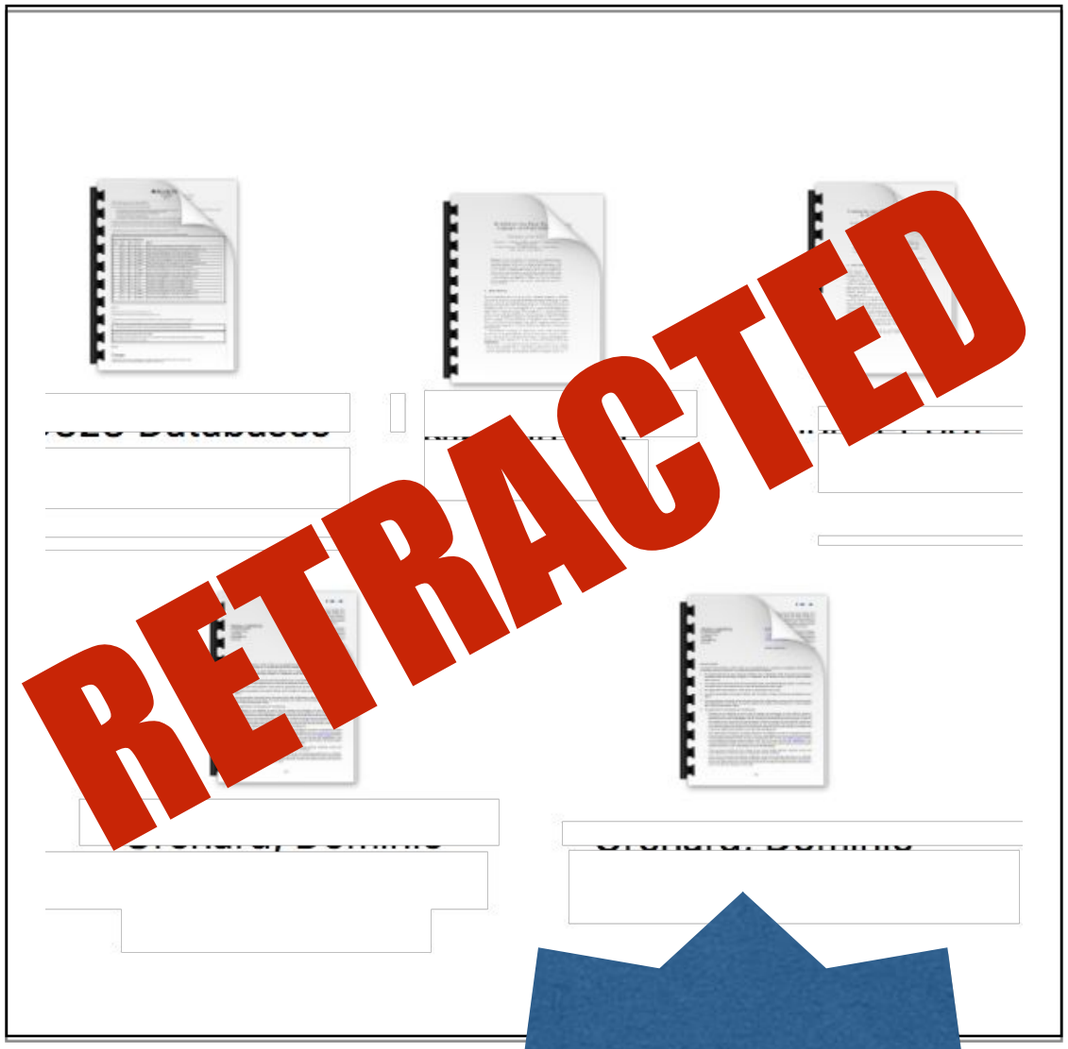
CamFort:

A verification tool for scientific software

Matthew Danish

UNIVERSITY OF CAMBRIDGE
University of Kent

CamFort



indexing error!
minus sign flipped in array access

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

NASA/JPL/Corby Waste

UNIVERSITY OF CAMBRIDGE

University of Kent

CamFort

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

UNIVERSITY OF CAMBRIDGE University of Kent

CamFort

Case study 1: energy calculation

```
1  program energy
2    real, parameter :: mass = 3.00, gravity = 9.81, height = 4.20
3    real :: potential_energy
4
5    potential_energy = mass * gravity * height
6  end program energy
```

