Formal Verification of Observational Transition Systems with Proof Scores

Kazuhiro Ogata (JAIST)
Nov 15, 2016
Tokyo, Japan
CafeOBJ Tutorial at ICFEM 2016

Outline of Lecture

Two mutual exclusion protocols as examples

- *2P-Mutex* – A simple two-process mutual exclusion protocol
- *Qlock* – Dijkstra's binary semaphore

are used to describe how to specify

Observational transition systems (*OTSs*)

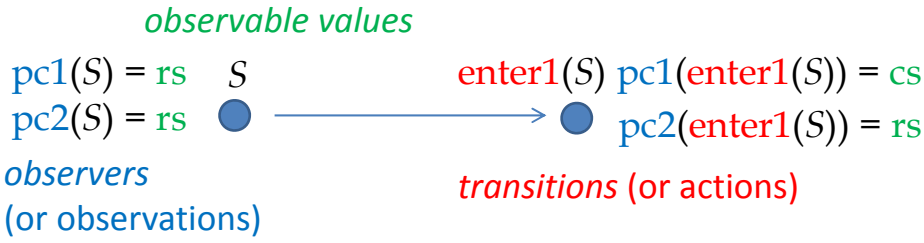
- state machines

in CafeOBJ and how to verify that OTSs enjoy invariant properties by writing

Proof scores

in CafeOBJ and executing them with the CafeOBJ system.

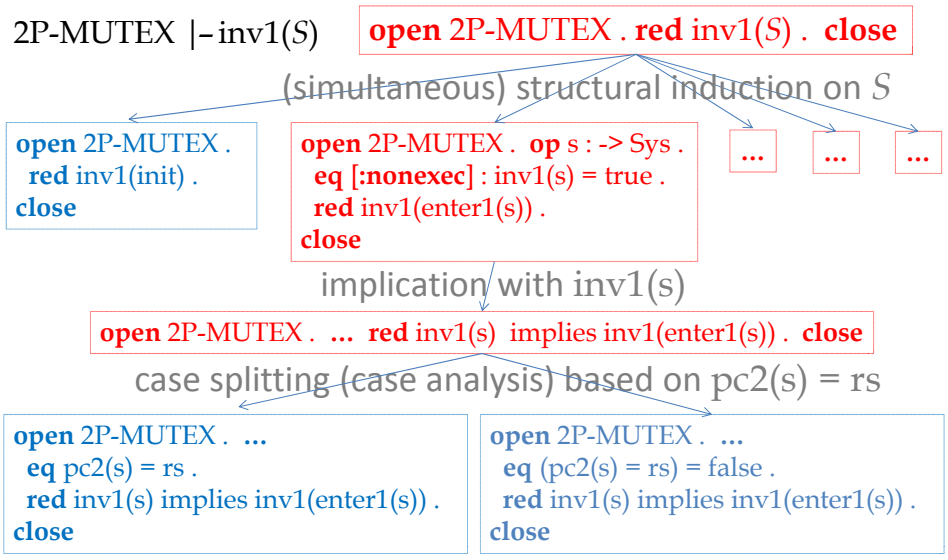
Outline of how to specify OTSs in CafeOBJ




```
ceq pc1(enter1(S)) = cs       if c-enter1(S) .
ceq pc2(enter1(S)) = pc2(S)   if c-enter1(S) .
ceq enter1(S)      = S        if not c-enter1(S) .
```

where c-enter1(S) is pc2(S) = rs

Outline of proof scores



Outline of remaining talk

Specify two mutual exclusion protocols as OTSs in CafeOBJ

Verify that the two mutual exclusion protocols enjoy the mutual exclusion properties by writing proof scores in CafeOBJ and executing them with the CafeOBJ system

2P-MUTEX

Pseudo-code for p_1:

Loop: "Remainder Section"
rs: **repeat while** p_2 is at cs;
"Critical Section"
cs: do nothing

Pseudo-code for p_2:

Loop: "Remainder Section"
rs: **repeat while** p_1 is at cs;
"Critical Section"
cs: do nothing

Initially, both p_1 and p_2 are in Remainder Section (or at label rs).

