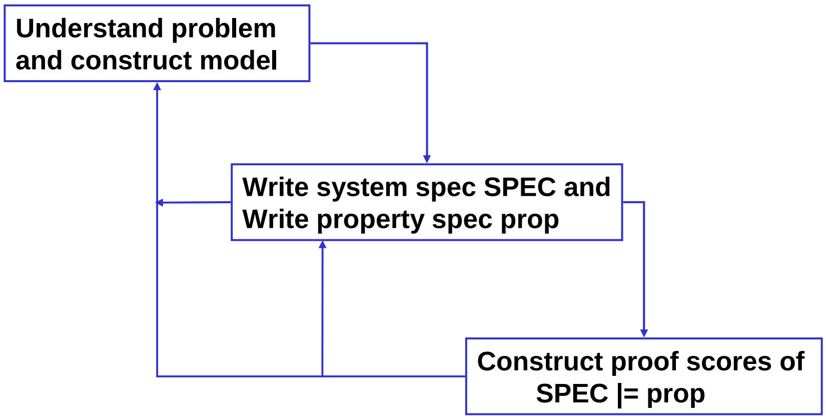
Verification of QLOCK with Proof Score

CafeOBJ Team of JAIST

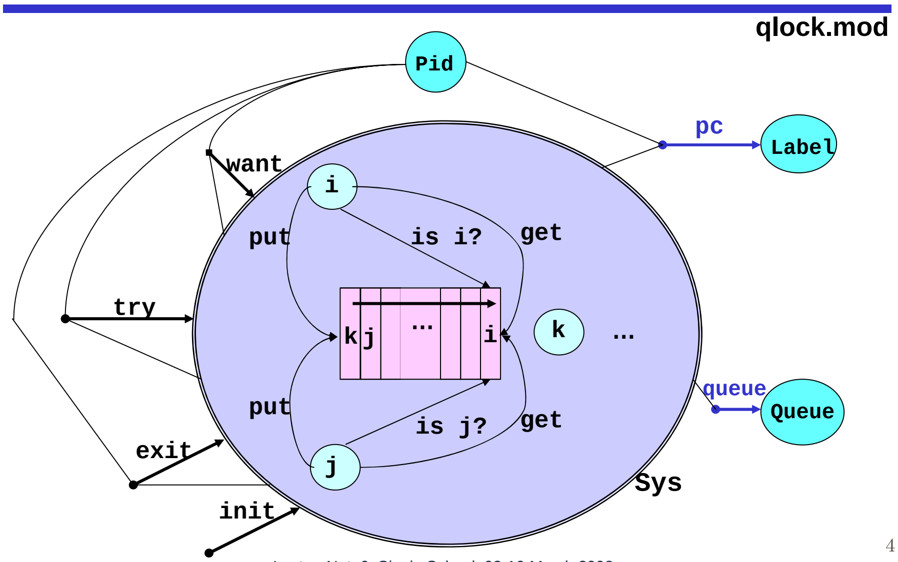
Topics

- **Verification method for SPEC |= prop with Proof Score**
- **QLOCK example is used to explain the method**

Flow Chart for Verification with Proof Scores in CafeOBJ



Signature Diagram of QLOCKviaOTS



R_{QLOCK} (set of reachable states) of OTS_{QLOCK} (OTS defined by the module QLOCK)

qlock.mod

Signature determining R_{QLOCK}

```
-- any initial state
  op init : -> Sys
-- actions
  bop want : Sys Pid -> Sys
  bop try  : Sys Pid -> Sys
  bop exit : Sys Pid -> Sys
```

Recursive definition of R_{QLOCK}

$$
\begin{aligned}
R_{QLOCK} = \{\mathtt{init}\} \; \cup \; & \{\mathtt{want}(s,i) \mid s \in R_{QLOCK}, i \in \mathtt{Pid}\} \; \cup \\
& \{\mathtt{try}(s,i) \mid s \in R_{QLOCK}, i \in \mathtt{Pid}\} \; \cup \\
& \{\mathtt{exit}(s,i) \mid s \in R_{QLOCK}, i \in \mathtt{Pid}\}
\end{aligned}
$$

Formalization of mutual exclusion property as an invariant

invariants-0.mod

```
mod INV1 {
   pr(QLOCK)
-- declare a predicate to verify to be an invariant
   pred inv1 : Sys Pid Pid
-- CafeOBJ variables
   var S :  Sys .
   vars I J : Pid .
-- define inv1 to be the mutual exclusion property
   eq inv1(S,I,J)
       = (((pc(S,I) = cs) and (pc(S,J) = cs)) implies I = J) .
}
```

Formulation of proof goal for mutual exclusion property

$$\mathtt{INV1} \models \forall s \in R_{QLOCK} \forall i,j \in \mathtt{Pid}.\mathtt{inv1}(s,i,j)$$

Theorem of constants

```
INV1 |= ∀s∈Sys∀i,j∈Pid.inv1(s,i,j)
```

||def

```
INV1 |= inv1(s:Sys,i:Pid,j:Pid)
```

||TC

```
INV1∪{op s : -> Sys}∪{ops i j : -> Pid} |= inv1(s,i,j)
```

All the above three state:
for any model (or **INV1**-algebra) andany objects **s:Sys, i:Pid, j:Pid**, the proposition **inv1(s,i,j)** holds.

