A Collaborative Use of CafeOBJ and Maude

Lecture Note 09b

CafeOBJ Team for JAIST-FSSV2010

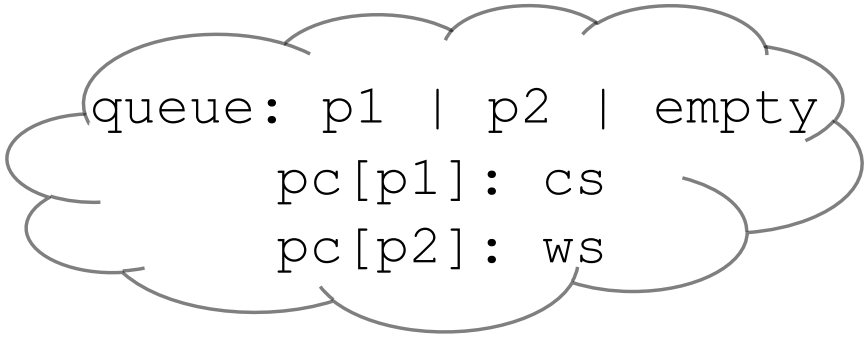
Topics

- Another way of specifying OTSs in both CafeOBJ & Maude for model checking.
  Qlock is used as an example.
- Another combination of inference & search.
  NSPK is used as an example.

Another Way of Specifying OTSs (1)

- States as collections of observable values.

  E.g., a state of Qlock is depicted as



Note that for each process i, at most one value observed by "pc", i.e., pc[i] : l, appears in a state.

Qlock

Loop:
  Remainder Section
rs: enq(*queue*,*i*);
ws: **repeat until** top(*queue*) = *i*;
  Critical Section
cs: deq(*queue*);

Another Way of Specifying OTSs (2)

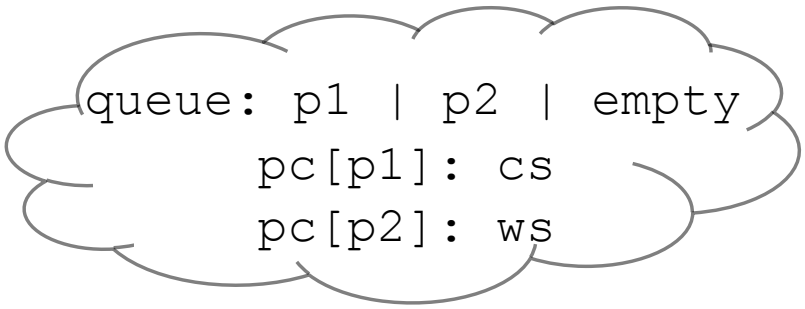
- Configuration

```
[Obs < Sys]
op void : -> Sys {constr}
op _ _ : Sys Sys -> Sys {constr assoc comm id: void}
```

- Operators for observable values

```
op (pc[_]:_) : Pid Label -> Obs {constr}
op queue:_ : Queue -> Obs {constr}
```

```
(pc[p1]: cs) (pc[p2]: ws) (queue: p1 | p2 | empty)
```



Loop:
  Remainder Section
rs: enq(*queue*,*i*);
ws: **repeat until** top(*queue*) = *i*;
  Critical Section
cs: deq(*queue*);

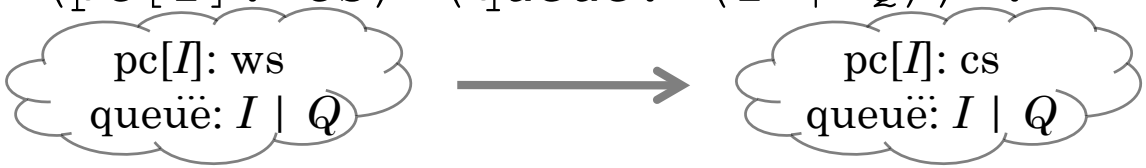
Another Way of Specifying OTSs (3)

- Transition `want`:

```
trans [want]: (pc[I]: rs) (queue: Q)
  => (pc[I]: ws) (queue :enq(Q,I)) .
```

