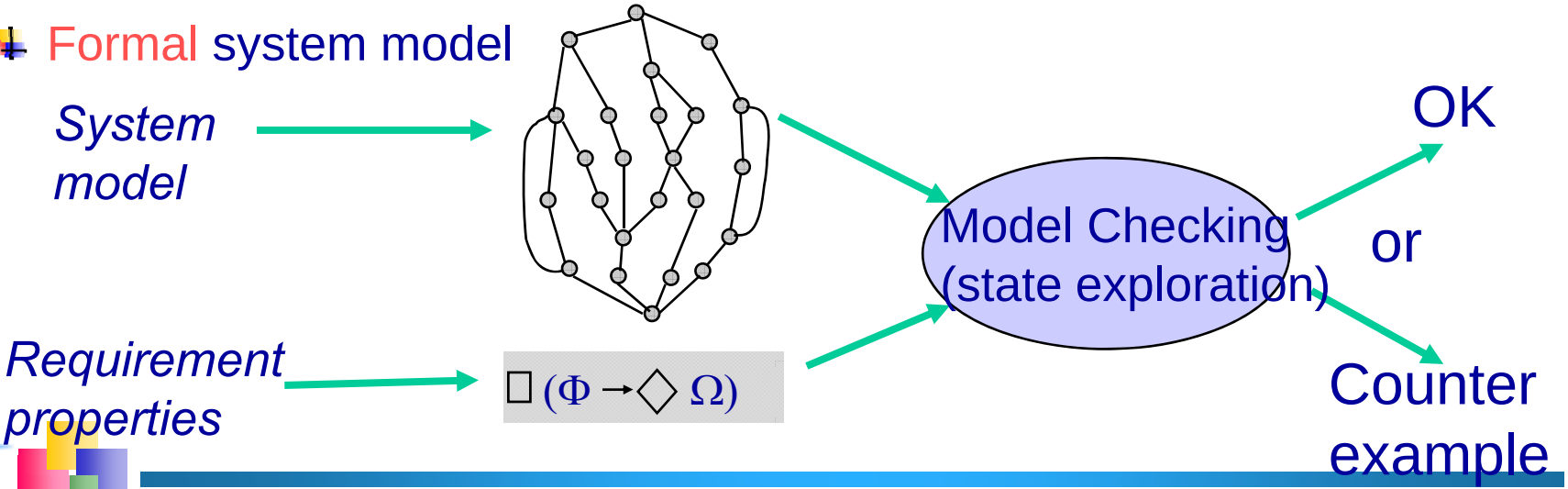
Introduction to Process Algebra

Weakness on Traditional Validation & Verification (V&V)

- We have seen tragic accidents due to software and specification bugs
- These bugs are hard to find because those bugs occurs only in “exceptional” cases
- Informal system specification and requirement specification makes automatic analysis infeasible, which results in incomplete coverage
- To provide better coverage, we need
  - Formal requirement specification
  - Formal system model



