

Extended AnB for Web Services

Allan Asp Olsen

Kongens Lyngby 2012
IMM-M.Sc.-2012-63

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-M.Sc: ISSN 0909-3192

Summary

Formal protocol analysis is an important tool for computer security. While there has been a lot of formal mathematical analysis of cryptology the usual security problems in practice are the results of poorly applied cryptology or security problems in the programs themselves. For example some protocols have flaws that aren't found in several years after their publication. While one solution might be to just use well defined protocols this is not always possible. E.g. Sometimes new protocols need to be invented. While security protocols have been analysed for many years by hand it is often easy to miss mistakes in the design of the protocol.

This has led to the development of several tools for formal analysis of protocols. This includes Casper, Common Authentication Protocol Specification Language (CAPSL)[1] and Open-Source Fixed-Point Model-Checker (OFMC)[4][5]. The purpose of this thesis is to extend on the Alice and Bob notation (AnB) which is used for specifying protocols, these can be analysed with OFMC or similar tools. Classic AnB lacks support for problems that require a more persistent storage than a single protocol run. One such protocol is ASW[3] which allows users to digitally sign a protocol in a fair way (ie. one participant of the negotiation cannot have a signed contract without the other part acquiring one). This may require a trusted third party in the case that they disagree and this trusted third party has to act differently depending on the messages received so far.

To accomplish it's task OFMC translates from AnB to Automated Validation of Internet Security Protocols and Applications Intermediate Format(AVISPA IF). To extend AnB into the language which we refer to as Web Service Alice and Bob notation (WS-AnB) it was necessary to define a formal description of

the translation from these additions into AVISPA IF as well.

Acknowledgements

I thank my supervisor Sebastian Alexander Mödersheim, he has provided me with much help throughout the project.

Contents

Summary	i
Acknowledgements	iii
1 Classic AnB	1
1.1 Overview	1
1.2 Specification	1
2 Strands	3
2.1 Overview	3
3 AVISPA IF	7
3.1 Overview	7
3.2 Specification	7
4 WS-AnB	9
4.1 Motivation	9
4.2 Specification	9

CHAPTER 1

Classic AnB

1.1 Overview

Alice and Bob Notation (also known as AnB), is a language for specifying protocols in an easily readable way. It can then be used for analysis by computer programs.

1.2 Specification

An AnB specification starts with a protocol name, then a type specification listing (honest) agents, variables, and functions used in the protocol. Then the knowledge of each participating agent is specified followed by the concrete actions that makes up the protocol. Finally the specification ends with a set of security goals that need to be verified. It is of note that nothing in the AnB specification refers directly to the intruder.

1.2.1 Example

The following is an example of the Needham-Schroeder protocol in AnB.

Protocol: NSPK

Types: Agent A,B,s;
 Number Na,Nb;
 Function pk

Knowledge: A: A, B, s, pk(A), inv(pk(A)), pk(s);
 B: B, s, pk(B), inv(pk(B)), pk(s);
 s: A, B, s, pk, inv(pk(s))

Actions:

A->s: (A, B)
 s->A: {pk(B),B}inv(pk(s))
 A->B: {Na,A}pk(B)
 B->s: (B, A)
 s->B: {pk(A),A}inv(pk(s))
 B->A: {Na,Nb}pk(A)
 A->B: {Nb}pk(B)

Goals:

A *->* B: Na
 B *->* A: Nb

The Open-Source Fixed-Point Model-Checker(OFMC) can be used to analyse this protocol specification. The final part of the specification is the goal states. This also showcases an extra feature of the specification language. A star can be used to denote a secure part of a channel. For a message transmission a star at the end of the arrow indicates encryption with the public key(so the sender can guarantee only the receiver can read it) and a star at the beginning is an indication of public message signing(so the receiver can verify the identity of the sender). As goal states this is then a similar specification but rather than being a transmission it indicates that a message was transmitted during the run of the protocol which had properties of that channel. In the example this states that B will receive the message Na in such a way that only A could have send it and only B can read it. Conversely Nb is send such that only A can read it and only B could have send it.

Strands

2.1 Overview

The notion of a strand is closely tied with message sequence diagrams, they follow very easily from AnB. A message sequence diagram of NSPK can be seen in figure 2.1.

This is also a good figure for understanding strands. To show how this can be translated into a strand I will give the example for the agent B. See figure 2.2.

