Simple Program

This example shows the basic mechanism.
Accesses to Derived%X (a POINTER), can
alias with Real_B (a TARGET), but not
with Real_A
For Derived%X, both the base address and
the actual data access have the same analysis.
Analysing more accurately the access to
the base address will be left for a
future effort.
==========

==========
MODULE simple_prog

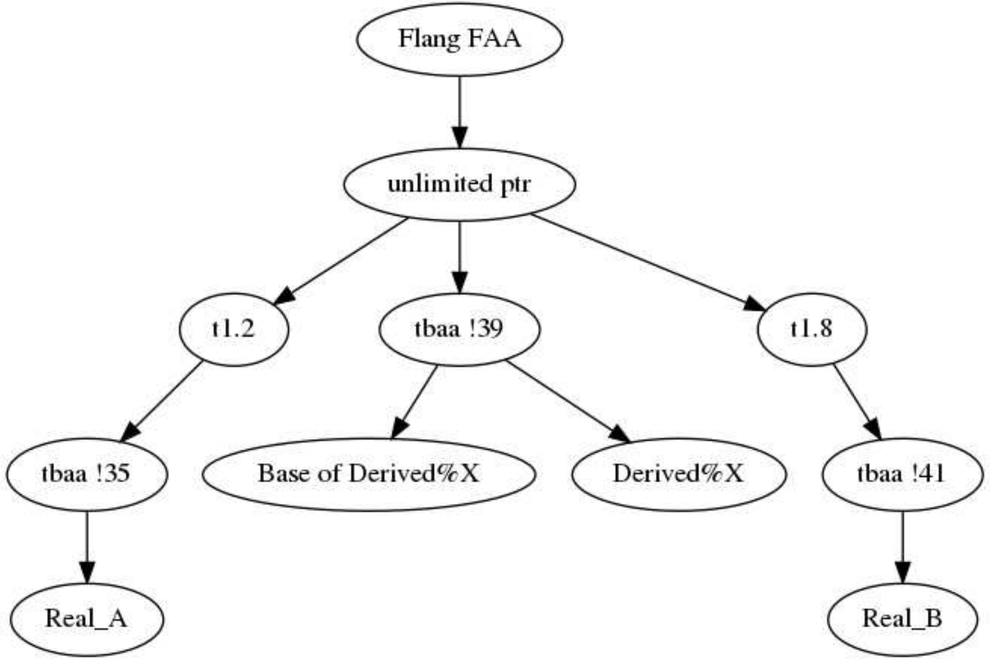
TYPE My_Derived_Type
Real:: Y
Real, POINTER:: X
END TYPE My_Derived_Type

