Notes on CSP for Protocol Analysis

CSG 399

February 19, 2006

CSP: communicating sequential processes.

- Calculus due to Hoare in the 80s for describing systems of communicating agents (called processes)
- At the conceptual level of the λ calculus.

CSP processes communicate via events (or actions). Let Σ be a set of events. (Kept abstract for now).

Processes are described by expressions in a particular grammar.

- *Stop* is a process
- If $a \in \Sigma$ and P is a process, then $a \rightarrow P$ is a process; intuitively, perform a, then behave as P.
- If $A \subseteq \Sigma$ is a set of actions, and P is a process, then $?x : A \rightarrow P$ is a process; intuitively, the process offers a choice of actions in A; the environment picks which to perform, and the process then behaves as P, where x is replaced b the chosen action in P.
- If P and Q are processes, then $P \,\Box\, Q$ is a process; intuitively, it offers a choice to the environment, which chooses whether the process behaves as P or Q (nondeterminism).

We add to the syntax in a bit. For now, note that we can define processes by recursion, by using declarations such as $N = P$, where N is a name, and P is a process. We can use the name of a process anywhere a process is expected.

Eg: $LOOP = a \rightarrow LOOP$.

We also allow parameterized definitions:

$$\begin{aligned} COUNT(0) &= up \rightarrow COUNT(1) \\ COUNT(n+1) &= up \rightarrow COUNT(n+2) \\ &\quad \Box\ down \rightarrow COUNT(n) \end{aligned}$$

We often want to actually communicate not just via actions, with values.

Assume integers — easy to extend to other kind of values, and even tuples of values. Let's do so.

We can define a special class of actions parameterized by a value, that is, actions of the form $c(v)$; for reasons that will become clear, we call these channels. In other words, a channel c in Σ is an element such that $c(v)$ is in Σ for every value v.

To talk about performing parameterized actions, we introduce some special syntax.

- If c is a channel in Σ and v is a value, then $c!v \to P$ is a process; intuitively, if performs action $c(v)$ and behaves as P.
- If c is a channel, V is a set of values, then $c?x : V \to P$ is a process; intuitively, it offers the choice to perform any of $c(v)$ for $v \in V$; the environment chooses, and the process then behaves as P, with x substituted by the value v chosen by the environment.

We write $c?x \to P$ for $c?x : V \to P$, where V is the set of all values. Whenever convenient, we use pattern-matching notation.

The above notation for processes is suitable for describing a single process — what about systems of many processes communicating with each other?

We introduce an operator that executes processes in parallel:

- If P and Q are processes, then $P \mid\mid Q$ is a process; intuitively, it behaves as both P and Q together; however, we require that P can only perform an action when Q performs exactly the same action (this is called a synchronization).

For example, consider the process $a \to b \to \mathit{Stop} \mid\mid a \to c \to \mathit{Stop}$. The processes on each side of $\mid\mid$ can perform an a, so they can proceed and behave as $b \to \mathit{Stop} \mid\mid c \to \mathit{Stop}$. At this point, the process is blocked, since the processes on each side of $\mid\mid$ can only perform different actions.

A more illuminating example is the following process: $c!v \to P \mid\mid c?x \to Q$. The left-hand side can perform $c(v)$ and become P, while the right-hand side can perform $c(v')$ for any v' and then become Q with x replaced by the v' chosen. Thus, both sides can perform $c(v)$, for the specific v given on the left-hand side. Thus, the process as a while can perform $c(v)$ and become $P \mid\mid Q[v/x]$, where $Q[v/x]$ denotes the process Q with every free occurrence of variable x replaced by v.

More flexibility is provided by the following operator:

- If P and Q are processes and $X \subseteq \Sigma$, then $P \mid\mid_X Q$ is a process; intuitively, this process behaves as P and Q interleaved, except that synchronization is required only for actions in X. The processes P and Q are free to perform any other action not in X.

We sometimes write $P \mid\mid\mid Q$ for $P \mid\mid_\varnothing Q$.

The following operator is useful to keep an action local to a process.

- If P is a process and $X \subseteq \Sigma$, then $P \backslash X$ is a process; intuitively, it behaves just like P, but the fact that the process performs an action in X is hidden from other processes.

The above gives a rather complex syntax for processes. How do we formally describe the behavior of processes in the above syntax? (i.e., what is the semantics?)

Many models of behavior for CSP have been developed. We focus on the simplest here.

Intuitively, we can think of a process as describing a transition system, where the process can perform one of a number of actions and become some other process. (The states of the transition system are the processes.)

