```
model {
# chlamydia informative priors
r.phi ~ dbin(phi,129)
lambda.c ~ dnorm(0.743,193)
dur.symp ~ dunif(0.077,0.15)  # 4 - 8 weeks
lambda.t <- 1 / cut(dur.symp)

# Prospective PID analysis constant progression rate
# Likelihood
for (s in 1:4) {
 for (i in 1:2) {
  r[s,i] ~ dbin(p[s,i],n[s,i])
  }
 }
# transistion probability calulations
for (s in 1:4) {
 for (i in 1:2) {
  for(j in 1:3) {
   theta[s,1,index[i,j]] <- lambda[s,1,i,j]
   theta[s,2,index[i,j]] <- lambda[s,2,i,j]
   }
  }
 lambda[s,1,1,1] <- - lambda[s,1,1,2] - lambda[s,1,1,3]
 lambda[s,1,1,3] <- theta.CTpos1[s]
 lambda[s,1,2,2] <- - lambda[s,1,2,1] - lambda[s,1,2,3]
 lambda[s,1,2,3] <- theta.CTneg[s]

 lambda[s,2,1,1] <- - lambda[s,2,1,2] - lambda[s,2,1,3]
 lambda[s,2,1,3] <- theta.CTpos2[s]
 lambda[s,2,2,2] <- - lambda[s,2,2,1] - lambda[s,2,2,3]
 lambda[s,2,2,3] <- theta.CTneg[s]
 }

# Infection rate
for (s in 1:2) {
 lambda[s,1,2,1] <- 0.0
 lambda[s,2,2,1] <- 0.0
 }

for (s in 3:4) {
 lambda[s,1,2,1] <- 0.0
 lambda[s,2,2,1] <- 0.0
 }

# Clearance + treatment rate
lambda[1,1,1,2] <- lambda.c + 0.64
lambda[1,2,1,2] <- lambda.c + 0.64

lambda[2,1,1,2] <- lambda.c + 0.43
lambda[2,2,1,2] <- lambda.c + 0.43

lambda[3,1,1,2] <- lambda.c + 0.32
lambda[3,2,1,2] <- lambda.c + 0.32

lambda[4,1,1,2] <- lambda.c + 0.32
lambda[4,2,1,2] <- lambda.c + 0.32

# Models for rates
for (s in 1:4) {
 theta.CTpos1[s] <- alpha[s] + delta[1]
```

```
 theta.CTpos2[s] <- alpha[s] + delta[2]
 theta.CTneg[s] <- alpha[s]
 }

# Other data
r.sch ~ dbin(pi.sch,n.sch) # Scholes - ct prevalence
r.ost ~ dbin(pi.ost,n.ost) # Ostergaard - ct prevalence

# Priors
delta[1] ~ dexp(0.00001)
delta[2] ~ dexp(0.00001)
for (s in 1:4) {
 alpha[s] ~ dexp(0.00001)
 }
pi.sch ~ dbeta(1,1)
pi.ost ~ dbeta(1,1)
phi ~ dbeta(1,1)

# Calculation of parameters in the likekihood
for (s in 1:4) {
 solution1d[s,1,1:dim] <-
three.state(init[1:dim],time,theta[s,1,1:n.par],
                    origin, tol)
 solution1d[s,2,1:dim] <-
three.state(init[1:dim],time,theta[s,2,1:n.par],
                    origin, tol)
 for (i in 1:2) {
  for (z in 1:6) {
   vectorforwbdev[s,i,z] <- solution1d[s,1,z]
   vectorforwbdev[s,i,z+6] <- solution1d[s,2,z]
   }
  vectorforwbdev[s,i,13] <- round(t[s] * 365)
  vectorforwbdev[s,i,14] <- B
  vectorforwbdev[s,i,15] <- lambda.c
  vectorforwbdev[s,i,16] <- lambda.t
  vectorforwbdev[s,i,17] <- phi
  vectorforwbdev[s,i,18] <- clinorscreen[s]
  vectorforwbdev[s,i,19] <- caseprop[s,i]

  p[s,i] <- generatep(vectorforwbdev[s,i,1:19])
  }
 caseprop[s,2] <- 0
 }
caseprop[1,1] <- 1
caseprop[2,1] <- 1
caseprop[3,1] <- pi.sch
caseprop[4,1] <- pi.ost

clinorscreen[1] <- 0 # clinic based study
for (s in 2:4) {
 clinorscreen[s] <- 1   # screening studies
 }

# Residual Deviance
for (s in 1:4) {
 for (i in 1:2) {
  dev[s,i] <- 2 * (r[s,i] * log(r[s,i] / (p[s,i] * n[s,i])) +
                  (n[s,i] - r[s,i]) * log((n[s,i] - r[s,i]) /
                  (n[s,i] - (n[s,i] * p[s,i]))))
  }
 }
```

```
dev.sch <- 2 * (r.sch * log(r.sch / (pi.sch * n.sch)) +
               (n.sch - r.sch) * log((n.sch - r.sch) /
               (n.sch - (n.sch * pi.sch))))

dev.ost <- 2 * (r.ost * log(r.ost / (pi.ost * n.ost)) +
