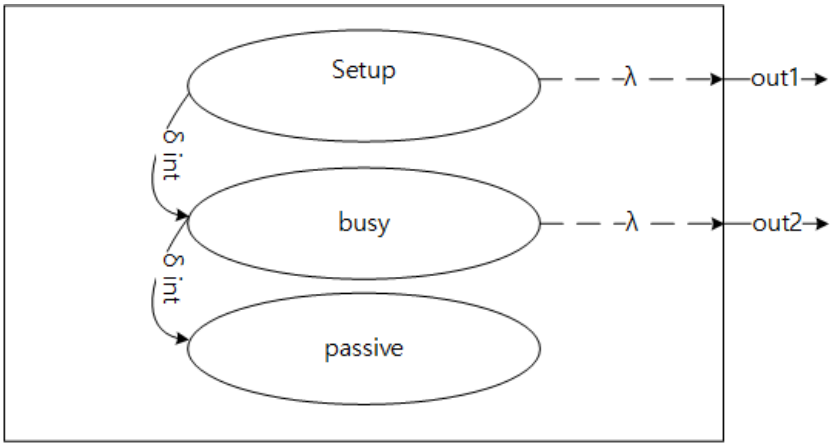
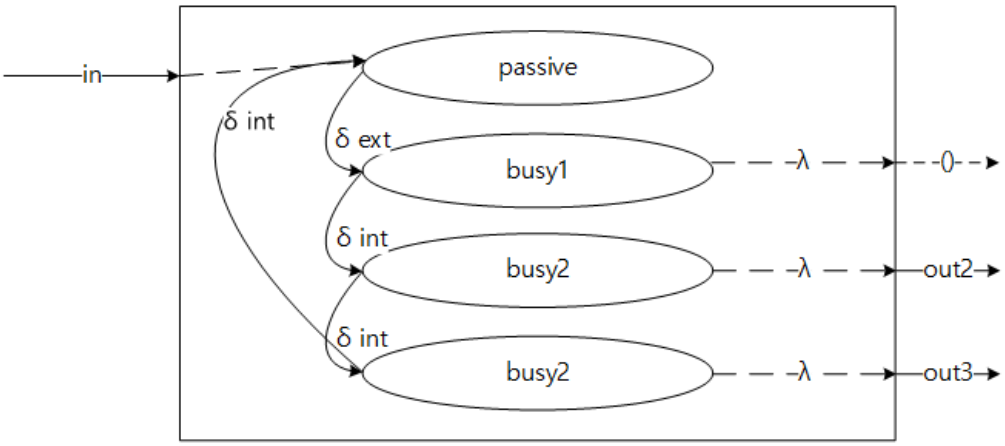
1. nep-to




```
(make-pair atomic-models 'nep-to)
(send nep-to def-state '(job-id))
(send nep-to set-s
          (make-state 'sigma     5
                      'phase     'setup
                      'job-id    '(g1 g2)
          )
)
(define (in-f s)
      (case (state-phase s)
         ('setup (hold-in 'busy 2))
          ('busy  (passivate))
      )
 )
(define (out-f s)
  (case (state-phase s)
     ('setup
              (let ((temp (car (state-job-id s))))
                      (set! (state-job-id s)(cdr(state-job-id s)))
                      (make-content 'port 'out1 'value temp)
      ))
      ('busy
            (let ((temp (car (state-job-id s))))
                      (set! (state-job-id s)(cdr(state-job-id s)))
                      (make-content 'port 'out2 'value temp)
      ))
      (else (make-content))
  )
)
(send nep-to set-int-transfn in-f)
(send nep-to set-outputfn out-f)
```

```
OK
[2] (load "mbase/nep-to.m")

Model of type atomic-models with name NEP-TO made.
Processor of type simulators with name S:NEP-TO made.
OK
[3] (send nep-to output?)
output y =     output y = OUT1 G1#(((|#!STRUCTURE| . CONTENT)) OUT1 G1)
[4] (send nep-to int-transition)
state s =
                                                                state s = (2 BUSY
(G2))state s = ()()
[5] (send nep-to output?)
output y =       output y = OUT2 G2#(((|#!STRUCTURE| . CONTENT)) OUT2 G2)
[6] (send nep-to int-transition)
state s =
                                                            state s = (INF PASSIVE
-)state s = ()()
[7] (transcript-off)
```

