Protocol Verification Using FDR

Software Security

Steffen Helke

Chair of Software Engineering

16th January 2019

Brandenburgische Technische Universität Cottbus - Senftenberg

Objectives of today's lecture

- Getting to know how to *model* and *verify* security protocols
- Understanding a *CSP formalizations for the Needham Schroeder protocol*
- Being able to *prove important security properties* on a given protocol using the model checker FDR

How to model a security protocol using CSP?

Specification Language CSP

What does the abbreviation CSP mean?

Communicating Sequential Processes

For which purposes was CSP designed?

It can be used to formally describe the interactions between communicating processes

Who invented CSP?

Tony Hoare 1978, later extensions of Bill Roscoe and others

Should you know the language CSP?

Yes, because CSP is one of the most popular traditional modeling languages!

➜ Note, CSP book was long time on the 2nd place (currently 14th place) of the most cited computer science articels

`http://citeseer.ist.psu.edu/stats/articles`

Modeling with CSP

Basic Concept

The behavior of a system is described by communicating events between processes!

Ingredients

Events
: Abstractions of atomic, timeless actions
e.g. *receiving* a message

Processes
: Computations represented as a sequence of *executed* and/or *refused events*

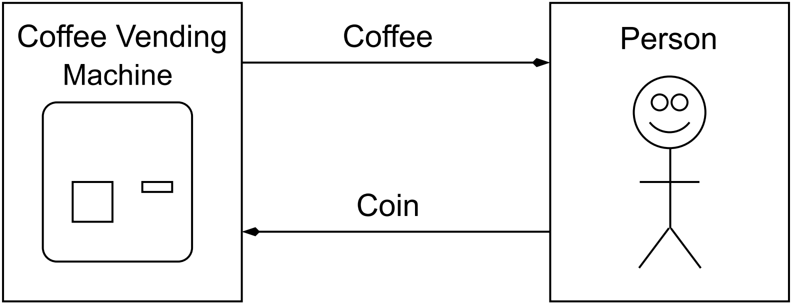
CSP Syntax

Basic notation for defining processes (only a selection)

$P ::=$	$a \rightarrow Q$	prefix operator
$\mid$	$P \mid[A]\mid Q$	parallel, synchronized using the events of A
$\mid$	$P \mid\mid\mid Q$	parallel, not synchronized (interleaved)
$\mid$	$P \, \Box \, Q$	external choice
$\mid$	$P \sqcap Q$	internal choice
$\mid$	$P \setminus A$	hiding
$\mid$	$P \, [[a \leftarrow b]]$	renaming
$\mid$	$P \, ; \, Q$	sequential composition
$\mid$	$Stop$	stopping
$\mid$	$Skip$	termination

Note: a and b represent events, P and Q represent processes

Example: How to model the behavior of a coffee machine?



Modeling using CSP

Specification

$$CoffeeMachine = coin \rightarrow coffee \rightarrow Stop$$

$$Person = (coin \rightarrow coffee \rightarrow Stop) \,\Box\, (card \rightarrow coffee \rightarrow Stop)$$

$$System = CoffeeMachine \,|[\, \{coin, coffee\} \,]|\, Person$$

Trace Semantics

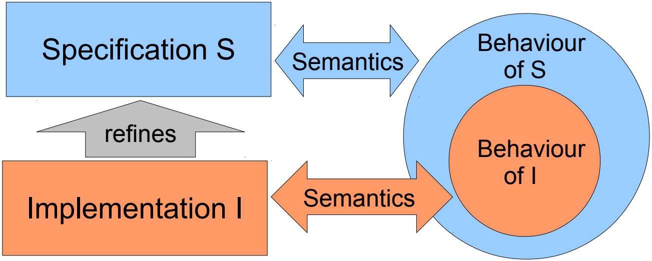
$$traces(CoffeeMachine) = \{\langle\rangle, \langle coin\rangle, \langle coin, coffee\rangle\}$$

$$traces(Person) = \{\langle\rangle, \langle coin\rangle, \langle card\rangle, \langle coin, coffee\rangle, \langle card, coffee\rangle\}$$

$$traces(System) = \{\langle\rangle, \langle coin\rangle, \langle card\rangle, \langle coin, coffee\rangle\}$$

Conformance by Refinement

- *Abstract specification* defines acceptable behaviour
- Behavior of a more *concrete implementation* must be included in the behavior of the abstract specification
- The simplest way to define the behavior of a CSP process is to use a trace sematics



How to refine processes of CSP?

