

Combining Inference and Search in Proof Scores (for QLOCK)

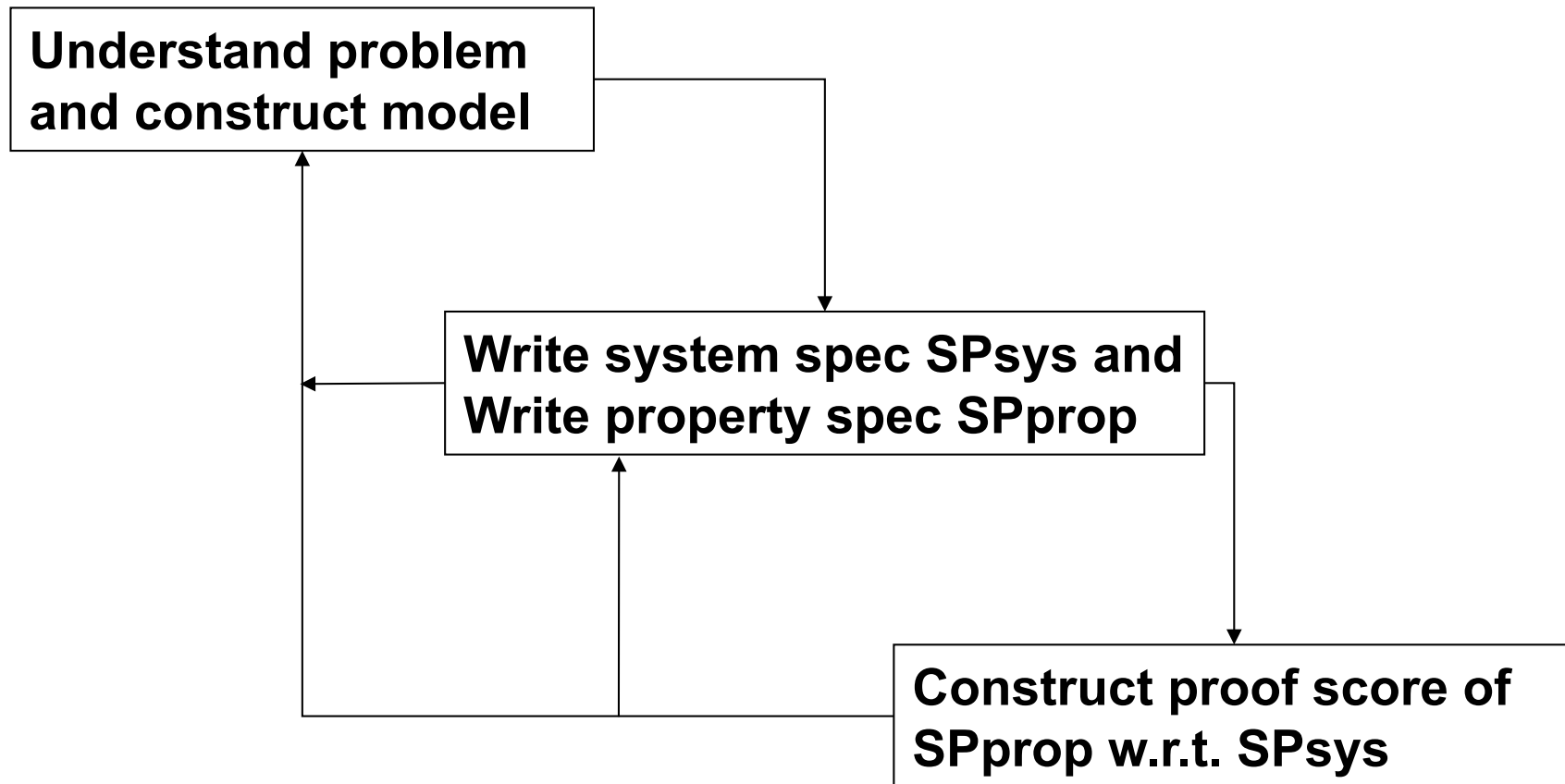
Lecture Note 09a

CafeOBJ Team for JAIST FSSV2010

Topics