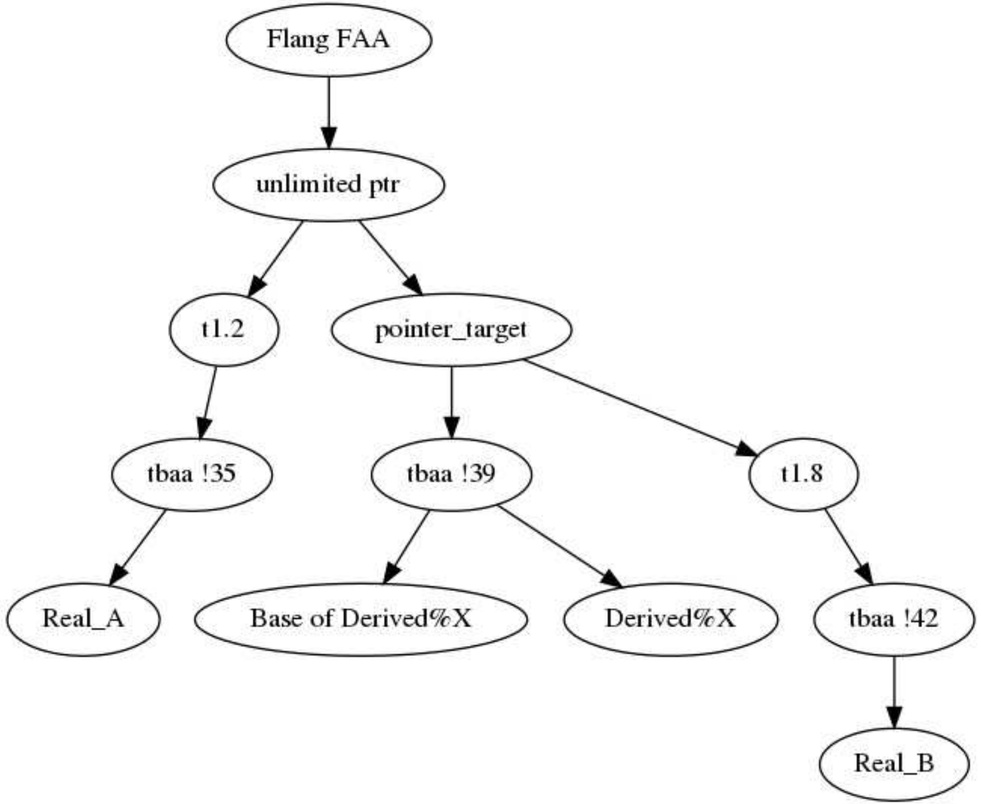
CONTAINS

RECURSIVE SUBROUTINE FGMRES_Threaded(Derived,&
Real_A, Real_B)
TYPE (My_Derived_Type) Derived
Real Real_A
Real, TARGET :: Real_B

Derived%X = Real_A
Derived%X = Real_B

END SUBROUTINE FGMRES_Threaded
END MODULE simple_prog





Pointers in 2 derived types

We ensure that all pointers are
marked as aliasing with each other, even
if they are located in separate Derived types.
==========

==========
MODULE simple_prog

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

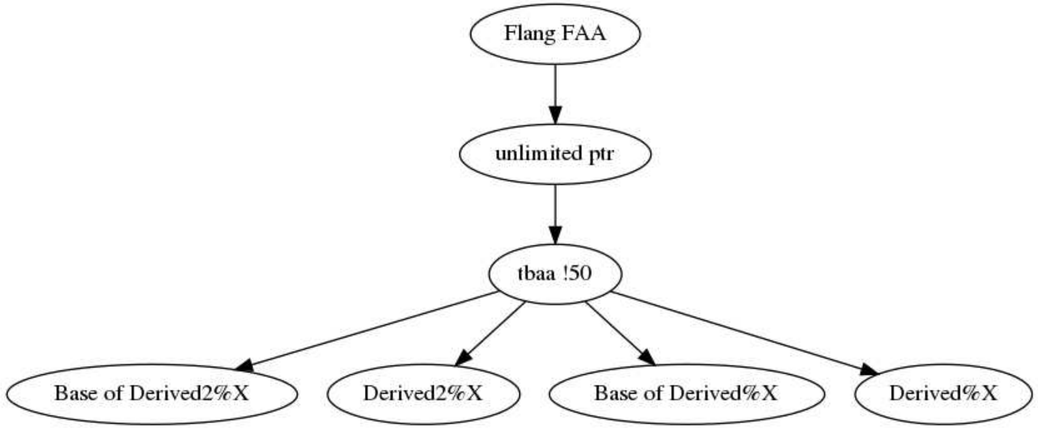
TYPE My_Derived_Type2
Real:: Y
Real, POINTER:: X
END TYPE My_Derived_Type2