Outline

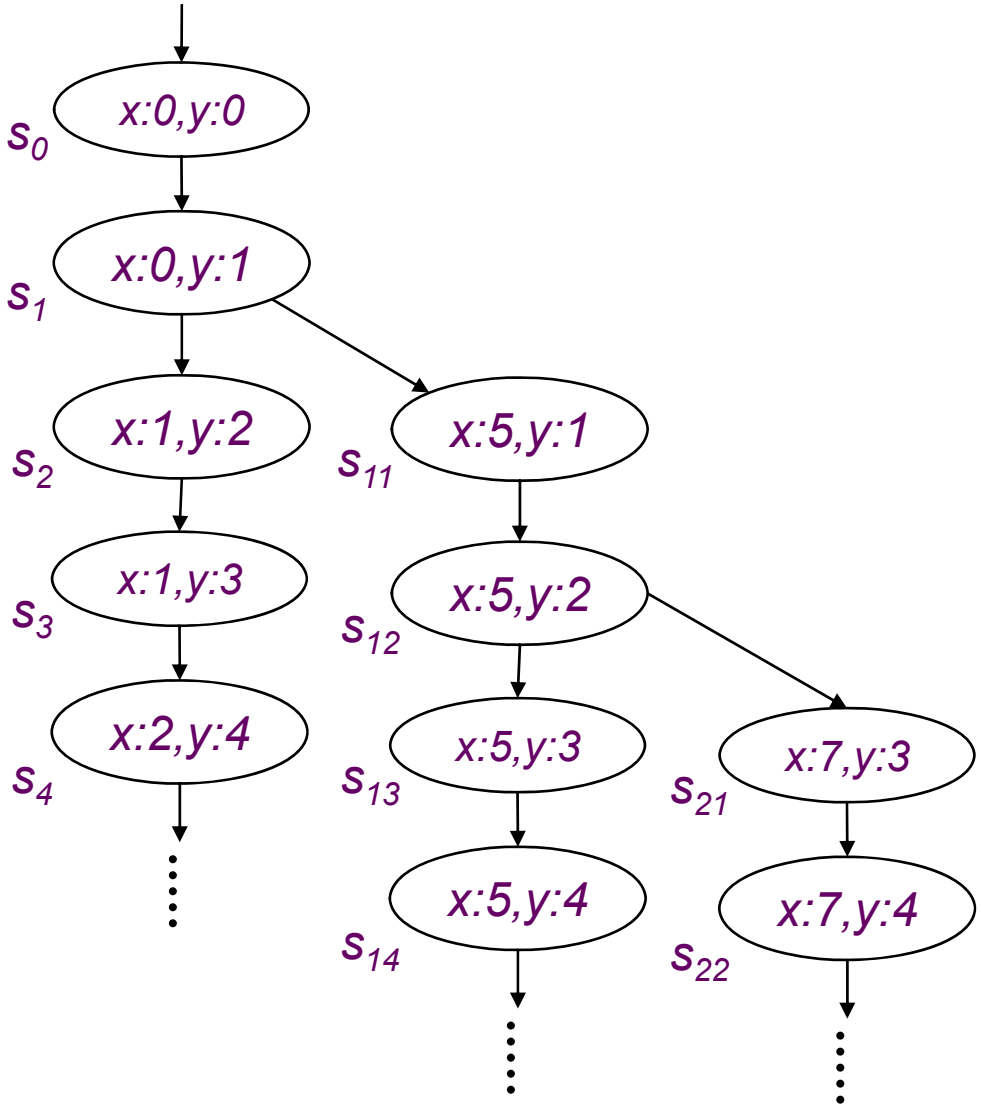
- Requirement specification problems
- Viewpoint on “meaning”(semantics) of system
- Complexity of a system
- Formal modeling v.s. programming
- Introduction to process algebra

Requirement Specification Problems

- Ambiguity
  - Expression does not have unique meaning, but can be interpreted as several different meaning.
    - Ex. `long` type in C programming language
- Incompleteness
  - Relevant issues are not addressed , e.g. what to do when user errors occur or software faults show.
    - Ex. Retail chain management software
- Inconsistency
  - Contradictory requirements in different parts of the specification.

Viewpoint on Semantics of a System

- A system execution σ is a sequence of states $s_0 s_1 \ldots$
  - A state has an environment ρ_s: *Var-> Val*
- A system has its semantics as a set of system executions



Complexity of Systems

- The complexity of a system is sometimes more accurately expressed using semantic viewpoint (# of reachable states) rather than syntactic viewpoint (line # of source code)
  - the number of different *states* a system can reach
    - Ex> An integer has 2^{32} (~4000000000) possible values

Formal Modeling V.S. Programming

		Formal Modeling	Programming
Static Aspects	Abstraction Level	High	Low
	Development Time	Short	Long
Dynamic Aspects	Executable	Yes (model checking) No (theorem proving)	Always
	System Semantics	Mathematically defined	Usually given by examples
	Environment Semantics (i.e. testbeds)	Mathematically defined	Usually given by examples
	Program State Space	Manageable (i.e. tractable state space)	Unmanageable (i.e. beyond computing power)
	Validation	By exhaustive exploration or deductive proof	By testing (incomplete coverage)

