



Lecture 3

- The CSP approach to the specification and analysis of Security protocols
 - Communicating Sequential Processes [Hoare 78]
 - Mathematical framework for the description and analysis of systems consisting of processes interacting via exchange of messages
 - Automatic tools available for proving properties of CSP specifications
 - Model-checker FDR
 - Theorem prover PVS



The CSP formalism

- A small mathematical language containing the main constructs for specifying concurrency, parallelism, communication, choice, hiding etc.
- The evolution of processes is based on a sequence of *events* or *actions*
 - *Visible actions* S
 - Interaction with other processes, communication
 - *Invisible action* t
 - Internal computation steps



The CSP language: Syntax

- Inaction: **Stop**
 - Termination, deadlock, incapability of performing any action, either internal or external
- Input: **$in ? x : A \rightarrow P(x)$**
 - Execute an input action on channel **in** , get message **x** of type **A** , then continue as **$P(x)$**
- Output: **$out ! m \rightarrow P(x)$**
 - Execute an output action on channel **out** , send message **m** , then continue as **$P(x)$**
- Recursion: **$P(y_1, \dots, y_n) = Body(y_1, \dots, y_n)$**
 - Process definition. **P** is a process name, **y_1, \dots, y_n** are the parameters, **$Body(y_1, \dots, y_n)$** is a process expression
- Example: **$Copy = in ? x \rightarrow out ! m \rightarrow Copy$**



CSP's Syntax (Cont'ed)

- External (aka guarded) choice: $P \parallel Q$
 - Execute a choice between P and Q . Do not choose a process which cannot proceed
 - Example: $(a ? x \rightarrow P(x)) \parallel (b ? x \rightarrow Q(x))$

Execute one and only one input action. If only one is available then choose that one. If both are available then choose arbitrarily. If none are available then block. The unchosen branch is discarded (commitment)
- Internal choice: $P + Q$
 - Execute an arbitrary choice between P and Q . It is possible to choose a process which cannot proceed



CSP's Syntax (Cont'ed)

- Parallel operator w/synchronization: $P \parallel Q$
 - P and Q proceed in parallel and are obliged to synchronize on all the common actions
- Example: $(c ? x \rightarrow P(x)) \parallel (c ! m \rightarrow Q)$
 - *Synchronization*: the two processes can proceed only if their actions correspond
 - *Handshaking*: sending and receiving is symultaneous (clearly an abstraction. Buffered communication can anyway be modeled by implementing a buffer process)
 - *Communication*: m is transmitted to the first process, which continues as $P(m)$.
 - *Broadcasting*: $c ! m$ is available for other parallel procs
- *Question*: what happens with the process

$$((c?x \rightarrow P(x)) [] (d?y \rightarrow Q(y))) \parallel (c!m \rightarrow R)$$



CSP's Syntax (Cont'ed)

- Parallel operator w/synchronization and interleaving: $P \parallel_A Q$
 - P and Q are obliged to synchronize only on the common actions in A
 - They *interleave* on all the actions not in A
- Example:
$$(c ? x \rightarrow P(x)) \parallel_{\{c\}} ((c ! m \rightarrow Q) [] (d ! n \rightarrow R))$$
 - the two processes can either synchronize on the action on channel c , or the second process can perform an action on d . In this second case the first process will remain blocked, though, until the second will decide to perform (if ever) an output action on c .
 - **Question:** in what part of the second process could this action on c be performed ?
- **Abbreviation:** $P \parallel Q$ stands for $P \parallel_f Q$



CSP's Syntax (Cont'ed)

- Hiding: $P \setminus A$
 - $P \setminus A$ behaves as P except that all the actions in A are turned into invisible actions. So they cannot be used anymore to synchronize with other processes.
 - One possible use of this mechanism is to avoid that external processes interfere with the communication channels in P . (Internalization of communication in P .)
- Renaming: $P[y/x]$
 - $P[x/y]$ behaves as P except that all the occurrences of x are renamed by y .
 - Typically it serves to create different instances of the same process scheme
 - Abbr: $P[y_1, y_2 / x_1, x_2]$ will stand for $P[y_1/x_1][y_2/x_2]$



Modeling Security Protocols in CSP

- Security protocols work through the interaction of a number of processes in parallel that send messages to each other. CSP is therefore an obvious notation for describing the participants and their role in the protocol
- **Example:** The Yahalom protocol
 - Message 1 $a \rightarrow b : a.n_a$
 - Message 2 $b \rightarrow s : b.\{a.n_a.n_b\}_{\text{ServerKey}(b)}$
 - Message 3 $s \rightarrow a : \{b.k_{ab}.n_a.n_b\}_{\text{ServerKey}(a)}.\{a.k_{ab}\}_{\text{ServerKey}(b)}$
 - Message 4 $a \rightarrow b : \{a.k_{ab}\}_{\text{ServerKey}(b)}.\{n_b\}_{k_{ab}}$
- **Questions:**
 - Is it based on symmetric or asymmetric cryptography?
 - what are n_a and n_b , and what is their purpose?