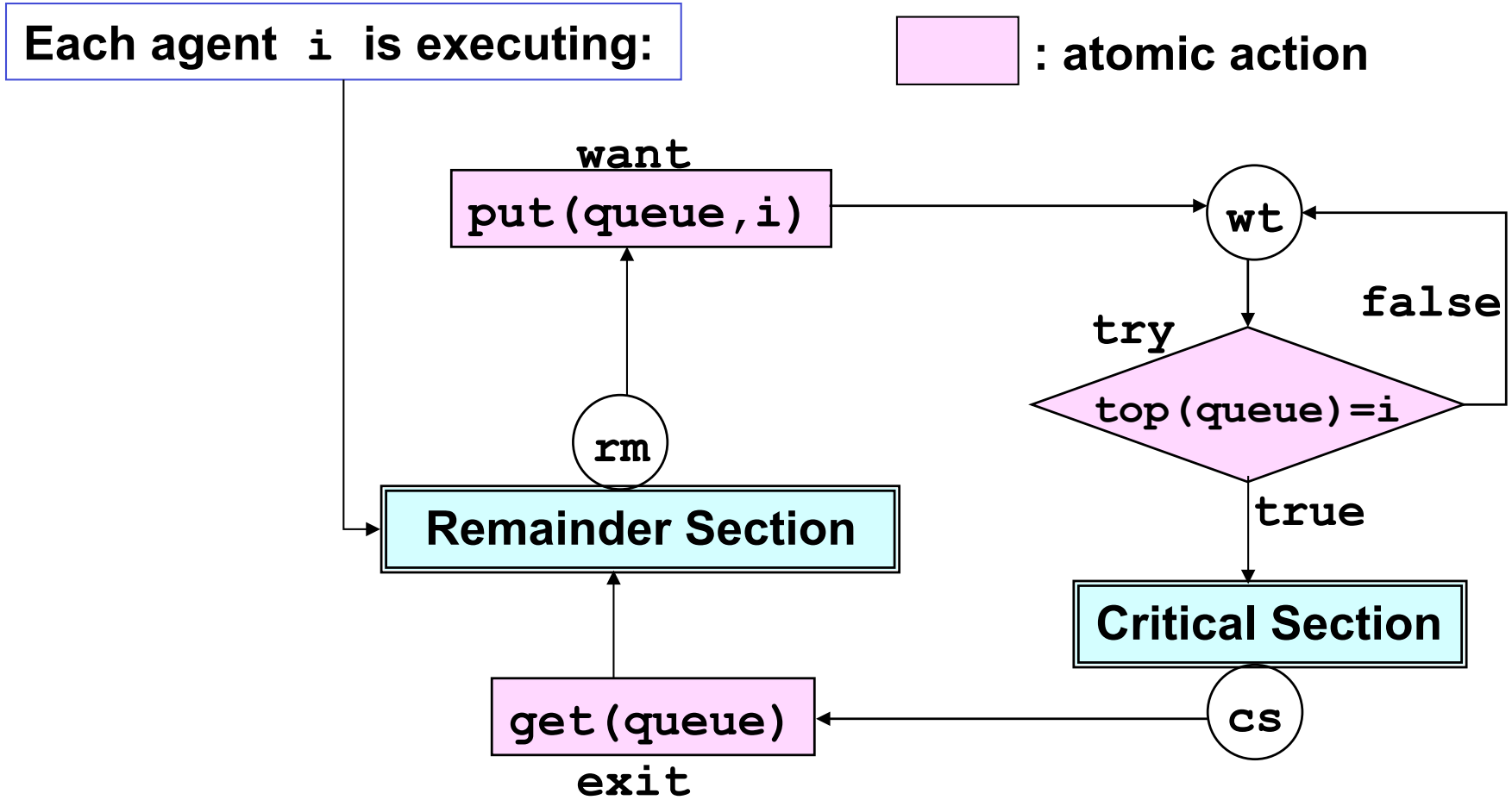
- **Search predicate of CafeOBJ**
- **Falsification with search**
- **Verification with search**
- **Verification with search and inference**

