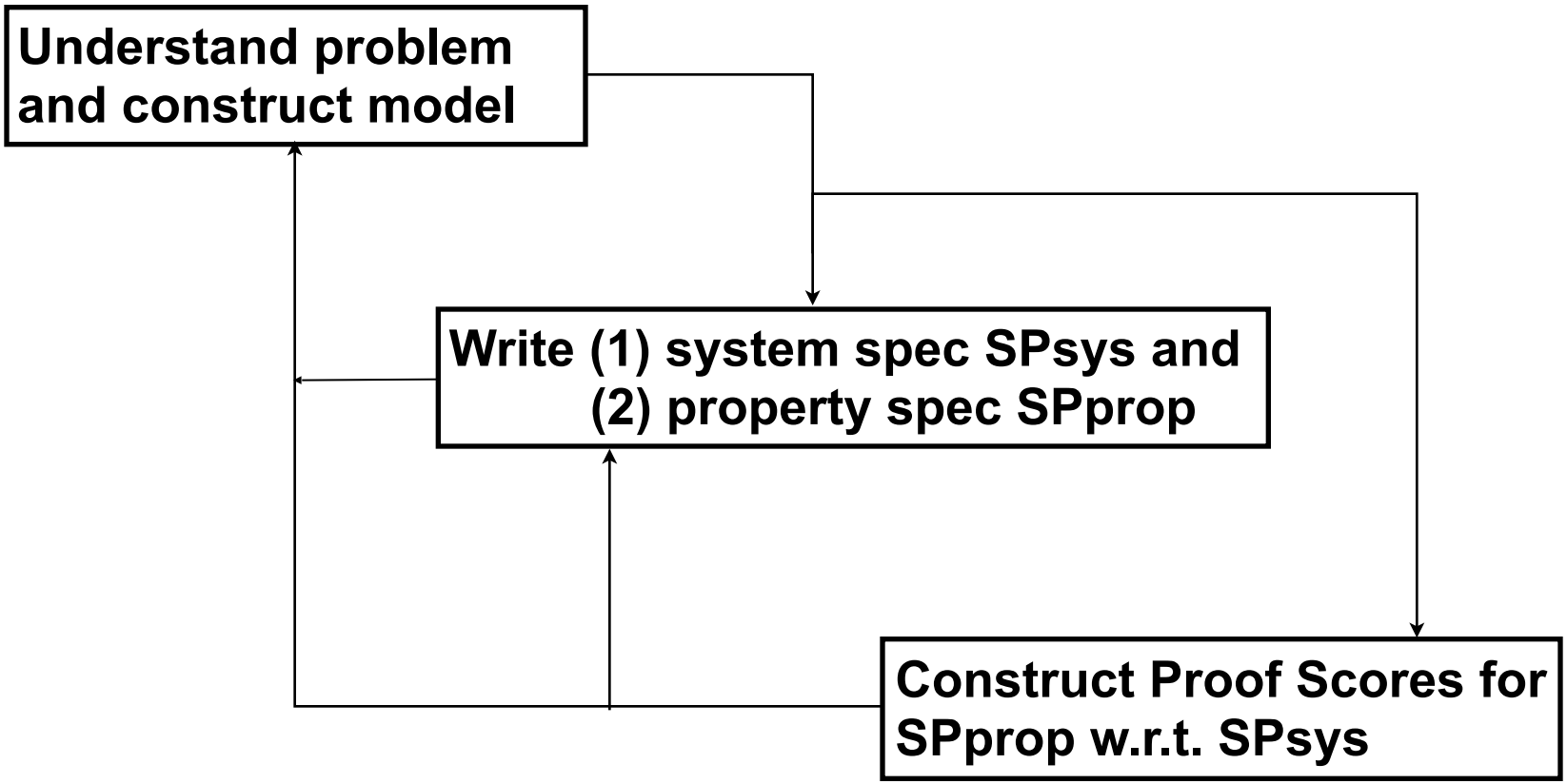
Proof Scores for Invariant Verification with State Patterns (+ Transition Rules + Search Predicates)

Lecture Note 08

Verification with Proof Scores in CafeOBJ

1. Model and describe a system/problem in order-sorted (equational + rewriting) algebraic specification.
2. Construct “proof scores” and verify the specification by reductions/rewritings

Modeling, Specification, and Verification with Proof Scores in CafeOBJ



Transition Systems (State Machines) and Their Invariants (1)

The majority of systems/problems in many fields can be modeled with transition systems and their invariants.

