

Equivalence Semantics of CCS

Moonzoo Kim
CS Dept. KAIST

- Trace Equivalence
- Observational Trace Equivalence
- Bisimulation Equivalence
- Observational Bisimulation Equivalence
- May Preorder and Must Preorder
- Example
- Usage of Concurrent Workbench



Trace Equivalence

- Sys is a **design** for buffer with separated input/output ports

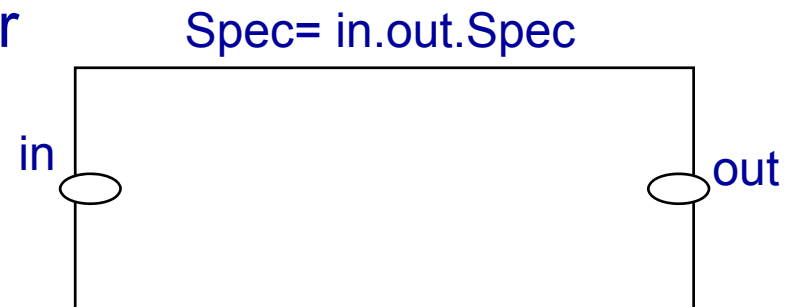
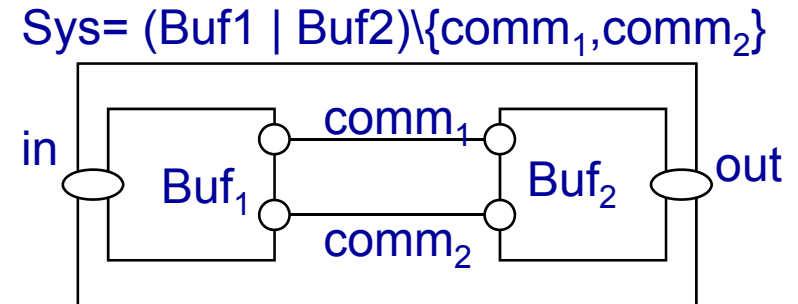
- ✦ $\text{Sys} = (\text{Buf}_1 \mid \text{Buf}_2) \setminus \{\text{comm}_1, \text{comm}_2\}$
 - $\text{Buf}_1 = \text{in}.\text{comm}_1'.\text{Buf}_1'$, $\text{Buf}_1' = \text{comm}_2.\text{Buf}_1$
 - $\text{Buf}_2 = \text{comm}_1.\text{Buf}_2'$, $\text{Buf}_2' = \text{out}.\text{comm}_2'.\text{Buf}_2$

- Spec is a **requirement** for the buffer design

- ✦ $\text{Spec} = \text{in}.\text{Spec}'$, $\text{Spec}' = \text{out}.\text{Spec}$

- Question: $\text{Sys} == \text{Spec}$?

- ✦ Let us consider **trace equivalence** (i.e. language equivalence) $=_T$
 - $T(P) = \{s \in \text{Act}^* \mid s \text{ is an execution trace of } P\}$
 - $P =_T Q$ iff $T(P) = T(Q)$



Observational Trace Equivalence

■ Sys =_T Spec?

✚ No. Sys has τ which Spec does not

- $T(\text{Sys}) = \{\text{in}, \text{in}.\tau, \text{in}.\tau.\text{out}', \text{in}.\tau.\text{out}'.\tau, \dots\}$
- $T(\text{Spec}) = \{\text{in}, \text{in}.\text{out}' \dots\}$

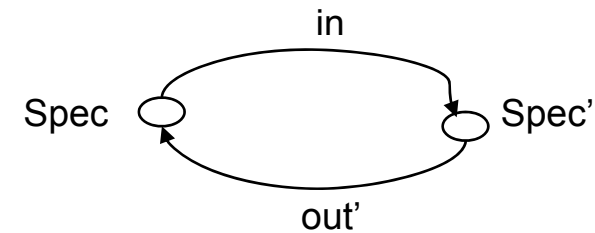
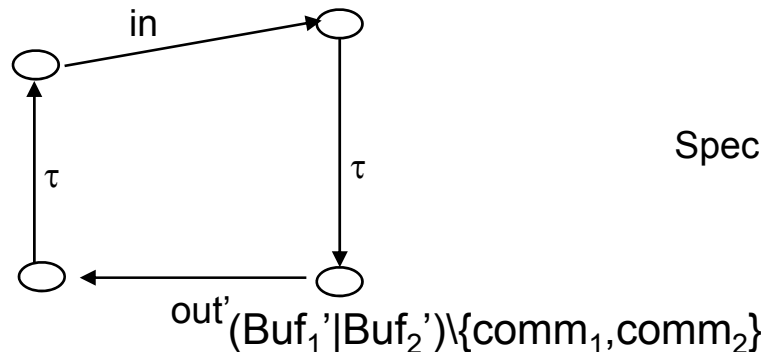
$$\begin{aligned} \text{Sys} &= (\text{Buf}_1 \mid \text{Buf}_2) \setminus \{\text{comm}_1, \text{comm}_2\} \\ \text{Buf}_1 &= \text{in}.\text{comm}_1.\text{Buf}_1', \quad \text{Buf}_1' = \\ &\text{comm}_2.\text{Buf}_1 \\ \text{Buf}_2 &= \\ &\text{comm}_1'.\text{Buf}_2', \text{Buf}_2' = \text{out}.\text{comm}_2'.\text{Buf}_2 \\ \text{Spec} &= \text{in}.\text{out}.\text{Spec} \end{aligned}$$