Finding variables to annotate

```fortran
1  program energy
2    real, parameter :: mass = 3.00, gravity = 9.81, height = 4.20
3    real :: potential_energy
4
5    potential_energy = mass * gravity * height
6  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

```
1  program energy
2     != unit kg :: mass
3     != unit m/s**2 :: gravity
4     real, parameter :: mass = 3.00, gravity = 9.81, height = 4.20
5     != unit kg m**2/s**2 :: potential_energy
6     real :: potential_energy
7
8     potential_energy = mass * gravity * height
9  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
```

out/energy.f90

```
1   program energy
2     != unit kg :: mass
3     != unit m/s**2 :: gravity
4     != unit m :: height
5     real, parameter :: mass = 3.00, gravity = 9.81, height = 4.20
6     != unit kg m**2/s**2 :: potential_energy
7     real :: potential_energy
8
9     potential_energy = mass * gravity * height
10  end program energy
```

CamFort-generated comment

UNIVERSITY OF CAMBRIDGE University of Kent CamFort

Extended example

```
1   program energy
2      != unit kg :: mass
3      != unit m/s**2 :: gravity
4      != unit m :: height
5      real, parameter :: mass = 5.00, gravity = 9.81, height = 4.20
6      != unit kg m**2/s**2 :: potential_energy
7      real :: potential_energy
8
9      != unit 1 :: half
10     != unit m/s :: velocity
11     real, parameter :: half = 0.5, velocity = 4.00
12     real :: kinetic_energy, total_energy
13
14     potential_energy = mass * gravity * height
15     kinetic_energy = half * mass * velocity
16
17     total_energy = potential_energy + kinetic_energy
18  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
```

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

```
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
```

BUG! should be velocity**2

Unit aliases

```
!= unit :: joule = kg m**2/s**2
!= unit joule :: potential_energy
real :: potential_energy
```

Functions and “polymorphism”

Monomorphic

“inches” to “cm”

```
real function inch_to_cm(inch)
  != unit in :: inch
  real, intent(in) :: inch
  != unit (cm / in) :: k
  real :: k = 2.54
  inch_to_cm = inch * k
end function inch_to_cm
```

Polymorphic

implicitly works for any units

```
real function abs(x)
  real, intent(in) :: x

  if (x >= 0) abs = x
  if (x < 0) abs = -x
end function abs
```

Functions and “polymorphism”

Polymorphic

“explicit annotations”

```
function sqr(x)
  != unit 'a :: x
  real, intent(in) :: x
  != unit 'a**2 :: sqr
  real :: sqr
  sqr = x * x
end function sqr
```

CamFort specifications 2

Stencil shape specifications

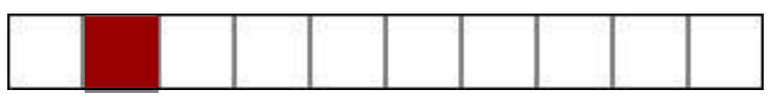
Case Study 2: Heat equation

```
subroutine stencils(r1, r2, n, u, v)
  integer, intent (in) :: n, r1, r2
  integer, intent (in), dimension(n) :: v
  integer, intent (out), dimension(n) :: u
  integer :: i

  ! 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
```

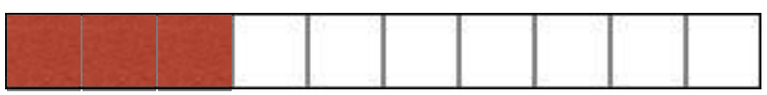
Stencil execution

```
u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
```

u



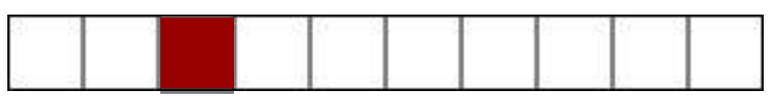
v



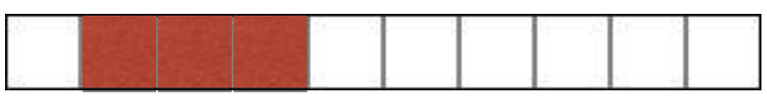
Stencil execution