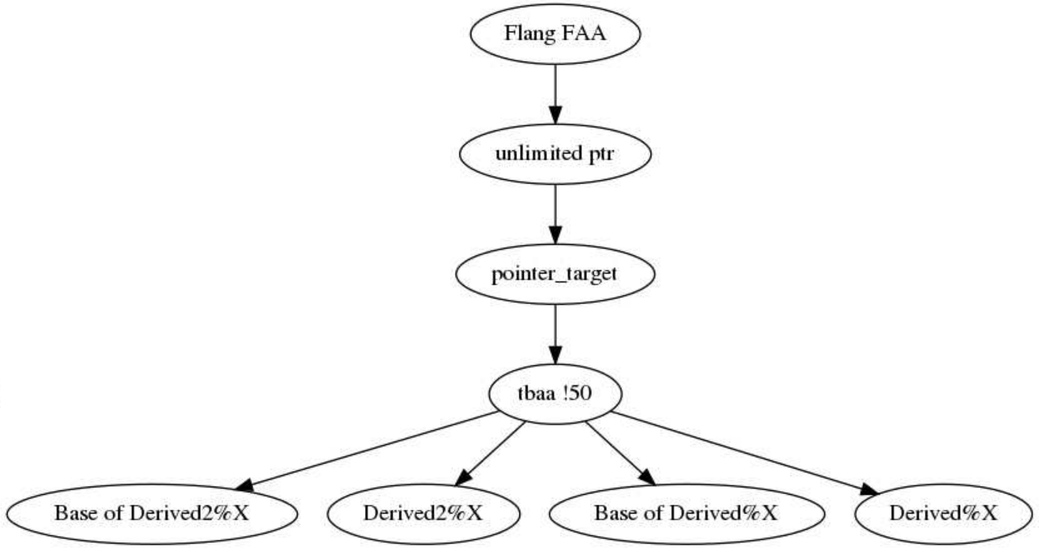
CONTAINS

RECURSIVE SUBROUTINE FGMRES_Threaded(Derived,&
Derived2)
TYPE (My_Derived_Type) Derived
TYPE (My_Derived_Type2) Derived2

Derived%X = Derived2%X

END SUBROUTINE FGMRES_Threaded
END MODULE simple_prog





Pointers in nested derived types.

- The nesting of Derived types doesn't prevent
all POINTERS from aliasing with each other.
- The POINTERS in various nested types are
correctly aliasing with the local TARGET variables.
- The POINTERS in various nested types are not
aliasing with any of the non TARGET/POINTER
variables, including local variables as well
as members of all nested types.
==

==
MODULE simple_prog

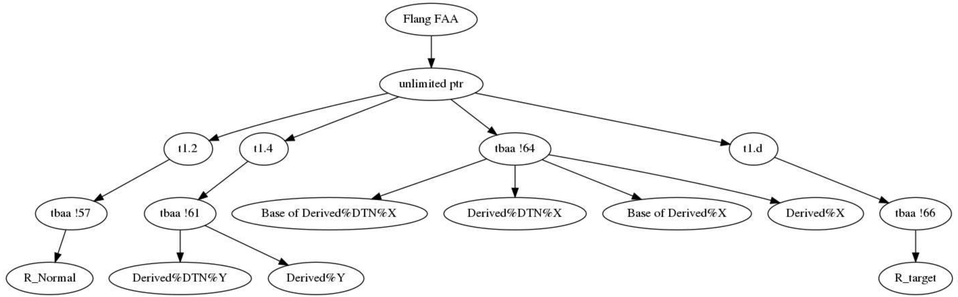
TYPE My_Derived_Type_Nested
Real:: Y
Real, POINTER:: X
END TYPE My_Derived_Type_Nested

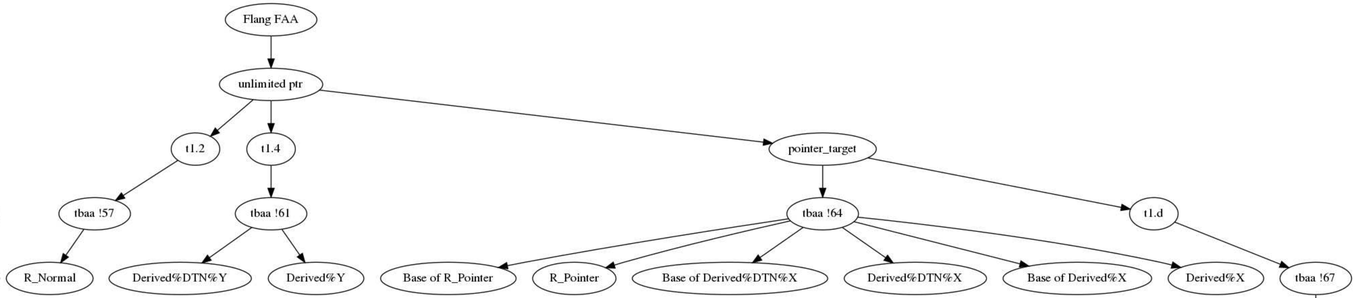
TYPE My_Derived_Type
Real:: Y
Real, POINTER:: X
TYPE (My_Derived_Type_Nested) DTN
END TYPE My_Derived_Type

