Implementing the APA model for the symmetric Needham-Schroeder protocol in state transition pattern notation in the SH Verification Tool

Roland Rieke
SIT – Fraunhofer - Institute for Secure Telecooperation,
Rheinstr. 75, D-64295 Darmstadt, Germany
E-Mail: rieke@sit.fraunhofer.de

July 26, 2002

The protocol

1. $A \longrightarrow S: \quad A, B, R_A$
2. $S \longrightarrow A: \quad \{R_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \longrightarrow B: \quad \{K_{AB}, A\}_{K_{BS}}$
4. $B \longrightarrow A: \quad \{R_B\}_{K_{AB}}$
5. $A \longrightarrow B: \quad \{R_B - 1\}_{K_{AB}}$

An APA model for the symmetric Needham-Schroeder protocol and a detailed explanation of that model is given in [SOR02]. Here a translation of a sligthly modified version of that model into the syntax of the SH verification tool is shown.

The reachability graph for a simple protocol run computed by the tool is appended.

As a basis for the implementation the APA model as specified in the appendix of [SOR02] is used.

Corresponding syntax of the APA model and SH verification tool [Fra02b, Fra02a, ORR00] are shown side by side.

The complete specification in SH verification tool syntax is included in the appendix.

A short tour of SH Verification Tool syntax for APA state transition pattern notation:

Some examples used in the Needham-Schroeder model and their SH verification tool counterpart:

$(start, B) \in \mathrm{State_A}$		`['start',B] ? A_State`
$(start, B) \hookleftarrow \mathrm{State_A}$	*tool syntax:*	`['start',B] << A_State`
$(B, R_A, S) \hookrightarrow \mathrm{State_A}$		`[B,RA,S] >> A_State`

Example 1: A simple preamble for the SH verification tool and the computed reachability graph:

A global state `multiset` is defined and initialised:

```
defset nat0_seq = seq(nat_0);
def_state multiset: nat0_seq := 2.3.3;
```

A role `role` is defined and bound to `'instance'`:

```
def_role role;
def_pattern_bind role :=  'instance' ;
```

A transition pattern for that role just takes out one element from the global state `multiset`:

```
def_trans_pattern role pattern
  (x)
  x << multiset;
```

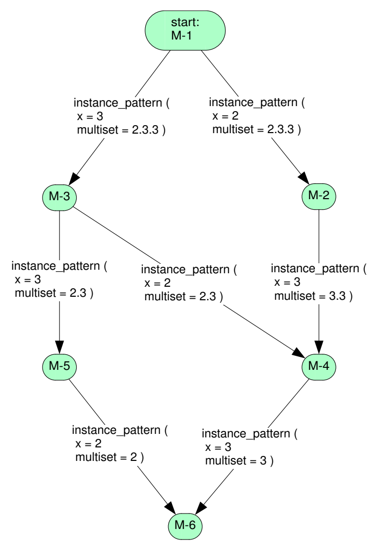

Figure 1: Reachability graph of example 1

Example 2: An example using a global state `network` and two instances of different roles one putting some information from local state `role1_my_set` into global state `network` and the other taking the data out of `network` into the local state `role2_my_set`.

```
defset nat0_seq = seq(nat_0);
def_state network: nat0_seq := ::;

def_role role1;
def_role role2;
def_pattern_bind role1 :=  'instance1' ;
def_pattern_bind role2 :=  'instance2' ;

def_state role1 my_set: nat0_seq := 2.3.3;
def_state role2 my_set: nat0_seq := ::;

def_trans_pattern role1 pattern1
  (x)
  x << role1_my_set,
  network = ::, /* check whether network is empty */
  x >> network;

def_trans_pattern role2 pattern1
  (y)
  y << network,
  y >> role2_my_set;
```

Figure 2 shows the reachability graph computed by the SH Verification Tool for this example.

