gNuSMV

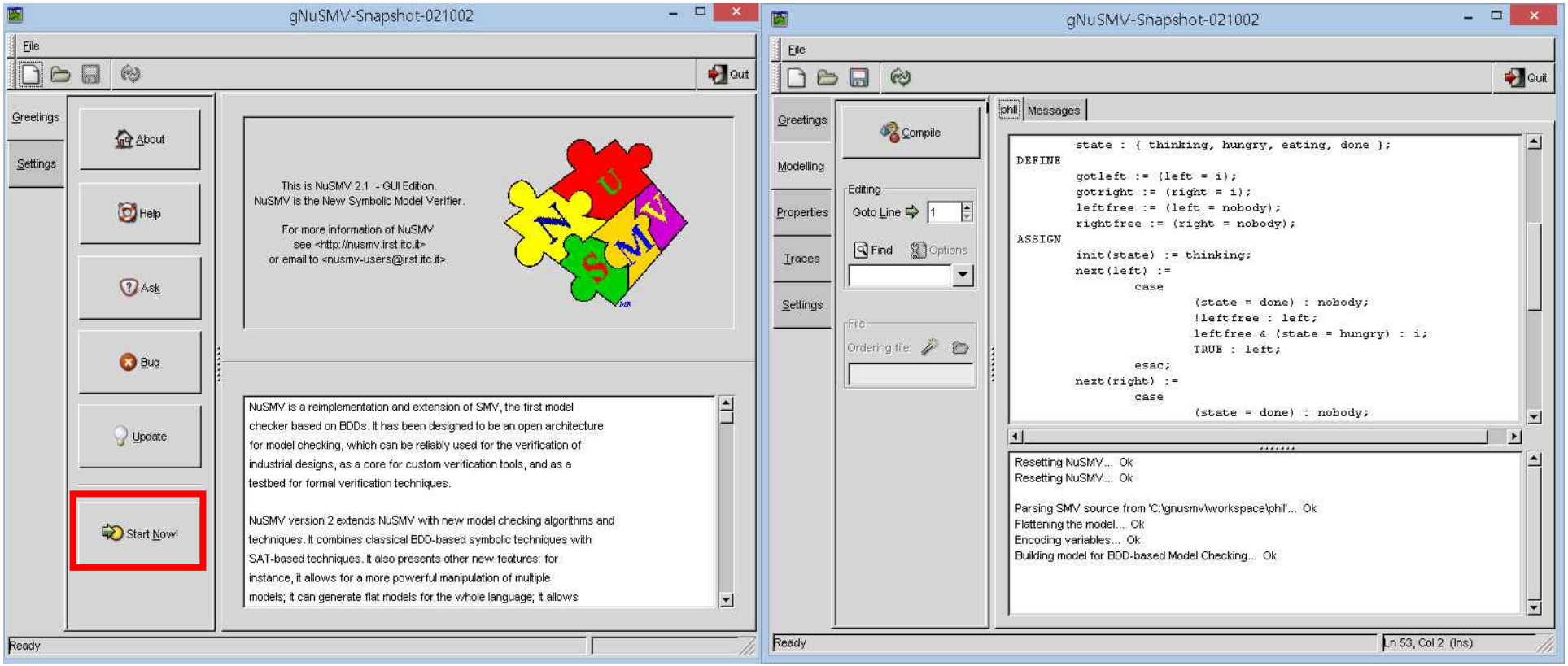
DMS Lab

Lim Dam-sub

Oh Jun

gNuSMV

- http://nusmv.fbk.eu/gnusmv/



Philosopher Code

```
MODULE philosopher(i, left, right)
VAR
        state : { thinking, hungry, eating, done };
DEFINE
        gotleft := (left = i);
        gotright := (right = i);
        leftfree := (left = nobody);
        rightfree := (right = nobody);
ASSIGN
        init(state) := thinking;
        next(left) :=
                case
                        (state = done) : nobody;
                        !leftfree : left;
                        leftfree & (state = hungry) : i;
                        TRUE : left;
                esac;
        next(right) :=
                case
                        (state = done) : nobody;
                        !gotleft : right;
                        !rightfree : right;
                        rightfree & (state = hungry) : i;
                        TRUE : right;
                esac;
        next(state) :=
                case
                        (state = thinking) : {thinking, hungry};
                        (state = hungry) & gotleft & gotright : eating;
                        (state = hungry) : hungry;
                        (state = eating) : {eating, done};
                        (state = done) : thinking;
                        TRUE : state;
                esac;
```

Done state is used to drop the forks properly

macro

```
MODULE main
VAR
        forks : array 0..3 of {nobody, 0, 1, 2, 3};
        p0 : process philosopher(0, forks[0], forks[1]);
        p1 : process philosopher(1, forks[1], forks[2]);
        p2 : process philosopher(2, forks[2], forks[3]);
        p3 : process philosopher(3, forks[3], forks[0]);
ASSIGN
        init(forks[0]) := nobody;
        init(forks[1]) := nobody;
        init(forks[2]) := nobody;
        init(forks[3]) := nobody;
```

