Flang alias analysis improvements

This page is a design document on some proposed flang changes, based on the work done with the SN-Bone benchmark.

Background/Motivation

Here is a reproducer to help clarify what we are aiming for. In this example the loop performs multiple loads and stores

1. Loads on User_Krylov, to extract base pointers. User_Krylov is not marked with any specific attribute
2. Loads on Basis and Hessenberg, marked as POINTER
3. Store in Basis, marked as POINTER

Given that User_Krylov is not marked as TARGET, we know that there is no possible aliasing between 1 and (2,3), so we should be able to hoist 1 outside of the loop.

```
MODULE snbone_reproducer

        TYPE Method_Krylov_Type
           Real, POINTER :: Basis(:,:)
           Real, POINTER :: Hessenberg(:,:)
        END TYPE Method_Krylov_Type

CONTAINS

        RECURSIVE SUBROUTINE FGMRES_Threaded(User_Krylov,MyStart,MyEnd, &
                             NumThreads,Inner)
        TYPE (Method_Krylov_Type)  User_Krylov
        Integer  MyStart,MyEnd
        Integer  NumThreads
        Real A, B, C, D
        Integer  I,J,K,Inner

        DO J = 1,Inner
           DO I = MyStart,MyEnd
              User_Krylov%Basis(I,Inner+1) = User_Krylov%Basis(I,Inner+1) - User_Krylov%Hessenberg(J,Inner)
*User_Krylov%Basis(I,J)
           END DO
        END DO

        END SUBROUTINE FGMRES_Threaded
END MODULE snbone_reproducer
```

