T-79.231 Parallel and Distributed Digital Systems

Process Algebra

Teemu Tynjälä

November 18, 2002

Petri Nets Against Process Algebra

- Petri Nets represent parallel and distributed digital systems graphically.
- Unfortunately large systems are not amenable to the graphical representation. Today's high performance Petri Net tools use textual representation for systems instead.
- Process Algebra is by its very definition a text based method of representing parallel and distributed digital systems.

Process Algebra was invented in Great Britain during the early 80's and its fathers were Hoare and Milner. In this lecture I will introduce CCS Process Algebra (Calculus of Communicating Systems), which was published by Milner in 1980.

Process Algebras in general (1/2)

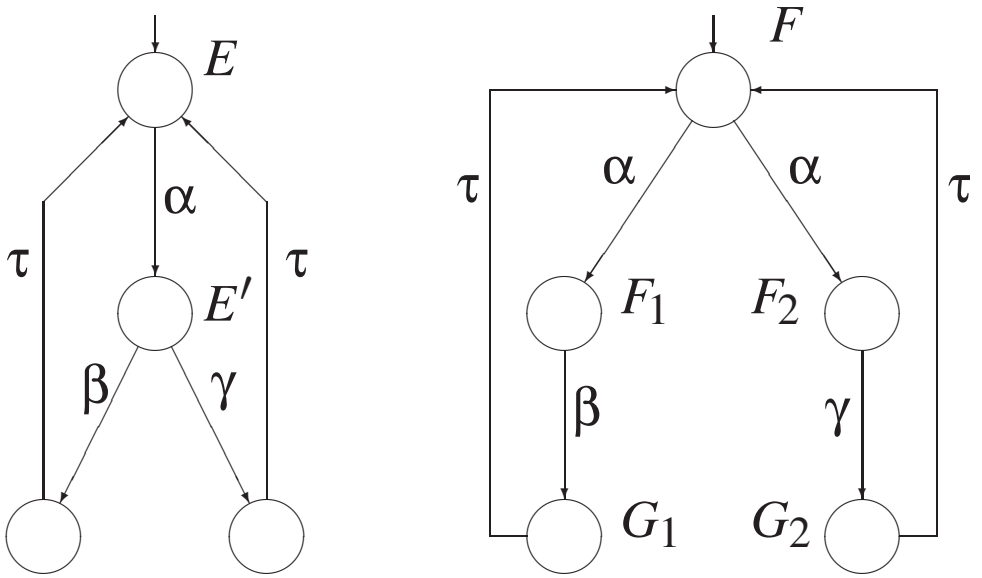
- The central concept in process algebra is that of an agent. An agent has an “in” port from which it reads messages from the environment, and an “out” port to which it outputs messages.
- Quite often an agent works as follows:
  1. An agent reads message(s) from its “in” port
  2. An agent performs some internal computation
  3. Finally, an agent outputs messages to its “out” port and starts waiting for the next message
- Usually process algebra is used to analyse systems which consist of several mutually communicating agents.

Process Algebras in General (2/2)

- The biggest strength of process algebra as compared with Petri nets is their compositionality. New agents may be formed from existing agents by using process algebraic operations on them. For example, one can create a new agent by connecting two existing agents in parallel.

- Process algebra provides the capability of checking whether two agents are observationally equivalent.

- Process algebra enables its user to represent systems either by text or by simple diagrams. In the next slide, I give examples of two agents specified using both text and diagrams.



The agent at left can be described with the equation $E ::= \alpha.(\beta.\tau + \gamma.\tau)$, and the agent at right with the equation $F ::= \alpha.\beta.\tau + \alpha.\gamma.\tau$.