"do nothing" at cs means that each process does nothing just before leaving the critical section, although there may be something they are supposed to do in the critical section.

Observers and transitions

Observers

op pc1 : Sys -> Label
op pc2 : Sys -> Label

Observable values

init

eq pc1(init) = rs .
eq pc2(init) = rs .

Transitions

Pseudo-code for p_1:

Loop: "Remainder Section"
rs: **repeat while** p_2 is at cs;
"Critical Section"
cs: do nothing

op enter1 : Sys -> Sys

op leave1 : Sys -> Sys

How to change observable values with transitions

pc1(S) = l pc2(S) = rs S → S' = enter1(S) pc1(S') = cs pc2(S') = rs

ceq pc1(enter1(S)) = cs **if** c-enter1(S) .
ceq pc2(enter1(S)) = pc2(S) **if** c-enter1(S) .
ceq enter1(S) = S **if not** c-enter1(S) .

where c-enter1(S) is pc2(S) = rs

pc1(S) = cs pc2(S) = l S → S' = leave1(S) pc1(S') = rs pc2(S') = l

ceq pc1(leave1(S)) = rs **if** c-leave1(S) .
ceq pc2(leave1(S)) = pc2(S) **if** c-leave1(S) .
ceq leave1(S) = S **if not** c-leave1(S) .

where c-leave1(S) is pc1(S) = cs

How to change observable values with transitions


```
ceq pc1(enter2(S)) = pc1(S) if c-enter2(S) .
ceq pc2(enter2(S)) = cs     if c-enter2(S) .
ceq enter2(S)      = S      if not c-enter2(S) .
```

where c-enter2(S) is pc1(S) = rs


```
ceq pc1(leave2(S)) = pc1(S) if c-leave2(S) .
ceq pc2(leave2(S)) = rs     if c-leave2(S) .
ceq leave2(S)      = S      if not c-leave2(S) .
```

where c-leave2(S) is pc2(S) = cs

Mutual exclusion property

One desired property 2P-MUTEX should enjoy

There exists at most one process in Critical Section (or at label cs) at any given moment.

Can be expressed as an invariant of 2P-MUTEX

To verify that 2P-MUTEX enjoys the property, it suffices to proves inv1(S) is an invariant of 2P-MUTEX, namely 2P-MUTEX |– (∀S:Sys) inv1(S).

```
eq inv1(S:Sys) = not (pc1(S) = cs and pc2(S) = cs) .
```

What to do for writing proof scores

SI - Use of (Simultaneous) structural induction

TC - Use of Theorem of constants (elimination of universal quantifiers)

CS - Case splitting based on constructors

IMP - Conjecturing lemmas + Use of lemmas and induction hypotheses

RD - Reduction

Simultaneous structural induction

$(\forall S{:}Sys)(\forall \ldots)p_1(S,\ldots) \wedge \ldots \wedge (\forall S{:}Sys)(\forall \ldots)p_n(S,\ldots)$

if and only if

$(\forall S{:}Sys)((\forall \ldots)p_1(S,\ldots) \wedge \ldots \wedge (\forall \ldots)p_n(S,\ldots))$

if and only if

$(\forall \ldots)p_1(\mathrm{init},\ldots) \wedge \ldots \wedge (\forall \ldots)p_n(\mathrm{init},\ldots)$

