Architecture modeling in Calculus of Communicating Systems (CCS) Structure and Interactions

Rushikesh K Joshi

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

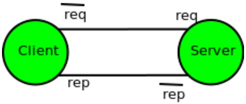
Outline

Outline

1. About the CCS Approach
2. The calculus
3. Semantics
4. Readings

Components and Connections between them

- Components are seen as Agents in CCS
- No separate abstraction is provided for connections
- If connections need to have behavior of their own, they are modeled as agents.
- A send of an agent and a corresponding receive act together as an indivisible (Atomic) action.. there is no delay or separation between them.



Abilities of CCS

- Agent are expressed through agent expressions
- Agents have input and output ports
- Agents perform input and output actions on ports
- Agent Expressions can be sequences of these actions
- Agent Expressions make use of non-determinism
- Agents can be composed together to form bigger systems and so on
- Before composing a system with another, some ports can be hidden
- Before composing a system with another, some ports can be renamed

Outline

1. About the CCS Approach
2. The calculus
3. Semantics
4. Readings

Agent expressions: actions, sequences and connections

$$\begin{aligned} Client &= \overline{req}.rep.Client \\ Server &= req.\overline{rep}.Server \\ System &= Client|Server \end{aligned}$$

- A non-terminating system of client and server
- rep, req are **input actions** on input ports
- $\overline{rep}$, $\overline{req}$ are **output actions** on output ports
- *dot* operator called **prefix combinator** makes a sequences of actions
- $|$ operator called **composition combinator** makes a composition of two agents, connecting the corresponding input and output ports

Agent expressions: Non-determinism

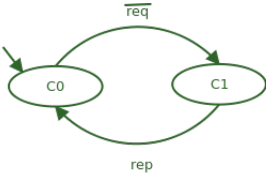
$$\begin{aligned} Client_1 &= \overline{req_1}.rep_1.Client_1 \\ Client_2 &= \overline{req_2}.rep_2.Client_2 \\ Server &= (req_1.\overline{rep_1} + req_2.\overline{rep_2}).Server \\ System &= Client_1 | Client_2 | Server \end{aligned}$$

- A non-terminating system of 2 clients and a server
- The server may pick up any of its inputs
- + is the **summation combinator** which represents a non-deterministic choice between two agent sub-expressions. Once a choice is made, the expression must be completely executed.

Outline

The state machine (transition diagram) of the client

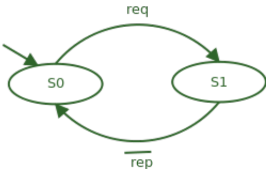
$$Client = \overline{req}.rep.Client$$



$$\overline{req} \rightarrow rep \rightarrow \overline{req} \rightarrow rep \rightarrow \ldots$$

The state machine of the server

$$Server = req.\overline{rep}.Server$$

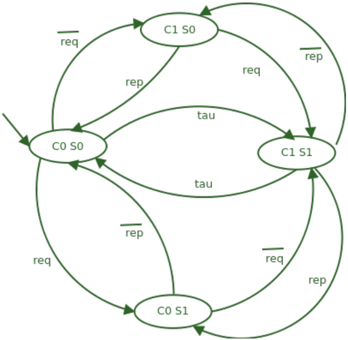


$$req \rightarrow \overline{rep} \rightarrow req \rightarrow \overline{rep} \rightarrow \ldots$$

The state machine of the composition

$Client = \overline{req}.rep.Client \qquad Server = req.\overline{rep}.Server$

$System = Client|Server$



- Client and Server may proceed independently or communicate via corresponding actions

Transitions including τ actions

$$A = p.A' \qquad A' = \overline{q}.A \qquad B = q.B' \qquad B' = r.B$$

$$System = (A|B)$$

- We know that $A \xrightarrow{p} A'$, $B \xrightarrow{q} B'$, $A' \xrightarrow{\overline{q}} A$, and $B' \xrightarrow{r} B$,
- So $A|B \xrightarrow{p} A'|B$. Similarly, $A|B \xrightarrow{q} A|B'$
- Also $A'|B \xrightarrow{\overline{q}} A|B$. Similarly, $A'|B \xrightarrow{q} A'|B'$
- We also have a τ action, due to which, $A'|B \xrightarrow{\tau} A|B'$
- Similarly, work out other possible transitions?