Modeling Security Protocols in CSP

- We assume that each process has channels
 - Receive
 - Send

that it uses for all communications with the other nodes via the medium

- Let us assume that **A** (Anne) and **B** (Bob) use the protocol, with **A** as initiator and **B** as responder, and that **J** (Jeeves) is the secure server



Modeling Security Protocols in CSP

- **A's view (initiator):**
 - Message 1 **a** sends to **b**: $a.n_a$
 - Message 3 **a** gets from '**j**': $\{b.k_{ab}.n_a.n_b\}_{\text{ServerKey}(a)}.\{a.k_{ab}\}_{\text{ServerKey}(b)}$
 - Message 4 **a** sends to **b**: $\{a.k_{ab}\}_{\text{ServerKey}(b)}.\{n_b\}_{k_{ab}}$
- In CSP this behavior can be modeled as follows:

Initiator(a, n_a) =
env?b: Agent
→ send.a.b.a.n_a
→ [] (receive.J.a{ $b.k_{ab}.n_a.n_b$ }_{ServerKey(a)}.m
→ send.a.b.m.{ n_b }_{k_{ab}} → Session(a, b, k_{ab}, n_a, n_b))
k_{ab} ∈ Key
n_b ∈ Nonce
m ∈ T



Modeling Security Protocols in CSP

- B's view (responder):
 - Message 1 b gets from 'a': $a.n_a$
 - Message 2 b sends to j : $b.\{a.n_a.n_b\}_{\text{ServerKey}(b)}$
 - Message 4 b gets from 'a': $\{a.k_{ab}\}_{\text{ServerKey}(b)} \cdot \{n_b\}_{k_{ab}}$
- In CSP this behavior can be modeled as follows:

Responder(b, n_b) =

[] (receive.a.b.a. $n_a \rightarrow$ send.b.J.b. $\{a.n_a.n_b\}_{\text{ServerKey}(b)}$
 $k_{ab} \in \text{Key} \rightarrow$ receive.a.b. $\{a.k_{ab}\}_{\text{ServerKey}(b)} \cdot \{n_b\}_{k_{ab}}$
 $n_b \in \text{Nonce} \rightarrow$ Session(b, a, k_{ab}, n_a, n_b))
 $m \in T$

Modeling Security Protocols in CSP

- **J's view (server):**

- Message 2 **j** gets from 'b': $b.\{a.n_a.n_b\}_{\text{ServerKey}(b)}$
- Message 3 **j** sends to **a**: $\{b.k_{ab}.n_a.n_b\}_{\text{ServerKey}(a)}.\{a.k_{ab}\}_{\text{ServerKey}(b)}$

- In CSP this behavior can be modeled as follows:

$\text{Server}(J, k_{ab}) =$

$[[\text{receive}.b.J.b.\{a.n_a.n_b\}_{\text{ServerKey}(b)}$

$A, B \in \text{Agent} \quad \rightarrow \text{send}.J.a.\{b.k_{ab}.n_a.n_b\}_{\text{ServerKey}(a)}.\{a.k_{ab}\}_{\text{ServerKey}(b)}$

$Nb, nb \in \text{Nonce} \quad \rightarrow \text{Server}(J, k_s))$

$\text{Server}(J) = ||| \text{Server}(J, k_{ab})$

$k_{ab} \in \text{Keys}_{\text{Server}}$

Question: why several server processes in parallel?