Rather than directly give the transition system, we instead describe the finite traces in the transition system corresponding to the behavior of a process, and recoding only the actions performed. Because of nondeterminism, there can be more than one trace per process. We use the notation $\langle a_1, \ldots, a_k \rangle$ for traces, and we use $\frown$ for trace concatenation. We also write $s \upharpoonright A$ for the trace s where only elements of A are kept.

$$
\begin{aligned}
\mathit{traces}(\mathit{Stop}) &= \{\langle\rangle\} \\
\mathit{traces}(a \to P) &= \{\langle\rangle\} \cup \{\langle a\rangle \frown s \mid s \in \mathit{traces}(P)\} \\
\mathit{traces}(?x : A \to P) &= \{\langle\rangle\} \cup \{\langle a\rangle \frown s \mid a \in A, s \in \mathit{traces}(P)\} \\
\mathit{traces}(c!v \to P) &= \{\langle\rangle\} \cup \{\langle c(v)\rangle \frown s \mid s \in \mathit{traces}(P)\} \\
\mathit{traces}(c?x : V \to P) &= \{\langle\rangle\} \cup \{\langle c(v)\rangle \frown s \mid v \in V, s \in \mathit{traces}(P)\} \\
\mathit{traces}(P \,\Box\, Q) &= \mathit{traces}(P) \cup \mathit{traces}(Q) \\
\mathit{traces}(P \mid\mid Q) &= \mathit{traces}(P) \cap \mathit{traces}(Q) \\
\mathit{traces}(P \mid\mid_X Q) &= \cup_{s \in \mathit{traces}(P), t \in \mathit{traces}(Q)} \mathit{traces}(s \mid\mid_X t) \\
\mathit{traces}(P \backslash X) &= \{s \upharpoonright (\Sigma - X) \mid s \in \mathit{traces}(P)\}
\end{aligned}
$$

where $s \mid\mid_X t$ is defined as the set of all traces obtained by identifying all event from X in s and t, and arbitrarily interleaving all other events from s and t between identified events. (This definition can be found in the references on CSP, if you are really curious.)

What about recursive definitions? We use a fixed point formulation. Assume a recursive definition $N = F(N)$, where F is a process using name N. Define $F^n(N) = F(F(F(\ldots(F(N))\ldots)))$, the n-fold application of F to N. We define $\mathit{traces}(N)$ to be $\cup_{n \geq 0} \mathit{traces}(F^n(\mathit{Stop}))$.

Properties of processes in CSP are properties of traces. How do you specify a property? Refinement: $Q \sqsubseteq P$ (P refines Q) if $\mathit{traces}(P) \subseteq \mathit{traces}(Q)$.

Intuitively, in refinement, Q represents a set of possible executions; P refines Q if P can do some of the things that Q can do. Assume that Q represents all the traces that satisfy a property, then $Q \sqsubseteq P$ means that all the traces of P satisfy the property as well.

Why is this interesting? There are tools (e.g., FDR) that use model-checking techniques to establish that $Q \sqsubseteq P$ given P and Q. So if we can bring our specification under this form, we can use such tools.

Consider an example. Suppose the specification is whenever a process does a b, it has done an a immediately before.

Take $S = ?x : \Sigma - \{b\} \to S \,\Box\, a \to b \to S$.

Easy to check that $traces(S)$ is all traces where every b is preceded by an a.

So, if $S \sqsubseteq P$ for some process P, then $traces(P) \subseteq traces(S)$, meaning that every trace in P is such that every b is preceded by an a.

Modeling Protocols in CSP

Consider the NS protocol.

Assume a function $PKey(a)$ returning public key of agent $a \in A$.

$$\begin{aligned} INITIATOR(a, n_a) = recv?b : A \rightarrow send!\{a, n_a\}_{PKey(b)} \rightarrow \\ recv?\{n_a, x\}_{PKey(a)} \rightarrow send!\{x\}_{PKey(a)} \rightarrow SESSIONInit(a, b, n_a, x) \end{aligned}$$

This process receives who to talk to as the first message (could be made an argument); as it is, we let the adversary choose.

$$\begin{aligned} RESPONDER(b, n_b) = recv?\{x, y\}_{PKey(b)} \rightarrow send!\{y, n_b\}_{PKey(x)} \rightarrow \\ recv?\{n_b\}_{PKey(b)} \rightarrow SESSIONResp(b, x, n_b, y) \end{aligned}$$

Let N_I be a set of nonces for initiators, and N_R be a set of nonces of the responder.

$$USER_a = (|||_{n \in N_I}\ INITIATOR(a, n)) \ ||| \ (|||_{n \in N_R}\ RESPONDER(a, n))$$

Let Msg be the set of all messages.

