CamFort:
A verification tool for scientific software

Matthew Danish
RETRACTED

indexing error!
minus sign flipped in array access

units mismatch!
foot-pounds (lbf) vs.
Newtons (N)

NASA/JPL/Corby Waste
CamFort paradigm

• **Lightweight** specification: verify some (not all)
• All specifications are in comments

• **Check** *(does it do what I think it does?)*
• **Infer** *(what does it do?)*
• **Synthesise** *(capture what it does for documentation & future-proofing)*
CamFort specifications 1

Units-of-measure types
Case study 1: energy calculation

```fortran
program energy
  real, parameter :: mass = 3.00, gravity = 9.81, height = 4.20
  real :: potential_energy

  potential_energy = mass * gravity * height
end program energy
```
Finding variables to annotate

```fortran
program energy
  real, parameter :: mass = 3.00, gravity = 9.81, height = 4.20
  real :: potential_energy

  potential_energy = mass * gravity * height
end program energy
```

$ camfort units-suggest energy1.f90

Suggesting variables to annotate with unit specifications in "energy1.f90"

energy1.f90:
  (2:22) mass
  (2:51) height
  (3:11) potential_energy
User inserts unit annotations

```fortran
program energy

  ! unit kg :: mass
  ! unit m/s**2 :: gravity

real, parameter :: mass = 3.00, gravity = 9.81, height = 4.20

! unit kg m**2/s**2 :: potential_energy

real :: potential_energy

potential_energy = mass * gravity * height

end program energy
```
Unit checking and inference

$ camfort units-check energy1.f90
Checking units for "energy1.f90"
energy1.f90: Consistent. 4 variables checked.

$ camfort units-infer energy1.f90
Inferring units for "energy1.f90"
energy1.f90:
  (5:22) unit kg :: mass
  (5:35) unit m / s**2.0 :: gravity
  (5:51) unit m :: height
  (8:11) unit (kg m**2.0) / s**2.0 :: potential_energy
Unit synthesis

```
$ camfort units-synth energy1.f90 out/energy1.f90

Synthesising units for energy1.f90

Writing out/energy1.f90

program energy
  ! unit kg :: mass
  ! unit m/s**2 :: gravity
  ! unit m :: height
  real, parameter :: mass = 3.00, gravity = 9.81, height = 4.20
  ! unit kg m**2/s**2 :: potential_energy
  real :: potential_energy
  potential_energy = mass * gravity * height
end program energy
```
Extended example

```fortran
program energy
  ! = unit kg :: mass
  ! = unit m/s**2 :: gravity
  ! = unit m :: height
  real, parameter :: mass = 5.00, gravity = 9.81, height = 4.20
  ! = unit kg m**2/s**2 :: potential_energy
  real :: potential_energy

  ! = unit 1 :: half
  ! = unit m/s :: velocity
  real, parameter :: half = 0.5, velocity = 4.00
  real :: kinetic_energy, total_energy

  potential_energy = mass * gravity * height
  kinetic_energy = half * mass * velocity

  total_energy = potential_energy + kinetic_energy
end program energy
```

Special unit for "unitless" coefficients

Spot the bug?
Extended example

```
program energy
    ! = unit kg :: mass
    ! = unit m/s**2 :: gravity

$ camfort units-check energy2.f90

Checking units for energy2.f90

energy2.f90 : Inconsistent:
- at 15:18 'kinetic_energy' should be '(kg m**2.0) / s**2.0'
  instead 'kinetic_energy' is '1 kg (m / s)'

   real, parameter :: half = 0.5, velocity = 4.00
   real :: kinetic_energy, total_energy

   potential_energy = mass * gravity * height
   kinetic_energy = half * mass * velocity

   total_energy = potential_energy + kinetic_energy

end program energy
```
Unit aliases

! = unit :: joule = kg m**2/s**2
! = unit joule :: potential_energy
real :: potential_energy
Functions and “polymorphism”

Monomorphic
“inches” to “cm”

```
1  real function inch_to_cm(inch)
2    != unit in :: inch
3    real, intent(in) :: inch
4    != unit (cm / in) :: k
5    real :: k = 2.54
6    inch_to_cm = inch * k
7  end function inch_to_cm
```

Polymorphic
implicitly works for any units

```
1  real function abs(x)
2    real, intent(in) :: x
3    if (x >= 0) abs = x
4    if (x < 0) abs = -x
5  end function abs
```
Functions and “polymorphism”

Polymorphic
“explicit annotations”

```
1  function sqr(x)
2      != unit 'a :: x
3      real, intent(in) :: x
4      != unit 'a**2 :: sqr
5      real :: sqr
6      sqr = x * x
7  end function sqr
```
CamFort specifications 2