Modeling an intruder

- We want to model an intruder that represents all potential intruder behaviors

$\text{Intruder}(X) = \text{learn?m: messages} \rightarrow \text{Intruder}(\text{close}(X \cup \{m\}))$

$[]$

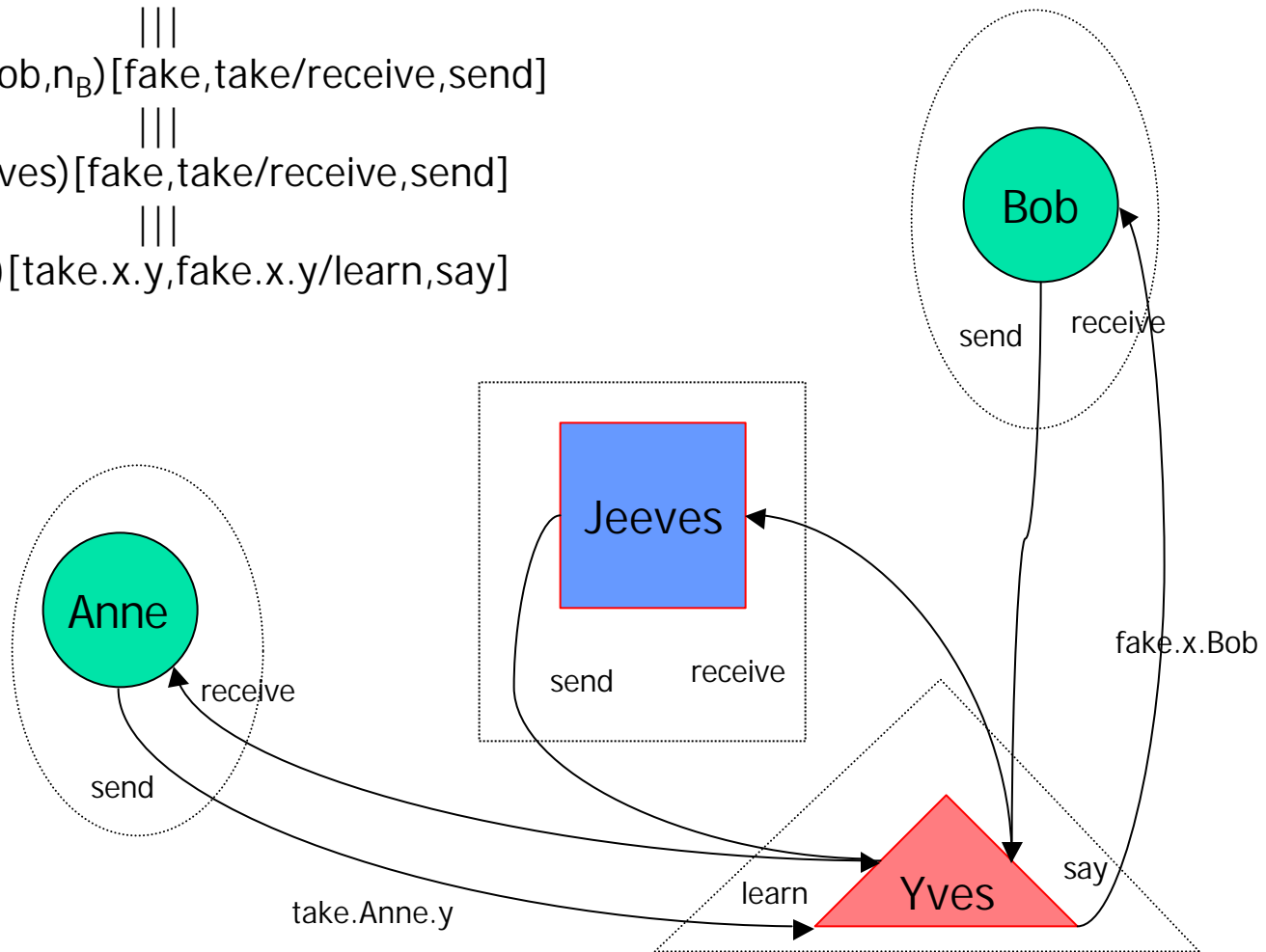
$\text{say!m: } X \wedge \text{messages} \rightarrow \text{Intruder}(X)$

- $\text{Close}(X)$ represents all the possible information that the attacker can infer from X . Typically we assume

- $\{k, m\} \vdash \text{encrypt}(k, m)$
- $\{\text{encrypt}(k, m), k^{-1}\} \vdash m$
- $\{\text{Sq}\langle x_1, \dots, x_n \rangle\} \vdash x_i$
- $\{x_1, \dots, x_n\} \vdash \text{Sq}\langle x_1, \dots, x_n \rangle$

Putting the network together

Initiator(Anne, n_A) [fake, take/receive, send]
|||
Responder(Bob, n_B) [fake, take/receive, send]
|||
Server(Jeeves) [fake, take/receive, send]
|||
Intruder(ϕ) [take.x.y, fake.x.y/learn, say]



Alternative with direct channels