Adversary:

$$\begin{aligned} ADV(X) = send?m : Msg \rightarrow ADV(\{m' \mid X \cup \{m\} \vdash m'\}) \\ \Box\ recv?m : X \rightarrow ADV(X) \end{aligned}$$

Putting the system together:

$$\begin{aligned} SYSTEM = (USER_{Alice} \ ||| \ USER_{Bob} \ ||| \ USER_{Charlie}) \\ ||_{\{send, recv\}}\ ADV(\{Alice, Bob, Charlie, PKey(Alice), PKey(Bob), PKey(Charlie)\}) \end{aligned}$$

How do we specify message secrecy?

- Add a new event "claim secret" for the agents
- Add a new event "leaked valued" for the adversary

Then, check that no claimed secret is ever leaked.

For example, to check the secrecy of the initiator nonce in NS:

$$INITIATOR'(a, n_a) = recv?b : A \rightarrow claimsecret!n_a \rightarrow send!\{a, n_a\}_{PKey(b)} \rightarrow \\ recv?\{n_a, x\}_{PKey(a)} \rightarrow send!\{x\}_{PKey(a)} \rightarrow SESSIONInit(a, b, n_a, x)$$

$$USER'_a = (|||_{n \in N_I}\ INITIATOR'(a, n)) \ ||| \ (|||_{n \in N_R}\ RESPONDER(a, n))$$

$$ADV'(X) = send?m : Msg \rightarrow ADV'(\{m' \mid X \cup \{m\} \vdash m'\}) \\ \Box\ recv?m : X \rightarrow ADV'(X) \\ \Box\ leak!m : X \rightarrow ADV'(X)$$

Putting the system together:

$$SYSTEM' = (USER'_{Alice} \ ||| \ USER'_{Bob} \ ||| \ USER'_{Charlie}) \\ ||_{\{send, recv\}}\ ADV'(\{Alice, Bob, Charlie, PKey(Alice), PKey(Bob), PKey(Charlie)\})$$

Message secrecy: in no trace where event $claimsecret(m)$ is performed is $leak(m)$ ever performed afterwards.

As a process, can be written as follows:

$$SECRECY_0(s) = claimsecret!s \rightarrow SECRECY_1(s) \\ \Box\ leak!s \rightarrow SECRECY_0(s)$$

$$SECRECY_1(s) = claimsecret!s \rightarrow SECRECY_1(s)$$

$$SECRECY(S) = (|||_{s \in S}\ SECRECY_0(s)) \ ||| \ (ANY(\Sigma \backslash \{claimsecret, leak\}))$$

where $ANY(X) = ?x : X \rightarrow ANY(X)$.

$SECRECY(S)$ is the set of all traces where there is no $leak(m)$ following a $claimsecret(m)$, for any $m \in S$.

So, the system satisfies message secrecy of initiator nonces if and only if $SECRECY(N_I) \sqsubseteq SYSTEM'$.

Something similar can be done for authentication. Instead of using claim/leak, we check for correspondence assertions, via events *begin* and *end*.

$$INITIATOR''(a, n_a) = recv?b : A \rightarrow begin!(a, b, n_a) \rightarrow send!\{a, n_a\}_{PKey(b)} \rightarrow \\ recv?\{n_a, x\}_{PKey(a)} \rightarrow send!\{x\}_{PKey(a)} \rightarrow SESSIONInit(a, b, n_a, x)$$

$$RESPONDER''(b, n_b) = recv?\{x, y\}_{PKey(b)} \rightarrow send!\{y, n_b\}_{PKey(x)} \rightarrow recv?\{n_b\}_{PKey(b)} \rightarrow \\ end!(x, b, y) \rightarrow SESSIONResp(b, x, n_b, y)$$

$$USER''_a = (|||_{n \in N_I}\ INITIATOR''(a,n)) \ |||\ (|||_{n \in N_R}\ RESPONDER''(a,n))$$

Putting the system together:

$$\begin{aligned} SYSTEM'' = (&USER''_{Alice} \ |||\ USER''_{Bob} \ |||\ USER''_{Charlie}) \\ &||_{\{send,recv\}}\ ADV(\{Alice, Bob, Charlie, PKey(Alice), PKey(Bob), PKey(Charlie)\}) \end{aligned}$$

What is the specification? Every trace that performs $end(x,y,z)$ has performed a distinct $begin(x,y,z)$ beforehand.

$$\begin{aligned} CORR_0(x,y,z) &= begin!(x,y,z) \rightarrow CORR_1(x,y,z) \\ CORR_{n+1}(x,y,z) &= begin!(x,y,z) \rightarrow CORR_{n+2}(x,y,z) \\ &\quad\ end!(x,y,z) \rightarrow CORR_n(x,y,z) \end{aligned}$$

$$CORR = (|||_{a \in A, b \in A, n \in N_I}\ CORR_0(a,b,n)) \ |||\ ANY(\Sigma \backslash \{begin, end\})$$

So, a system satisfies the correspondence assertion if and only if $CORR \sqsubseteq SYSTEM''$.