Fresh constants and variables: Theorem of Constants

Let **p'** be the boolean term obtained from a boolean term **p** by replacing all variables in **p** with "fresh constants" which do not appear anywhere in the specification **SP=<Σ,E>** then
(TC) Σ,E |= **p** iff **(Σ U C)**,E |= **p'**
where **C** is the set of the introduced fresh constants.

Notice that different occurrences of a "same variable" should be replaced with a same fresh constant. Notice also that a fresh constant introduced will have much longer scope than the original variable.

Constants and variables: Differences

(model/interpretation is assumed to be fixed and p is assumed to contain no variables in $\Sigma,E|=$ p)

A variable of the same name which appears in the same equation in $\Sigma,E|=$ **p** denotes arbitrarily but the same object.

A variable of the same name which appears in several different equations in $\Sigma,E|=$ **p** denotes any object independently, and does not necessary denote the same object.

A constant of the same name which appears in several different places in $\Sigma,E|=$ **p** denotes the same object, because a constant constitutes the signature Σ.

Assertion and Proof Passage

proof-00.mod

Assertion

```
INV1 ∪
{op s : -> Sys} ∪
{ops i j : -> Pid}
  |=
inv1(s,i,j)
```

Logical Statement of stating that Specification satisfies property

~

Proof Passage

```
-- [mx]*
open INV1
 op s : -> Sys .
 ops i j : -> Pid .
-- |=
 red inv1(s,i,j) .
close
```

Logical Statement and CafeOBJ Code

If reduction part of the CafeOBJ code returns `true` then the assertion holds

If `[mx]*` returns `true` the verification is done, but ...

```
-- [mx]*
open INV1
 op s : -> Sys .
 ops i j : -> Pid .
-- |=
 red inv1(s,i,j) .
close
```

This assertion states that mutual exclusion property holds for any reachable states.

If this proof passage returns **true** the game is over. But it does not return **`true`**.

A name of assertion/proof-passage(p.-p.) which ends with the character **`*`** (like **`[mx]*`**) indicates that it does not return **`true`**. An assertion/p.-p. which return **`true`** is called to be effective. An effective p.p. has a name without **`*`** at the end.

Proof Scores and Assertion Splitting Rules (1)

The goal of verification of an assertion **A** via proof score is to get a set of assertions/p.-p.s **`{A1,A2,…An}`** such that:
 (1) (**`A1 and A2 and … and An)`** implies **A**, and
 (2) All the **`A1,A2,…,An`** are effective.
A set of assertions which satisfies (1) and (2) is called a proof score (in a narrow sense) for **A**.

Proof Scores and Assertion Splitting Rules (2)

For constructing proof scores, several kinds of assertion splitting rules of the form:

`(Ai1 and Ai2 and … and Ain)` implies **`Ai`**

are used. An assetion splitting rule is also written as:

`{Ai1,Ai2, … Ain} implies Ai .`

If **`n = 1`** the assertion splitting rule is also called assertion transformation rule and is written like:

`Ai1 implies A .`

Assertion Splitting via Induction Scheme induced by R_{QLOCK}

proof-01.mod

$$
\begin{aligned}
R_{QLOCK} = \{\mathtt{init}\} \cup & \\
& \{\mathtt{want}(s,i) \mid s \in R_{QLOCK}, i \in \mathtt{Pid}\} \cup \\
& \{\mathtt{try}(s,i) \mid s \in R_{QLOCK}, i \in \mathtt{Pid}\} \cup \\
& \{\mathtt{exit}(s,i) \mid s \in R_{QLOCK}, i \in \mathtt{Pid}\}
\end{aligned}
$$

In `[mx]*, s : -> Sys` means `s : ->` R_{QLOCK}, and the following induction scheme follows.

Induction Scheme (Assertion Splitting via I.S.)

```
{[1-init],[1-want]*,[1-try]*,[1-exit]*}
          implies [mx]*
```

Assertion Splitting via Case Splitting

proof-02.mod

For any **`INV1`**-algebra (a model of **`INV1`**) c-want(s,k) is either true or false. Hence the following assertion splitting is justified.