Who has the fork

```
FAIRNESS
        running
SPEC
        AG((state = eating) -> (gotleft)&(gotright))
SPEC
        AG((state = hungry) -> AF(state = eating))
```

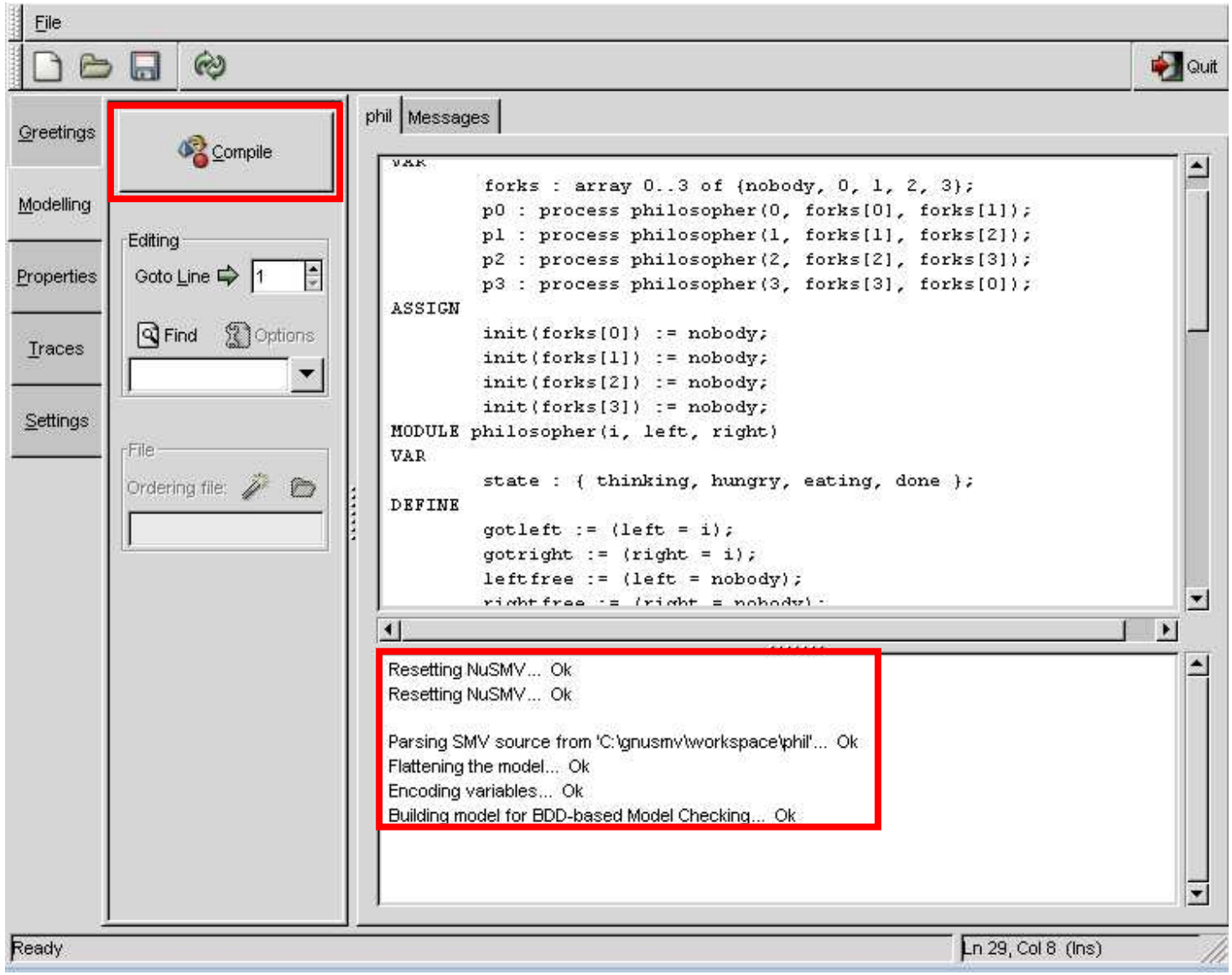