MSV with proof scores in CafeOBJ

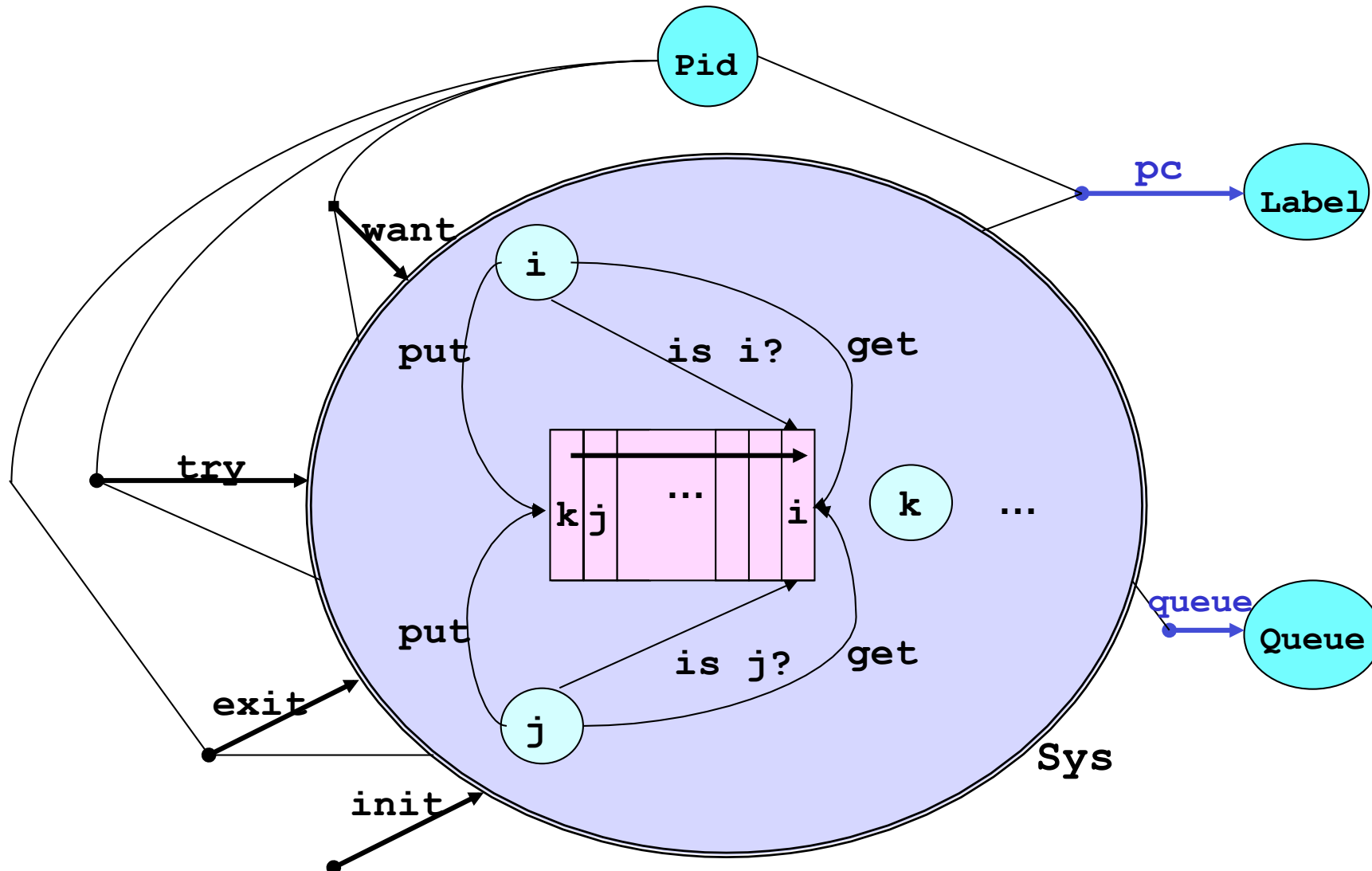


QLOCK using operators in the CafeOBJ module QUEUE

qlock.mod



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



CafeOBJ signature for QLOCKwithOTS

```
-- state space of the system
```

```
[Sys]
```

system sort declaration

```
-- visible sorts for observation
```

```
[Queue Pid Label]
```

visible sort declaration

```
-- observations
```

```
op pc : Sys Pid -> Label
```

```
op queue : Sys -> Queue
```

observation declaration

```
-- any initial state
```

```
init : -> Sys {constr}
```

initial state declaration

```
-- actions
```

```
op want : Sys Pid -> Sys {constr}
```

```
op try : Sys Pid -> Sys {constr}
```

```
op exit : Sys Pid -> Sys {constr}
```

action declaration

Transition system for QLOCK (1)

qlockTrans.mod

```
mod* QLOCKconfig {
  inc(QLOCK)
  [ Config ]
  op <_> : Sys -> Config .
}
-- pre-transition system with an agent/process p
mod* QLOCKpTrans {
  inc(QLOCKconfig)
  op p : -> PidConst .
  var S : Sys .
  -- possible transitions
  ctrans < S > => < want(S,p) > if c-want(S,p) .
  ctrans < S > => < try(S,p) > if c-try(S,p) .
  ctrans < S > => < exit(S,p) > if c-exit(S,p) .
}
```