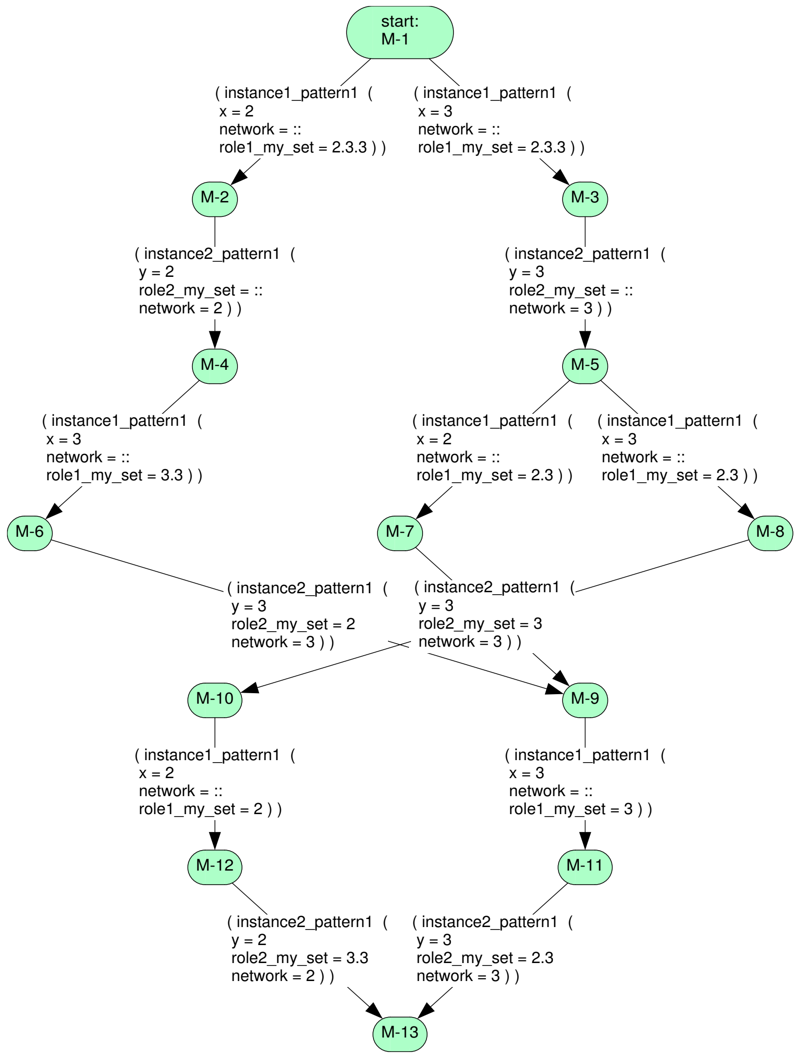

Figure 2: Reachability graph of example 2

Now the corresponding syntax of the APA model in [SOR02] and SH Verification Tool [Fra02b, Fra02a, ORR00] are shown side by side.

APA model (initial state):

$$\begin{array}{lcl}
\text{State}_{\text{A}} & := & \{(\mathbf{B}, \textit{agent}), (\mathbf{S}, \textit{server}), (\textit{start}, \mathbf{B})\} \\
\text{Symkeys}_{\text{A}} & := & \{(\mathbf{S}, sym, (\mathbf{A}, \mathbf{S}, sym))\} \\
\text{Asymkeys}_{\text{A}} & := & \emptyset \\
 & & \\
\text{State}_{\text{B}} & := & \{(\mathbf{A}, \textit{agent}), (\mathbf{S}, \textit{server}), (\textit{respond}, \mathbf{A})\} \\
\text{Symkeys}_{\text{B}} & := & \{(\mathbf{S}, sym, (\mathbf{B}, \mathbf{S}, sym))\} \\
\text{Asymkeys}_{\text{B}} & := & \emptyset \\
 & & \\
\text{State}_{\text{S}} & := & \{(\mathbf{A}, \textit{agent}), (\mathbf{B}, \textit{agent})\} \\
\text{Symkeys}_{\text{S}} & := & \{(\mathbf{A}, sym, (\mathbf{A}, \mathbf{S}, sym)), (\mathbf{B}, sym, (\mathbf{B}, \mathbf{S}, sym))\} \\
\text{Asymkeys}_{\text{S}} & := & \emptyset \\
 & & \\
\text{Network} & := & \emptyset
\end{array}$$