```
u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
```

u



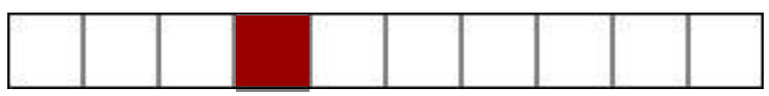
v



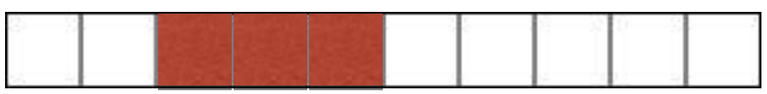
Stencil execution

```
u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
```

u



v



Stencil execution

```
u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
```

u

v

Stencil execution

```
u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
```

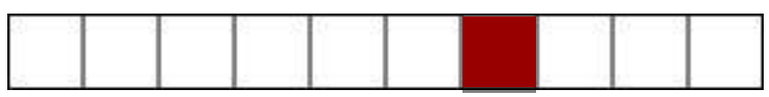
u

v

Stencil execution

```
u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
```

u

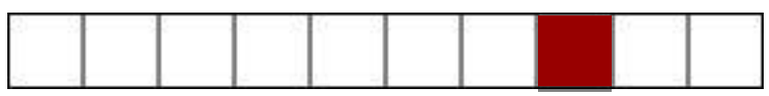


v

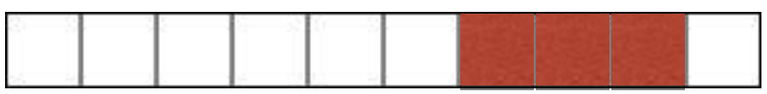
Stencil execution

```
u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
```

u



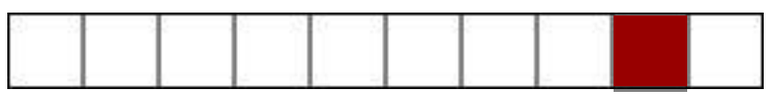
v



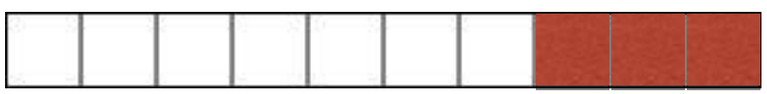
Stencil execution

```
u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
```

u



v



Inferring stencil specifications

```
6
7     ! Discretisation for heat equation
8     do i=2, n-1
9       u(i) = r1*v(i-1) + r2*v(i) + r1*v(i+1)
10    end do
11  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

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

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

automatically generated and inserted

```
7     ! Discretisation for heat equation
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
```

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

```
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
```

Illegal repetition of access pattern

```
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
```

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(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)

All region executions

centered(depth=1,dim=1)

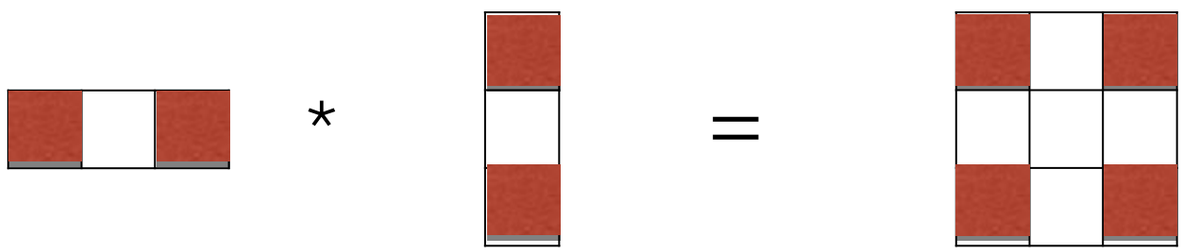
pointed(dim=1)

