Equivalence Hierarchy

Moonzoo Kim

CS Dept. KAIST

Outline

- Equivalence semantics and SW design
- Preliminary
- Hierarchy Diagram
- Trace-based Semantics
  - Trace EQ
  - Complete Trace EQ
  - Failure EQ
- Branching-based Semantics
  - Simulation EQ
  - Bisimulation EQ

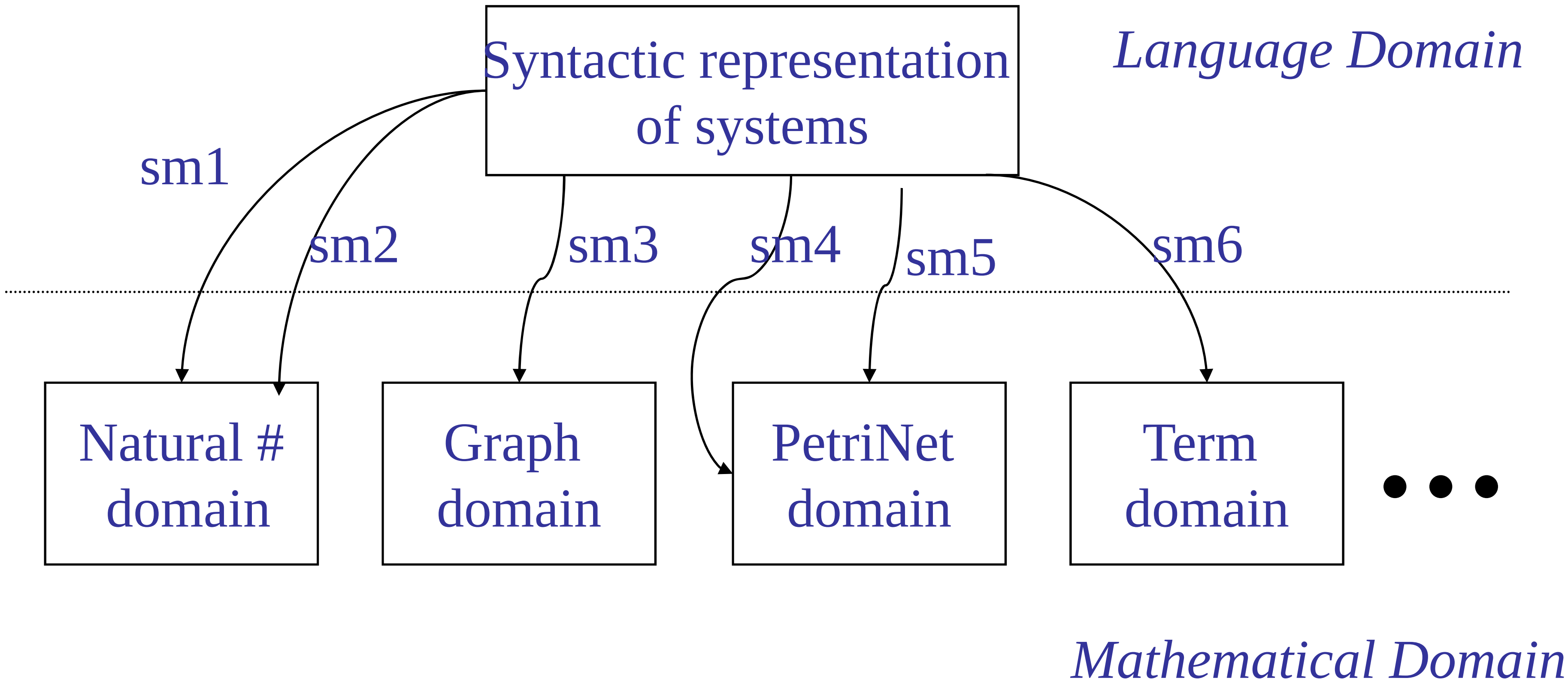
Equivalence Preserving Refinement and SW Design

- Design can start with a very abstract specification, representing the requirements
- Then, using equivalence-preserving transformations, this specification can be gradually refined into an implementation-oriented specification.
- Maintenance may require to replace some components with others, while maintaining the same system behavior (congruence property)