$(\forall \ldots)p_1(s',\ldots) \wedge \ldots \wedge (\forall \ldots)p_n(s',\ldots)$

assuming $(\forall \ldots)p_1(s,\ldots) \wedge \ldots \wedge (\forall \ldots)p_n(s,\ldots)$

if and only if

$(\forall \ldots)p_1(\mathrm{init},\ldots)$

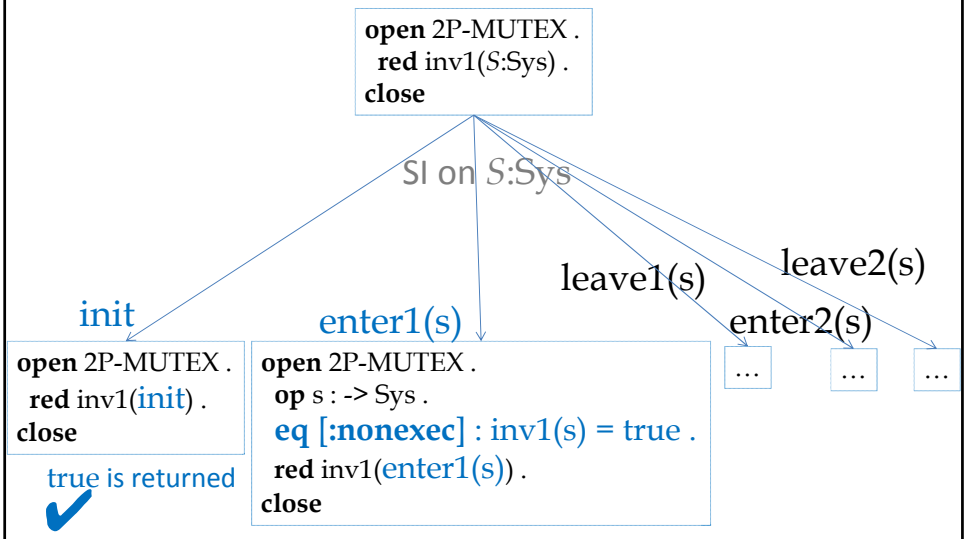
$(\forall \ldots)p_1(s',\ldots)$ assuming $(\forall \ldots)p_1(s,\ldots)$, … , $(\forall \ldots)p_n(s,\ldots)$

…

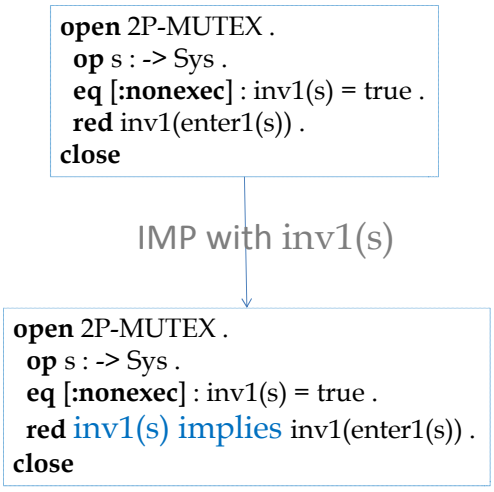
$(\forall \ldots)p_n(\mathrm{init},\ldots)$

$(\forall \ldots)p_n(s',\ldots)$ assuming $(\forall \ldots)p_1(s,\ldots)$, …, $(\forall \ldots)p_n(s,\ldots)$

Writing proof scores



Writing proof scores



Writing proof scores

```
open 2P-MUTEX .
  op s : -> Sys .
  eq [:nonexec] : inv1(s) = true .
  red inv1(s) implies inv1(enter1(s)) .
close
```

CS based on pc2(s) = rs

```
open 2P-MUTEX .
  op s : -> Sys .
  eq [:nonexec] : inv1(s) = true .
  eq pc2(s) = rs .
  red inv1(s) imlies inv1(enter1(s)) .
close
```

✔ true is returned

```
open 2P-MUTEX .
  op s : -> Sys .
  eq [:nonexec] : inv1(s) = true .
  eq (pc2(s) = rs) = false .
  red inv1(s) implies inv1(enter1(s)) .
close
```

✔ true is returned

Qlock

(An abstract version of) the Dijkstra's binary semaphore

Loop: "Remainder Section"
rs: enq(*queue*,*i*);
ws: **repeat until** top(*queue*) = *i*;
"Critical Section"
cs: deq(*queue*)

queue is atomic (or indivisible).