Transition system for QLOCK (2)

qlockTrans.mod

```
-- transition system with 2 agents i j
mod* QLOCKijTrans {
    inc((QLOCKpTrans * {op p -> i}) +
        (QLOCKpTrans * {op p -> j}))
}

-- transition system with of 3 agents i j k
mod* QLOCKijkTrans {
    inc(QLOCKijTrans +
        (QLOCKpTrans * {op p -> k}))
}
```


Search predicate of CafeOBJ a la Maude's search command

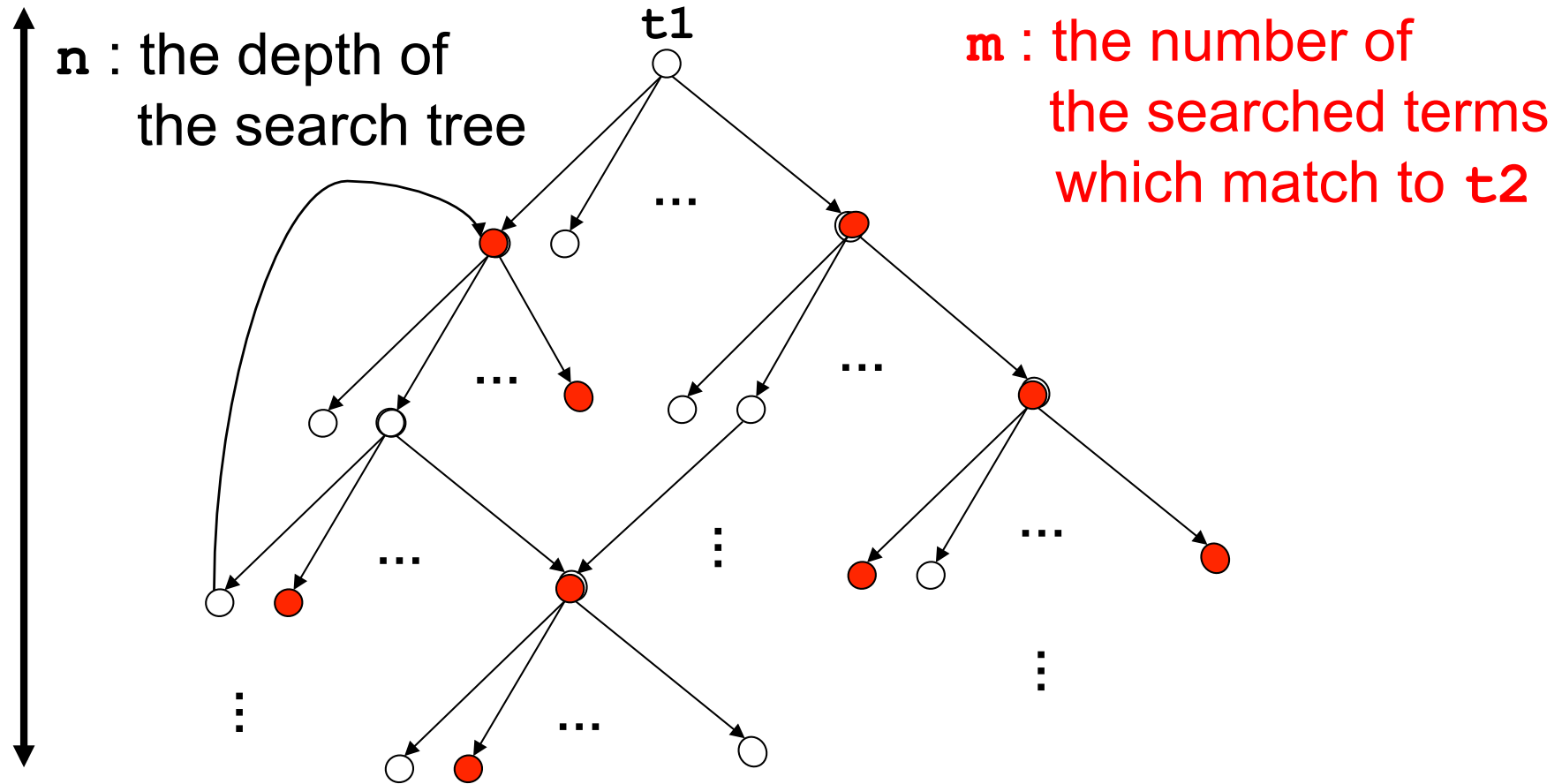
CafeOBJ System has the following built-in predicate:

- 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.

$$t1 = (m, n) \Rightarrow^* t2$$



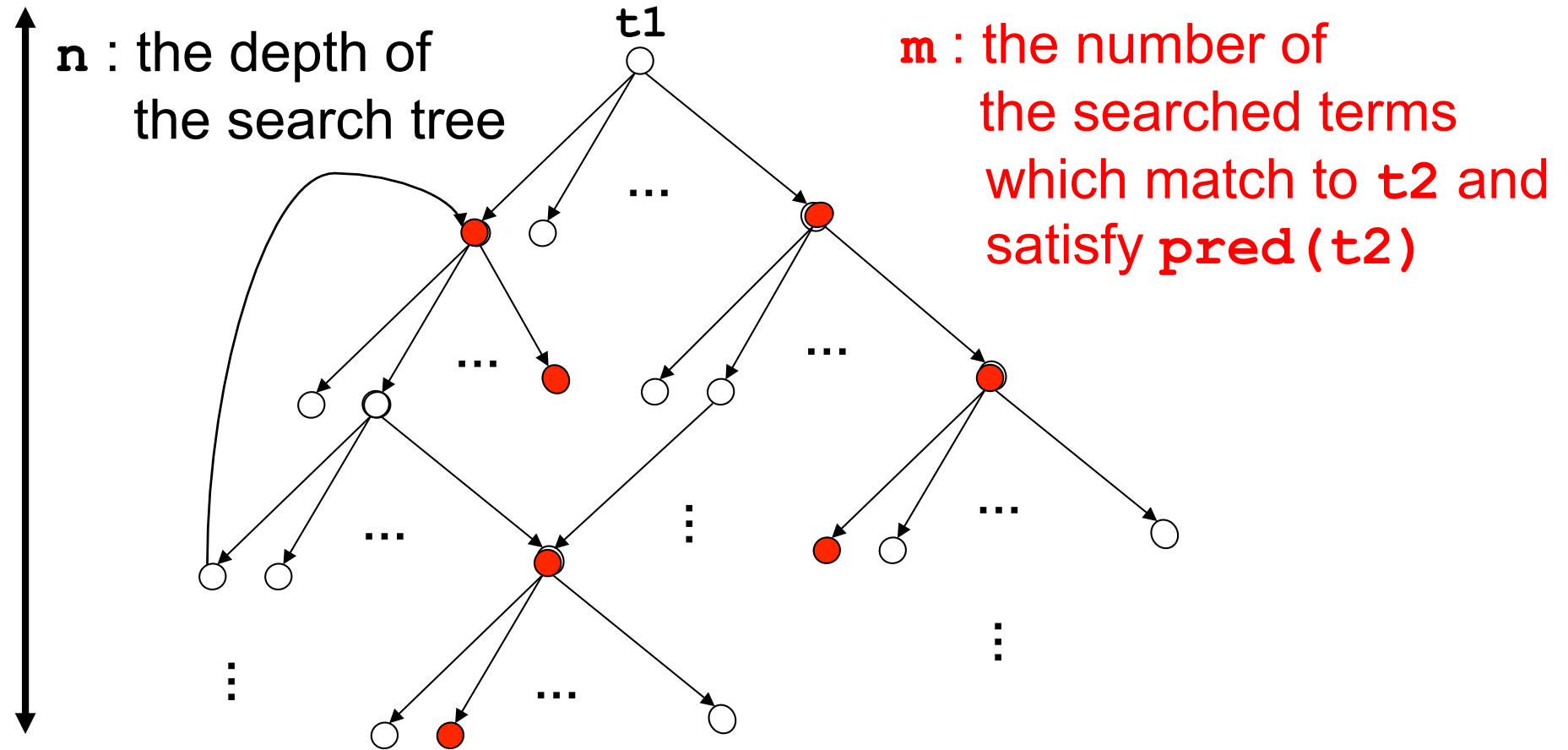
suchThat predicate

$t1 = (m, n) \Rightarrow^* t2 \text{ suchThat } \text{pred1}(t2)$

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**.

