Examples of CCS models

Moonzoo Kim
School of Computing
KAIST

Inference of Process Execution

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

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

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

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

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

Exercises

- Derive the following process executions
  - (a.E + b.0) | a'.F
    - (a.E + b.0) | a'.F –t-> E | F
    - (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}
    - ((a.E + b.0) | a'.F)\{a} –t-> (E |F)\{a}
    - ((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

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

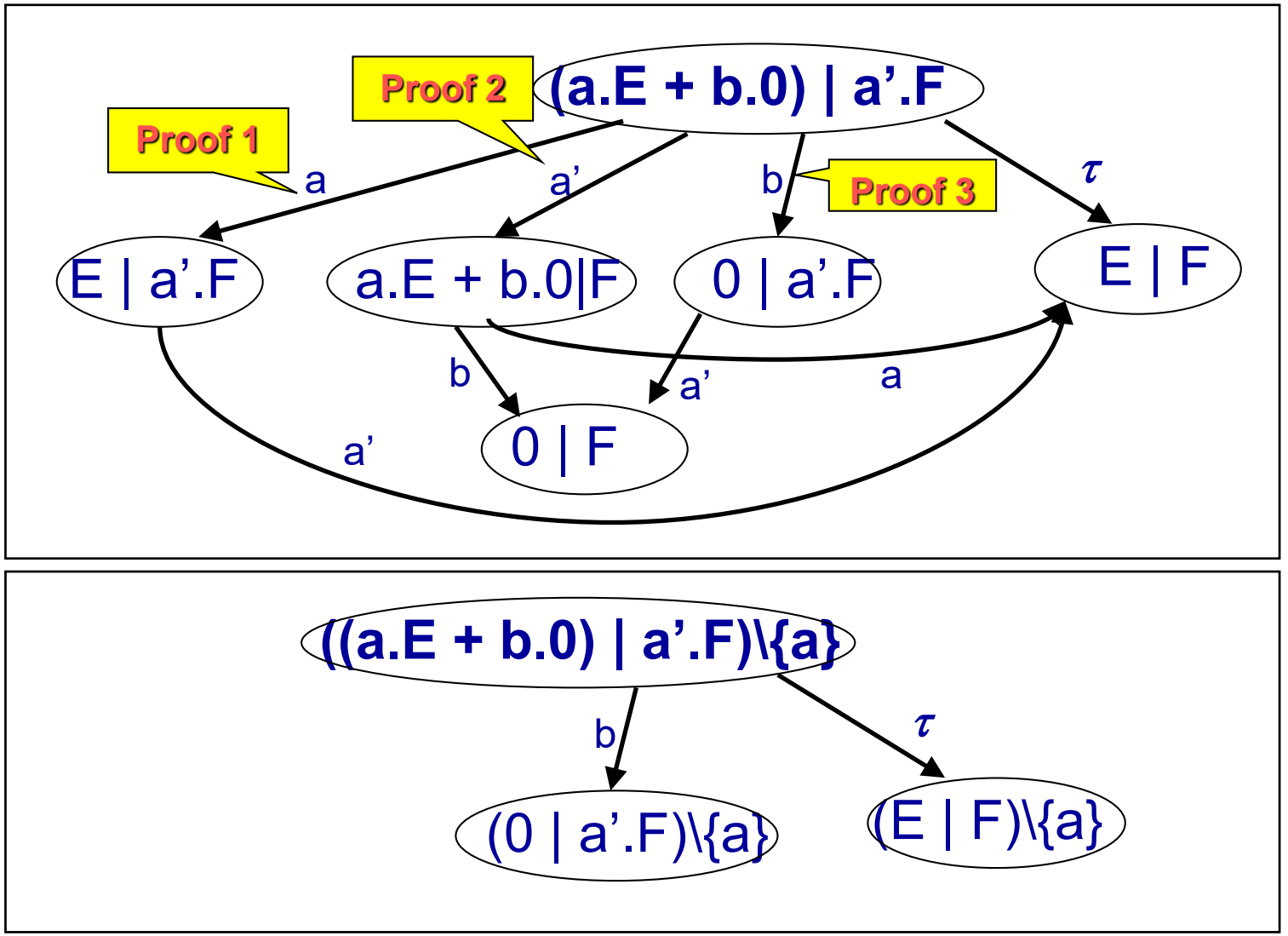