How Does CCS Work? (1/2)

- Each agent has a name. Names are usually capital letters, $P, Q, R, A, B, C, \ldots$
- Process algebra must define *s*et of visible events. Visible events are ones that agents perform and which outside observers see. Generally visible events are defined as $Act = \{\alpha, \overline{\alpha}, \beta, \overline{\beta}, \gamma, \overline{\gamma}, \ldots\}$.
- Each action has a *c*omplementary action. If α means the reception of message α from the "in" port, the action $\overline{\alpha}$ denotes the writing of the message to the "out" port. The following identity describes how the complementation of messages works: $\overline{\overline{\alpha}} = \alpha$.
- In CCS we also have the invisible event, τ, which an outside observer can not see.

How Does CCS Work? (2/2)

- In process algebra the question of importance is how an agent behaves after a certain event takes place. If we write $E \xrightarrow{\alpha} E'$, agent E behaves as E' after the action α.

- An agent can also execute the invisible event. In this case we write $G \xrightarrow{\tau} G'$. Now the outside observer does not see anything.

How Are Agents Constructed? (1/6)

Agents may be constructed by employing process algebraic operations on other agents. In the following we describe five different methods of constructing agents.

(Action Prefixing) If we want that our new agent behaves as E does with the exception that it first performs the action α, we can define

$$F ::= \alpha.E$$

For this agent the following also holds: $\alpha.E \xrightarrow{\alpha} E$

How Can Agents Be Constructed? (2/6)

(Choice) If we want our new agent to behave as either one of previously defined agents, we can write

$$C ::= A + B$$

The semantics of this non-deterministic choice can be divided to two cases:

1. If $A \xrightarrow{\alpha} A'$, then $A + B \xrightarrow{\alpha} A'$
2. If $B \xrightarrow{\alpha} B'$, then $A + B \xrightarrow{\alpha} B'$

How Can Agents Be Constructed? (3/6)

(Parallel Composition) If two agents are composed in parallel, a new agent is formed and it is denoted by

$$L ::= M||N$$

The semantics of parallel composition are:

1. If $M \xrightarrow{\alpha} M'$, and N can not perform $\overline{\alpha}$, then $M||N \xrightarrow{\alpha} M'||N$.
2. If $N \xrightarrow{\alpha} N'$ and M can not perform $\overline{\alpha}$, then $M||N \xrightarrow{\alpha} M||N'$
3. If $M \xrightarrow{\alpha} M'$, and $N \xrightarrow{\overline{\alpha}} N'$, then $M||N \xrightarrow{\tau} M'||N'$. This is called synchronisation

How Can Agents Be Constructed? (4/6)

(Restriction) If we want to construct an agent which is identical to another with the exception that we do not allow certain actions to take place, we write

$$T = S \setminus \text{Restr}$$

where Restr is the set of actions we do not allow T to perform. It should be noted that if certain action is in Restr, its co-action belongs to the set as well.

The semantics of this operator are as follows: If $S \xrightarrow{\alpha} S'$ and $\alpha, \overline{\alpha} \notin \text{Restr}$, we may write $S \setminus \text{Restr} \xrightarrow{\alpha} S' \setminus \text{Restr}$.

How Can Agents Be Constructed? (5/6)

(Relabelling) If we give new names to the actions of an agent J by utilising the function $m : Act \rightarrow Act : m(\overline{\alpha}) = \overline{m(\alpha)}$, we get a new agent K as a result, and it is written as follows

$$K ::= J[m]$$

The condition on function m simply means that if m maps α into β, it must map $\overline{\alpha}$ into $\overline{\beta}$.