Initially, *queue* is empty and each process is at label rs.

queue is used in neither Remainder Section nor Critical Section.

Observable values and transitions

Observers

op queue : Sys -> Queue
op pc : Sys Pid -> Label

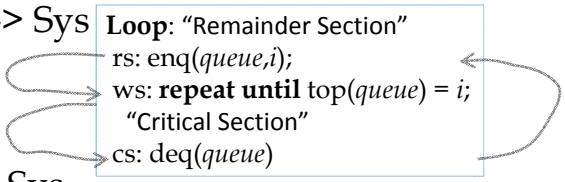
Observable values

init

eq queue(init) = empty .
eq pc(init,*I*) = rs .

Transitions

op want : Sys Pid -> Sys



op try : Sys Pid -> Sys

op exit : Sys Pid -> Sys

How to change observable values with transitions

ceq pc(want(*S*,*I*),*J*) = (if *I* = *J* then ws else pc(*S*,*J*) fi) **if** c-want(*S*,*I*) .
ceq queue(want(*S*,*I*)) = enq(queue(*S*),*I*) **if** c-want(*S*,*I*) .
ceq want(*S*,*I*) = *S* **if** not c-want(*S*,*I*) .

where c-want(*S*,*I*) is pc(*S*,*I*) = rs

ceq pc(try(*S*,*I*),*J*) = (if *I* = *J* then cs else pc(*S*,*J*) fi) **if** c-try(*S*,*I*) .
eq queue(try(*S*,*I*)) = queue(*S*) .
ceq try(*S*,*I*) = *S* **if** not c-try(*S*,*I*) .

where c-try(*S*,*I*) is pc(*S*,*I*) = ws and top(queue(*S*)) = *I*

ceq pc(exit(*S*,*I*),*J*) = (if *I* = *J* then rs else pc(*S*,*J*) fi) **if** c-exit(*S*,*I*) .
ceq queue(exit(*S*,*I*)) = deq(queue(*S*)) **if** c-exit(*S*,*I*) .
ceq exit(*S*,*I*) = *S* **if** not c-exit(*S*,*I*) .

where c-exit(*S*,*I*) is pc(*S*,*I*) = cs

Mutual exclusion property

One desired property Qlock should enjoy

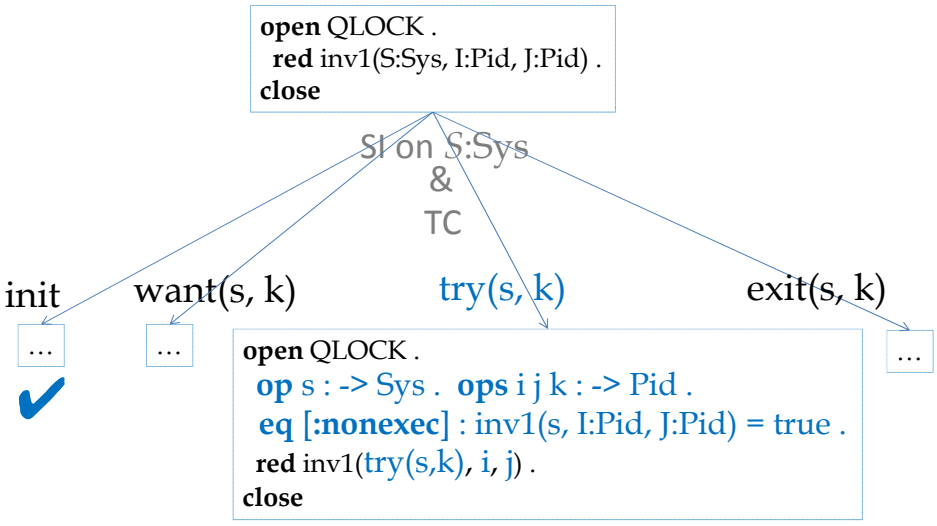
There exists at most one process in Critical Section (or at label cs) at any given moment.