✚ Yes. τ is an internal hidden action **not visible outside (not observable)**.

Thus, τ should not be included in an execution

- If $s \in \text{Act}^*$, then $\hat{s} \in (\text{Act} - \{\tau\})^*$ is the action sequence obtained by deleting all occurrences of τ from s .
 - Ex> $s = a.\tau.b.\tau.c$, then $\hat{s} = a.b.c$
- A set of **observable** execution traces: $T'(P) = \{\hat{s} \mid s \in T(P)\}$
- $P =_{\text{OT}} Q$ iff $T'(P) = T'(Q)$
- $\text{Sys} =_{\text{OT}} \text{Spec}$ because $T'(\text{Sys}) = \{\text{in}, \text{in}.\text{out}', \dots\}$, $T'(\text{Spec}) = \{\text{in}, \text{in}.\text{out}', \dots\}$

$\text{Sys} = (\text{Buf}_1 \mid \text{Buf}_2) \setminus \{\text{comm}_1, \text{comm}_2\}$



Bisimulation Equivalence

■ $P =_{BS} Q$ iff for all $\alpha \in Act$

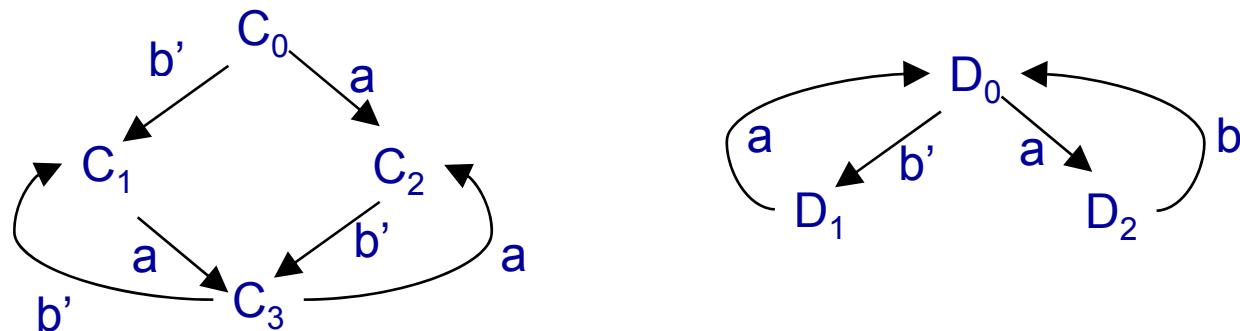
- ✦ Whenever $P \xrightarrow{\alpha} P'$, then for some Q' , $Q \xrightarrow{\alpha} Q'$ and $P' =_{BS} Q'$
- ✦ Whenever $Q \xrightarrow{\alpha} Q'$, then for some P' , $P \xrightarrow{\alpha} P'$ and $P' =_{BS} Q'$

■ Note

- ✦ $=_{BS}$ is an equivalence relation (reflexive, transitive, symmetric)
- ✦ $P =_{BS} Q$ implies $P =_T Q$, but **not vice versa**

■ Example>

- ✦ $C_0 = b'.C_1 + a.C_2$, $C_1 = a.C_3$, $C_2 = b'.C_3$, $C_3 = b'.C_1 + a.C_2$
- ✦ $D_0 = b'.D_1 + a.D_2$, $D_1 = a.D_0$, $D_2 = b'.D_0$
- ✦ A binary relation R proves that $C_0 =_{BS} D_0$
 - $R = \{(C_0, D_0), (C_1, D_1), (C_2, D_2), (C_3, D_0)\}$



Observational Bisimulation Equivalence

- We cannot simply ignore τ for observational bisimulation equivalence.