Must have both forks to eat

If hungry, must eventually get to eat

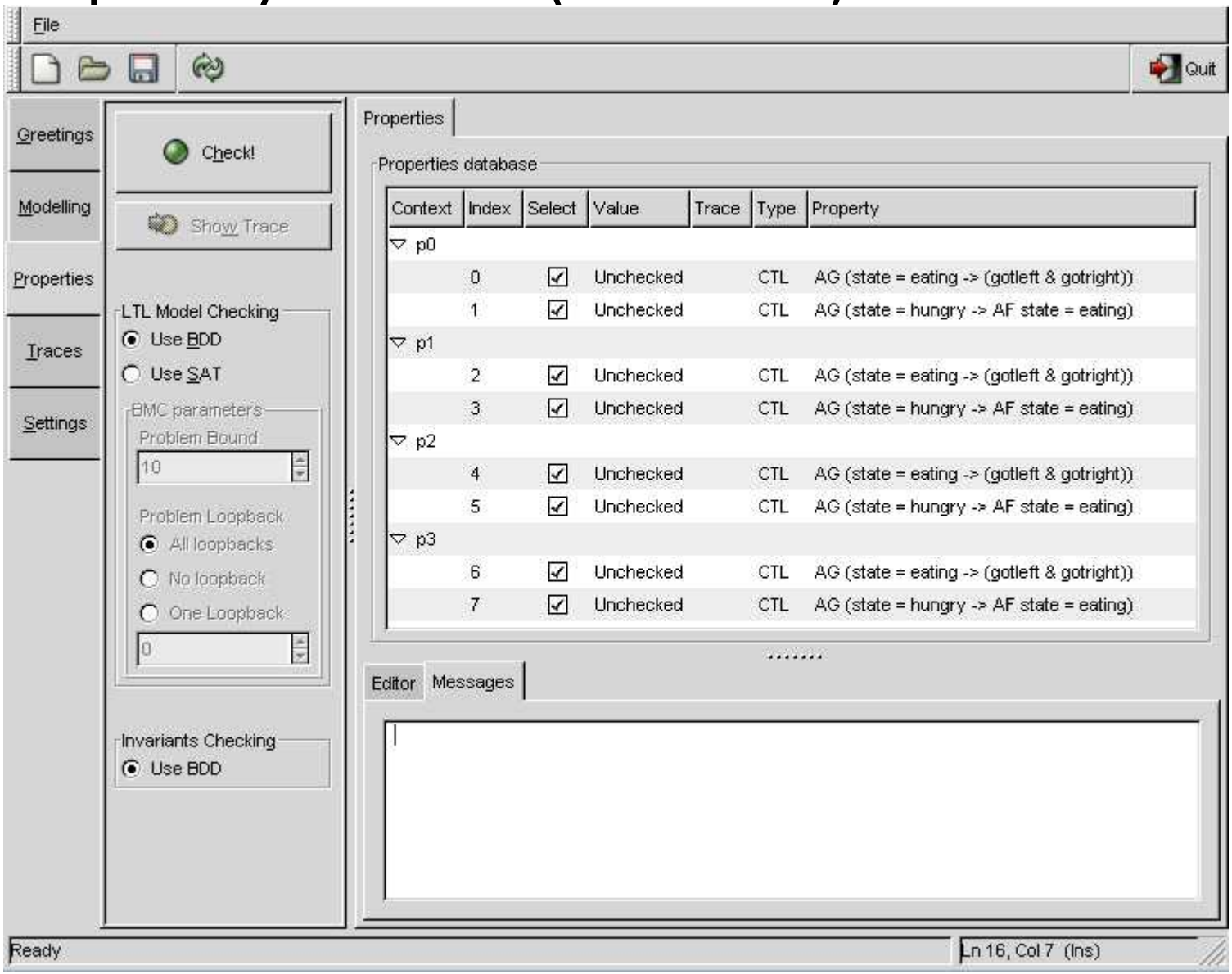
AG((eating) -> (left)&(right)) : eating remains true until left & right true)

