Proof Score Writing for QLOCK in OTS/CafeOBJ

Lecture Note 05b

CafeOBJ Team for JAIST-FSSV2010

Topics

- How to write proof scores with derived proof rules by using Qlock as an example.
- Proof score templates.
- Case splitting & lemma conjecture/use.

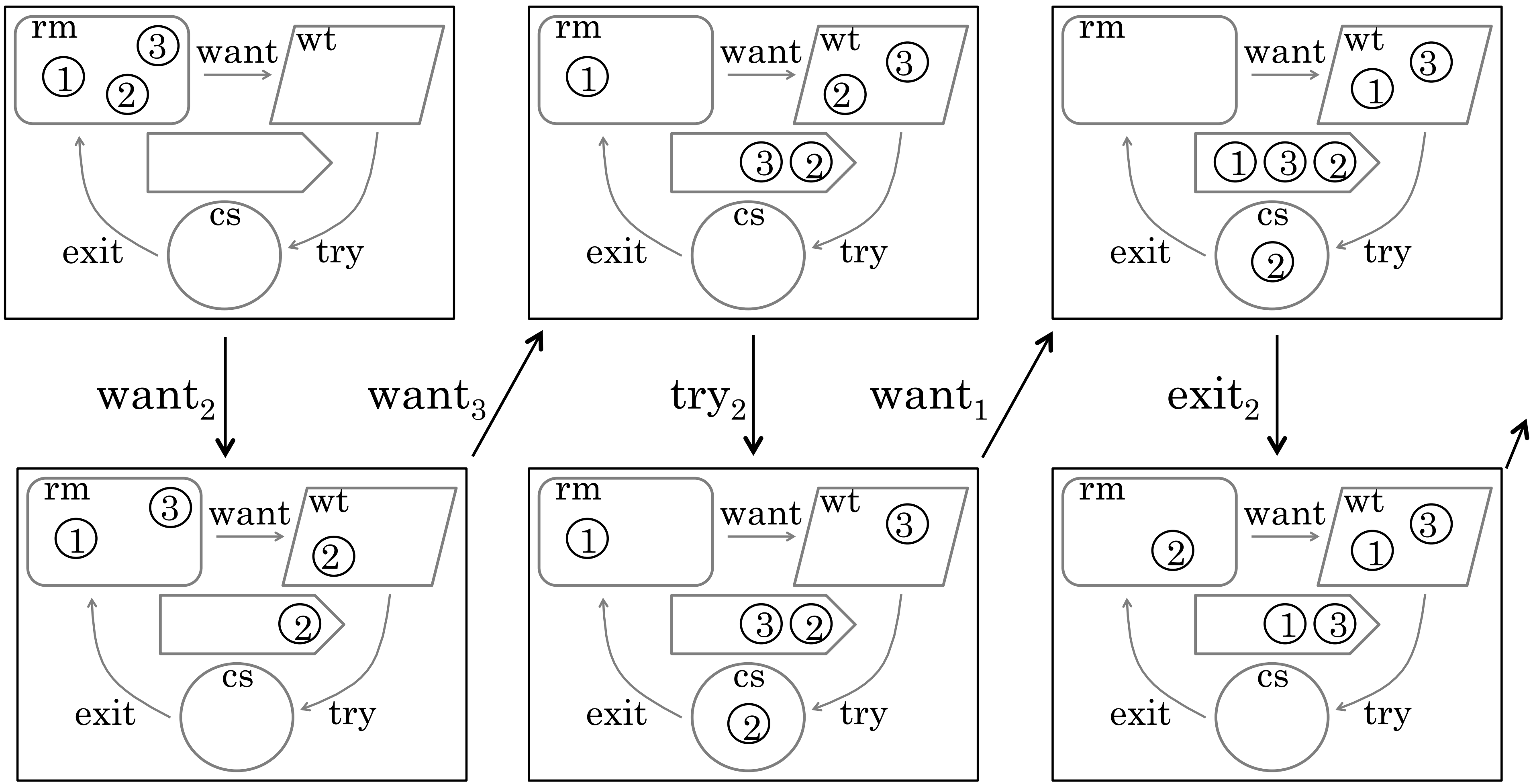
Qlock

- The pseudo-program executed by each process i can be written as follows:

