# Formal Verification of Observational Transition Systems with CafeOBJ CITP

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

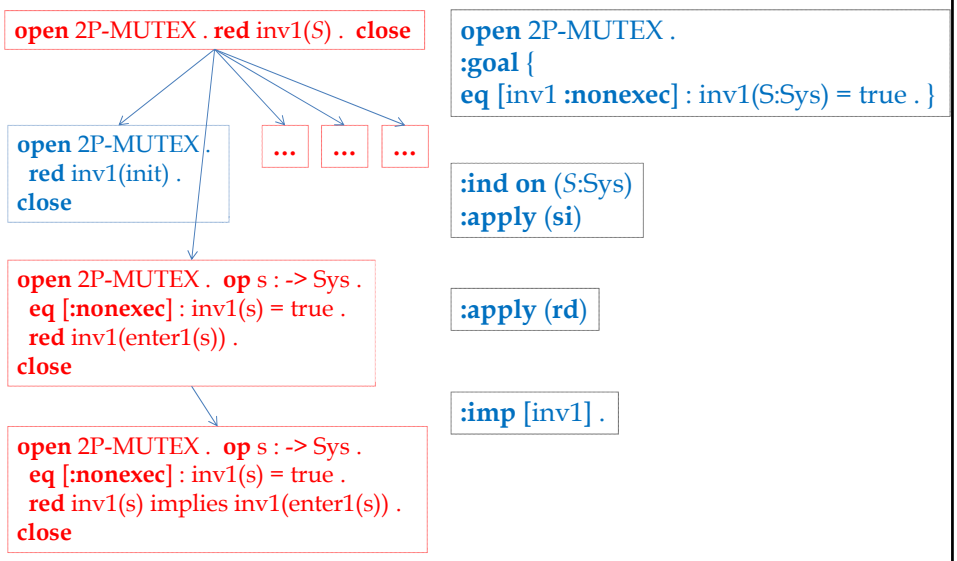
are used to describe how to verify that observational transition systems (OTSs) enjoy invariant properties with

*CafeOBJ Constructor-based Inductive Theorem Prover (CITP)*

a proof assistant for CafeOBJ

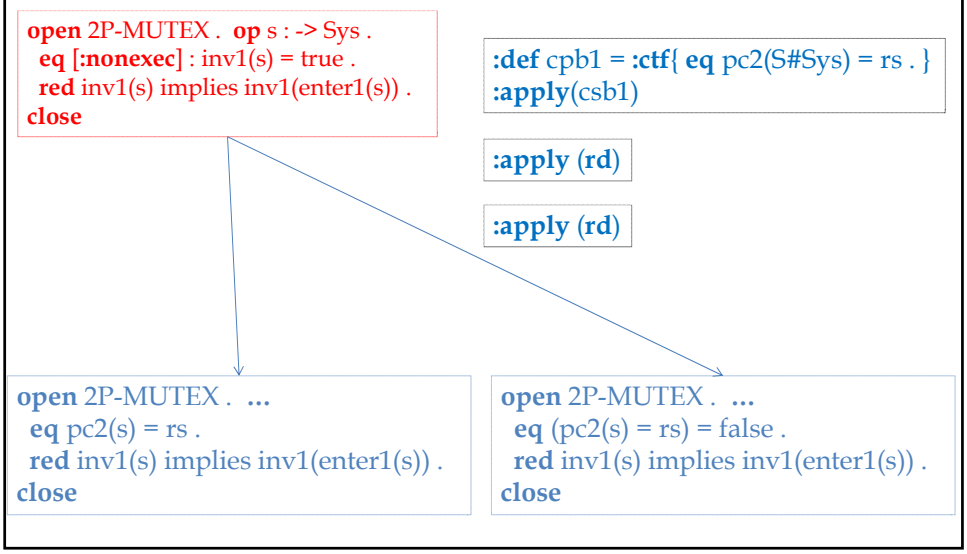
based on how proof scores have been written.

# Outline of CafeOBJ CITP






# Outline of CafeOBJ CITP



# Outline of remaining talk

Review how to specify the two mutual exclusion protocols as OTSs in CafeOB

Verify that the two mutual exclusion protocols enjoy the mutual exclusion properties with CafeOBJ CITP based on how the proof scores have been written to verify that the protocols enjoy the property



# How to specify 2P-MUTEX

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 specify 2P-MUTEX


```
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


