Equivalence Semantics of CCS

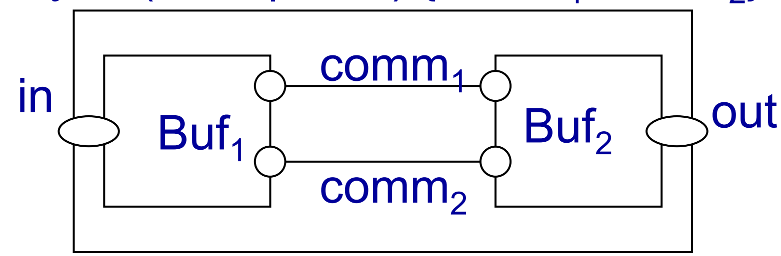
Moonzoo Kim
School of Computing
KAIST

Oultline

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

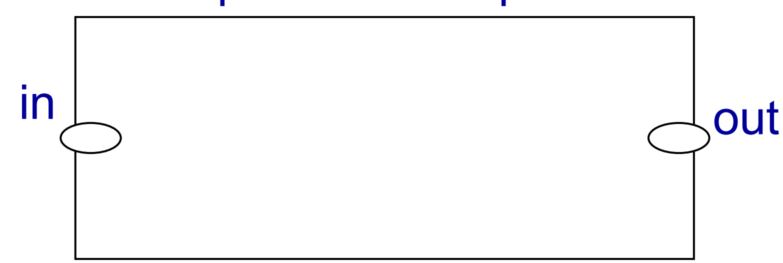
- Sys is a design for buffer with separated input/output ports
  - Sys= $(Buf_1 \mid Buf_2)\backslash\{comm_1,comm_2\}$
    - Buf_1 = in.$comm_1$'.Buf_1', Buf_1' = $comm_2$.Buf_1
    - Buf_2 = $comm_1$.Buf_2',Buf_2'= out'.$comm_2$'.Buf_2

Sys= $(Buf1 \mid Buf2)\backslash\{comm_1,comm_2\}$



- Spec is a requirement for the buffer design
  - Spec = in.Spec', Spec'=out'.Spec

Spec= in.out.Spec



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

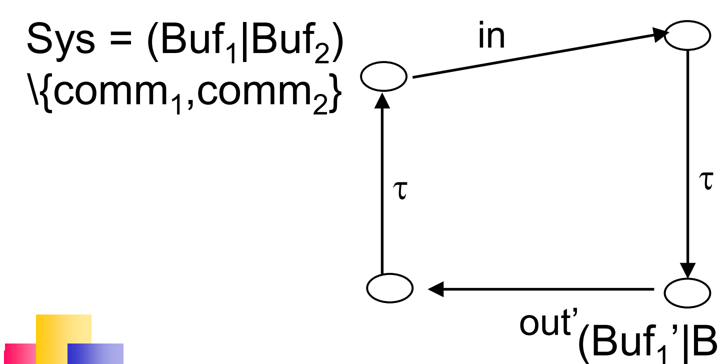
KAIST

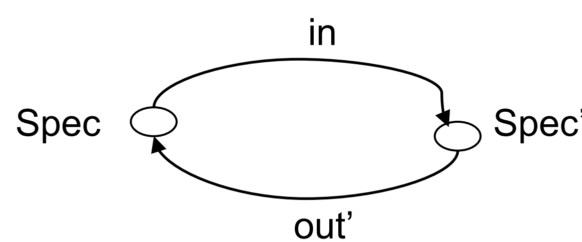
Observational Trace Equivalence

- Sys $=_T$ Spec?
  - No. Sys has τ which Spec does not
    - T(Sys) = {in, in.τ, in.τ.out' , in.τ.out'.τ,…}
    - T(Spec) = {in , in.out' …}

Sys= (Buf1 | Buf2)\{comm1,comm2}
Buf1 = in.comm1.Buf1', Buf1' = comm2.Buf1
Buf2 = comm1'.Buf2',Buf2'=out.comm2'.Buf2
Spec = in.out.Spec