An invariant is a predicate that holds for all reachable states.

Transition Systems (State Machines) and Their Invariants (2)

```
**> System Spec
-- (1) states
[State]
-- (2) transitions over states
trans[<tr>]: <CurrentState> => <NextState> .
    ...
-- transitions are defined with
-- unconditional trans rules
**> Property Spec
-- (3) initial states predicate
op init : State -> Bool .
-- (4) inductive invariant predicate
op inv : State -> Bool .
```

Invariants and Inductive Invariants

inv = p_1 and p_2 … and p_n

inv : inductive invariant

p_1 , p_2, … p_n : invariant

- p_1 , p_2, … p_n are supposed to be as elemental properties of the system/ problem as possible
- a property to be proved is the conjunction of a subset of {p_1 , p_2, … p_n}

Invariant Verification

For proving that the inductive invariant:

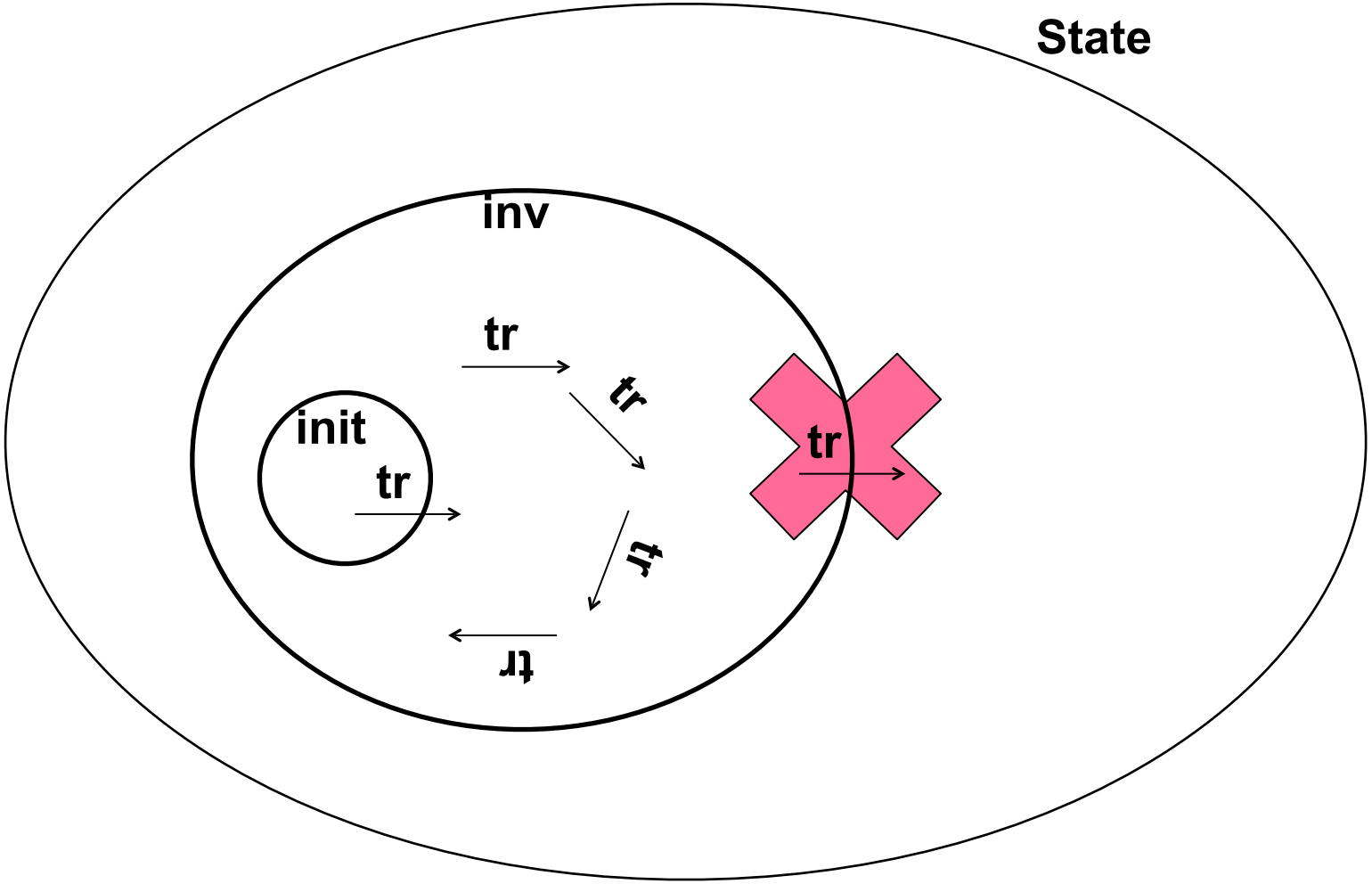
inv = p_1 and p_2 … and p_n

is true for all reachable states, proving the following two conditions are sufficient.

```
** (a) initial state condition
op init-c : State -> Bool .
eq init-c(S:State) = (init(S) implies inv(S)) .
-- (∀S:State)(init-c(S))

** (b) inductive invariant condition
op inv-c : <Transitions> -> Bool .
eq inv-c(T:<Transitions>) =
     (inv(cnt(T)) implies inv(nxt(T))) .
-- (∀T:<Transitions>)(inv-c(T))
```

State, tr, init, inv



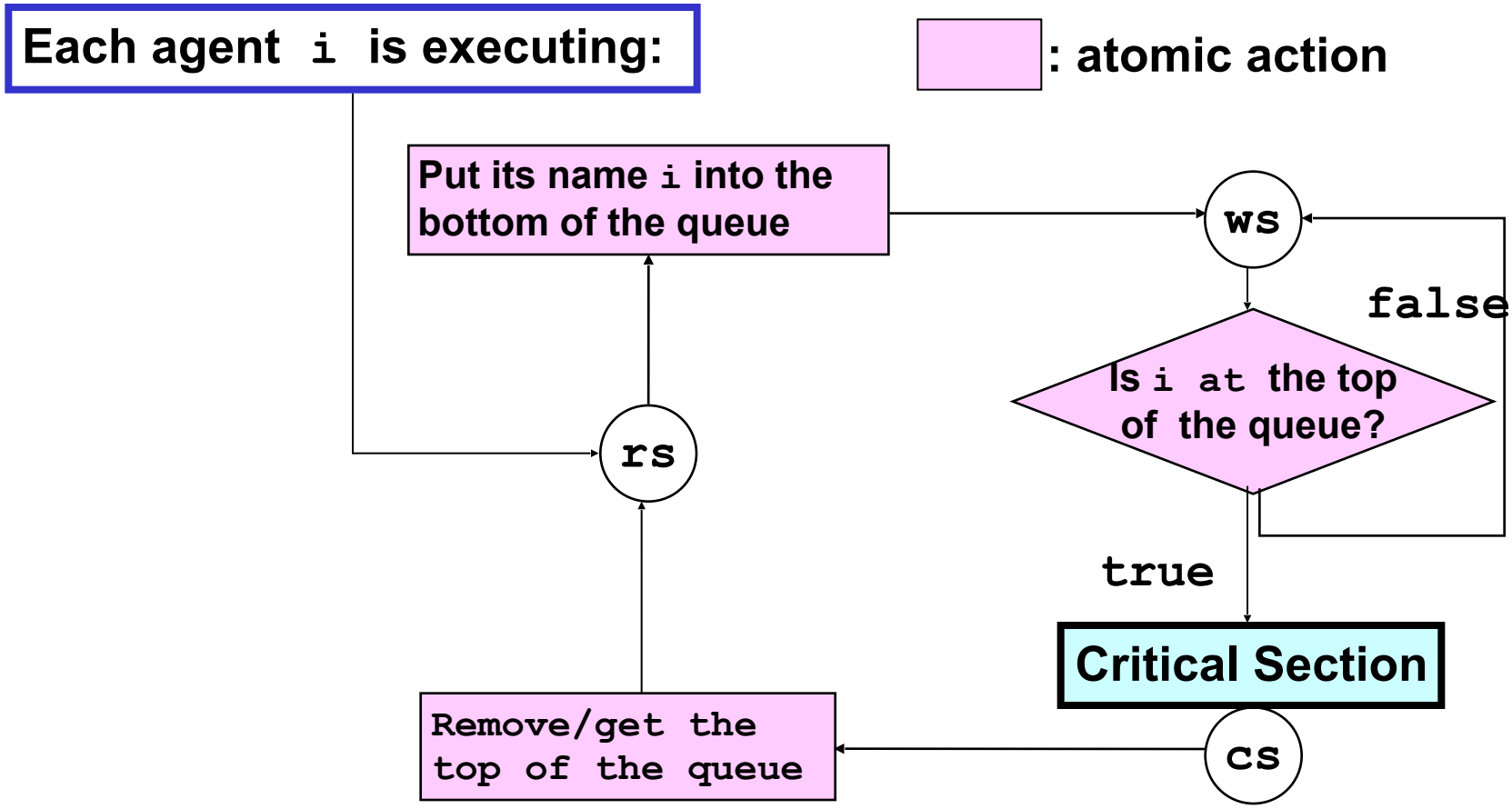
An example: mutual exclusion protocol

Assume that many agents 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".

