The OTS/CafeOBJ Method and Some Future Directions

Kazuhiro Ogata[1,2] *Kokichi Futatsugi*[2]

[1] NEC Software Hokuriku, Ltd.

[2] Graduate School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)

Introduction

We have been successfully applying the OTS/CafeOBJ method to modeling and verification of distributed systems such as security protocols.

- A system is modeled as an observational transition system, or an OTS.
- The OTS is written in CafeOBJ.
- Properties to be proved are expressed as CafeOBJ terms.
- Proofs, or proof scores showing that the OTS has the properties are written in CafeOBJ.
- The proof scores are veriified by executing them with the CafeOBJ system.

Outline of Talk

- Observational Transition Systems (OTSs)
- Example: Queuing Lock
- Compositionally Writing Proof Scores
- Ongoing and Future Work
  - Generating Proof Scores
  - Model-Checking OTS/CafeOBJ specifications

Observational Transition Systems

An OTS $\mathcal{S}$ consists of $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$:

- $\mathcal{O}$: A set of observers.

  Each $o \in \mathcal{O}$ is a function $o : \Upsilon \rightarrow D$, where D is a data type.

  $v_1 =_{\mathcal{S}} v_2 \stackrel{\text{def}}{=} \forall o \in \mathcal{O}. o(v_1) = o(v_2)$.

- $\mathcal{I}$: A set of initial states.

- $\mathcal{T}$: A set of conditional transition rules.

  Each $\tau \in \mathcal{T}$ is a function $\tau : \Upsilon/{=_{\mathcal{S}}} \rightarrow \Upsilon/{=_{\mathcal{S}}}$ on equivalence classes of Υ wrt $=_{\mathcal{S}}$.

  The condition c_τ of a transition rule $\tau \in \mathcal{T}$ is called *the effective condition*.

Executions and Invariants

An execution of $\mathcal{S}$ is an infinite seuqence $\upsilon_0, \upsilon_1, \ldots$ of states satisfying:

- *Initiation*: $\upsilon_0 \in \mathcal{I}$.
- *Consecution*: For each $i \in \{0, 1, \ldots\}$, $\upsilon_{i+1} =_{\mathcal{S}} \tau(\upsilon_i)$ for some $\tau \in \mathcal{T}$.

A state is called *reachable* wrt $\mathcal{S}$ iff there exists an execution of $\mathcal{S}$ in which the state appears.
Let $\mathcal{R}_{\mathcal{S}}$ be the set of all the reachable states wrt an OTS $\mathcal{S}$.

If predicate p is true in every state of $\mathcal{R}_{\mathcal{S}}$, p is called invariant to S, which is defined as follows:

$$\textbf{invariant } p \stackrel{\text{def}}{=} \forall \upsilon \in \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)\,.$$

Indexed Observers and Transition Rules

Observers and transition rules may be indexed.

- Observers are generally denoted by $o_{i_1,\ldots,i_m}$.
- Transition rules are generally denoted by $\tau_{j_1,\ldots,j_n}$.

where $m, n \geq 0$ and there exists data types D_k such that $k \in D_k \, (k = i_1, \ldots, i_m, j_1, \ldots, j_n)$.

For example,

- a_p : Observer denoting an interger array a possessed by a process p.
- $inc\text{-}a_{p,i}$: Transition rule denoting the increment of the ith element of the array.

CafeOBJ

- Two kinds of sorts.
  - *Visible sort* denotes an abstract data type.
  - *Hidden sort* denotes the state space of an abstract machine.
- Two kinds of operators for hidden sorts.
  - *Action operators* denote state transitions of an abstract machine.
  - Only *observation operators* can be used to know the inside of an abstract machine.

