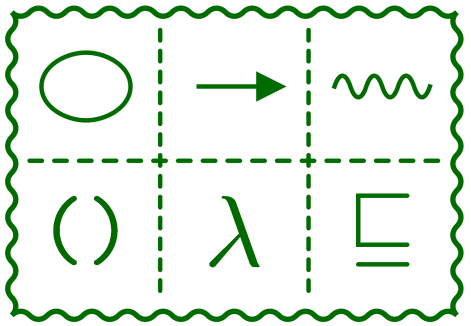
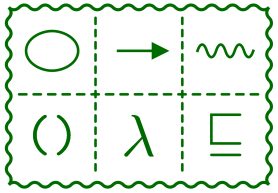


並行システムの検証と実装

第2章 式によるプロセスのモデル化

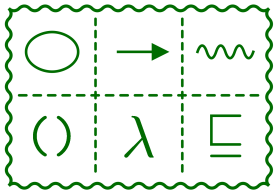
PRINCIPIA Limited
初谷 久史



式によるプロセスのモデル化

- プログラミング言語によるコードと同様に，式（テキスト）によってプロセスを記述できる

	利点	欠点
式	• 編集しやすい • 高度な記述が可能	• 直観的ではない
状態遷移図	• 直観的でわかりやすい	• 編集コストが高い



式によるプロセス記述の例

プロセス定義

選択

チャネル受信とガード

イベント同期

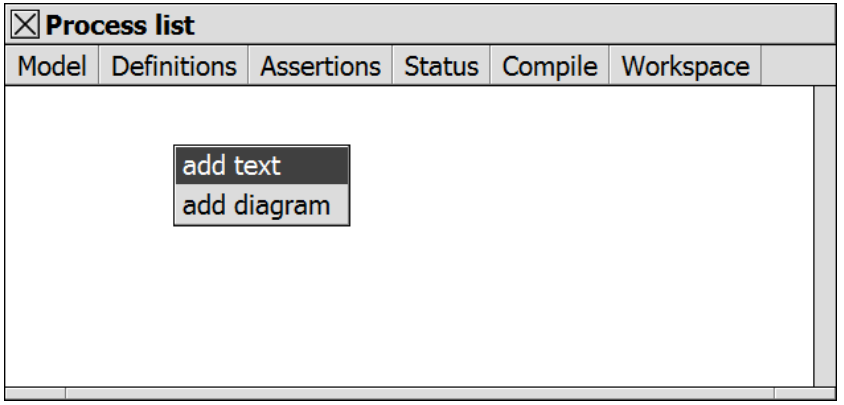
プロセスの再帰呼び出し（ループ）

Process - SEMAPHORE

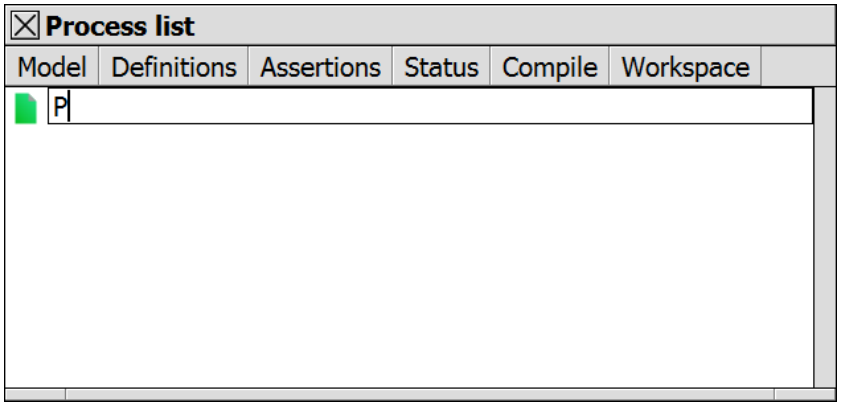
```
(define-process (SEM initial-count)
  (SEMX initial-count '()))

(define-process (SEMX c cs)
  (alt
    (? wait (r) (not (memq r cs))
       (if (= c 0)
           (SEMX 0 (cons r cs))
           (! r (SEMX (- c 1) cs))))
    (? signal (r) (not (memq r cs))
       (! r
          (if (null? cs)
              (if (= c M)
                  STOP
                  (SEMX (+ c 1) cs))
              (xndc q cs
                (! q
                   (SEMX 0 (remq1 q cs)))))))))
```