```
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 specify 2P-MUTEX


```
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

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

We tackle the goal:

2P-MUTEX |– (∀*S*:Sys) inv1(*S*) = true .



# Some commands of CafeOBJ CITP

**:goal** { *eqs* } where *eqs* is a set of conditional equations (or sentences) to prove

E.g. *eqs* is **eq** [inv1 **:nonexec**] : inv1(*S*:Sys) = true .

or **eq** [inv1 **:nonexec**] : inv1(*S*:Sys,*I*:Pid,*J*:Pid) = true .
**eq** [inv2 **:nonexec**] : inv2(*S*:Sys,*I*:Pid) = true .

A module *Spec* is supposed to be opened or selected.

An initial current goal *Spec* |– *eqs* is defined.

**:ind on** *X*:*S* where *X* is a variable of a constrained sort *S*

A variable *X* is specified to which simultaneous structural induction is applied.

# Some commands of CafeOBJ CITP

**:apply (si)**

Simultaneous structural induction is applied to the current goal on the variable $X$ specified with **:ind on**, replacing the current goal with $n$ sub-goals, where $n$ is the number of the constructors of the sort $S$ of $X$, and introducing fresh constants for the non-constant constructors.

**:apply (tc)**

Each variable in the sentences of the current goal is replaced with a fresh constant. If the current goal has two or more sentences, say $n$, then the goal is replaced with $n$ sub-goals.



# Some commands of CafeOBJ CITP

**#def** *base-eq*$_i$ = **:ctf** { **eq** $l_i = r_i$ . }

A base for case splitting is defined. The current goal is split into two sub-goals for $l_i = r_i$ and $(l_i = r_i) = \mathrm{false}$, respectively, when it is used by **:apply**.

**#def** *base-ctr*$_j$ = **:ctf** [ $t_j$ . ] where the sort of $t_j$ is constrained

A base for case splitting is defined. The current goal is split into $n$ sub-goals, where $n$ is the number of the constructors of the sort of $t_j$ when *base-ctr*$_j$ is used by **:apply**.

**:apply** ($b_1 \ldots b_m$) where $b_k$ is *base-eq*$_k$ or *base-ctr*$_k$ and splits the goal into $n_k$ sub-goals

The current goal is split into $n_1 \times \ldots \times n_m$ sub-goals.

# Some commands of CafeOBJ CITP

**:imp** [*label*] by {$x_1$ <- $t_1$ ; … $x_n$ <- $t_n$ ; }

where *label* is the label of an equation that is in the form **eq** $l$ = true ., $x_k$ is a variable in $l$, and $t_k$ is a ground term

Let $gl$ be the ground term obtained by replacing each $x_k$ wih $t_k$ in $l$, and $gs$ be the sentence to prove in the current goal.

The current goal is replaced with the goal in which $gl$ implies $gs$ is the sentence to prove.

**:imp** [*label*] .

In the case the equation referred by *label* has no variables.

**:apply (rd)**

The sentence to prove in the current goal is reduced. If it reduces to true, then the current goal is discharged.



# Some commands of CafeOBJ CITP

**:show proof**

An outline of the proof conducted so far is shown. The current goal is marked with >.

**:desc proof**

The sub-goals generated so far are shown.

**:desc .**

The current sub-goal is shown.

# Proving with CafeOBJ CITP

The specification under verification is 2P-MUTEX in which the sentence to prove is described.

```
open 2P-MUTEX .
:goal {
eq [inv1 :nonexec] : inv1(S:Sys) = true .
}
```

SI is applied to the current goal on *S*:Sys, replacing the current goal with five sub-goals because of the five constructors.

```
:ind on (S:Sys)
:apply (si)
```

```
open 2P-MUTEX .
 red inv1(init) .
close
```

```
-- init
:apply (rd)
```

The goal is discharged with **:apply** (**rd**).

Note that a comment starts with **--** until the end of the line.



# Proving with CafeOBJ CITP

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

```
-- enter1(S#Sys)
:imp [inv1] .
:def ctf-0 = :ctf{ eq pc2(S#Sys) = rs . }
:apply(ctf-0)
:apply (rd)
:apply (rd)
```

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

Each goal is discharged with **:apply** (**rd**).

# Proving with CafeOBJ CITP

```
open 2P-MUTEX .
 op s : -> Sys .
 eq [:nonexec] : inv1(s) = true .
 eq pc1(s) = cs .
 red inv1(s) implies inv1(leave1(s)) .
close
```

✔

```
open 2P-MUTEX .
 op s : -> Sys .
 eq [:nonexec] : inv1(s) = true .
 eq (pc1(s) = cs) = false .
 red inv1(s) implies inv1(leave1(s)) .
close
```

✔

Each goal is discharged with **:apply** (**rd**).