AG(hungry) -> AF (eating) : any state will be eventually eating state

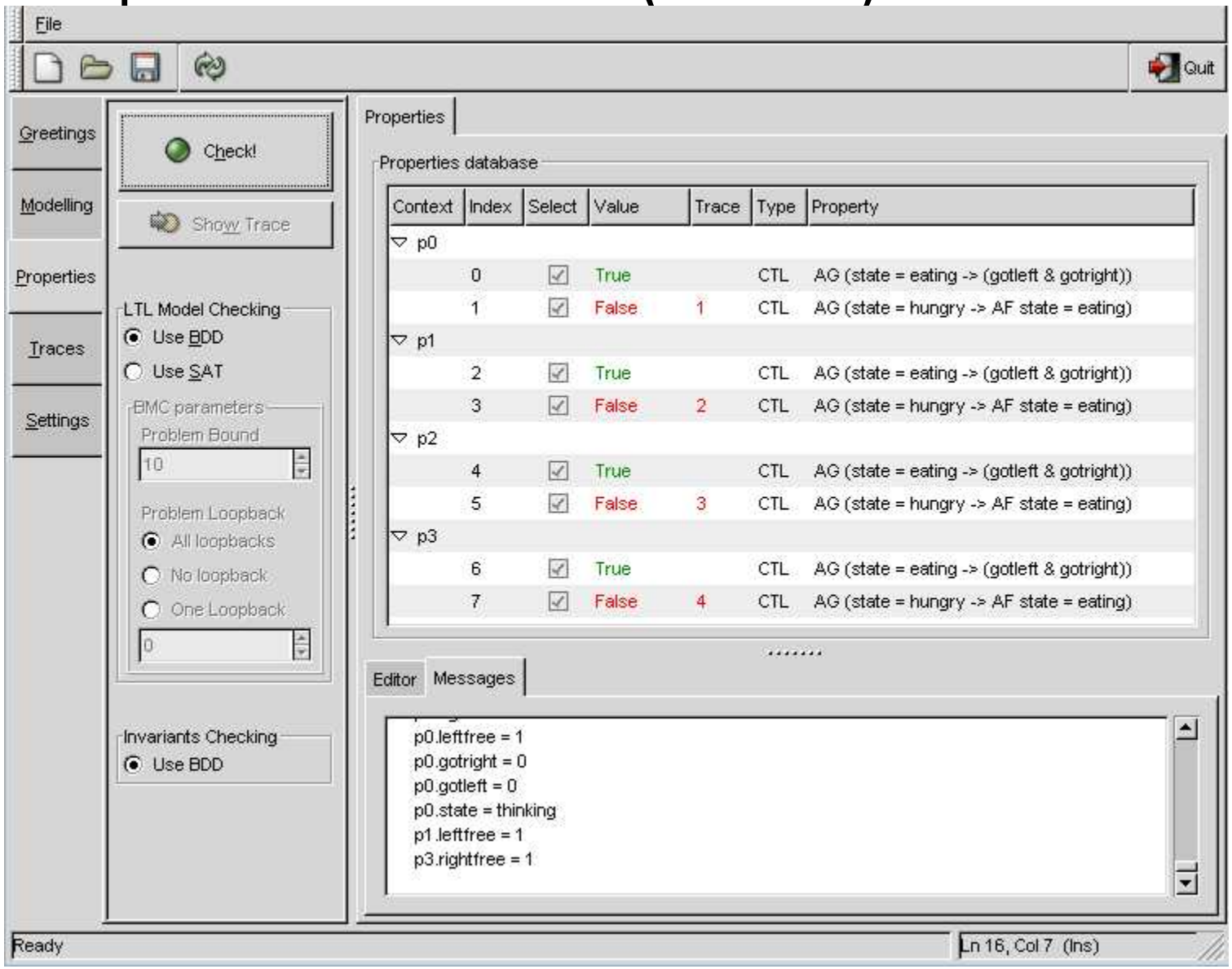
Compile


File
Quit
Greetings
Modelling
Properties
Traces
Settings
Compile
Editing
Goto Line
1
Find
Options
File
Ordering file:
phil
Messages
forks : array 0..3 of {nobody, 0, 1, 2, 3};
p0 : process philosopher(0, forks[0], forks[1]);
p1 : process philosopher(1, forks[1], forks[2]);
p2 : process philosopher(2, forks[2], forks[3]);
p3 : process philosopher(3, forks[3], forks[0]);
ASSIGN
init(forks[0]) := nobody;
init(forks[1]) := nobody;
init(forks[2]) := nobody;
init(forks[3]) := nobody;
MODULE philosopher(i, left, right)
VAR
state : { thinking, hungry, eating, done };
DEFINE
gotleft := (left = i);
gotright := (right = i);
leftfree := (left = nobody);
Resetting NuSMV... Ok
Resetting NuSMV... Ok
Parsing SMV source from 'C:\gnusmv\workspace\phil'... Ok
Flattening the model... Ok
Encoding variables... Ok
Building model for BDD-based Model Checking... Ok
Ready
Ln 29, Col 8 (Ins)

Property Check(Before)



Properties Check(After)



Properties Check(Property <1>)

```
Checking CTL property <0>...
-- specification AG (state = eating -> (gotleft & gotright)) (in module p0) is true

Checking CTL property <1>...
-- specification AG (state = hungry -> AF state = eating) (in module p0) is false
-- as demonstrated by the following execution sequence
-> State 1.1 <-
   [executing process p1]
   forks[0] = nobody
   forks[1] = nobody
   forks[2] = nobody
   forks[3] = nobody
   p0.rightfree = 1
   p0.leftfree = 1
   p0.gotright = 0
   p0.gotleft = 0
   p0.state = thinking
   p1.rightfree = 1
   p1.leftfree = 1
   p1.gotright = 0
   p1.gotleft = 0
   p1.state = thinking
   p2.rightfree = 1
   p2.leftfree = 1
   p2.gotright = 0
   p2.gotleft = 0
   p2.state = thinking
   p3.rightfree = 1
   p3.leftfree = 1
   p3.gotright = 0
   p3.gotleft = 0
   p3.state = thinking
-> State 1.2 <-
   [executing process p0]
-> State 1.3 <-
   [executing process p1]
   p0.state = hungry
-> State 1.4 <-
   [executing process p0]
```

```
-- loop starts here --
-> State 1.5 <-
   [executing process p1]
   forks[0] = 0
   p0.leftfree = 0
   p0.gotleft = 1
   p3.rightfree = 0
-> State 1.6 <-
   [executing process p2]
-> State 1.7 <-
   [executing process p3]
-- loop starts here --
-> State 1.8 <-
   [executing process p1]
-> State 1.9 <-
   [executing process p1]
   p1.state = hungry
-> State 1.10 <-
   [executing process p0]
   forks[1] = 1
   p0.rightfree = 0
   p1.leftfree = 0
   p1.gotleft = 1
-> State 1.11 <-
   [executing process p1]
-> State 1.12 <-
   [executing process p2]
   forks[2] = 1
   p1.rightfree = 0
   p1.gotright = 1
   p2.leftfree = 0
-> State 1.13 <-
   [executing process p3]
-> State 1.14 <-
   [executing process p1]
-> State 1.15 <-
   [executing process p1]
   p1.state = eating
-> State 1.16 <-
   [executing process p1]
   p1.state = done
```

```
-> State 1.17 <-
   [executing process p1]
   forks[1] = nobody
   forks[2] = nobody
   p0.rightfree = 1
   p1.rightfree = 1
   p1.leftfree = 1
   p1.gotright = 0
   p1.gotleft = 0
   p1.state = thinking
   p2.leftfree = 1
```