Writing OTSs in CafeOBJ

- The state space Υ is denoted by a hidden sort, say H.
- An observer $o_{i_1,\ldots,i_m}$ is denoted by an observation operator.

$$\texttt{bop}\ o\ \texttt{:}\ H\ V_{i_1}\ \ldots\ V_{i_m}\ \texttt{->}\ V$$

- A transition rule $\tau_{j_1,\ldots,j_n}$ is denoted by an action operator.

$$\texttt{bop}\ a\ \texttt{:}\ H\ V_{j_1}\ \ldots\ V_{j_n}\ \texttt{->}\ H$$

- $\tau_{j_1,\ldots,j_n}$ is defined with equations by describing how the value returned by each observer $o_{i_1,\ldots,i_m}$ changes.

$$\begin{array}{l}\texttt{eq}\ o(a(S, X_{j_1}, \ldots, X_{j_n}), X_{i_1}, \ldots, X_{i_m})\ \texttt{=}\ \mathit{NewValue}\\ \qquad \texttt{if}\ c\text{-}a(S, X_{j_1}, \ldots, X_{j_n})\ \texttt{.}\end{array}$$

  c-a denotes the effective condition of $\tau_{j_1,\ldots,j_n}$.

Queuing Lock

Program executed by process i:

```
l1: put($queue, i$)
l2: repeat until top($queue$) $= i$
Critical Section
cs: get($queue$)
```

- *queue* is the queue of process IDs shared by all processes.
- put($queue, i$) puts i into *queue* at the end.
- top($queue$) returns the top of *queue*.
- get($queue$) removes the top of *queue*.

Modeling Queuing Lock as an OTS

- Observers.
  - *queue* returns the queue shared by all processes. It initially returns the empty queue.
  - pc_i ($i \in Pid$) returns the label of a command that process i will execute next. Each pc_i initially returns label l1.
- Transition rules.
  - $want_i$ ($i \in Pid$) denotes that process i executes the command at label l1.
  - try_i ($i \in Pid$) denotes that process i executes one iteration of the loop at label l2.
  - $exit_i$ ($i \in Pid$) denotes that process i executes the command at label cs.

Description of the OTS in CafeOBJ (1)

Signature of the CafeOBJ specification of the OTS:

```
*[Sys]*
-- any initial state
op init : -> Sys
-- 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
```

- Observers.
  - $queue$
  - $pc_i \, (i \in Pid)$
- Transition rules.
  - $want_i \, (i \in Pid)$
  - $try_i \, (i \in Pid)$
  - $exit_i \, (i \in Pid)$

Description of the OTS in CafeOBJ (2)

Action operator `want` is defined with the equations:

```
op c-want : Sys Pid -> Bool
eq c-want(S,I) = (pc(S,I) = l1) .
--
ceq pc(want(S,I),J)
    = (if I = J then l2 else pc(S,J) fi) if c-want(S,I) .
ceq queue(want(S,I)) = put(queue(S),I)   if c-want(S,I) .
ceq want(S,I)        = S               if not c-want(S,I) .
```

Description of the OTS in CafeOBJ (3)

Action operator `try` is defined with the equations:

```
op c-try : Sys Pid -> Bool
eq c-try(S,I) = (pc(S,I) = l2 and top(queue(S)) = I) .
--
ceq pc(try(S,I),J)
    = (if I = J then cs else pc(S,J) fi) if c-try(S,I) .
eq  queue(try(S,I)) = queue(S) .
ceq try(S,I)        = S                  if not c-try(S,I) .
```

Description of the OTS in CafeOBJ (4)

Action operator `exit` is defined with the equations:

```
op c-exit : Sys Pid -> Bool
eq c-exit(S,I) = (pc(S,I) = cs) .
--
ceq pc(exit(S,I),J)
    = (if I = J then l1 else pc(S,J) fi) if c-exit(S,I) .
ceq queue(exit(S,I)) = get(queue(S))     if c-exit(S,I) .
ceq exit(S,I)        = S                 if not c-exit(S,I) .
```

Mutual Exclusion

To prove the queuing lock mutually exclusive, all we have to do is to prove that inv1 is invariant to the OTS modeling the queuing lock.