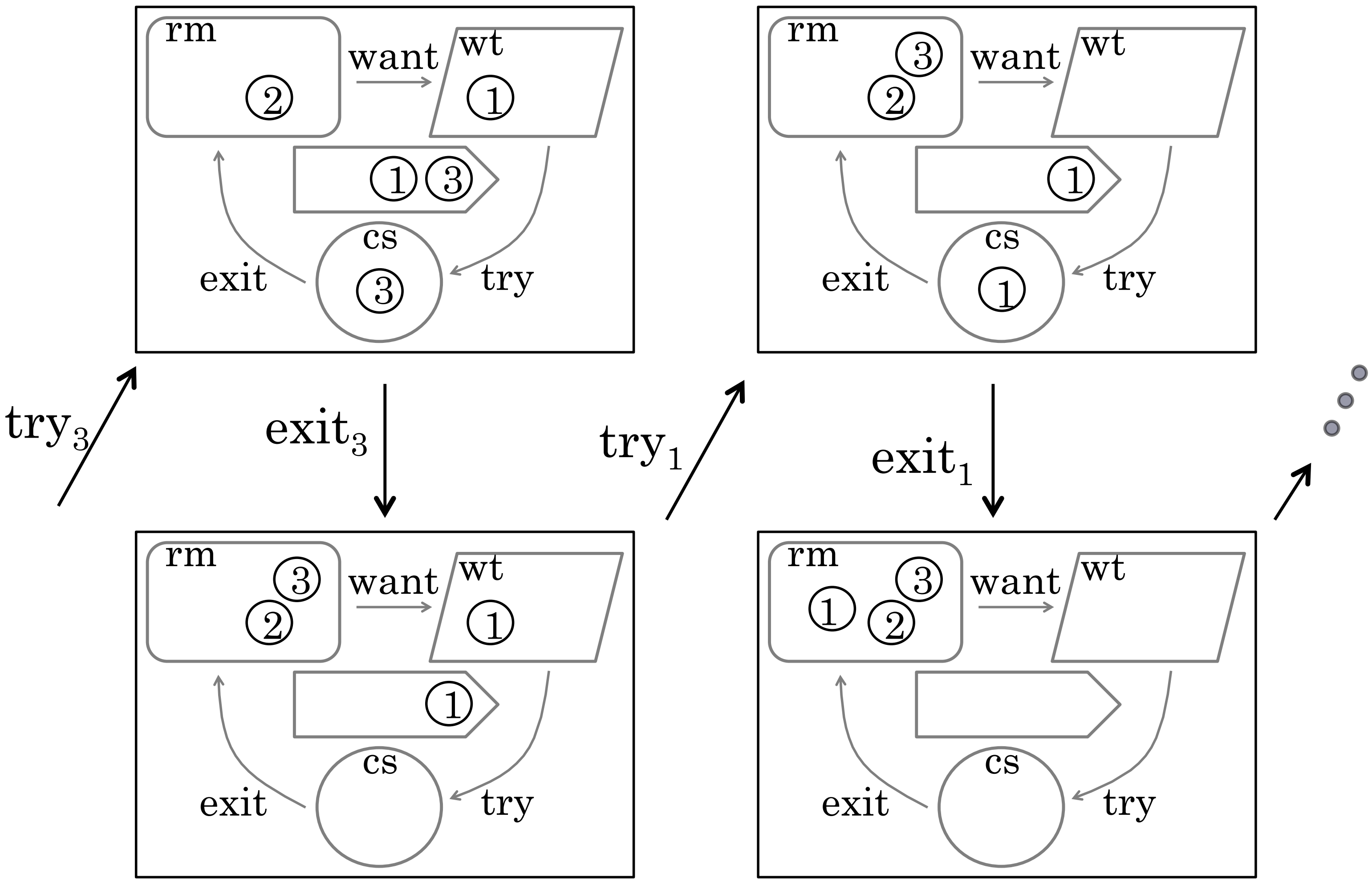
Loop
  Remainder Section
 rm: enq($queue$,i);
 wt: **wait until** top($queue$) = i;
  Critical Section
 cs: deq($queue$);

✓ $queue$ is the queue of process identifiers (Pids) shared by all processes.
✓ Initially, $queue$ is empty and each process is in Remainder Section (or at label rm).

