Interacting State Machines
– a stateful approach to proving security –

David von Oheimb
Siemens AG, Corporate Technology, D-81730 Munich
David.von.Oheimb@siemens.com

Overview

- Introduction
- Formalism: ISMs
  - Concept
  - Trace Semantics
  - Tool Support
- Case Study: Needham-Schroeder-Lowe
  - AutoFocus Representation
  - Isabelle Representation
  - Proofs
- Conclusion

Introduction

Aims:

Formal security modeling and verification
for requirements analysis and evaluation

Applications:

Abstractions of practical IT systems
involving both state and interaction,
e.g. protocols, smartcards, databases, ...

Tools:

Some suitable specification language and proof assistant

Closely Related Work

I/O Automata (IOAs) modeling asynchronous distributed computation

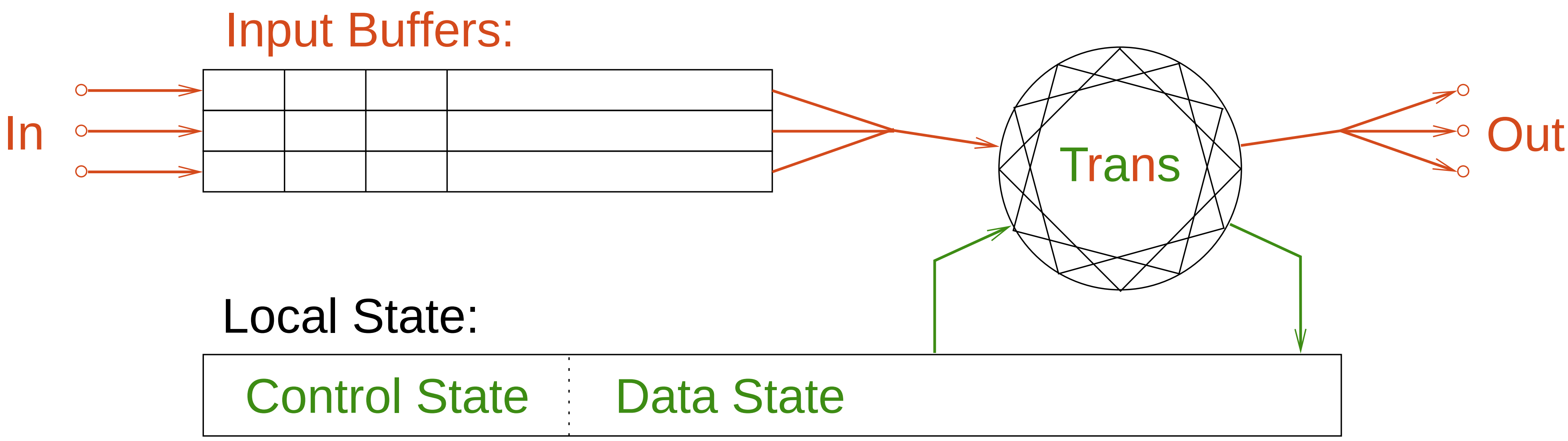
- $\oplus$ well-developed meta theory, e.g. compositionality
- $\oplus$ free tool support for specification, simulation, theorem proving, model checking, and code generation
- $\ominus$ low-level interaction scheme (singleton & unbuffered input/output)

AutoFocus Automata modeling embedded systems

- $\oplus$ multiple input/output, hierarchical components
- $\oplus$ graphical design, simulation, model checking, code generation (but most tools commercial)
- $\ominus$ clock-synchronous semantics

Interacting State Machines (ISMs)

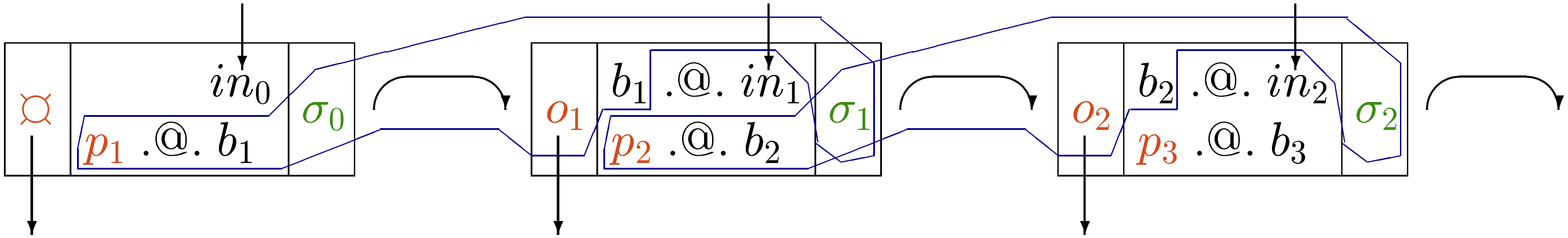
Automaton with (nondeterministic) state transitions +
Buffered i/o simultaneously on multiple connections