Main Idea

A process P is refined by a process Q if and only if the behavior of Q is contained in P

$$P \sqsubseteq_{\mathsf{T}} Q = \mathit{traces}(Q) \subseteq \mathit{traces}(P)$$

Example

$\mathit{Person} \sqsubseteq_T \mathit{System}$

$$\mathit{traces}(\mathit{Person}) = \{\langle\rangle, \langle \mathit{coin}\rangle, \langle \mathit{card}\rangle, \langle \mathit{coin}, \mathit{coffee}\rangle, \langle \mathit{card}, \mathit{coffee}\rangle\}$$
$$\mathit{traces}(\mathit{System}) = \{\langle\rangle, \langle \mathit{coin}\rangle, \langle \mathit{card}\rangle, \langle \mathit{coin}, \mathit{coffee}\rangle\}$$

$\mathit{System} \sqsubseteq_T \mathit{CoffeeMachine}$

$$\mathit{traces}(\mathit{System}) = \{\langle\rangle, \langle \mathit{coin}\rangle, \langle \mathit{card}\rangle, \langle \mathit{coin}, \mathit{coffee}\rangle\}$$
$$\mathit{traces}(\mathit{CoffeeMachine}) = \{\langle\rangle, \langle \mathit{coin}\rangle, \langle \mathit{coin}, \mathit{coffee}\rangle\}$$

$\mathit{CoffeeMachine} \sqsubseteq_T \mathit{Stop}$

$$\mathit{traces}(\mathit{CoffeeMachine}) = \{\langle\rangle, \langle \mathit{coin}\rangle, \langle \mathit{coin}, \mathit{coffee}\rangle\}$$
$$\mathit{traces}(\mathit{Stop}) = \{\langle\rangle\}$$

Tools for CSP

Automatic Refinement Checker

FDR, PAT, ARC

Interactive Refinement Checker

CSP-Prover

Model Checker

ProB, PAT

Animators

ProBE, ProB, PAT

Machine Readable CSP

CSP Dialect of FDR

- How to define data types?

  datatype X = Value1 | Value2 | Value3

- How to define events of a channel?

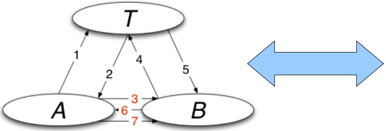
  channel a : X

- Events that can be communicated via the channel *a*

  $\{| \, a \, |\} = \{$a.Value1, a.Value2, a.Value3$\}$

Event as an input	a?x $\rightarrow P(x)$
Event as an output	a!Value1 $\rightarrow P$
Event without an explicit direction	a.Value2 $\rightarrow P$

Example: The Needham-Schroeder Protocol



???

Procedure

- How to model Needham-Schroeder protocol using CSPm?
- How to formulate important properties and how to verify these properties on the model?

Learning Objectives

- Getting a feeling how to benefit from CSP/FDR
- There is no intention to train you as a CSP specialist, i.e. the CSP model of NSPs does not have to be completely memorised

Formal Verification of the Needham-Schroeder Protocol

History of the Needham-Schroeder Protocol

- **1978** Publication of the Needham-Schroeder protocol by R. Needham & M. Schroeder

  ➜ Aim is to develop a secure authentication mechanism

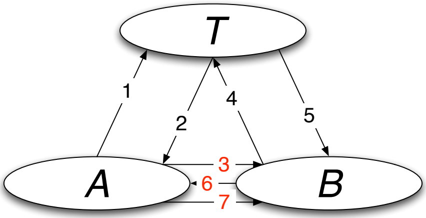
- **1990** Publication of M. Burrows, M. Abadi & R. Needham: Proof of correctness of the protocol based on BAN logic

  ➜ Unfortunately, the proof later turns out to be faulty

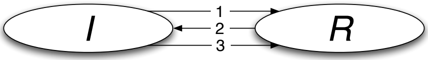
- **1995** Gavin Lowe detects an attack on the NSP by hand
- **1997** Gavin Lowe proves the correctness of a new protocol variant using FDR