Some Scenario of Qlock (1)



Some Scenario of Qlock (2)



Some Preparation for Verification

- The formalized property is referred as operator `inv1`.

```
mod* PRED-QLOCK {
  inc(QLOCK)
  op inv1 : Sys Pid Pid -> Bool
  var S : Sys
  vars I J : Pid
  eq inv1(S,I,J)
   = (pc(S,I) = cs and pc(S,J) = cs implies I = J) .
}
```

- Our goal is

  G: `PRED-QLOCK` |– (∀ `S:Sys`)(∀ `I,J:Pid`) `inv1` (`S,I,J`)

First Thing to Do

Write a proof score template!

- Suppose that our goals is

$$\texttt{PRED-LOCK} \vdash (\forall S{:}\texttt{Sys})(\forall X{:}\texttt{Pid})p(S,X).$$

```
open PRED-QLOCK
  red p(S:Sys,X:Pid) .
close
```

- To this end, what we do is approximately
  - to apply the structural induction scheme of `Sys` on `S`,
  - for each induction case for transition function t (`want`, `try` and `exit`), to conduct case splitting based on `c-`t, and
  - for the case (sub-goal) such that `c-`t holds, the equation

    "`c-`t`(s,k) = true`"

    is transformed into other equivalent equations.

Structural Induction Scheme of `Sys`

$$\frac{\texttt{QLOCK} \vdash_{\{x\}} p(\texttt{init},\texttt{x}) \qquad \begin{array}{l} \texttt{QLOCK} \cup \{(\forall X)(p(s,X) = \texttt{true})\} \vdash_{\{s,k,x\}} p(\texttt{want}(s,k),x) \\ \texttt{QLOCK} \cup \{(\forall X)(p(s,X) = \texttt{true})\} \vdash_{\{s,k,x\}} p(\texttt{try}(s,k),x) \\ \texttt{QLOCK} \cup \{(\forall X)(p(s,X) = \texttt{true})\} \vdash_{\{s,k,x\}} p(\texttt{exit}(s,k),x) \end{array}}{\texttt{QLOCK} \vdash (\forall S)(\forall X)p(S,X)}$$

```
open PRED-QLOCK
  red p(S:Sys,X:Pid) .
close
```

⇩

```
open PRED-QLOCK
 op x : -> Pid .
 red p(init,x) .
close
```

• • •

```
open PRED-QLOCK
 op s : -> Sys .
 ops k x : -> Pid .
 --> eq p(s,x) = true .
 red p(try(s,k),x) .
close
```

• • •

Elimination of Object-level Implications

$$\frac{S \cup \{(\forall X)(q = \mathtt{true})\} \vdash q[X \leftarrow t] \ \mathtt{implies}\ p}{S \cup \{(\forall X)(q = \mathtt{true})\} \vdash p}$$

```
open PRED-QLOCK
 op s : -> Sys .
 ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 red p(try(s,k),x) .
close
```

⇩

```
open PRED-QLOCK
 op s : -> Sys .
 ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 red p(s,x) implies p(try(s,k),x) .
close
```

Case Splitting on Effective Conditions

```
open PRED-QLOCK
 op s : -> Sys .
 ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 red p(s,x) implies p(try(s,k),x) .
close
```

$$\frac{S \cup \{q = \texttt{true})\} \vdash p \qquad S \cup \{q = \texttt{false})\} \vdash p}{S \vdash p}$$

By Binary Case Analysis

```
open PRED-QLOCK
 op s : -> Sys .
 ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 eq c-try(s,k) = true .
 red p(s,x)
     implies p(try(s,k),x) .
close
```

```
open PRED-QLOCK
 op s : -> Sys .
 ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 eq c-try(s,k) = false .
 red p(s,x)
     implies p(try(s,k),x) .
close
```

Transitivity in Specification

$$\frac{t_1 = t_3 \vdash p}{\{t_1 = t_2,\ t_2 = t_3\} \vdash p}$$

```
open PRED-QLOCK
 op s : -> Sys .  ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 eq c-try(s,k) = true .
 red p(s,x) implies p(try(s,k),x) .
close
```

⇩

```
eq c-try(S,I)
= (pc(S,I) = wt and top(queue(S)) = I) .
```

```
open PRED-QLOCK
 op s : -> Sys .  ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 eq (pc(s,k) = wt and top(queue(s)) = k) = true .
 red p(s,x) implies p(try(s,k),x) .
close
```