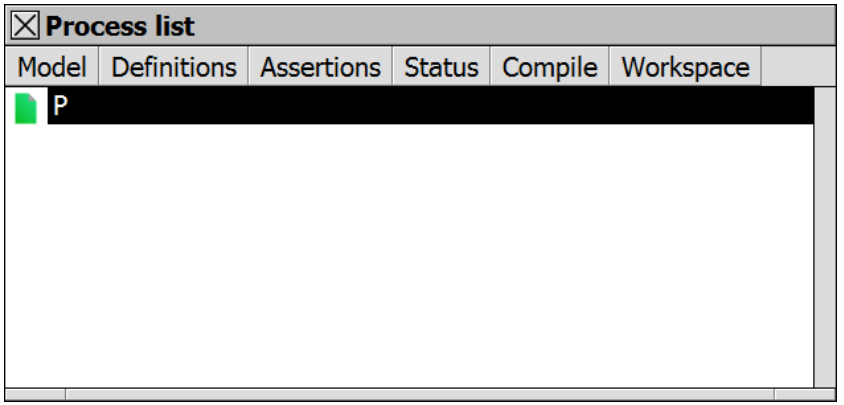
プロセスの作成（テキスト）



クライアントエリアで右クリック
メニューから "add text" を選択

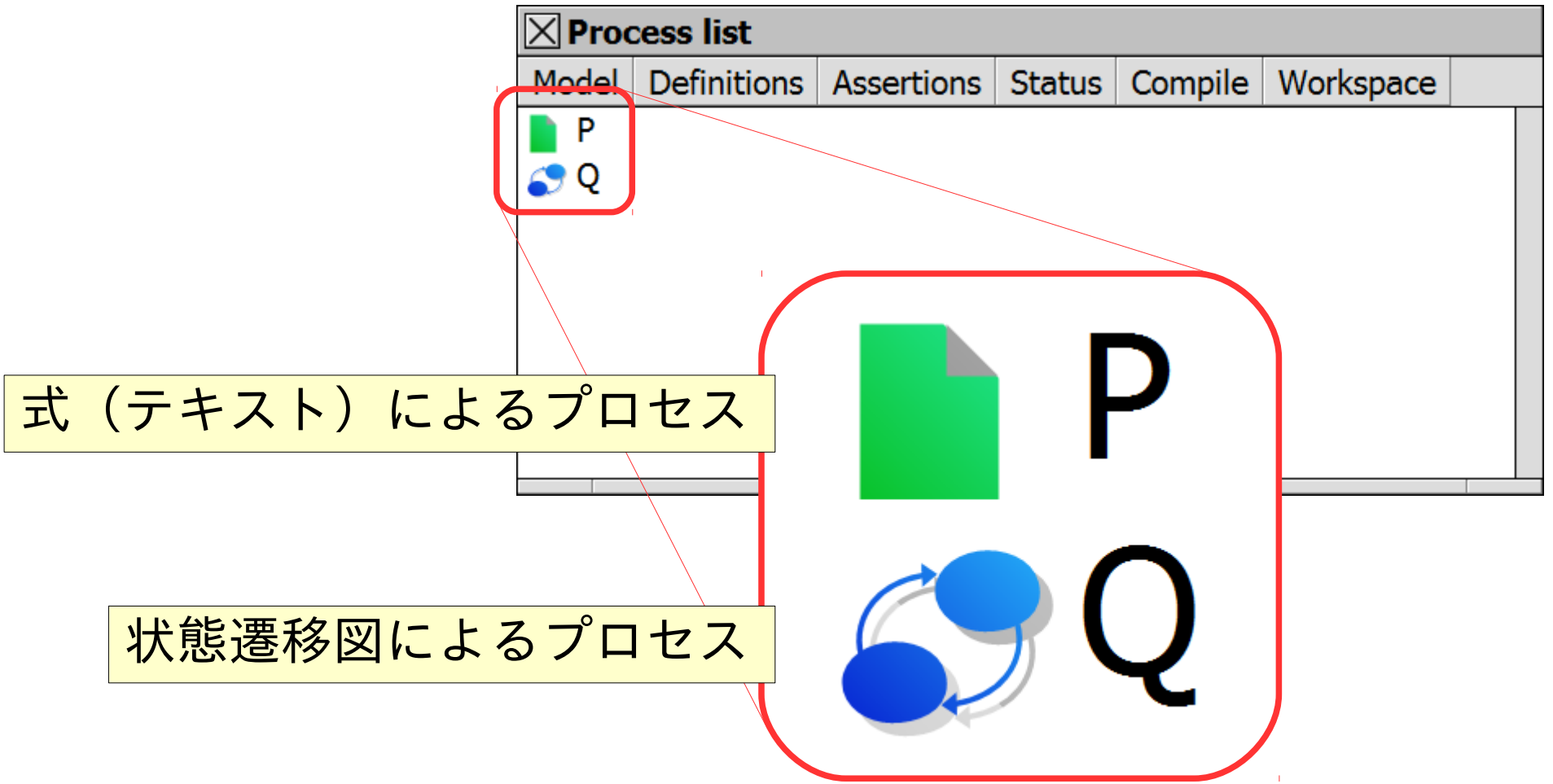


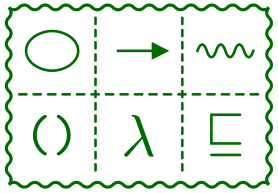
プロセス名を入力して Enter キーで確定



式で記述するプロセスが作成される

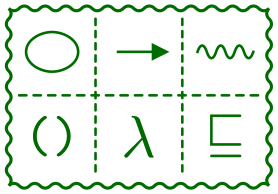
プロセス記述の種類





オペレータの種類

- プロセス定義
- 停止プロセス STOP
- 終了プロセス SKIP
- イベント同期
- チャネル送信
- チャネル受信
- 選択
- プロセス呼び出し
- 再帰
- 場合分け if
- ローカル変数 let



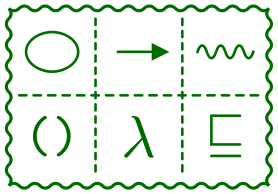
プロセス定義

書式

```
(define-process プロセス名 プロセス式)
```

```
(define-process (プロセス名 パラメータ ...)
    プロセス式)
```