$t1 = (m, n) \Rightarrow^* t2$ suchThat $\text{pred1}(t2)$



Falsification by Searching

qlock.mod
qlockTrans.mod
mex.mod
falsificationBySearch.mod

```
Sys(i,j) = {init} U  
  {want(s,a) | s∈Sys, a∈{i,j}} U  
  {try(s,a) | s∈Sys, a∈{i,j}} U  
  {exit(s,a) | s∈Sys, a∈{i,j}}
```

```
red in (QLOCKijTrans + MEX) :  
  < init > =(1,5)=>* < S:Sys >  
  suchThat (not mutualEx(S,i,j)) .
```

This CafeOBJ code searches for a counter example of mutual exclusion property in the state space Sys(i,j) of two agents system up to 5 transitions.

Falsification with two agents system

```
-- reduce in %QLOCKijTrans + MEX :  
-- (( < init > ) = ( 1 , 5 ) =>* (< S > )  
-- suchThat (not mutualEx(S,i,j))):Bool  
-- reached to the specified search depth 5.  
(false):Bool  
(0.000 sec for parse,  
 153655 rewrites(1.670 sec),  
 318255 matches)  
  
-- reduce in %QLOCKijTrans + MEX :  
-- (( < init > ) = ( 1 , 6 ) =>* (< S > )  
-- suchThat (not mutualEx(S,i,j))):Bool  
-- reached to the specified search depth 6.  
(false):Bool  
(0.000 sec for parse,  
 491383 rewrites(5.120 sec),  
 1019619 matches)
```

Falsification with three agents system (1)

```
-- reduce in %QLOCKijkTrans + MEX :  
-- (( < init > ) = ( 1 , 5 ) =>* ( < S > )  
-- suchThat  
--   (not (mutualEx(S,i,j) and  
--         (mutualEx(S,i,k) and  
--         mutualEx(S,j,k))))):Bool  
-- reached to the specified search depth 5.  
(false):Bool  
(0.000 sec for parse,  
 1232435 rewrites(13.080 sec),  
2557809 matches)
```

Falsification with three agents system (2)

```
-- reduce in %QLOCKijkTrans + MEX :
--  ((< init >) = ( 1 , 6 ) =>* (< S >))
--  suchThat
--    (not (mutualEx(S,i,j) and
--           (mutualEx(S,i,k) and
--            mutualEx(S,j,k))))):Bool
-- reached to the specified search depth 6.
(false):Bool
(0.000 sec for parse,
 5526233 rewrites(57.730 sec),
11492616 matches)
```


withStateEq predicate

```
t1 = (m, n) =>* t2
  withStateEq pred2 (V1 : St, V2 : St)
```