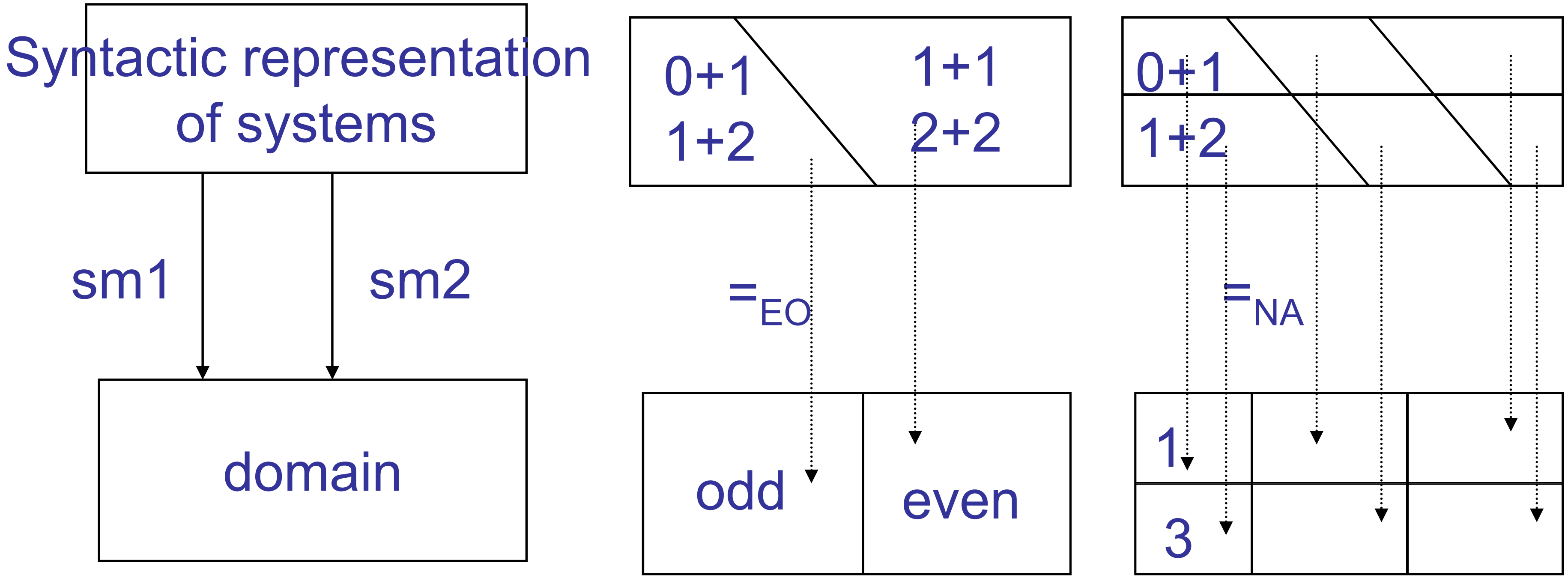
Semantic Mapping

- An example of small language
  - Syntax
    - F := 0 | 1 | F + 1 | 1 + F
    - Ex. 0, 0+1+1, 1+0+1, but not 0+0
  - Possible semantics
    - 1 + 1 == 1 + 1 + 0 ?
      - Yes (interpreting formula as a natural #),
        - $[1 + 1]_{N1} = 2$, $[1 + 1 + 0]_{N1} = 2 \rightarrow 1 + 1 =_{N1} 1 + 1 + 0$
      - No (interpreting formula as string),
        - $[1+1]_S$="1+1",$[1+1+0]_S$="1+1+0"$\rightarrow$ $1+1 \;!\!=_S 1+1+0$
      - No (interpreting formula as a natural # of string length)
        - $[1 + 1]_{N2} = 3$, $[1 + 1 + 0]_{N2} = 5 \rightarrow 1 + 1 \;!\!=_{N2} 1 + 1 + 0$

Semantic Mapping (cont.)



