Combining Inference and Search in Proof Scores (for QLOCK)

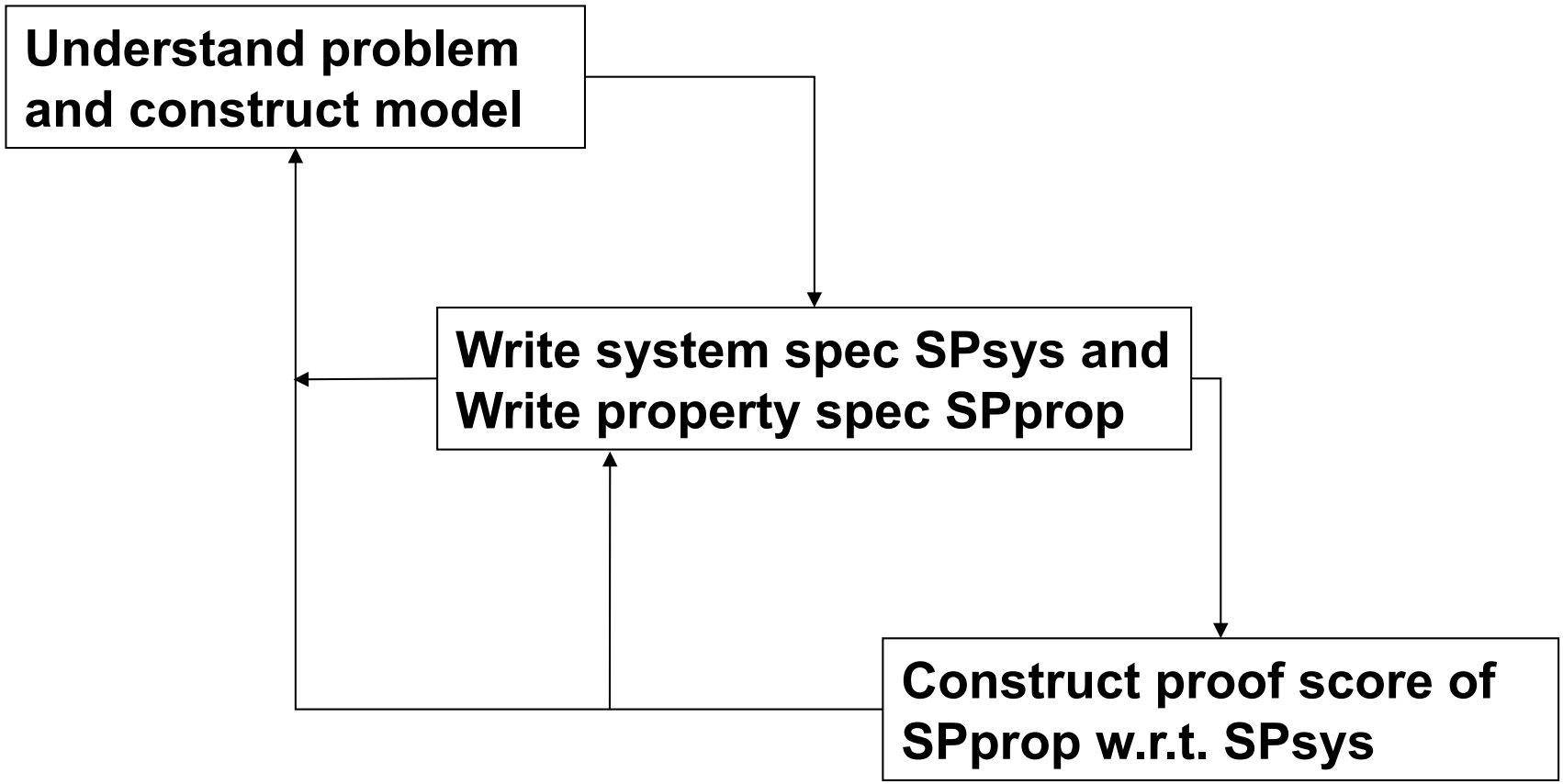
Lecture Note 11

Formal Methods (i613-0912)