Complex System Attributes

- You may not need to model a simple system such as +,*, or HelloWorld.
- However, you must have a scientific way of abstracting/modeling a system with complex structure, e.g.,
  - Hierarchy
  - Concurrency
  - Communication
- Also, you need to have a systematic way to analyze the correctness of your design

Process Algebra

- A process algebra consists of
  - a set of operators and syntactic rules for constructing processes
  - a semantic mapping which assigns meaning or interpretation to every process
  - a notion of equivalence or partial order between processes
- Advantages: A large system can be broken into simpler subsystems and then proved correct in a modular fashion.
  - A hiding or restriction operator allows one to abstract away unnecessary details.
  - Equality for the process algebra is also a congruence relation; and thus, allows the substitution of one component with another equal component in large systems.
- Note that the model is constructed in a component-based way, but the analysis is not.

Calculus of Communicating Systems (CCS)

- Developed by R.Milner (Univ. of Cambridge)
  - ACM Turing Award 1991
- Provides many interesting paradigms
  - Emphasis on communication and concurrency
    - Provides compact representation on both communication and concurrency
      - Ex> a (receive) and a' (send)
      - Ex> | (parallel operator)
  - Provides observation based abstraction
    - Hiding internal behaviors using \ (restriction) operator, i.e., considering all internal behaviors as an invisible special action τ
  - Provides correctness claim based on equivalence
    - Branching time based equivalence
      - Strong equivalence v.s. weak equivalence

Overview on CCS Syntax and Semantics

- CCS describes a system as a set of communicating Processes
- Behavior of a process is expressed using actions
  - Act =input_actions U output_actions U {τ}
- Each process is built based on the following 7 operators
  - Nil (null-ary opeartor): 0
  - Prefix: a.P
  - Definition: P = a.b.Q
  - Choice: a.P + b.P
  - Parallel: P | Q
  - Restriction: P \ {a,b}
  - Relabelling: P[a/b]
- Each operator has a clear formal semantics via inference rules (premises-conclusion rules)
  - Based on these inference rules, a meaning/semantincs of a process is given as a labelled transition system

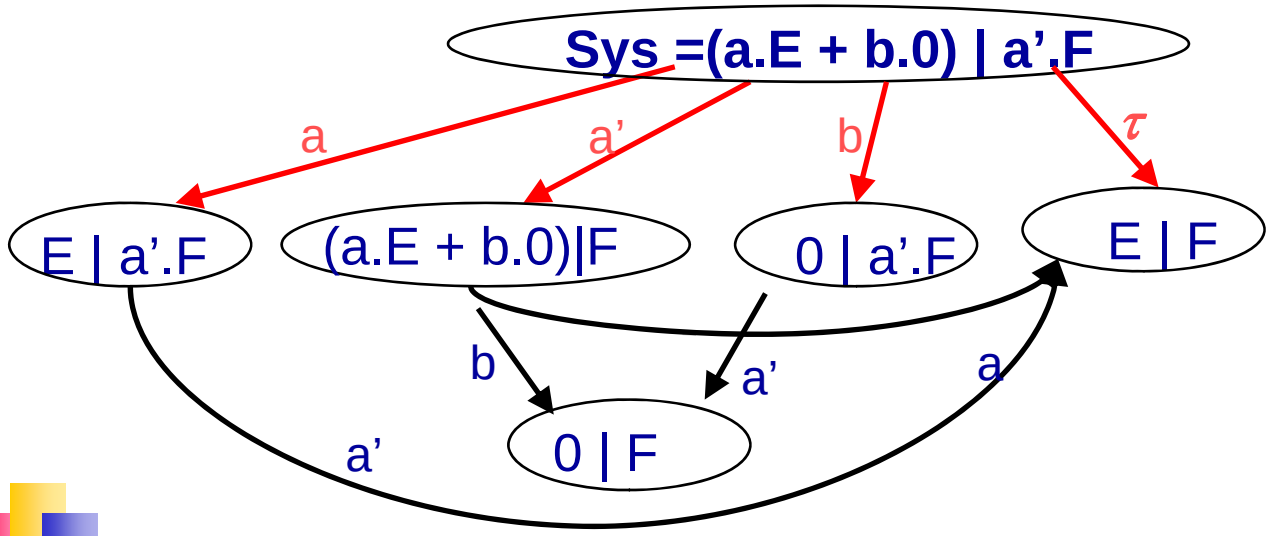
Example of a CCS System

- A set of actions Act = {a,a',b,τ}
- We define a CCS system Sys as
  - Sys = (a.E + b.0) | a'.F
- Sys can executes one of the following 4 actions
  - Sys –a-> E | a'F
  - Sys –a'-> (a.E + b.0)|F
  - Sys –b-> 0 | a'.F
  - Sys - τ-> E|F