This shows an important difference from the earlier specifications, rather than A and NA the values are now replaced by X1 and X2, the reason for this is that the agent B has no way at the time to confirm the contents of these messages. For the purpose of this thesis this difference has been abstracted away.

This leaves us with a good visual representation of what a strand is. Later I will extend this notion from a list of messages send and received to a tree-like structure.

Figure 2.1: NSPK as message sequence diagram

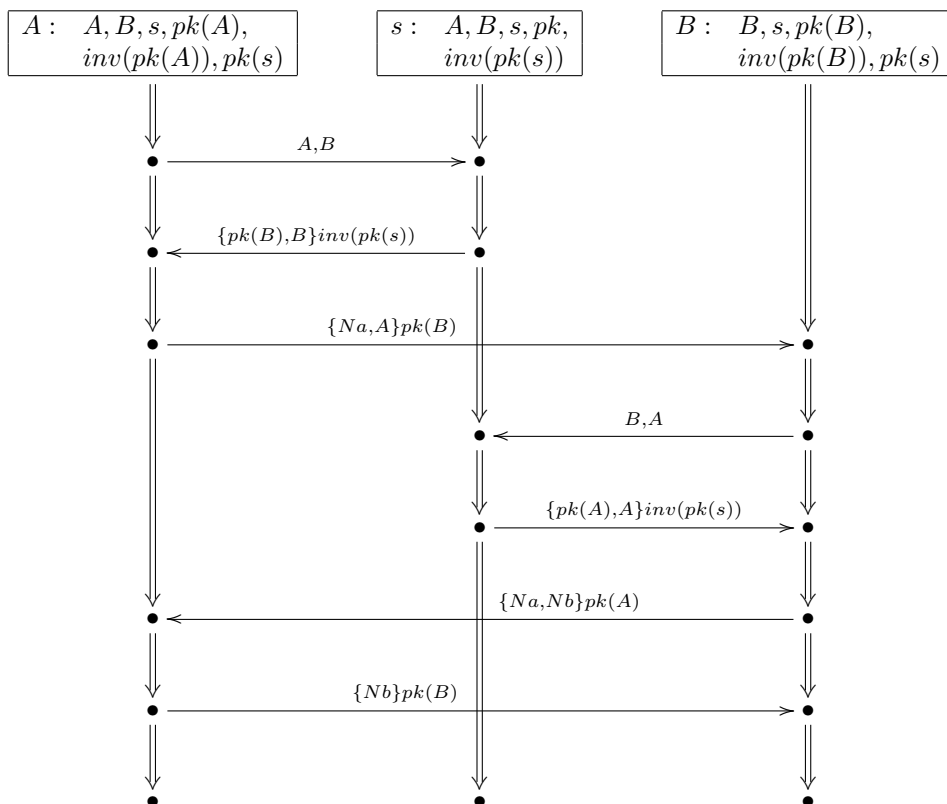
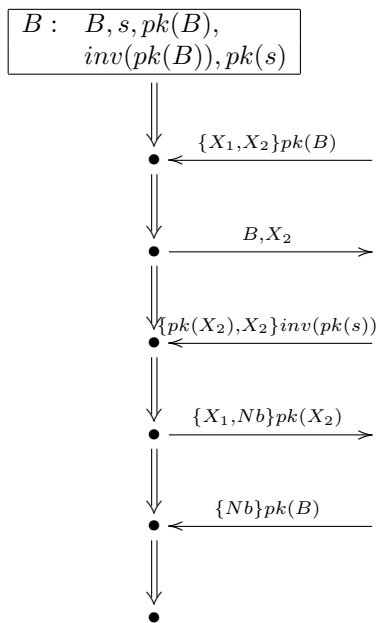


Figure 2.2: Strand of NSPK



AVISPA IF

3.1 Overview

the Automated Validation of Internet Security Protocols and Applications Intermediate Format (hereafter just referred to as IF or AVISPA IF) is a language used internally in OFMC as well as other tools for analysing protocols.

Rather than specifying messages that are sent or received AVISPA works on a lower level, specifying knowledge of the intruder and state changes of the agents. On one hand this gives access to a more expressive language than AnB on the other hand it is a lot harder to read and write.