SH Verification Tool:

```
def_role A;
def_role B;
def_role S;

def_state A State: Messages_seq := [B,'agent'].[S,'server'].['start',B];
def_state A Symkeys: Symkeys_seq := (S,'sym',(A,S,'sym'));
def_state A Asymkeys: Asymkeys_seq := ::;

def_state B State: Messages_seq := [A,'agent'].[S,'server'].['respond',A];
def_state B Symkeys: Symkeys_seq := (S,'sym',(B,S,'sym'));
def_state B Asymkeys: Asymkeys_seq := ::;

def_state S State: Messages_seq := [A,'agent'].[B,'agent'];
def_state S Symkeys: Symkeys_seq := (A,'sym',(A,S,'sym')).
                                  (B,'sym',(B,S,'sym'));
def_state S Asymkeys: Asymkeys_seq := ::;

def_state Network: net_elem_seq := ::;

def_pattern_bind A := 'Alice';
def_pattern_bind B := 'Bob';
def_pattern_bind S := 'Server';
```

APA model (step 1):

Variables: R_A, B, S

$$
\begin{array}{lll}
(start, B) \in \mathrm{State_A} \\
(B, \textit{agent}) \in \mathrm{State_A} \\
(S, \textit{server}) \in \mathrm{State_A} \\
\xrightarrow{\mathrm{A}} \\
R_A \in \textit{new\_nonce} \\
(start, B) & \hookleftarrow & \mathrm{State_A} \\
(B, R_A, S) & \hookrightarrow & \mathrm{State_A} \\
(A, S, (A, B, R_A)) & \hookrightarrow & \mathrm{Network}
\end{array}
$$

SH Verification Tool:

```
def_trans_pattern A step_1
  (RA,B,S)
  ['start',B] ? A_State,
  [B,'agent'] ? A_State,
  [S,'server'] ? A_State,
  RA << new_nonce,
  tail(RA) >> new_nonce,
  ['start',B] << A_State,
  [B,head(RA),S] >> A_State,
  (A,S,[A,B,head(RA)]) >> Network;
```

For the example `Nonces` are modeled in the tool by a list of `Nonce`. The state `new_nonce` initially contains a multiset (sequence) of one list of `Nonces`. Here we use only two nonces 13 and 17 needed for one run of the protocol in step 1 above and step 4.

```
defset Nonce =  13, 17, 19, 23;
defset Nonces = [Nonce];
defset Nonce_seq = seq(Nonces);
def_state new_nonce: Nonce_seq := [13,17];
```

In the pattern above `RA` is bound to the list `[13,17]`, the single element of the multiset `new_nonce`, then the tail of `RA` (in this case `[17]`) is put back into the multiset and the head of `RA` (in this case `13`) is used in the further lines of the above pattern.

In a concrete implementation a pseudo-random number generator should be used to generate nonces.

APA model (step 2):

Variables: $X, A, B, M, K_{AS}, K_{BS}, K_{AB}, R_A$

$(X, S, M) \in \text{Network}$
$(A, \textit{agent}) \in \text{State}_\text{S} \wedge elem(1, M) = A$
$(B, \textit{agent}) \in \text{State}_\text{S} \wedge elem(2, M) = B$
$(A, sym, K_{AS}) \in Symkeys_S$
$(B, sym, K_{BS}) \in Symkeys_S$
$\xrightarrow{\text{S}}$
$R_A := elem(3, M)$
$(A, B, sym) \in \textit{new\_key}$
$K_{AB} := (A, B, sym)$
$(X, S, M) \hookleftarrow \text{Network}$
$(S, A, (encrypt(K_{AS}, (R_A, B, K_{AB}, encrypt(K_{BS}, (K_{AB}, A)))))) \hookrightarrow \text{Network}$

SH Verification Tool:

```
def_trans_pattern S step_2
  (X,A,B,M,KAS,KBS)
  (X,S,M) ? Network,
  [A,'agent'] ? S_State,
  A = head(M),
  [B,'agent'] ? S_State,
  B = head(tail(M)),
  (A,'sym',KAS) ? S_Symkeys,
  (B,'sym',KBS) ? S_Symkeys,
/*  RA = head(tail(tail(M))),*/
  (X,S,M) << Network,
  (S,A,encrypt(KAS,[head(tail(tail(M))),B,(A,B,'sym'),
   encrypt(KBS,[(A,B,'sym'),A])])) >> Network;
```

In the tool the abbreviation `RA` used in the pattern above is currently not used. Instead *elem(3,M)* is inserted directly notated here by `head(tail(tail(M)))`.