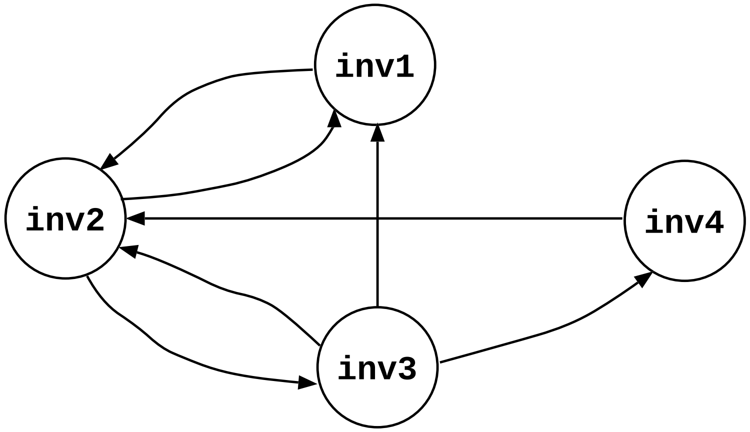
```
eq inv1(S,I,J)
  = (pc(S,I) = cs and pc(S,J) = cs implies I = J) .
```

To this end, we need the following:

```
eq inv2(S,I) = (pc(S,I) = cs implies top(queue(S)) = I) .
eq inv3(S,I) = (pc(S,I) = l2 or pc(S,I) = cs
                        implies not empty?(queue(S))) .
eq inv4(S,I) = (pc(S,I) = l2 implies I \in queue(S)) .
```

Strategies of Proofs

- First prove that `inv2`, `inv3` and `inv4` are invariant, and then prove that `inv1` is invariant using the proved invariants.



- Prove that `inv1` ∧ `inv2` ∧ `inv3` ∧ `inv4` is invariant.
- Prove that `inv1`, `inv2`, `inv3` and `inv4` are invariant simultaneously.

Proof of `inv1`

- Consider the inductive case where `try` preserves `inv1`.

  We should prove $\texttt{inv1}(s, i, j) \Rightarrow \texttt{inv1}(\texttt{try}(s, k), i, j)$.

- `inv2` is used to strengthen the inductive hypothesis.

$$(\texttt{inv2}(s, i) \wedge \texttt{inv2}(s, j) \wedge \texttt{inv1}(s, i, j)) \Rightarrow \texttt{inv1}(\texttt{try}(s, k), i, j)$$

- When the effective condition is true, the case is split into 4 subcases.

  1. $(i = k) \wedge (j = k)$, which needs nothing.
  2. $(i = k) \wedge (j \neq k)$, which needs $\texttt{inv2}(s, j)$.
  3. $(i \neq k) \wedge (j = k)$, which needs $\texttt{inv2}(s, i)$.
  4. $(i \neq k) \wedge (j \neq k)$, which needs nothing.

Proof Passage of `inv1`

Proof passage of subcase 2 ($(i = k) \wedge (j \neq k)$):

```
open ISTEP
-- arbitrary objects
  op k : -> Pid .
-- assumptions
  -- eq c-try(s,k) = true .
  eq pc(s,k) = l2 .  eq top(queue(s)) = k .
  --
  eq i = k .  eq (j = k) = false .
-- successor state
  eq s' = try(s,k) .
-- check; istep1(i,j) = inv1(s,i,j) implies inv1(s',i,j)
  red inv2(s,j) implies istep1(i,j) .
close
```

Proof of `inv2`

- Consider the inductive case where `exit` preserves `inv2`.

  We should prove $\mathtt{inv2}(s, i) \Rightarrow \mathtt{inv2}(\mathtt{exit}(s, k), i)$.

- `inv1` is used to strengthen the inductive hypothesis.

$$(\mathtt{inv1}(s, i, k) \wedge \mathtt{inv2}(s, i)) \Rightarrow \mathtt{inv2}(\mathtt{try}(s, k), i)$$

- When the effective condition is true, the case is split into 3 subcases.

  1. $i = k$, which needs nothing.
  2. $(i \neq k) \wedge (\mathtt{pc}(s, i) = \mathtt{cs})$, which needs $\mathtt{inv1}(s, i, k)$.
  3. $(i \neq k) \wedge (\mathtt{pc}(s, i) \neq \mathtt{cs})$, which needs nothing.