Properties Check(Property <3>)

```
Checking CTL property <3>...
-- specification AG (state = hungry -> AF state = eating) (in module p1) is false
-- as demonstrated by the following execution sequence
-> State 2.1 <-
   [executing process p1]
   forks[0] = nobody
   forks[1] = nobody
   forks[2] = nobody
   forks[3] = nobody
   p0.rightfree = 1
   p0.leftfree = 1
   p0.gotright = 0
   p0.gotleft = 0
   p0.state = thinking
   p1.rightfree = 1
   p1.leftfree = 1
   p1.gotright = 0
   p1.gotleft = 0
   p1.state = thinking
   p2.rightfree = 1
   p2.leftfree = 1
   p2.gotright = 0
   p2.gotleft = 0
   p2.state = thinking
   p3.rightfree = 1
   p3.leftfree = 1
   p3.gotright = 0
   p3.gotleft = 0
   p3.state = thinking
-> State 2.2 <-
   [executing process p1]
   p1.state = hungry
-> State 2.3 <-
   [executing process p0]
   forks[1] = 1
   p0.rightfree = 0
   p1.leftfree = 0
   p1.gotleft = 1
-> State 2.4 <-
   [executing process p2]
```

```
-> State 2.5 <-
   [executing process p2]
   p2.state = hungry
-> State 2.6 <-
   [executing process p1]
   forks[2] = 2
   p1.rightfree = 0
   p2.leftfree = 0
   p2.gotleft = 1
-> State 2.7 <-
   [executing process p2]
-> State 2.8 <-
   [executing process p3]
   forks[3] = 2
   p2.rightfree = 0
   p2.gotright = 1
   p3.leftfree = 0
-- loop starts here --
-> State 2.9 <-
   [executing process p1]
-> State 2.10 <-
   [executing process p0]
-- loop starts here --
-> State 2.11 <-
   [executing process p1]
-> State 2.12 <-
   [executing process p2]
-> State 2.13 <-
   [executing process p3]
   p2.state = eating
-> State 2.14 <-
   [executing process p2]
-> State 2.15 <-
   [executing process p2]
   p2.state = done
```

```
-> State 2.16 <-
   [executing process p2]
   forks[2] = nobody
   forks[3] = nobody
   p1.rightfree = 1
   p2.rightfree = 1
   p2.leftfree = 1
   p2.gotright = 0
   p2.gotleft = 0
   p2.state = thinking
   p3.leftfree = 1
-> State 2.17 <-
   [executing process p2]
   p2.state = hungry
-> State 2.18 <-
   [executing process p2]
   forks[2] = 2
   p1.rightfree = 0
   p2.leftfree = 0
   p2.gotleft = 1
-> State 2.19 <-
   [executing process p1]
   forks[3] = 2
   p2.rightfree = 0
   p2.gotright = 1
   p3.leftfree = 0
```

Properties Check(Property <5>)

```
Checking CTL property <5>...
-- specification AG (state = hungry -> AF state = eating) (in module p2) is false
-- as demonstrated by the following execution sequence
-> State 3.1 <-
   [executing process p1]
   forks[0] = nobody
   forks[1] = nobody
   forks[2] = nobody
   forks[3] = nobody
   p0.rightfree = 1
   p0.leftfree = 1
   p0.gotright = 0
   p0.gotleft = 0
   p0.state = thinking
   p1.rightfree = 1
   p1.leftfree = 1
   p1.gotright = 0
   p1.gotleft = 0
   p1.state = thinking
   p2.rightfree = 1
   p2.leftfree = 1
   p2.gotright = 0
   p2.gotleft = 0
   p2.state = thinking
   p3.rightfree = 1
   p3.leftfree = 1
   p3.gotright = 0
   p3.gotleft = 0
   p3.state = thinking
-> State 3.2 <-
   [executing process p2]
-> State 3.3 <-
   [executing process p1]
   p2.state = hungry
-> State 3.4 <-
   [executing process p0]
-> State 3.5 <-
   [executing process p1]
-> State 3.6 <-
   [executing process p2]
-> State 3.7 <-
   [executing process p3]
   forks[2] = 2
   p1.rightfree = 0
   p2.leftfree = 0
   p2.gotleft = 1
-- loop starts here --
-> State 3.8 <-
   [executing process p1]
-> State 3.9 <-
   [executing process p0]
-- loop starts here --
-> State 3.10 <-
   [executing process p1]
-> State 3.11 <-
   [executing process p3]
-> State 3.12 <-
   [executing process p3]
   p3.state = hungry
-> State 3.13 <-
   [executing process p2]
   forks[3] = 3
   p2.rightfree = 0
   p3.leftfree = 0
   p3.gotleft = 1
-> State 3.14 <-
   [executing process p3]
-> State 3.15 <-
   [executing process p3]
   forks[0] = 3
   p0.leftfree = 0
   p3.rightfree = 0
   p3.gotright = 1
-> State 3.16 <-
   [executing process p3]
   p3.state = eating
-> State 3.17 <-
   [executing process p3]
   p3.state = done
-> State 3.18 <-
   [executing process p1]
   forks[0] = nobody
   forks[3] = nobody
   p0.leftfree = 1
   p2.rightfree = 1
   p3.rightfree = 1
   p3.leftfree = 1
   p3.gotright = 0
   p3.gotleft = 0
   p3.state = thinking
```