Topics

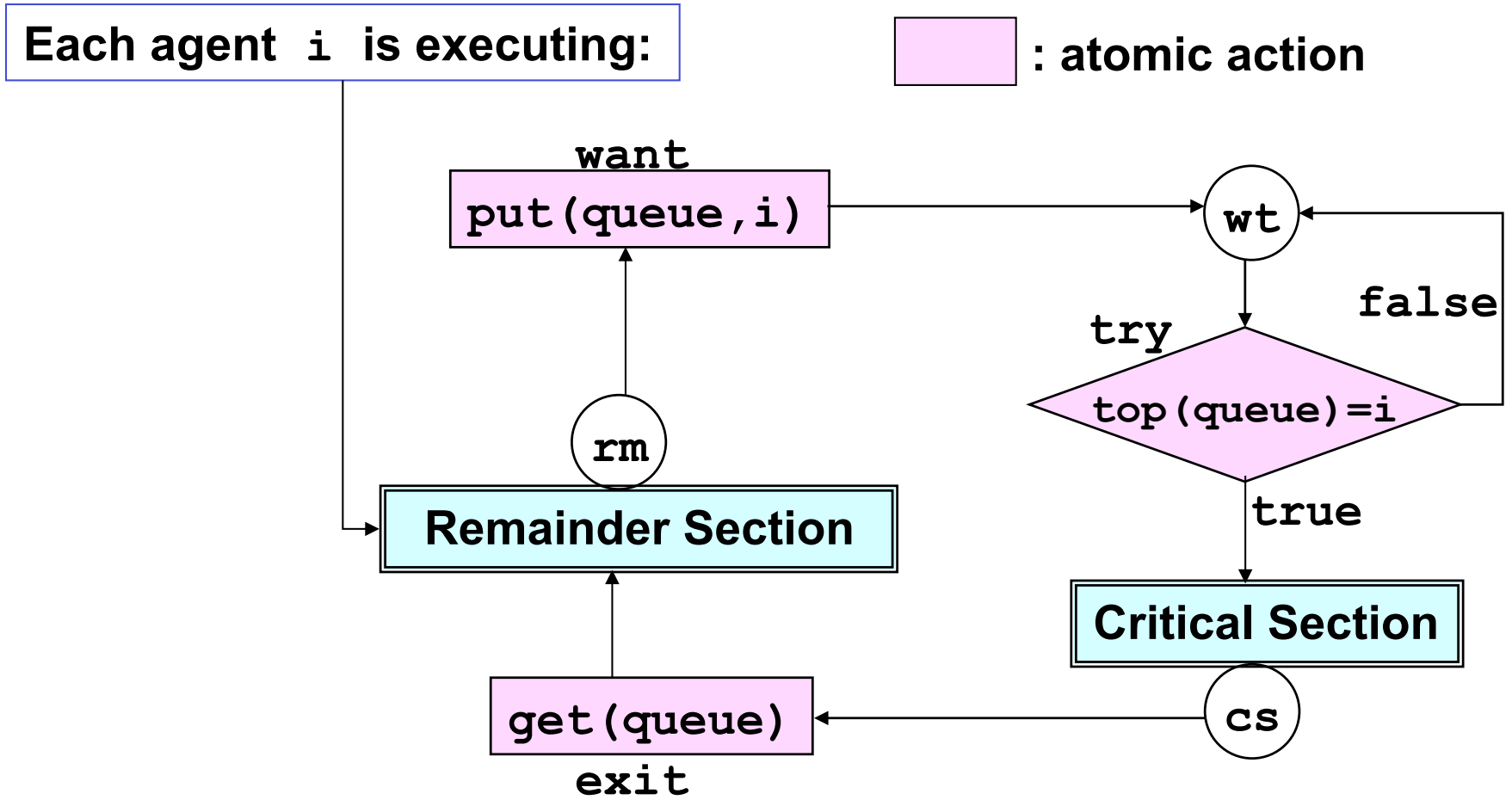
- **Search predicate of CafeOBJ**
- **Falsification with search**
- **Verification with search**
- **Verification with search and inference**