Proof Passage of `inv2`

Proof passage of subcase 2 ($(i \neq k) \wedge (\texttt{pc}(s, i) = \texttt{cs})$):

```
open ISTEP
-- arbitrary objects
  op k : -> Pid .
-- assumptions
  -- eq c-exit(s,k) = true .
  eq pc(s,k) = cs .
  --
  eq (i = k) = false .  eq pc(s,i) = cs .
-- successor state
  eq s' = exit(s,k) .
-- check; istep2(i) = inv2(s,i) implies inv2(s',i)
  red inv1(s,i,k) implies istep2(i) .
close
```

Compositional Proofs of Invariants (1)

- Consider proving that $\mathit{pred}_1(s, x_1)$ is invariant to an OTS.
- Together with $\mathit{pred}_2(s, x_2), \ldots, \mathit{pred}_n(s, x_n)$.

  Let $\mathit{pred}(s, x_1, \ldots, x_n)$ be $\mathit{pred}_1(s, x_1) \wedge \ldots \mathit{pred}_n(s, x_n)$.
- Consider the inductive case where an action operator a preserves $\mathit{pred}(s, x_1, \ldots, x_n)$.

  We should prove

  $$\mathit{pred}(s, x_1, \ldots, x_n) \Rightarrow \mathit{pred}(\mathrm{a}(s, y), x_1, \ldots, x_n)$$

  or

  $$\begin{array}{c} \mathit{pred}_1(s, x_1) \Rightarrow \mathit{pred}_1(\mathrm{a}(s, y), x_1) \\ \vdots \\ \mathit{pred}_n(s, x_n) \Rightarrow \mathit{pred}_n(\mathrm{a}(s, y), x_n) \end{array} \tag{1}$$

Compositional Proofs of Invariants (2)

- Let $\mathit{pred}_i(s, x_i) \Rightarrow \mathit{pred}_i(\mathrm{a}(s, y), x_i)$ be one of such formulas.
- What strengthens the inductive hypothesis can be $\mathit{pred}_{j_1}(s, t_{j_1}) \wedge \ldots \wedge \mathit{pred}_{j_k}(s, t_{j_k})$, where $1 \leq j_1, \ldots, j_k \leq n$, where t_j is a term denoting an instance of x_j.

  Let SIH_i be $\mathit{pred}_{j_1}(s, t_{j_1}) \wedge \ldots \wedge \mathit{pred}_{j_k}(s, t_{j_k})$.
- The proof of the $i$the formula of (1) can be replaced with the proof of the formula:

$$(\mathit{SIH}_i \wedge \mathit{pred}_i(s, x_i)) \Rightarrow \mathit{pred}_i(\mathrm{a}(s, y), x_i) \tag{2}$$

Compositional Proofs of Invariants (3)

$$(\mathit{SIH}_1 \wedge \mathit{pred}_1(s, x_1)) \Rightarrow \mathit{pred}_1(\mathsf{a}(s, y), x_1)$$
$$\vdots$$
$$(\mathit{SIH}_n \wedge \mathit{pred}_n(s, x_n)) \Rightarrow \mathit{pred}_n(\mathsf{a}(s, y), x_n)$$

where $\mathit{SIH}_i = \mathit{pred}_{j^i_1}(s, t_{j^i_1}) \wedge \ldots \wedge \mathit{pred}_{j^i_{k_i}}(s, t_{j^i_{k_i}})$, $i = 1, \ldots, n$.

From these n formulas, the following can be deduced.

$$\begin{aligned}&(\mathit{SIH}_1 \wedge \ldots \wedge \mathit{SIH}_n) \wedge (\mathit{pred}_1(s, x_1) \wedge \ldots \wedge \mathit{pred}_n(s, x_n))\\ &\Rightarrow (\mathit{pred}_1(\mathsf{a}(s, y), x_1) \wedge \ldots \wedge \mathit{pred}_n(\mathsf{a}(s, y), x_n))\end{aligned}$$

$\mathit{SIH}_1 \wedge \ldots \wedge \mathit{SIH}_n$ can be used as the inductive hypothesis becasue $x_1, \ldots, x_n$ are just instantiated.