3.2 Specification

A specification in IF starts with a signature section, this section typically specifies the internal state of the participating honest agents. This is followed by a section specifying the possible initial states of the protocol. Then the state changes are described for each agent. Finally a list of goals to be achieved is specified. IF states are sets of facts such as intruder knowledge or the state of an agent.

4.1 Motivation

Although regular AnB is very easy to read and understand it has some limitations, the goal of this thesis is to show how such an extension could be made to extend AnB in a conservative manor. One could of course skip AnB entirely and work directly on the IF level but as described in the AVISPA IF chapter this is rather complicated due to the low-level nature of IF.

One of the goals with this extension is to keep WS-AnB as simple as possible while making it expressive enough to cover what we need. Another important goal is to make it a conservative extension meaning that the original AnB is left intact(both semantically and syntactically).

4.2 Specification

The translation from WS-AnB to IF has two steps, first it is translated into agent specific strands and then from strands into IF. This choice was made for several reasons, going to strands first makes it easier to later incorporate a split

between the knowledge of the agents. It is also easier to understand the two minor steps than one big step. Lastly having the strands makes some procedures simpler, for instance finding out which variables are fresh is easier to understand when the translation to strands has completed.

For practical reasons (primarily performance issues) many steps are usually added together. This aggregation runs from one receive to the next. This can be done without changing the properties of the protocol(intuitively one can consider the agents to do such calculations locally so the intruder can't interfere)[2]. This is another transformation that works well on the strand level.

An IF state transition rule has 3 basic components, the left hand side is a set of facts that must hold for the state change to be possible. The right hand side is a set of facts that will hold after the transition and in the middle(as part of the arrow) there can be a set of new variables which must be introduced to state the right hand side. An important note is that any fact not mentioned at all will remain unchanged, if a fact is specified only on the left hand side it will be removed and if specified on only the right hand side it will be introduced. Lastly if specified on both sides it will remain.

4.2.1 Abstract syntax of WS-AnB

The basic grammar of WS-AnB actions are defined as:

$$\begin{aligned}
 S ::= & A \rightarrow B \\
 & | S_1; S_2 \\
 & | \text{if}(C) \text{ then } S_1 \text{ else } S_2 \text{ fi} \\
 & | ?M \\
 & | \sim?M \\
 & | + M \\
 & | - M \\
 & | A \rightarrow B : M \text{ or } S \\
 & | \epsilon
 \end{aligned}$$

Where ϵ is the empty string, M is a message on the classic AnB form:

$$\begin{aligned}
M ::= & \textit{identifier} \\
& | \{M_1\}M_2 \\
& | \{|M_1|\}M_2 \\
& | M_1, M_2 \\
& | \textit{identifier}'(\textit{identifier}, \textit{identifier})*' \\
& | ('M')'
\end{aligned}$$

Where '(' and ')' is used to denote the string (and) rather than the grammar rules.

C is a condition of the form:

$$\begin{aligned}
C ::= & ?M \\
& | M_1 = M_2 \\
& | \sim C
\end{aligned}$$

and A and B are agent names. One example of such a specification is ASW[3]:

Protocol: ASW

Types:

Agent O, R, t;
Number No, Nr;
Function h, contract;

Knowledge:

O: O, R, t, inv(pk(O)), pk(O), pk(R), pk(t), contract;
R: O, R, t, inv(pk(R)), pk(O), pk(R), pk(t), contract;
t: inv(pk(t)), pk(O), pk(R), pk(t), contract;
DB(t, pair(O, pk(O)))

Actions (Exchange):

O → R: {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O))
R → O: {{pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O))
, h(Nr)} inv(pk(R)) OR Abort
O → R: No OR ResolveO
+ std, {{pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)), h
(Nr)} inv(pk(R)), No, Nr

$R \rightarrow O$: Nr OR ResolveR
 + std, { {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)), h(Nr)} inv(pk(R)), No, Nr

Actions (Abort):

$O \rightarrow t$: {aborted, {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O))} inv(pk(O))
if(? resolved, {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)))
then
 $t \rightarrow O$: { {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)) },
 { {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)), h(Nr)} inv(pk(R)) } inv(pk(t))
else
 + aborted, {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O))
 $t \rightarrow O$: {aborted, {aborted, {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O))} inv(pk(O))} inv(pk(t))
fi

Actions (ResolveO):