Repetition: Needham-Schroeder Protocol

Complete Version of the Asymmetric Protocol Variant



Simplified Version without using T



Repetition: Protocol Steps of the NSP

1 $A \rightarrow T : \{A, B\}$

2 $T \rightarrow A : \{B, PK_B\}_{SK_T}$

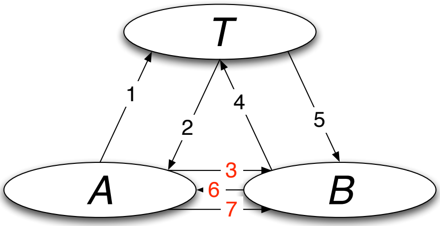
3 $A \rightarrow B : \{A, N_A\}_{PK_B}$

4 $B \rightarrow T : \{B, A\}$

5 $T \rightarrow B : \{A, PK_A\}_{SK_T}$

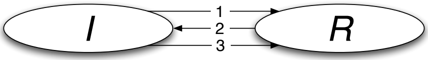
6 $B \rightarrow A : \{N_A, N_B\}_{PK_A}$

7 $A \rightarrow B : \{N_B\}_{PK_B}$



Attack for the Simplified Protocol Variant

Simplified NSP Version without using T



Attack Scenario

1.1 $A \rightarrow C : \{N_A, A\}_{PK(C)}$

2.1 $C(A) \rightarrow B : \{N_A, A\}_{PK(B)}$

2.2 $B \rightarrow C(A) : \{N_A, N_B\}_{PK(A)}$

1.2 $C \rightarrow A : \{N_A, N_B\}_{PK(A)}$

1.3 $A \rightarrow C : \{N_B\}_{PK(C)}$

2.3 $C(A) \rightarrow B : \{N_B\}_{PK(B)}$

How to code NSP using CSP?

1. Model the roles *Initiator* and *Responder* using generic CSP processes and run these processes in parallel

2. Define concrete participants, e.g. A, B and C who can play any of these roles

3. Describe a protocol step using a CSP event

4. Communicate all messages via appropriate CSP channels

Enrichment of Protocol Messages

Which participants are related to a message?

Extend protocol messages in such a way that information about the sender and receiver is also transferred

1.1 $A \to C : A.C.\{N_A, A\}_{PK(C)}$

2.1 $C(A) \to B : A.B.\{N_A, A\}_{PK(B)}$

2.2 $B \to C(A) : B.A.\{N_A, N_B\}_{PK(A)}$

1.2 $C \to A : C.A.\{N_A, N_B\}_{PK(A)}$

1.3 $A \to C : A.C.\{N_B\}_{PK(C)}$

2.3 $C(A) \to B : A.B.\{N_B\}_{PK(B)}$

How to formalize the three different message types for NSP?

$$
\begin{aligned}
MSG1 = \; & \{Msg_1.a.b.Encrypt_1.k.n_a.a' \mid \\
& \quad a, a' \in Initiator, b \in Responder, k \in Key, n_a \in Nonces\}
\end{aligned}
$$

$$
\begin{aligned}
MSG2 = \; & \{Msg_2.b.a.Encrypt_2.k.n_a.n_b \mid \\
& \quad a \in Initiator, b \in Responder, k \in Key, n_a, n_b \in Nonces\}
\end{aligned}
$$

$$
\begin{aligned}
MSG3 = \; & \{Msg_3.a.b.Encrypt_3.k.n_b \mid \\
& \quad a \in Initiator, b \in Responder, k \in Key, n_b \in Nonces\}
\end{aligned}
$$

$$
MSGs = MSG1 \cup MSG2 \cup MSG3
$$

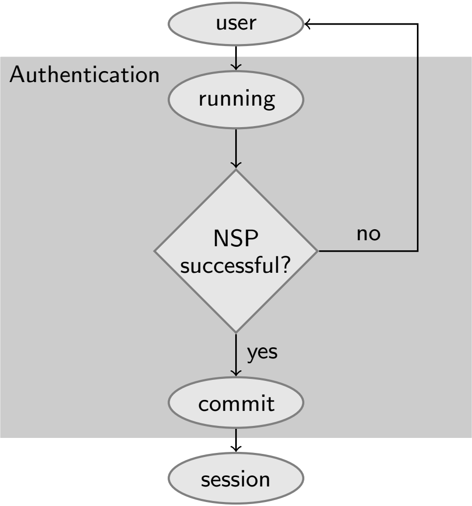
How to code the messages using CSPm?

```
datatype KEY     = ka     | kb     | kc
datatype AKTEUR  = A      |  B     | C
datatype NONCE   = NonceA | NonceB | NonceC
datatype TICKET1 = Encrypt1.KEY.NONCE.AKTEUR
datatype TICKET2 = Encrypt2.KEY.NONCE.NONCE
datatype TICKET3 = Encrypt3.KEY.NONCE

datatype MSG = Msg1.AKTEUR.AKTEUR.TICKET1
  | Msg2.AKTEUR.AKTEUR.TICKET2
  | Msg3.AKTEUR.AKTEUR.TICKET3

channel comm : MSG
```

Question: How many different events can be communicated via the channel *comm*?

