Process Algebra

- Modelling formalism
- Useful for showing relations of different systems
  - simulation
  - refinement
  - implementation
- System is composed of *Agents*

Based on the chapter 8 of the book Software Reliability Methods,

Doron A. Peled, Springer 2001

Agents in Process Algebra

- An agent is an abstract description of a system
- Agents are described as terms and operations for combining terms
- Agents can evolve into a different agent by executing *actions*
- The system state changes as the agents change

- The set of *visible actions* is denoted by Act

- Agents can have *internal actions* $\tau \notin Act$ not visible to outside observer

- For each action there exists a co-action $\overline{\alpha}$, $\overline{\overline{\alpha}} = \alpha$

  - A co-action can be executed synchronously with the corresponding action.

- $E \stackrel{\alpha}{\longrightarrow} E'$: agent E evolves into E' by executing action α , $\alpha \in Act \cup \{\tau\}$

- $E \stackrel{\alpha}{\Longrightarrow} E'$: agent E evolves into E' by executing α and any number of τ actions before or after α

Semantics of actions

- Action prefixing: $\alpha.A \xrightarrow{\alpha} A$

- Choice: $\dfrac{E \xrightarrow{\alpha} E'}{E+F \xrightarrow{\alpha} E'}$ $\quad$ $\dfrac{F \xrightarrow{\alpha} F'}{E+F \xrightarrow{\alpha} F'}$

- Concurrent composition:

$$\frac{E \xrightarrow{\alpha} E'}{E \parallel F \xrightarrow{\alpha} E' \parallel F}$$

$$\frac{F \xrightarrow{\alpha} F'}{E \parallel F \xrightarrow{\alpha} E \parallel F'}$$

$$\frac{E \xrightarrow{\alpha} E', F \xrightarrow{\overline{\alpha}} F'}{E \parallel F \xrightarrow{\tau} E' \parallel F'}$$

- Restriction: Let $R \subseteq Act$ be a set of disallowed actions

$$\frac{E \xrightarrow{\alpha} E', \alpha, \overline{\alpha} \notin R}{E \backslash R \xrightarrow{\alpha} E' \backslash R}$$

- Relabeling:Let $m : Act \rightarrow Act, m(\overline{\alpha}) = \overline{m(\alpha)}$

$$\frac{E \xrightarrow{\alpha} E'}{E[m] \xrightarrow{m(\alpha)} E'[m]}$$

- Equational definition: allows the definition of agents with infinite behavior

$$\frac{E \xrightarrow{\alpha} E', A \triangleq E}{A \xrightarrow{\alpha} E'}$$

- The agent 0 (Nil) has no rules, and does not allow any action

Value Passing Agents

- Actions can be parametrised

- $a(x).E$: agent takes parameter x as input when it performs action a

- $\overline{a}(y).F$: agent outputs the value y when the co-action $\overline{a}(y)$ is performed

- When the actions a and $\overline{a}$ are performed, the value of y will be bound to x in agent E

Modelling Dekker in CCS

- Variables have only 2 possible values, so we can model them in the following way:

  $$C1\_0 \triangleq \overline{is\_0\_C1?}.C1\_0 + \overline{set\_1\_C1}.C1\_1 + \overline{set\_0\_C1}.C1\_0$$

  $$C1\_1 \triangleq \overline{is\_1\_C1?}.C1\_1 + \overline{set\_0\_C1}.C1\_0 + \overline{set\_1\_C1}.C1\_1$$

- The processes $P1$ and $P2$ use the choice operator to select actions based on variable values.

- The processes change the values of the variables by the co-actions of the actions of the variable agents.

- The variables $C2$ and $Turn$ are defined in a similar fashion

- Definition of $P1$:

$$\begin{aligned}
P1 &\triangleq noncritical1.set_0_C1.INSIDELOOP1 \\
INSIDELOOP1 &\triangleq is_1_C2?.REST1 + \\
&\quad is_0_C2?. \\
&\quad (is_2_Turn?.set_1_C1.INNERMOST1 + \\
&\quad is_1_Turn?.INSIDELOOP1) \\
INNERMOST1 &\triangleq is_2_Turn?.INNERMOST1 + \\
&\quad is_1_Turn?.set_0_C1.INSIDELOOP1 \\
REST1 &\triangleq critical1.set_1_C1.set_2_Turn.P1
\end{aligned}$$

- The set of restricted actions R contains all the actions related to the variables

- After defining the variables and processes we can define the Dekker's algorithm:
$DEKKER \triangleq (C1\_1 \parallel C2\_1 \parallel TURN\_1 \parallel P1 \parallel P2) \backslash R$

- The *wait* statements in the processes have been modeled to use busy waiting

- A process is assumed to always complete its noncritical or critical section

- If the model is verified without fairness assumptions, a situation where one process is unable to proceed will be found (a process can loop forever in the busy waiting state)

Equivalences between agents

- There are different ways to compare agents.

- Each criterion is an equivalence relation, and the relations form a partial order. Here we will discuss *trace, failure, simulation* and *bisimulation* equivalences.

- Equivalences can be used to make models smaller

- Each equivalence class preserves certain types of properties (e.g. LTL)

- If we can find a smaller model that is in some sense equivalent to an another model, we can use the smaller model to verify properties.

