Automatic Translation from OTS/CafeOBJ to OTS/Maude

Min Zhang

JAIST

2010-2-5

Table of Contents

OTS/Maude specification

An OTS/Maude specification is a program that specifies an OTS in Maude language. Some tips for writing an OTS/Maude specification :

- Representation of states :

```
fmod STATE is
  sort State .
  op empty : -> State [ctor] .
  op __ : State State -> State [assoc comm id: empty] .
endfm
```

A state consists of a finite set of sub-states. A sub-state is of the form like :

`(O [X1,...,Xm] : X )` or `(O : X)`

Example 3. (Representation of states for *QLOCK*)

`(pc[p] : Lp) (pc[q] : Lq)(queue : Q)`

OTS/Maude specification

- Representation of transitions :
  A transition is usually specified by a rewriting rule like :

```
rl [t] : (pc[p] : Lp) (pc[q] : Lq)(queue : Q) =>
   (pc[p] : Lp') (pc[q] : Lq')(queue : Q') . or

crl [t] : (pc[p] : Lp) (pc[q] : Lq)(queue : Q) =>
(pc[p] : Lp') (pc[q] : Lq')(queue : Q') if cond .
```

Note that

- We only need to focus on those sub-states whose values are changed after the transition ;
- Variables in the RHS or condition part of a rewriting rule must appear in LHS to make the rule executable.

Basic rules for translation (1) :

- Translation of observations :

$$\frac{o\ :\ H\ S_{o1}\ \ldots\ S_{om}\ \rightarrow\ S_o}{o[\_,\ldots,\_]{:}\_\ :\ S_{o1}\ \ldots\ S_{om}\ S_o\ \rightarrow\ State} \qquad (1)$$

Example 4.

$$\frac{\texttt{bop pc : Sys Pid -> Label}}{\texttt{op pc[\_] :\_ : Pid Label -> State}}\ ;$$

Basic rules for translation (2) :

- Translation of transitions :
  - For each observer in a transition :

$$\frac{\texttt{ceq } o(t(S,Y_{t1},\ldots,Y_{tn}),X_{o1},\ldots,X_{om})=o\text{-}t(S,Y_{t1},\ldots,Y_{tn},X_{o1},\ldots,X_{om}) \text{ if } c\text{-}t(S,Y_{t1},\ldots,Y_{tn}).}{\texttt{crl } (o[X_{o1},\ldots,X_{om}]:\ X_o)\Rightarrow(o[X_{o1},\ldots,X_{om}]:\ A_o) \text{ if } c\text{-}x'.} \quad (2)$$

Where

$$A_o = o\text{-}t(S, Y_{t1}, \ldots, Y_{tn}, X_{o1}, \ldots, X_{om})[o(S, X_{o1}, \ldots, X_{om}) \leftarrow X_o]$$

Example 4.

```
ceq pc(try(S,I),J) = (if I = J then cs else pc(S,J) fi) if c-try(S,I) .
crl (pc [J] : L1) => pc [J] : (if I = J then cs else L1 fi) if c-try' .
```

```
eq queue(try(S,I)) = queue(S) if c-try(S,I) .
crl (queue : Q) => (queue : Q) if c-try' .
```

Basic rules for translation (3) :

- Translation of transitions
  - Combination of rewriting rules for a transition :

$$\frac{\begin{array}{c}\texttt{crl}\ S_o \Rightarrow S'_o\ \textit{if}\ \textit{c-t}' \\ \texttt{crl}\ S_{o'} \Rightarrow S'_{o'}\ \textit{if}\ \textit{c-t}'\end{array}}{\texttt{crl}\ S_o\ S_{o'} \Rightarrow S'_o\ S'_{o'}\ \textit{if}\ \textit{c-t}'} \quad (3)$$

Where $S_o, S'_o, S_{o'}, S'_{o'} : \mathit{State}$

Example 5.

```
crl (pc [J] : L1) => pc [J] : (if I = J then cs else L1 fi) if c-try' .
crl (queue : Q) => (queue : Q) if c-try' .
-------------------------------------------------------------------
 crl (pc [J] : L1) (queue : Q) =>
     (pc [J] : (if I = J then cs else L1 fi)) (queue : Q) if c-try' .
```

Basic rules for translation (4) :

- Translation of transitions :
  - For the effective condition of a transition :

$$\frac{\texttt{eq } c\text{-}t(S,Y_{t1},\ldots,Y_{tn})=C\ .}{c\text{-}t'=C[o(S,X_{o1},\ldots,X_{om})\leftarrow X_o,\ldots]}\ (4)$$

Example 6.

$$\frac{\texttt{eq c-try(S,I) = (pc(S,I) = ws and top(queue(S)) = I) .}}{\texttt{c-try' = ((L2 = ws) and top(Q) = I)}}$$

Basic rules for translation (5) :