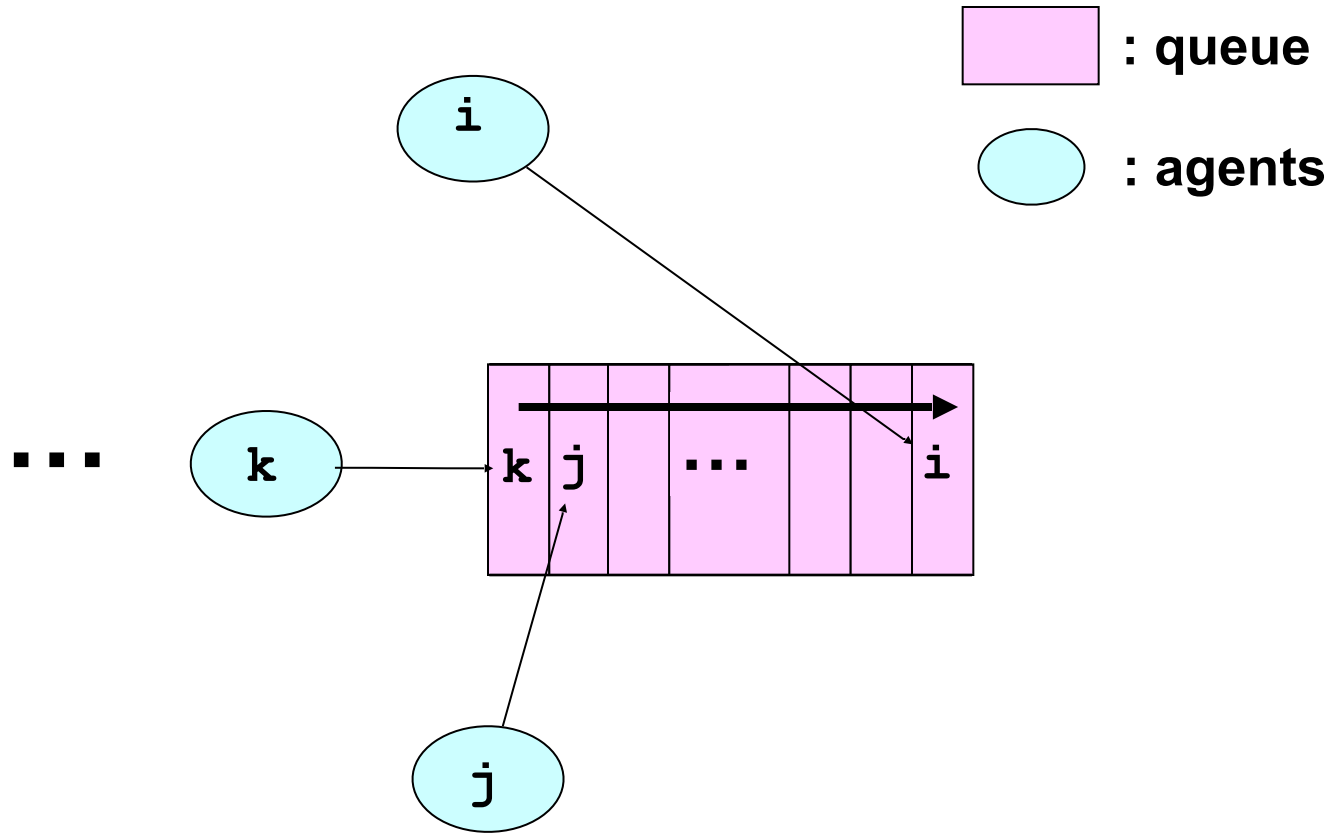
QLOCK: an mutual exclusion with the queue of agent names

(1) If the agent wants to use the common equipment and its name is not in the queue yet, put its name into the bottom of the queue.
(2) If the agent wants to use the common equipment and its name is already in the queue, check if its name is on the top of the queue. If its name is on the top of the queue, start to use the common equipment. If its name is not on the top of the queue, wait until its name is on the top of the queue.
(3) If the agent finishes to use the common equipment, remove its name from the top of the queue.