CONTAINS

RECURSIVE SUBROUTINE FGMRES_Threaded(Derived,&
R_Normal, R_Pointer, R_Target)
TYPE (My_Derived_Type) Derived
Real R_Normal
Real, POINTER :: R_Pointer
Real, TARGET :: R_Target

Derived%DTN%Y = R_Normal
Derived%DTN%X = R_Pointer
Derived%DTN%X = R_Target
Derived%DTN%X = Derived%Y
Derived%DTN%X = Derived%X



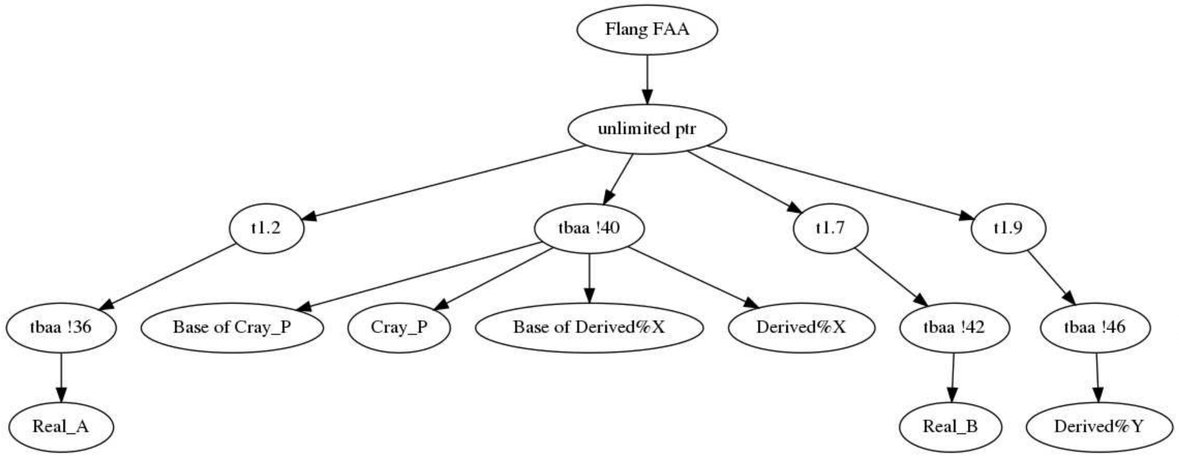


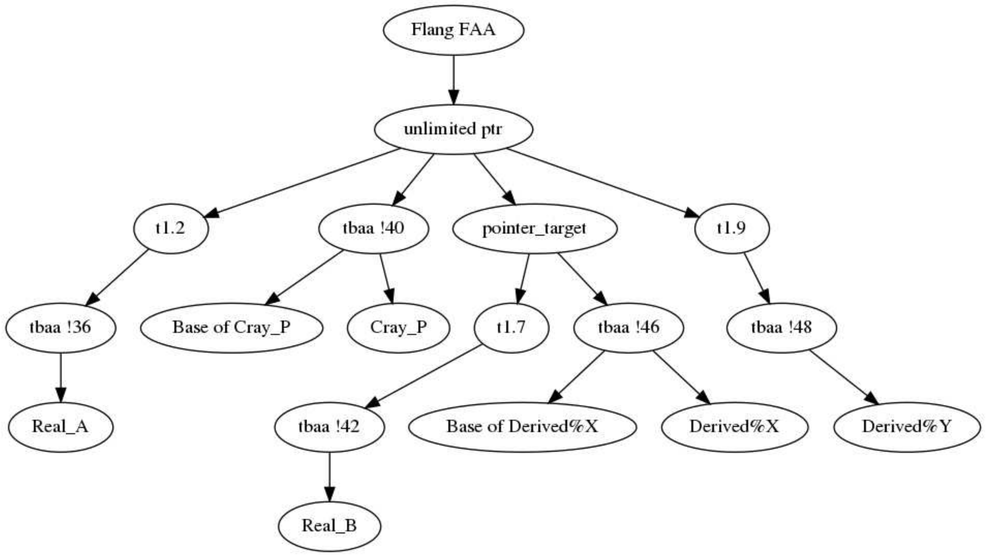
Cray pointers

Cray pointers are not subject to the same
optimizations as the POINTER/TARGET variables
This test checks that the aliasing analysis
is unchanged by the patch for cray pointers,
i.e. they may alias with everything
==========