- Translation of transitions :
  - Addition of translated effective condition to rewriting rule :

Condition :

$$c\text{-}t' = C[o(S, X_{o1}, \ldots, X_{om}) \leftarrow V_o, \ldots]$$
$$\texttt{crl}\ S_t \Rightarrow S'_t\ \textit{if}\ c\text{-}t'\ .$$

1. if $(o[X_{o1}, \ldots, X_{om}] : V_o)$ is not in S_t.

$$\frac{}{\texttt{crl}\ (o[X_{o1},\ldots,X_{om}]{:}V_o)\ S_t \Rightarrow (o[X_{o1},\ldots,X_{om}]{:}V'_o)\ S'_t\ \textit{if}\ C[o(S,X_{o1},\ldots,X_{om}) \leftarrow V_o,\ldots]\ .} \quad (5)$$

2. if $(o[X_{o1}, \ldots, X_{om}] : X_o)$ is in S_t.

$$\frac{}{\texttt{crl}\ S_t \Rightarrow S'_t\ \textit{if}\ C[o(S,X_{o1},\ldots,X_{om}) \leftarrow V_o,\ldots]\ .} \quad (6)$$

Basic rules for translation (6) :

Example 8.

```
c-try'(I,L2,Q) = (pc(S,I) = ws and top(queue(S)) = I)[pc(S,I) ← L2, queue(S) ← Q].

crl try'(I) (pc [J] : L1) (queue : Q) => try'(I)
 (pc [J] : (if I = J then cs else L1 fi)) (queue : Q) if c-try(S,I) .
```

```
crl try'(I) (pc [J] : L1) (queue : Q) (pc [I] : L2) =>
    try'(I) (pc [J] : (if I = J then cs else L1 fi)) (queue : Q)
    (pc [I] : (if I = I then cs else L2 fi)) if (L2 = ws and top(Q) = I) .
```

After optimization 1 :

```
crl [try] : (try[I] : ws) (try[J] : L2) (queue : Q) =>

          (try[I] : cs) (try[J] : L2) (queue : Q) if top(Q) = I .
```

After optimization 2 :

```
crl [try] : (try[I] : ws) (queue : Q) => (try[I] : cs) (queue : Q) if top(Q) = I .
```

Basic rules for translation (7) :

- Translation of initial state :
  - Construction of sub-state :

$$\frac{\texttt{eq } o(init,X_{o1},\ldots,X_{om})=C_o}{(o[X_{o1},\ldots,X_{om}]:C_o\downarrow)[X_{o1}\leftarrow x_{o1},\ldots,X_{om}\leftarrow x_{om}]} \quad (7)$$

    Where, x_{oi} is a constant of $S_{oi}, (1 \leq i \leq m)$
  - Representation of an initial state in OTS/Maude :

$$\texttt{eq } init' = (o[X_{o1},\ldots,X_{om}] : C_o \downarrow)[X_{o1} \leftarrow x_{o1},\ldots,X_{om} \leftarrow x_{om}]\ldots$$

Example 9.

$$\frac{\texttt{eq pc(init, I :Pid) = rm .}}{\texttt{(pc [p] : rm)(pc [q] : rm)}}$$

NSPK (1) :

- Mechanism

$$\begin{array}{lll} init: & p \longrightarrow q & \{n_p, p\}_{k(q)} \\ resp: & q \longrightarrow p & \{n_p, n_q\}_{k(p)} \\ ack: & p \longrightarrow q & \{n_q\}_{k(q)} \end{array}$$

- Some basic data types :

Prin :	principals
PrinSet :	set of principals
Rand :	random numbers
Nonce :	nonce
NonceSet :	set of nonces
Cipher :	cipher text
Msg :	messages
MsgSet :	set of message

NSPK (2) : OTS $\mathcal{S}_{\mathsf{NSPK}}$.

$$\mathcal{S}_{\mathsf{NSPK}} = <\mathcal{O}_{\mathsf{NSPK}}, \mathcal{I}_{\mathsf{NSPK}}, \mathcal{T}_{\mathsf{NSPK}}>$$