APA model (step 3):

Variables: $X, B, S, M, \textit{Chiffretext}, K_{AS}, K_{AB}, R_A$

$$
\begin{array}{lll}
(X, A, M) \in \text{Network} & & \\
(B, R_A, S) \in \text{State}_\text{A} & & \\
(S, sym, K_{AS}) \in \text{Symkeys}_\text{A} & & \\
elem(1, decrypt(K_{AS}, M)) = R_A & & \\
elem(2, decrypt(K_{AS}, M)) = B & & \\
\xrightarrow{\text{A}} & & \\
K_{AB} := elem(3, decrypt(K_{AS}, M)) & & \\
\textit{Chiffretext} := elem(4, decrypt(K_{AS}, M)) & & \\
(X, A, M) & \hookleftarrow & \text{Network} \\
(B, R_A, S) & \hookleftarrow & \text{State}_\text{A} \\
(\textit{new session key}, B, K_{AB}) & \hookrightarrow & \text{State}_\text{A} \\
(A, B, \textit{Chiffretext}) & \hookrightarrow & \text{Network}
\end{array}
$$

SH Verification Tool:

```
def_trans_pattern A step_3
  (X,B,S,M,KAS,RA)
  (X,A,M) ? Network,
/*  [B,RA,S] ? A_State,*/
  (S,'sym',KAS) ? A_Symkeys,
  head(decrypt(KAS,M)) = RA,
  head(tail(decrypt(KAS,M))) = B,
  [B,RA,S] ? A_State,
/*  KAB = head(tail(tail(decrypt(KAS,M)))),*/
/*  Chiffretext = head(tail(tail(tail(decrypt(KAS,M))))),*/
  (X,A,M) << Network,
  [B,RA,S] << A_State,
  [new_session_key,B,head(tail(tail(decrypt(KAS,M))))] >> A_State,
  (A,B,head(tail(tail(tail(decrypt(KAS,M)))))) >> Network;
```

In the tool the abbreviations `KAB` and `Chiffretext` used in the pattern above are currently not used and the corresponding expression shown in the commented lines is inserted directly.

APA model (step 4):

Variables: $X, A, S, M, K_{BS}, K_{AB}, R_B$

$(X, B, M) \in \text{Network}$
$(\textit{respond}, A) \in \text{State}_\text{B}$
$(A, \textit{agent}) \in \text{State}_\text{B}$
$(S, \textit{server}) \in \text{State}_\text{B}$
$(S, sym, K_{BS}) \in \text{Symkeys}_\text{B}$
$elem(2, decrypt(K_{BS}, M)) = A$
$\xrightarrow{\text{B}}$
$K_{AB} := elem(1, decrypt(K_{BS}, M))$
$R_B \in \textit{new\_nonce}$
$(X, B, M) \hookleftarrow \text{Network}$
$(\textit{respond}, A) \hookleftarrow \text{State}_\text{B}$
$(\textit{new session key}, A, K_{AB}, R_B) \hookrightarrow \text{State}_\text{B}$
$(B, A, (encrypt(K_{AB}, R_B))) \hookrightarrow \text{Network}$

SH Verification Tool:

```
def_trans_pattern B step_4
  (X,A,S,M,KBS,RB)
  (X,B,M) ? Network,
  ['respond',A] ? B_State,
  [A,'agent'] ? B_State,
  [S,'server'] ? B_State,
  (S,'sym',KBS) ? B_Symkeys,
  head(tail(decrypt(KBS,M))) = A,
/*  KAB = head(decrypt(KBS,M)),*/
/*  RB ? new_nonce,*/
  RB << new_nonce,
  (X,B,M) << Network,
  ['respond',A] << B_State,
  [new_session_key,A,head(decrypt(KBS,M)),RB] >> B_State,
  (B,A,encrypt(head(decrypt(KBS,M)),[RB])) >> Network;
```

The usage chosen for nonces in the tool notation is described in step 1. In the pattern above `RA` is bound to the list `[17]`, the single element of the multiset `new_nonce`, then the tail of `RA` (in this case the empty list) is put back into the multiset and the head of `RA` (in this case `17`) is used in the further lines of the above pattern.