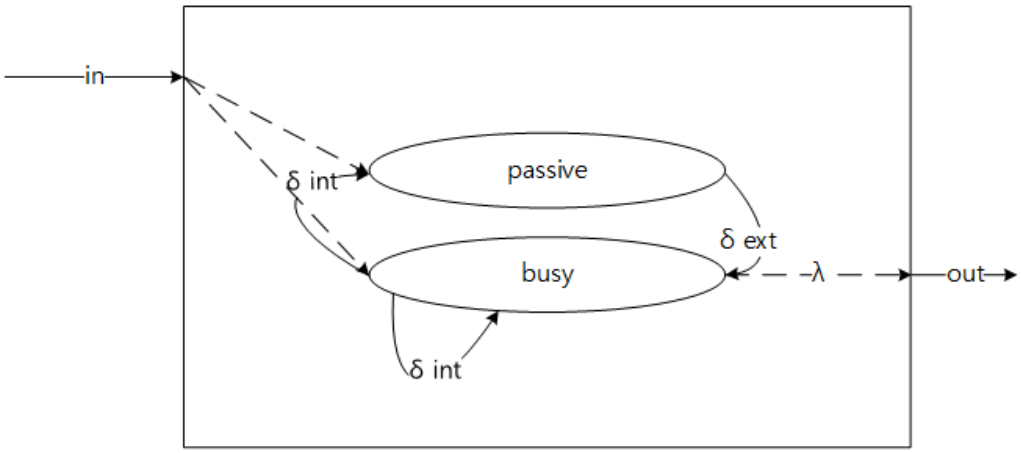
2. tsp




```
(make-pair atomic-models 'tsp)
(send tsp def-state
  '(
     job-id
     processing-time1
     processing-time2
     processing-time3
   )
)
(send tsp set-s
  (make-state 'sigma     'inf
              'phase     'passive
              'job-id    '()
              'processing-time1 2
              'processing-time2 3
              'processing-time3 4
  )
)
(define (ex-f s e x)
      (case (content-port  x)
            ('in (case (state-phase s)
                 ('passive (set! (state-job-id s) (content-value x))
                           (hold-in 'busy1 (state-processing-time1 s))
                 )
                 ('busy1 (continue))('busy2 (continue))('busy3 (continue))
                 )
            )
     )
)
(define (in-f s)
 (case (state-phase s)
   ('busy1 (hold-in 'busy2 (state-processing-time2 s)))
   ('busy2 (hold-in 'busy3 (state-processing-time3 s)))
   ('busy3 (passivate))
))
(define (out-f s)
 (case (state-phase s)
   ('busy1
      (make-content 'port '() 'value ())
   )
   ('busy2
      (make-content 'port 'out1 'value (state-job-id s))
   )
   ('busy3
      (make-content 'port 'out2 'value (state-job-id s))
   )
 (else (make-content))
 )
)
(send tsp set-ext-transfn ex-f)
(send tsp set-int-transfn in-f)
(send tsp set-outputfn out-f)
```

```
OK
[14] (load "mbase/tsp.m")

Model of type atomic-models with name TSP made.
Processor of type simulators with name S:TSP made.
OK
[15] (send tsp inject 'in 'g1 5)
state s =
                                                                    state s = (2
BUSY1 G1 2 3 4)state s = ()()
[16] (send tsp output?)
output y =      output y = () ()#((((|#!STRUCTURE| . CONTENT)) () ())
[17] (send tst int-transition)

[VM ERROR encountered!] Variable not defined in current environment
TST

[Inspect] Where am I?
Stack frame for ()

[Inspect] Quit
[18] (send tsp int-transition)
state s =
                                                                    state s = (3 BUSY2 G1
2 3 4)state s = ()()
[19] (send tsp output?)
output y =      output y = OUT1 G1#((((|#!STRUCTURE| . CONTENT)) OUT1 G1)
[20] (send tsp int-transition)
state s =
                                                                    state s = (4 BUSY3 G1
2 3 4)state s = ()()
[21] (send tsp output?)
output y =       output y = OUT2 G1#((((|#!STRUCTURE| . CONTENT)) OUT2 G1)
[22] (send tsp int-transition)
state s =
                                                                    state s = (INF
PASSIVE G1 2 3 4)state s = ()()
[23] (transcript-off)
```