$$\text{Prefix}\ \frac{}{a.E\ \text{–a->}\ E}$$

$$\text{Choice}_L\ \frac{}{(a.E + b.0))\ \text{–a->}\ E}$$

$$\text{Par}_L\ \frac{}{\mathbf{(a.E + b.0)) \mid a'.F\ \text{–a->}\ E \mid a'.F}}$$



Usage of Process Algebra

- Sequential system v.s. Reactive system
  - Ex1. Mathematical functions with given inputs generate outputs
    - Usually no environment consideration and timing consideration.
  - Ex2. Ad-hoc On-Demand Vector routing protocol
    - Should model multiple concurrent nodes (environment)
    - Should model communication among the nodes
    - Should model timely behavior (e.g. time-out, etc)
- Modeling of a complex system
  - Concurrency => interleaving semantics
  - Communication => synchronization
  - Hierarchy => refinement

Notations (1/2)

- A system is described as a set of communicating processes
  - Each process executes a sequence of actions
  - Actions represents either inputs/outputs or internal computation steps

- A set of actions/events $Act = L \cup L' \cup \{\tau\}$
  - L ={a,b,…} is a set of *names* and L' ={a',b',…} is a set of *co-names*
    - a$\in$ L can be considered as the act of receiving a signal
    - a'$\in$ L' can be considered as the act of emitting a signal
    - τ is a special action to represent internal hidden action
  - $Act - \{\tau\}$ represents the set of externally visible actions:

KAIST

Notations (2/2)

- Operational (transitional) semantics of CCS process
  - Define the “execution steps” that processes may engaged in
  - P –a-> P’ holds if a process P is capable of engaging in action a and then behaving like P’
  - Define –a-> inductively using inference rules for operators
    - premises
      ---------- *(side condition)*
      conclusion

Example 1:

$$\text{Choice}_R \quad \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$$

Example 2:

$$\text{Prefix} \quad \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

Operators for Sequential Process

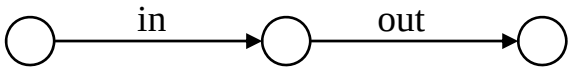
The idea: 7 elementary ways of producing or putting together labelled transition systems

1.Nil 0 — No transitions (deadlock)

2.Prefix $\alpha.P$ ($\alpha \in Act$) — in.out.0 –*in*-> out.0 –*out*-> 0

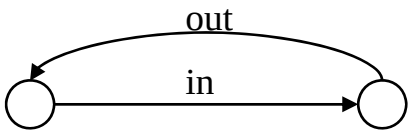
Prefix

$$\text{Prefix}\ \frac{\text{(empty)}}{\alpha.P \xrightarrow{\alpha} P}$$



3.Defn $A = P$ — Buffer = in.out.Buffer

Buffer-*in*->out.Buffer-*out*->Buffer



Operators for Sequential Process (cont.)

4.Choice *P* + *Q*

$$\text{Choice}_L \quad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$$

$$\text{Choice}_R \quad \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$$

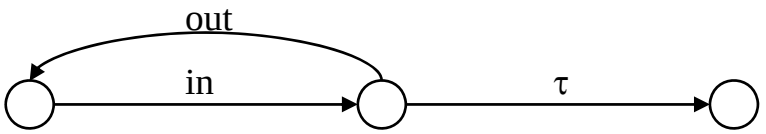
BadBuf = in.(τ.0 + out.BadBuf)

Prefix

BadBuf –*in*-> τ.0 + out.BadBuf

Choice_L Choice_R

-*τ*-> 0 **or** –*out*-> BadBuf



Obs: No priorities between τ's, a's or a's !