Can be expressed as an invariant of Qlock

To verify that Qlock enjoys the property, it suffices to proves (∀*I*,*J*:Pid) inv1(*S*,*I*,*J*) is an invariant of Qlock, namely Qlock |– (∀*S*:Sys)(∀*I*,*J*:Pid) inv1(*S*,*I*,*J*).

eq inv1(*S*:Sys,*I*:Pid,*J*:Pid)
= ((pc(*S*,*I*) = cs and pc(*S*,*J*) = cs) implies *I* = *J*) .

Writing proof scores



Writing proof scores

```
open QLOCK .
  op s : -> Sys .  ops i j k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  red inv1(try(s,k), i, j) .
close
```

IMP with inv1(s, i, j)

```
open QLOCK .
  op s : -> Sys .  ops i j k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  red inv1(s, i, j) implies inv1(try(s,k), i, j) .
close
```


Writing proof scores

```
open QLOCK .
  op s : -> Sys .  ops i j k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  red inv1(s, i, j) implies inv1(try(s,k), i, j) .
close
```

CS based on pc(s,k) = ws & top(queue(s)) = k

pc(s,k) = ws
top(queue(s)) = k

```
open QLOCK .
  op s : -> Sys .  ops i j k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq pc(s, k) = ws .  eq top(queue(s)) = k .
  red inv1(s, i, j) implies inv1(try(s,k), i, j) .
close
```

pc(s,k) = ws
top(queue(s)) ≠ k

pc(s,k) ≠ ws
top(queue(s)) = k

pc(s,k) ≠ ws
top(queue(s)) ≠ k

… ✔ … ✔ … ✔

Writing proof scores

```
open QLOCK .
 op s : -> Sys .  ops i j k : -> Pid .
 eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
 eq pc(s, k) = ws .  eq top(queue(s)) = k .
 red inv1(s, i, j) implies inv1(try(s,k), i, j) .
close
```

CS based on i = k & j = k

i = k
j = k
…
✔

i = k
j ≠ k

```
open QLOCK .
 op s : -> Sys .  ops i j k : -> Pid .
 eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
 eq pc(s, k) = ws .  eq top(queue(s)) = k .
 eq i = k . eq (j = k) = false .
 red inv1(s, i, j) implies inv1(try(s,k), i, j) .
close
```

i ≠ k
j = k
…

i ≠ k
j ≠ k
…
✔

Writing proof scores

```
open QLOCK .
 op s : -> Sys . op i j k : -> Pid .
 eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
 eq pc(s, k) = ws .  eq top(queue(s)) = k .
 eq i = k .  eq (j = k) = false .
 red inv1(s,i,j) implies inv1(try(s,k), i, j) .
close
```

CS based on pc(s,j) = cs

pc(s,j) = cs

```
open QLOCK .
 op s : -> Sys . op i j k : -> Pid .
 eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
 eq pc(s, k) = ws .  eq top(queue(s)) = k .
 eq i = k .  eq (j = k) = false .
 eq pc(s,j) = cs .
 red inv1(s,i,j) implies inv1(try(s,k), i, j) .
close
```

false is returned

pc(s,j) ≠ cs

```
open QLOCK .
 op s : -> Sys . op i j k : -> Pid .
 eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
 eq pc(s, k) = ws .  eq top(queue(s)) = k .
 eq i = k .  eq (j = k) = false .
 eq (pc(s,j) = cs) = false .
 red inv1(s,i,j) implies inv1(try(s,k), i, j) .
close
```

✔ true is returned

A systematic way to conjecture lemmas

Each goal (case) is characterized by equations

```
open QLOCK .
 op s : -> Sys . op i j k : -> Pid .
 eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
 eq pc(s, k) = ws .  eq top(queue(s)) = k .
 eq i = k .  eq (j = k) = false .
 eq pc(s,j) = cs .
 red inv1(s,i,j) implies inv1(try(s,k), i, j) .
close
```

Combining them with conjunctions to be a Boolean term, negating it, and replacing fresh constants with variables in it

$$\text{not}(\text{pc}(S,K) = \text{ws and top}(\text{queue}(S)) = K \text{ and } I = K \text{ and (not } J = K) \text{ and pc}(S,J) = \text{cs})$$

One possible lemma candidate

Let $\text{inv2-0}(S,I,J,K)$ be this Boolean term

A systematic way to conjecture lemmas

The shorter, the better.