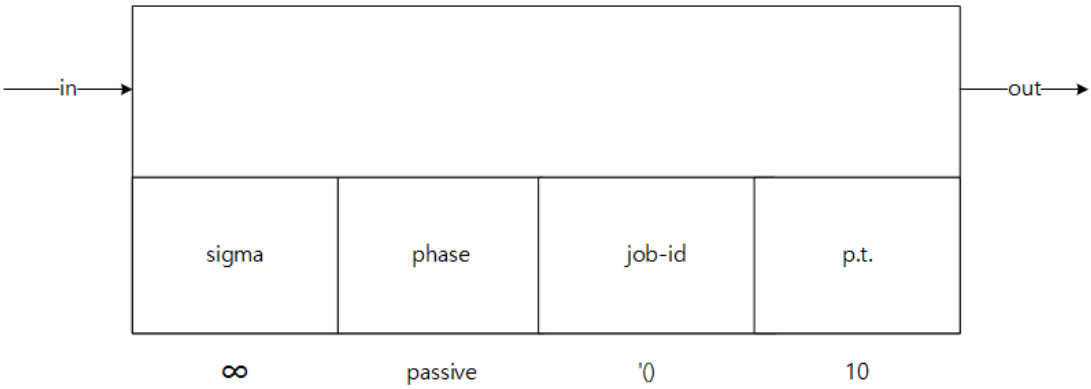
3. ps

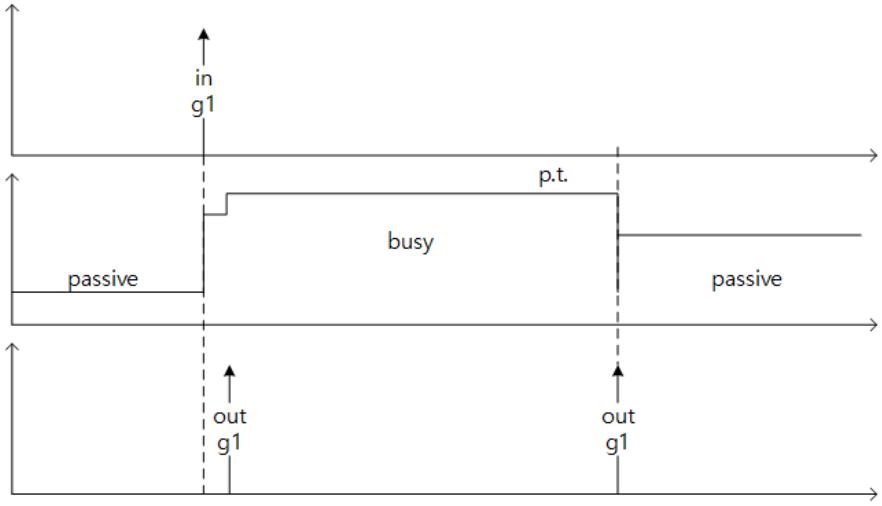


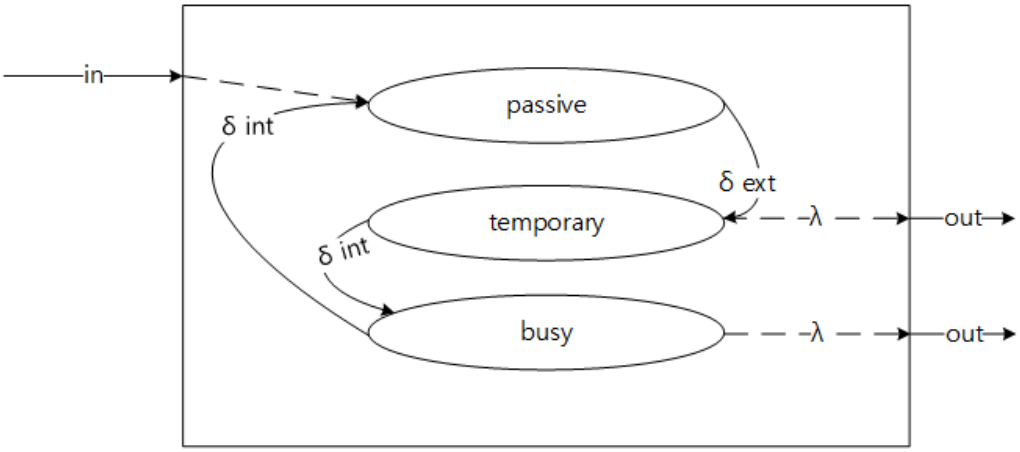

```
(make-pair atomic-models 'ps)
(send ps def-state '(job-id stack processing-time))
(send ps set-s
          (make-state   'sigma          'inf
                        'phase          'passive
                        'job-id         '()
                        'stack          '()
                        'processing-time  10
          )
)
(define (ext-ps s e x)
       (case (content-port x)
            ('in (case (state-phase s)
                       ('passive
                             (set! (state-job-id s) (content-value x))
                             (hold-in 'busy (state-processing-time s))
                       )
                       ('busy
                             (set! (state-stack s)
                                 (append (list (content-value x))(state-stack s))
                             )
                             (continue)
                       )
                 )
            )
       )
)
(define(int-ps s)
       (case (state-phase s)
            ('busy (if (null? (state-stack s))
                       (passivate)
                       (begin (set! (state-job-id s) (car(state-stack s)))
                             (set! (state-stack s)  (cdr(state-stack s)))
                             (hold-in 'busy (state-processing-time s))
                       )
                   )
            )
       )
)
(define (out-ps s)
       (case (state-phase s)
            ('busy  (make-content  'port 'out 'value (state-job-id s)))
            (else   (make-content))
       )
)
(send ps set-ext-transfn ext-ps)
(send ps set-int-transfn int-ps)
(send ps set-outputfn out-ps)
```

```
OK
[25] (load "mbase/ps.m")

Model of type atomic-models with name PS made.
Processor of type simulators with name S:PS made.
OK
[26] (send ps inject 'in 'g1 3)
state s =
                                                                    state s =
(10 BUSY G1 - 10)state s = ()()
[27] (send ps inject 'in 'g2 4)
state s =
                                                                 state s = (6
BUSY G1 (G2) 10)state s = ()()
[28] (send ps inject 'in 'g3 6)
state s =
                                                                  state s = (0
BUSY G1 (G3 G2) 10)state s = ()()
[29] (send ps output?)
output y =     output y = OUT G1#((((|#!STRUCTURE| . CONTENT)) OUT G1)
[30] (send ps int-transition)
state s =
                                                                    state s =
(10 BUSY G3 (G2) 10)state s = ()()
[31] (send ps output?)
output y =      output y = OUT G3#((((|#!STRUCTURE| . CONTENT)) OUT G3)
[32] (send ps int-transition)
state s =
                                                                   state s =
(10 BUSY G2 - 10)state s = ()()
[33] (send ps output?)
output y =      output y = OUT G2#((((|#!STRUCTURE| . CONTENT)) OUT G2)
[34] (send ps int-transition)
state s =
                                                                 state s =
(INF PASSIVE G2 - 10)state s = ()()
[35] (transcript-off)
```