Stencil shape specifications
Case Study 2: Heat equation

```fortran
1 subroutine stencils(r1, r2, n, u, v)
2 integer, intent (in) :: n, r1, r2
3 integer, intent (in), dimension(n) :: v
4 integer, intent (out), dimension(n) :: u
5 integer :: i

6 ! Discretisation for heat equation
7 do i=2, n-1
8     u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
9 end do
10 end subroutine stencils
```
Stencil execution

\[ u(i) = r_1 v(i-1) + r_2 v(i) + r_1 v(i+1) \]
Stencil execution

\[ u(i) = r_1 v(i-1) + r_2 v(i) + r_1 v(i+1) \]
Stencil execution

\[ u(i) = r_1 v(i-1) + r_2 v(i) + r_1 v(i+1) \]
Stencil execution

\[ u(i) = r1 \cdot v(i-1) + r2 \cdot v(i) + r1 \cdot v(i+1) \]
Stencil execution

\[ u(i) = r_1v(i-1) + r_2v(i) + r_1v(i+1) \]
Stencil execution

\[ u(i) = r1 * v(i-1) + r2 * v(i) + r1 * v(i+1) \]
Stencil execution

\[ u(i) = r1 \cdot v(i-1) + r2 \cdot v(i) + r1 \cdot v(i+1) \]
Stencil execution

\[ u(i) = r_1 v(i-1) + r_2 v(i) + r_1 v(i+1) \]
Inferring stencil specifications

! Discretisation for heat equation

do i=2, n-1
    u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
end subroutine stencils

$ camfort stencils-infer heat.f90

Inferring stencil specifications for heat.f90

heat.f90
(9:6)-(9:43) stencil readOnce, (centered(depth=1, dim=1)) :: v
Synthesise stencil specifications

! Discretisation for heat equation
do i=2, n-1
   u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
end do
end subroutine stencils

$ camfort stencils-synth heat.f90 out/
Synthesising stencil specifications for heat.f90
Writing out/heat.f90

automatically generated and inserted
Two potential mistakes

8     do i=2, n-1
9        ! stencil readOnce, centered(dim=1, depth=1) :: v
10        u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1) + r1*v(i+1)
11     end do

8     do i=2, n-1
9        ! stencil readOnce, centered(dim=1, depth=1) :: v
10        u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+2)
11     end do
Two potential mistakes

\begin{verbatim}
8     do i=2, n-1
9         != stencil readOnce, centered(dim=1, depth=1) :: v
10        u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1) + r1*v(i+1)
11     end do
\end{verbatim}

Illegal repetition of access pattern

\begin{verbatim}
8     do i=2, n-1
9         != stencil readOnce, centered(dim=1, depth=1) :: v
10        u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
11     end do
\end{verbatim}

Out of bounds stencil access/
Does not conform with the shape.
More advanced specifications

- There are other primitive regions: pointed, forward, and backward
- Two operators for composition: +, *
- Specifications acting on multiple dimensions
All region executions

centered(depth=1,dim=1)

pointed(dim=1)

forward(depth=1,dim=1)

backward(depth=1,dim=1)
All region executions

centered(depth=1, dim=1)

pointed(dim=1)

forward(depth=1, dim=1)

backward(depth=1, dim=1)
All region executions

centered(depth=1, dim=1)

pointed(dim=1)

forward(depth=1, dim=1)

backward(depth=1, dim=1)
All region executions

centered\text{(depth=1, dim=1)}

pointed\text{(dim=1)}

forward\text{(depth=1, dim=1)}

backward\text{(depth=1, dim=1)}
All region executions

centered\((depth=1, dim=1)\)

pointed\((dim=1)\)

forward\((depth=1, dim=1)\)

backward\((depth=1, dim=1)\)
All region executions

centered(depth=1, dim=1)

pointed(dim=1)

forward(depth=1, dim=1)

backward(depth=1, dim=1)
All region executions

- centered\((depth=1, dim=1)\)
- pointed\((dim=1)\)
- forward\((depth=1, dim=1)\)
- backward\((depth=1, dim=1)\)
All region executions

centered(depth=1,dim=1)

pointed(dim=1)

forward(depth=1,dim=1)

backward(depth=1,dim=1)
More advanced specifications

- Two operators for composition: $+, \ast$

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\text{\ast}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\text{=}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\text{=}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]
From Navier-Stokes fluid simulation

\begin{verbatim}
\texttt{du2dx} = ((u(i,j)+u(i+1,j))*(u(i,j)+u(i+1,j)) + \\
gamma*abs(u(i,j)+u(i+1,j))*(u(i,j)-u(i+1,j)) - \\
(u(i-1,j)+u(i,j))*(u(i-1,j)+u(i,j)) - \\
gamma*abs(u(i-1,j)+u(i,j))*(u(i-1,j)-u(i,j)))/\texttt{(4.0*delx)}
\end{verbatim}