Therefore, if this formula is proved, then it is shown that action operator a preserves $\mathit{pred}_1(s, x_1) \wedge \ldots \wedge \mathit{pred}_n(s, x_n)$.

Compositional Proofs of Invariants (4)

- The case may have to be split into multiple subcases to prove

$$(\mathit{SIH}_i \wedge \mathit{pred}_i(s, x_i)) \Rightarrow \mathit{pred}_i(\mathrm{a}(s, y), x_i) \tag{2}$$

- Suppose that the case is split into l subcases denoted by $\mathit{case}^i_1, \ldots, \mathit{case}^i_l$, which should satisfy

$$(\mathit{case}^i_1 \vee \ldots \vee \mathit{case}^i_l) = \mathrm{true}\,.$$

- The proof of (2) can be replaced with the proofs of the l formulas:

$$\begin{gathered}(\mathit{case}^i_1 \wedge \mathit{SIH}_i \wedge \mathit{pred}_1(s, x_i)) \Rightarrow \mathit{pred}_1(\mathrm{a}(s, y), x_i)\\ \vdots \\ (\mathit{case}^i_l \wedge \mathit{SIH}_i \wedge \mathit{pred}_1(s, x_i)) \Rightarrow \mathit{pred}_1(\mathrm{a}(s, y), x_i)\end{gathered}$$

Proof Scores of Invariants (1)

In module `INV`:

`op` inv_1 `:` H V_1 `-> Bool` `--` $pred_1(s, x_1)$

$\ldots$ `--` $\ldots$

`op` inv_n `:` H V_n `-> Bool` `--` $pred_n(s, x_n)$

`eq` $inv_1(S, X_1)$ `=` $pred_1(S, X_1)$ `.`

$\ldots$

`eq` $inv_n(S, X_n)$ `=` $pred_n(S, X_n)$ `.`

In module `ISTEP`:

`op` $istep_1$ `:` V_1 `-> Bool` `--` $pred_1(s, x_1) \Rightarrow pred_1(\mathrm{a}(s, y), x_1)$

$\ldots$ `--` $\ldots$

`op` $istep_n$ `:` V_n `-> Bool` `--` $pred_1(s, x_n) \Rightarrow pred_1(\mathrm{a}(s, y), x_n)$

`eq` $istep_1(X_1)$ `=` $inv_1(s, X_1)$ `implies` $inv_1(s', X_1)$ `.`

$\ldots$

`eq` $istep_n(X_n)$ `=` $inv_n(s, X_n)$ `implies` $inv_n(s', X_n)$ `.`

Proof Scores of Invariants (2)

Proof passage of $case^i_j$ where an action operator a preserves $pred_i(s, t_i)$:

```
open ISTEP
-- arbitary objects
 op $y$ : -> $V$ .
 Declare constants if necessary
-- assumptions
 Declare equations denoting $case^i_j$
-- successor state
 eq $s'$ = $a(s, y)$ .
-- check; $istep_i(x_i) = pred_i(s, t_i)$ implies $pred_i(s', t_i)$
 red $(pred_{j_1}(s, t_{j_1})$ and $\ldots$ $pred_{j_k}(s, t_{j_k}))$ implies $istep_i(x_i)$ .
close
```

Some Merits

- Relieve the complexity of the reduction of logical formulas.

  Because the compositional writing of proof scores makes it possible to focus on each conjunct $pred_i(s, x_i)$ of a large formula $pred_1(s, x_1) \wedge \ldots pred_n(s, x_n)$.

- Ease the complexity of case analysis.

  Because the compositional writing of proof scores makes it possible to do case analysis for each conjunct $pred_i(s, x_i)$ only.