APA model (step 5a):

Variables: X, B, M, K_{AB}, R_B

$(X, A, M) \in$ Network
$($*new session key*$, B, K_{AB}) \in \mathrm{State_A}$
$\xrightarrow{\mathbf{A}}$
$R_B := elem(1, decrypt(K_{AB}, M))$

(X, A, M)	$\hookleftarrow$	Network
$($*new session key*$, B, K_{AB})$	$\hookleftarrow$	$\mathrm{State_A}$
$($*session key*$, B, K_{AB})$	$\hookrightarrow$	$\mathrm{State_A}$
(B, sym, K_{AB})	$\hookrightarrow$	$\mathrm{Symkeys_A}$
$(A, B, (encrypt(K_{AB}, (R_B - 1))))$	$\hookrightarrow$	Network

SH Verification Tool:

```
def_trans_pattern A step_5a
  (X,B,M,KAB)
  (X,A,M) ? Network,
  ['new_session_key',B,KAB] ? A_State,
/*  RB = head(decrypt(KAB,M)),*/
  (X,A,M) << Network,
  ['new_session_key',B,KAB] << A_State,
  ['session_key',B,KAB] >> A_State,
  (B,'sym',KAB) >> A_Symkeys,
  (A,B,encrypt(KAB,[head(decrypt(KAB,M))-1])) >> Network;
```

APA model (step 5b):

Variables: X, A, M, K_{AB}, R_B

$(X, B, M) \in$ Network
$(\textit{new session key}, A, K_{AB}, R_B) \in \text{State}_\text{B}$
$elem(1, decrypt(K_{AB}, M)) + 1 = R_B$
$\xrightarrow{\text{B}}$

(X, B, M)	$\hookleftarrow$	Network
$(\textit{new session key}, A, K_{AB}, R_B)$	$\hookleftarrow$	State_B
$(\textit{session key}, A, K_{AB})$	$\hookrightarrow$	State_B
(A, sym, K_{AB})	$\hookrightarrow$	Symkeys_B

SH Verification Tool:

```
def_trans_pattern B step_5b
  (X,A,M,KAB,RB)
  (X,B,M) ? Network,
  ['new_session_key',A,KAB,RB] ? B_State,
  head(decrypt(KAB,M))+1 = RB,
  (X,B,M) << Network,
  ['new_session_key',A,KAB,RB] << B_State,
  ['session_key',A,KAB] >> B_State,
  (A,'sym',KAB) >> B_Symkeys;
```

APA model (newA):

Variables: B, K_{AB}

$$(\textit{session key}, B, K_{AB}) \in \mathrm{State_A}$$
$$\xrightarrow{\mathbf{A}}$$
$$(\textit{start}, B) \hookrightarrow \mathrm{State_A}$$

SH Verification Tool:

```
def_trans_pattern A newA
  (B,KAB)
  ['session_key',B,KAB] ? A_State,
  ['start',B] ~? A_State,
  ['start',B] >> A_State;
```

In the tool notation it is checked whether the element `['start',B]` is already in `A_State` to prevent an endless loop here.

APA model (newB):

Variables: A, K_{AB}

$$(\textit{session key}, A, K_{AB}) \in \mathrm{State_B}$$
$$\xrightarrow{\mathbf{B}}$$
$$(\textit{respond}, A) \hookrightarrow \mathrm{State_B}$$

SH Verification Tool:

```
def_trans_pattern B newB
  (A,KAB)
  ['session_key',A,KAB] ? B_State,
  ['respond',A] ~? B_State,
  ['respond',A] >> B_State;
```

In the tool notation it is checked whether the element `['respond',A]` is already in `B_State` to prevent an endless loop here.

SH Verification Tool: Reachability graph

The reachability graph computed by the SH Verification Tool for this configuration is shown in figures 3 and 4.