Relation between (Equivalence) Semantics

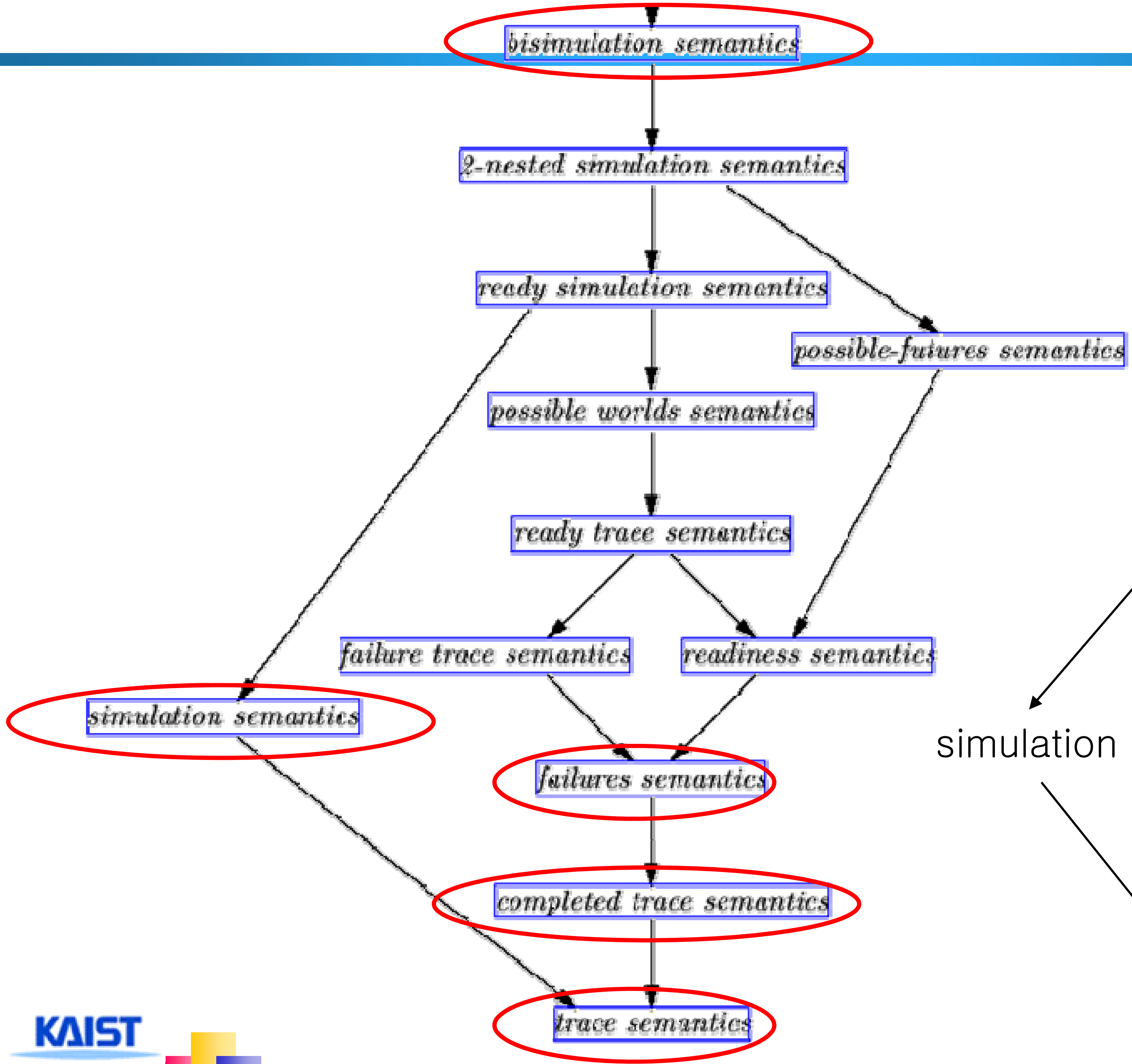


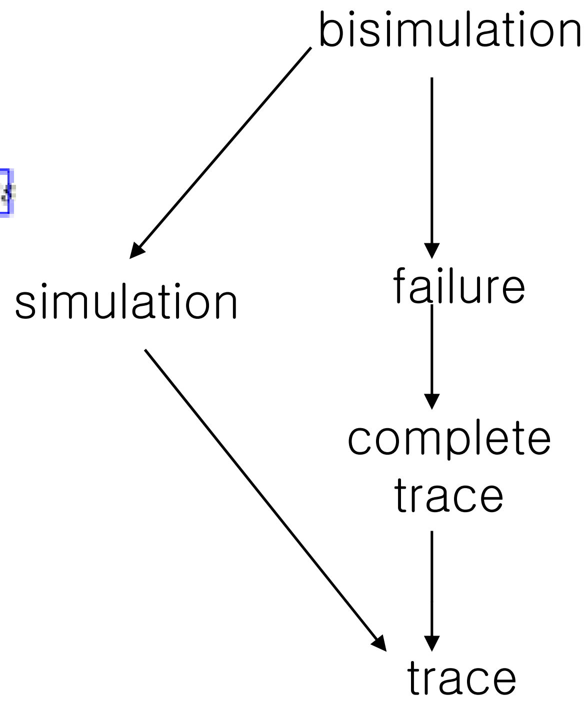
$0+1 =_{EO} 1+2$
$1+1 =_{EO} 2+2$

$0+1 \,!=_{NA} 1+2$

$P =_{NA} Q \rightarrow P =_{EO} Q$ but not vice versa

Therefore, $=_{EO} < =_{NA}$


bisimulation semantics
2-nested simulation semantics
ready simulation semantics
possible-futures semantics
possible worlds semantics
ready trace semantics
failure trace semantics
readiness semantics
simulation semantics
failures semantics
completed trace semantics
trace semantics


bisimulation
simulation
failure
complete
trace
trace

Labeled Transition System

- Process Theory
  - A *process* represents behavior of a system
  - Two main activities of process theory are *modeling* and *verification*
    - The semantics of equalities is required to verify system