➜ This channel accepts $3^5 + 3^5 + 3^4 = 567$ different events

How to observe the current state of the Protocol?



channel
user, *session*, *I_running*, *R_running*, *I_commit*, *R_commit* : *Initiator*.*Responder*

Initiator Process

$$
\begin{aligned}
& \mathit{INITIATOR}(a, n_a) = \\
& \qquad \mathit{user}!a?b \to \mathit{I\_running}.a.b \to \\
& \qquad \mathit{comm}.\mathit{Msg}_1.a.b.\mathit{Encrypt}_1.\mathit{key}(b)!n_a.a \to \\
& \qquad \mathit{comm}.\mathit{Msg}_2.b.a.\mathit{Encrypt}_2.\mathit{key}(a)?n_a'.n_b \to \\
& \qquad \textbf{if } n_a = n_a' \\
& \qquad \textbf{then } \mathit{comm}.\mathit{Msg}_3.a.b.\mathit{Encrypt}_3.\mathit{key}(b)!n_b \to \\
& \qquad \qquad \mathit{I\_commit}.a.b \to \mathit{session}.a.b \to \mathit{Skip} \\
& \qquad \textbf{else } \mathit{Stop}
\end{aligned}
$$

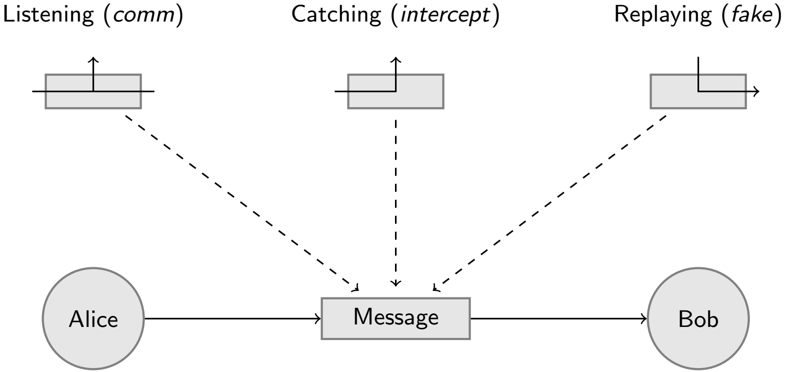
Responder Process

$$
\begin{aligned}
&\mathit{RESPONDER}(b, n_b) = \\
&\qquad user?a!b \rightarrow R\_running.a.b \rightarrow \\
&\qquad comm.Msg_1.a.b.Encrypt_1.key(b)?n_a.a \rightarrow \\
&\qquad comm.Msg_2.b.a.Encrypt_2.key(a)!n_a.n_b \rightarrow \\
&\qquad comm.Msg_3.a.b.Encrypt_3.key(b)?n'_b \rightarrow \\
&\qquad \textbf{if } n_b = n'_b \\
&\qquad \textbf{then } R\_commit.a.b \rightarrow session.a.b \rightarrow Skip \\
&\qquad \textbf{else } Stop
\end{aligned}
$$

Initiator and responder synchronization is based on the event set S

$$S = \{|\ comm, session.A.B\ |\}$$

How to model attacker channels?



channel $comm, fake, intercept : MSGs$

How do I rename the channels of the initiator process to obtain a suitable attacker interface?

$$
\begin{aligned}
& \mathit{INITIATOR}1 = \\
& \quad \mathit{INITIATOR}(A, N_a) \\
& \qquad [[\; \mathit{comm}.\mathit{Msg}_1 \leftarrow \mathit{comm}.\mathit{Msg}_1, \\
& \qquad\quad \mathit{comm}.\mathit{Msg}_1 \leftarrow \mathit{intercept}.\mathit{Msg}_1, \\
& \qquad\quad \mathit{comm}.\mathit{Msg}_2 \leftarrow \mathit{comm}.\mathit{Msg}_2, \\
& \qquad\quad \mathit{comm}.\mathit{Msg}_2 \leftarrow \mathit{fake}.\mathit{Msg}_2, \\
& \qquad\quad \mathit{comm}.\mathit{Msg}_3 \leftarrow \mathit{comm}.\mathit{Msg}_3, \\
& \qquad\quad \mathit{comm}.\mathit{Msg}_3 \leftarrow \mathit{intercept}.\mathit{Msg}_3 \;]]
\end{aligned}
$$