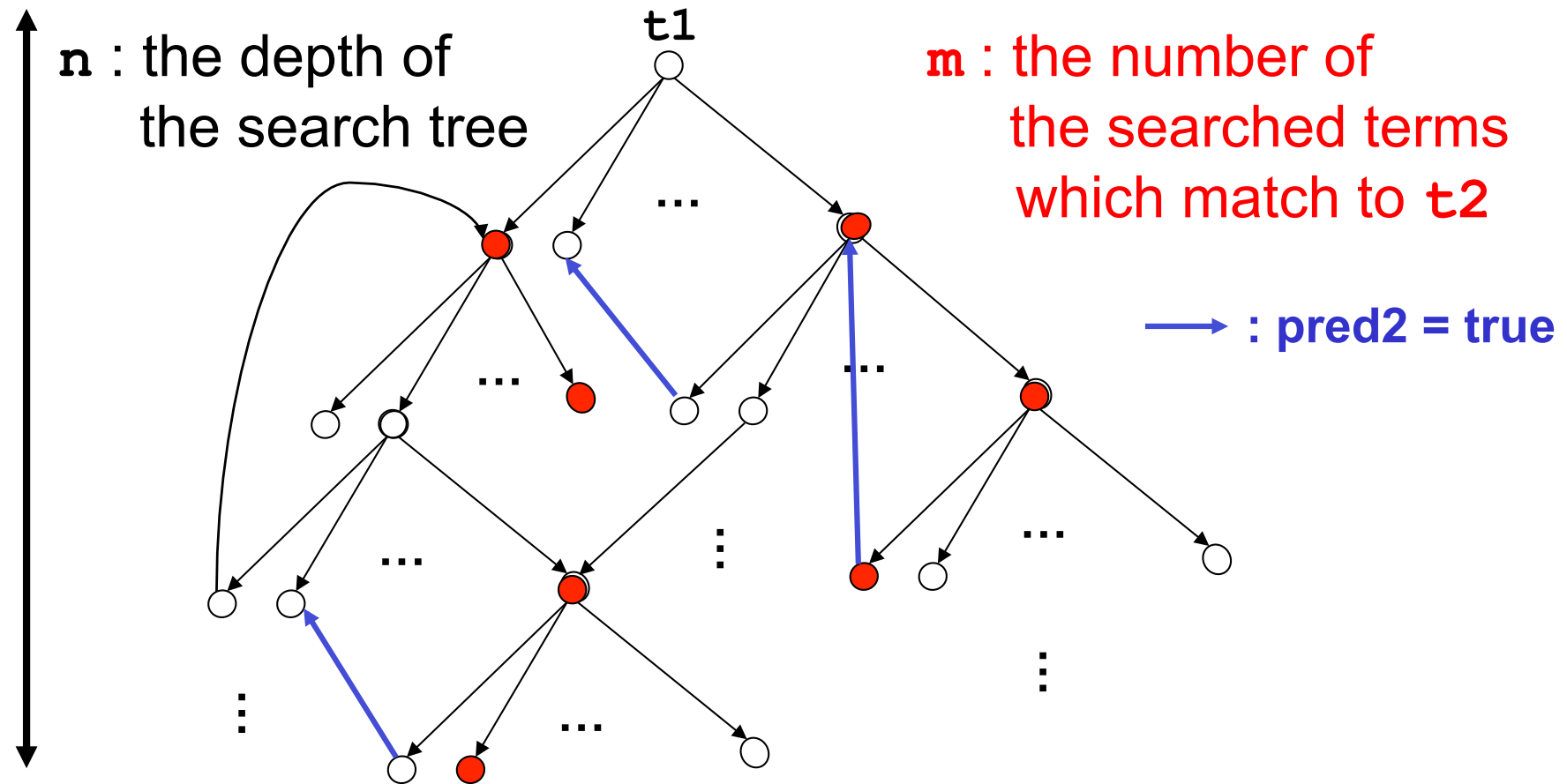
Pred2 (V1 : St, V2 : St) is a binary predicate of two arguments with the same sort **St** of the term **t2**.
Pred2 (V1 : St, V2 : St) is used to determine a newly searched term (a state configuration) is already searched one. If this **withStateEq** predicate is not given, the term identity binary predicate 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)
```

$t1 = (m, n) \Rightarrow^* t2$

with StateEq pred2(V1:St, V2:St)



Verification by Searching with Observational Equivalence

qlockObEq.mod
verificationBySearchWithObEq.mod