    - Determine which semantics is suitable for which applications

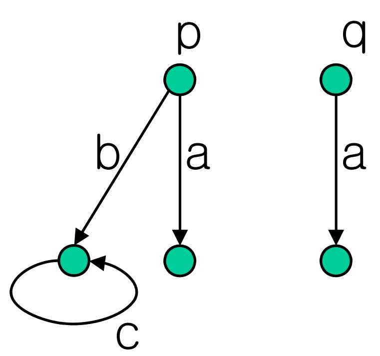
- Labeled Transition System (LTS)
  - *Act*: a set of actions which process performs
  - LTS: $(P, \rightarrow)$
    - Where P is a set of processes and $\rightarrow \subseteq P \times Act \times P$
  - In this presentation, we deal with only *finitely branching, concrete, sequential processes*

- Useful notations
  - Equivalence notation for each semantics
    - $=_T, =_{CT}, =_F, =_R, =_{FT}, =_{RT}, =_S, =_{RS}, =_B$
    - $I(p)$ is $\{a \in Act \mid \exists q.\ p \xrightarrow{a} q\}$

Trace v.s. Complete Trace

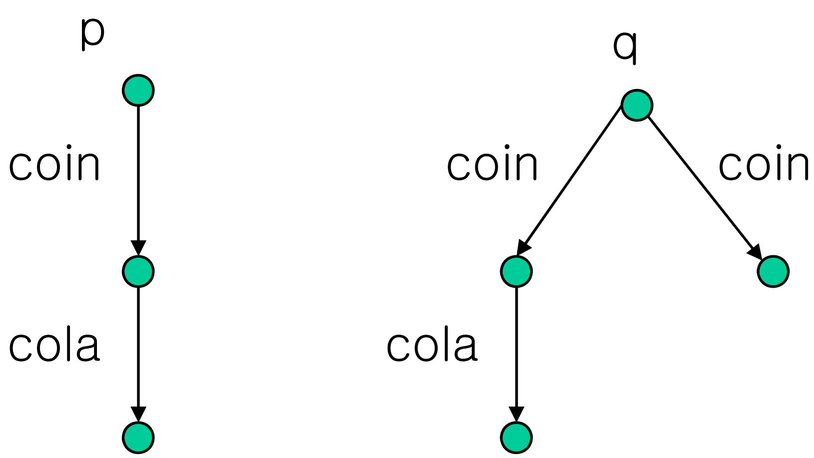
- Trace semantics (T)
  - $\sigma \in Act^*$ is a *trace* of a process p if there is a process q s.t. p -σ-> p
  - T(p) is a set of traces of a process p
  - $p =_T q$ iff T(p) = T(q)

