Modeling, Specification, and Verification of QLOCK in CafeOBJ

FUTATSUGI, Kokichi
JAIST

Introduction

- Give an overview of modeling, specification, and verification in CafeOBJ.
- Describe an attempt of combining search and inference in proof scores of CafeOBJ by using a QLOCK example.

- This can be seen as an example of combining behavioral specs and rewriting specs.
- Methodology sketched seems to have a potential of becoming a powerful verification technique

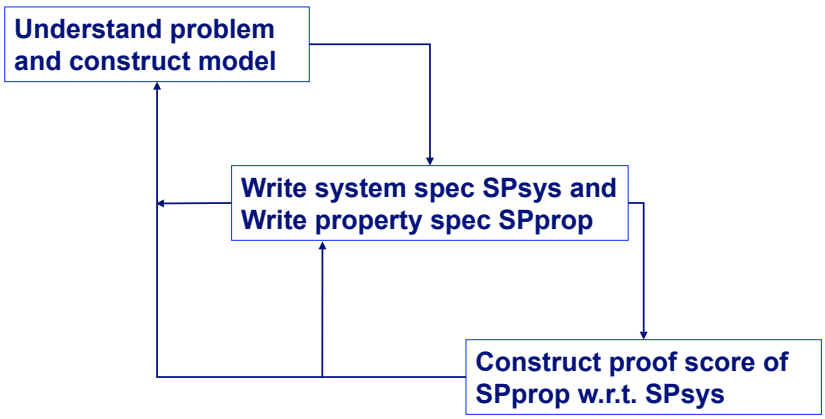
Modeling, Specifying, and Verifying (MSV) in CafeOBJ with Proof Scores

1. By understanding a problem to be modeled/ specified, determine several sorts of objects (entities, data, agents, states) and operations (functions, actions, events) over them for describing the problem

2. Define the meanings/functions of the operations by declaring equations over expressions/terms composed of the operations

3. Write proof scores for properties to be verified

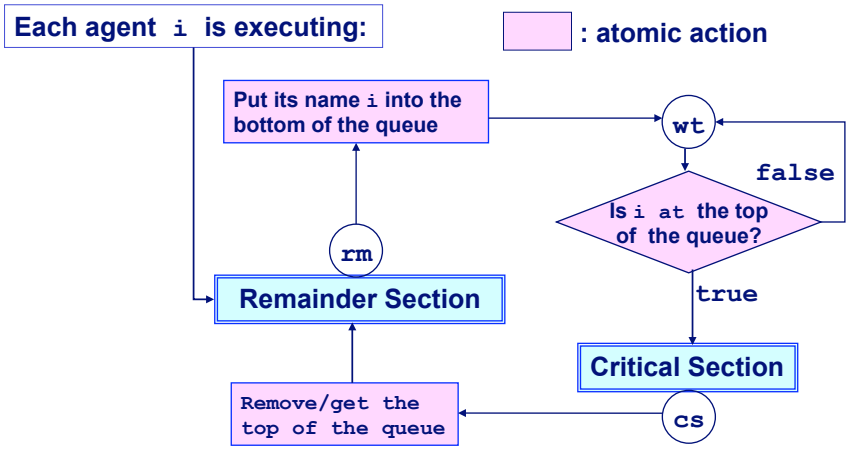
MSV with proof scores in CafeOBJ



An example: mutual exclusion protocol

Assume that many agents (or processes) are competing for a common equipment, but at any moment of time only one agent can use the equipment. That is, the agents are mutually excluded in using the equipment. A protocol (mechanism or algorithm) which can achieve the mutual exclusion is called "mutual exclusion protocol".