- $\mathcal{O}_{\mathsf{NSPK}}$:
  $\{\mathsf{rand} : \Upsilon \rightarrow \mathsf{Rand}, \mathsf{nw} : \Upsilon \rightarrow \mathsf{MsgSet}, \mathsf{nonces} : \Upsilon \rightarrow \mathsf{NonceSet}\}$
- $\mathcal{I}_{\mathsf{NSPK}}$:
  $\{init \in \Upsilon |$
  $\mathsf{rand}(init) = \mathsf{seed} \wedge \mathsf{nw}(init) = \mathsf{empty} \wedge \mathsf{nonces}(init) = \mathsf{empty}\}$
- $\mathcal{T}_{\mathsf{NSPK}}$:
  $\{\mathsf{send1}_{p:\mathsf{Prin},q:\mathsf{Prin}} : \Upsilon \rightarrow \Upsilon,$
  $\mathsf{send2}_{p:\mathsf{Prin},q:\mathsf{Prin},r:\mathsf{Prin},n:\mathsf{Nonce}} : \Upsilon \rightarrow \Upsilon,$
  $\mathsf{send3}_{p:\mathsf{Prin},q:\mathsf{Prin},r:\mathsf{Prin},n1:\mathsf{Nonce},n2:\mathsf{Nonce}} : \Upsilon \rightarrow \Upsilon,$
  $\mathsf{fake1}_{p:\mathsf{Prin},q:\mathsf{Prin},n:\mathsf{Nonce}} : \Upsilon \rightarrow \Upsilon,$
  $\mathsf{fake2}_{p:\mathsf{Prin},q:\mathsf{Prin},n1:\mathsf{Nonce},n2:\mathsf{Nonce}} : \Upsilon \rightarrow \Upsilon,$
  $\mathsf{fake3}_{p:\mathsf{Prin},q:\mathsf{Prin},n:\mathsf{Nonce}} : \Upsilon \rightarrow \Upsilon,$
  $\mathsf{fake4}_{p:\mathsf{Prin},q:\mathsf{Prin},m:\mathsf{Msg}} : \Upsilon \rightarrow \Upsilon\}$

NSPK (3) : Specification of $\mathcal{S}_{\mathtt{NSPK}}$ in CafeOBJ

Declarations of observations and transitions symbols

```
--- observers
bop rand   : Sys -> Rand .
bop nw     : Sys -> MsgSet .
bop nonces : Sys -> NonceSet .
--- actions
bop send1 : Sys Prin Prin -> Sys .
bop send2 : Sys Prin Prin Prin Nonce -> Sys .
bop send3 : Sys Prin Prin Prin Nonce Nonce -> Sys .
bop fake1 : Sys Prin Prin Nonce -> Sys .
bop fake2 : Sys Prin Prin Nonce Nonce -> Sys .
bop fake3 : Sys Prin Prin Nonce -> Sys .
bop fake4 : Sys Prin Prin Msg -> Sys .
```

NSPK (4) : Translated specification of $\mathcal{S}_{\mathtt{NSPK}}$ in Maude

Corresponding observations in Maude obtained by the rule (1)

```
--- observations
op rand:_   : Rand -> State .
op nw:_     : MsgSet -> State .
op nonces:_ : NonceSet -> State .
```

NSPK (5) : Transition *send2* in CafeOBJ

Definition of the transition *send2* :

```
op c-send2 : Sys Prin Prin Prin Nonce -> Bool .
eq c-send2(S,P1,P2,P3,N) =
   m(P3,P2,P1,enc1(P1,N,P2)) \in nw(S) .
---
ceq rand(send2(S,P1,P2,P3,N)) = next(rand(S))
    if c-send2(S,P1,P2,P3,N) .
ceq nw(send2(S,P1,P2,P3,N))=
    m(P1,P1,P2,enc2(P2,N,n(P1,P2,rand(S)))) nw(S)
    if c-send2(S,P1,P2,P3,N) .
ceq nonces(send2(S,P1,P2,P3,N))  =
    if P2 eqs intr then
       (N n(P1,P2,rand(S)) nonces(S))
    else nonces(S) fi
if c-send2(S,P1,P2,P3,N) .
---
ceq send2(S,P1,P2,P3,N) = S if not c-send2(S,P1,P2,P3,N) .
```

NSPK (6) : Transition of *send2* CafeOBJ from Maude

rand for the definition of *send2* :

```
ceq rand(send2(S,P1,P2,P3,N)) =
    next(rand(S)) if c-send2(S,P1,P2,P3,N) .
```

```
crl send2(P1,P2,P3,N) (rand : R) =>
    send2(P1,P2,P3,N) (rand : next(R))
    if c-send2(S,P1,P2,P3,N) .
```

by rule (2)

nw for the definition of *send2* :

```
ceq nw(send2(S,P1,P2,P3,N))=
    m(P1,P1,P2,enc2(P2,N,n(P1,P2,rand(S)))) nw(S)
    if c-send2(S,P1,P2,P3,N) .
```

---

(2)

```
crl send2(P1,P2,P3,N) (nw : NW) =>
    send2(P1,P2,P3,N) (nw : m(P1,P1,P2,enc2(P2,N,n(P1,P2,rand(S)))) NW)
    if c-send2(S,P1,P2,P3,N) .
```

NSPK (7) : Transition of *send2* CafeOBJ from Maude

nonces for the definition of *send2* :

```
ceq nonces(send2(S,P1,P2,P3,N)) =
    if P2 eqs intr then (N n(P1,P2,rand(S)) nonces(S)) else nonces(S) fi
   if c-send2(S,P1,P2,P3,N) .
```