4. io


in
out
sigma
phase
job-id
p.t.
∞
passive
'()
10

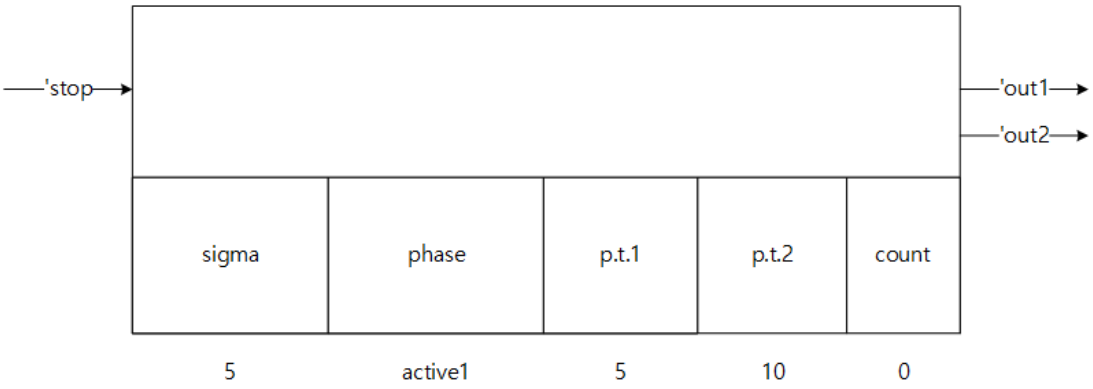

in
g1
p.t.
busy
passive
passive
out
g1
out
g1

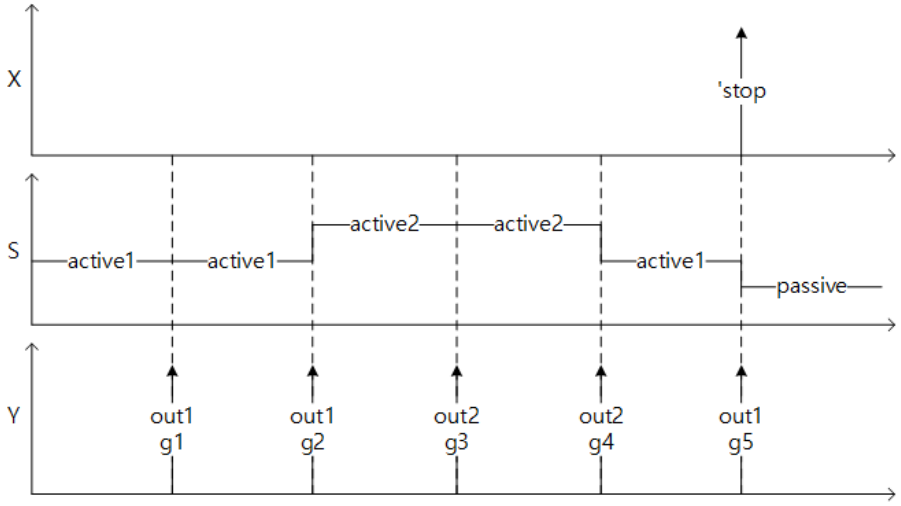

in
δ int
passive
δ ext
temporary
δ int
λ
out
busy
λ
out

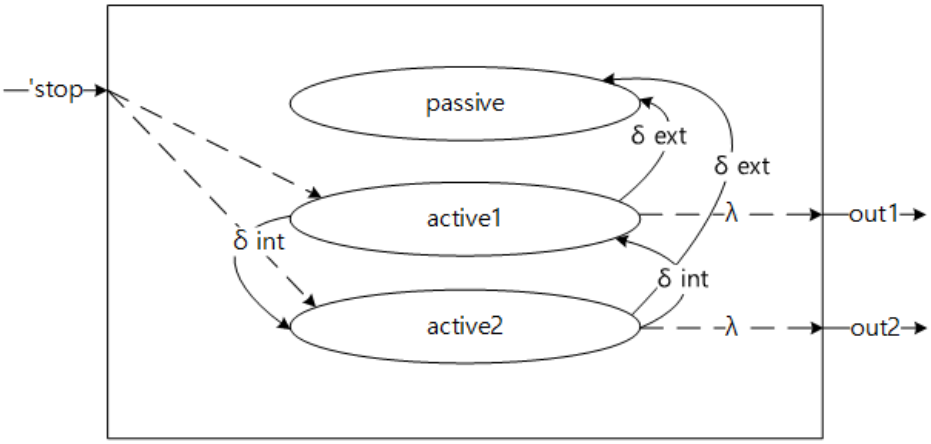
```
(make-pair atomic-models 'io)
(send io def-state
        '(
          job-id
          processing-time  ; processing time of this processor
         )
)
(send io set-s
        (make-state 'sigma     'inf
                    'phase     'passive
                    'processing-time 10

        )
)
(define (ex-f s e x)
      (case (content-port  x)
           ('in (case (state-phase s)
                ('passive
                        (set! (state-job-id s) (content-value x))
                        (hold-in 'temporary 0)
                )
                ('busy (continue))
                )
           )
     )
)
(define (in-f s)
 (case (state-phase s)
   ('temporary (hold-in 'busy (state-processing-time s)))
   ('busy (passivate))
))
(define (out-f s)
 (case (state-phase s)
   ('busy
       (make-content 'port 'out 'value (state-job-id s))
   )
   ('temporary
(make-content 'port 'out 'value (state-job-id s))
   )
 (else (make-content))
 )
)
(send io set-ext-transfn ex-f)
(send io set-int-transfn in-f)
(send io set-outputfn out-f)
```

```
OK
[2] (load "mbase/io.m")

Model of type atomic-models with name IO made.
Processor of type simulators with name S:IO made.
OK
[3] (send io inject 'in 'g1 5)
state s =
                                                                state s = (0
TEMPORARY G1 10)state s = ()()
[4] (send io output?)
output y =     output y = OUT G1#((((|#!STRUCTURE| . CONTENT)) OUT G1)
[5] (send io int-transition)
state s =
                                                                state s = (10 BUSY
G1 10)state s = ()()
[6] (send io output?)
output y =      output y = OUT G1#((((|#!STRUCTURE| . CONTENT)) OUT G1)
[7] (send io int-transition)
state s =
                                                            state s = (INF PASSIVE
G1 10)state s = ()()
[8] (transcript-off)
```

5. cg2


'stop
'out1
'out2
sigma
phase
p.t.1
p.t.2
count
5
active1
5
10
0

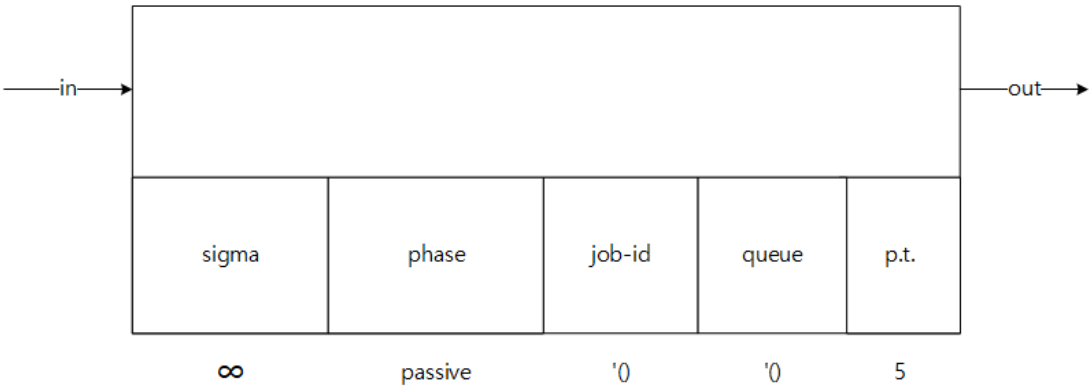

X
'stop
S
active1
active1
active2
active2
active1
passive
Y
out1
g1
out1
g2
out2
g3
out2
g4
out1
g5