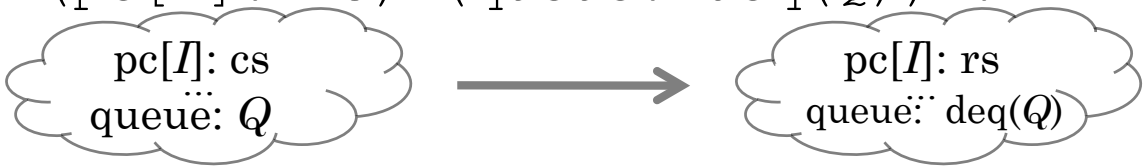
- Transition `try`:

```
trans [try]: (pc[I]: ws) (queue: (I | Q))
  => (pc[I]: cs) (queue: (I | Q)) .
```



- Transition `exit`:

```
rl [exit]: (pc[I]: cs) (queue: Q)
  => (pc[I]: rs) (queue: deq(Q)) .
```



Model Checking Invariants with CafeOBJ

- When six processes participates in Qlock, the initial state is expressed as

```
eq init = (pc[p1]: rs) (pc[p2]: rs) (pc[p3]: rs)
 (pc[p4]: rs) (pc[p5]: rs) (pc[p6]: rs) (queue: empty) .
```

- Checking if Qlock enjoys the mutex property.

```
open QLOCK
  red init =(1,*)=>* (pc[I]: cs) (pc[J]: cs) S .
close
```

✓ No counterexample was found.
✓ It took about 60 sec . for CafeOBJ to execute the command on a laptop with 2.33GH CPU and 3GB RAM.

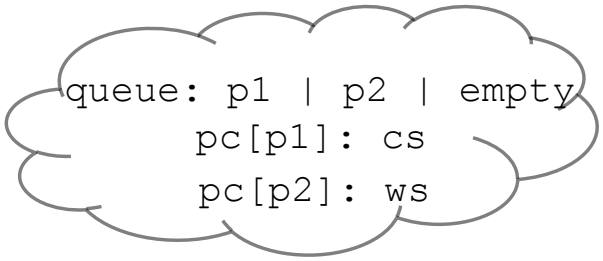
Maude in a Nutshell

- Another direct successor of OBJ3; a sibling language of CafeOBJ.
- Based on *rewriting logic* (RWL) & *membership equational logic* (MEL):
  - Static data are specified in terms of MEL as *equational specifications*.
  - Dynamic behaviors of systems are specified in terms of RWL as *rewrite theory specifications*.
- Equipped with meta-programming facilities, model checking facilities (the `search` command & the LTL model checker), etc.
- Maude (processors), some documents on Maude, and some examples can be obtained from

  `http://maude.cs.uiuc.edu/`

Rewrite Theory Sepcs. of OTSs

- A rewrite theory spec. of Qlock:




```
sorts Obs Sys .  subsort Obs < Sys .
*** configurations
op void : -> Sys [ctor] .
op _ _ : Sys Sys -> Sys [ctor assoc comm id: void] .
*** observable values
op pc[_]:_ : Pid Label -> Obs [ctor] .
op queue:_ : Queue -> Obs [ctor] .

rl [want]: (pc[I]: rs) (queue: Q)
  => (pc[I]: ws) (queue: enq(Q,I)) .
rl [try]: (pc[I]: ws) (queue: (I | Q))
  => (pc[I]: cs) (queue: (I | Q)) .
rl [exit]: (pc[I]: cs) (queue: Q)
  => (pc[I]: rs) (queue: deq(Q)) .
```

Loop:
  Remainder Section
rs: enq(*queue*,*i*);
ws: **repeat until** top(*queue*) = *i*;
  Critical Section
cs: deq(*queue*);

Model Checking Invariants with Maude (1)

- When six processes participates in Qlock, the initial state is expressed as

```
eq init = (pc[p1]: rs) (pc[p2]: rs) (pc[p3]: rs)
  (pc[p4]: rs) (pc[p5]: rs) (pc[p6]: rs) (queue: empty) .
```

- Checking if Qlock enjoys the mutex property.

```
search [1] in QLOCK :
init =>* (pc[I]: cs) (pc[J]: cs) S .
```

✓ No counterexample was found.
✓ It took about 0.4 sec . for Maude to execute the command on a laptop with 2.33GH CPU and 3GB RAM.
✓ 2 orders of magnitude faster than CafeOBJ.