Proof 2

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

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

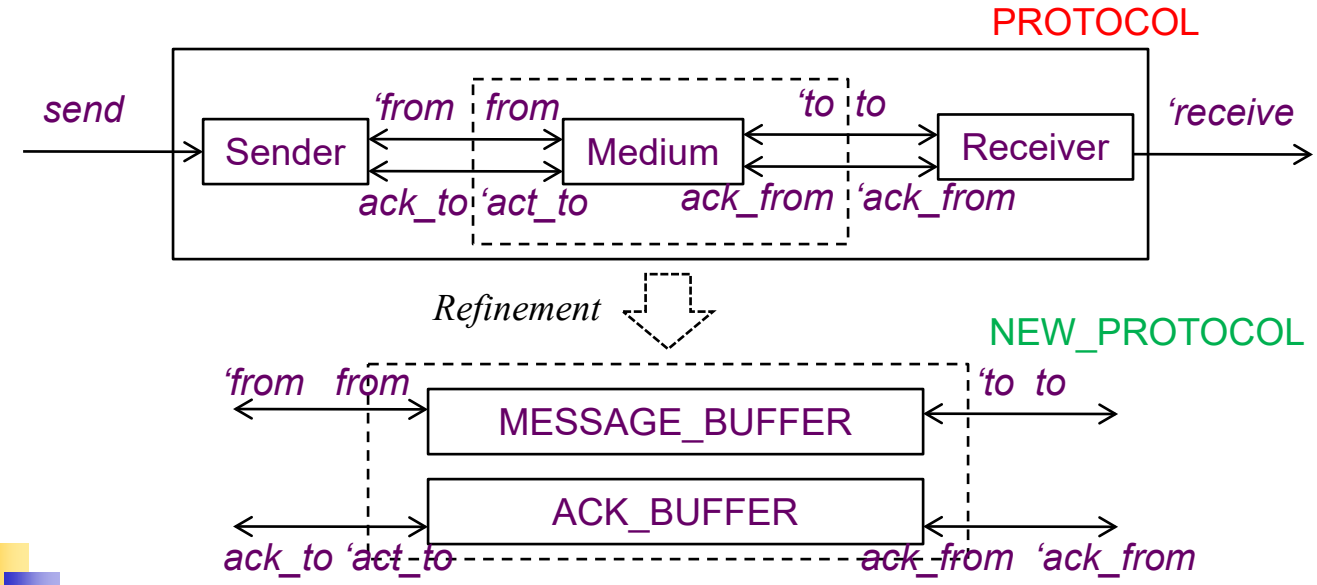


CWB-NC Commands

- 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

Simple Protocol Example

```
proc PROTOCOL =
  (SENDER | MEDIUM | RECEIVER) \ {from,to,ack_from,ack_to}
proc SENDER = send.'from.ack_to.SENDER
proc MEDIUM = from.'to.MEDIUM + ack_from.'ack_to.MEDIUM
proc RECEIVER = to.'receive.'ack_from.RECEIVER

proc NEW_PROTOCOL =
  (SENDER | NEW_MEDIUM | RECEIVER) \ {to, from, ack_to, ack_from}
proc NEW_MEDIUM         =  MESSAGE_BUFFER | ACK_BUFFER
proc MESSAGE_BUFFER =   from.'to.MESSAGE_BUFFER
proc ACK_BUFFER          =   ack_from.'ack_to.ACK_BUFFER
```