Trace equivalence

- A *trace* is a finite sequence of states that can be performed from a given agent

- $T(E)$ is the set of all traces of agent E

- E is *trace equivalent* with F, $E \equiv_{tr} F$, when $T(E) = T(F)$

- $T(E) = T(F)$ also implies that the infinite traces of E and F are the same

- Preserves properties that can be specified using LTL

Failure equivalence

- A *failure* of an agent E is $< \sigma, X >$, where σ is such a trace of agent E leading to agent E' that none of the actions of X are enabled in E'. A failure of E can also contain disabled actions not in X

- Due to nondeterminism, it is possible to execute the actions in σ and get an agent E'' where some actions in X are enabled

- If $Fail(E)$ is the set of failures of E, we say that E and F are failure equivalent, $E \equiv_{fl} F$ if $Fail(E) = Fail(F)$

- Preserves deadlocks

Simulation equivalence

- We define S a set of agents and Act a set of actions

- An agent E simulates F if there exists a relation $R \subseteq S \times S$ satisfying

  1. $E\mathcal{R}F$

  2. If $E'\mathcal{R}F'$ and $E' \xrightarrow{\alpha} E''$, there exists a F'' such that $F' \xrightarrow{\alpha} F''$ and $E''\mathcal{R}F''$

- If E simulates F with $\mathcal{R}$ and F simulates E with $\mathcal{Q}$ we write $E \equiv_{sim} F$. $\mathcal{R} = \mathcal{Q}^{-1}$ is *not* required

- Preserves properties that can be specified using CTL

- Consider the agents $A \triangleq \alpha.(\beta.\gamma.Nil + \beta.\delta.Nil)$ and $B \triangleq \alpha.\beta.\gamma.Nil + \alpha.\beta.\delta.Nil$

- The agent B can simulate A with $\mathcal{R} = \{(s_1, r_1), (s_2, r_2), (s_4, r_3), (s_6, r_5), (s_3, r_2), (s_5, r_4), (s_7, r_6)\}$

- However, A cannot simulate B, because there is no state that could simulate r_2

  - If we choose $r_2 \mathcal{Q} s_2$ we would get also $r_3 \mathcal{Q} s_4$ and $r_4 \mathcal{Q} s_4$. $r_4 \mathcal{Q} s_4$ will not do, because δ is not enabled in s_4.

  - The reasoning for $r_2 \mathcal{Q} s_3$ is similar

Bisimulation equivalence

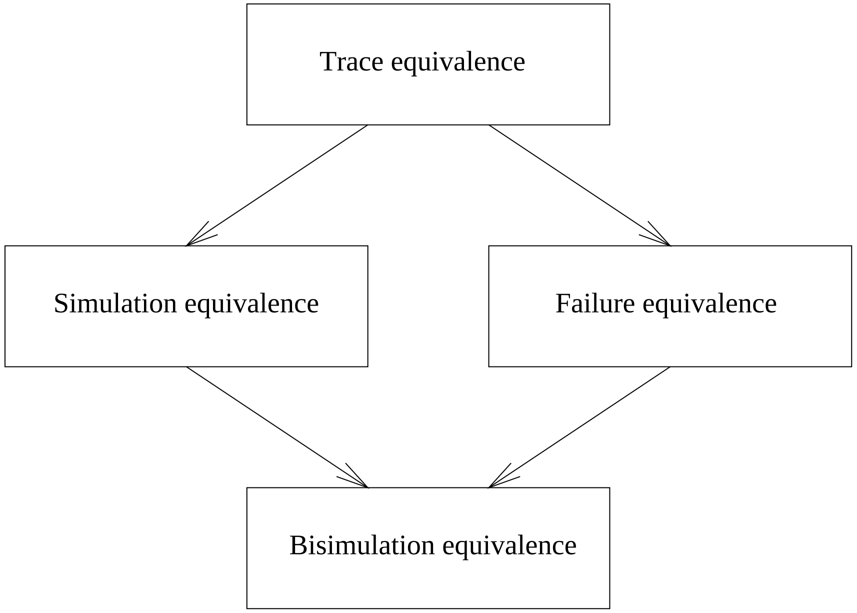
- For agents E and F to be bisimilar, $E \equiv_{bis} F$, we require that the relations $\mathcal{R}$ and $\mathcal{Q}$ used in simulation equivalence satisfy $\mathcal{Q} = \mathcal{R}^{-1}$

- Bisimulation is a refinement of both simulation and failure equivalence

- Weak bisimulation abstracts away the invisible actions in the agents

- Preserves properties that can be specified using CTL*

A hierarchy of equivalences



This hierarchy depends on the fact that actions can be nondeterministic. If we do not allow nondeterministic actions the hierarchy collapses. Note that we can still have nondeterministic choice, we need only to prohibit actions that can lead to several successor agents (e.g. $\alpha.\beta + \alpha.\gamma$ not allowed, but $\alpha + \beta$ is ok)

Concurrency and nondeterministic choice

- Let $E = \alpha \parallel \beta.\gamma$ and $E' = \alpha.\beta.\gamma + \beta.\alpha.\gamma + \beta.\gamma.\alpha$. Now $E \equiv_{bis} E'$ because concurrent actions use interleaving.

- Let $F = \alpha \parallel \delta$ and $F' = \alpha.\delta + \delta.\alpha$. If we refine δ in F to be $\beta.\gamma$, the result is the agent E. However, if we refine F' in the same fashion, the resulting agent is $C = \alpha.\beta.\gamma + \beta.\gamma.\alpha$ and $C \not\equiv E$