Thus, we define a new observational transition $=_{\alpha}=>$

- $P =_{OBS} Q$ iff for all $\alpha \in Act$

- ✦ Whenever $P =_{\alpha}=> P'$, then for some Q' , $Q =_{\alpha}=> Q'$ and $P' =_{OBS} Q'$

- ✦ Whenever $Q =_{\alpha}=> Q'$, then for some P' , $P =_{\alpha}=> P'$ and $P' =_{OBS} Q'$

- $P =_{\alpha}=> Q$ iff $P (-\tau->)^* -\alpha-> (-\tau->)^* Q$ where $\alpha \in Act - \{\tau\}$

- ✦ Let $s \in (Act - \{\tau\})^*$. Then $q =_s=> q'$ if there exists s' s.t. $q -s'-> q'$ and $s = \hat{s}'$

- ✦ $P = a.P + b.P$, $Q1 = a.Q1 + \tau.Q2$, $Q2 = b.Q1$

- Suppose that 'a' means pushing button 'a'. Similarly for 'b'

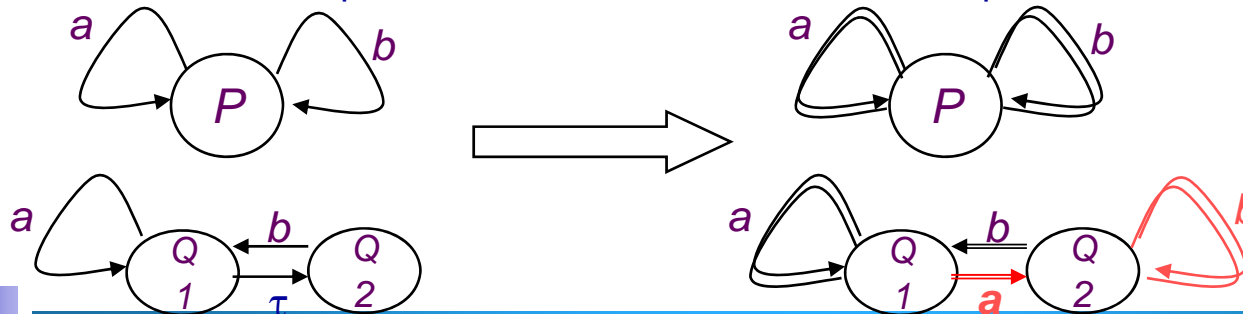
- P always allows a user to push any buttons.

- Q1 allows a user to push button 'a' sometimes, button 'b' sometimes.

- Thus, we need to distinguish P from Q1 (P and Q1 are **not observationally bisimilar**), which can be done using $=_{\alpha}=>$ instead of $-\alpha->$

- $Q1 -a-> Q1$ implies $Q1 =_a=> Q1$. Similarly $Q2 -b-> Q1$ implies $Q2 =_b=> Q1$

- $Q1 -a-> Q1 -\tau-> Q2$ implies $Q1 =_a=> Q2$. $Q2 -b-> Q1 -\tau-> Q2$ implies $Q2 =_b=> Q2$



Observational Bisimulation Equivalence (cont)

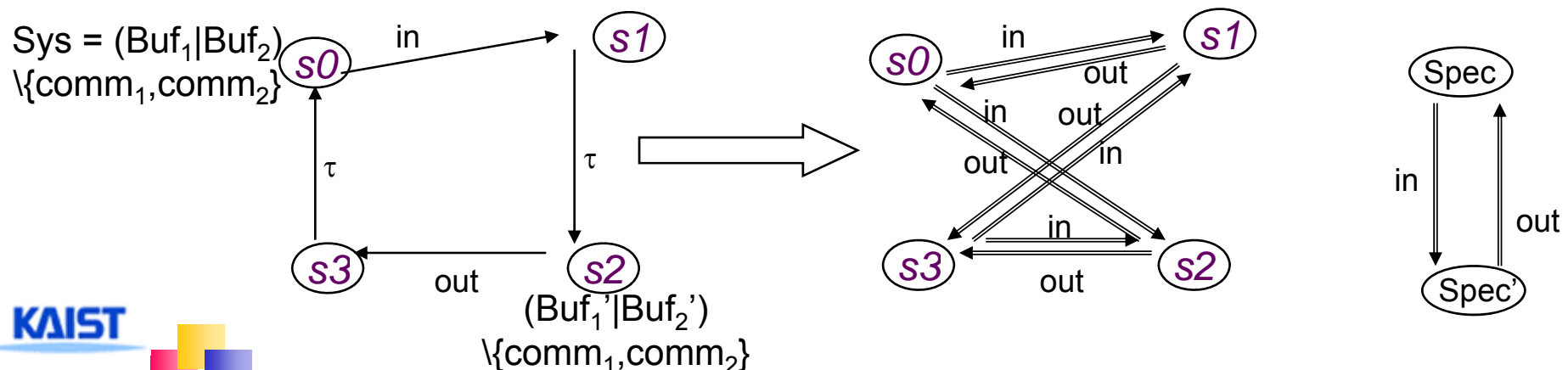
■ $\text{Sys} =_{\text{BS}} \text{Spec?}$ (see slide 3)