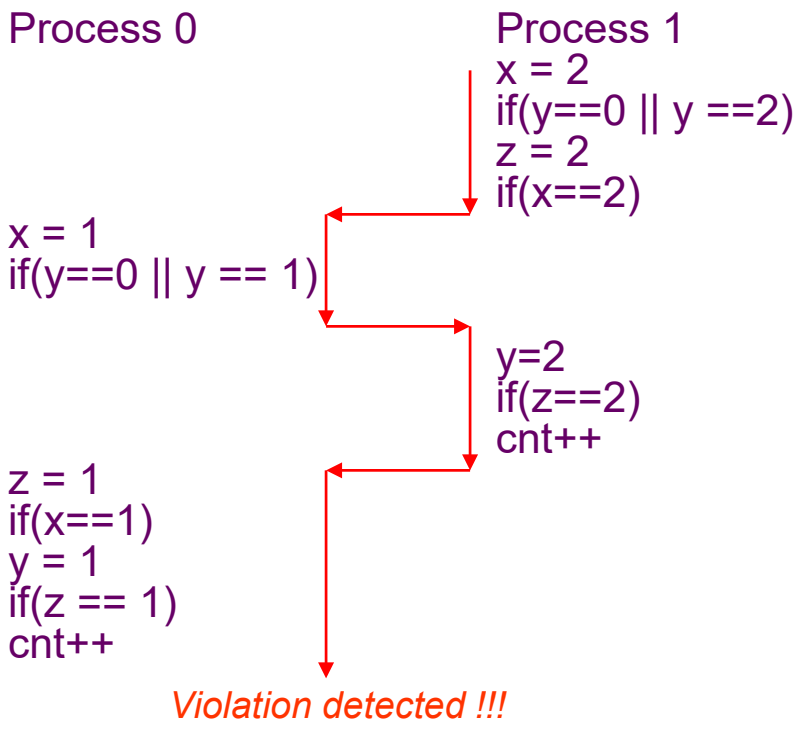
Example: Faulty Mutual Exclusion Protocol (1/2)

```
char cnt=0,x=0,y=0,z=0;

void enter_crit_sect() {
        char me = _pid +1; /* me is 1 or 2*/
again:
        x = me;
        If (y ==0 || y== me) ;
        else goto again;

        z =me;
        If (x == me) ;
        else goto again;

        y=me;
        If(z==me);
        else goto again;

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

Software locks

Critical section

Mutual Exclusion Algorithm



Counter Example

Example: Faulty Mutual Exclusion Protocol (2/2)

```
char cnt=0,x=0,y=0,z=0;

void enter_crit_sect() {
        char me = _pid +1; /* me is 1 or 2*/
again:
        x = me;
        If (y ==0 || y== me) ;
        else goto again;

        z =me;
        If (x == me) ;
        else goto again;

        y=me;
        If(z==me);
        else goto again;

        /* 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
```

KAIST

Homework #1: Due Sep 21

- Draw LTS diagrams of Sys (slide 10 of lecture 3) with proofs for all transitions. Also specify which two actions make τ if any.
- Minimize Sys specification (faulty mutual exclusion in the previous slide) by using relabelling functions
- Specify Peterson's mutual exclusion protocol for 2 processes and verify its correctness using CWB-NC

```
/* Peterson's solution to the mutual exclusion problem - 1981 */
boolean turn, flag[2];
byte ncrit;
active [2] proctype user(){
again:  flag[_pid] = 1;
        turn = _pid;
        while(!(flag[1 - _pid] == 0 || turn == 1 - _pid));

        ncrit++;
        assert(ncrit == 1);          /* critical section */
        ncrit--;

        flag[_pid] = 0;
        goto again;
}
```

Example: Scheduler

Action and Process Def.

a_i: start $task_i$

b_i: stop $task_i$

Requirements:

- $a_1,...,a_n$ to occur cyclically
- a_i/b_i to occur alternately beginning with a_i

$Sched_{i,X}$ for $X \subseteq \{1,...,n\}$

- i to be scheduled
- X pending completion

$Scheduler = Sched_{i,\emptyset}$

$Sched_{i,X}$

$= \Sigma_{j \in X} b_j.Sched_{i,X-\{j\}}$, if $i \in X$

$= \Sigma_{j \in X} b_j.Sched_{i,X-\{j\}}$

$+ a_i.Sched_{i+1,X \cup \{i\}}$, if $i \notin X$