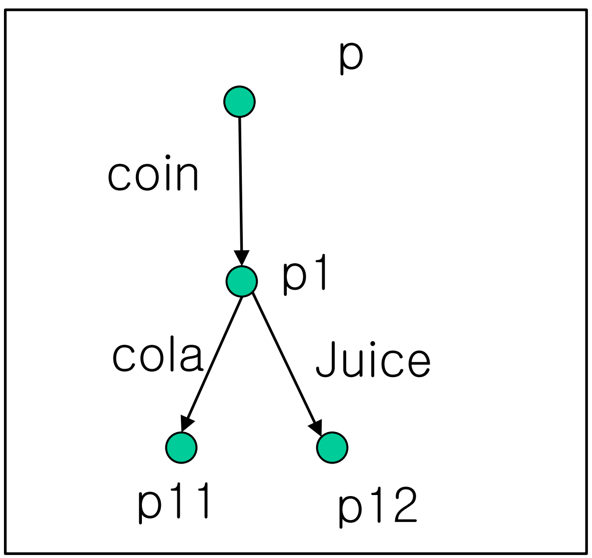
  - Yes. τ is an internal hidden action not visible outside (not observable). Thus, τ should not be included in an execution
    - If s$\in$Act*, then ŝ $\in$(Act –{τ})* is the action sequence obtained by deleting all occurrences of τ from s.
      - Ex> s = a.τ.b.τ.c, then ŝ = a.b.c
    - A set of observable execution traces: T'(P) = {ŝ | s $\in$ T(P)}
    - P $=_{OT}$ Q iff T'(P) = T'(Q)
    - Sys $=_{OT}$ Spec because T'(Sys) = {in, in.out',…}, T'(Spec) = {in, in.out', …}

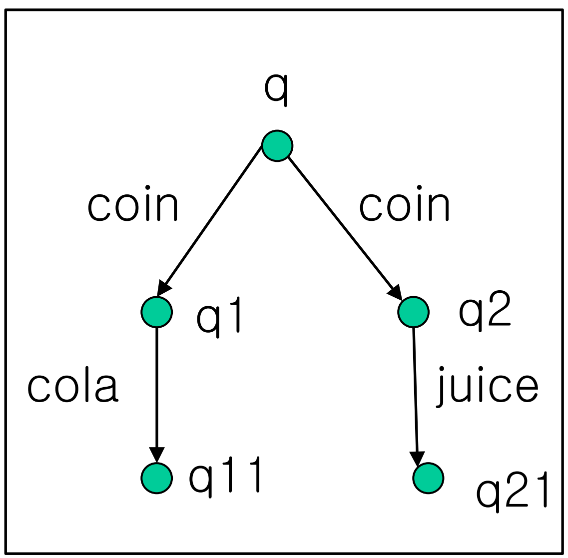


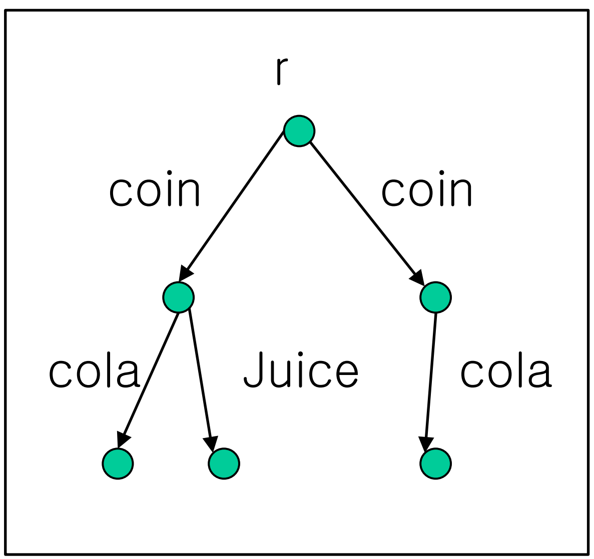


Importance of Branching Behavior

Which vending machine do you prefer?
p? q? r?