- ✚ No. Sys has τ which Spec does not (i.e. not strongly bisimilar)

■ $\text{Sys} =_{\text{OBS}} \text{Spec?}$

- ✚ Yes. Sys is **observationally bismilar** to Spec

- Proof: $R = \{ (s_0, \text{Spec}), (s_1, \text{Spec}'), (s_3, \text{Spec}), (s_2, \text{Spec}') \}$
 - $s_0 \text{-in-} \rightarrow s_1$ implies $s_0 = \text{in} \Rightarrow s_1$. Similarly, $s_2 \text{-out-} \rightarrow s_3$ implies $s_2 = \text{out} \Rightarrow s_3$
 - $s_0 \text{-in-} \rightarrow s_1 \text{-}\tau \rightarrow s_2$ implies $s_0 = \text{in} \Rightarrow s_2$.
 - $s_2 \text{-out-} \rightarrow s_3 \text{-}\tau \rightarrow s_0$ implies $s_2 = \text{out} \Rightarrow s_0$



CWB-NC Commands

- load <ccs filename>
- help <command>
- ls
- cat <process>
- compile <process>
- es <script file> <output file>
- eq -S <trace|bisim|obseq> <proc1> <proc2>
- le -S may <proc1> <proc2> /* Trace subset relation */
- sim <process>
 - + semantics <bisim|obseq>
 - + random <n>
 - + back <n>
 - + break <act list>
 - + history
 - + quit
- quit



Example: Faulty Mutual Exclusion Protocol

```
byte cnt, byte x,y,z;
active[2] proctype user()
{
    byte me = _pid +1; /* me is 1 or 2*/
again:
    x = me;
    if
    :: (y ==0 || y== me) -> skip
    :: else -> goto again
    fi;

    z =me;
    if
    :: (x == me) -> skip
    :: else -> goto again
    fi;

    y=me;
    if
    :: (z==me) -> skip
    :: else -> goto again
    fi;

    /* enter critical section */
    cnt++
    assert( cnt ==1);
    cnt --;
    goto again
}
```

```
proc Sys = (P1|P2|X0|Y0|Z0|CNT0){x_[0-2],y_[0-2],z_[0-2],
test_x_[0-2],test_y_[0-2],test_z_[0-2], inc_cnt,dec_cnt}
```

```
proc P1 = x_1.(test_y_0.P1' + test_y_1.P1' + test_y_2.P1)
proc P1' = z_1.(test_x_0.P1 + test_x_1.P1" + test_x_2.P1)
proc P1" = y_1.(test_z_0.P1 + test_z_1.P1"' + test_z_2.P1)
proc P1"' = inc_cnt.dec_cnt.P1
```

```
proc P2 = x_2.(test_y_0.P2' + test_y_1.P2 + test_y_2.P2')
proc P2' = z_2.(test_x_0.P2 + test_x_1.P2 + test_x_2.P2'')
proc P2'' = y_2.(test_z_0.P2 + test_z_1.P2 + test_z_2.P2''')
proc P2''' = inc_cnt.dec_cnt.P2
```

* Variable x, y,z, and cnt

```
proc UpdateX = 'x_0.X0 + 'x_1.X1 + 'x_2.X2
```

```
proc X0 = 'test_x_0.X0 + UpdateX
```

```
proc X1 = 'test_x_1.X1 + UpdateX
```

```
proc X2 = 'test_x_2.X2 + UpdateX
```

```
proc UpdateY = 'y_0.Y0 + 'y_1.Y1 + 'y_2.Y2
```

```
proc Y0 = 'test_y_0.Y0 + UpdateY
```

```
proc Y1 = 'test_y_1.Y1 + UpdateY
```

```
proc Y2 = 'test_y_2.Y2 + UpdateY
```

```
proc UpdateZ = 'z_0.Z0 + 'z_1.Z1 + 'z_2.Z2
```

```
proc Z0 = 'test_z_0.Z0 + UpdateZ
```

```
proc Z1 = 'test_z_1.Z1 + UpdateZ
```

```
proc Z2 = 'test_z_2.Z2 + UpdateZ
```

```
proc CNT0 = 'inc_cnt.cnt_1.CNT1
```

```
proc CNT1 = 'inc_cnt.cnt_2.CNT2 + 'dec_cnt.cnt_0.CNT0
```

```
proc CNT2 = 'dec_cnt.cnt_1.CNT1
```

```
proc Spec = cnt_1.cnt_0.Spec
```