Transitions in executable and/or property-oriented style

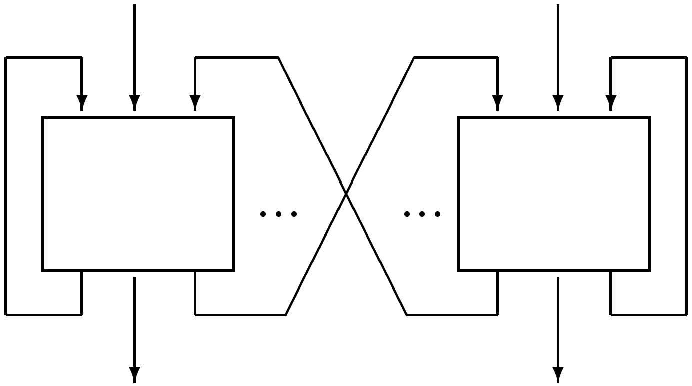
Trace Semantics

Runs by repeated application of transitions, with input from environment



Finite executions only ($\leadsto$ no liveness properties)

Parallel composition by interleaving transitions on product state



With internal communication and feedback, no built-in mulitcast

Tool Support

AutoFocus: graphical specification (and simulation) tool

Syntactic perspective

Graphical documentation

Type and consistency checks

Isabelle/HOL: powerful interactive theorem prover

Semantic perspective

Textual documentation

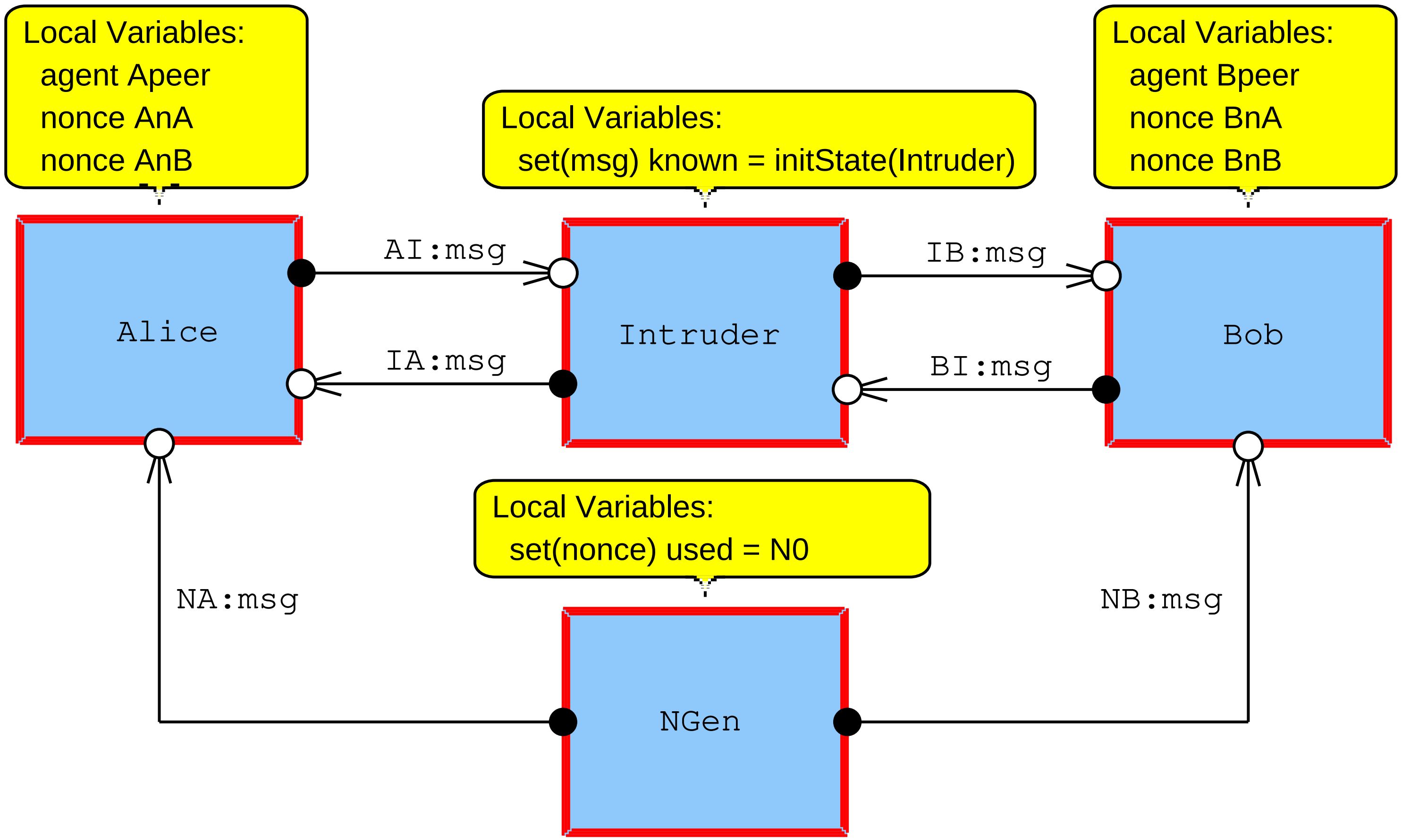
Validation and correctness proofs

AutoFocus drawing $\longrightarrow$ Quest file $\xrightarrow{\text{Conv}_1}$ Isabelle theory file

Within Isabelle: **ism** sections $\xrightarrow{\text{Conv}_2}$ Standard HOL definitions