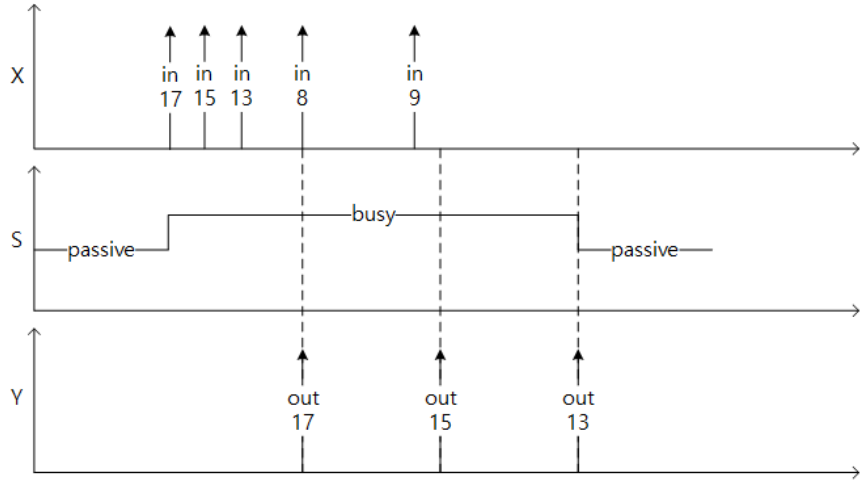

'stop
passive
δ ext
δ ext
active1
λ
out1
δ int
δ int
active2
λ
out2

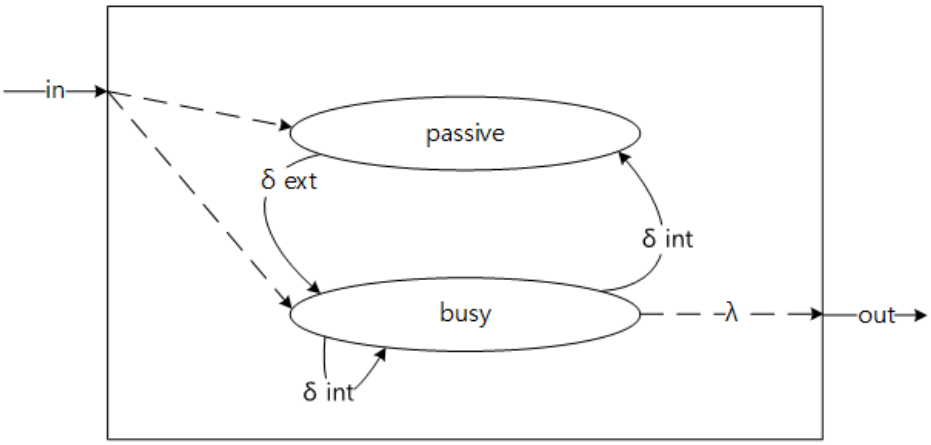
```
(make-pair atomic-models 'cg2)
(send cg2 def-state '(inter-arrival-time1 inter-arrival-time2 index))
(send cg2 set-s (make-state 'sigma              5
                        'phase            'active1
                        'inter-arrival-time1 5
                        'inter-arrival-time2 10
                        'index 0)
)
(define (ext-cg2 s e x)
    (case (content-port x)
        ('stop
            (passivate)  ;when receive stop signal passivate
        )
        (else (continue))
))
(define (int-cg2 s)
    (case (state-phase s)
        ('active1
        (set! (state-index s)(+ 1 (state-index s)))
        (if (= 0 (remainder (state-index s) 2))
            (hold-in 'active2 (state-inter-arrival-time2 s))
            (set! (state-sigma s) (state-inter-arrival-time1 s))
        );if
    )
        ('active2
           (set! (state-index s)(+ 1 (state-index s)))
           (if (= 0 (remainder (state-index s) 2))
               (hold-in 'active1 (state-inter-arrival-time1 s))
               (set! (state-sigma s) (state-inter-arrival-time2 s))
            );if
        )
))
(define (out-cg2 s)
    (case (state-phase s)
        ('active1
            (make-content 'port 'out1 'value (gensym 'job-))
        )
        ('active2
            (make-content 'port 'out2 'value (gensym 'job-))
        )
        (else (make-content))
))
(send cg2 set-int-transfn int-cg2)
(send cg2 set-ext-transfn ext-cg2)
(send cg2 set-outputfn out-cg2)
```

```
OK
[33] (load "mbase/cg2.m")

Model of type atomic-models with name CG2 made.
Processor of type simulators with name S:CG2 made.
OK
[34] (send cg2 output?)
output y =         output y = OUT1 JOB-15#(((|#!STRUCTURE| . CONTENT)) OUT1 JOB-15)
[35] (send cg2 int-transition)
state s =
                                                                        state s = (5
ACTIVE1 5 10 1)state s = ()()
[36] (send cg2 output?)
output y =           output y = OUT1 JOB-16#(((|#!STRUCTURE| . CONTENT)) OUT1 JOB-16)
[37] (send cg2 int-transition)
state s =
                                                                        state s = (10
ACTIVE2 5 10 2)state s = ()()
[38] (send cg2 output?)
output y =           output y = OUT2 JOB-17#(((|#!STRUCTURE| . CONTENT)) OUT2 JOB-17)
[39] (send cg2 int-transition)
state s =
                                                                         state s = (10
ACTIVE2 5 10 3)state s = ()()
[40] (send output?)

[Syntax Error] Invalid identifier in expression
()
(ACCESS () %SC-ENV)
[Returning to top level]
[41] (send cg2 output?)
output y =           output y = OUT2 JOB-18#(((|#!STRUCTURE| . CONTENT)) OUT2 JOB-18)
[42] (send cg2 int-transition)
state s =
                                                                         state s = (5
ACTIVE1 5 10 4)state s = ()()
[43] (transcript-off)
```