$O \rightarrow t$: {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)),
 { {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)), h(Nr)} inv(pk(R))
if(? aborted, {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)))
then
 $t \rightarrow O$: {aborted, {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O))} inv(pk(t))
else
 + resolved, {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O))
 $t \rightarrow O$: { {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)) },
 { {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)), h(Nr)} inv(pk(R)) } inv(pk(t))
fi

Actions (ResolveR):

$R \rightarrow t$: {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)),
 { {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)), h(Nr)} inv(pk(R))
if(? aborted, {pk(O), pk(R), T, contract(O, R, T), h(No)} inv(pk(O)))

```

(O)))
then
  t -> R: {aborted , {pk(O) ,pk(R) ,T, contract (O,R,T) ,h(No)
              }inv (pk(O)) }inv (pk(t))
else
  + resolved , {pk(O) ,pk(R) ,T, contract (O,R,T) ,h(No) }inv (pk
    (O)))
  t -> R: { {pk(O) ,pk(R) ,T, contract (O,R,T) ,h(No) }inv (pk(O)
    ) ,
    { {pk(O) ,pk(R) ,T, contract (O,R,T) ,h(No) }inv (pk(O)) ,h(Nr) }
      inv (pk(R)) }inv (pk(t))
fi

```

4.2.2 Translation from WS-AnB to WS-Strands

The translation function from WS-AnB to WS-Strands is defined as the function $tr_\alpha(A, B, AnB)$ where A is the agent for which the translation is running (basically the function is called once with each participating agent). B is the currently active agent in the sense that only this agent can send messages (this should be initially called as the first agent to act since otherwise it will cause errors). And AnB is the WS-AnB statements to be translated. This function is recursively defined as follows.

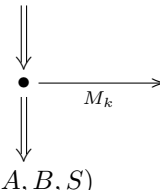
$$tr_\alpha \left(A, B, \begin{array}{c} B \rightarrow A : M_k \\ S \end{array} \right) =$$

Intuitively this is the agent A receiving a message from the network (presumably from B but A cannot verify this). Since the intruder has full control of the network this corresponds to the intruder knowing a message which fits the pattern of M_k as perceived by A .

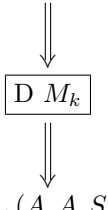
$$tr_\alpha \left(A, C, \begin{array}{c} B \rightarrow A : M_k \\ S \end{array} \right) = \text{error}$$

Even though A cannot verify the identity of B we cannot replace B with another agent, in order to send a message an agent has to be active. So if a non-active

agent tries to send a message we give an error message.

$$tr_\alpha \left(A, A, \begin{array}{l} A \rightarrow B : M_k \\ S \end{array} \right) =$$


Intuitively this is the agent A sending a message over the network (with the intended recipient being B). Since the intruder has full control of the network this corresponds to giving the information to the intruder.

$$tr_\alpha \left(A, A, \begin{array}{l} DM_k \\ S \end{array} \right) =$$


Where D can be '+', '-', '?' or '~?' This is the notation for A manipulating the database, if D is '+' the message is added to the database of A. Since an IF state is a set of fact no special care is taken to see if the data is already in the database.

if D is '-' the message is removed from the database of A. Due to the nature of IF states this may make impossible transitions since something has to match on the left hand side. This may seem problematic at first but by adding a branch before the - statement testing for the presence of the message(M_k) it can cover both situations, without always splitting states which may be quite inefficient when there are multiple removals in a row (n removals would lead to 2^n possible state transitions). It is assumed that mostly when messages are removed from a database they are known to be present before hand.

If D is '?' the database is checked for the knowledge M_k and locking up if it is not there(similar to the notion of an assertion in C or Java). It is a simple way to truly and immediately abandon a protocol and as such should be kept to a minimum(assuming that most agents would prefer to have a safe fail-over rather than a crash). Similarly '~?' can be used for the opposite case, i.e. to

ascertain that the database does not contain M_k .

$$tr_\alpha \left(A, B, \begin{array}{c} DM_k \\ S \end{array} \right) = \Downarrow tr_\alpha(A, B, S)$$

Database manipulations are local to a particular agent, so nothing to do here.

$$tr_\alpha \left(A, A, \begin{array}{c} \text{if}(C) \\ \text{then} \\ S_1 \\ \text{else} \\ S_2 \\ \text{fi} \\ S_3 \end{array} \right) = \begin{array}{c} \swarrow \text{tr}_\gamma(A,p,C) \quad \searrow \text{tr}_\gamma(A,n,C) \\ tr_\alpha(A, A, S_1; S_3) \quad tr_\alpha(A, A, S_2; S_3) \end{array}$$