QLOCK (locking with the queue): a mutual exclusion protocol



Global (or macro) view of QLOCK

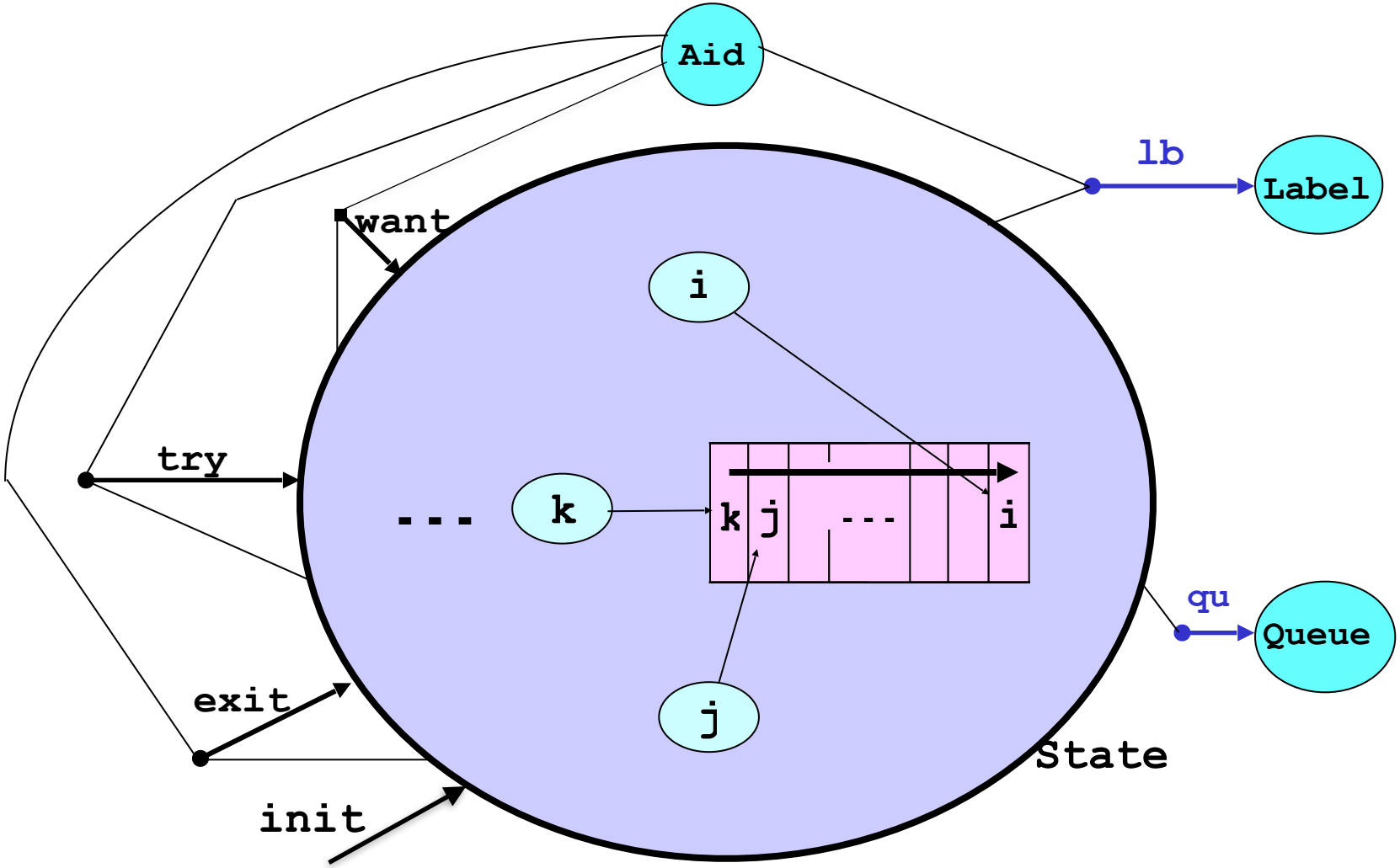


QLOCK: basic assumptions/characteristics

- There is only one queue and all agents 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 `rs`), 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.