6. selp


in
out
sigma
phase
job-id
queue
p.t.
∞
passive
'0
'0
5

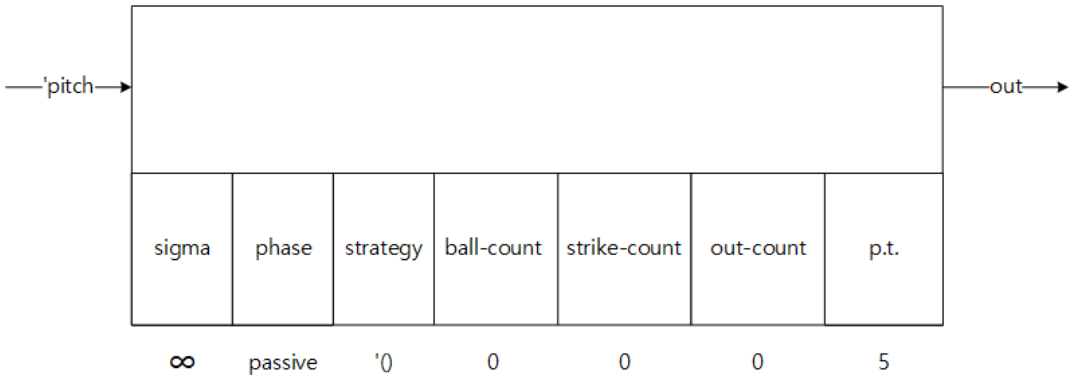

X
in 17
in 15
in 13
in 8
in 9
S
passive
busy
passive
Y
out 17
out 15
out 13

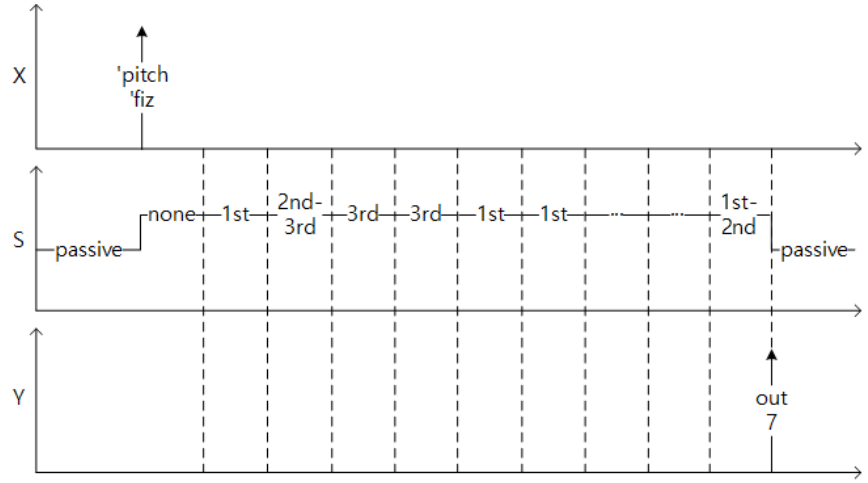

in
passive
δ ext
δ int
busy
λ
out
δ int

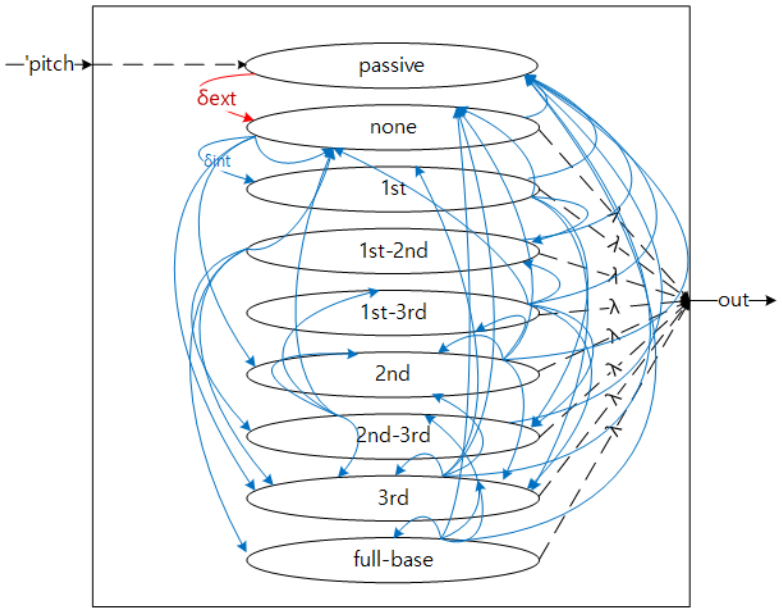
```
(make-pair atomic-models 'selp)
(send selp def-state
      '(
           job-id
           queue
           processing-time
       )
)
(send selp set-s
          (make-state   'sigma          'inf
                      'phase          'passive
                      'job-id         '()
                      'queue         '()
                      'processing-time  5
          )
)
(define (ext-selp s e x)
  (case (content-port x)
    ('in
      (if ( < 9 (content-value x))
        (case (state-phase s)
          ('passive
            (set! (state-job-id s) (content-value x))
            (hold-in 'busy (state-processing-time s))
          )
          ('busy
            (set! (state-queue s)
             (append  (state-queue s) (list (content-value x)))
            )
            (continue)
          )
        );case
        (continue)
      );if
    )
  )
)
(define(int-selp s)
       (case (state-phase s)
            ('busy (if (null? (state-queue s))
                       (passivate)
                       (begin (set! (state-job-id s) (car(state-queue s)))
                              (set! (state-queue s)  (cdr(state-queue s)))
                              (hold-in 'busy (state-processing-time s))
                       )
                   )
            )
       )
)
(define (out-selp s)
       (case (state-phase s)
            ('busy  (make-content  'port 'out 'value (state-job-id s)))
            (else   (make-content))
       )
)

(send selp set-ext-transfn ext-selp)
(send selp set-int-transfn int-selp)
(send selp set-outputfn out-selp)
```

```
OK
[10] (load "mbase/selp.m")

Model of type atomic-models with name SELP made.
Processor of type simulators with name S:SELP made.
OK
[11] (send selp inject 'in 17 5)
state s =
                                                                state s = (5 BUSY
17 - 5)state s = ()()
[12] (send selp inject 'in 15 1)
state s =
                                                            state s = (4 BUSY 17
(15) 5)state s = ()()
[13] (send selp inject 'in 13 1)
state s =
                                                              state s = (3 BUSY 17
(15 13) 5)state s = ()()
[14] (send selp inject 'in 8 3)
state s =
                                                                state s = (0 BUSY
17 (15 13) 5)state s = ()()
[15] (send selp output?)
output y =     output y = OUT 17#((((|#!STRUCTURE| . CONTENT)) OUT 17)
[16] (send selp int-transition)
state s =
                                                                state s = (5 BUSY
15 (13) 5)state s = ()()
[17] (send selp inject 'in 9 5)
state s =
                                                              state s = (0 BUSY 15
(13) 5)state s = ()()
[18] (send selp output?)
output y =      output y = OUT 15#((((|#!STRUCTURE| . CONTENT)) OUT 15)
[19] (send selp int-transition)
state s =
                                                              state s = (5 BUSY 13
- 5)state s = ()()
[20] (send output?)

[Syntax Error] Invalid identifier in expression
()
(ACCESS () %SC-ENV)
[Returning to top level]
[21] (send selp output?)
output y =      output y = OUT 13#((((|#!STRUCTURE| . CONTENT)) OUT 13)
[22] (send selp int-transition)
state s =
                                                            state s = (INF PASSIVE
13 - 5)state s = ()()
[23] (transcript-off)
```