MSV with proof scores in CafeOBJ

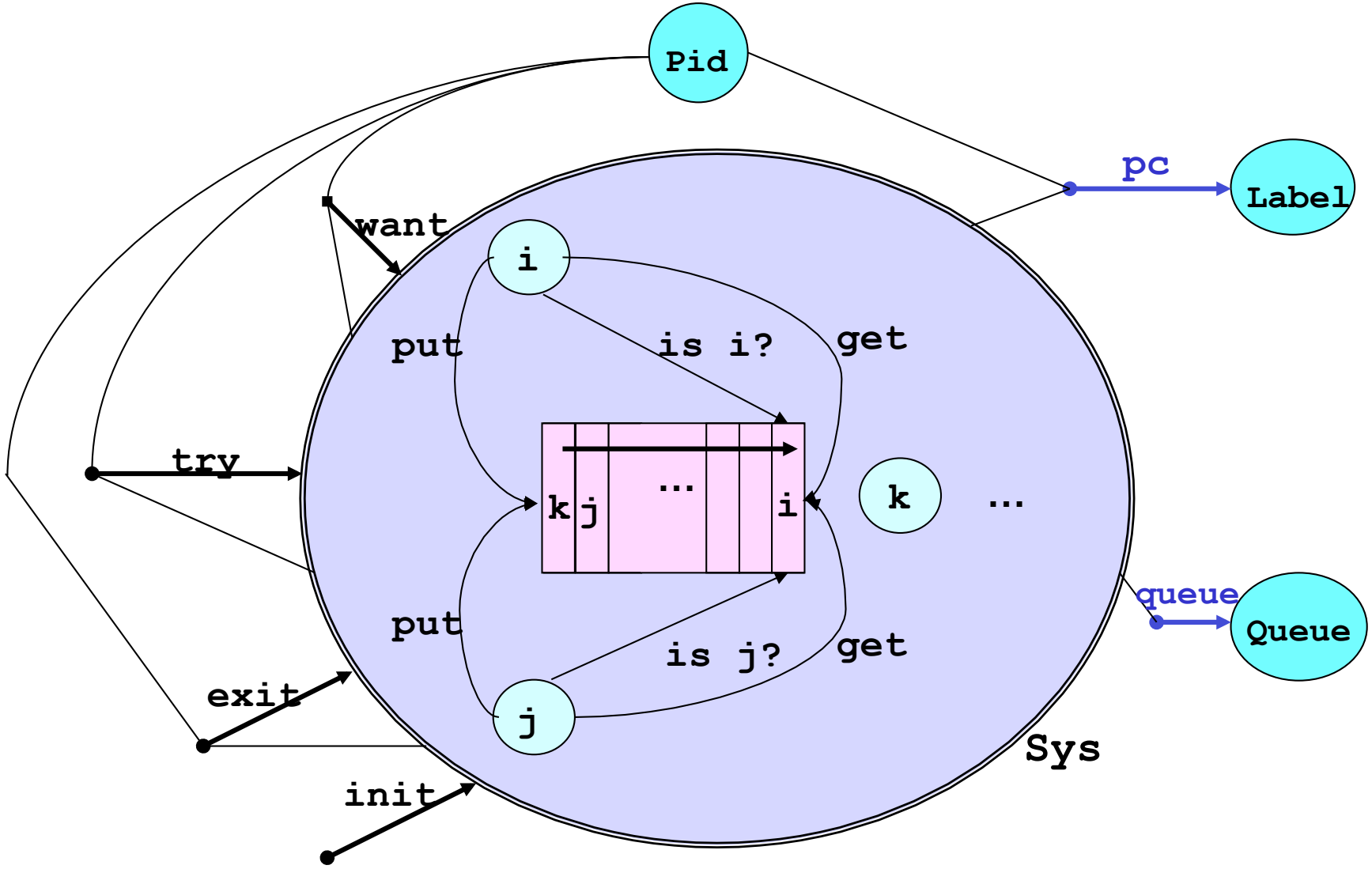


QLOCK using operators in the CafeOBJ module QUEUE

qlock.mod



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



CafeOBJ signature for QLOCKwithOTS

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

-- visible sorts for observation
[Queue Pid Label]

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

-- any initial state
init : -> Sys (constr)
-- actions
bop want : Sys Pid -> Sys (constr)
bop try  : Sys Pid -> Sys (constr)
bop exit : Sys Pid -> Sys (constr)
```

system sort declaration

visible sort declaration

observation declaration

initial state declaration

action declaration

Transition system for QLOCK (1)

qlockTrans.mod