```
red in (QLOCKijTrans + QLOCKobEq + MEX) :  
  < init > =(*,*)=>* < S:Sys >  
  suchThat (not mutualEx(S,i,j))  
  withStateEq (C1:Config =ob= C2:Config) .
```

This CafeOBJ code searches for a counter example of mutual exclusion property in the whole state space $Sys(i,j)$ of two agents system. If this returns **false**, the two agents system is verified to have the mutual exclusion property.

Simulation of any number of agents systems by the two agents system (1)

Proof scores of

**simOfQLOCKbyQLOCKijPS.mod
csQtopPS.mod**

verify the following

let i and j be any two distinctive process identifiers, and let $\text{Sys}(i,j)$ be the state space of QLOCK with only the two processes i and j , then for any reachable state $s:\text{Sys}$ of QLOCK, there exists a reachable state $t:\text{Sys}(i,j)$ such that $((\text{pc}(s,i) = \text{pc}(t,i)) \text{ and } (\text{pc}(s,j) = \text{pc}(t,j)))$ and $(\text{p}_{ij}(\text{queue}(s)) = \text{queue}(t))$ where $\text{p}_{ij}(\text{nil}) = \text{nil}$
 $\text{p}_{ij}(Q:\text{Queue},l:\text{Pid}) = \text{p}_{ij}(Q),l$ if $(l=i \text{ or } l=j)$
 $= \text{p}_{ij}(Q)$ if $\text{not}(l=i \text{ or } l=j)$.

Simulation of any number of agents systems by the two agents system (2)

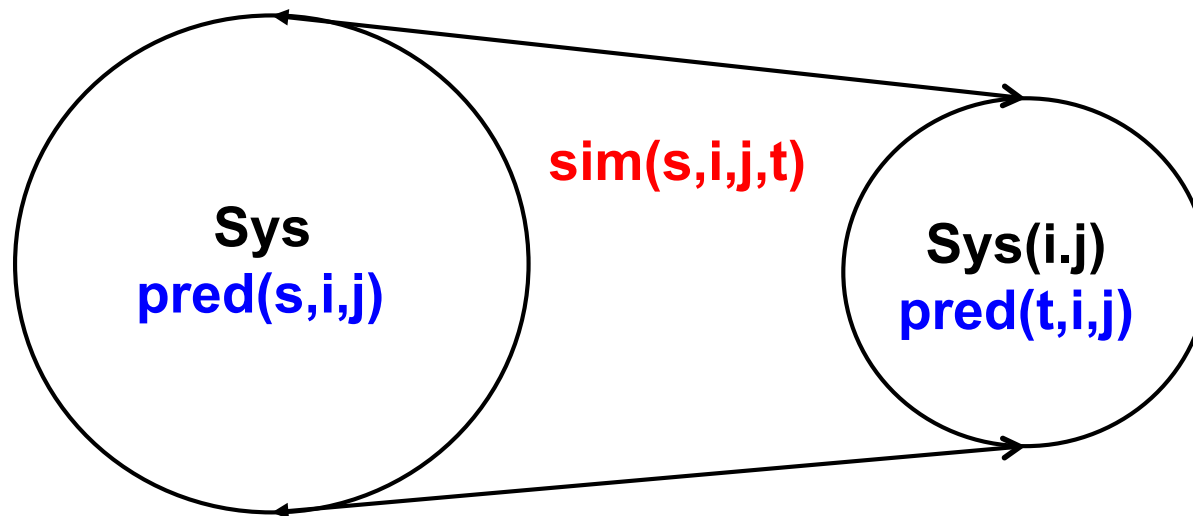
Proposition-A

for any reachable state $s:\text{Sys}$ of QLOCK, and
for any $i:\text{Pid}$ and $j:\text{Pid}$ such that $\text{not}(i=j)$,
there exists a reachable state $t:\text{Sys}(i,j)$ such that
 $\text{sim}(s,i,j,t)$
where
 $\text{sim}(s,i,j,t) = (\text{pc}(s,i) = \text{pc}(t,i))$ and $(\text{pc}(s,j) = \text{pc}(t,j))$
and $(\text{p}_{ij}(\text{queue}(s)) = \text{queue}(t))$

Simulation of any number of agents systems by the two agents system (2)

Proposition-B

for any predicate $\text{pred}(S:\text{Sys}, P:\text{Pid}, Q:\text{Pid})$
over the specification QLOCK,
for any $s:\text{Sys}$, $i:\text{Pid}$, $j:\text{Pid}$, $t:\text{Sys}(i,j)$ such that $\text{not}(i=j)$,
($\text{sim}(s,i,j,t)$ and $\text{pred}(t,i,j)$) implies $\text{pred}(s,i,j)$



Simulation of any number of agents systems by the two agents system (3)

Propotion-A implies that for any predicate:
 $\text{pred}(S:\text{Sys}, P:\text{Pid}, Q:\text{Pid})$
over the specification QLOCK which satisfies
Proposition-B,

$(\text{for-all } t:\text{Sys}(i,j)).\text{pred}(t,i,j)$
implies
 $(\text{for-all } s:\text{Sys}).\text{pred}(s,i,j)$

or

$(\text{no existance of } t:\text{Sys}(i,j) \text{ such that } (\text{not } \text{pred}(t,i,j)))$
implies
 $(\text{for any } s:\text{Sys}).\text{pred}(s,i,j)$

Examples of $\text{pred}(s:\text{Sys}, i:\text{Pid}, j:\text{Pid})$ which satisfies Proposition-B

```
pred mutualEx : Sys Pid Pid
var S : Sys . vars I J : Pid .
eq mutualEx(S, I, J)
  = ((pc(S, I) = cs) and (pc(S, J) = cs))
    implies (I = J) .
```

```
pred twoWt : Sys Pid Pid
var S : Sys . vars I J : Pid .
eq twoWt(S, I, J)
  = ((pc(S, I) = wt) and (pc(S, J) = wt))
    implies ((I in queue(S)) and
             (J in queue(S))) .
```


CafeOBJ proof scores for verifying the simulation

```
simOfQLOCKbyQLOCKijPS.mod  
csQtopPS.mod
```

These proof scores are almost same amount to the original proof score for verifying mutual exclusion. However, once the simulation is verified, many properties other than mutual exclusion can be verified by searching over the two-agents systems.

Remarks

- **OTS style definition of transition directly corresponds to rewriting transition.**
- **Search is sometimes quite effective and easy to use not only in falsification but also in verification. Especially for small values of parameters.**
- **Proper combination of search and inference (with proof score) can constitute transparent and effective verification.**