Introduction of Conjunction in Spec

$$\frac{S \cup \{q = \mathtt{true}, r = \mathtt{true}\} \vdash p}{S \cup \{q \ \mathtt{and}\ r = \mathtt{true}\} \vdash p}$$

```
open PRED-QLOCK
 op s : -> Sys .  ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 eq (pc(s,k) = wt and top(queue(s)) = k) = true .
 red p(s,x) implies p(try(s,k),x) .
close
```

```
open PRED-QLOCK
 op s : -> Sys .  ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 eq (pc(s,k) = wt) = true .
 eq (top(queue(s)) = k) = true .
 red p(s,x) implies p(try(s,k),x) .
close
```

Introduction of Object-level Eq in Spec

$$\frac{S \cup \{t_1 = t_2\} \vdash p}{S \cup \{(t_1 = t_2) = \mathtt{true}\} \vdash p}$$

```
open PRED-QLOCK
 op s : -> Sys .  ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 eq (pc(s,k) = wt) = true .
 eq (top(queue(s)) = k) = true .
 red p(s,x) implies p(try(s,k),x) .
close
```

⇩

```
open PRED-QLOCK
 op s : -> Sys .  ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 eq pc(s,k) = wt .
 eq top(queue(s)) = k .
 red p(s,x) implies p(try(s,k),x) .
close
```

Elimination of Queue Constructor

$$\frac{S \cup \{queue = q\ ,\ elt\} \vdash_{\{q\}} p}{S \cup \{\mathtt{top}(queue) = elt\} \vdash p} \quad \text{if } S \text{ contains } \mathtt{QUEUE}$$

```
open PRED-QLOCK
 op s : -> Sys .  ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 eq pc(s,k) = wt .
 eq top(queue(s)) = k .
 red p(s,x) implies p(try(s,k),x) .
close
```

⇩

```
open PRED-QLOCK
 op s : -> Sys . ops k x : -> Pid . op q : -> Queue.
 --> eq p(s,X:Pid) = true .
 eq pc(s,k) = wt .
 eq queue(s) = q , k .
 red p(s,x) implies p(try(s,k),x) .
close
```

Replacement of Equation with Lemma

$$\frac{\mathrm{S} \cup \{queue = q \; , \; elt\} \vdash_{\{q\}} p}{\mathrm{S} \cup \{\mathtt{top}(queue) = elt\} \vdash p} \quad \textbf{if } \mathrm{S} \text{ contains } \mathtt{QUEUE}$$

✓ This is an instance of the following proof rule:

$$\frac{\mathrm{S} \cup \{l_2[X \leftarrow a] = r_2[X \leftarrow a]\} \vdash_{\{a\}} p}{\mathrm{S} \cup \{l_1 = r_1\} \vdash p} \quad \textbf{if } \mathrm{S} \vdash (\exists X)(l_2(X) = r_2(X)) \text{ if } l_1 = r_1$$

Let $l_1 = r_1$ be $\mathtt{top}(queue) = elt$ and $l_2(X) = r_2(X)$ be $queue = (Q \; , \; elt)$.

$$\mathrm{S} \vdash (\exists Q)(queue = (Q \; , \; elt)) \text{ if } \mathtt{top}(queue) = elt \; \ldots \; (1)$$

If $queue$ is `empty`, (1) vacuously holds.
If $queue$ is $(q \; , \; e)$ and e does not equal elt, (1) vacuously holds.
If $queue$ is $(q \; , \; e)$ and e equals elt, q is a witness.

(1) holds.

Proof Score Template

```
open PRED-QLOCK
 op x : -> Pid .
 red p(init,x) .
close
      •••
open PRED-QLOCK
 op s : -> Sys . ops k x : -> Pid . op q : -> Queue.
 --> eq p(s,X:Pid) = true .
 eq pc(s,k) = wt .
 eq queue(s) = q , k .
 red p(s,x) implies p(try(s,k),x) .
close
open PRED-QLOCK
 op s : -> Sys . ops k x : -> Pid .
 --> eq p(s,X:Pid) = true .
 eq c-try(s,k) = false .
 red p(s,x) implies p(try(s,k),x) .
close
      •••
```

