1 Weak Bisimuluation Discussion

1.1 Processes P and D

Building on our discussion of weak bisimulation from last class, we return to the following two processes from last week, P and D, shown in Figures 1 and 2 respectively.

$$
\begin{aligned}
A &= a \cdot A' \\
A' &= \bar{b} \cdot A \\
B &= b \cdot B' \\
B' &= \bar{c} \cdot B \\
P &= \text{new } b(A|B)
\end{aligned}
$$

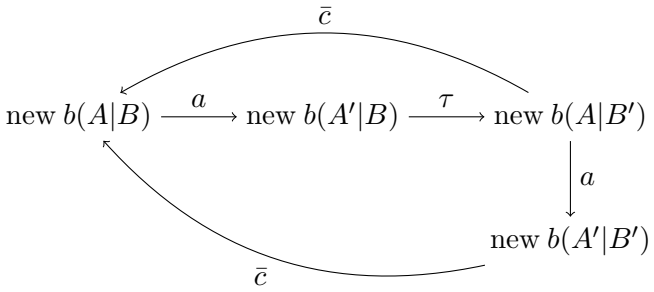

Figure 1: Process P and its transition graph

$$
\begin{aligned}
D &= a \cdot \tau \cdot D' \\
D' &= a \cdot D'' + \bar{c} \cdot D \\
D'' &= \bar{c} \cdot \tau \cdot D'
\end{aligned}
$$

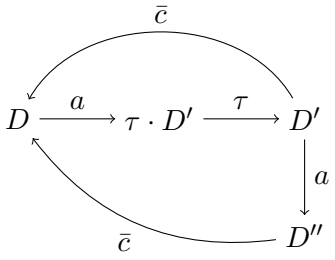

Figure 2: Process D and its transition graph

1.2 Showing Weak Bisimulation

We showed last week that $P \sim D$, i.e they are strong bisimulations of one another. This should be fairly obvious from inspecting their transition graphs. We'd like to show that the following process E is weak bisimulation of both P and D, i.e. $E \approx P$ and $E \approx D$.

We can do this by proving that $E \approx P$. Recall that the whole point of weak bisimulation is to capture the specification of a process without having to deal with the implementation details, i.e. internal transitions.

E is shown below in Figure 3.

$$
\begin{aligned}
E &= a \cdot E' \\
E' &= a \cdot E'' + \bar{c} \cdot E \\
E'' &= \bar{c} \cdot E'
\end{aligned}
$$

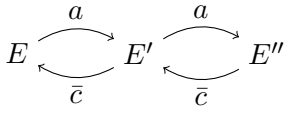

Figure 3: Process E and its transition graph

Looking at the transition systems for P and E in Figures 1 and 3, it's apparent that they aren't strong bisimulations of one another, but it's not clear that they are weak bisimulations. In order to *prove* that they are weak bisimulations of one another, we have to find the set S of pairs of states that satisfy the weak bisimulation criteria shown in Figure 4.

Figure 4: Weak Bisimulation Criteria

We can now construct S using Figures 1, 3, and 4 as follows:

$$
\begin{aligned}
S = \{&(\text{new } b(A|B), E), \\
&(\text{new } b(A'|B), E'), \\
&(\text{new } b(A|B'), E'), \\
&(\text{new } b(A'|B'), E'')\}
\end{aligned}
$$

So long as the pairs in set S are representative of the states of the processes and they satisfy the criteria from Figure 4, we can say that $E \approx P$. This in turn implies that $E \approx D$ as well.

1.3 Additional Examples

One might be tempted to assume that we can simply collapse any τ transitions in a process in order to show weak bisimulation. While this intuition is a good starting point, it does NOT hold in all cases, as in Figure 5 where $P \not\approx Q \not\approx R$.

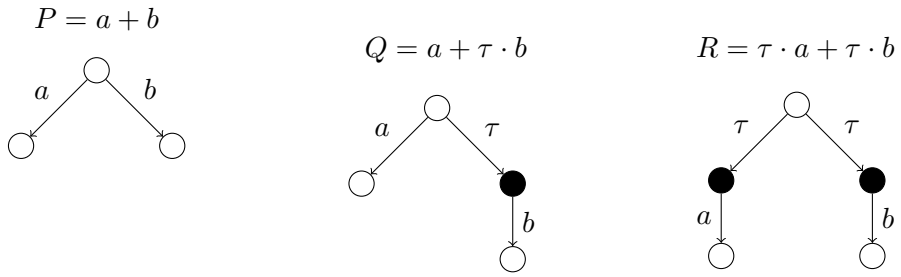

Figure 5: Transition Graphs for P, Q, and R. Filled in nodes denote states which cannot bisimulate states of P.