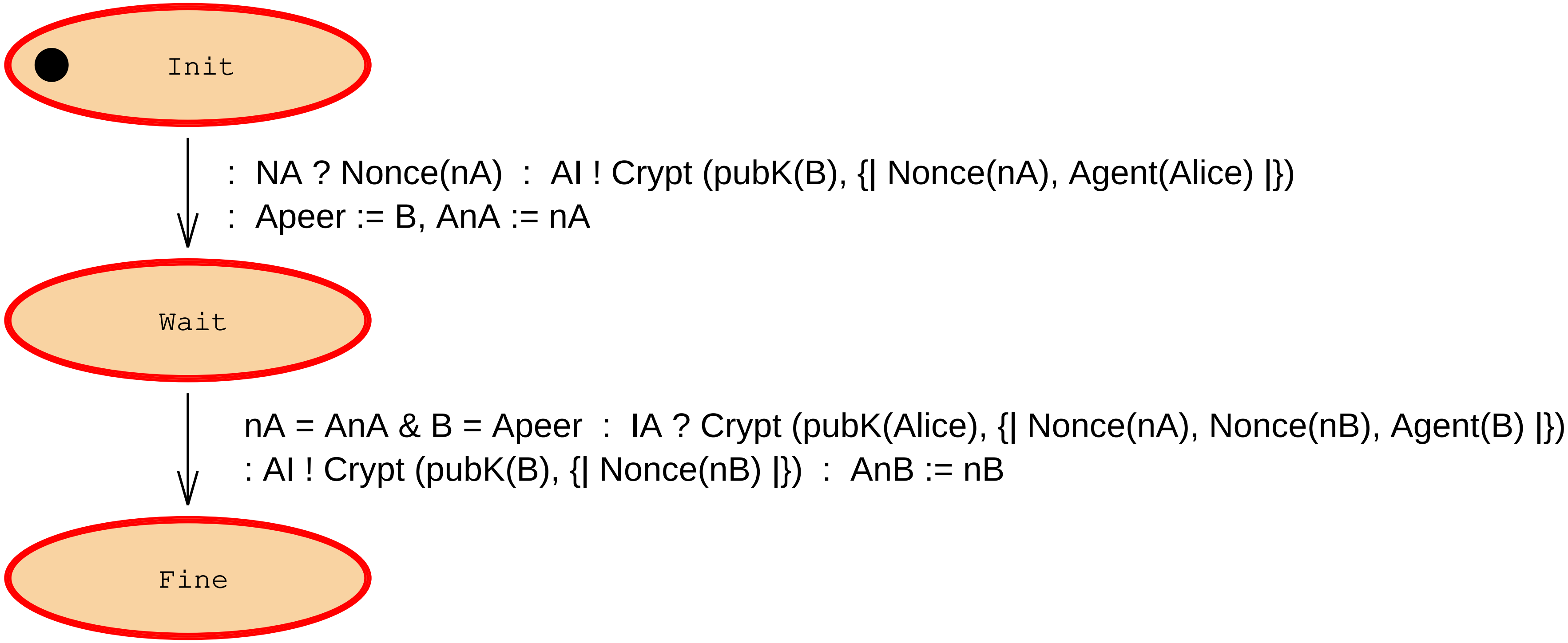
NSL System Structure Diagram

Lowe's fix of the *Needham-Schroeder* Public-Key Protocol
as *benchmark* for security in distributed systems



NSL State Transition Diagrams: `Alice`

Transition format: [`guard`] : [`inputs`] : [`outputs`] : [`assignments`]



Agents do have state: current knowledge and expectations

$\leadsto$ more detail than with event-based approach, but more natural

Isabelle Representation of NSL

```
ism Alice =
  ports channel
    inputs   "{NA,IA}"
    outputs  "{AI}"
  messages msg
  states   state
    control "Init"
          :: A_control
    data    "A0", σ
          :: A_data
```

```
transitions
Req: Init → Wait
  in  NA "[Nonce nA]"
  out AI "[Crypt (pubK B)
              {|Nonce nA, Agent Alice|}]"
  post Apeer := B, AnA := nA
Ack: Wait → Fine
  pre "nA = AnA(σ)", "B = Apeer(σ)"
  in  IA "[Crypt (pubK Alice)
              {|Nonce nA, Nonce nB, Agent B|}]"
  out AI "[Crypt (pubK B) (Nonce nB)]"
  post AnB := nB
```

High-level: multiple I/O, underspecification, genericity, nondeterminism

Transition rules $\leadsto$ rule induction principle

Cryptographic primitives (`Crypt`, `analz`, etc.) from *Paulson*'s formalization

NSL: Proofs

Authentication of `Alice` for Bob (even session agreement):

theorem $\mathit{Alice} \notin \mathit{bad} \longrightarrow$
$(b,\sigma)\#cs \in \mathit{PRuns}(\mathit{NSL}) \longrightarrow \mathit{Bob\_Conn\_with}\ \mathit{Alice}\ nB\ \sigma \longrightarrow$
