What is Alias Analysis?

Given two pointers:

Do they always point at different memory?

Do they always point at the same memory? (miss a CSE? ;-))
for (size_t i = 1; i < n; ++i) {
    p[i] = p[i - 1] * 3;
}
LLVM's Alias Analysis API
Location, Location, Location

Pointer
Size
TBAA tag

Sizes are given in address units (bytes usually)
AliasAnalysis.h basics

alias
  - 2 Locations

getModRefInfo and getModRefBehavior

pointsToConstantMemory
The language of alias

NoAlias = can reorder*
MustAlias = redundant load, dead store
MayAlias = I don't know
PartialAlias = Inexact overlap
   (Perhaps this should be renamed?)
LLVM IR features

- noalias
- arguments *and* return values
- tail
- nocapture
- readonly, readnone
- getelementptr (aka gep)
  - gep(p, 0) vs gep(0, p)
- inttoptr, ptrtoint
- no guessing!
A tale of two pointer arithmetics

%p = gep %base, %n

%x = inttoptr %base
%y = add %x, %n
%p = ptrtoint %y
LLVM IR non-features

Union types
Typed memory
restrict anywhere but function arguments
restrict on a struct member
Real multi-dimensional array access
Multiple "variables" in one allocation
AA Implementations

BasicAA
SCEV-AA?
TBAA
Globals ModRef
etc.
Implementation infrastructure

The theory: Multiple chained analyses

NoAlias or MustAlias = best possible answer

MayAlias = I don't know, keep looking

PartialAlias = stop looking
%a = getelementptr @Z, 10
%b = bitcast %a to float*
%c = select i1 %p, %b, %x
%d = phi [ ... %c ... ]
%e = getelementptr %d, %n
SCEV-AA

An interesting concept hack. BasicAA can now do most of this.

Also, how do we keep the ScalarEvolution analysis up to date?
Globals Mod/Ref

Global Variables are Values, with use lists.
Use-list escape analysis
Check for read-only, etc.
NoAA

Says "I don't know" to all queries.
What about Andersen's?

stateful alias analyses

compile time
Pointers to different "types" don't alias.
TBAA: "Tibah", from the Vulcan

T'PAU

just kidding
TBAA (in C)

Introduced in C89, refined in C99

C++ inherited the C89 version and made its own adaptations.

int *a = ???;
float *b = ???;
Practical TBAA

It's all about the lvalues
void foo(int *x, float *y) {
    *x = 1;
    int i = *x;
    *y = 1.0f;
    float f = *y;
    use(i, f);
}
TBAA in C++?

For C++ types, the same problems as C

However, virtual classes are more constrained!

Maybe?
TBAA in LLVM

Memory has no types.

Separate mechanism from policy.

Use chaining to be conservative about punning.

Support cross-language inlining.
A Type Tree

Ancestors, Roots

Regular C
- char
  - float
  - int

Crazy C
- float
- int
struct MyClass {
    int foo;
    float bar;
};
How about a more precise DAG...

```
struct MyClass {
  int foo;
  float bar;
};
```
Alternatives

Type DAG?
Instructions get multiple tags?
A separate datastructure for aggregates?
Type punning

```c
int x;
*(float *)&x = 2.3f;
x = 4;
```

TBAA says NoAlias

BasicAA says MustAlias
Questions?