In Figure 5, we claim that $P \not\approx Q$. We'll now show a proof sketch by contradiction.
Suppose $P \approx Q$. Then $\exists$ a weak bisimulation S such that PSQ. This would imply the following:

- $Q \to b \cdot 0$
- $\exists P' \text{such that } P \Rightarrow P' \text{and } PSb \cdot 0$
- $P = P' \text{implies } PSb \cdot 0$

However, P can do $P \xrightarrow{a} 0$ and $\stackrel{a}{\Rightarrow}$ doesn't exist, therefore S is not a weak bisimulation and $P \not\approx Q$.

1.4 Weak Bisimulation as a *Process Congruence*

The following are facts:

- $P \approx \tau \cdot P$
- $M + N + \tau \cdot N \approx M + \tau \cdot N \Rightarrow N + \tau \cdot N \approx \tau \cdot N$
- $M + \alpha \cdot P + \alpha \cdot (\tau \cdot P + N) \approx M + \alpha \cdot (\tau \cdot P + N)$

Furthermore, $P \approx Q$ implies:

- $\alpha \cdot P + M \approx \alpha \cdot Q + M$
- $\text{new } a \; P \approx \text{new } a \; Q$
- $P|R \approx Q|R$
- $R|P \approx R|Q$

Note that $N_1 \approx N_2 \not\Rightarrow M + N_1 \approx M + N_2$. A counter example can be seen in Figure 5 or in the following system:

$$N_1 = b$$
$$N_2 = \tau \cdot b$$
$$M = a$$

2 Weak Bisimulation Examples

2.1 Agency vs Job Shop

In this example, we will look at *agents* and *jobbers*. They can each do one of three types of jobs at a time: easy (E), neutral (N), or hard (H) jobs. The agents are experienced workers and can do one job at a time without assistance, but the jobbers are new to the job–no pun intended–and require the use of tools to accomplish any jobs harder than easy jobs. They require the use of a mallet to do neutral jobs, and a hammer to do hard jobs. Note that they can do neutral jobs with a hammer as well. Once an agent or jobber is done they will use an $\bar{o}$ transition to signal they have output, i.e. they are ready for a new job.

We will first examine an agent, A. Two agents make up an *Agency*.

$$\begin{aligned} A &= \sum_{x \in \{E,N,D\}} i_x \cdot A' \\ &= i_E \cdot A' + i_N \cdot A' + i_D \end{aligned}$$

$$A' = \bar{o} \cdot A$$

$$Agency = A|A$$

Jobbers J and Job Shops are slightly more complicated. They require the mallets and hammers, and the actions gm and gh to get the mallet and hammer–and the corresponding pm and ph to replace the tools.

$$\begin{aligned} M &= gm \cdot M' \\ M' &= pm \cdot M \end{aligned}$$

$$\begin{aligned} H &= gh \cdot H' \\ H' &= ph \cdot H \end{aligned}$$

$$J = \sum_{x \in \{E,N,D\}} i_x \cdot J'$$

$$\begin{aligned} J_E &= \overline{o} \cdot J \\ J_N &= \overline{gm} \cdot \overline{pm} \cdot J_E + \overline{gh} \cdot \overline{ph} \cdot J_E \\ J_D &= \overline{gh} \cdot \overline{ph} \cdot J_D \end{aligned}$$

$$Job\ Shop = \text{new } \vec{t}(H|M|J|J)$$

We'd like to show that $Agency \approx Job\ Shop$. We can do this the same way we showed $E \approx P$ in Section 1.2.