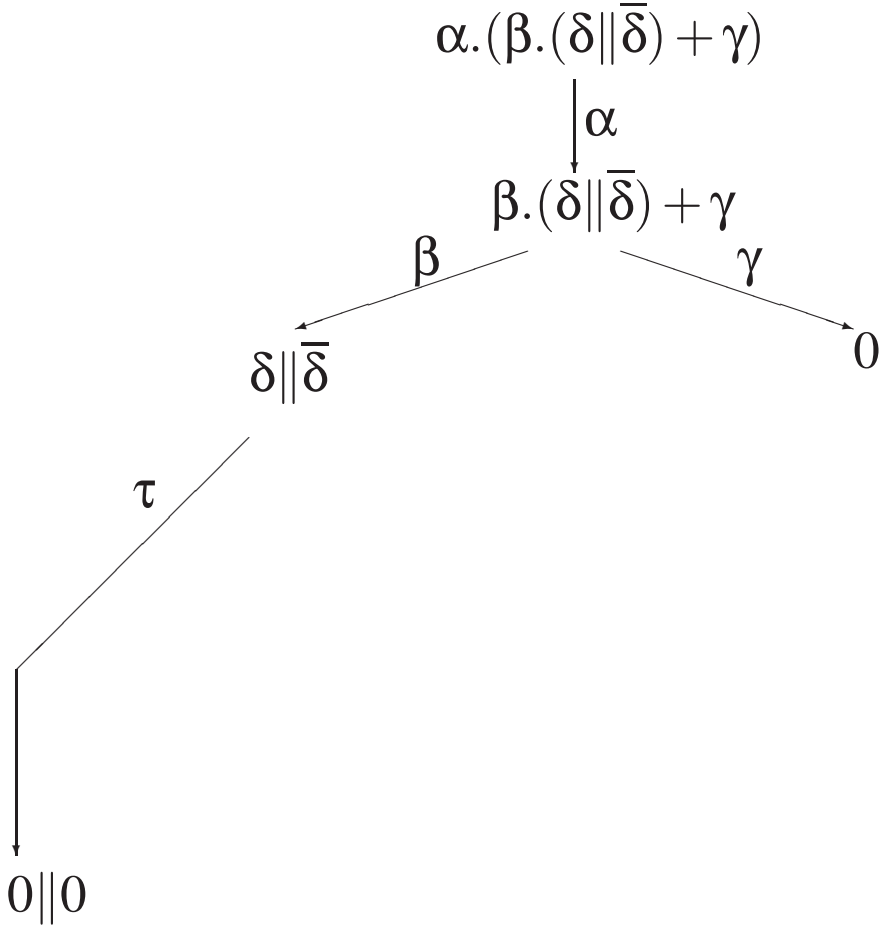
The semantics of relabelling is as follows: If $E \xrightarrow{\alpha} E'$, then $E[m] \xrightarrow{m(\alpha)} E'[m]$.

How Can Agents Be Constructed? (6/6)

- The previous five operations are useful in combining simple agents into more complex agents.
- If we are faced with the chaining of previous operations, it may be simpler to describe the situation with process algebraic equations.
- If $E ::= \alpha.E'$, and $E' ::= \beta.\tau.E + \gamma.\tau.E$, we may combine these two equations into a single one:

$$E ::= \alpha.(\beta.\tau.E + \gamma.\tau.E)$$

On the following page we have translated the agent $\alpha.(\beta.(\delta||\overline{\delta}) + \gamma)$ into a diagram. In the translation process we have employed the rules of operations described above. Each state is labelled with a process algebraic expression that describes the agent's operation from that state onwards.


$\alpha.(\beta.(\delta \| \overline{\delta}) + \gamma)$
α
$\beta.(\delta \| \overline{\delta}) + \gamma$
β
γ
$\delta \| \overline{\delta}$
0
τ
$0 \| 0$

Process Algebraic Equivalences (1/3)

- CCS defines several notions of equivalence, which may be used to compare agents
- We will introduce trace equivalence, failure equivalence and simulation equivalence.

(Trace equivalence) Let P and Q be two agents. If the sequences of events that P and Q may execute are the same, the two agents are trace equivalent.

For example the traces of $\alpha.(\beta + \gamma)$ are $\{\langle\alpha,\beta\rangle, \langle\alpha,\gamma\rangle\}$. How about the traces of $\alpha.\beta + \alpha.(\beta + \gamma)$? They are also $\{\langle\alpha,\beta\rangle, \langle\alpha,\gamma\rangle\}$. Thus the two agents are trace equivalent.