Assertion Splitting via Case Splitting
`{[1-want,c-w]*, [1-want,~c-w]}` **`implies [1-want]*`**

(CS) $\{ (\mathbf{E} \models (p_1 \text{ or } p_2)), (\mathbf{E} \cup \{p_1\text{=true}\} \models p) , (\mathbf{E} \cup \{p_2\text{=true}\} \models p) \}$ implies $\mathbf{E} \models p$

Meta Level Equation and Object Level Equation

(EQ) $\mathbf{E} \cup \{ t_1 = t_2 \} \models p$ iff $\mathbf{E} \cup \{ (t1 = t2) = \text{true} \} \models p$

(p = true) iff p

```
--> [1-want,c-w-org]*
open INV1
 op s : -> Sys .
 ops i j k : -> Pid .
 eq inv1(s,I:Pid,J:Pid) = true .
 eq c-want(s,k) = true .
-- |=
 red inv1(want(s,k),i,j) .
close
```

as assertion ⟷ **iff**

as p.p. ⟹ **EQ**

```
--> [1-want,c-w]*
open INV1
 op s : -> Sys .
 ops i j k : -> Pid .
 eq inv1(s,I:Pid,J:Pid) = true .
 -- eq c-want(s,k) = true .
 eq pc(s,k) = rm .
-- |=
 red inv1(want(s,k),i,j) .
close
```

Assertion Transformation: INST, TRANS, HIDE, IMP

proof-4.mod

The proof passage **`[1-want,c-w,~i=k,~j=k]*`** returns a boolean term witch is equivalent to **`inv1(s,i,j).`** This should return **`true`** because the boolean term is an instance of **`inv1(s,I:Pid,J:Pid)`** which is declared in the premise part (the part before **`|=)`** of this proof passage. This assertion/proof-passage is transformed to the effective one (the one which return true) by using INST, TRANS, and HIDE transforming rules.

INST

```
-- [1-want,c-w,~i=k,~j=k]*
open INV1
   op s : -> Sys . ops i j k : -> Pid .
   eq inv1(s,I:Pid,J:Pid) = true .
   -- eq c-want(s,k) = true .
   eq pc(s,k) = rm .
   eq (i = k) = false .
   eq (j = k) = false .
-- |=
   red inv1(want(s,k),i,j) .
close
```

$\longleftrightarrow$ **iff**

$\Longrightarrow$ **INST**

```
-- [1-want,c-w,~i=k,~j=k,inst]*
open INV1
   op s : -> Sys . ops i j k : -> Pid .
   eq inv1(s,I:Pid,J:Pid) = true .
   eq inv1(s,i,j) = true .  **
   -- eq c-want(s,k) = true .
   eq pc(s,k) = rm .
   eq (i = k) = false .
   eq (j = k) = false .
-- |=
   red inv1(want(s,k),i,j) .
close
```

TRANS, HIDE

iff

TRANS

```
-- [1-want,c-w,~i=k,~j=k,inst,trans]*
open INV1
   op s : -> Sys . ops i j k : -> Pid .
   eq inv1(s,I:Pid,J:Pid) = true .
   -- eq c-want(s,k) = true .
   eq pc(s,k) = rm .
   eq (i = k) = false .
   eq (j = k) = false .
-- |=
   red inv1(s,i,j) implies
         inv1(want(s,k),i,j) .   **
close
```

including comments

iff

Comment-out-ed equations are effective for “assertions”

excluding comments

implies

HIDE

```
-- [1-want,c-w,~i=k,~j=k,inst,trans,hide]
open INV1
   op s : -> Sys . ops i j k : -> Pid .
   -- eq inv1(s,I:Pid,J:Pid) = true .   **
   -- eq c-want(s,k) = true .
   eq pc(s,k) = rm .
   eq (i = k) = false .
   eq (j = k) = false .
-- |=
   red inv1(s,i,j) implies
         inv1(want(s,k),i,j) .
close
```


Some basic properties of E |= p (1)
(term or equation which moves across |= is assumed to include no variables)

Let `t1’` and `t2’` be the terms obtained from (a boolean: error) terms `t1` and `t2` by replacing variables in `t1` and `t2` with corresponding ground terms respectively then:

(INST) (**E** U {t1=t2})|= p iff
(**E** U {t1=t2} U {t1’=t2’})|= p

(TRANS) **E** |= ((t_1 = t_2) implies p) iff
E U { t_1 = t_2 } |= p

Some basic properties of E |= p (2)

(term or equation which moves across |= is assumed to include no variables)

(HIDE) (**E** U {t1=t2})|= p implies (**E** U {t1=t2} U {t3=t4})|= p

This justifies to comment out any equation (removing by making it comment) at any moment. (as a p.-p.)