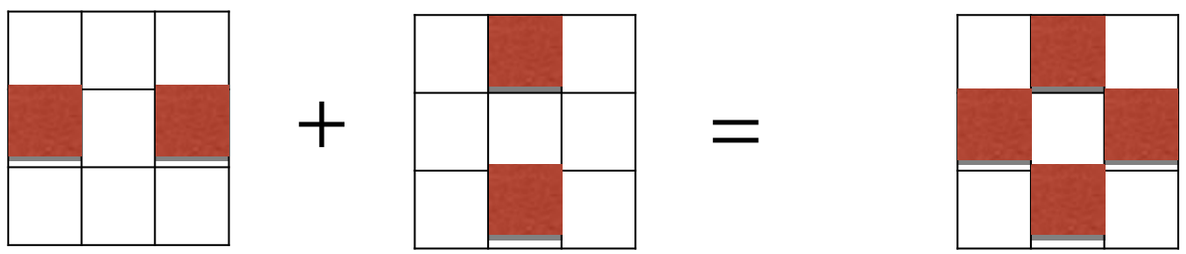
forward(depth=1,dim=1)

backward(depth=1,dim=1)

More advanced specifications

- Two operators for composition: **+**, *





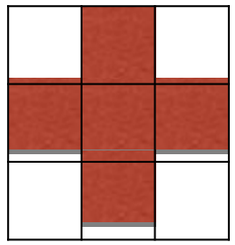
```
du2dx = ((u(i,j)+u(i+1,j))*(u(i,j)+u(i+1,j))+      &
  gamma*abs(u(i,j)+u(i+1,j))*(u(i,j)-u(i+1,j))-    &
  (u(i-1,j)+u(i,j))*(u(i-1,j)+u(i,j))-             &
  gamma*abs(u(i-1,j)+u(i,j))*(u(i-1,j)-u(i,j)))    &
  /(4.0*delx)

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))) / (4.0*dely)

laplu = (u(i+1,j)-2.0*u(i,j)+u(i-1,j))/delx/delx+ &
  (u(i,j+1)-2.0*u(i,j)+u(i,j-1))/dely/dely

f(i,j) = u(i,j)+del_t*(laplu/Re-du2dx-duvdy)
```

From Navier-Stokes fluid simulation

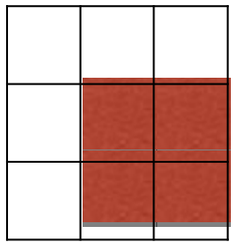
```
du2dx = ((u(i,j)+u(i+1,j))*(u(i,j)+u(i+1,j))+      &
  gamma*abs(u(i,j)+u(i+1,j))*(u(i,j)-u(i+1,j))-    &
  (u(i-1,j)+u(i,j))*(u(i-1,j)+u(i,j))-             &
  gamma*abs(u(i-1,j)+u(i,j))*(u(i-1,j)-u(i,j)))    &
  /(4.0*delx)

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))) / (4.0*dely)

laplu = (u(i+1,j)-2.0*u(i,j)+u(i-1,j))/delx/delx+ &
  (u(i,j+1)-2.0*u(i,j)+u(i,j-1))/dely/dely

!= stencil centered(depth=1,dim=1)*  pointed(dim=2) +
    ↪   centered(depth=1,dim=2)*  pointed(dim=1) :: u
!= stencil forward(depth=1,dim=1) *
    ↪   backward(depth=1,dim=2) :: v
f(i,j) = u(i,j)+del_t*(laplu/Re-du2dx-duvdy)
```

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
```



:: u

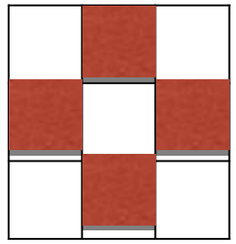


:: v

Inferred specification for Navier-Stokes

How effective is it?

- Jacobi method for system of linear equations



```
!= stencil   pointed(dim=1)*centered(depth=1,dim=2,nonpointed)
           +  pointed(dim=2)*centered(depth=1,dim=1,nonpointed)
a(i,j) = (a(i-1,j)+a(i+1,j)+a(i,j+1)+a(i,j-1))/4
```

- 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

UNIVERSITY OF CAMBRIDGE University of Kent

CamFort

References

1. https://en.wikipedia.org/wiki/Mars_Climate_Orbiter
2. Z. Merali, *Computational science: Error, why scientific programming does not compute*, Nature 467 (2010), no. 7317, 775–777.