Applying PST to `inv1`

```
open PRED-QLOCK
 ops i j : -> Pid .
 red inv1(init,i,j) .
close
      •••
open PRED-QLOCK
 op s : -> Sys . ops k i j : -> Pid . op q : -> Queue.
 --> eq inv1(s,I:Pid,J:Pid) = true .
 eq pc(s,k) = wt .
 eq queue(s) = q , k .
 red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
open PRED-QLOCK
 op s : -> Sys . ops k x : -> Pid .
 --> eq inv1(s,I:Pid,J:Pid) = true .
 eq c-try(s,k) = false .
 red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
      •••
```

Two Main Tasks to Complete Proof Scores

- Case splitting
  - For a proof passage for which CafeOBJ returns neither `true` nor `false`, select a term on which case splitting is done.
  - Such a term may be found in a result returned by CafeOBJ and/or properties to verify.
- Lemma conjecture/use
  - For a proof passage in which a contradiction exists, conjecture a lemma (or another state predicate).
  - If CafeOBJ returns `false` for a proof passage, then there exists a contradiction in it or the property to verify does not hold.
  - Some scenarios of a system surely help you conjecture lemmas.

Case Splitting (1)

```
open PRED-QLOCK
 op s : -> Sys . ops k i j : -> Pid . op q : -> Queue.
 --> eq inv1(s,I:Pid,J:Pid) = true .
 eq pc(s,k) = wt .
 eq queue(s) = q , k .
 red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
```

⇩ By Binary Case Analysis & Intro Object-level Eq in Spec

```
open PRED-QLOCK
 …
 eq pc(s,k) = wt . eq queue(s) = q , k .
 eq i = k .
 red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
open PRED-QLOCK
 …
 eq pc(s,k) = wt . eq queue(s) = q , k .
 eq (i = k) = false .
 red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
```

Case Splitting (2)

```
open PRED-QLOCK
 …
 eq pc(s,k) = wt . eq queue(s) = q , k .
 eq i = k .
 red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
```

⇩ By Binary Case Analysis & Intro Object-level Eq in Spec

```
open PRED-QLOCK
 …
 eq pc(s,k) = wt . eq queue(s) = q , k .
 eq i = k . eq j = k .
 red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
open PRED-QLOCK
 …
 eq pc(s,k) = wt . eq queue(s) = q , k .
 eq i = k . eq (j = k) = false .
 red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
```

Lemma Conjecture

```
open PRED-QLOCK
 …
 eq pc(s,k) = wt . eq queue(s) = q , k .
 eq i = k . eq (j = k) = false .
 red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
```

✓ CafeOBJ returns `(pc(s,j) = cs) xor true`.
✓ If `pc(s,j)` is `cs`, then the result becomes `false`, which means that the assumption seems to contradict the four equations.
✓ Hence, `pc(s,j)` should not be `cs` in any reachable state characterized by the four equations.
✓ The discussion lets us conjecture

```
op inv2 : Sys Pid -> Bool
eq inv2(S,I)
 = (pc(S,I) = cs implies top(queue(S)) = I) .
```

✓ This can be observed in the scenario on p.4 & p.5.

Simultaneous Struct Ind Scheme of `Sys`

$$\left[\begin{array}{l}
\texttt{QLOCK} \vdash_{\{x_i\}} p_i(\texttt{init}, x_i) \\
\texttt{QLOCK} \cup \{(\forall X_j)(p_i(s, X_j) = \texttt{true}) \text{ for } j = 1,\ldots,n\} \vdash_{\{s,k,x_i\}} p_i(\texttt{want}(s,k), x_i) \\
\texttt{QLOCK} \cup \{(\forall X_j)(p_i(s, X_j) = \texttt{true}) \text{ for } j = 1,\ldots,n\} \vdash_{\{s,k,x_i\}} p_i(\texttt{try}(s,k), x_i) \\
\texttt{QLOCK} \cup \{(\forall X_j)(p_i(s, X_j) = \texttt{true}) \text{ for } j = 1,\ldots,n\} \vdash_{\{s,k,x_i\}} p_i(\texttt{exit}(s,k), x)
\end{array}\right] \text{ for } i = 1,\ldots,n$$

$$\overline{\texttt{QLOCK} \vdash (\forall S)(\forall X_l) p_l(S, X_l) \quad \text{where } l \in \{1,\ldots,n\}}$$

✓ instead of $(\forall$ `S`$)(\forall$ `I,J`$)$`inv1`$($`S,I,J`$)$, we prove $(\forall$ `S`$)(\forall$ `I,J`$)$`inv1`$($`S,I,J`$)$ and $(\forall$ `S`$)(\forall$ `I`$)$`inv2`$($`S,I`$)$ simultaneously with this induction scheme.