Model Checking Invariants with Maude (2)

- For checking if a state predicate P always holds for a system S,

  `search [1] in` S `:` $init$ `=>*` $pattern$ `such that` $cond$ `.`

  where the negation of P is expressed as $pattern$ & $cond$.

```
search [1] in QLOCK :
init =>* (pc[I]: L1) (pc[J]: L2) S
such that not (L1 == cs and L2 == cs implies I == J) .
```

⇩ `I` is different from `J`.

```
search [1] in QLOCK :
init =>* (pc[I]: L1) (pc[J]: L2) S
such that L1 == cs and L2 == cs .
```

⇩ Both `L1` & `L2` are replaced with `cs`.

```
search [1] in QLOCK :
init =>* (pc[I]: cs) (pc[J]: cs) S .
```

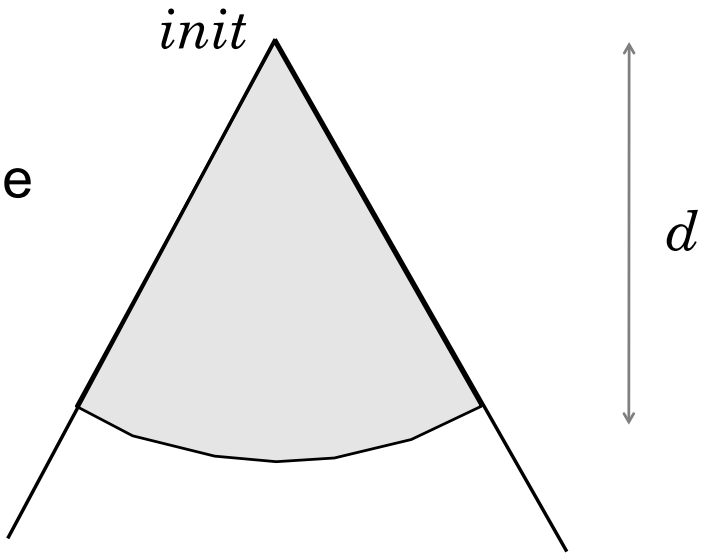
Model Checking Invariants with Maude (3)

- The maximum depth d can be specified.

`search [1,`d`] in` S `:` $init$ `=>*` $pattern$ `such that` $cond$ `.`

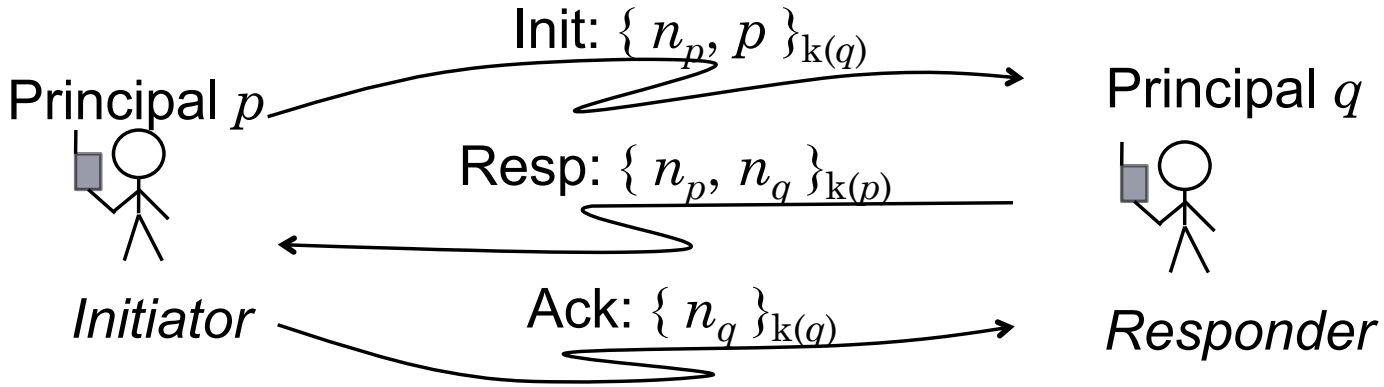
✓ Bounded model checking (BMC) of invariants can also be conducted.

The bounded reachable state space from $init$ whose depth is d.