QLOCK (locking with queue): a mutual exclusion protocol

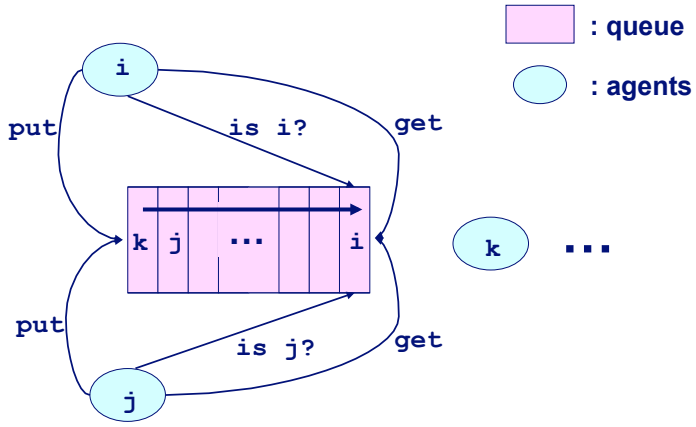


QLOCK: basic assumptions/characteristics

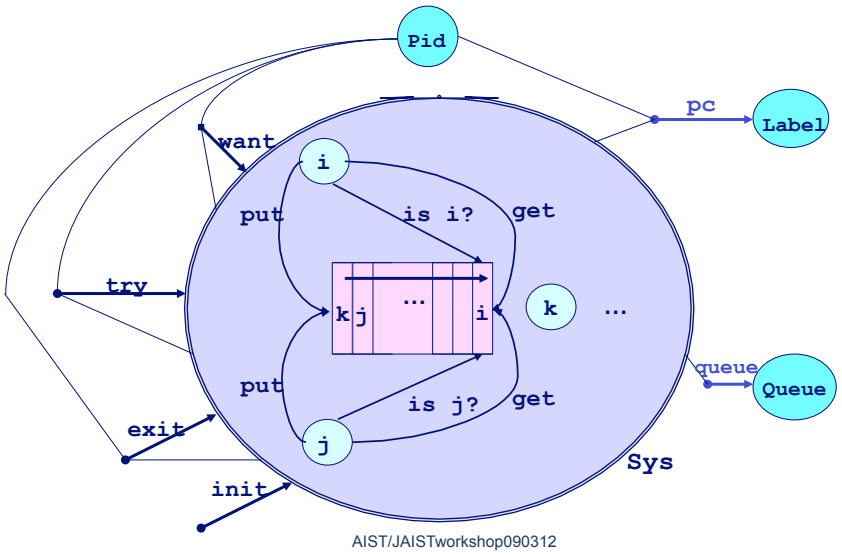
- There is only one queue and all agents/processes share the queue.
- Any basic action on the queue is inseparable (or atomic). That is, when any action is executed on the queue, no other action can be executed until the current action is finished.
- There may be unbounded number of agents.
- In the initial state, every agents are in the remainder section (or at the label `rm`), and the queue is empty.

The property to be shown is that at most one agent is in the critical section (or at the label `cs`) at any moment.

Global (or macro) view of QLOCK



Modeling QLOCK (via Signature Diagram) with OTS (Observational Transition System)



Signature for QLOCKwithOTS

- `Sys` is the sort for representing the state space of the system.
- `Pid` is the sort for the set of agent/process names.
- `Label` is the sort for the set of labels; i.e. `{rm, wt, cs}`.
- `Queue` is the sort for the queues of `Pid`
- `pc` (program counter) is an observer returning a label where each agent resides.
- `queue` is an observer returning the current value of the waiting queue of `Pid`.
- `want` is an action for agent `i` of putting its name/id into the queue.
- `try` is an action for agent `i` of checking whether its name/id is at the top of the queue.
- `exit` is an action for agent `i` of removing/getting its name/id from the top of the queue.

CafeOBJ signature for QLOCKwithOTS

```
-- state space of the system
*[Sys]*

-- visible sorts for observation
[Queue Pid Label]

-- observations
bop pc : Sys Pid -> Label
bop queue : Sys -> Queue

-- actions
bop want : Sys Pid -> Sys
bop try  : Sys Pid -> Sys
bop exit : Sys Pid -> Sys
```