```
mod* QLOCKconfig {
  inc(QLOCK)
  [ Config ]
  op <_> : Sys -> Config .
}
-- pre-transiton 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}))
}
```

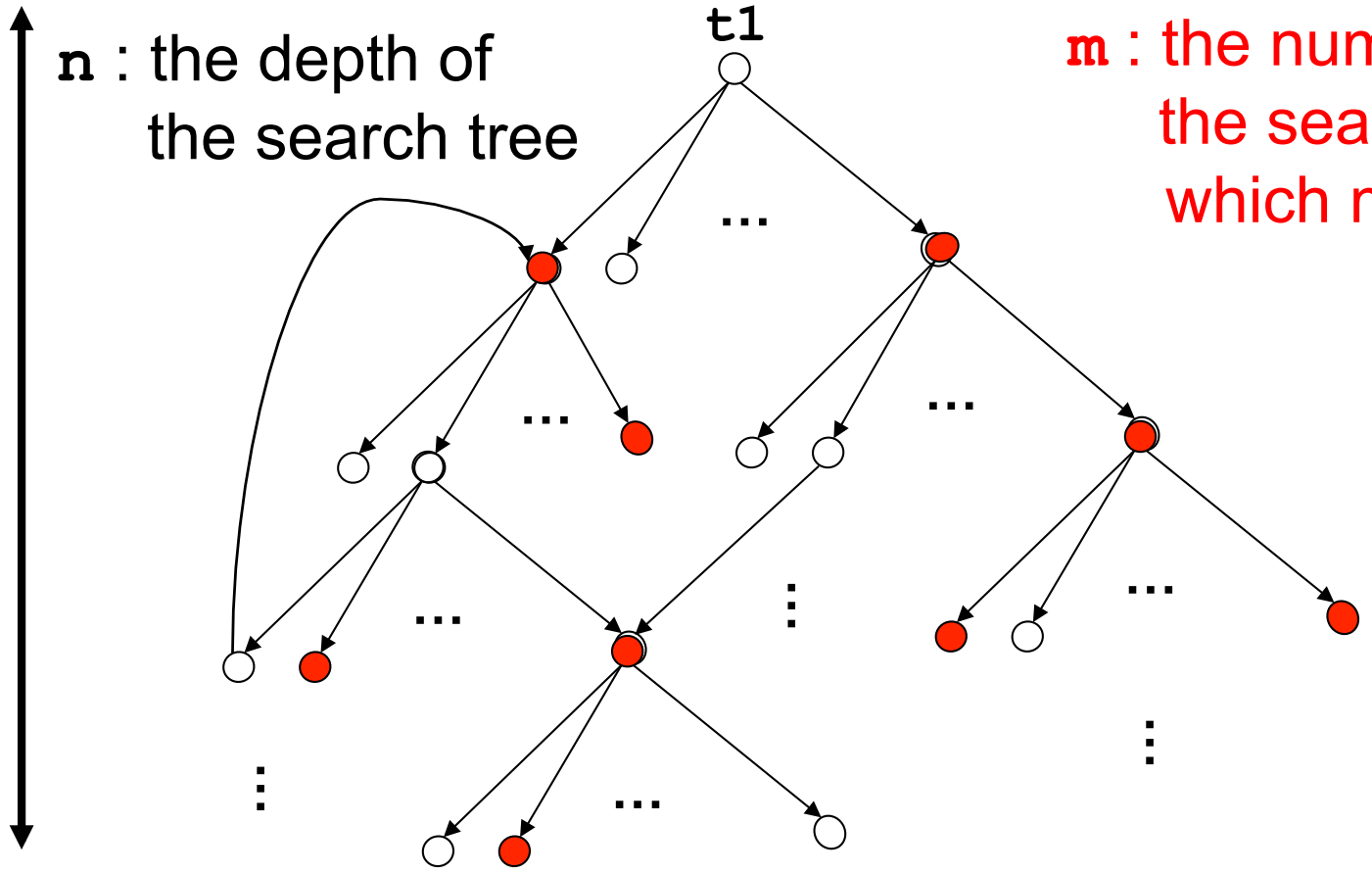
Search predicate of CafeOBJ a la OBJ/Maude's search command