7. sbb





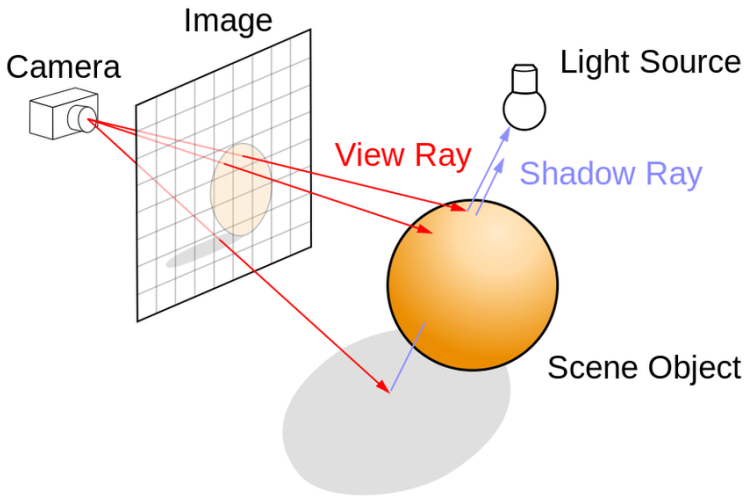


**sbb.m 와 sbb.tst 코드 길이가 너무 길어 첨부합니다.

8. custom – pseudo Ray Intersection Simulator

A. description

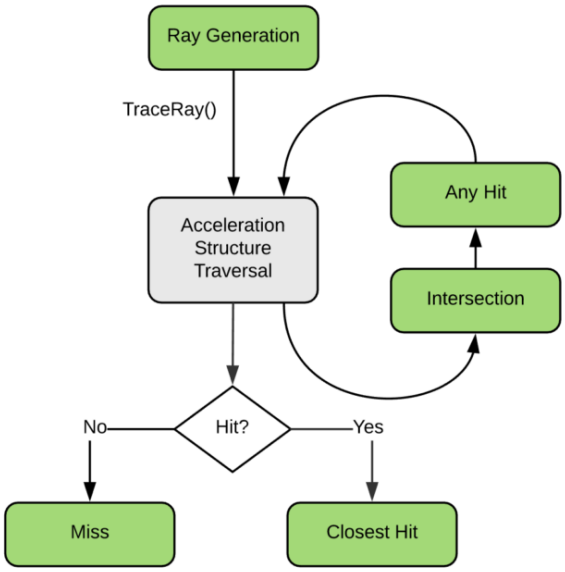
해당 모델은 컴퓨터 그래픽스 방법론 중 하나인 레이트레이싱 알고리즘의 일부분을 모방하였다. 이미지를 향해 쏜 광선이 물체와 교차를 하는지 여부를 시뮬레이션한다.



[그림 1] Illustration of the ray tracing algorithm

(출처: Irradiance modelling for bi-facial PV modules using the ray tracing technique)

다음 이미지는 Nvidia사의 RTX에서 사용하는 레이트레이싱 알고리즘을 도식화한 것이다. 이 중 Any Hit과 Intersection 부분은 실제로 구현하지 않고 random함수로 대체하였다.



[그림 2] The ray tracing pipeline

(출처: https://devblogs.nvidia.com/introduction-nvidia-rtx-directx-ray-tracing/)

레이트레이싱은 화면을 향해 화면의 픽셀마다 원하는 샘플의 개수만큼 광선을 쏜다. 수많은 광선을 순서대로 처리하기 위해 pq모델을 변형하여 만들었다.

external-transition은 기존의 pq모델과 역할이 거의 동일하다. passive phase를 바꾸는 역할을 하며, passive가 아닌 phase에 추가의 input이 발생했을 때, 이를 queue에 집어넣는 역할도 한다.

```
(define (ex-f s e x)
      (case (content-port  x)
           ('pixel (case (state-phase s)
                       ('passive (set! (state-ray-id s) (content-value x))
                            (hold-in 0 (state-processing-time s))
                        )
                       (else
              (set!(state-queue s)(append (state-queue s)(list (content-value x)))
                        )
                       (continue)
                     )
                  )
            )
      )
)
```

internal-transition의 결과는 rand-isect가 반환하는 값에 의해 정해진다.

rand-isect 함수는 30%의 확률로 'hit을 반환하고 70%의 확률로 'miss를 반환한다. rand-isect이 'hit을 반환하면 internal-transition은 내부적으로 phase의 숫자를 하나 증가시키며, 'miss를 반환하면 phase를 'done이나 'miss 로 바꾼다.

숫자로 구성된 phase는 광선이 물체에 부딪힌 횟수, depth를 나타낸다. 프로그램이 광선을 무한정 추적할 수 없으므로 max-depth를 상한선으로 두고 이를 넘기면 강제로 'done phase에 돌입하게 한다. 혹은 추적된 광선이 물체와 부딪힌 후, 다시 물체와 부딪히지 않고 빗나간다면 이 역시 'done phase에 돌입하게 한다.

다만, 한 번도 물체와 맞닿지 않고 처음부터 빗나간다면, 즉 phase가 0인 상태에서 rand-isect이 'miss를 반환한다면 phase 역시 miss가 된다.

```
(define (in-f s)
       (cond
          ((equal? 'done (state-phase s))
             (if (null? (state-queue s))
                    (passivate)
                    (begin (set! (state-ray-id s) (car(state-queue s)))
                           (set! (state-queue s)  (cdr(state-queue s)))
                           (hold-in 0 (state-processing-time s))
                     )
             )
          )
       ((equal? 'miss (state-phase s))
             (if (null? (state-queue s))
                    (passivate)
                    (begin (set! (state-ray-id s) (car(state-queue s)))
                          (set! (state-queue s)  (cdr(state-queue s)))
                          (hold-in 0 (state-processing-time s))
             )
             )
          )
          ((< (state-phase s)(state-max-depth s))
             (let ((n (rand-isect)))
                    (case n
                    ('miss (if (= (state-phase s) 0)
                           (hold-in 'miss (state-processing-time s))
                           (hold-in 'done (state-processing-time s))
                           )
                    )
                    ('hit  (hold-in (+ (state-phase s) 1)(state-processing-time s)))
                    )
             );let
          )
       )
)
```