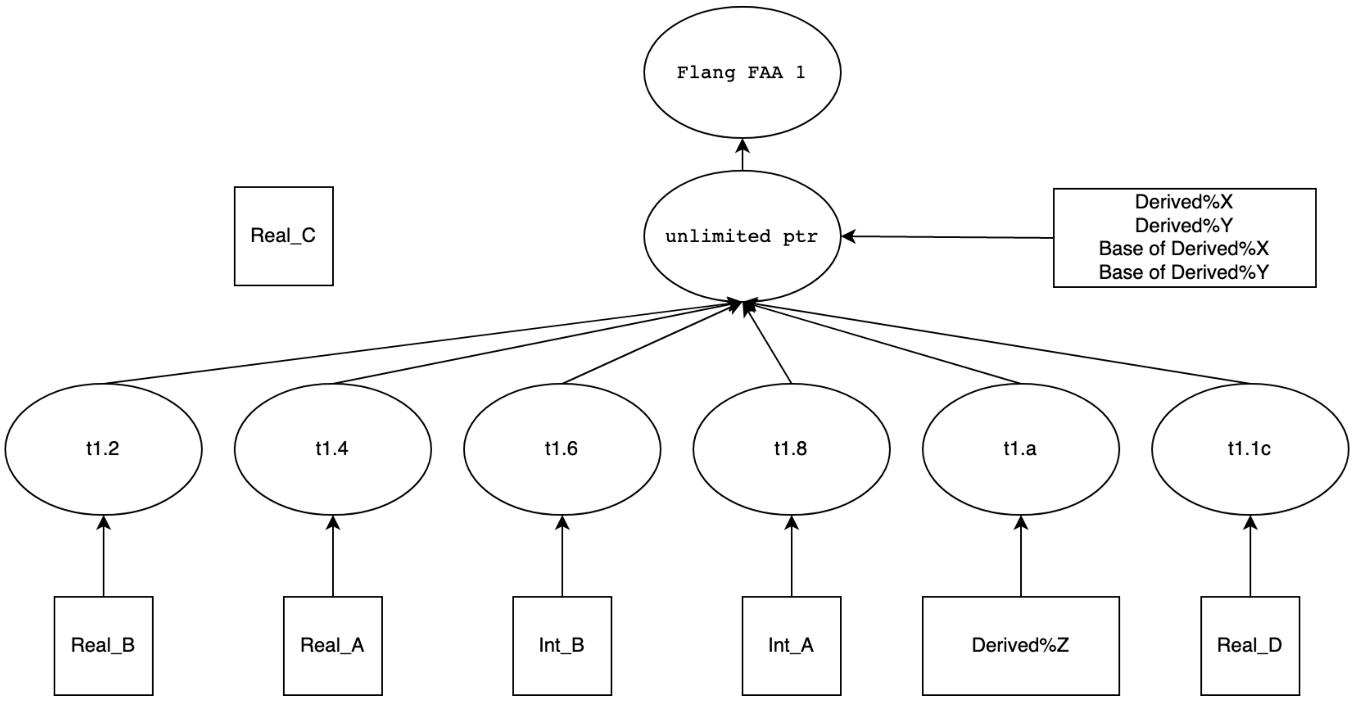
State of TBAA in ACfL master branch

Here is an even more basic program, showing different cases of types and POINTER/TARGET attributes. The graph on the right shows the TBAA generated for that program by a top-of-tree build of ACfL. The ellipses are types defined through TBAA metadata, and the rectangles are memory accesses, being assigned a TBAA type by flang.

Alias analysis can be inferred from the TBAA diagram. 2 accesses may alias if the type of one access can lead to the type of the other access through parenthood relationships. For instance:

- Real_A is of type "t1.4". The parenthood lineage includes "unlimited ptr" and "Flang FAA 1", but not "t1.2". The same is true when starting the other way around, from "t1.2". This is how we can tell that Real_A and Real_B do not alias.
- Derived%Y is of type "unlimited ptr", which is in the parenthood lineage of "t1.4". So Derived%Y and Real_A may alias

```
MODULE simple_prog

        TYPE My_Derived_Type
           Real, POINTER :: X(:,:)
           Real, POINTER :: Y
           Integer :: Z
        END TYPE My_Derived_Type

CONTAINS

        RECURSIVE SUBROUTINE FGMRES_Threaded(Derived, Int_A,
                                                Int_B, &
                                  Real_A, Real_B, Real_C, Real_D)
        TYPE (My_Derived_Type) Derived
        Integer  Int_A, Int_B
        Real Real_A, Real_B
        Real, POINTER :: Real_C
        Real, TARGET :: Real_D

        Real_A = Real_B
        Int_A = Int_B
        Real_A = Derived%X(1, 1)
        Real_B = Derived%Y
        Real_C => Real_D
        Derived%Y => Real_D
        Derived%Z = Int_A

        END SUBROUTINE FGMRES_Threaded
END MODULE simple_prog
```



Aliasing analysis to be improved

Based on the F90 specification (https://wg5-fortran.org/N001-N1100/N692.pdf, page 261), we can extract the following rules regarding TARGET/POINTER variables.

- A TARGET variable can not alias with a variable which has no POINTER attribute
- A POINTER variable can not alias with a variable which has no TARGET/POINTER attribute
- A TARGET variable can alias with any POINTER variable

- POINTER variables can alias with each other
- TARGET variables can not alias with each other