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)=>* t2


n : the depth of
the search tree
t1
m : the number of
the searched terms
which match to t2

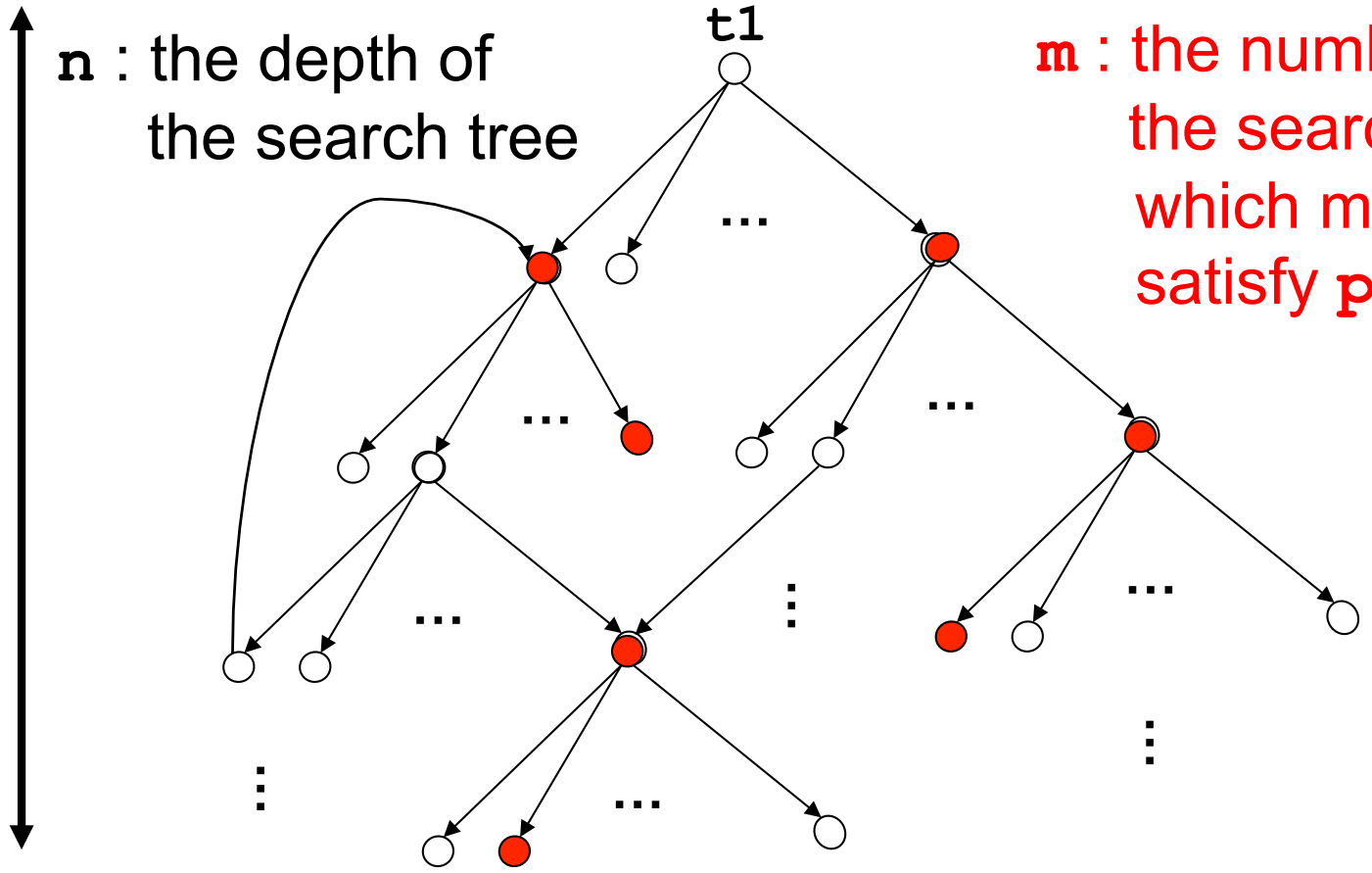
`suchThat` predicate

