A variety of AD tools

- Overloading-based single-language
  - CppAD
  - FADBAD
  - MAD
  - AD for Matlab

- Multi-language
  - Adol-C
dco
complex-step

- Source transformation single-language
  - ADIFOR
  - ADiMat

- Multi-language
  - data-flow reversal by recomputation
  - TAF/TAC++

- Compiler-embedded
  - data-flow reversal by storage
  - OpenAD
  - Tapenade

- NAG compiler

Narayanan, Hascoët
Interfacing OpenAD and Tapenade
September 2016
Disturbing observations

Tool development is expensive, but resources are scarce

Sustained flow of new ideas, but little time to implement and measure

Comparing/combining strategies is tough (different platforms, different applications)

Maintenance issues on external front-ends and back-ends
Objectives

OpenAD-Tapenade interoperation

- combining medium-grain components (parsers, individual data-flow analyses, AD transformation, optimizations, back-ends)
- one tool’s component as a backup for the other’s
- or as an addition to the other’s (e.g. pointers+DU-UD, TBR + preaccumulation)
- long-term dream of “à la carte” AD
Starting situation

Two independent tools, reasonably modular:

Natural data exchange format: XAIF (an XML schema)
- XAIF can represent the features of the imperative languages of interest, C, and Fortran.
XAIF forward mode output contains new assignments and derivative propagators.
XAIF reverse mode output contains replacements to perform joint/split mode.
Added components Tap2Xaif and Xaif2Tap in Tapenade
A Fortran file, prepared for OpenAD differentiation:

\begin{verbatim}
C$openad xxx file_start [head.f]
C$openad xxx Template ad_template.f
   subroutine head(x,y)
      use OAD_intrinsics
      double precision, dimension(5), intent(in) :: x
      double precision, dimension(4), intent(out) :: y
C$openad INDEPENDENT(x)
      y=x(1:4)
      call foo(x,y(1),1,3,1)
      call foo(x,y(4),5,1,-2)
C$openad DEPENDENT(y)
   end subroutine

C$openad xxx Template ad_template.f
   subroutine foo(x,y,l,u,s)
      use OAD_intrinsics
      double precision, dimension(5), intent(in) :: x
      double precision, intent(inout) :: y
      integer i,l,u,s
C$openad xxx simple loop
      do i=l,u,s
         y = y*x(i)
      end do
   end subroutine
\end{verbatim}
calls Tapenade Fortran parser, then Tapenade Data-Flow analysis (including activity), then produces XAIF:

```xml
<xaif:CallGraph
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xaif="http://www.mcs.anl.gov/XAIF"
  xsi:schemaLocation="http://www.mcs.anl.gov/XAIF xaif.xsd"
  program_name="Test" />
<xaif:ScopeHierarchy>
  ...
  <xaif:Symbol symbol_id="x" active="1" type="real" />
  ...
</xaif:ScopeHierarchy>
...
<xaif:ControlFlowGraph vertex_id="Unit1" symbol_id="head" />
  ...
  <xaif:BasicBlock vertex_id="B0" scope_id="Scope5" />
    <xaif:SubroutineCall symbol_id="foo" formalArgCount="2" />
      <xaif:Argument position="1"> ...
      <xaif:Argument position="2"> ...
    </xaif:SubroutineCall>
</xaif:ControlFlowGraph>
```
Step 2

```bash
#> oadDriver -p -v -i head.tappre.xaif
    -o head.tappre.xbr.xaif
    -s xaif/schema/ -c inlinable_inlinable_inlinable_inlinable_inlinable
```
calls OpenAD differentiation ("booster")
and produces a new XAIF.
calls Tapenade back-end on the XAIF, sends differentiated source back to OpenAD for final polishing:

```
... CALL OAD_CONVERT(ad_tyc_1, _VALUE_(y))
CALL FOO(ad_tyc_0, ad_tyc_1, 1, 3, 1)
CALL OAD_CONVERT(_VALUE_(x), ad_tyc_0)
...
SUBROUTINE FOO(x, y, l, u, s)
  TYPE(OPENDATY_ACTIVE) x(5)
  TYPE(OPENDATY_ACTIVE) y
  ...
  DO i=l,u,s
    ad_lin_0 = _VALUE_(x(i))
    ad_lin_1 = _VALUE_(y)
    CALL _VALUE_(y) = _VALUE_(y)*_VALUE_(x(i))
    CALL SETDERIV(_DERIV_(ad_prp_1), _DERIV_(y))
    CALL SETDERIV(_DERIV_(ad_prp_2), _DERIV_(x(i)))
    CALL SAX(_DERIV_(ad_prp_1), ad_lin_0, _DERIV_(y))
  ENDDO
...```
Example recap.

- Used Tapenade parser (maintained at INRIA)
- Used Tapenade activity data-flow analysis (robust, interprocedural)
- Used OpenAD differentiation engine (with preaccumulation of partial derivatives)
- Used Tapenade back-end (and a bit of OpenAD’s... )
Further interoperation objectives:

- Explore the symmetric routes
- Combine both sources of analyses
- Split the differentiation machinery further