This is because lemmas should be proved.

Any formula that implies $\text{inv2-0}(S,I,J,K)$ is a lemma candidate.

The following two valid formulas are used to make the lemma candidate shorter:

(1) $(\forall x,x',z)\ p(x,q(x,x'),z) \Rightarrow (\forall x,x',y,z)\ (y = q(x,x') \Rightarrow \text{p}(x,y,z))$

(2) $(\forall x,y)\ p(x,y) \Rightarrow (\forall x,y,z)\ (\text{p}(x,y) \vee \text{q}(y,z))$

A systematic way to conjecture lemmas

not(pc(*S*,*K*) = ws and top(queue(*S*)) = *K* and
I = *K* and (not *J* = *K*) and pc(*S*,*J*) = cs)

⇐

not(pc(*S*,*I*) = ws and top(queue(*S*)) = *I* and
(not *J* = *I*) and pc(*S*,*J*) = cs) By (1)

⇐

not(pc(*S*, top(queue(*S*))) = ws and
(not *J* = top(queue(*S*))) and pc(*S*,*J*) = cs) By (1)

⇐

not((not *J* = top(queue(*S*))) and pc(*S*,*J*) = cs) By (2)

⇔

pc(*S*,*J*) = cs implies top(queue(*S*)) = *J*

Let inv2(*S*,*J*) be this Boolean term

Writing proof scores (cont.)

QLOCK |– inv2(*S*,*J*) is tackled simultaneously.

SI on *S* is applied.

Then,

eq [:nonexec] : inv2(s, *J*:Pid) = true .

can be used as an induction hypothesis, in addition to

eq [:nonexec] : inv1(s, *I*:Pid , *J*:Pid) = true .

Writing proof scores (cont.)

```
open QLOCK .
  op s : -> Sys . op i j k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, J:Pid) = true .
  eq pc(s, k) = ws .  eq top(queue(s)) = k .
  eq i = k .  eq (j = k) = false .
  eq pc(s,j) = cs .
  red inv1(s,i,j) implies inv1(try(s,k), i, j) .
close
```

IMP with inv2(s,j)

```
open QLOCK .
  op s : -> Sys . op i j k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, J:Pid) = true .
  eq pc(s, k) = ws .  eq top(queue(s)) = k .
  eq i = k .  eq (j = k) = false .
  eq pc(s,j) = cs .
  red inv2(s, j) implies inv1(s,i,j) implies inv1(try(s,k), i, j) .
close
```

true is returned ✔

Writing proof scores (cont.)

```
open QLOCK .
  op s : -> Sys .   ops i k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, J:Pid) = true .
  eq pc(s,k) = cs .   eq (i = k) = false .   eq pc(s,i) = cs .
  red inv2(s,i) implies inv2(exit(s,k),i) .
close
```

IMP with inv1(s,i,k)

```
open QLOCK .
  op s : -> Sys .   ops i k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, J:Pid) = true .
  eq pc(s,k) = cs .   eq (i = k) = false .   eq pc(s,i) = cs .
  red inv1(s,i,k) implies inv2(s,i) implies inv2(exit(s,k),i) .
close
```

true is returned ✔

Summary

Two mutual exclusion protocols as examples

- *2P-Mutex* – A simple two-process mutual exclusion protocol
- *Qlock* – Dijkstra's binary semaphore

have been used to describe how to specify

Observational transition systems (*OTSs*)

- state machines

in CafeOBJ and how to verify that OTSs enjoy invariant properties by writing

Proof scores

in CafeOBJ and executing them with the CafeOBJ system.