Properties Check(Property <7>)

```
Checking CTL property <7>...
-- specification AG (state = hungry -> AF state = eating) (in module p3) is false
-- as demonstrated by the following execution sequence
-> State 4.1 <-
   [executing process p1]
   forks[0] = nobody
   forks[1] = nobody
   forks[2] = nobody
   forks[3] = nobody
   p0.rightfree = 1
   p0.leftfree = 1
   p0.gotright = 0
   p0.gotleft = 0
   p0.state = thinking
   p1.rightfree = 1
   p1.leftfree = 1
   p1.gotright = 0
   p1.gotleft = 0
   p1.state = thinking
   p2.rightfree = 1
   p2.leftfree = 1
   p2.gotright = 0
   p2.gotleft = 0
   p2.state = thinking
   p3.rightfree = 1
   p3.leftfree = 1
   p3.gotright = 0
   p3.gotleft = 0
   p3.state = thinking
-> State 4.2 <-
   [executing process p3]
-> State 4.3 <-
   [executing process p1]
   p3.state = hungry
-> State 4.4 <-
   [executing process p0]
-> State 4.5 <-
   [executing process p1]
```

```
-> State 4.6 <-
   [executing process p2]
-> State 4.7 <-
   [executing process p3]
-- loop starts here --
-> State 4.8 <-
   [executing process p1]
   forks[3] = 3
   p2.rightfree = 0
   p3.leftfree = 0
   p3.gotleft = 1
-> State 4.9 <-
   [executing process p0]
-- loop starts here --
-> State 4.10 <-
   [executing process p1]
-> State 4.11 <-
   [executing process p2]
-> State 4.12 <-
   [executing process p0]
-> State 4.13 <-
   [executing process p0]
   p0.state = hungry
-> State 4.14 <-
   [executing process p3]
   forks[0] = 0
   p0.leftfree = 0
   p0.gotleft = 1
   p3.rightfree = 0
-> State 4.15 <-
   [executing process p0]
-> State 4.16 <-
   [executing process p0]
   forks[1] = 0
   p0.rightfree = 0
   p0.gotright = 1
   p1.leftfree = 0
```



```
-> State 4.17 <-
   [executing process p0]
   p0.state = eating
-> State 4.18 <-
   [executing process p0]
   p0.state = done
-> State 4.19 <-
   [executing process p1]
   forks[0] = nobody
   forks[1] = nobody
   p0.rightfree = 1
   p0.leftfree = 1
   p0.gotright = 0
   p0.gotleft = 0
   p0.state = thinking
   p1.leftfree = 1
   p3.rightfree = 1
```

Trace View(Trace 1, 2)

Trace 1 | Trace 2 | Trace 3 | Trace 4

CTL property 1: AG (state = hungry -> AF state = eating)