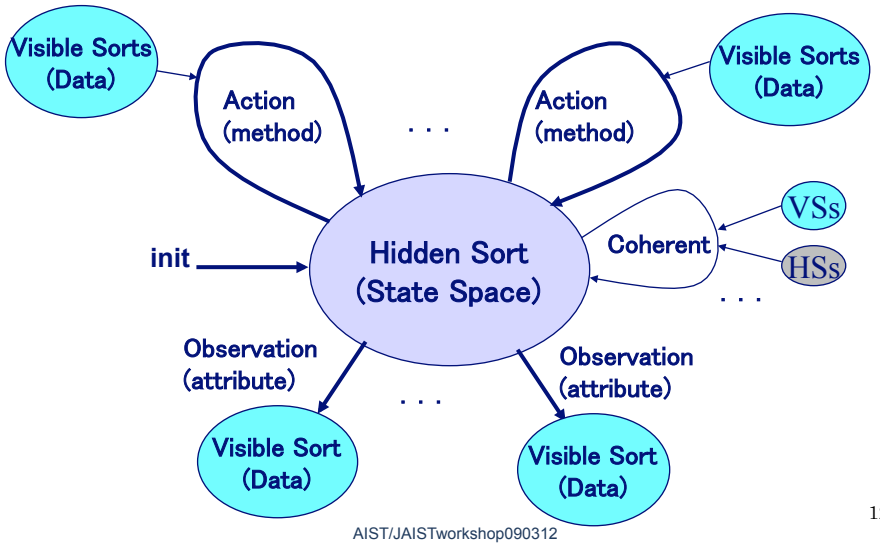
Hiden sort declaration

visible sort declaration

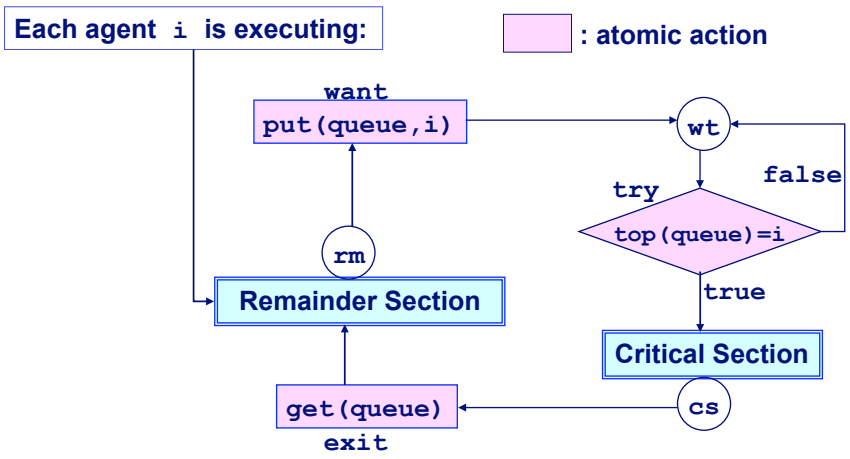
Observation declaration

action declaration

Schematic signature diagram for OTS



QLOCK using operators in the CafeOBJ module QUEUE



CafeOBJ Codes

qlock.mod

(_ =*= _) is congruent for OTS

The binary relation `(S1:Sys =*= S2:Sys)` is defined to be true iff `S1` and `S2` have the same observation values.

OTS style of defining the possible changes of the values of obervations is characterized by the equations of the form:

$o(a(s,d),d')$
$= \ldots o_1(s,d_1)\ldots o_2(s,d_2)\ldots o_n(s,d_n)\ldots$

for appropriate data values of $d,d',d_1,d_2,\ldots,d_n$.

It can be shown that OTS style guarantees that (_ =*= _) is congruent with respect to all actions.

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

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}

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

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

$$\mathrm{INV1} \models \forall s \in \mathrm{R}_{\mathrm{QLOCK}}\, \forall i,j \in \mathrm{Pid}.\ \mathrm{inv1}(s,i,j)$$

CafeOBJ Codes

qlockTrans.mod

mexStarve.mod

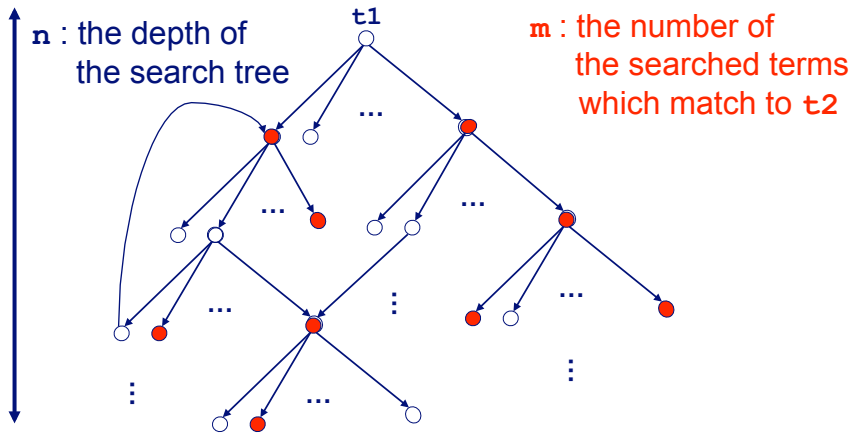
Search command of CafeOBJ

CafeOBJ System has the following built-in predicate:
- **ANY** is any sort (that is, the command is available for any sort)
- **NzNat*** is a built-in sort containing non-zero natural number and the special symbol "*" which stands for infinity

```
pred _=(_,_)=>*_ : Any NzNat* NzNat* Any
```

(t1 =(m,n)=>* t2) returns **true** if **t1** can be translated (or rewritten), via more than 0 times transitions, to some term which matches to **t2**. Otherwise, it returns **false**. Possible transitions/rewritings are searched in breadth first fashion. **n** is upper bound of the depth of the search, and **m** is upper bound of the number of terms which match to **t2**. If either of the depth of the search or the number of the matched terms reaches to the upper bound, the search stops.