==========

```
MODULE simple_prog

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

CONTAINS

RECURSIVE SUBROUTINE FGMRES_Threaded(Derived,&
Real_A, Real_B, ipt)
TYPE (My_Derived_Type) Derived
Real Real_A
Real, TARGET :: Real_B
POINTER (ipt, Cray_P)

Cray_P = Real_A
Cray_P = Real_B
Cray_P = Derived%X
Cray_P = Derived%Y

END SUBROUTINE FGMRES_Threaded
END MODULE simple_prog
```





Target Derived Type

Checking we have correct aliasing information
when all members of a Derived Type have the
TARGET attribute.
As expected, we obtain the following improvements:
- Real_B used to alias with everything. It now aliases
only with POINTER/TARGET variables
- Pointer variables no longer alias with Real_A

Observation:
- Derived%DTN%Y and Derived%Y should not be aliasing.
We can leave this for future work
==========

==========
MODULE simple_prog

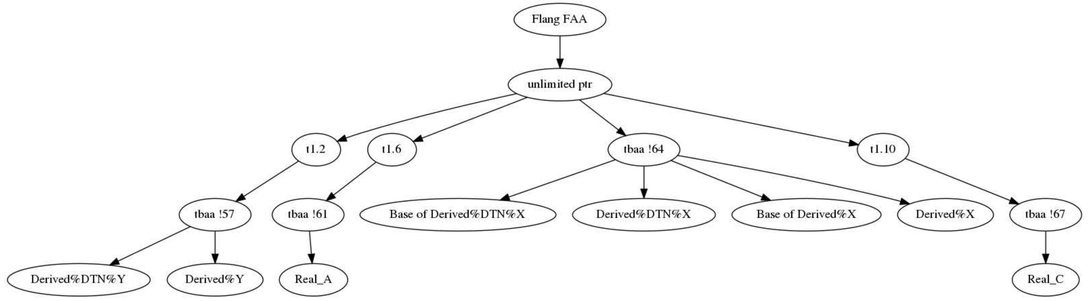
TYPE My_Derived_Type_Nested
Real:: Y
Real, POINTER:: X
END TYPE My_Derived_Type_Nested

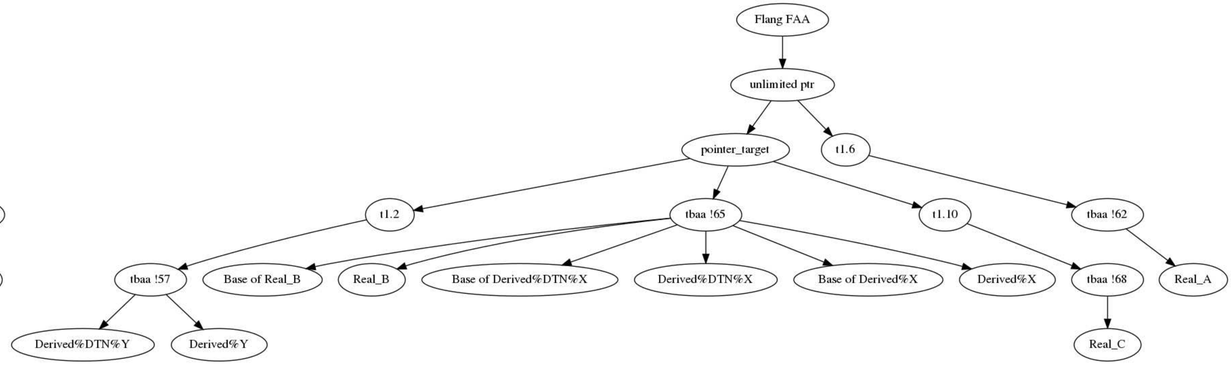
TYPE My_Derived_Type
Real:: Y
Real, POINTER:: X
TYPE (My_Derived_Type_Nested) DTN
END TYPE My_Derived_Type