This is the classical if sentence, intuitively this is an unconditional jump with a condition on each branch that are assured to be mutually exclusive. While the branching is done by A care should be taken that other agents will be able to tell which branch A took on the first message (to prevent indeterminism which may lock the other agents in unwanted states). The translation function tr_γ is defined in section 4.2.5.

$$tr_\alpha \left(A, B, \begin{array}{c} \text{if}(C) \\ \text{then} \\ S_1 \\ \text{else} \\ S_2 \\ \text{fi} \\ S_3 \end{array} \right) = \begin{array}{c} \swarrow \quad \searrow \\ tr_\alpha(A, B, S_1; S_3) \quad tr_\alpha(A, B, S_2; S_3) \end{array}$$

The choice is made by the active agent so intuitively if B makes the choice A has to be ready to accept either outcome, as described above this might be a problem. Luckily we have already covered that only one agent is active at a time so in order for A to act A has to receive a message first or make a timeout.

$$tr_\alpha \left(A, B, \begin{array}{l} B \rightarrow A : M_k \text{ OR } S_1 \\ S_2 \end{array} \right) =$$

Intuitively this is the time-out statement. In practice S_1 is replaced by an identifier specifying a protocol to jump to at the choice of the inactive agent waiting on receiving M_k . As the translation shows, this will always result in this agent becoming the active one. This is exactly the desired behaviour of a time-out in an agent, rather than wait for a message the agent takes action.

Lastly there is the case of missing statements inside if statements or what to do when there are no more statements in general. Covering this case requires the introduction of the empty statement denoted ϵ . $tr_\alpha(A, B, \epsilon) = \epsilon$

4.2.3 Strand preprocessing

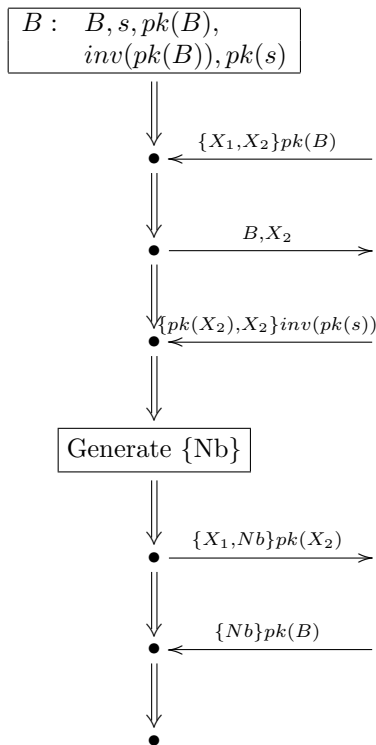
Before translating from Strands into IF it is important to have a preprocessor that runs through the strands and finds all variables not already known to the agent (either as initial knowledge or through messages received or generated by the agent) which needs to be either send or put into the database. The full specification for this is not interesting for this thesis and is rather complicated. Figure 4.1 has an example of how to do it for agent B of the NSPK example.

As described earlier this is also a good place to make an aggregation of multiple states so that each transition runs from one receive to the next. To avoid this complication the following translation deals with only a small part of a strand at a time.

4.2.4 Translation from WS-Strands to IF

The translation is defined as the function $tr_\beta(Nr, Strand)$, where Nr is a unique identifier of the state that the agent is in. Usually this is initialised to be 0, and $Strand$ is a strand initialised with the initial knowledge of the agent. The function is defined as follows.

Figure 4.1: Fresh variable detection in NSPK



In IF the knowledge of the intruder is specified as $iknows$. Since the model of the intruder that IF

$$tr_{\beta} \left(\begin{array}{c} Nr, \boxed{A: K} \\ \Downarrow \\ \bullet \xleftarrow{M_k} \\ \Downarrow \\ Rest \end{array} \right) = tr_{\beta} \left(\begin{array}{c} state_A(Nr, K). \\ iknows(M_k) \\ \Rightarrow \\ state_A(s(Nr), K \cup var(M_k)) \\ \cup \\ s(Nr), \boxed{A: K \cup var(M_k)} \\ \Downarrow \\ Rest \end{array} \right)$$

This is essentially giving the message to the intruder.