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



System Specification of QLOCK

Assumption: Transitions are defined with unconditional transition rules over state terms/expressions

```
-- wt: want transition
trans[wt]:
       (Q:Qu    $ ((lb[A:Aid]: rs) AS:Aobs))
  => ((Q & A) $ ((lb[A     ]: ws) AS)) .
-- ty: try transition
trans[ty]:
     ((A:Aid & Q:Qu) $ ((lb[A]: ws) AS:Aobs))
  => ((A      & Q)    $ ((lb[A]: cs) AS)) .
-- ex: exit transition
trans[ex]:
     ((A1:Aid & Q:Qu) $ ((lb[A2:Aid]: cs) AS:Aobs))
  =>            (Q     $ ((lb[A2    ]: rs) AS)) .
```

Property Specification

Proof Score

Built-in search predicate of CafeOBJ:
(S:State =(*,1)=>+ SS:State suchThat p(S,SS) {t(S,SS})

If the first argument s is given,
(1) it searches all the transitions from s,
(2) check whether the next state s' satisfies the predicate p(s,s'),
(3) prints out value of t(s,s') if p(s,s') is true.

If some next state s' that satisfies the predicate p(s,s') is found, the predicate

```
(s =(*,1)=>+ SS:State suchThat p(s,SS)
 {t(s,SS)})
```

returns 'true', otherwise it returns 'false'.

Predicate for verifying validity of p(s,s') for transition (s->s')

```
valid(S:State,SS:State) =def
 not(S =(*,1)=>+ SS suchThat
      not(p(S,SS) == true)
      {t(S,SS}) .
```

For any state term s,

```
valid(s,SS:State) ->* true
```

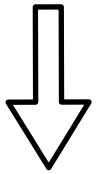
Implies

```
p(s,s') ->* true
```

for any transition (s -> s') from s.

Inductive Invariant Condition

```
** (b) inductive invariant condition
op inv-c : <Transitions> -> Bool .
eq inv-c(T:<Transitions>) =
     (inv(cnt(T)) implies inv(nxt(T))) .
-- (∀T:<Transitions>)(inv-c(T))
```



```
** (b) inductive invariant condition
op inv-c : S:State SS:State -> Bool .
eq inv-c(S:State,SS:State) =
   not(S =(*,1)=>+ SS suchThat
       (not((inv(S) implies inv(SS)) == true))) .
-- (∀S:State)(inv-c(S,SS:State))
```

Generate & Check Method
— for `inv-c` (`init-c` is simpler without condition (1))

Generate a finit set of terms

$$CS = \{sp_1, sp_2, \ldots, sp_n\}$$

(sp_i may be non ground, i.e. may contains variables or **literal variables**)

of sort State systematically such that the following two conditions hold.

(0) For any grand state term s of sort State, there exit sp $\in$ CS such that s is an instance of sp.

(1) **(CS covers LHS)** for the set LHS of all left hand sides of the transition rules.

(2) inv-c(sp_i,SS:State) for i $\in$ {1,2,...,n} are reduced to 'true' by reduction/rewriting (**check**).

(C covers LHS) and Unconditional Trans Rules

$C \subset T(X)$ is defined to cover $LHS \subset T(X)$

=

For any ground instance sgi of any $s \in LHS$, there exits $si \in C$ such that sgi is an instance of si and si is an instance of s.

For showing the correctness of Generate&Check Method, the assumption "all transitions are defined with unconditional trans rules" is used!

Cover Set for QLOCK

```
eq s2 = (empQ $ ((lb[b1]: rs) as)) .          -- wt
eq s3 = (empQ $ ((lb[b1]: ws) as)) .
eq s4 = (empQ $ ((lb[b1]: cs) as)) .

eq s5 = (empQ $ ((lb[b2]: rs) as)) .          ** redundant
eq s6 = (empQ $ ((lb[b2]: ws) as)) .          ** redundant
eq s7 = (empQ $ ((lb[b2]: cs) as)) .          ** redundant

eq s8  = ((b1 & q) $ ((lb[b1]: rs) as)) .     -- wt
eq s9  = ((b1 & q) $ ((lb[b1]: ws) as)) .     -- ty
eq s10 = ((b1 & q) $ ((lb[b1]: cs) as)) .     -- ex

eq s11 = ((b1 & q) $ ((lb[b2]: rs) as)) .     -- wt
eq s12 = ((b1 & q) $ ((lb[b2]: ws) as)) .
eq s13 = ((b1 & q) $ ((lb[b2]: cs) as)) .     -- ex
```

covers

```
WT: (Q:Qu) $ (lb[A:Aid]: rs) S:State)
TY: ((A:Aid & Q:Qu) $ (lb[A]: ws) S:State)
EX: ((A1:Aid & Q:Qu) $ (lb[A2:Aid]: cs) S:State)
```

What has been proved

For any state s_{init} that satisfies the predicate init, any reachable state from s_{init} satisfies inv.