Software Model Checking

Formal Semantics of CCS

Moonzoo Kim

CS Dept. KAIST

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.

Notations

- 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 $Act = L \cup L' \cup \{\tau\}$
  - 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
- 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

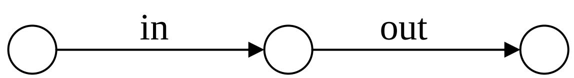
Operators for Sequential Process

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

1.Nil 0 No transitions (deadlock)

Prefix Prefix

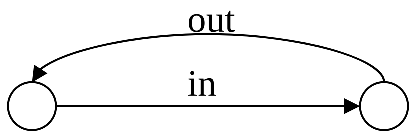
2.Prefix $\alpha.P$ ($\alpha \in Act$) in.out.0 –*in*-> out.0 –*out*-> 0



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

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

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



Operators for Sequential Process (cont.)

4.Choice $P + Q$

BadBuf = in.(τ.0 + out.BadBuf)

Prefix

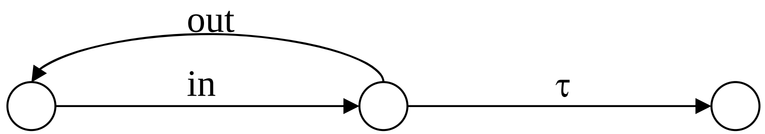
BadBuf –*in*-> τ.0 + out.BadBuf

Choice_L $\quad \dfrac{P \text{ -}\alpha\text{-> } P'}{P+Q \text{ -}\alpha\text{-> } P'}$

Choice_R $\quad \dfrac{Q \text{ -}\alpha\text{-> } Q'}{P+Q \text{ -}\alpha\text{-> } Q'}$