How do I rename the channels of the responder process to obtain a suitable attacker interface?

$$
\begin{aligned}
& RESPONDER1 = \\
& \quad RESPONDER(B, N_b) \\
& \qquad [[\ comm.Msg_1 \leftarrow comm.Msg_1, \\
& \qquad\quad comm.Msg_1 \leftarrow fake.Msg_1, \\
& \qquad\quad comm.Msg_2 \leftarrow comm.Msg_2, \\
& \qquad\quad comm.Msg_2 \leftarrow intercept.Msg_2, \\
& \qquad\quad comm.Msg_3 \leftarrow comm.Msg_3, \\
& \qquad\quad comm.Msg_3 \leftarrow fake.Msg_3\ ]]
\end{aligned}
$$

What could an attacker do in principle?

1. He/she is able to listen to and/or intercept messages
2. He/she is able to learn nonces
3. He/she is able to send new messages using the learned nonces
4. He/she is able to replay old messages (possibly modified)
5. It is also possible to replay old encrypted messages that the attacker cannot decrypt

Note, the formalization of such an attacker behaviour is also called *Dolev-Yao model* based on a research paper from 1983[1]

[1] *D. Dolev and A. Yao: On the security of public key protocols, IEEE Journal Transactions on Information Theory, 29/2, 1983.*

Attacker Process (1)

$$
\begin{array}{l}
\mathit{INTRUDER}(m1s, m2s, m3s, ns) = \\
\quad \mathit{comm}.\mathit{Msg}_1?a.b.\mathit{Encrypt}_1.k.n.a' \rightarrow \\
\qquad \textbf{if } k = K_I \textbf{ then } \mathit{INTRUDER}(m1s, m2s, m3s, ns \cup \{n\}) \\
\qquad \textbf{else } \mathit{INTRUDER}(m1s \cup \{\mathit{Encrypt}_1.k.n.a'\}, m2s, m3s, ns) \\
\Box \ \mathit{intercept}.\mathit{Msg}_1?a.b.\mathit{Encrypt}_1.k.n.a' \rightarrow \\
\qquad \textbf{if } k = K_I \textbf{ then } \mathit{INTRUDER}(m1s, m2s, m3s, ns \cup \{n\}) \\
\qquad \textbf{else } \mathit{INTRUDER}(m1s \cup \{\mathit{Encrypt}_1.k.n.a'\}, m2s, m3s, ns) \\
\Box \ \mathit{comm}.\mathit{Msg}_2?b.a.\mathit{Encrypt}_2.k.n.n' \rightarrow \\
\qquad \textbf{if } k = K_I \textbf{ then } \mathit{INTRUDER}(m1s, m2s, m3s, ns \cup \{n, n'\}) \\
\qquad \textbf{else } \mathit{INTRUDER}(m1s, m2s \cup \{\mathit{Encrypt}_2.k.n.n'\}, m3s, ns) \\
\Box \ \mathit{intercept}.\mathit{Msg}_2?b.a.\mathit{Encrypt}_2.k.n.n' \rightarrow \\
\qquad \textbf{if } k = K_I \textbf{ then } \mathit{INTRUDER}(m1s, m2s, m3s, ns \cup \{n, n'\}) \\
\qquad \textbf{else } \mathit{INTRUDER}(m1s, m2s \cup \{\mathit{Encrypt}_2.k.n.n'\}, m3s, ns)
\end{array}
$$

Attacker Process (2)