✓The PS written so far for `inv1` can be reused.

✓ All needed to do is to add `inv2(s,I:Sys) = true` to the PPs of the ICs.

✓ The PST can be applied to `inv2`, but `inv1(s,I:Sys,J:Sys) = true` is added to the PPs of the ICs.

✓ `inv2` is not used as an ordinary lemma. We abuse term *lemma* to refer to another property such as `inv2` used in this induction scheme.

Lemma Use (1)

```
open PRED-QLOCK
 op s : -> Sys . ops k i j : -> Pid . op q : -> Queue.
 --> eq inv1(s,I:Pid,J:Pid) = true .
 --> eq inv2(s,I:Pid) = true .
 eq pc(s,k) = wt . eq queue(s) = q , k .
 eq i = k. eq (j = k) = false .
 red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
```

⇩ By Elimination of Object-level Implications

```
open PRED-QLOCK
 op s : -> Sys . ops k i j : -> Pid . op q : -> Queue.
 --> eq inv1(s,I:Pid,J:Pid) = true .
 --> eq inv2(s,I:Pid) = true .
 eq pc(s,k) = wt . eq queue(s) = q , k .
 eq i = k. eq (j = k) = false .
 red inv2(s,j) implies
     inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
```

Lemma Use (2)

```
open PRED-QLOCK
 op s : -> Sys . ops k i : -> Pid .
 --> eq inv1(s,I:Pid,J:Pid) = true .
 --> eq 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
```

⇩ By Elimination of Object-level Implications

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

- ✓ if `pc(s,i)` is `cs`, then we need to check if `top(queue(exit(s,k)))` is `i`.
- ✓ But, `inv1` can eliminate the possibility that `pc(s,i)` is `cs`.