プロセスはパラメータを持つことができる
これをプロセスパラメータと呼ぶ
状態における状態変数に相当する



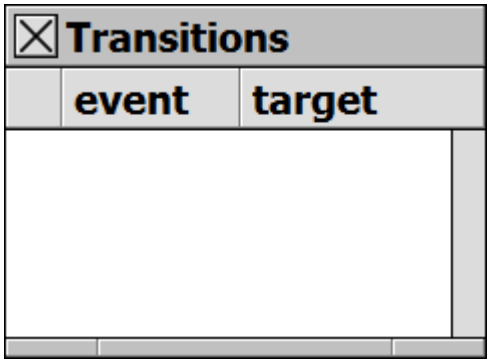
停止プロセス STOP

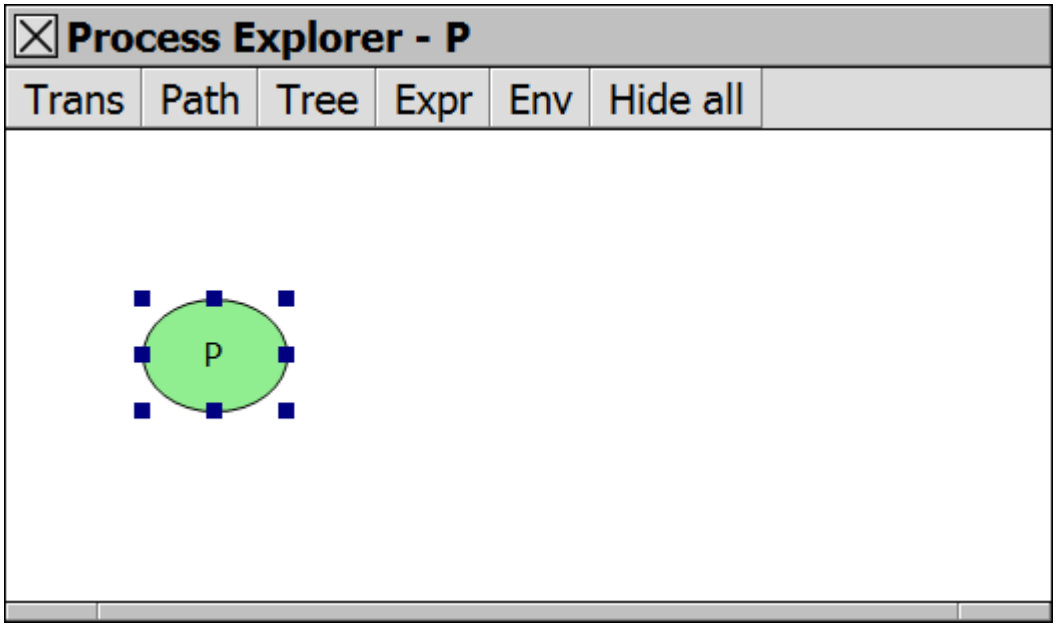
書式

```
STOP
```

例

```
(define-process P STOP)
```





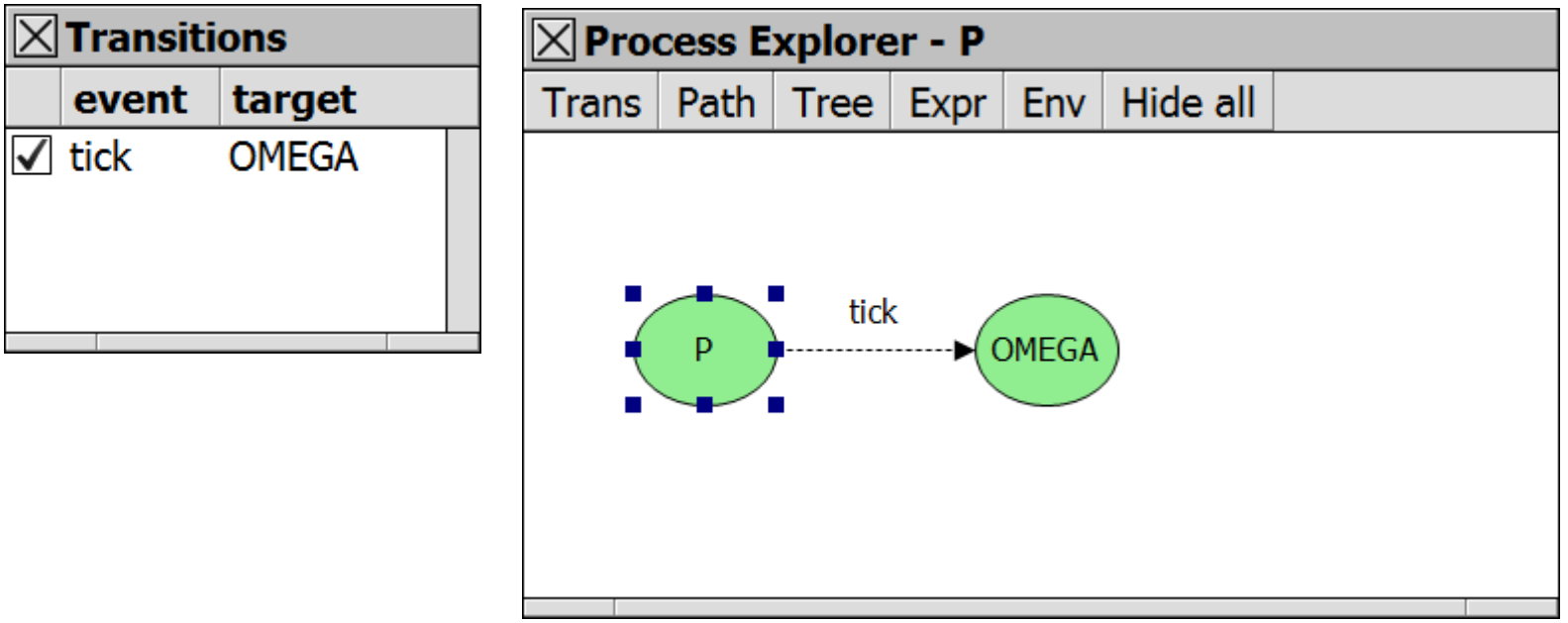