Choice_L $\qquad$ 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

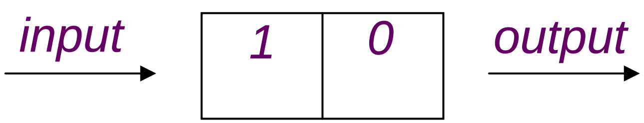
$$P = \sum_{i \in I} \alpha_i . P_i$$

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}

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

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

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

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

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

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

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

5. Composition

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

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

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

Buf_1 = in.comm.Buf_1

Buf_2 = comm'.out.Buf_2

Buf_1 | Buf_2

Par_L -*in*-> comm.Buf_1 | Buf_2

Par_τ -τ-> Buf_1 | out.Buf_2

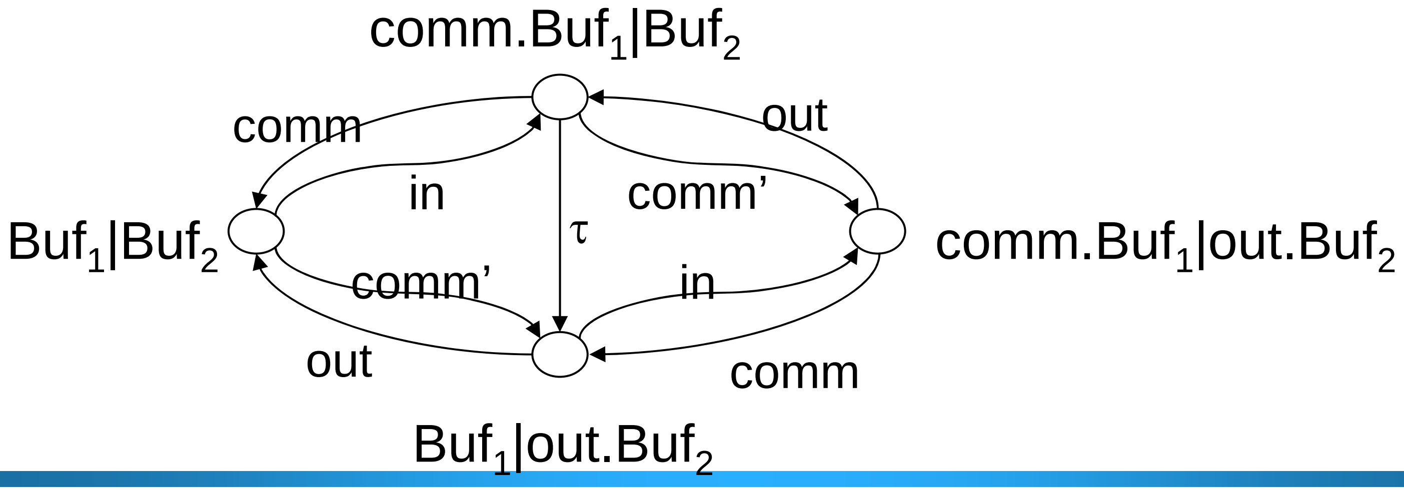
Par_R -*out*-> Buf_1 | Buf_2

But also, for instance:

Buf_1 | Buf_2

Par_R -*comm'*-> Buf_1 | out.Buf_2

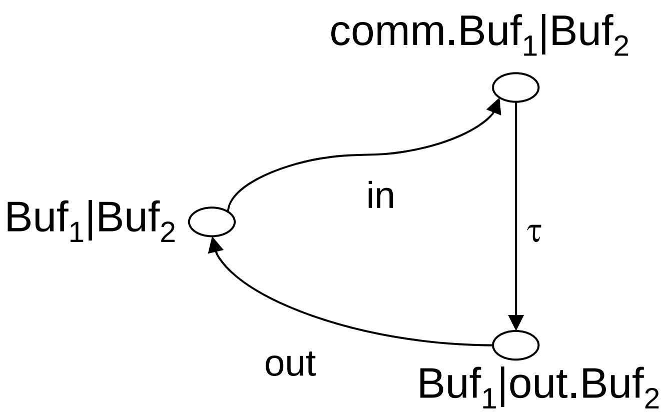
Par_R -*out*-> Buf_1 | Buf_2



Operators for Concurrent Process (cont.)

6. Restriction *P\L*

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



Buf_1 = in.comm.Buf_1
Buf_2 = comm'.out.Buf_2
(Buf_1 | Buf_2)\{comm}
-*in*-> (comm.Buf_1 | Buf_2)\{comm}
-τ-> (Buf_1 | out.Buf_2)\{comm}
-*out*-> (Buf_1 | Buf_2)\{comm}

But *not*:
(Buf_1 | Buf_2)\{comm}
-*comm'*-> Buf_1 | 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)'

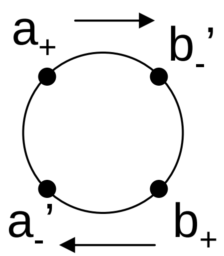
Relabelling function often given by name substituti on as above

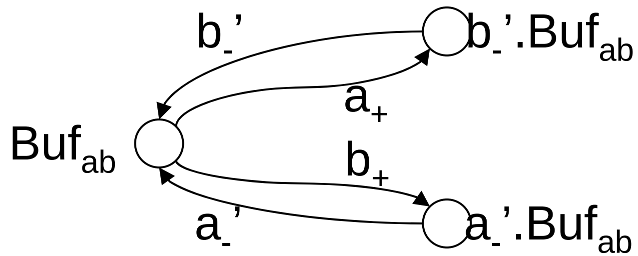
Example: 2-way Buffers

1-place 2-way buffer:

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

LTS:



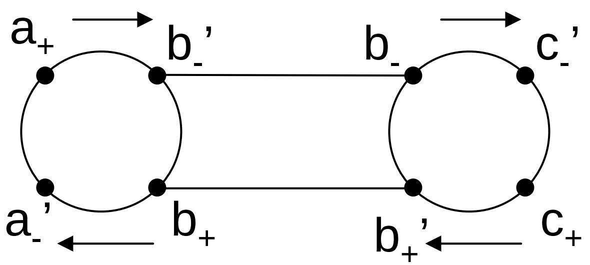


$Buf_{bc} =$

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

(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 $$\frac{}{\alpha.P \ \text{–}\alpha\text{->} \ P}$$

in.P -in-> P

$$Choice_L \ \frac{P \ \text{-}\alpha\text{->} P'}{P+Q \ \text{-}\alpha\text{->} \ P'} \qquad Choice_R \ \frac{Q \ \text{-}\alpha\text{->} Q'}{P+Q \ \text{-}\alpha\text{->} \ Q'}$$

in.P + out.Q -in-> P *or* –out-> Q

$$Par_L \ \frac{P \ \text{-}\alpha\text{->} P'}{P|Q \ \text{-}\alpha\text{->} \ P'|Q} \qquad Par_R \ \frac{Q \ \text{-}\alpha\text{->} Q'}{P|Q \ \text{-}\alpha\text{->} \ P|Q'}$$

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

$$Par\tau \ \frac{P\text{-}a\text{->}P', \ Q\text{–}a'\text{->}Q'}{P|Q \ \text{-}\tau\text{->} \ P'|Q'}$$

in.P | in'.Q - τ-> P|Q

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

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

$$\text{Rel} \ \frac{\text{P -}\alpha\text{->P'}}{\text{P[f] –f(}\alpha\text{)-> P'[f]}}$$

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

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

Act ------------------

a.E –a-> E

Choice$_L$ -------------------------- Act ------------------

(a.E + b.0) -a-> E a'.F –a'-> F

Parτ ---

(a.E + b.0)| a'.F -τ-> (E|F)

Res ---

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

- 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}

Proof 1

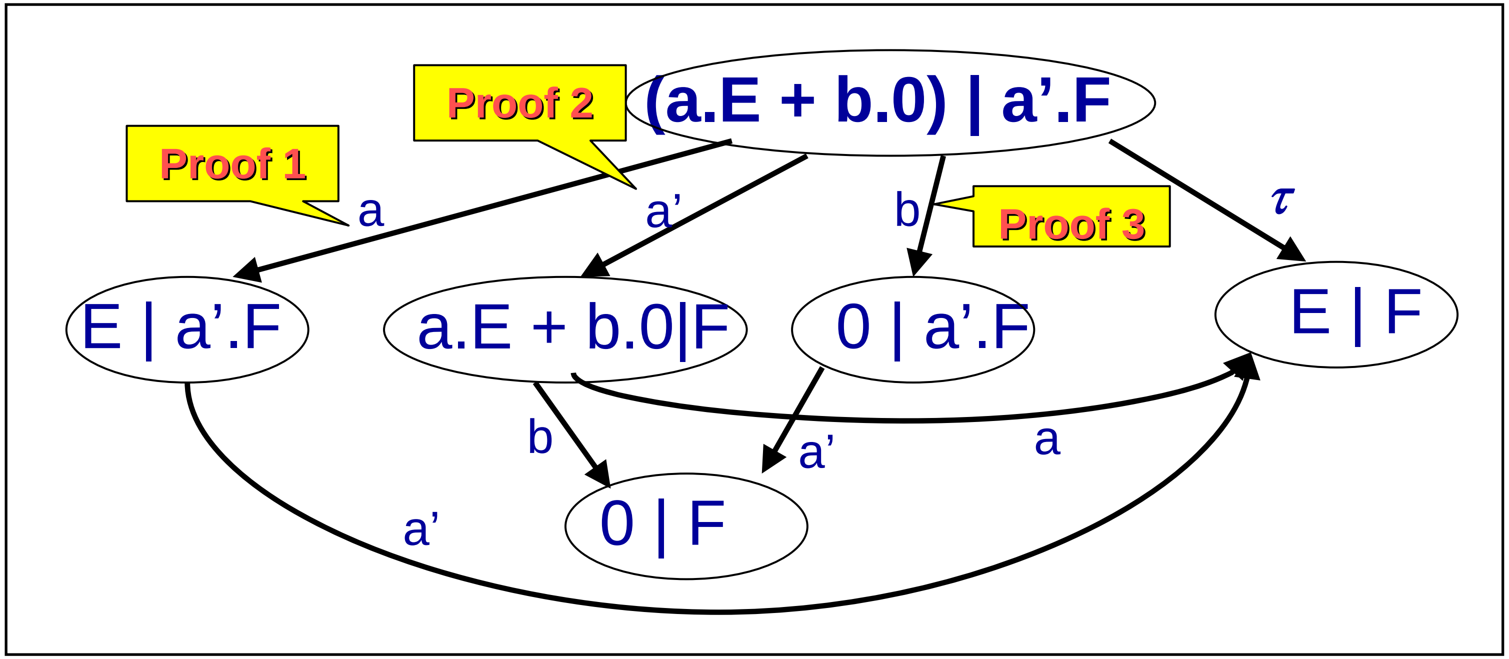
Prefix ——————————
a.E –a-> E
Choice_L ——————————————
(a.E + b.0)) –a-> E
Par_L ————————————————————
(a.E + b.0)) | a’.F –a-> E | a’.F