Case Splitting on Queue Constructors

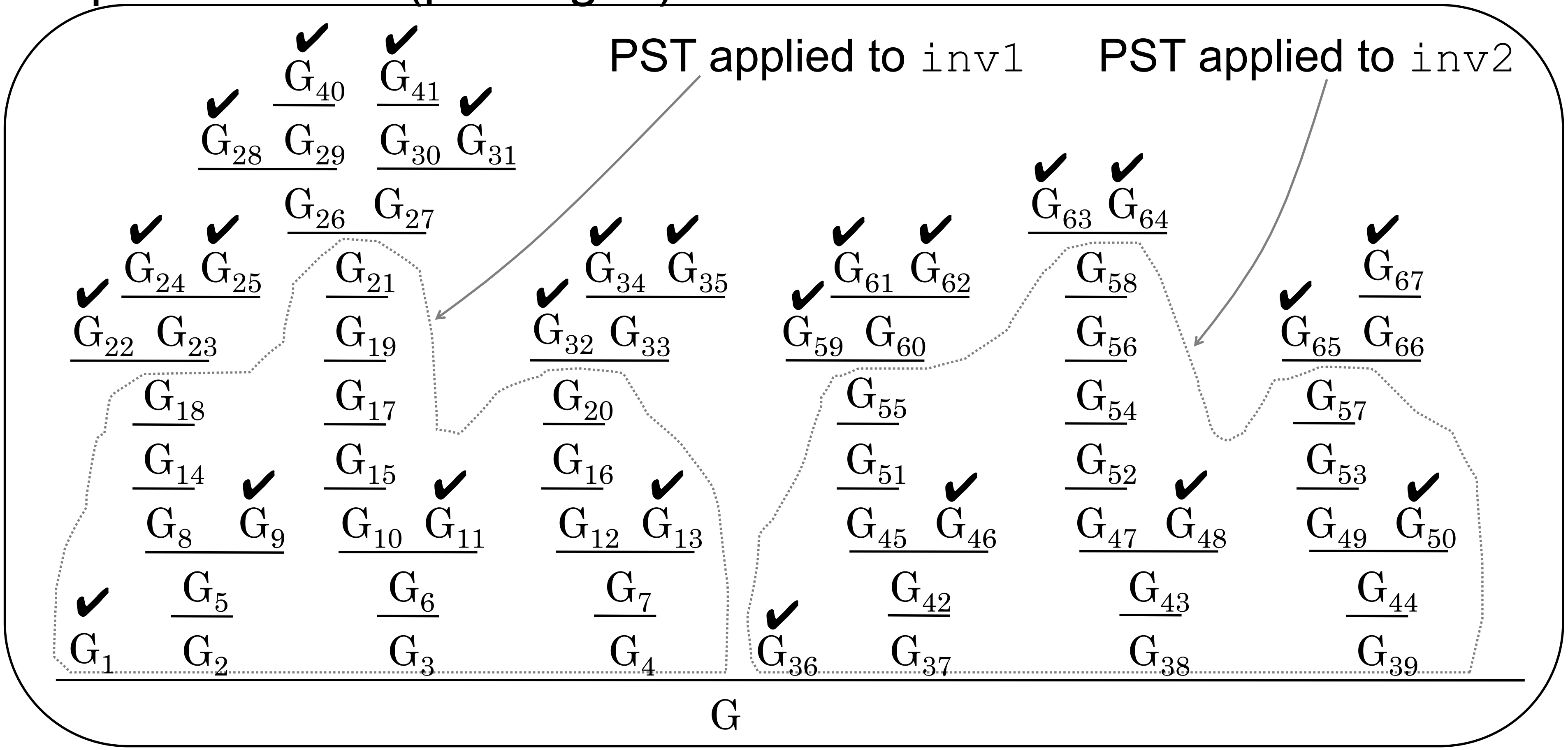
$$\frac{S \cup \{queue = \mathrm{empty}\} \vdash p \quad S \cup \{queue = q\ ,\ a\} \vdash_{\{q,a\}} p}{S \vdash p}$$ **if** S contains `QUEUE`

```
open PRED-QLOCK
 op s : -> Sys . ops k i : -> Pid .
 --> eq inv1(…) = true . --> eq inv2(…) = true .
 eq pc(s,k) = rm . eq (i = k) = false .
 eq queue(s) = empty .
 red inv2(s,i) implies inv2(want(s,k),i) .
close

open PRED-QLOCK
 op s : -> Sys . ops k i l : -> Pid . op q : -> Queue .
 --> eq inv1(…) = true . --> eq inv2(…) = true .
 eq pc(s,k) = rm . eq (i = k) = false .
 eq queue(s) = q , l.
 red inv2(s,i) implies inv2(want(s,k),i) .
close
```

Proof Tree of G

- See file `proof.mod` describing the entire process for constructing the proof tree of G and the corresponding proof scores (passages) in CafeOBJ.



Housekeeping for Proof Score Writing (1)

- Fresh constants are often declared and long terms are often used.
- To avoid this, two modules are declared:

```
mod* BASE-QLOCK { inc(PRED-QLOCK)
  ops s s' : -> Sys  ops i j k : -> Pid
}

mod* ISTEP-QLOCK {  inc(BASE-QLOCK)
  op istep1 : -> Bool
  op istep2 : -> Bool
  eq istep1 = inv1(s,i,j) implies inv1(s',i,j) .
  eq istep2 = inv2(s,i) implies inv2(s',i) .
  " eq inv1(s,I:Pid,J:Pid) = true .
    eq inv2(s,I:Pid) = true . "
}
```

Housekeeping for Proof Score Writing (2)

- Some proof passages:

```
open BASE-QLOCK
  red inv1(init,i,j) .
close
```

```
open ISTEP-QLOCK
  op k : -> Pid .
  eq pc(s,k) = rs .
  eq i = k .
  eq s' = want(s,k) .
  red istep1 .
close
```

✓ See file `proof1.mod` describing the proof score of `inv1`.
✓ See file `proof2.mod` describing the proof score of `inv2`.
✓ See file `template.mod` describing a proof score template of `QLOCK`.

Summary

- Qlock has been used to describe how to write proof scores with derived proof rules.
  - First write a proof score template for a system.
    - ✓ It can be systematically written.
    - ✓ It can be used for any (invariant) properties of the system.
  - Then do case splitting & conjecture/use lemmas.
    - ✓ You need to select a term on which case splitting is done.
    - ✓ Such a term can be found in the result & the property.
    - ✓ If you notice a contradiction in a proof passage, you can conjecture a lemma.
    - ✓ Some scenarios of the system surely help you conjecture lemmas.