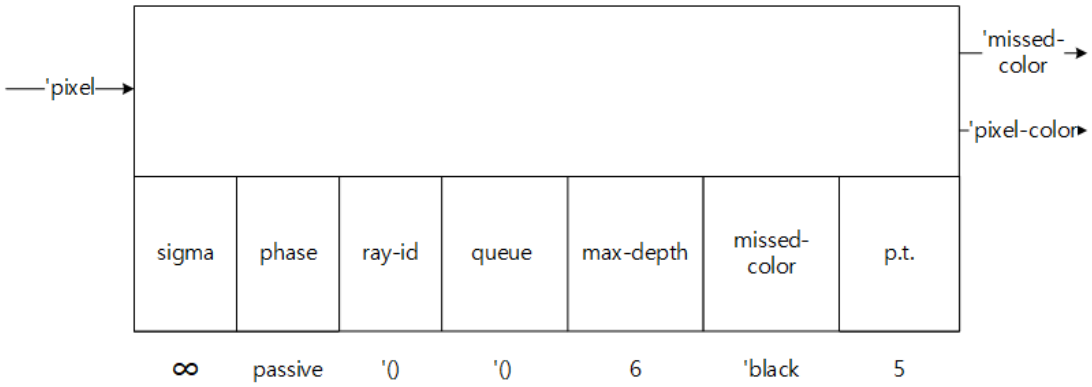
phase가 'done이나 'miss인 경우에만 output function이 작동하는데, 'done인 경우 작업된 광선의 번호를 'pixel-color port에 반환한다. 'miss인 경우 작업된 광선과 관계없이 바탕색(현재 모델에서 missed-color로 지정된) 'black을 'missed-color port로 반환한다.

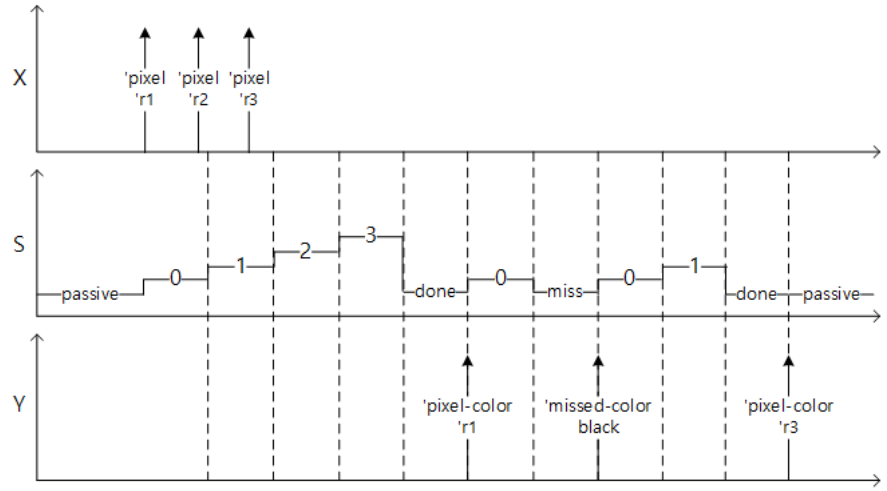
```
(define (out-f s)
       (case (state-phase s)
              ('done (make-content 'port 'pixel-color 'value (state-ray-id s)))
       ('miss (make-content 'port 'missed-color 'value (state-missed-color s)))
              (else (make-content))
       )
)
```

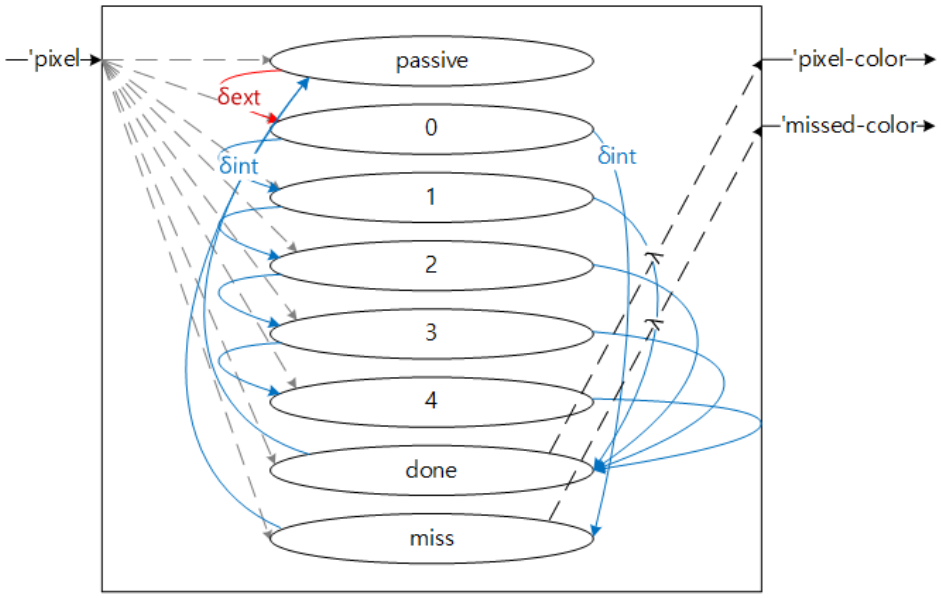
done 이나 miss 인 상태에서 queue 에 작업할 광선이 남았다면 dequeue 하여 작업하고 queue 가 비었다면 passivate 한다.

```
((equal? 'done (state-phase s))
              (if (null? (state-queue s))
                     (passivate)
                     (begin (set! (state-ray-id s) (car(state-queue s)))
                     (set! (state-queue s)  (cdr(state-queue s)))
                     (hold-in 0 (state-processing-time s))
              )
              )
          )
```

B. diagram







**custom.m 와 custom.tst 코드 길이가 너무 길어 첨부합니다.