```
-- leave1(S#Sys)
:imp [inv1] .
:def ctf-1 = :ctf{ eq pc1(S#Sys) = cs . }
:apply(ctf-1)
:apply (rd)
:apply (rd)
```



# Proving with CafeOBJ CITP

```
open 2P-MUTEX .
 op s : -> Sys .
 eq [:nonexec] : inv1(s) = true .
 eq pc1(s) = rs .
 red inv1(s) implies inv1(enter2(s)) .
close
```

✔

```
open 2P-MUTEX .
 op s : -> Sys .
 eq [:nonexec] : inv1(s) = true .
 eq (pc1(s) = rs) = false .
 red inv1(s) implies inv1(enter2(s)) .
close
```

✔

Each goal is discharged with **:apply** (**rd**).

```
-- enter2(S#Sys)
:imp [inv1] .
:def ctf-2 = :ctf{ eq pc1(S#Sys) = rs . }
:apply(ctf-2)
:apply (rd)
:apply (rd)
```

# Proving with CafeOBJ CITP

```
open 2P-MUTEX .
  op s : -> Sys .
  eq [:nonexec] : inv1(s) = true .
  eq pc2(s) = cs .
  red inv1(s) implies inv1(leave2(s)) .
close
```

✔

```
open 2P-MUTEX .
  op s : -> Sys .
  eq [:nonexec] : inv1(s) = true .
  eq (pc2(s) = cs) = false .
  red inv1(s) implies inv1(leave2(s)) .
close
```

✔

Each goal is discharged with **:apply** (**rd**).

```
-- leave2(S#Sys)
:imp [inv1] .
:def ctf-3 = :ctf{ eq pc2(S#Sys) = cs . }
:apply(ctf-3)
:apply (rd)
:apply (rd)
```



# How to specify Qlock

Observers

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

Observable values

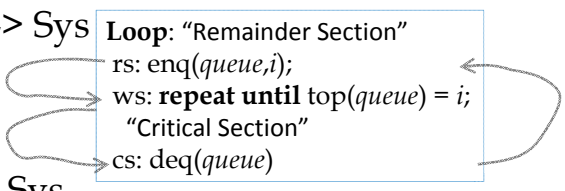
init

**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 specify Qlock

```
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

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

Since we have written proof scores to verify that Qlock enjoys the property, we know the following lemma:

```
eq inv2(S:Sys,I:Pid)
   = (pc(S,J) =  cs implies top(queue(S)) = J) .
```

Then, we tackle the following goal:

QLOCK |– {inv1(*S*,*I*,*J*) = true, inv2(*S*,*J*) = true}

in which there are two sentences to prove.

# Proving with CafeOBJ CITP

```
open QLOCK .
  red inv1(S:Sys,I:Pid,J:Pid) .
  red inv2(S:Sys,I:Pid) .
close
```

SI on S:Sys

want(s, k) … exit(s, k) …

init

```
open QLOCK .
  red inv1(init,I:Pid,J:Pid) .
  red inv2(init,I:Pid) .
close
```

try(s, k)

```
open QLOCK .
  op k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, I:Pid) = true .
  red inv1(try(s,k), I:Pid, J:Pid) .
  red inv2(try(s,k), I:Pid) .
close
```

```
open QLOCK .
:goal {
eq [inv1 :nonexec] :
   inv1(S:Sys,I:Pid,J:Pid) = true .
eq [inv2 :nonexec] :
   inv2(S:Sys,I:Pid) = true .
}

:ind on (S:Sys)
:apply (si)
```



# Proving with CafeOBJ CITP

```
open QLOCK .
  red inv1(init,I:Pid,J:Pid) .
  red inv2(init,I:Pid) .
close
```

TC

inv1

```
open QLOCK .
  ops i j : -> Pid .
  red inv1(init, i, j) .
close
```

✔

inv2

```
open QLOCK .
  op i  : -> Pid .
  red inv2(init, i) .
close
```

✔

```
-- init
:apply (tc)
-- inv1
:apply (rd)
-- inv2
:apply (rd)
```

# Proving with CafeOBJ CITP

```
open QLOCK .
  op k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, I:Pid) = true .
  red inv1(try(s,k), I:Pid, J:Pid) .
  red inv2(try(s,k), I:Pid) .
close
```

```
-- try(S#Sys, S#Pid)
:apply (tc)
```

TC

inv2

```
open QLOCK .
  ops i k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, I:Pid) = true .
  red inv2(try(s,k), i) .
close
```

inv1

```
open QLOCK .
  ops i j k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, I:Pid) = true .
  red inv1(try(s,k), i, j) .
close
```



# Proving with CafeOBJ CITP

```
open QLOCK .
  ops i j k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, I:Pid) = true .
  red inv1(try(s,k), i, j) .
close
```