May use Σ notation to comactly represent sequential process

$$P = \sum_{i \in I} \alpha_i.P_i$$

Example: Boolean Buffer of Size 2

Action and Process Def.

in_0 :0 is coming as input
in_1 :1 is coming as input
out_0 :0 is going out as output
out_1 :1 is going out as output

Buf^2 : Empty 2-place buffer
Buf^2_0 : 2-place buffer holding 0
Buf^2_{01}: 2-place buffer holding
0 at head and 1 at tail

Buf^2_{01}

$$Buf^2 = in_0.Buf^2_0 + in_1.Buf^2_1$$

$$Buf^2_0 = out_0.Buf^2 + in_0.Buf^2_{00} + in_1.Buf^2_{01}$$

$$Buf^2_1 = out_1.Buf^2 + in_0.Buf^2_{10} + in_1.Buf^2_{11}$$

$$Buf^2_{00} = out_0.Buf^2_0$$

$$Buf^2_{01} = out_0.Buf^2_1$$

$$Buf^2_{10} = out_1.Buf^2_0$$

$$Buf^2_{11} = out_1.Buf^2_1$$

Operators for Concurrent Process

5. Composition

$$\text{Par}_L \quad \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

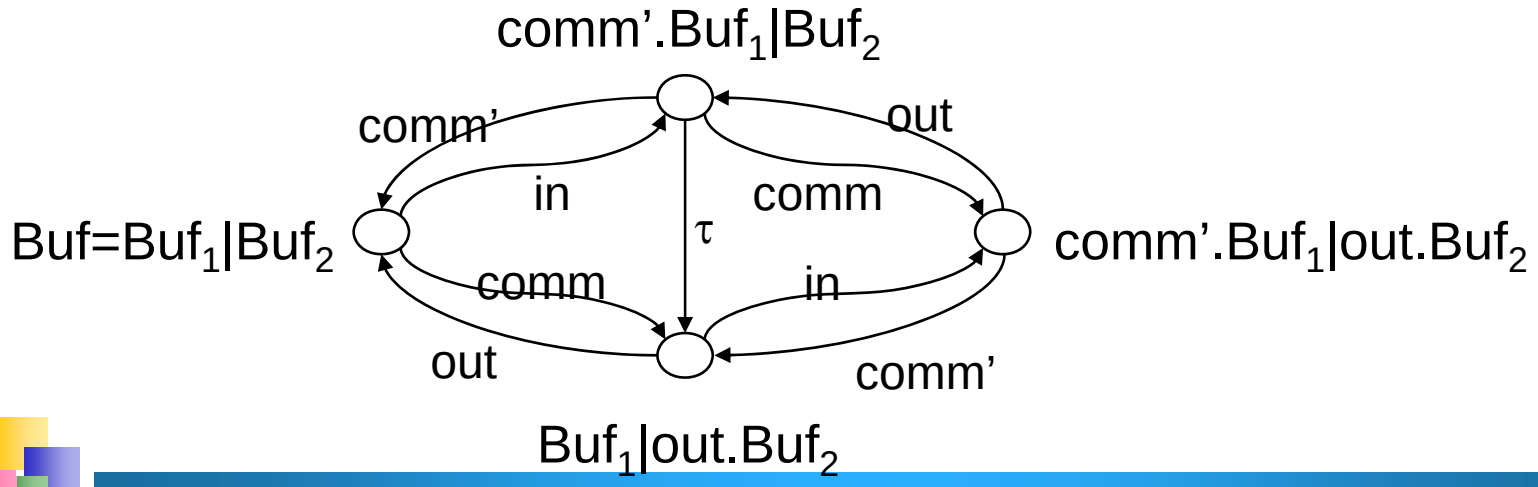
$$\text{Par}_R \quad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

$$\text{Par}_\tau \quad \frac{P \xrightarrow{a} P',\ Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$Buf_1 = in.comm'.Buf_1$
$Buf_2 = comm.out.Buf_2$
$Buf = Buf_1 \mid Buf_2$