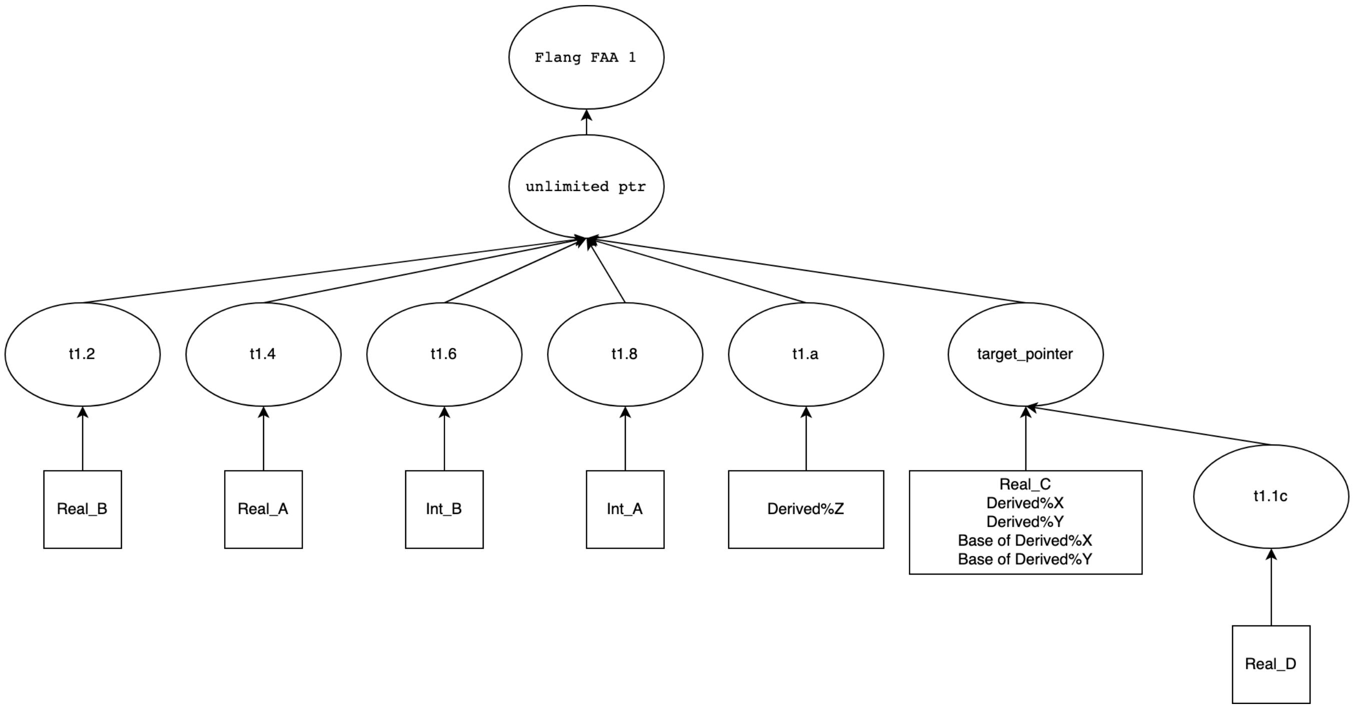
Proposed improvements for Flang TBAA generation

We are proposing the following change, which puts all the POINTER and TARGET variables under the same TBAA type, called "target_pointer"

This includes several changes from the current design:

- All POINTER variables are no longer associated with "unlimited ptr", but rather with "target_pointer"
- Any variable marked as TARGET should get a new type as it is now, but the parent is "target_pointer" rather than "unlimited ptr".

This design ensures all the rules listed in the previous section.



Further considerations

C-style pointers

C-style Cray pointers are implemented in Flang through variables with a pointer type.

> *DTY(DTYPEG(sym)) == TY_PTR*

The proposed improvements will leave these pointers as they are. This means that they will be marked as "MayAlias" will all other memory accesses.

The proposed improvements will only affect variables explicitely marked with the POINTER attribute.

Load of the base address

Let's consider the following fortran snippet

```
TYPE My_Derived_Type
    Real, POINTER :: Y
END TYPE My_Derived_Type

TYPE (My_Derived_Type) Derived
Real Real_A

Derived%Y = Real_A
```

Access to Derived%Y is done through the following IR:

```
%2 = bitcast i64* %derived to float**, !dbg !52
%3 = load float*, float** %2, align 8, !dbg !52, !tbaa !57
store float %1, float* %3, align 4, !dbg !52, !tbaa !57
```

We first load the base address of the pointer Y then do a store at that address. This currently uses the same TBAA type for both operations, which is the "unlimited ptr" type. This means that both operations may alias with all other memory accesses.

This is not the right thing to do, as the pointer itself is actually a local variable, so it should be treated like other variables with no attributes. However, it is not the purpose of the proposed improvements to fix this particular issue.

We propose to keep the same TBAA type for both the load of the base address and the memory access. The TBAA type will now be the new "pointer_target", instead of "unlimited ptr", so the new status for the loads of base address will be

- It does not alias with variables with no TARGET/POINTER attributes (this is correct)
- It may alias with any POINTER/TARGET. That's suboptimal, but this is not a regression. Fixing this will be left as future work.

Autogenerated examples

The considerations above, as well as multiple combinations of target/pointer/derived types have been described in small Fortran examples. The following document provides a list of those Fortran examples, as well as the current state of the alias analysis, and the proposed alias analysis. All diagrams have been generated in a reproducible way, through automated scripts. The proposed alias analysis comes from a local build of armflang, with a patch in development which implements the proposed improvements.

tbaa_analysis.pdf