$$tr_{\beta} \left(\begin{array}{c} Nr, \boxed{A: K} \\ \Downarrow \\ \bullet \xrightarrow{M_k} \\ \Downarrow \\ Rest \end{array} \right) = tr_{\beta} \left(\begin{array}{c} state_A(Nr, K) \\ \Rightarrow \\ iknows(M_k). \\ state_A(s(Nr), K) \\ \cup \\ s(Nr), \boxed{A: K} \\ \Downarrow \\ Rest \end{array} \right)$$

Generating fresh variables in IF has a special notation.

$$\begin{array}{c}
 \left(\begin{array}{c}
 tr_{\beta} \quad Nr, \quad \boxed{A: K} \\
 \Downarrow \\
 \boxed{\text{Generate } V} \\
 \Downarrow \\
 Rest
 \end{array} \right) = \begin{array}{c}
 state_A(Nr, K) \\
 = [existsV] \Rightarrow \\
 state_A(s(Nr), K \cup V) \\
 \cup \\
 \left(\begin{array}{c}
 tr_{\beta} \quad s(Nr), \quad \boxed{A: K \cup V} \\
 \Downarrow \\
 Rest
 \end{array} \right)
 \end{array}
 \end{array}$$

Adding knowledge to the database is a right-hand side statement.

$$\begin{array}{c}
 \left(\begin{array}{c}
 tr_{\beta} \quad Nr, \quad \boxed{A: K} \\
 \Downarrow \\
 \boxed{+ M_k} \\
 \Downarrow \\
 Rest
 \end{array} \right) = \begin{array}{c}
 state_A(Nr, K) \\
 \Rightarrow \\
 db(A, M_k). \\
 state_A(s(Nr), K) \\
 \cup \\
 \left(\begin{array}{c}
 tr_{\beta} \quad s(Nr), \quad \boxed{A: K} \\
 \Downarrow \\
 Rest
 \end{array} \right)
 \end{array}
 \end{array}$$

Similarly removing knowledge from the database is a left-hand side statement. It is worth noting that this may get the agent stuck if the knowledge is not contained in the database.

$$\begin{array}{c}
 \left(\begin{array}{c}
 tr_{\beta} \quad Nr, \boxed{A: K} \\
 \Downarrow \\
 \boxed{- M_k} \\
 \Downarrow \\
 Rest
 \end{array} \right) = \begin{array}{c}
 state_A(Nr, K). \\
 db(A, M_k) \\
 \Rightarrow \\
 state_A(s(Nr), K) \\
 \cup \\
 \left(\begin{array}{c}
 tr_{\beta} \quad s(Nr), \boxed{A: K} \\
 \Downarrow \\
 Rest
 \end{array} \right)
 \end{array}
 \end{array}$$

When (positively) querying the database the "db" statement will appear on both sides, as seen above removal has no special symbol in IF and anything on the left-hand side which is not repeated on the right-hand side is removed.

$$\begin{array}{c}
 \left(\begin{array}{c}
 tr_{\beta} \quad Nr, \boxed{A: K} \\
 \Downarrow \\
 \boxed{? M_k} \\
 \Downarrow \\
 Rest
 \end{array} \right) = \begin{array}{c}
 state_A(Nr, K). \\
 db(A, M_k) \\
 \Rightarrow \\
 db(A, M_k). \\
 state_A(s(Nr), K) \\
 \cup \\
 \left(\begin{array}{c}
 tr_{\beta} \quad s(Nr), \boxed{A: K} \\
 \Downarrow \\
 Rest
 \end{array} \right)
 \end{array}
 \end{array}$$

Negatively querying the database is very similar, except that a "not" is put in front of the "db" statement on the left-hand side and it is not mentioned on the right hand side.

$$\begin{array}{c}
\left(\begin{array}{c}
Nr, \boxed{A: K} \\
\Downarrow \\
\boxed{\sim?M_k} \\
\Downarrow \\
Rest
\end{array} \right) = \begin{array}{c}
state_A(Nr, K). \\
not(db(A, M_k)) \\
\Rightarrow \\
state_A(s(Nr), K) \\
\cup \\
\left(\begin{array}{c}
tr_\beta \left(\begin{array}{c}
s(Nr), \boxed{A: K} \\
\Downarrow \\
Rest
\end{array} \right)
\end{array} \right)
\end{array}
\end{array}$$