Process Algebraic Equivalences (2/3)

(Failure Equivalence) The *failure* of an agent P is a pair $\langle \sigma, X \rangle$ meaning that starting from agent P and by executing trace σ we end up with another agent Q, where none of the actions in set X is possible. If all the failures of two agents are the same, these two agents are called failure equivalent.

The failures of agent $\alpha.(\beta + \gamma)$ are $\langle \langle \alpha \rangle, \{\alpha\} \rangle$, $\langle \langle \alpha, \beta \rangle, \{\alpha, \beta, \gamma\} \rangle$ and $\langle \langle \alpha, \gamma \rangle, \{\alpha, \beta, \gamma\} \rangle$.

Similarly the failures of agent $\alpha.\beta + \alpha.(\beta + \gamma)$ are $\langle \langle \alpha \rangle, \{\alpha, \gamma\} \rangle$, $\langle \langle \alpha \rangle, \{\alpha\} \rangle$, $\langle \langle \alpha, \beta \rangle, \{\alpha, \beta, \gamma\} \rangle$ and $\langle \langle \alpha, \gamma \rangle, \{\alpha, \beta, \gamma\} \rangle$. As can be seen, these two agents are not failure equivalent, although they are trace equivalent.

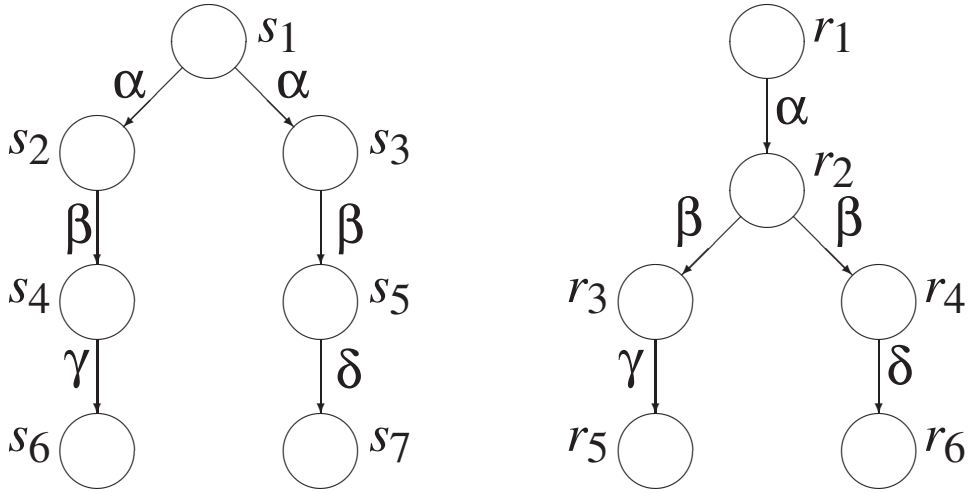
Process Algebraic Equivalences (3/3)

(Simulation Equivalence) An agent F simulates an agent E if there exists a binary relation $\mathcal{R} \subseteq S \times S$ defined over all agents, which satisfies

1. $E \; \mathcal{R} \; F$, that is, $(E, F) \in \mathcal{R}$
2. If $E' \; \mathcal{R} \; F'$ and $E' \xrightarrow{\alpha} E''$, there exists F'' such that $F' \xrightarrow{\alpha} F''$ and $E'' \; \mathcal{R} \; F''$

In the following slide we have computed the simulation relation between two agents and thus show that the two are simulation equivalent.

Example for Simulation Equivalence



Simulation relation $\mathcal{R}$ is $\{(s_1, r_1), (s_2, r_2), (s_4, r_3), (s_6, r_5), (s_3, r_2), (s_5, r_4), (s_7, r_6)\}$.

Conclusion

- Process Algebra is a textual way of describing the behaviour of agents.
- Agents may be combined and defined part by part by employing process algebraic operations.
- Agents may also be described with groups of process algebraic equations.
- Two agents may be efficiently compared using different equivalence relations. We treated three notions of equivalence: Trace equivalence, failure equivalence and simulation equivalence.