$\exists\,(\_,\sigma') \in \mathit{set}\ cs.\ \mathit{Alice\_Fine\_with}\ \mathit{Bob}\ nB\ \sigma'$

. . . expressed wrt. agent state and not communication events

Proof using a variant of Schneider's rank function approach:

lemma $\mathit{Alice} \notin \mathit{bad} \longrightarrow cs \in \mathit{NSL\_Runs} \longrightarrow$
$(\forall(\_,\sigma)\in \mathit{set}\ cs.\ \mathit{Bob\_Resp\_only\_to}\ \mathit{Alice}\ \sigma \wedge \neg \mathit{Alice\_Wait\_nA}\ nB\ \sigma) \longrightarrow$
$(\forall(\_,\sigma)\in \mathit{set}\ cs.\ \neg \mathit{Alice\_Fine\_with}\ \mathit{Bob}\ nB\ \sigma) \longrightarrow$
$(\forall(b,\sigma)\in \mathit{set}\ cs.\ \forall m\in(\bigcup \mathit{set}`(b`\{AI,IA,IB,BI\}))\cup \mathit{I\_known}\ \sigma.\ \mathit{noleak}\ nB\ m)$

. . . takes 70 + 20 + 40 lines of proof script (semi-automatic tactics)
More detail and less automatic, but more flexible, more insights

Conclusion

- ISMs can be seen . . .
  - as high-level Input-/Output Automata
  - as asynchronous AutoFocus Automata
  - as stateful communicating processes
- Graphical representation suits design and documentation
- Using a theorem proving system . . .
  - helps handling all the details
  - reduces errors and omissions

$\Rightarrow$ ISMs provide good support for practical formal security analysis

Future work: more applications, extended proof support