(IMP) **E** |= (t_1 = t_2) implies (**E** U { t_1 = t_2 } |= p iff **E** |= p)

21

LectureNote6, Sinaia School, 03-10 March 2008

module ISTEP1

invarients-1.mod, proof-05.mod->proof-06.mod

```
red inv1(s,i,j) implies inv1(want(s,k),i,j) .
```

```
eq istep1(I:Pid,J:Pid) =
   inv1(s,I,J) implies inv1(s',I,J) .
```

```
red istep1(i,j).
```

Notice that using INST,TRANS,HIDE, and **`istep1`**,

```
ops k l : -> Pid .  …
-- |=
red inv1(s,k,l) implies istep1(i,j) .
```

can also be used instead of instead of **`istep1(i,j)`** for any **`k and l`**.

22

LectureNote6, Sinaia School, 03-10 March 2008

Simultaneous Induction Scheme

simultaniousIS.txt,invariants-2.mod, proof-09.mod->proof-10.mod

Lemma Discovery/Introduction:
Invariant **inv2** is decided to be a lemma to be introduced.

Simultaneous Induction Scheme

**{ [1-init], [1-want2]*, [1-try2]*, [1-exit2]*,
[2-init], [2-want2]*, [2-try2]*, [2-exit2]* }
implies [mx]***

**{ [1-init], [1-want2]*, [1-try2]*, [1-exit2]*,
[2-init], [2-want2]*, [2-try2]*, [2-exit2]* }
implies} [inv2]***

Lemma Usage: Invariant predicate **inv2** can be declared in the premise part of any assertion/proof-passage after the introduction.

**[1-init]
[1-want,c-w,i=k]
[1-want,c-w,~i=k,j=k]
[1-want,c-w,~i=k,~j=k,istep1]
[1-want,~c-w]
[1-try,c-t,i=k,j=k]
[1-try,c-t,i=k,~j=k]*
[1-try,c-t,~i=k]*
[1-try,~c-t]
[1-exit]***

**[1-init]
[1-want,c-w,i=k]
[1-want,c-w,~i=k,j=k]
[1-want,c-w,~i=k,~j=k,istep1]
[1-want,~c-w]
[1-try,c-t,i=k,j=k]
[1-try2,c-t,i=k,~j=k,inv2]
[1-try2,c-t,~i=k]*
[1-try2,~c-t]
[1-exit2]*
[2-inite]
[2-want2]*
[2-try2]*
[2-exit2]***

Lemma declaration and its usage

```
-- [1-try2,c-t,i=k,~j=k,inv2]
open INV2
   ops i j k : -> Pid .
   -- eq inv1(s,I:Pid,J:Pid) = true .
   -- eq inv2(s,J:Pid) = true .
   -- eq c-try(s,k) = true .
   eq pc(s,k) = wt .
   eq top(queue(s)) = k .
   eq i = k .
   eq (j = k) = false .
-- successor state
   eq s' = try(s,k) .
-- |=
   red inv2(s,j) implies istep1(i,j) .
close
```

declared lemma

used lemma

No need to change already constructed assertions/proof-passages by lemma introduction

```
-- [1-want]*
open INV1
   op s : -> Sys .
   ops i j k : -> Pid .
   eq inv1(s,I:Pid,J:Pid) = true .
-- |=
   red inv1(want(s,k),i,j) .
close
```

implies

```
-- [1-want2]*
open ISTEP2
   ops i j k : -> Pid .
   -- eq inv1(s,I:Pid,J:Pid) = true .
   -- eq inv2(s,I:Pid) = true .
   eq s' = want(s,k) .
-- |=
   red istep1(i) .
close
```


Final Proof Score for QLOCK

[1-init]
[1-want,c-w,i=k]
[1-want,c-w,~i=k,j=k]
[1-want,c-w,~i=k,~j=k,istep1]
[1-want,~c-w]
[1-try,c-t,i=k,j=k]
[1-try2,c-t,i=k,~j=k,inv2]
[1-try2,c-t,~i=k,j=k,inv2]
[1-try2,c-t,~i=k,~j=k]
[1-try,~c-t]
[1-exit2,c-e,i=k]
[1-exit2,c-e,~i=k,j=k]
[1-exit2,c-e,~i=k,~j=k]
[1-exit2,~c-e]

[2-init]
[2-want2,c-w,i=k]
[2-want2,c-w,~i=k,queue(s)=empty]
[2-want2,c-w,~i=k,queue(s)=j,q]
[2-want2,~c-w]
[2-try2,c-t,i=k]
[2-try2,c-t,~i=k]
[2-try2,~c-t]
[2-exit2,c-e,,i=k]
[2-exit2,c-e,~i=k,pc(s,i)=cs,inv1]
[2-exit2,c-e,~i=k,~pc(s,i)=cs]
[2-exit2,~c-e]