$INTRUDER(m1s, m2s, m3s, ns) =$

$\square$ $comm.Msg_3?a.b.Encrypt_3.k.n \rightarrow$

$\quad$ **if** $k = K_I$ **then** $I(m1s, m2s, m3s, ns \cup \{n\})$

$\quad$ **else** $I(m1s, m2s, ms3 \cup \{Encrypt_3.k.n\}, ns)$

$\square$ $intercept.Msg_3?a.b.Encrypt_3.k.n \rightarrow$

$\quad$ **if** $k = K_I$ **then** $I(m1s, m2s, m3s, ns \cup \{n\})$

$\quad$ **else** $I(m1s, m2s, ms3 \cup \{Encrypt_3.k.n\}, ns)$

$\square$ $fake.Msg_1?a.b?m{:}m1s \rightarrow I(m1s, m2s, m3s, ns)$

$\square$ $fake.Msg_2?b.a?m{:}m2s \rightarrow I(m1s, m2s, m3s, ns)$

$\square$ $fake.Msg_3?a.b?m{:}m3s \rightarrow I(m1s, m2s, m3s, ns)$

$\square$ $fake.Msg_1?a.b!Encrypt_1?k?n{:}ns?a' \rightarrow I(m1s, m2s, m3s, ns)$

$\square$ $fake.Msg_2?b.a!Encrypt_2?k?n{:}ns?n'{:}ns \rightarrow I(m1s, m2s, m3s, ns)$

$\square$ $fake.Msg_3?a.b!Encrypt_3?k?n{:}ns \rightarrow I(m1s, m2s, m3s, ns)$

Note: The identifier *INTRUDER* is abbreviated here in the recursive call by *I*

How to construct a complete system process including the capabilities of an attacker?

$$AGENTS = \\ \quad INITIATOR1 \, [| \{| \, comm, session.A.B \, |\} |] \, RESPONDER1$$

$$INTRUDER1 = INTRUDER(\varnothing, \varnothing, \varnothing, \{N_C\})$$

$$SYSTEM = \\ \quad AGENTS \, [| \{| \, fake, comm, intercept \, |\} |] \, INTRUDER1$$

Specification for a Correct Authentication of the Initiator

$$AI_0 = \quad I\_running.A.B \rightarrow R\_commit.A.B \rightarrow AI_0$$

$$AI = \quad AI_0 \mid\mid\mid RUN(\Sigma \setminus A_2)$$

where $A_2 = \{\mid I\_running.A.B, R\_commit.A.B \mid\}$,
$\Sigma \mathrel{\widehat{=}}$ complete communication alphabet

and $RUN(M) \mathrel{\widehat{=}}$ infinite process that communicates the events of M in an arbitrary order

Specification for a Correct Authentication of the Responder

$$AR_0 = \quad R\_running.A.B \to I\_commit.A.B \to AR_0$$

$$AR = \quad AR_0 \mid\mid\mid RUN(\Sigma \setminus A_1)$$

where $A_1 = \{\mid R\_running.A.B, I\_commit.A.B \mid\}$

$\Sigma \mathrel{\widehat{=}}$ complete communication alphabet

and $RUN(M) \mathrel{\widehat{=}}$ infinite process that communicates the events of M in an arbitrary order

Proof of Correctness by Refinement

Tool Support

Automatic verification by the refinement checker FDR

Proof Obligations

$traces(SYSTEM) \subseteq traces(AR)$

damit gilt $AR \sqsubseteq_T SYSTEM$

$traces(SYSTEM) \not\subseteq traces(AI)$

damit gilt $AI \not\sqsubseteq_T SYSTEM$

Counterexample: Intruder Attack Scenario

Trace of the model checker

$$\langle user.A.B,\ user.A.C,\ I\_running.A.C,$$
$$intercept.Msg_1.A.C.Encrypt_1.K_c.N_a.A, \tag{1.1}$$
$$R\_running.A.B,$$
$$fake.Msg_1.A.B.Encrypt_1.K_b.N_a.A, \tag{2.1}$$
$$intercept.Msg_2.B.A.Encrypt_2.K_a.N_a.N_b, \tag{2.2}$$
$$fake.Msg_2.C.A.Encrypt_2.K_a.N_a.N_b, \tag{1.2}$$
$$intercept.Msg_3.A.C.Encrypt_3.K_c.N_b, \tag{1.3}$$
$$fake.Msg_3.A.B.Encrypt_3.K_b.N_b, \tag{2.3}$$
$$R\_commit.A.B\rangle$$

What is the cause of this counterexample?

$R\_commit.A.B$ occurs without a previous $I\_running.A.B$!

References

- Gavin Lowe: An Attack on the Needham-Schroeder Public-Key Authentication Protocol, Information Processing Letters, 1995.
- Gavin Lowe: Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR,Tools and Algorithms for the Construction and Analysis of Systems, Springer Verlag, pages 147-166, 1996.
- C. A. R. Hoare: Communicating Sequential Processes. http://www.usingcsp.com/, Prentice Hall International Series in Computer Science, 1985.