Case Studies

- NSLPK authentication protocol

  17 invariants verified; 9,500 lines.
- *i*KP electronic payment protocols

  18 invariants verified; proof scores of 22,000 lines.
- Horn-Preneel micropayment protocol

  24 invariants verified; proof scores of 22,000 lines.
- NetBill electronic commerce protocol

  36 invariants verified; proof scores of 79,000 lines.
- SET payment protocol

  20 invariants verified; proof scores of 40,000 lines.
- TLS handshake protocol

  18 invariants verified; proof scores of 14,000 lines.

Ongoing and Future Work

- A script language and a tool (called Gateau) that translates scripts written in the language into proof scores have been being designed.

  A prototype has been implemented and used for verifying the queuing lock and the NSLPK authentication protocol.

- A tool that translates OTS/CafeOBJ specifications into SMV ones has been designed.

  A prototype has been implemented and used for model-checking an OTS/CafeOBJ specification of the NSPK authentication protocol.

  But, it is difficult to translate CafeOBJ user-defined data type into those encoded in SMV limited data types.

  Therefore, we are going to design a tool that translates OTS/CafeOBJ specifications into Maude ones.

Gateau Scripts

The Gateau script of `inv1` of the queuing lock:

```
#base: inv1(init,i,j)

#inductive: istep1(i,j)

#successor: s'

#action: want(s,k)
#constants: k : -> Pid
#effective: pc(s,k) = l1
#case: i = k
       (i = k) = false
#case: j = k
       (j = k) = false
```

```
#action: try(s,k)
#constants: k : -> Pid
#effective: pc(s,k) = l2
            top(queue(s)) = k
#lemma: inv2(s,i) and inv2(s,j)
#case: i = k
       (i = k) = false
#case: j = k
       (j = k) = false

#action: exit(s,k)
#constants: k : -> Pid
#effective: pc(s,k) = cs
#case: i = k
       (i = k) = false
#case: j = k
       (j = k) = false
```

Experimental Data of Gateau

- QLOCK – We have verified that the queuing lock is mutually exclusive with Gateau.
- NSLPK – We have verified that the NSLPK authentication protocol has the nonce secrecy and one-to-many agreement properties with Gateau.

Examples	Size of Gateau Scripts	Size of Generated Proof Cores
QLOCK	142 lines	2946 lines
NSLPK	1997 lines	19251 lines

The size of the hand-written proof scores are

QLOCK	966 lines
NSLPK	9664 lines

Writing OTSs in Maude (1)

- $\mathcal{O}$ and $\mathcal{T}$ are denoted by sorts, say `OValue` and `TRule`; Υ is denoted by a sort, say `Sys`.
- A snapshot of $\mathcal{S}$ is represented by a bag of observers and transition rules.

```
subsort OValue TRule < Sys .

op none : -> Sys .
op __ : Sys Sys -> Sys [assoc comm id: none]
```

Generally, a snapshot of $\mathcal{S}$ is as the following form:

$$\textit{ovalue}\text{-}1 \ \ldots\ \textit{ovalue}\text{-}M\ \textit{trule}\text{-}1\ \ldots\ \textit{trule}\text{-}N$$

where *ovalue-i* $(i = 1, \ldots, M)$ is a term denoting an observer, and *trule-i* $(i = 1, \ldots, N)$ is a term denoting a transition rule.

Writing OTSs in Maude (2)

- $o_{i_1,\ldots,i_m} : \Upsilon \to D$, where $k \in D_k\,(k = i_1, \ldots, i_m)$, is denoted by the operator declared as follows:

  $$\texttt{op}\ (o\texttt{[}\_, \ldots, \_\texttt{]}\ \texttt{:}\_)\ \texttt{:}\ V_{i_1}\ \ldots\ V_{i_m}\ V\ \texttt{->}\ \texttt{OValue}\ \texttt{.}$$

  where $V_k\,(k = i_1, \ldots, i_m)$ and V correspond to D_k and D.

- $\tau_{j_1,\ldots,j_n} : \Upsilon \to \Upsilon$, where $k \in D_k\,(k = j_1, \ldots, j_n)$, is denoted by the operator declared as follows:

  $$\texttt{op}\ r\ \texttt{:}\ V_{j_1}\ \ldots\ V_{j_n}\ \texttt{->}\ \texttt{TRule}\ \texttt{.}$$

  where $V_k\,(k = j_1, \ldots, j_n)$ correspond to D_k.