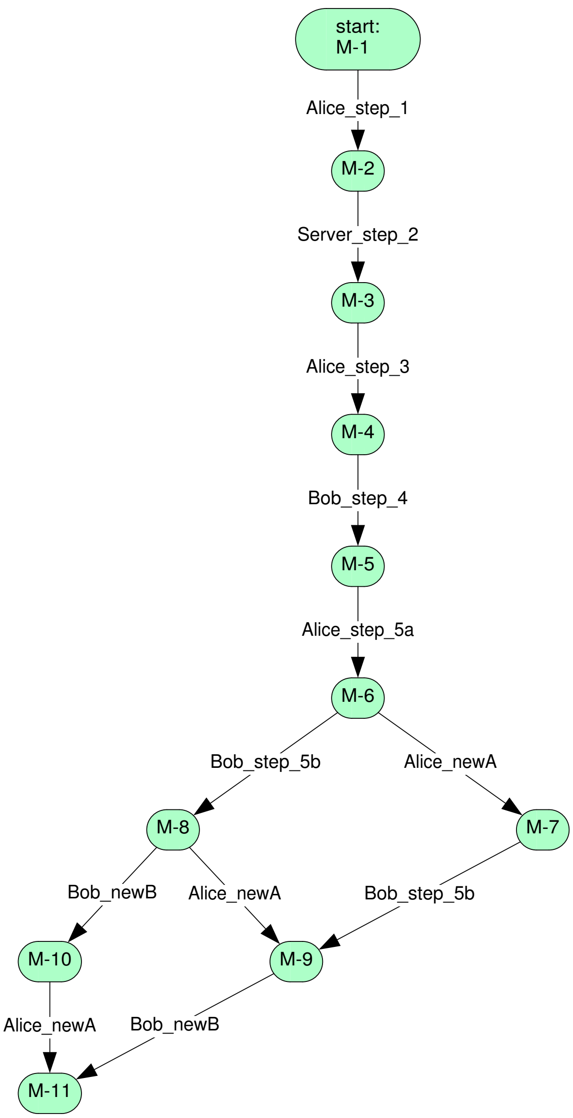

Figure 3: Reachability graph

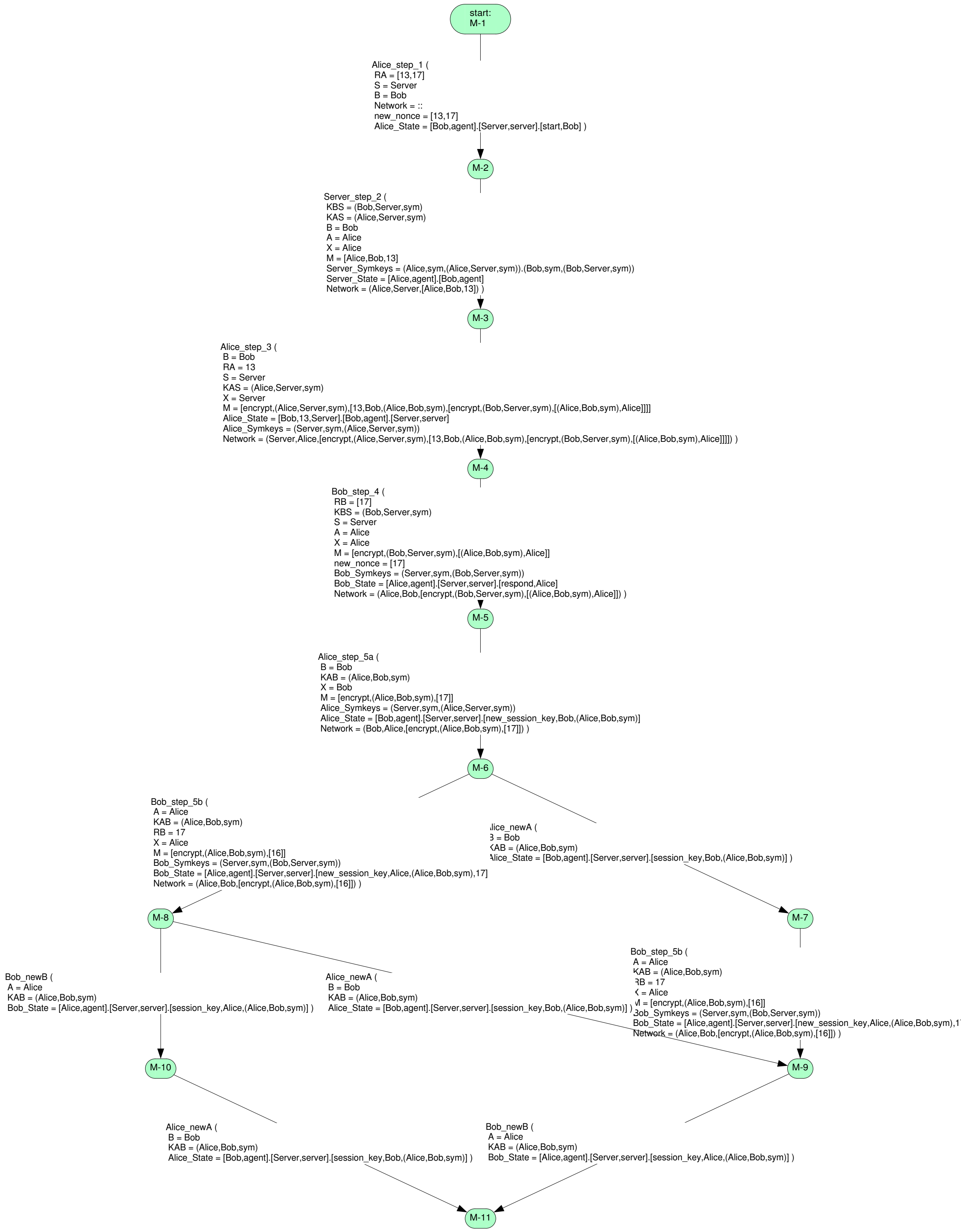

Figure 4: Reachability graph with interpretation

References

[Fra02a] Fraunhofer Institute for Secure Telecooperation SIT, Darmstadt. *Simple Homomorphism Verification Tool – Manual*, 2002.

[Fra02b] Fraunhofer Institute for Secure Telecooperation SIT, Darmstadt. *Simple Homomorphism Verification Tool – Tutorial*, 2002.

[ORR00] P. Ochsenschläger, J. Repp, and R. Rieke. The SH-Verification Tool. In *Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000)*, pages 18–22, Orlando, FL, USA, May 2000. AAAI Press.

[SOR02] Gürgens S., P. Ochsenschläger, and Carsten Rudolph. Role based specification and analysis with APA. GMD Report 151, Fraunhofer Institute for Secure Telecooperation SIT, 2002.