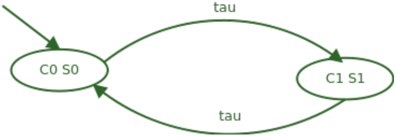
τ actions

- τ action represents a handshake
- It's a perfect (completed) action
- It is not visible like the other actions that are visible
- Visible actions can be used in a subsequent composition with another agent
- τ action is also called *unobservable action*
- Whenever a pair of complementary actions $(a, \overline{a})$ is possible in a composite agent, a τ action is possible

The Restriction Operator '\'

$Client = \overline{req}.rep.Client \qquad Server = req.\overline{rep}.Server$

$System = (Client|Server)\backslash\{req, rep\}$



- Independent actions $req, \overline{req}, rep, \overline{rep}$ are restricted (prohibited), only the τ actions occur inside the composition.
- The restricted ports are also not available for further composition with other agents
- Both input and output ports corresponding to names in restriction set are restricted

Exercise

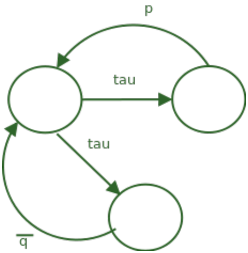
$$UI = input.\overline{rpc\_request}.rpc\_reply.\overline{print\_result}.0$$

$$BL = rpc\_request.\overline{log\_request}.\overline{rpc\_reply}.0$$

$$System = (UI|BL)\backslash\{rpc\_request, rpc\_reply\}$$

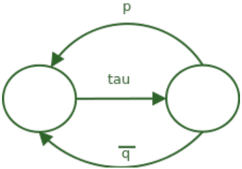
Build the transition diagram (state machine) for agent 'System'?

Exercise



Build CCS agent expressions which result in the above State transition diagram. ?

Exercise



Build CCS agent expressions which result in the above State transition diagram. ?

Exercises

Build CCS expressions representing the following architectural patterns: (1) OR split (2) OR join (2) AND split (3) AND join (4) MVC (5) 3-tiered architecture (6) Semaphore Synchronization

Exercise: Semaphore Synchronization- fill in the blanks?

$$Sem = \text{..............}??$$

$$Client\_1 = \overline{p}.start\_print.end\_print.\overline{v}.Client\_1$$

$$Client\_2 = \text{............}??$$

$$System = (Client\_1 | Client\_2 | Sem) \backslash \{\text{............}??\}$$

Value passing CCS

$$
\begin{aligned}
Client_1 &= \overline{req(1)}.rep_1.Client_1 \\
Client_2 &= \overline{req(2)}.rep_2.Client_2 \\
Server &= req(v).if(v = 1)\ \overline{rep_1}.Server\ else\ \overline{rep_2}.Server \\
System &= Client_1 | Client_2 | Server
\end{aligned}
$$

- we can eliminate some ports

Relabeling of Agents

$$
\begin{aligned}
Client_1 &= \overline{req_1}.rep_1.Client_1 \\
Client_2 &= Client_1[req_2/req_1, rep_2/rep_1] \\
Server &= req_1.\overline{rep_1}.Server + req_2.\overline{rep_2}.Server \\
System &= Client_1|Client_2|Server
\end{aligned}
$$

- we can reuse agent expressions

Agent that diverts the odds from the evens

$$Diverter = req(v).if\ (v\%2)\overline{req_1}.Diverter\ else\ \overline{req_2}.Diverter$$

Abstracting the Diverter by removing value passing

$$
\begin{aligned}
Diverter &= req(v).Diverter' \\
Diverter' &= \overline{req_1}.Diverter + \overline{req_2}.Diverter
\end{aligned}
$$

- In the architectural abstraction, we bring in all possibilities and remove computation (as much as possible).
- Abstract out conditional interactions as possibilities through non-determinism

Outline

1. About the CCS Approach
2. The calculus
3. Semantics
4. Readings

Readings

- Robin Milner, Communication and Concurrency, Prentice Hall, 1989.
- David Walker, Introduction to Calculus of Communicating Systems, Technical Report, University of Edinburgh, 1987.