Par_L Buf
Par_τ -*in*-> $comm'.Buf_1 \mid Buf_2$
Par_R -τ-> $Buf_1 \mid out.Buf_2$
-*out*-> $Buf_1 \mid Buf_2$

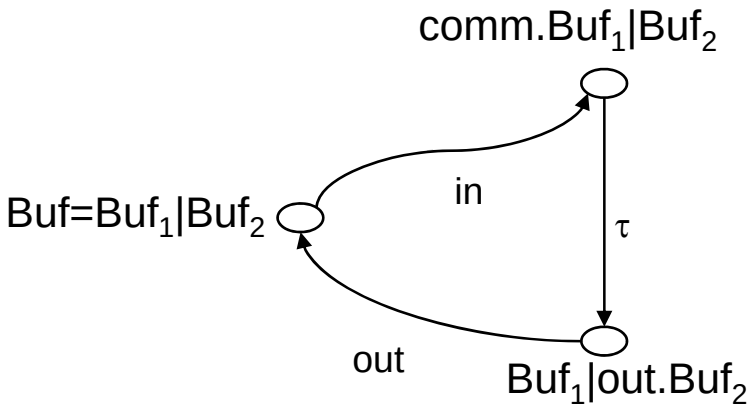
Par_R Buf
Par_R -*comm*-> $Buf_1 \mid out.Buf_2$
-*out*-> $Buf_1 \mid Buf_2$



Operators for Concurrent Process (cont.)

6. Restriction *P\L*

$$\text{Res}\ \frac{P \xrightarrow{\alpha} P'}{P\backslash L \xrightarrow{\alpha} P'\backslash L}\ \alpha \notin L \cup L'$$



$Buf_1 = in.comm.Buf_1$
$Buf_2 = comm'.out.Buf_2$
$Buf=(Buf_1 \mid Buf_2)\backslash\{comm\}$

Buf
-*in*-> $(comm.Buf_1 \mid Buf_2)\backslash\{comm\}$
-τ-> $(Buf_1 \mid out.Buf_2)\backslash\{comm\}$
-*out*-> $(Buf_1 \mid Buf_2)\backslash\{comm\}$

Buf
~~-*comm'*-> $Buf_1 \mid out.Buf_2$~~

(Buf1 | Buf2)\{comm} : a design for buffer with separated input/output ports
ReqBuf = in.out.ReqBuf : a requirement for buffer design
(Buf1 | Buf2)\{comm} == ReqBuf means that buffer design satisfies the requirement

Operators for Concurrent Process (cont.)

7. Relabelling ***P*[*f*]**

$$\text{Rel}\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

$Buf = in.out.Buf$

$Buf_1 = Buf[comm/out]$

$= in.comm.Buf_1$

$Buf_2 = Buf[comm'/in]$

$= comm'.out.Buf_2$

Relabelling function *f* must preserve complements:

$f(a') = f(a)'$

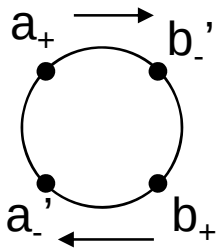
Relabelling function often given by name substitution as above

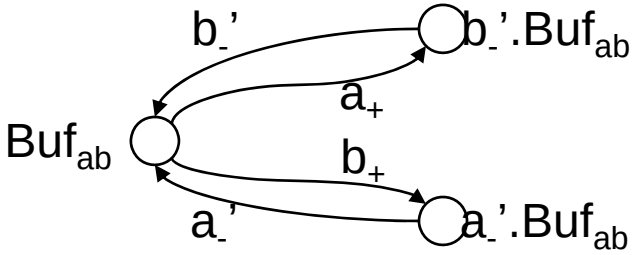
Example: 2-way Buffers

1-place 2-way buffer:

$Buf_{ab} = a_+.b_-'.Buf_{ab} + b_+.a_-'.Buf_{ab}$

LTS:



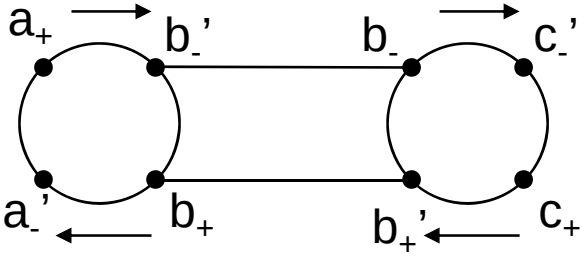


$Buf_{bc} =$

$Buf_{ab}[c_+/b_+, c_-/b_-, b_-/a_+, b_+/a_-]$

(Obs:simultaneous substitution!)