- Complete trace semantics (CT)
  - $\sigma \in Act^*$ is a *complete trace* of a process p if there is a process q s.t. p -σ-> q and $I(q) = \emptyset$
  - CT(p) is a set of complete traces of a process p
  - $p =_{CT} q$ iff T(p) = T(q) and CT(p) = CT(q)
  - Note that CT(p) = CT(q) does not imply T(p) = T(q)



- $=_T < =_{CT}$
  - $p =_{CT} q$ implies $p =_T q$, but not vice versa

Counter Example 1

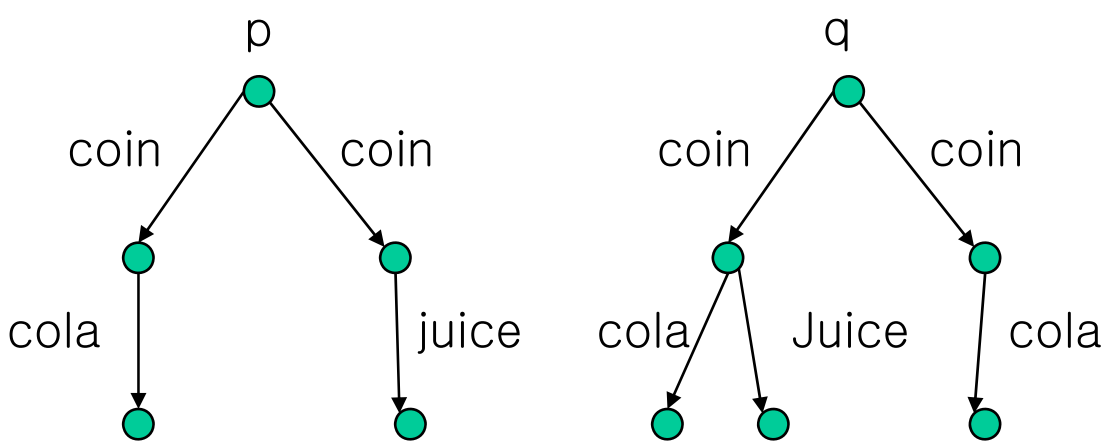


- $p =_T q$
  - T(p) = {coin.cola, coin}
  - T(q) = {coin.cola, coin}
- $p \neq_{CT} q$
  - CT(p) = {coin.cola}
  - CT(q) = {coin.cola, coin}

Failure Semantics

- Failure Semantics (F)
  - $<\sigma, X> \in Act^* \times \Pi(Act)$ is a failure pair of p if $\exists$ q s.t. p $-\sigma\text{->}$ q and $I(q) \cap X = \varnothing$
  - $F(p)$ is a set of failure pairs of p
  - $p =_F q$ iff $F(p) = F(q)$
- $=_{CT} < =_F$
  - $p =_F q$ implies $p =_{CT} q$
    - $\sigma \in CT(p)$ iff $<\sigma, Act> \in F(p)$
    - $\sigma \in T(p)$ iff $<\sigma, X> \in F(p)$ for some X s.t. $X \cap I(q) = \varnothing$ Where $p -\sigma\text{->} q$
  - not vice versa