\begin{verbatim}
\texttt{duvdy} = ((v(i,j)+v(i+1,j))*(u(i,j)+u(i,j+1)) + \\
gamma*abs(v(i,j)+v(i+1,j))*(u(i,j)-u(i,j+1)) - \\
(v(i,j-1)+v(i+1,j-1))*(u(i,j-1)+u(i,j)) - \\
gamma*abs(v(i,j-1)+v(i+1,j-1))*(u(i,j-1)- u(i,j)))/\texttt{(4.0*dely)}
\end{verbatim}

\begin{verbatim}
\texttt{laplu} = (u(i+1,j)-2.0*u(i,j)+u(i-1,j))/\texttt{delx/delx} + \\
(u(i,j+1)-2.0*u(i,j)+u(i,j-1))/\texttt{delx/delx}
\end{verbatim}

\begin{verbatim}
f(i,j) = u(i,j)+\texttt{del_t*(laplu/Re-du2dx-duvdy)}
\end{verbatim}
\begin{align*}
\text{du2dx} &= ((u(i,j)+u(i+1,j))*(u(i,j)+u(i+1,j))+
& \quad \gamma \cdot \text{abs}(u(i,j)+u(i+1,j))*(u(i,j)-u(i+1,j))-\nonumber \\
& \quad (u(i-1,j)+u(i,j))*(u(i-1,j)+u(i,j))-\nonumber \\
& \quad \gamma \cdot \text{abs}(u(i-1,j)+u(i,j))*(u(i-1,j)-u(i,j))) \nonumber \\
& \quad / (4.0 \cdot \Delta x) \nonumber \\
\text{duvdy} &= ((v(i,j)+v(i+1,j))*(u(i,j)+u(i,j+1))+
& \quad \gamma \cdot \text{abs}(v(i,j)+v(i+1,j))*(u(i,j)-u(i,j+1))-\nonumber \\
& \quad (v(i,j-1)+v(i+1,j-1))*(u(i,j-1)+u(i,j))-\nonumber \\
& \quad \gamma \cdot \text{abs}(v(i,j-1)+v(i+1,j-1))*(u(i,j-1)-
& \quad u(i,j))) / (4.0 \cdot \Delta y) \nonumber \\
\text{laplu} &= (u(i+1,j)-2.0 \cdot u(i,j)+u(i-1,j)) / \Delta x / \Delta x + \nonumber \\
& \quad (u(i,j+1)-2.0 \cdot u(i,j)+u(i,j-1)) / \Delta y / \Delta y \nonumber \\
\text{!} = \text{stencil centered}(\text{depth}=1, \text{dim}=1) * \text{pointed}(\text{dim}=2) + \nonumber \\
& \quad \text{!} = \text{stencil forward}(\text{depth}=1, \text{dim}=1) * \nonumber \\
& \quad \quad \text{!} = \text{stencil backward}(\text{depth}=1, \text{dim}=2) :: u \nonumber \\
\text{f}(i,j) &= \text{u}(i,j) + \text{del_t} * (\text{laplu} / \text{Re} - \text{du2dx} - \text{duvdy}) \nonumber 
\end{align*}

Inferred specification for Navier-Stokes
16  ! stencil centered(depth=1, dim=1) * pointed(dim=2) +
    ← centered(depth=1, dim=2) * pointed(dim=1) :: u
17  ! stencil forward(depth=1, dim=1) *
    ← backward(depth=1, dim=2) :: v

Inferred specification for Navier-Stokes
How effective is it?

• Jacobi method for system of linear equations

\[
! = \text{stencil } \begin{array}{c}
\text{pointed}(\text{dim}=1) \ast \text{centered}(\text{depth}=1, \text{dim}=2, \text{nonpointed}) \\
+ \text{pointed}(\text{dim}=2) \ast \text{centered}(\text{depth}=1, \text{dim}=1, \text{nonpointed}) \\
a(i,j) = (a(i-1,j) + a(i+1,j) + a(i,j+1) + a(i,j-1))/4
\end{array}
\]

• 6,537 ways of getting this wrong
• 24 correct ways (swapping order of +)
• CamFort rejects all wrong and accepts all correct
Future plans

• Multi-file unit analysis: *now supported*
• QuickCheck like test generation
• Dependency derivation (where does this variable get set?)
• First order logical specifications e.g. preconditions
Tell us what you need!

- We want to know about the problems you face.
- Please send us your code!
Follow CamFort updates and access the code on camfort.github.io

or subscribe to the mailing list at: lists.cam.ac.uk/mailman/listinfo/cl-camfort

Thank you!

Mistral Contrastin  Matthew Danish  Dominic Orchard  Andrew Rice
References

1. https://en.wikipedia.org/wiki/Mars_Climate_Orbiter