CONTAINS

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

Real_A = Derived%DTN%Y
Derived%DTN%X = Real_B
Real_B = Derived%X

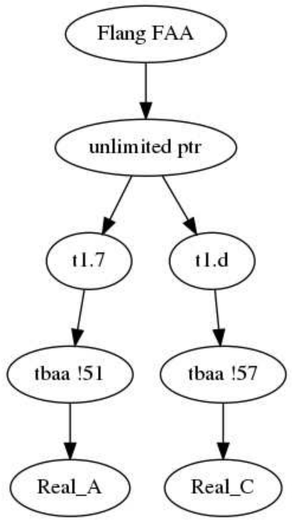


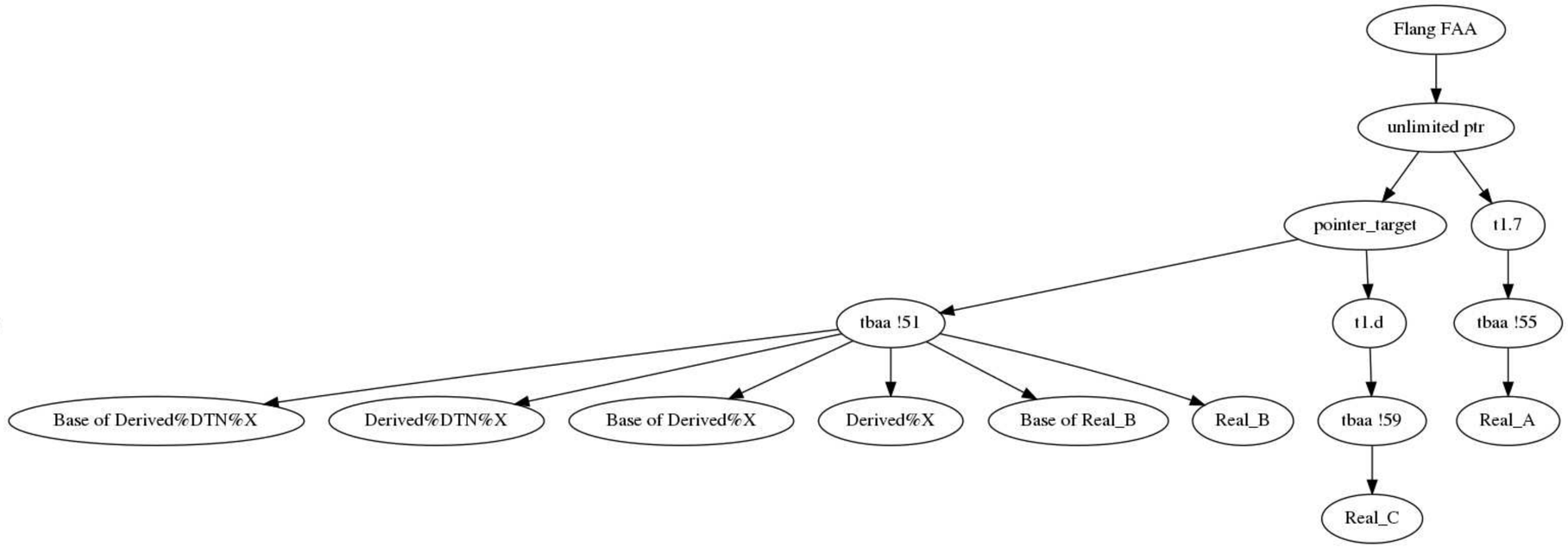


Pointer Derived Type

Checking we have correct aliasing information
when all members of a Derived Type have the
POINTER attribute.
As expected, we obtained the following improvement:
- All the pointers are no longer aliasing with Real_A,
including all the members of the Derived Type marked as
POINTER.
Observation:
- Derived types can't have TARGET members. It's illegal.
==========

==========
```
MODULE simple_prog

TYPE My_Derived_Type_Nested
Real:: X
END TYPE My_Derived_Type_Nested

TYPE My_Derived_Type
Real:: X
TYPE (My_Derived_Type_Nested) DTN
END TYPE My_Derived_Type

CONTAINS

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

Real_A = Derived%DTN%X
Real_B = Derived%X
Real_C = Derived%X

END SUBROUTINE FGMRES_Threaded
END MODULE simple_prog
```





Multiple TARGET variables

All TARGET variables are not aliasing
with each other. This is preserved by
the proposed improvements.
==========

==========
MODULE simple_prog

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

CONTAINS

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

Real_A = Derived%Y
Real_B = Derived%X
Real_C = Real_D

END SUBROUTINE FGMRES_Threaded
END MODULE simple_prog