               (n.ost - r.ost) * log((n.ost - r.ost) /
               (n.ost - (n.ost * pi.ost))))

sumdev <- sum(dev[ , ]) + dev.sch + dev.ost


# Results
kappa  <- (1 - phi) * (
          (1 - (exp( - (lambda.c + delta[1]) * (B / 365)))) *
           delta[1] / (lambda.c + delta[1]) +
           exp( - (lambda.c + delta[1]) * (B / 365)) *
           delta[2] / (lambda.c + delta[2])
           ) +
           phi * (
          (1 - (exp( - (lambda.t + delta[1]) * (B / 365)))) *
           delta[1] / (lambda.t + delta[1]) +
           exp( - (lambda.t + delta[1]) * (B / 365)) *
           delta[2] / (lambda.t + delta[2])
           )

# proportion of PIDs prevented by annual screening
# assumes two-week delay between test and treatment
for (i in 1:B) {
 temp1[i] <- phi * (
             ((1 - exp(-(lambda.t + delta[1]) * (i/365))) -
              (1 - exp(-(lambda.t + delta[1]) * ((i-1)/365)))) *
              (delta[1] / (lambda.t + delta[1])) *
              (1 - (max(0,(i-14)) / 365))) +
             (1 - phi) * (
             ((1 - exp(-(lambda.c + delta[1]) * (i/365))) -
              (1 - exp(-(lambda.c + delta[1]) * ((i-1)/365)))) *
              (delta[1] / (lambda.c + delta[1])) *
              (1 - (max(0,(i-14)) / 365)))
 }
for (i in B+1:379) {
 temp1[i] <- phi * (
             ((1 - exp(-(lambda.t + delta[2]) * (i/365))) -
              (1 - exp(-(lambda.t + delta[2]) * ((i-1)/365)))) *
              (delta[2] / (lambda.t + delta[2])) *
              (1 - (max(0,(i-14)) / 365))) +
             (1 - phi) * (
             ((1 - exp(-(lambda.c + delta[2]) * (i/365))) -
              (1 - exp(-(lambda.c + delta[2]) * ((i-1)/365)))) *
              (delta[2] / (lambda.c + delta[2])) *
              (1 - (max(0,(i-14)) / 365)))
 }
prop.prevent <- 1 - (sum(temp1[ ]) / kappa)

# Bayesian p-value
test <- delta[1] - delta[2]
B.p <- step(test)
}

# Data
list(
```

```
# PID (1 month) prospective
# 1. Rees
# 2. POPI
# 3. Scholes
# 4. Ostergaard

r=structure(.Data=c(
8,3,
7,1,
33,7,
20,9
),.Dim=c(4,2)),

n=structure(.Data=c(
67,62,
74,63,
1598,645,
487,443
),.Dim=c(4,2)),
t = c(0.125,1,1,1),
B = 60,   # If this were set above 365 WBDEV code would need changing
r.sch = 44,
n.sch = 645,
r.ost = 43,
n.ost = 867,
r.phi=30,
time = 0.00274,

# forward equations
dim=6,origin=0,tol=1.0E-4, init=c(1,0,0, 0,1,0),n.par=6,
index=structure(.Data=c(1,2,3,
                        4,5,6), .Dim=c(2,3))
)


# Initial values - 1
list(
# Prospective
phi = 0.23, lambda.c = 0.74, delta = c(0.2,0.1),
alpha = c(0.01,0.01,0.01,0.01),
pi.sch = 0.051, pi.ost = 0.07, dur.symp = 0.1
)

# Initial values - 2
list(
# Prospective
phi = 0.7, lambda.c = 2, delta = c(0.1,0.6),
alpha = c(0.15,0.15,0.15,0.15),
pi.sch = 0.2, pi.ost = 0.2, dur.symp = 0.145
)
```

WBDiff Program to calculate the transition probabilities

```
MODULE WBDiffThreeState;

 IMPORT
  WBDiffODEMath,
  Math;

 TYPE
  Equations = POINTER TO RECORD (WBDiffODEMath.Equations) END;
```

```
  Factory = POINTER TO RECORD (WBDiffODEMath.Factory) END;

 CONST

  nEq = 6;
  nSt = 4; (* one higher as arrays start at zero*)


 VAR
  fact-: WBDiffODEMath.Factory;

 PROCEDURE (e: Equations) Derivatives (IN theta, C: ARRAY OF REAL; n: INTEGER; t:
REAL;
                                       OUT dCdt: ARRAY OF REAL);
 VAR

  index: ARRAY nSt, nSt OF INTEGER;

 BEGIN

(* define index of parameters (look-up table) *)
  index[1,1] := 0;
  index[1,2] := 1;
  index[1,3] := 2;
  index[2,1] := 3;
  index[2,2] := 4;
  index[2,3] := 5;

(* define system of nEq Differential Equations *)
  dCdt[index[1,1]]:= C[index[1,1]]*theta[index[1,1]] + C[index[1,2]]*theta[index[2,1]];
  dCdt[index[1,2]]:= C[index[1,1]]*theta[index[1,2]] + C[index[1,2]]*theta[index[2,2]];
  dCdt[index[1,3]]:= C[index[1,1]]*theta[index[1,3]] + C[index[1,2]]*theta[index[2,3]];

  dCdt[index[2,1]]:= C[index[2,1]]*theta[index[1,1]] + C[index[2,2]]*theta[index[2,1]];
  dCdt[index[2,2]]:= C[index[2,1]]*theta[index[1,2]] + C[index[2,2]]*theta[index[2,2]];
  dCdt[index[2,3]]:= C[index[2,1]]*theta[index[1,3]] + C[index[2,2]]*theta[index[2,3]];


  END Derivatives;

  PROCEDURE (equations: Equations) SecondDerivatives (IN theta, x: ARRAY OF REAL;
                                  numEq: INTEGER; t: REAL; OUT d2xdt2: ARRAY OF REAL);
  BEGIN
   HALT(126)
  END SecondDerivatives;

PROCEDURE (equations: Equations) Jacobian (IN theta, x: ARRAY OF REAL;
                    numEq: INTEGER; t: REAL; OUT jacob: ARRAY OF ARRAY OF REAL);
  BEGIN
   HALT(126)
  END Jacobian;

PROCEDURE (f: Factory) New (option: INTEGER): WBDiffODEMath.GraphNode;
  VAR
   equations: Equations;
   node: WBDiffODEMath.GraphNode;
  BEGIN
   NEW(equations);
   node := WBDiffODEMath.New(equations, nEq);
   RETURN node
```

```
 END New;

 PROCEDURE Install*;
 BEGIN
  WBDiffODEMath.Install(fact)
 END Install;

 PROCEDURE Init;
 VAR
  f: Factory;
 BEGIN
  NEW(f); fact := f
 END Init;

BEGIN
 Init
END WBDiffThreeState.
```

WBDEV code to calcuate the Binomial likelihood probabilities of PID for the 2-rate model

```
MODULE WBDevgeneratep;

 IMPORT
  WBDevScalar,
  Math;

 TYPE
   Function = POINTER TO RECORD (WBDevScalar.Node) END;
   Factory = POINTER TO RECORD (WBDevScalar.Factory) END;

 VAR
  fact-: WBDevScalar.Factory;

 PROCEDURE (func: Function) DeclareArgTypes (OUT args: ARRAY OF CHAR);
 BEGIN
  args := "v";
 END DeclareArgTypes;

 PROCEDURE calculation (func: Function; OUT output: REAL);
 VAR
  term1, term2, omega, omega_cum,pi_hi, pi_low:          ARRAY 366 OF REAL;
  pi:                                                    ARRAY 4,366+1 OF REAL;

  p_hi,  p_low:                                          ARRAY 3,4 OF REAL;
  H,B,h,i,b:                                             INTEGER;
  Hin, Bin, lambdaC, lambdaT, phi, clinicorscreen, caseprop:  REAL;
 BEGIN
  (* Read in the parameter values *)
  p_hi[1,1] := func.arguments[0][0].Value();
  p_hi[1,2] := func.arguments[0][1].Value();
  p_hi[1,3] := func.arguments[0][2].Value();
  p_hi[2,1] := func.arguments[0][3].Value();
  p_hi[2,2] := func.arguments[0][4].Value();
  p_hi[2,3] := func.arguments[0][5].Value();
  p_low[1,1] := func.arguments[0][6].Value();
  p_low[1,2] := func.arguments[0][7].Value();
  p_low[1,3] := func.arguments[0][8].Value();
```

```
  p_low[2,1] := func.arguments[0][9].Value();
  p_low[2,2] := func.arguments[0][10].Value();
  p_low[2,3] := func.arguments[0][11].Value();