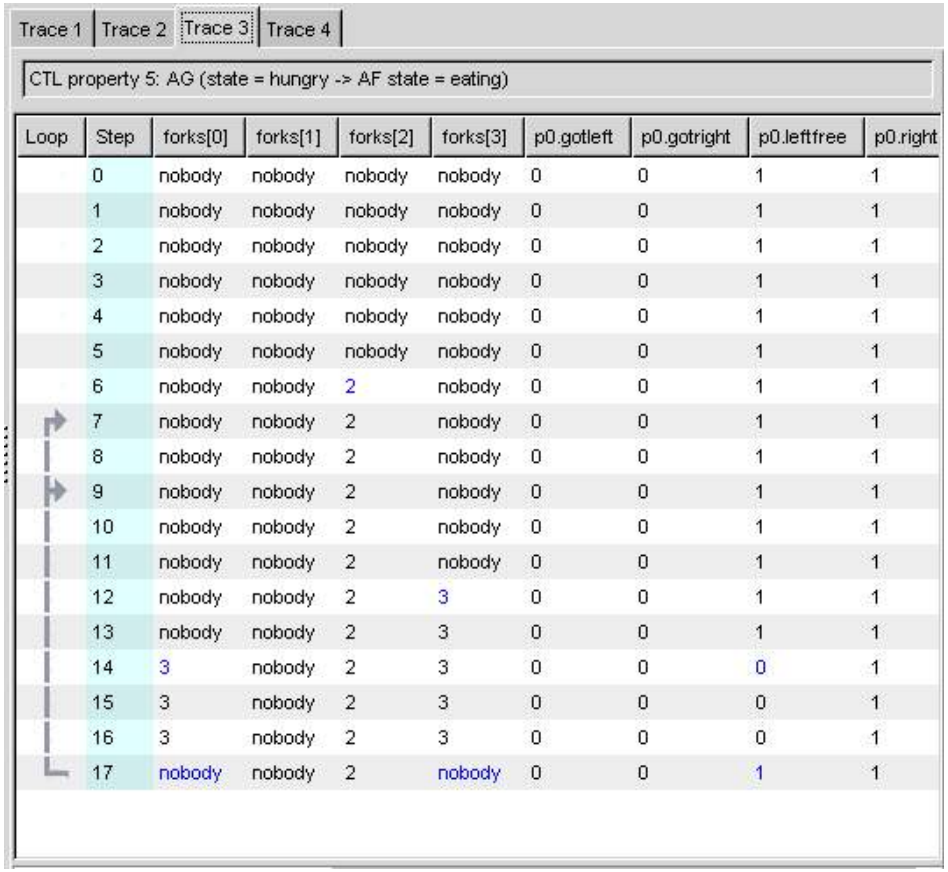
Loop	Step	forks[0]	forks[1]	forks[2]	forks[3]	p0.gotleft	p0.gotright	p0.leftfree	p0.right
	0	nobody	nobody	nobody	nobody	0	0	1	1
	1	nobody	nobody	nobody	nobody	0	0	1	1
	2	nobody	nobody	nobody	nobody	0	0	1	1
	3	nobody	nobody	nobody	nobody	0	0	1	1
	4	0	nobody	nobody	nobody	1	0	0	1
	5	0	nobody	nobody	nobody	1	0	0	1
	6	0	nobody	nobody	nobody	1	0	0	1
	7	0	nobody	nobody	nobody	1	0	0	1
	8	0	nobody	nobody	nobody	1	0	0	1
	9	0	1	nobody	nobody	1	0	0	0
	10	0	1	nobody	nobody	1	0	0	0
	11	0	1	1	nobody	1	0	0	0
	12	0	1	1	nobody	1	0	0	0
	13	0	1	1	nobody	1	0	0	0
	14	0	1	1	nobody	1	0	0	0
	15	0	1	1	nobody	1	0	0	0
	16	0	nobody	nobody	nobody	1	0	0	1

Trace 1 | Trace 2 | Trace 3 | Trace 4

CTL property 3: AG (state = hungry -> AF state = eating)

Loop	Step	forks[0]	forks[1]	forks[2]	forks[3]	p0.gotleft	p0.gotright	p0.leftfree	p0.right
	0	nobody	nobody	nobody	nobody	0	0	1	1
	1	nobody	nobody	nobody	nobody	0	0	1	1
	2	nobody	1	nobody	nobody	0	0	1	0
	3	nobody	1	nobody	nobody	0	0	1	0
	4	nobody	1	nobody	nobody	0	0	1	0
	5	nobody	1	2	nobody	0	0	1	0
	6	nobody	1	2	nobody	0	0	1	0
	7	nobody	1	2	2	0	0	1	0
	8	nobody	1	2	2	0	0	1	0
	9	nobody	1	2	2	0	0	1	0
	10	nobody	1	2	2	0	0	1	0
	11	nobody	1	2	2	0	0	1	0
	12	nobody	1	2	2	0	0	1	0
	13	nobody	1	2	2	0	0	1	0
	14	nobody	1	2	2	0	0	1	0
	15	nobody	1	nobody	nobody	0	0	1	0
	16	nobody	1	nobody	nobody	0	0	1	0
	17	nobody	1	2	nobody	0	0	1	0
	18	nobody	1	2	2	0	0	1	0