For $x, y \in \{E, N, D\}$,

$$
\begin{aligned}
S = \{&((A|A),\ \text{new}\ \vec{t}\,(J|J|H|M)),\\
&((A|A'),\ \text{new}\ \vec{t}\,(J|J_x|H|M)),\\
&((A|A'),\ \text{new}\ \vec{t}\,(J|\overline{ph}\cdot J_E|H'|M)),\\
&((A|A'),\ \text{new}\ \vec{t}\,(J|\overline{pm}\cdot J_E|H|M'),\\
&((A'|A'),\ \text{new}\ \vec{t}\,(J_y|J_x|H|M)),\\
&((A'|A'),\ \text{new}\ \vec{t}\,(J_y|\overline{ph}\cdot J_E|H'|M)),\\
&((A'|A'),\ \text{new}\ \vec{t}\,(J_y|\overline{pm}\cdot J_E|H|M')),\\
&((A'|A'),\ \text{new}\ \vec{t}\,(\overline{ph}\cdot J_E|\overline{pm}\cdot J_E|H'|M'))\}
\end{aligned}
$$

The system we just described has the Jobbers return the hammer and mallet to the public tool set after they are done using them. We could envision a system where this is NOT the case, i.e. we kept holding the tool until we didn't need it–we got an easy job, or we we need a better tool–we have the mallet but we need the hammer.

If we have Jobbers that don't put down tools unless they have to, it could be that for a Job Shop with two Jobbers, there is a trace where one Jobber is starved for tools. In other words, one Jobber is hogging the hammer or mallet. If we have some notion of job ordering, and we submit three hard jobs in a 123 ordering we cannot have jobs finishing in the following orders: 312, 321, 213.

It's relatively clear why 3 cannot come first, but the 213 ordering is slightly more subtle. In this case, if a Jobber servicing job 2 picks up the hammer, he will keep holding the hammer and service job 3, assuming the other Jobber is servicing job 1. Thus, if all three jobs are hard jobs, the only correct trace is 231.

2.2 Scheduler Example

Now we're going to turn our attention to a token-ring based scheduler. If we look at a single node in the ring, it has 4 ports: $a, b, c, \bar{d}$. c and $\bar{d}$ are for communicating to its neighbors in the ring and a and b are for scheduling.

a is connected to another process which requests a job start for the task scheduled by that node. b is connected to the task scheduled by that node and a communication action occurs when the task finishes.

Thus, the scheduler can be defined as follows. X defines the set of tasks that are running. b_j represents a task in that set finishing, and a_i represents a task at node i starting. We've loosened the syntax a little, so $X - j$ is effectively a removal of j from the set, i.e. the job at node j has finished.

$$
Sched_{i,X} = \begin{cases} \displaystyle\sum_{j\in X} b_j \cdot Sched_{i,X-j} & : i \in X \\ \displaystyle\sum_{j\in X} b_j \cdot Sched_{i,X-j} + a_i \cdot Sched_{i+1,x:=x\cup i} & : i \notin X \end{cases}
$$

The process $Scheduler$ is initialized as such:

$$
Scheduler = Sched_{1,\emptyset}
$$

Another way of defining the scheduler is shown below as S:

$$
\begin{aligned}
A_j &= A < a_i, b_i, c_i, c_{i-1} > \\
A(a,b,c,d) &= a \cdot c \cdot b \cdot \bar{d} \cdot A \\
A &= a \cdot C \\
C &= c \cdot B \\
B &= b \cdot D \\
D &= \bar{d} \cdot A \\
S &= \text{new } \vec{c}\,(A_1|D_1|\ldots|D_n)
\end{aligned}
$$

A sample partial trace of this process as follows:

1. S takes an a transition to:
2. new $\vec{c}\,(C_1|D_2|\ldots|D_n)$ transitions to
3. new $\vec{c}\,(B_1|A_2|D_2|\ldots|D_n)$

We'd like to show that $S \approx Scheduler$, but it does *NOT*. Once you get to the C state in S, your neighbor MUST be ready to act on $\bar{d}$ before you can release on b. In $Scheduler$, you can release your b at any time. To fix this, we can change S to be the following:

$$
\begin{aligned}
A &= a \cdot C \\
C &= c \cdot E \\
E &= b \cdot D + \bar{d}\cdot \\
B &= b \cdot A \\
D &= \bar{d} \cdot A \\
S &= \text{new } \vec{c}\,(A_1|D_1|\ldots|D_n)
\end{aligned}
$$

E alloows us to terminate a process *or* interact with our neighbor. This new process $S \approx Scheduler$.