IMP with inv1(s,i,j)

```
open QLOCK .
  ops i j k : -> Pid .
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, I:Pid) = true .
  red inv1(s, i, j) implies inv1(try(s,k), i, j) .
close
```

```
-- inv1
:imp [inv1]
   by {I:Pid <- I@Pid ;
       J:Pid <- J@Pid ;}
:def csb6 =
  :ctf{eq pc(S#Sys, S#Pid) = ws .}
:apply(csb6)
```

CS based on pc(s,k) = ws

pc(s,k) = ws

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

pc(s,k) ≠ ws

…

# Proving with CafeOBJ CITP

```
open QLOCK . ...
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, I:Pid) = true .
  eq pc(s, k) = ws .
  red inv1(s,i,j) implies inv1(try(s,k), i, j) .
close
```

CS based on top(queue(s)) = k

top(queue(s)) = k

```
open QLOCK . …
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, I: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 . …
  eq [:nonexec] : inv1(s, I:Pid, J:Pid) = true .
  eq [:nonexec] : inv2(s, I: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 …

top(queue(s)) ≠ k …

```
:def csb7 = :ctf{
    eq top(queue(S#Sys)) = S#Pid .}
:apply(csb7)
:def csb8 = :ctf{eq I@Pid = S#Pid .}
:def csb9 = :ctf{eq J@Pid = S#Pid .}
:apply(csb8 csb9)
:apply(rd)
```



# Proving with CafeOBJ CITP

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

CS based on pc(s,k) = cs

pc(s,k) = cs

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

CS based on i = k

i = k … ✔

i ≠ k

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

pc(s,k) ≠ cs …

```
-- exit(S#Sys, S#Pid)
…
-- inv2
:imp [inv2] by {I:Pid <- I@Pid ;}
:def csb18 = :ctf{
  eq pc(S#Sys, S#Pid) = cs .}
:apply(csb18)
:def csb19 = :ctf{eq I@Pid = S#Pid .}
:apply(csb19)
:apply(rd)
```

# Proving with CafeOBJ CITP

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

```
-- exit(S#Sys, S#Pid)
…
-- inv2
:imp [inv2] by {I:Pid <- I@Pid ;}
```

IMP with inv2(s,i)

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



# Proving with CafeOBJ CITP

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

```
:def csb18 = :ctf{
  eq pc(S#Sys, S#Pid) = cs .}
:apply(csb18)
:def csb19 = :ctf{eq I@Pid = S#Pid .}
:apply(csb19)
:apply(rd)
```

CS based on pc(s,k) = cs

pc(s,k) = cs

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

pc(s,k) ≠ cs …

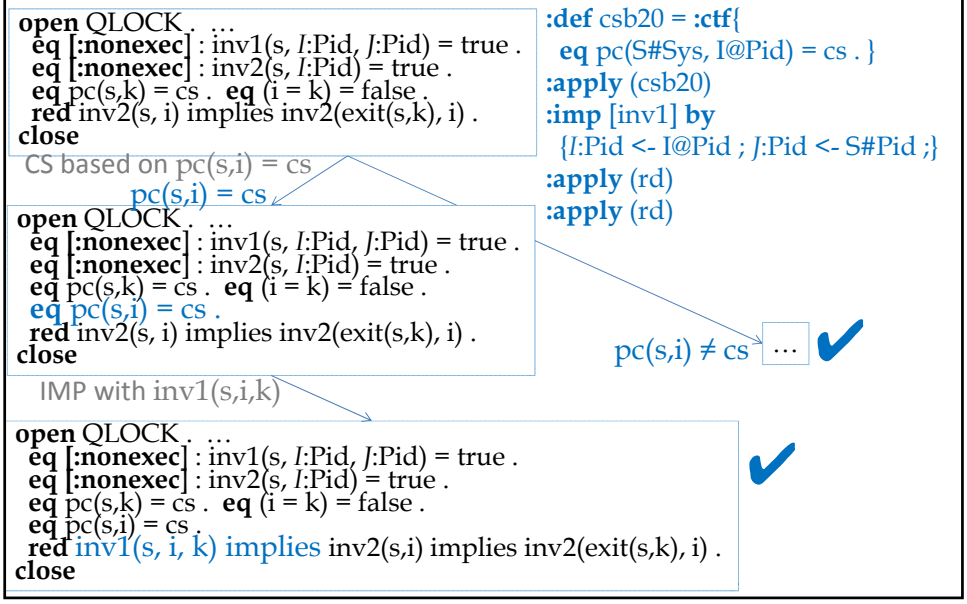
CS based on i = k

i = k … ✔

i ≠ k

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

# Proving with CafeOBJ CITP






# 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 verify that observational transition systems (OTSs) enjoy invariant properties with

*CafeOBJ Constructor-based Inductive Theorem Prover (CITP)*

- a proof assistant for CafeOBJ

based on how proof scores have been written.