Appendix

SH Verification Tool:

```
defset Agents =  'Alice', 'Alice2', 'Bob', 'Server' ;

defset Address = Agents;

defset Identity = Agents || nat_0;

defset Nonce =  13, 17, 19, 23;

defset Nonces = [Nonce];

defset Nonce_seq = seq(Nonces);

defset Run_seq = seq(nat_0);

defset Constants = 'start', 'respond', 'agent', 'server',
                    'new_session_key', 'session_key';

defset Symflags = 'sym', 'symmac';

defset Asymflags = 'pub', 'priv', 'pucipher', 'privcipher',
                    'pubverify', 'privsign';

defset Cryptfuncs = 'encrypt', 'decrypt', 'crypt', 'sign', 'mac', 'hash';

defset Keywords = Symflags || Asymflags || Constants || Cryptfuncs;

defset Skeys = pro(Agents, Agents, Symflags);

defset Akeys = pro(Agents, Asymflags);

defset Keys = Skeys || Akeys;

defset message = Keywords || Agents || Nonce || Keys || nat_0;
/* nat_0 wg. nonce -1 */

defset Messages = [message];

defset Messages_seq = seq(Messages);

defset Symkeys = pro(Identity, Symflags, Keys);

defset Asymkeys = pro(Identity, Asymflags, Keys);

defset Symkeys_seq = seq(Symkeys);

defset Asymkeys_seq = seq(Asymkeys);

defset net_elem = pro(Address, Address, Messages);

defset net_elem_seq = seq(net_elem);

defset nat0_seq = seq(nat_0);

def_state Network: net_elem_seq := ::;
```

```
def_state new_nonce: Nonce_seq := [13,17];
/*
def_state new_nonce: Nonce_seq := [13,17,19,23];
*/
/*
def_state new_runs: Run_seq := 1;
*/
def_role A;

def_role B;

def_role S;

def_state A State: Messages_seq := [B,'agent'].[S,'server'].['start',B];

def_state A Symkeys: Symkeys_seq := (S,'sym',(A,S,'sym'));

def_state A Asymkeys: Asymkeys_seq := ::;

def_state B State: Messages_seq := [A,'agent'].[S,'server'].['respond',A];

def_state B Symkeys: Symkeys_seq := (S,'sym',(B,S,'sym'));

def_state B Asymkeys: Asymkeys_seq := ::;

def_state S State: Messages_seq := [A,'agent'].[B,'agent'];

def_state S Symkeys: Symkeys_seq := (A,'sym',(A,S,'sym')).
                                    (B,'sym',(B,S,'sym'));

def_state S Asymkeys: Asymkeys_seq := ::;

/*
defcase encrypt: pro(Keys,Messages) >> Messages
  encrypt(key,message) = if head(message) = 'decrypt' &
                            head(tail(message)) = key
                         then head(tail(tail(message)))
                         else ['encrypt',key,message];
*/

defterm encrypt: pro(Keys,Messages) >> Messages
  encrypt(key,message) = ['encrypt',key,message];

defcase decrypt: pro(Keys,Messages) >> Messages
  decrypt(key,message) = if head(message) = 'encrypt' &
                            head(tail(message)) = key
                         then head(tail(tail(message)))