Writing OTSs in Maude (3)

- Transition rules are defined in Maude rules.

  Suppose that observers needed and affected by the execution of the transition rule $\tau_{j_1,\ldots,j_n}$ are $o^1_{i^1_1,\ldots,i^1_{m_1}}, \ldots, o^l_{i^l_1,\ldots,i^l_{m_l}}$.

$$
\begin{array}{l}
\texttt{crl [rule-}r\texttt{] :} \\
\quad r(X_{j_1}, \ldots, X_{j_n}) \\
\quad (o^1[X_{i^1_1}, \ldots, X_{i^1_{m_1}}] \texttt{ : } X_1) \ \ldots \ (o^l[X_{i^l_1}, \ldots, X_{i^l_{m_l}}] \texttt{ : } X_l) \\
\quad \texttt{=>} \\
\quad r(X_{j_1}, \ldots, X_{j_n}) \\
\quad (o^1[X_{i^1_1}, \ldots, X_{i^1_{m_1}}] \texttt{ : } X'_1) \ \ldots \ (o^l[X_{i^l_1}, \ldots, X_{i^l_{m_l}}] \texttt{ : } X'_l) \\
\quad \texttt{if } c\text{-}r(X_{j_1}, \ldots, X_{j_n}, X_{i^1_1}, \ldots, X_{i^1_{m_1}}, X_1, \ldots, X_{i^l_1}, \ldots, X_{i^l_{m_l}}, X_l) \ \texttt{.}
\end{array}
$$

Writing the Queing Lock in Maude (1)

The operators denoting the observers and transition rules are declared as follows:

```
*** Observers
op pc[_] :_ : Pid Label -> OValue .
op queue :_ : Queue -> OValue .
*** Transition rules
op want : Pid -> TRule .
op try  : Pid -> TRule .
op exit : Pid -> TRule .
```

- Observers.
  - $pc_i\,(i \in Pid)$
  - $queue$
- Transition rules.
  - $want_i\,(i \in Pid)$
  - $try_i\,(i \in Pid)$
  - $exit_i\,(i \in Pid)$

Writing the Queuing Lock in Maude (2)

The transition rules are defined with these rules.

```
crl [want] :
  want(P) (pc[P] : L) (queue : Q)
  => want(P) (pc[P] : l2) (queue : put(Q,P))
  if L == l1 .

crl [try] :
  try(P) (pc[P] : L) (queue : Q)
  => try(P) (pc[P] : cs) (queue : Q)
  if L == l2 and top(Q) == P .

crl [exit] :
  exit(P) (pc[P] : L) (queue : Q)
  => exit(P) (pc[P] : l1) (queue : get(Q))
  if L == cs .
```

Model-Checking (1)

State predicates needed to describe properties to be checked:

```
mod QLOCK-PREDS is
  pr QLOCK .
  inc SATISFACTION .
  subsort Sys < State .
  op crit : Pid -> Prop .
  var P : Pid .
  var S : Sys .
  eq (pc[P] : cs) S |= crit(P) = true .
endm
```

Model-Checking (2)

Initial state and LTL formula denoting mutual exclusion:

```
mod QLOCK-CHECK is
 inc QLOCK-PREDS .  inc MODEL-CHECKER .  inc LTL-SIMPLIFIER .
 ops p1 p2 p3 : -> Pid .
 op init : -> Sys .  op mutex : -> Formula .
 eq init = want(p1) try(p1) exit(p1) want(p2) try(p2) exit(p2)
           want(p3) try(p3) exit(p3)
           (pc[p1] : l1) (pc[p2] : l1) (pc[p3] : l1) (queue : empty) .
 eq mutex = ([] ~(crit(p1) /\ crit(p2)))
         /\ ([] ~(crit(p1) /\ crit(p3)))
         /\ ([] ~(crit(p2) /\ crit(p3))) .
endm
```

Then, model-check that the finite OTS has the property.

```
red modelCheck(init,mutex) .
```