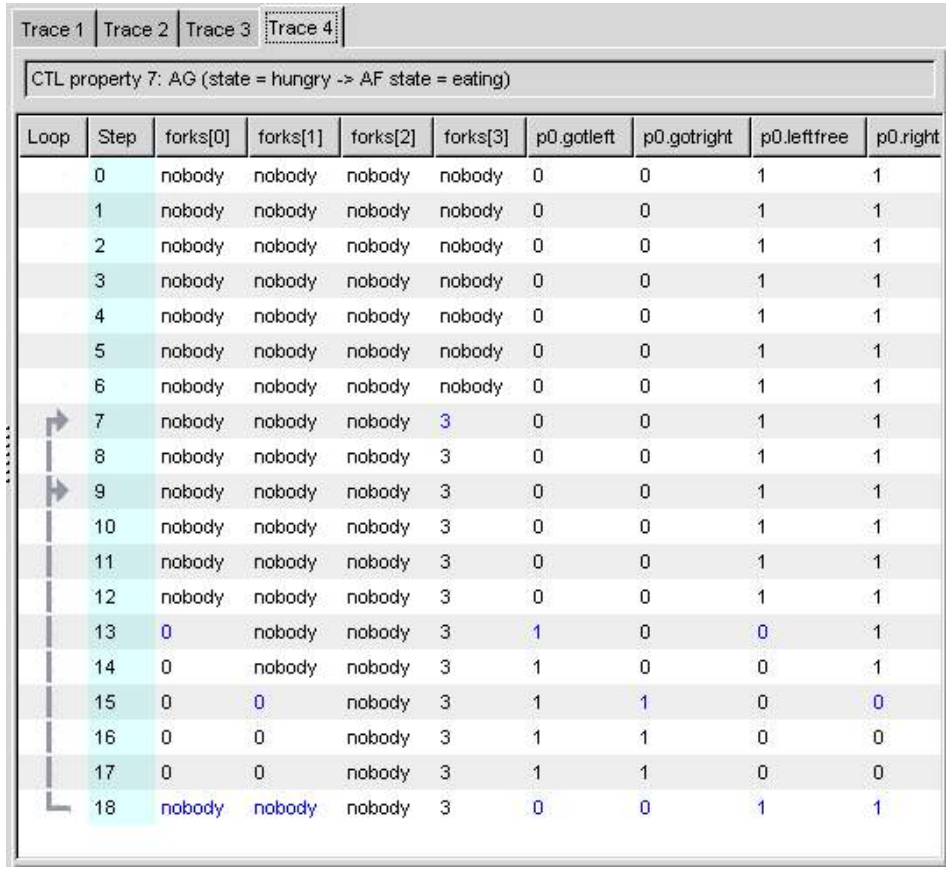
Trace View(Trace 3, 4)



Trace 1 | Trace 2 | Trace 3 | Trace 4

CTL property 5: AG (state = hungry -> AF state = eating)

Loop	Step	forks[0]	forks[1]	forks[2]	forks[3]	p0.gotleft	p0.gotright	p0.leftfree	p0.right
	0	nobody	nobody	nobody	nobody	0	0	1	1
	1	nobody	nobody	nobody	nobody	0	0	1	1
	2	nobody	nobody	nobody	nobody	0	0	1	1
	3	nobody	nobody	nobody	nobody	0	0	1	1
	4	nobody	nobody	nobody	nobody	0	0	1	1
	5	nobody	nobody	nobody	nobody	0	0	1	1
	6	nobody	nobody	2	nobody	0	0	1	1
↱	7	nobody	nobody	2	nobody	0	0	1	1
|	8	nobody	nobody	2	nobody	0	0	1	1
↱	9	nobody	nobody	2	nobody	0	0	1	1
|	10	nobody	nobody	2	nobody	0	0	1	1
|	11	nobody	nobody	2	nobody	0	0	1	1
|	12	nobody	nobody	2	3	0	0	1	1
|	13	nobody	nobody	2	3	0	0	1	1
|	14	3	nobody	2	3	0	0	0	1
|	15	3	nobody	2	3	0	0	0	1
|	16	3	nobody	2	3	0	0	0	1
└	17	nobody	nobody	2	nobody	0	0	1	1



Trace 1 | Trace 2 | Trace 3 | Trace 4

CTL property 7: AG (state = hungry -> AF state = eating)

Loop	Step	forks[0]	forks[1]	forks[2]	forks[3]	p0.gotleft	p0.gotright	p0.leftfree	p0.right
	0	nobody	nobody	nobody	nobody	0	0	1	1
	1	nobody	nobody	nobody	nobody	0	0	1	1
	2	nobody	nobody	nobody	nobody	0	0	1	1
	3	nobody	nobody	nobody	nobody	0	0	1	1
	4	nobody	nobody	nobody	nobody	0	0	1	1
	5	nobody	nobody	nobody	nobody	0	0	1	1
	6	nobody	nobody	nobody	nobody	0	0	1	1
↱	7	nobody	nobody	nobody	3	0	0	1	1
|	8	nobody	nobody	nobody	3	0	0	1	1
↱	9	nobody	nobody	nobody	3	0	0	1	1
|	10	nobody	nobody	nobody	3	0	0	1	1
|	11	nobody	nobody	nobody	3	0	0	1	1
|	12	nobody	nobody	nobody	3	0	0	1	1
|	13	0	nobody	nobody	3	1	0	0	1
|	14	0	nobody	nobody	3	1	0	0	1
|	15	0	0	nobody	3	1	1	0	0
|	16	0	0	nobody	3	1	1	0	0
|	17	0	0	nobody	3	1	1	0	0
└	18	nobody	nobody	nobody	3	0	0	1	1

Thank You