$Sys = (Buf_{ab} \mid Buf_{bc}) \backslash \{b_+, b_-\}$



But what's wrong? Deadlock occurs

In other words, Sys == Buf_{ac}?

Summary of CCS Semantics

$$Act \quad \frac{}{\alpha.P \xrightarrow{\alpha} P}$$

in.P -in-> P

$$Choice_L \quad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad Choice_R \quad \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$$

in.P + out.Q -in-> P *or* –out-> Q

$$Par_L \quad \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad Par_R \quad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

in.P|in'.Q -in->P|in'.Q *or* –in'-> in.P|Q

$$Par\tau \quad \frac{P \xrightarrow{a} P',\ Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

in.P | in'.Q -τ-> P|Q

$$Res \quad \frac{P \xrightarrow{\alpha} P'}{P\backslash L \xrightarrow{\alpha} P'\backslash L} \ \alpha \notin L \cup L'$$

(in.P | in'.Q)\{in} -τ-> (P|Q)\{in} only

$$Rel \quad \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

in.P [out/in] -out-> P[out/in]

Inference of Process Execution

Proof of ((a.E + b.0)| a'.F)\{a} -τ-> (E|F)\{a}

$$\text{Act}\ \frac{}{a.E \xrightarrow{a} E}$$

$$\text{Choice}_L\ \frac{(a.E + b.0) \xrightarrow{a} E}{(a.E + b.0) \xrightarrow{a} E} \qquad \text{Act}\ \frac{}{a'.F \xrightarrow{a'} F}$$

$$\text{Par}\tau\ \frac{(a.E + b.0) \xrightarrow{a} E \qquad a'.F \xrightarrow{a'} F}{(a.E + b.0)|\ a'.F \xrightarrow{\tau} (E|F)}$$

$$\text{Res}\ \frac{(a.E + b.0)|\ a'.F \xrightarrow{\tau} (E|F)}{\mathbf{((a.E + b.0)|\ a'.F)\backslash\{a\} \xrightarrow{\tau} (E|F)\backslash\{a\}}}$$

Exercises

- Derive following process execution from the inference rules
  - (a.E + b.0) | a'.F –a-> E | a'.F
  - (a.E + b.0) | a'.F –a'-> (a.E + b.0) | F
  - (a.E + b.0) | a'.F –b-> 0 | a'.F
  - ((a.E + b.0) | a'.F)\{a} –b-> (0 |a'.F)\{a}
- Draw corresponding labeled transition diagrams
  - (a.E + b.0) | a'.F
  - ((a.E + b.0) | a'.F)\{a}
  - A = a.c'.A, B = c.b'.B
    - A|B, (A|B)\{c}

Proofs

Proof 1

$$\dfrac{\dfrac{\dfrac{}{a.E \xrightarrow{-a->} E}\,\text{Prefix}}{(a.E + b.0) \xrightarrow{-a->} E}\,\text{Choice}_L}{\mathbf{(a.E + b.0) \mid a'.F \xrightarrow{-a->} E \mid a'.F}}\,\text{Par}_L$$

Proof 2

$$\dfrac{\dfrac{}{a'.F \xrightarrow{-a'->} F}\,\text{Prefix}}{\mathbf{(a.E + b.0) \mid a'.F \xrightarrow{-a'->} (a.E + b.0) \mid F}}\,\text{Par}_R$$

Proof 3

$$\dfrac{\dfrac{\dfrac{}{b.0 \xrightarrow{-b->} 0}\,\text{Prefix}}{(a.E + b.0) \xrightarrow{-b->} 0}\,\text{Choice}_R}{\mathbf{(a.E + b.0) \mid a'.F \xrightarrow{-b->} 0 \mid a'.F}}\,\text{Par}_L$$

Labeled Transition Systems