終了プロセス SKIP

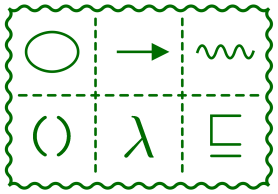
書式

```
SKIP
```

例

```
(define-process P SKIP)
```





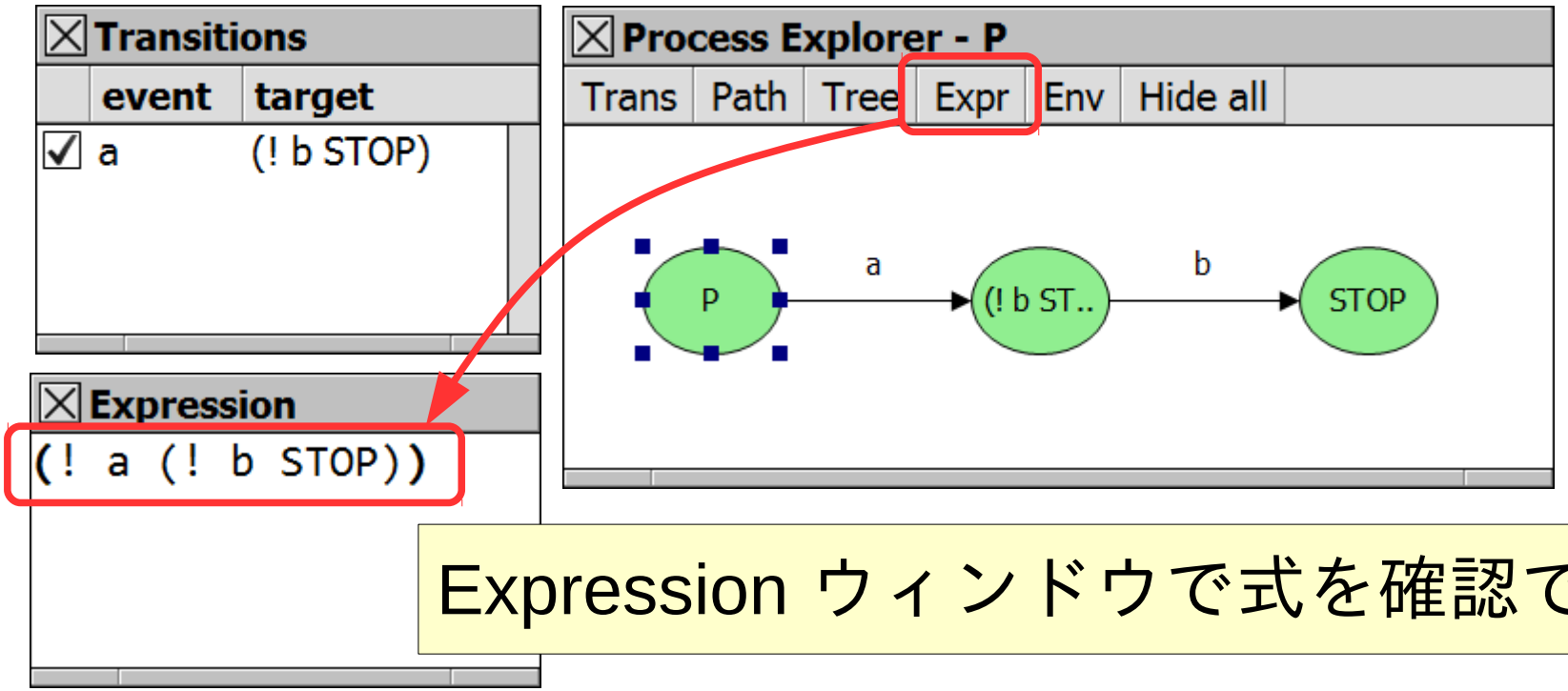
イベント同期

書式

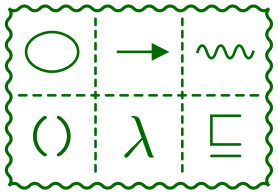
(! イベント プロセス式)