```
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)



Falsification by Searching

qlock.mod
qlockTrans.mod
mex.mod
falisificationBySearch.mod

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

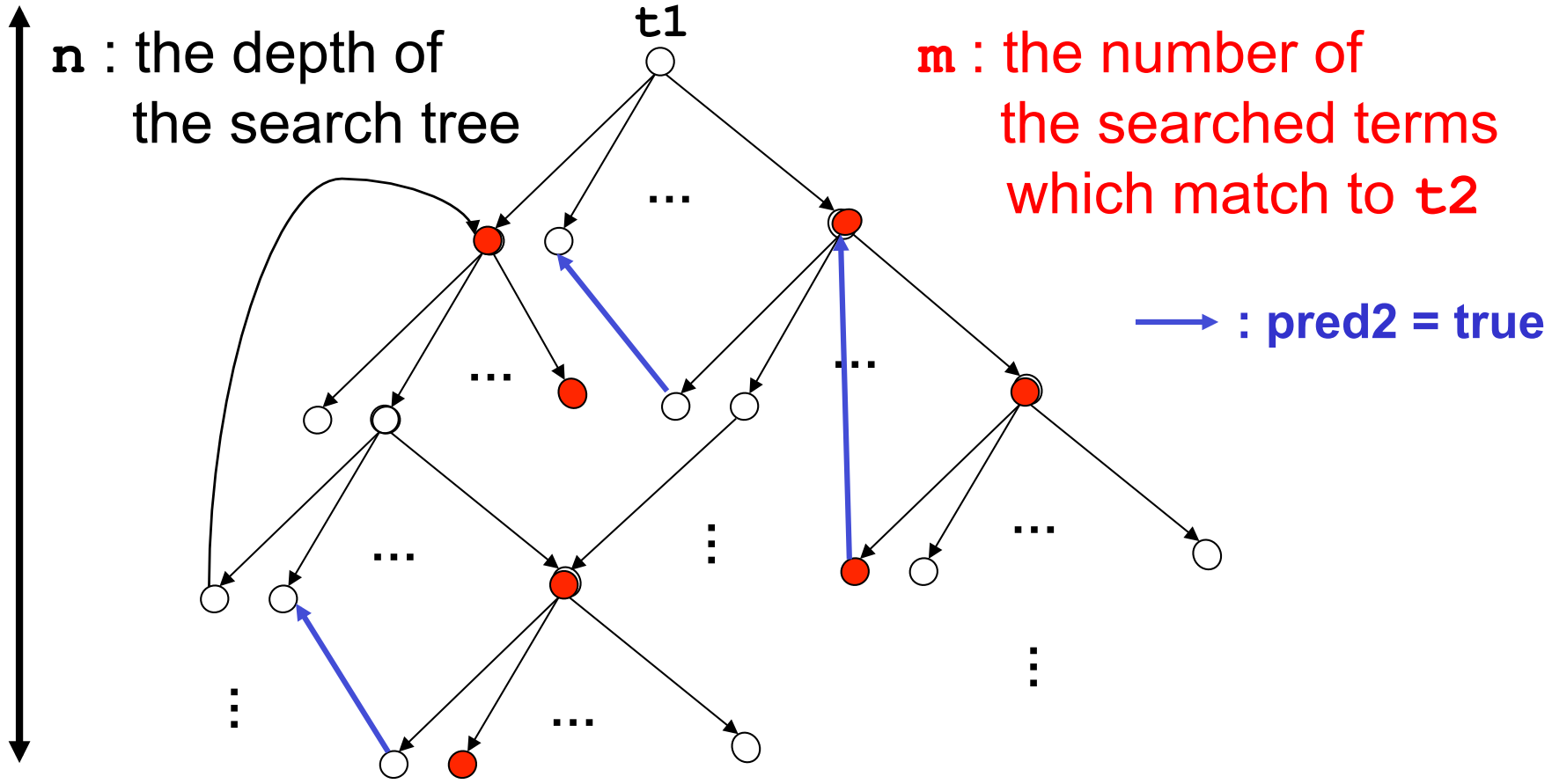
```
t1 =(m,n)=>* t2
   withStateEq pred2(V1:St,V2:St)
```

`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)=>* t2
withStateEq 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)

If all behaviors of any two agents with the system of any number of agents can be simulated by the system with only two agents, all the properties checked by searching all reachable states of the two-agents system are verified to hold for the system of any number of agents.

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

let
  (ops i j : -> PidConst .)
and let Sys(i,j) be a subsort of Sys which are composed of terms of sort Sys which contain only i and j.

for any two distinct process identifiers i and j, and
for any reachable state s:Sys of QLOCK,
there exists a reachable state t:Sys(i,j) of QLOCK such that
  ((pc(s,i) = pc(t,i)) and (pc(s,j) = pc(t,j))) and
   (pij(queue(s)) = queue(t))
   where pij(nil) = nil
            pij(Q:Queue,I:Pid) = pij(Q),I if (I=i or I=j)
                                = pij(Q)   if not(I=i or I=j)

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.

Exercise

Complete the proof score in the file:
csQtopPSforExcs.mod.