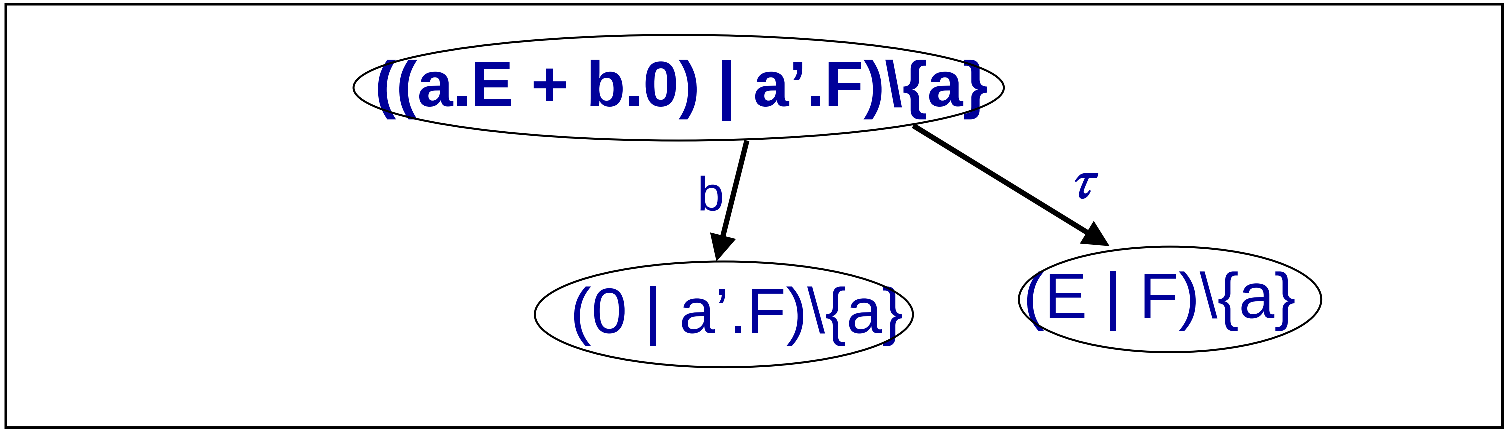
Proof 2

Prefix ——————————
a’.F –a’-> F
Par_R ————————————————————————
(a.E + b.0)) | a’.F –a’-> (a.E + b.0) | F

Proof 3

Prefix ——————————
b.0 –b-> 0
Choice_R ——————————————
(a.E + b.0)) –b-> 0
Par_L ————————————————————
(a.E + b.0)) | a’.F –b-> 0 | a’.F


(a.E + b.0) | a'.F
Proof 1
Proof 2
Proof 3
a
a'
b
τ
E | a'.F
a.E + b.0|F
0 | a'.F
E | F
b
a'
a
a'
0 | F


((a.E + b.0) | a'.F)\{a}
b
τ
(0 | a'.F)\{a}
(E | F)\{a}