(2)

```
  crl send2(P1,P2,P3,N) (nonces : NS) =>
      send2(P1,P2,P3,N)
      (nonces : if P2 eqs intr then (N n(P1,P2,rand(S)) NS) else NS fi)
      if c-send2(S,P1,P2,P3,N) .
```

Combination of Translated rules :

(3)

```
 crl send2(P1,P2,P3,N) (rand : R) (nw : NW) (nonces : NS) =>
     send2(P1,P2,P3,N) (rand : next(R))
     (nw : m(P1,P1,P2,enc2(P2,N,n(P1,P2,rand(S)))) NW)
     (nonces : if P2 eqs intr then (N n(P1,P2,rand(S)) NS) else NS fi)
     if c-send2(S,P1,P2,P3,N) .
```

NSPK (8) : Transition of *send2* CafeOBJ from Maude

Effective condition *send2* :

$$\frac{\texttt{eq c-send2(S,P1,P2,P3,N) = m(P3,P2,P1,enc1(P1,N,P2)) \textbackslash in nw(S) .}}{\begin{array}{l}\texttt{c-send2'(P1,P2,P3,N,NS) = c-send2(S,P1,P2,P3,N)[nw(S)} \leftarrow \texttt{NW]} \\ \qquad\qquad\qquad\qquad\qquad\qquad \texttt{= m(P3,P2,P1,enc1(P1,N,P2)) \textbackslash in NW}\end{array}} \quad (4)$$

Add condition

$$\frac{}{\begin{array}{l}\texttt{crl send2(P1,P2,P3,N) (rand : R) (nw : NW) (nonces : NS) =>} \\ \quad \texttt{send2(P1,P2,P3,N) (rand : next(R))} \\ \quad \texttt{(nw : m(P1,P1,P2,enc2(P2,N,n(P1,P2,rand(S)))) NW)} \\ \quad \texttt{(nonces : if P2 eqs intr then (N n(P1,P2,rand(S)) NS) else NS fi)} \\ \quad \texttt{if m(P3,P2,P1,enc1(P1,N,P2)) \textbackslash in NW .}\end{array}} \quad (5)$$

NSPK (9) : Optimization of the *send2* rule

Optimization :

```
crl send2(P1,P2,P3,N) (rand : R) (nw : NW) (nonces : NS) =>
    send2(P1,P2,P3,N) (rand : next(R))
    (nw : m(P1,P1,P2,enc2(P2,N,n(P1,P2,rand(S)))) NW)
    (nonces : if P2 eqs intr then (N n(P1,P2,rand(S)) NS) else NS fi)
    if m(P3,P2,P1,enc1(P1,N,P2)) \in NW .
```

```
rl send2(P1,P2,P3,N) (rand : R)
                     (nw : (m(P3,P2,P1,enc1(P1,N,P2)) NW)) (nonces : NS) =>
   send2(P1,P2,P3,N) (rand : next(R))
                     (nw : m(P1,P1,P2,enc2(P2,N,n(P1,P2,rand(S))))
                           m(P3,P2,P1,enc1(P1,N,P2)) NW)
    (nonces : if P2 eqs intr then (N n(P1,P2,rand(S)) NS) else NS fi) .
```

NSPK (10) : Build initial state in Maude

Assume that there are three principals (p, q, intr).

Initial state

$$\frac{\begin{array}{l}\texttt{eq rand(init) = seed .}\\ \texttt{eq nw(init) = empty .}\\ \texttt{eq nonces(init) = empty .}\end{array}}{\texttt{eq init = (rand : seed)(nw : empty)(nonces : empty) .}}\ (7)$$

Actually

```
eq init = (rand : seed)(nw : empty)(nonces : empty)
          (OPrin1 :(p q intr)) (OPrin1 :(p q intr)) .
```

A translator :

- How to run :
  1. Start Maude ;
  2. Load `Full Maude` with command `load full-maude` ;
  3. Load the translator with command `load ots-maude`.
- Input :
  1. Modules should be enclosed by parentheses (...) ;
  2. Statements in modules should be ended with "."
- Command :
  1. `(conv2maude .)` ;
  2. `(show module MODULE-NAME .)`

Some restrictions

1. Cannot support parameterised module ;
2. S_{oi} should correspond to a finite or bounded set ;
3. Only one sub-state among those observed by the same observer can be changed w.r.t. a transition ;

```
rl (o[I] : A)(o[J] :B)=>(o[I] : A')(o[J] :B')
```

4. RHSs of equations for transitions should not contain transition symbols ;
5. RHSs of equations for initial states should not contain transition symbols and observation symbols.

The End

The translator is available at

`www.jaist.ac.jp/~s0820005/ots-maude.tar.gz`

`Welcome any bug reports !`

`zhangmin@jaist.ac.jp`