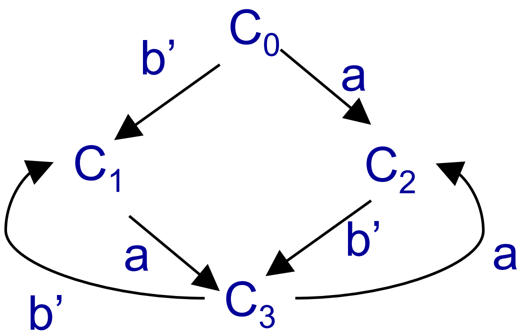


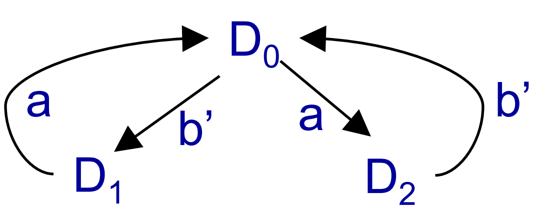




Bisimulation Equivalence

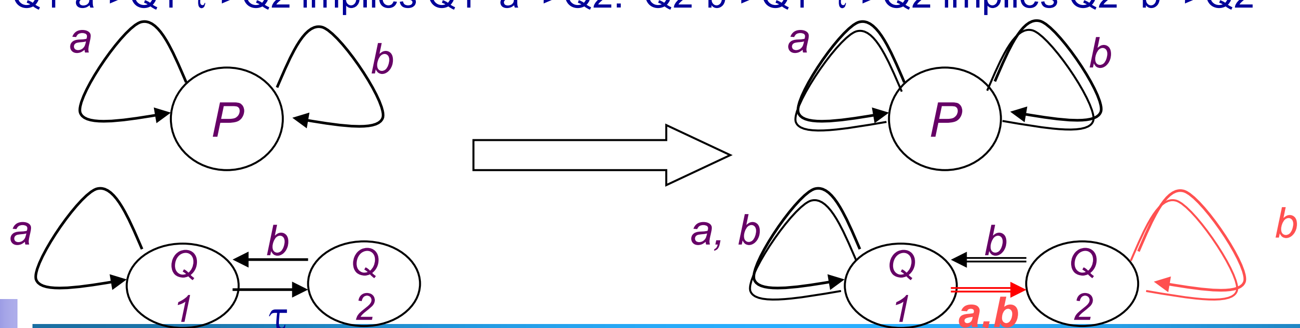
- P $=_{BS}$ Q iff for all $\alpha \in$ Act
  - Whenever P -α-> P', then for some Q', Q -α-> Q' and P' $=_{BS}$ Q'
  - Whenever Q -α-> Q', then for some P', P -α-> P' and P' $=_{BS}$ Q'
- Note
  - $=_{BS}$ is an equivalence relation (reflexive, transitive, symmetric)
  - P $=_{BS}$ Q implies P $=_T$ 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)}





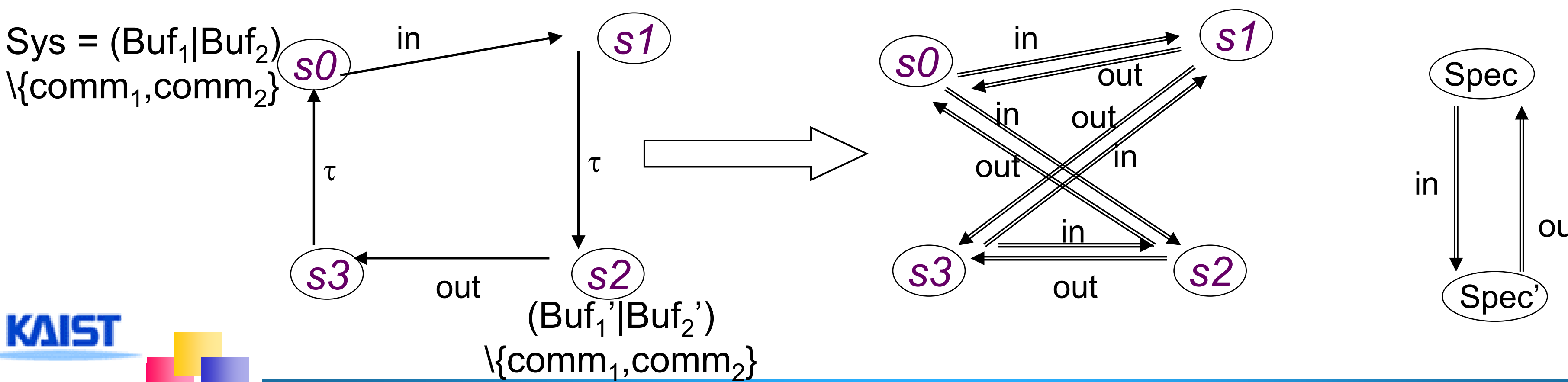
Observational Bisimulation Equivalence

- 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=ŝ'
  - P = a.P + b.P, Q1=a.Q1 + τ.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. Similary Q2-b->Q1 implies Q2=b=>Q1
      - Q1-a->Q1-τ->Q2 implies Q1=a=>Q2. Q2-b->Q1- τ->Q2 implies Q2=b=>Q2



Observational Bisimulation Equivalence (cont)

- Sys $=_{BS}$ Spec? (see slide 3)
  - No. Sys has τ which Spec does not (i.e. not strongly bisimilar)
- Sys $=_{OBS}$ Spec?
  - Yes. Sys is observationally bismilar to Spec
    - Proof: R = { (s0,Spec), (s1,Spec'),(s3,Spec),(s2,Spec')}
      - s0 –in->s1 implies s0=in=> s1. Similarly, s2-out->s3 implies s2=out=>s3
      - s0 -in->s1 -τ->s2 implies s0=in=>s2.
      - s2-out->s3-τ-> s0 implies s2=out=>s0



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

KAIST