t1 =(m,n)=>* t2



`suchThat` condition

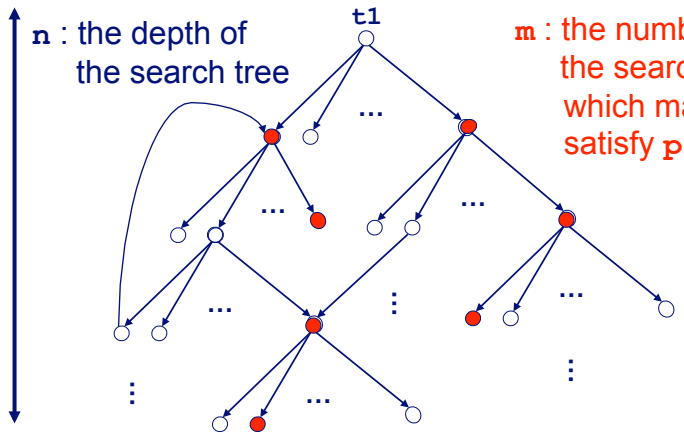
searchCommand.mod

```
t1 =(m,n)=>* t2 suchThat pred1(t2)
```

`pred1(t2)` is a predicate about **`t2`** and can refer to the variables which appear in **`t2`**.
`pred1(t2)` enhances the condition used to determine the term which matches to **t2**.

`t1 =(m,n)=>* t2 suchThat pred1(t2)`

`n` : the depth of the search tree



`m` : the number of the searched terms which match to **`t2`** and satisfy **`pred(t2)`**

CafeOBJ Codes

proofBySearchWithStateRed.mod

stateRedRulePS.mod

`withStateEq` predicate

searchCommand.mod

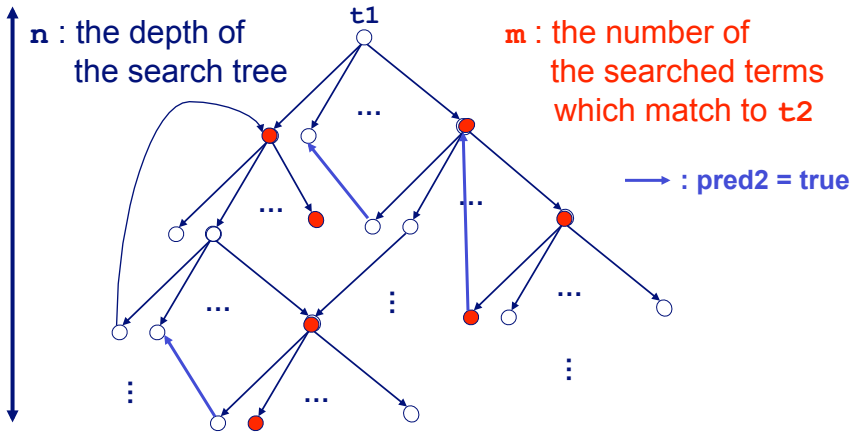
```
t1 =(m,n)=>* t2
   withStateEq pred2(S1:Sort,S2:Sort)
```

`pred2(S1:Sort,S2:Sort)` is a predicate of two arguments with the same (or greater) sort of **`t2`**.
`pred2(S1:Sort,S2:Sort)` is used to determine a newly searched term (a state configuration) is already searched one.
If this **`withStateEq`** predicate is not given, the term identity binary predicate is used for the purpose.

Using both of `suchTant` and `withStateEq` is also possible

```
t1 =(m,n)=>* t2 suchThat pred1(t2)
                withStateEq pred2(S1:Sort,S2:Sort)
```

```
t1 =(m,n)=>* t2
withStateEq pred2(S1:Sort,S2:Sort)
```



CafeOBJ Codes

qlockObEq.mod

proofBySearchWithObEq.mod

Induction scheme induced by the structure of R_{QLOCK}

$$mx(s) =_{def} \forall i,j \in Pid.inv1(s,i,j)$$

$$\{ \quad INV1 \models mx(init),$$
$$INV1 \cup \{mx(s)=true\} \models \forall k . mx(want(s,k)),$$
$$INV1 \cup \{mx(s)=true\} \models \forall k . mx(try(s,k)),$$
$$INV1 \cup \{mx(s)=true\} \models \forall k . mx(exit(s,k)) \ \}$$
$$\text{implies}$$
$$INV1 \models \forall s \in R_{QLOCK}.mx(s)$$

CafeOBJ Codes

inv.mod

proofScore.mod

proofByPS.mod

Tentative Remarks

- OTS style definition of transitions directly corresponds to rewriting style definition.
- Search is sometimes quite effective and easy to use not only in falsification but also in verification.
- OTS style of equations support fast and sound executions/reductions of proof scores. They are sufficiently fast; usually much faster than search.
- Developing proof scores requires and gives deep understanding of problems.
- Proper combination of search and inference (with proof score) can consist transparent and effective verification.

Agitation

Enjoy writing specs and proof scores!