Conditions are handled by tr_γ which is described in the section 4.2.5: Conditions of WS-AnB. The positive case needs to have the result of that translation on both sides.

$$\begin{array}{c}
\left(\begin{array}{c}
Nr, \boxed{A: K}(C) \\
\Downarrow^p \\
\bullet \\
\Downarrow \\
Rest
\end{array} \right) = \begin{array}{c}
state_A(Nr, K). \\
C \\
\Rightarrow \\
C. \\
state_A(s(Nr), K) \\
\cup \\
\left(\begin{array}{c}
tr_\beta \left(\begin{array}{c}
s(Nr), \boxed{A: K} \\
\Downarrow \\
Rest
\end{array} \right)
\end{array} \right)
\end{array}
\end{array}$$

The negative case needs is similar except without having the result of that translation on the right-hand side and of course the not on the left-hand side.

$$\begin{array}{c}
 \left(\begin{array}{c}
 tr_\beta \left(Nr, \boxed{A: K}(C) \right) \\
 \Downarrow^n \\
 \bullet \\
 \Downarrow \\
 Rest
 \end{array} \right) = \begin{array}{c}
 state_A(Nr, K). \\
 not(C) \\
 \Rightarrow \\
 state_A(s(Nr), K) \\
 \cup \\
 \left(\begin{array}{c}
 tr_\beta \left(s(Nr), \boxed{A: K} \right) \\
 \Downarrow \\
 Rest
 \end{array} \right)
 \end{array}
 \end{array}$$

Lastly there is the case of the splitting strand. Here it is important to distinguish between the two strands so rather than having one successor function s two additional successor functions are added l and r . These are also important markers for unifying strands.

$$\begin{array}{c}
 \left(\begin{array}{c}
 tr_\beta \left(Nr, \boxed{A: K} \right) \\
 \swarrow \quad \searrow \\
 Rest_1 \quad Rest_2
 \end{array} \right) = \begin{array}{c}
 \left(\begin{array}{c}
 tr_\beta \left(l(Nr), \boxed{A: K} \right) \\
 \Downarrow \\
 Rest_1
 \end{array} \right) \\
 \cup \\
 \left(\begin{array}{c}
 tr_\beta \left(r(Nr), \boxed{A: K} \right) \\
 \Downarrow \\
 Rest_2
 \end{array} \right)
 \end{array}
 \end{array}$$

Although we have chosen not to join strands it is possible to describe the circumstances in which this is possible. This might be an optimization because it leads to a smaller IF specification. Briefly the join can be translated on IF level as the pattern:

`state_A(Z(1(S(0))),K)`

```

state_A(Y(r(S(0))),K)
=>
state_A(s(S(Nr)),K)

```

Where S, Z and Y is a sequence of 0 or more successor calls. It is important that the two S(0) match to prevent nested if sentences from breaking.

4.2.5 Conditions of WS-AnB

To handle double negations I start by converting the condition to a base form.

$$tr_\gamma(A, p, \sim C) = tr_\gamma(A, n, C)$$

$$tr_\gamma(A, n, \sim C) = tr_\gamma(A, p, C)$$

Then there are two kinds of conditions in WS-AnB. The first is equality of messages.

$$tr_\gamma(A, O, M_1 = M_2) = O(equal(M_1, M_2))$$

The second is database checks.

$$tr_\gamma(A, O, ?M_1) = O(db(A, M_1))$$

Although it is possible to extend this to handle conjunctions or disjunctions we have decided not to implement this. This is not a restriction on the language however since it can be covered by nesting if sentences.

Bibliography

- [1] Grit Denker, Jon Millen, and Jon Millen. Capsl and cil language design - a common authentication protocol specification language and its intermediate language. 1999.
- [2] Grit Denker and Jonathan K. Millen. Optimizing protocol rewrite rules of cil specifications. In *In CSFW*, pages 52–62. IEEE Computer Society Press, 2000.
- [3] Paul Hankes Drielsma, Sebastian Mödersheim, and Sebastian Mödersheim. The asw protocol revisited: A unified view. pages 145–161, 2005.
- [4] Sebastian Mödersheim. Algebraic properties in alice and bob notation. In *ARES*, pages 433–440, 2009.
- [5] Sebastian Mödersheim, Luca Viganò, and Luca Viganò. The open-source fixed-point model checker for symbolic analysis of security protocols. 2009.