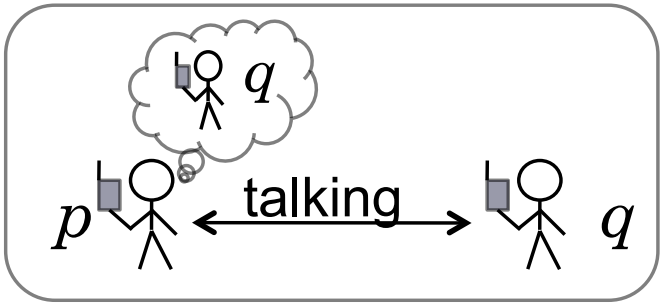


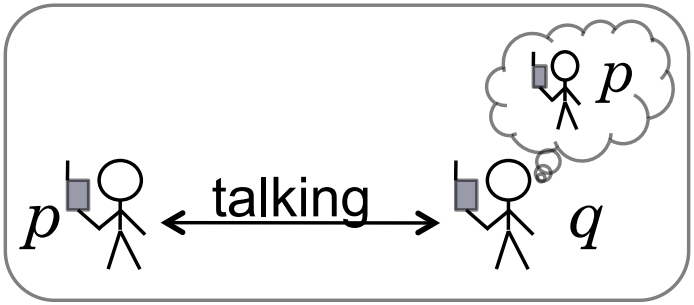
NSPK & Agreement Property

- NSPK ([Needham&Schroeder 1978]):



- Agreement Property (AP): Whenever a protocol run is successfully completed by p and q,
  - AP1: the principal with which p is communicating is really q, and
  - AP2: the principal with which q is communicating is really p.





Rewrite Theory Spec of NSPK (1)

- Configuration & observable values:

```
sorts Obs Sys .  subsort Obs < Sys .
*** configuration
op none : -> Sys [ctor] .
op _ _ : Sys Sys -> Sys [ctor assoc comm id: none] .
*** observable values
op network:_ : Soup{Message} -> Obs [ctor] .
op rands:_ : Soup{Random} -> Obs [ctor] .
op nonces:_ : Soup{Nonce} -> Obs [ctor] .
op prins:_ : Soup{Principal} -> Obs [ctor] .
op rands2:_ : Bag{Random} -> Obs [ctor] .
```

Rewrite Theory Spec of NSPK (2)

- A state:

  (network: NW) (rands: Rs) (nonces: Ns)
  (prins: Ps) (rands2: $Rs2$)

  ✓ NW: a collection of messages that have been sent up to the state.
  ✓ Rs: a collection of random numbers that have been sued up to the state.
  ✓ Ns: a collection of nonces that have been gleaned by the intruder up to the state.
  ✓ Ps: the collection of all the principals participating in NSPK. This value is not changed by any transitions.
  ✓ $\mathrm{Rs2}$: the collection of all the random numbers available in NSPK. This value is not changed by any transitions.

Rewrite Theory Spec of NSPK (3)

- Suppose that we have three principals including the intruder and two random numbers available.
- The initial state is expressed as

```
eq init
 = (network: empty) (rands: empty) (nonces: empty)
   (prins: (p q intruder)) (rands2: (r1 r2)) .
```

Rewrite Theory Spec of NSPK (4)

- The rewrite rule corresponding to sending a Resp message obeying NSPK:

```
crl [sdm2] :
 (network: (m1(Q?,Q,P,enc1(P,N,Q)) NW)) (rands: RS)
 (nonces: NS) (rands2: (R RS2))
 =>
 (network: (m2(P,P,Q,enc2(Q,N,n(P,Q,R)))
            m1(Q?,Q,P,enc1(P,N,Q)) NW))
 (rands: (R RS)) (rands2: (R RS2))
 (nonces: (if Q == intruder then N n(P,Q,R) NS else NS fi))
if not(R \in RS) .
```

- The rewrite rule corresponding to faking a Resp message using two nonces gleaned by the intruder:

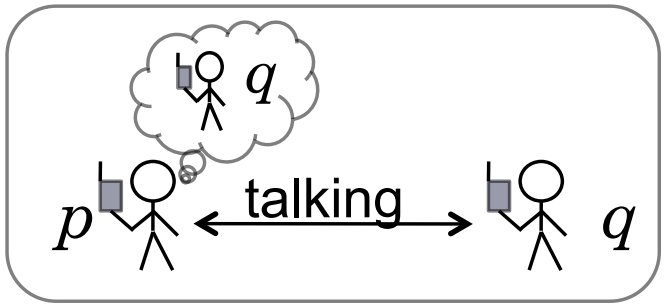
```
rl [fkm22] :
    (network: NW) (nonces: (N1 N2 NS)) (prins: (P Q PS))
    =>
    (network: (m2(intruder,P,Q,enc2(Q,N1,N2)) NW))
    (nonces: (N1 N2 NS)) (prins: (P Q PS)) .
```

Model Checking AP1

- The bounded reachable state space whose depth is up to 5 can be exhaustively traversed on a laptop with 2.33GH CPU and 3GB RAM.

```
search [1,5] in NSPK : init
=>* (network: (m1(P,P,Q,enc1(Q,n(P,Q,R),P))
               m2(Q?,Q,P,enc2(P,n(P,Q,R),N)) NW)) S
such that not(not(P == intruder) implies
              m2(Q,Q,P,enc2(P,n(P,Q,R),N))
               \in m2(Q?,Q,P,enc2(P,n(P,Q,R),N)) NW) .
```

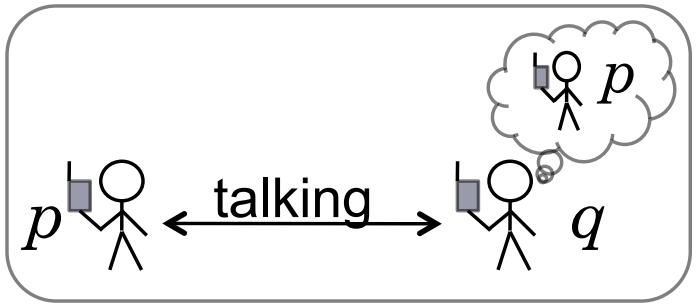
✓ No counterexample was found in the bounded reachable state space.



Model Checking AP2

```
search [1,5] in NSPK : init
=>* (network: (m2(Q,Q,P,enc2(P,N,n(Q,P,R)))
               m3(P?,P,Q,enc3(Q,n(Q,P,R))) NW)) S
such that not(not(Q == intruder) implies
              m3(P,P,Q,enc3(Q,n(Q,P,R)))
                \in m3(P?,P,Q,enc3(Q,n(Q,P,R))) NW) .
```

✓ No counterexample was found in the bounded reachable state space.



Lemmas for AP (1)

- One possible thing to do next is to try to prove AP1 & AP2 and conjecture some lemmas.
- We can do this with CafeOBJ, conjecturing 5 lemmas.

```
eq inv3(S,M2)
  = (M2 \in network(S)
     implies
     random(nonce1(cipher2(M2))) \in rands(S) and
     random(nonce2(cipher2(M2))) \in rands(S)) .

eq inv4(S,P,Q,N,R,M2)
  = (not(P = intruder) and not(Q = intruder) and
     m1(P,P,Q,enc1(Q,n(P,Q,R),P)) \in network(S) and
     M2 \in network(S) and cipher2(M2) = enc2(P,n(P,Q,R),N)
     implies
     m2(Q,Q,P,enc2(P,n(P,Q,R),N)) \in network(S)) .
```

Lemmas for AP (2)

```
eq inv5(S,N)
  = (N \in nonces(S)
     implies creator(N) = intruder or forwhom(N) = intruder) .

eq inv6(S,M3)
     = (M3 \in network(S)
        implies random(nonce(cipher3(M3))) \in rands(S)) .

eq inv7(S,P,Q,N,R,M3)
     = (not(P = intruder) and not(Q = intruder) and
        m2(Q,Q,P,enc2(P,N,n(Q,P,R))) \in network(S) and
        M3 \in network(S) and cipher3(M3) = enc3(Q,n(Q,P,R))
        implies
        m3(P,P,Q,enc3(Q,n(Q,P,R))) \in network(S)) .
```

✓ `inv5(S,N)` is what is called (*Nonce*) *Secrecy Property*.

