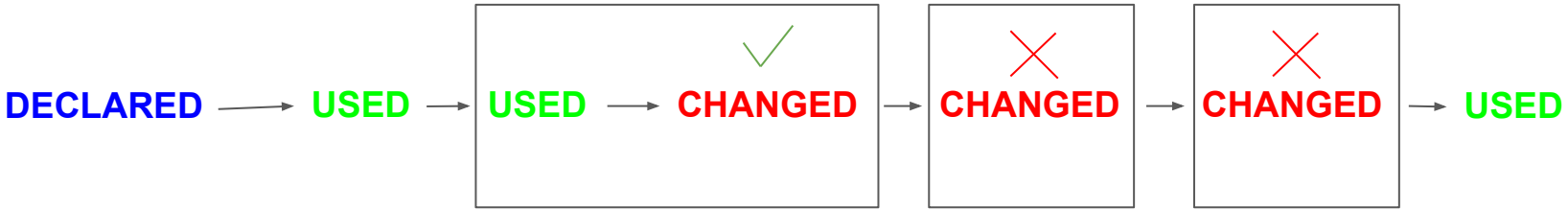
History of usage of a variable y

General TBR Analysis Strategy

History of usage of a variable x



History of usage of a variable y



General TBR Analysis Strategy

History of usage of a variable x

History of usage of a variable y

But what do we mean by used?

```
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;
```

But what do we mean by used?

The same logic applies to += and -=

```
y += x;      ⇔      y = y + x;

y -= x*x;    ⇔      y = y - x * x;
```

But what do we mean by used?

This only applies to *= and /= if the RHS is const

```
y *= 3;   ⇔   y = y * 3;
```

```
y /= x;   ⇔   y = y / x;
```

So how do we keep track of variables' usage?

Let's introduce

std::map<const clang::VarDecl*, bool> Req;

Safe choices

- When we don’t know for sure if a variable was used we should assume it was.
- Similarly, if our model doesn’t give enough information if we should store a variable we store it just in case.

What do we do with conditional statements?

```
if (cond)
   A;
```



```
Req0 = Req;
Visit A;
Req = Req || Req0;
```

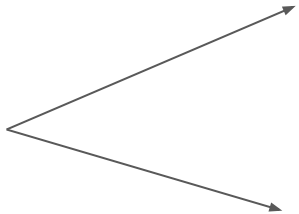
```
if (cond)
   A;
else
   B;
```



```
Req0 = Req;
Visit A;
Req1 = Req;
Req = Req0;
Visit B;
Req = Req || Req1;
```

What about loops?

```
A;
while (cond)
   B;
C;
```



```
A; B; B; B; C;

A; C;
```

```
while (cond)
   B;
```



```
Req0 = Req;
Visit B for TRB analysis only;
Req = Req || Req0;
Visit B;
Req = Req || Req0;
```

What about loops?

```
do (cond){
    B;
} while (cond);
```

```
Req0 = Req;
Visit B for TRB analysis only;
Req = Req || Req0;
Visit B;
```

break/continue statements

```
while (cond){
…
break; //could be the end of the loop
…
continue; //could be either the end or in the middle
}
```

So we have to consider:

- continue statements in the first pass
- both break and continue statements in the second pass

What about function calls?

```
double f (double x) {...}

double g (double &x) {...}
```

f(x); → Req[x] = true;

g(x); → Store x;
Req[x] = false;

Proposed Implementation Sequence

- Create a simple structure in ReverseModeVisitor to track TBR analysis
- Implement TBR analysis for non-array (non-pointer) type variables without control-flow and function calls
- Start tracking linear expressions
- Add support for conditional statements
- Add “TBR only” visiting mode in ReverseModeVisitor and add support for loops without break/continue statements
  Add support for break/continue statements
- Add support for function calls
- Add support for array (pointer) type variables with constant indices
- Add support for objects and member functions
- Add support for functors and lambda functions
- Possibly: Add support for expressions with non-constant indices

What I didn’t mention

- How declarations will be handled
- Storing in multiplication/division for efficiency reasons
- How exactly we will track linear expressions
- How we will analyze conditions
- The way TBR analyzer will be organized
- Member functions, functors, lambdas