  Hin := func.arguments[0][12].Value();
  Bin := func.arguments[0][13].Value();
  lambdaC := func.arguments[0][14].Value();
  lambdaT := func.arguments[0][15].Value();
  phi := func.arguments[0][16].Value();
  clinicorscreen := func.arguments[0][17].Value();
  caseprop := func.arguments[0][18].Value();

  (* Converts H and B to integer format *)
  FOR i:= 1 TO 5000 DO
   IF (Hin = i) THEN;
    H := i;
   END;
  END;
  FOR i:= 1 TO 5000 DO
   IF (Bin = i) THEN;
    B := i;
   END;
  END;

  (* Sets B to equal H if follow-up time is shorter than B - Note the program would need
changing if a
     screening study with a follow-up time shorter than B was included *)
  IF (H < B) THEN;
   B := H;
  END;

  (* Calculates the proportion of cases in screening studies infected in the last c = 1 to C days
*)
  IF (clinicorscreen = 1) THEN;
   omega_cum[0] := 0;
   FOR b := 1 TO B DO
    omega[b] := phi * (Math.Exp( - lambdaT * (b - 1) / 365) - Math.Exp( - lambdaT * b / 365)) +
                (1 - phi) * (Math.Exp( - lambdaC * (b - 1) / 365) - Math.Exp( - lambdaC * b /
365));                          omega_cum[b] := omega_cum[b-1] + omega[b];
   END;

  (* specifies the proportion of women in each state at time zero *)
   pi_hi[0] := caseprop * omega_cum[B];
  ELSE
   pi_hi[0] := caseprop;
  END;
  pi[1,0] := caseprop;
  pi[2,0]:= 1 - pi[1,0];
  pi[3,0] := 0;
  pi_low[0] := pi[1,0] - pi_hi[0];

  (* main analysis *)
  FOR h := 1 TO B DO;
   pi[1,h] := pi_low[h-1] * p_low[1,1] + pi_hi[h-1] * p_hi[1,1] + pi[2,h-1] * p_hi[2,1];
   pi[2,h] := pi[2,h-1] * p_hi[2,2] + pi_low[h-1] * p_low[1,2] + pi_hi[h-1] * p_hi[1,2];
   pi[3,h]:= 1 - pi[2,h] - pi[1,h];

   IF (clinicorscreen =0) THEN;
    pi_hi[h] := pi[1,h];
    pi_low[h] := 0
```

```
   END;

   IF (clinicorscreen =1) THEN;
    term1[h] := 0;
    FOR i := h TO B DO
     term1[h] := term1[h] + omega[B+1-i] * pi[1,0] * Math.Power(p_hi[1,1],h);
    END;
    term2[1] := 0;
    IF (h >= 2) THEN;
     FOR i := 1 TO h DO
      term2[h] := term2[h-1] + (pi[2,i-1] * p_hi[2,1]) * Math.Power(p_hi[1,1],h-i);
     END;
    END;
    pi_hi[h] := term1[h] + term2[h];
    pi_low[h] := pi[1,h] - pi_hi[h];
   END;
  END;

IF (H > B) THEN;
   FOR h := B+1 TO H DO
    pi[1,h] := pi_low[h-1] * p_low[1,1] + pi_hi[h-1] * p_hi[1,1] + pi[2,h-1] * p_hi[2,1];
    pi[2,h] := pi[2,h-1] * p_hi[2,2] + pi_low[h-1] * p_low[1,2] + pi_hi[h-1] * p_hi[1,2];
    pi[3,h]:= 1 - pi[2,h] - pi[1,h];

    term1[h] := 0;
    FOR i := h+1-B TO h DO
     term2[h] := term2[h-1] + (pi[2,i-1] * p_hi[2,1]) * Math.Power(p_hi[1,1],h-i);
    END;
    pi_hi[h] := term1[h] + term2[h];
    pi_low[h] := pi[1,h] - pi_hi[h];
   END;
  END;
  output := pi[3,H];

 END calculation;



 PROCEDURE (func: Function) Evaluate (OUT value: REAL);
 VAR
  output:                    REAL;

 BEGIN
  calculation(func, output);
  value := output;
 END Evaluate;

 PROCEDURE (f: Factory) New (option: INTEGER): Function;
 VAR
  func: Function;
 BEGIN
  NEW(func); func.Initialize; RETURN func;
 END New;

 PROCEDURE Install*;
 BEGIN
  WBDevScalar.Install(fact);
 END Install;

 PROCEDURE Init;
```

```
 VAR
  f: Factory;
 BEGIN
  NEW(f); fact := f;
 END Init;

BEGIN
 Init;
          END WBDevgeneratep.
```