Model Checking Lemmas

- No counterexample was found for `inv3`, `inv4`, `inv6` and `inv7`.
- But, a counterexample was found for `inv5`.

```
search [1,5] in NSPK : init
=>* (nonces: (N NS)) S
such that not(creator(N) == intruder
              or forwhom(N) == intruder) .
```

NSPK does not enjoy Secrecy Property!

✓ A state such that `inv5(S,N)` does not hold:

```
eq s115890 = (nonces: (n(q,p,r2) n(p,intruder,r1)))
 (network: (m1(intruder,p,q,enc1(q,n(p,intruder,r1),p))
   m1(p,p,intruder,enc1(intruder,n(p,intruder,r1),p))
   m2(intruder,intruder,p,enc2(p,n(p,intruder,r1),n(q,p,r2)))
   m2(q,q,p,enc2(p,n(p,intruder,r1),n(q,p,r2)))
   m3(p,p,intruder,enc3(intruder,n(q,p,r2)))))
 (rands: (r1 r2)) (prins: (intruder p q)) (rands2: (r1 r2)) .
```

Model Checking AP1 & AP2 from `s115890`

- No counterexample was found for AP1.
- But, a counterexample was found for AP2.

```
search [1,5] in NSPK : s115890
=>* (network: (m2(Q,Q,P,enc2(P,N,n(Q,P,R)))
               m3(P?,P,Q,enc3(Q,n(Q,P,R))) NW)) S
such that not(not(Q == intruder) implies
              m3(P,P,Q,enc3(Q,n(Q,P,R)))
                \in m3(P?,P,Q,enc3(Q,n(Q,P,R))) NW) .
```

NSPK does not enjoy Agreement Property!

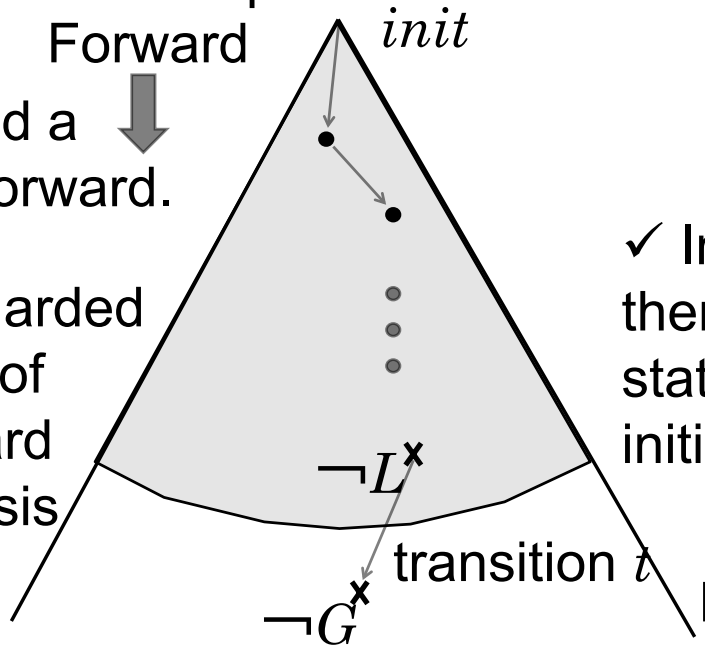
Another Combination of Inference & Search

✓ Suppose that a counterexample of property G exists outside of the bounded reachable state space that can be exhaustively traversed.
✓ Theorem proving (or induction) can conjecture a lemma L such that its counterexample exists in the space.

Forward

✓ BMC tries to find a counterexample forward.

✓ This can be regarded as a combination of forward & backward reachability analysis methods.



✓ Induction tries to show that there are no paths from any states such that $\neg G$ to any initial states.

Backward

K. Ogata, M. Nakano, W. Kong, K. Futatsugi: Induction-Guided Falsification, 8th ICFEM, LNCS 4260, Springer, pp.114-131 (2006).

Summary

- Qlock has been used as an example to describe another way of specifying OTSs for model checking and to demonstrate that Maude rewrite engine is much faster than CafeOBJ rewrite engine.
- Even though Maude rewrite engine is fast, the combinatorial explosion problem (the state explosion problem) is always with us.
- NSPK has been used as an example to describe one possible combination of inference & search, which can alleviate the problem.