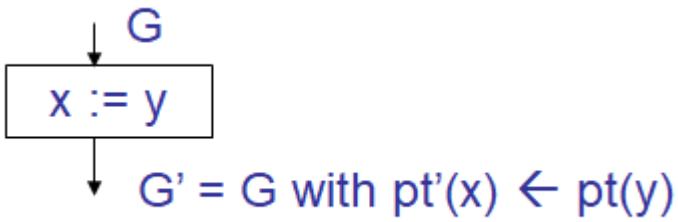
Alias Analysis Implementations in LLVM

Nikhil Hegde

Compiler Optimizations course @ QUALCOMM India Pvt. Ltd.

Exercise (Precision and Speed)

- After a statement x=y, using the dataflow equation shown below, we have “x points to everything that y points to”



- Complexity?
- Complexity as per Steensgaard’s algorithm?
- Precision?
- Precision as per Steensgaard’s algorithm?

Alias Analysis

- DSA – Data Structure Analysis
  - Flow-insensitive, field-sensitive, context-sensitive
  - Proposed by Chris Lattner in 2007
  - Recall: Anderson's is inter-procedural, flow- and context-insensitive
- BasicAA
- TypeBasedAA
- ScopedNoAliasAA
- GlobalsAA
- CFL-AndersAA / CFL-SteensAA

Alias Analysis – LLVM Terminology

In namespace

`llvm::AliasResult::Kind::`

- MayAlias
- PartialAlias
- MustAlias
- NoAlias

- ModRefs
  - Does an instruction modify or refer to a memory location? Useful when the instruction is a call to another function

Alias Analysis – Modelling LLVM Instructions

```
a=&b,     a=b,   a=*b,  *a=b
```

- store <ty> <value>, <ty>* <pointer> (equivalent to *a = b)

```
int x;                  %x = alloca i32 , align 4
int *p;                 %p = alloca i32*, align 8
P = &x;                 store i32* %x, i32** %p, align 8
```

- <result> = load <ty>, <ty>* <pointer> (equivalent to a = *b)

```
int *p;                 %p = alloca i32*, align 8
*p = 0;                 %0 = load i32** %p, align 8
                        store i32 0, i32* %0, align 4
```

Alias Analysis – Modelling LLVM Instructions

```
int g;
void f(int *p){ *fp=10;}

int main(){
int *p;
P = &g;
f(p);
return 0;
}
```

- <result> = call <ty> <fnptrval>(<function args>) (equivalent to a = b for parameter-pass-by-val)

Alias Analysis – Some LLVM Instructions involving ptrs

- `PHI`
- `GEP`
- `bitcast`
- `inttoptr`
- `select`

Alias Analysis - BasicAA

- Assumptions:
  - Distinct global, heap and stack allocations never alias (/ null pointer)
  - Different fields of a structure never alias
  - Array elements with indices that differ at static-time never alias
- Is intra-procedural, stateless
- E.g.

global

```
%a = getelementptr @Z, 10
%b = bitcast %a to float*
%c = select i1 %p, %b, %x
%d = phi [ ... %c ... ]
%e = getelementptr %d, %n
```

Given a ptr value, find out how the ptr value is computed: walk up the use-def chain. E.g., we get e points to `Z`. Using set intersection determine if ptr1 aliases with ptr2.

```
opt ../test/aa2.ll -disable-output -passes=print-alias-sets
-aa-pipeline=basic-aa
```

Alias Analysis - TBAA

- Type Based Alias Analysis

```
float* x;
int y = 10;
int z = 20;
*x = 30; //x can never be &y or &z
…
```

```
void foo(){
    int x;
    *(float *)&x=2.3;
    x=4;
}
```

tbaa.ll

```
%x = alloca i32, align 4
store float 0x4002666660000000, ptr %x, align 4
store i32 4, ptr %x, align 4
ret void
```

Illegal. So compiler could produce a wrong code.

```
opt ../test/tbaa.ll -disable-output -passes=print-alias-sets
-aa-pipeline=basic-aa
```

Alias Analysis - TBAA

- Type Based Alias Analysis

```
void foo(int *x, float *y){
    *x=1;
    int i=*x;
    *y= 1.0;
    float f=*y;
    use(i, f);
}
```

Can we reorder load from y because there is no alias with other ptr x?

Alias Analysis vs. Memory Dependence Analysis

```
for(int i=0;i < 100; i++){
   x[i] = x[i-1] + 100;
}
```

- Alias analysis would say noalias between x[i] and x[i+1]
- Memory dependence analysis would show a dependence between read to a memory location in an iteration and write of to the same memory location in previous iteration.

Alias Analysis – Try it yourself

```
1  int foo(){
2          int a, b, c[100], d;
3          extern int *q;
4          q=&a;
5          a=2;
6          b=*q+2;
7          q=&b;
8          for(int i=0;i<100;i++){
9                  c[i] = c[i] + a;
10                 *q = i;
11         }
12         d = *q + a;
13         return d;
14 }
```

- Line 10: can we move it out of the loop?
- Write an AA pass that enables foo to be optimized.

warning: major effort involved