                         else ['decrypt',key,message];


def_trans_pattern A step_1
  (RA,B,S)
  ['start',B] ? A_State,
  [B,'agent'] ? A_State,
  [S,'server'] ? A_State,
  RA << new_nonce,
  tail(RA) >> new_nonce,
  ['start',B] << A_State,
  [B,head(RA),S] >> A_State,
  (A,S,[A,B,head(RA)]) >> Network;
```

```
def_trans_pattern S step_2
  (X,A,B,M,KAS,KBS)
  (X,S,M) ? Network,
  [A,'agent'] ? S_State,
  A = head(M),
  [B,'agent'] ? S_State,
  B = head(tail(M)),
  (A,'sym',KAS) ? S_Symkeys,
  (B,'sym',KBS) ? S_Symkeys,
/*  RA = head(tail(tail(M))),*/
  (X,S,M) << Network,
  (S,A,encrypt(KAS,[head(tail(tail(M))),B,(A,B,'sym'),
   encrypt(KBS,[(A,B,'sym'),A])])) >> Network;

def_trans_pattern A step_3
  (X,B,S,M,KAS,RA)
  (X,A,M) ? Network,
/*  [B,RA,S] ? A_State,*/
  (S,'sym',KAS) ? A_Symkeys,
  head(decrypt(KAS,M)) = RA,
  head(tail(decrypt(KAS,M))) = B,
  [B,RA,S] ? A_State,
/*  KAB = head(tail(tail(decrypt(KAS,M)))),*/
/*  Chiffretext = head(tail(tail(tail(decrypt(KAS,M))))),*/
  (X,A,M) << Network,
  [B,RA,S] << A_State,
  [new_session_key,B,head(tail(tail(decrypt(KAS,M))))] >> A_State,
  (A,B,head(tail(tail(tail(decrypt(KAS,M)))))) >> Network;

def_trans_pattern B step_4
  (X,A,S,M,KBS,RB)
  (X,B,M) ? Network,
  ['respond',A] ? B_State,
  [A,'agent'] ? B_State,
  [S,'server'] ? B_State,
  (S,'sym',KBS) ? B_Symkeys,
  head(tail(decrypt(KBS,M))) = A,
/*  KAB = head(decrypt(KBS,M)),*/
/*  RB ? new_nonce,*/
  RB << new_nonce,
  tail(RB) >> new_nonce,
  (X,B,M) << Network,
  ['respond',A] << B_State,
  [new_session_key,A,head(decrypt(KBS,M)),head(RB)] >> B_State,
  (B,A,encrypt(head(decrypt(KBS,M)),[head(RB)])) >> Network;

def_trans_pattern A step_5a
  (X,B,M,KAB)
  (X,A,M) ? Network,
  ['new_session_key',B,KAB] ? A_State,
/*  RB = head(decrypt(KAB,M)),*/
  (X,A,M) << Network,
  ['new_session_key',B,KAB] << A_State,
  ['session_key',B,KAB] >> A_State,
  (B,'sym',KAB) >> A_Symkeys,
  (A,B,encrypt(KAB,[head(decrypt(KAB,M))-1])) >> Network;

def_trans_pattern B step_5b
  (X,A,M,KAB,RB)
  (X,B,M) ? Network,
  ['new_session_key',A,KAB,RB] ? B_State,
  head(decrypt(KAB,M))+1 = RB,
```

```
  (X,B,M) << Network,
  ['new_session_key',A,KAB,RB] << B_State,
  ['session_key',A,KAB] >> B_State,
  (A,'sym',KAB) >> B_Symkeys;

def_trans_pattern A newA
  (B,KAB)
  ['session_key',B,KAB] ? A_State,
  ['start',B] ~? A_State,
  ['start',B] >> A_State;

def_trans_pattern B newB
  (A,KAB)
  ['session_key',A,KAB] ? B_State,
  ['respond',A] ~? B_State,
  ['respond',A] >> B_State;


def_pattern_bind A :=  'Alice' ;

def_pattern_bind B :=  'Bob' ;
/*
def_pattern_bind B :=  'Alice2', 'Bob' ;
*/
def_pattern_bind S :=  'Server' ;
```