Counter Example 2

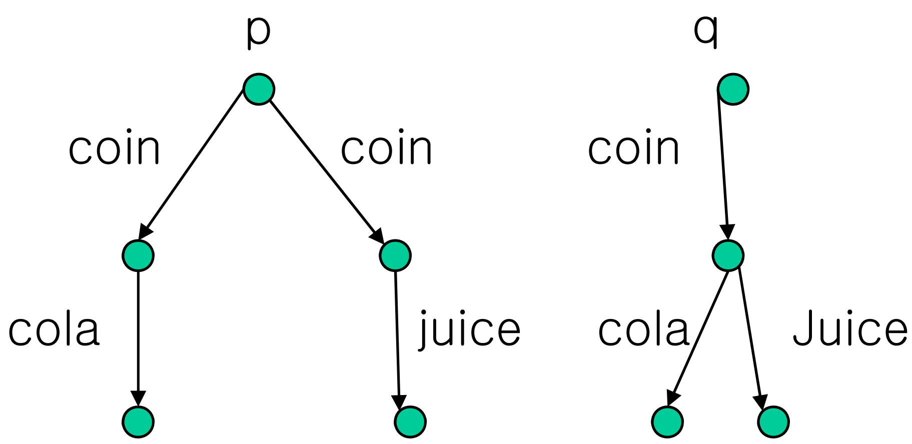


- $p =_{CT} q$
  - CT(p)={coin.cola, coin.juice}
  - CT(q)={coin.cola, coin.juice}
- $p \neq_F q$
  - {<coin,{coin,cola}>} $\in$ F(p)
  - {<coin,{coin,cola}>} $\notin$ F(q)

Simulation Semantics

- The set F_s of simulation formulas over Act is defined inductively by
  - *True* $\in F_s$
  - If $\Phi,\Psi \in F_s$ then $\Phi \wedge \Psi \in F_s$
  - If $\Phi \in F_s$ and $a \in Act$, then $a.\Phi \in F_s$
- The satisfaction relation $\models \subseteq P \times F_s$ is defined inductively by
  - $p \models True$ for all $p \in P$
  - $p \models \Phi \wedge \Psi$ if $p \models \Phi$ and $p \models \Psi$
  - $p \models a.\Phi$ if for some $q \in P$: $p \xrightarrow{a} q$ and $q \models \Phi$
- $p =_S q$ iff $S(p) = S(q)$ where $S(p)=\{\Phi \in F_s | p \models \Phi\}$

- $=_T < =_S$
  - $p =_S q$ implies $p =_T q$
    - $=_T < =_S$ by $\sigma \in T(p)$ iff $\sigma.True \in S(p)$
  - not vice versa



- $p \neq_S q$
  - S(*p*)= {*True*, coin.*True*, coin.cola.*True*, coin.juice.*True*, …,
    coin.cola.*True* $\wedge$ coin.juice.*True*}
  - S(q) = {*True*, coin.*True*, coin.cola.*True*, coin.cola.*True*, …,
    coin.cola.*True* $\wedge$ coin.juice.*True*,
    coin.(cola.*True* $\wedge$ juice.*True*) }

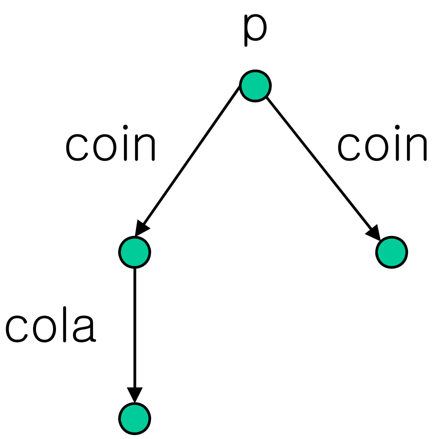
Simulation v.s. Bisimulation

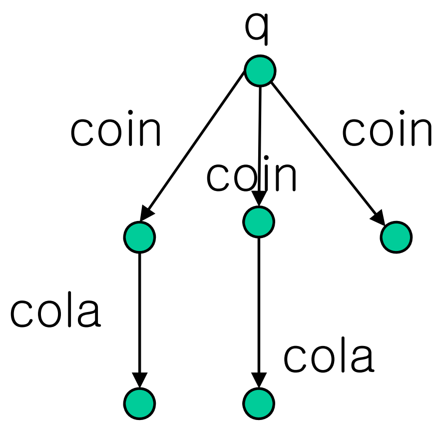
- A simulation is a binary relation R on processes satisfying for a $\in$ *Act*
  - If pRq and p-a->p' , then $\exists$ q':q-a->q' and $p'Rq'$
- $p =_S q$ iff there exist simulation relations R_1 and R_2 such that pR_1q and qR_2p
- A bisimulation is a binary relation R on processes satisfying for a $\in$ *Act*
  - If pRq and p-a->p' , then $\exists$ q':q-a->q' and $p'Rq'$
  - If pRq and q-a->q' , then $\exists$ p':p-a->p' and $p'Rq'$
- $P =_B q$ if there exists a bisimulation R with pRq

Counter Example 3

$p =_B q$

$p =_s q$





$p \neq_B q$

$p =_s q$