例

```
(define-process P (! a (! b STOP)))
```



Expression ウィンドウで式を確認できる



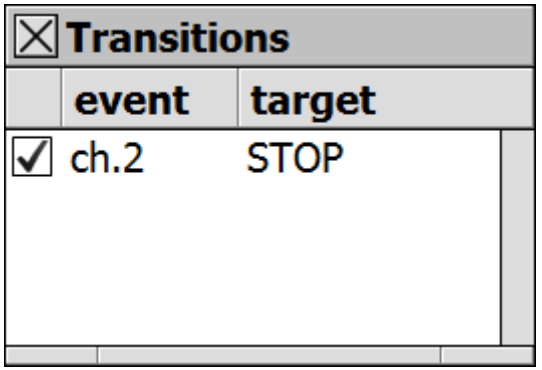
チャネル送信

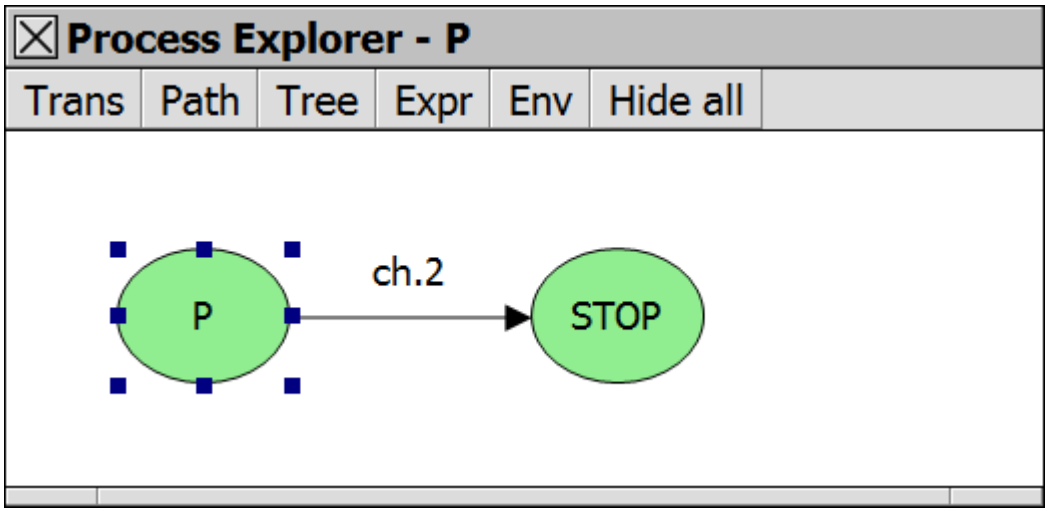
書式

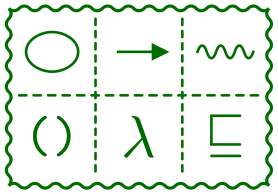
(! チャネル (式$_0$ 式$_1$ ...) プロセス式)

例

```
(define-process P (! ch (2) STOP))
```







チャネル受信

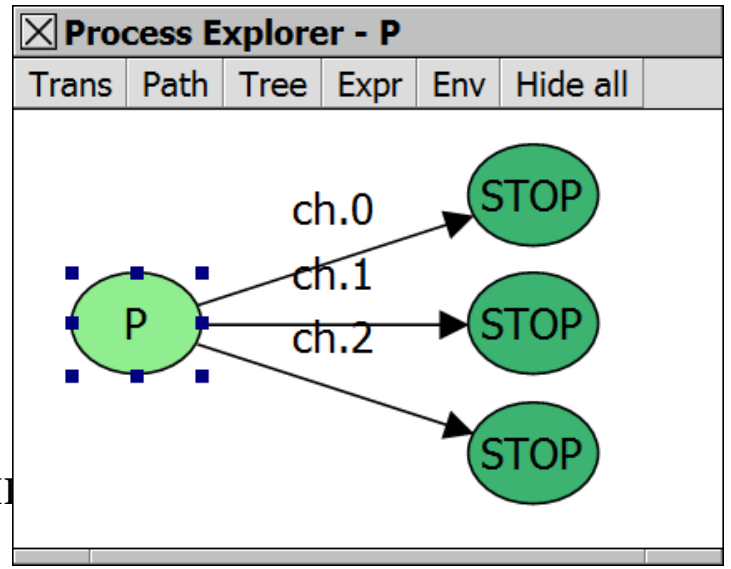
書式

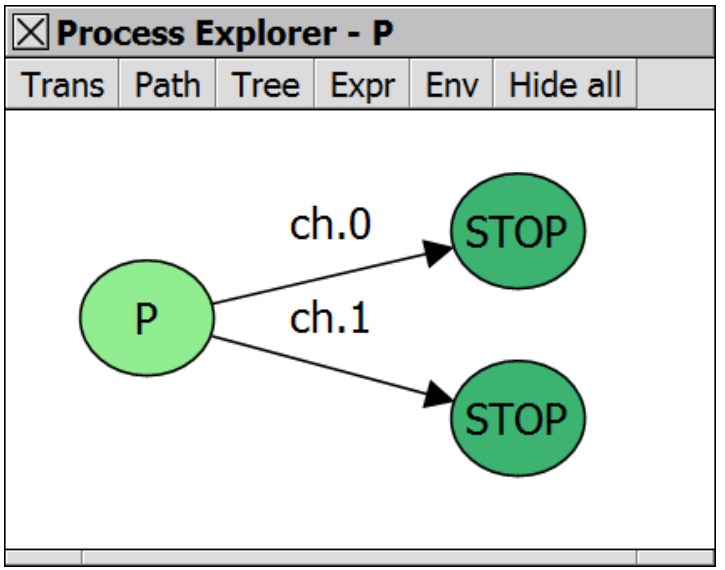
(? チャネル (変数$_0$ 変数$_1$ ...) [ガード]
　プロセス式)

例

```
(define-process P (? ch (x) STOP))
```

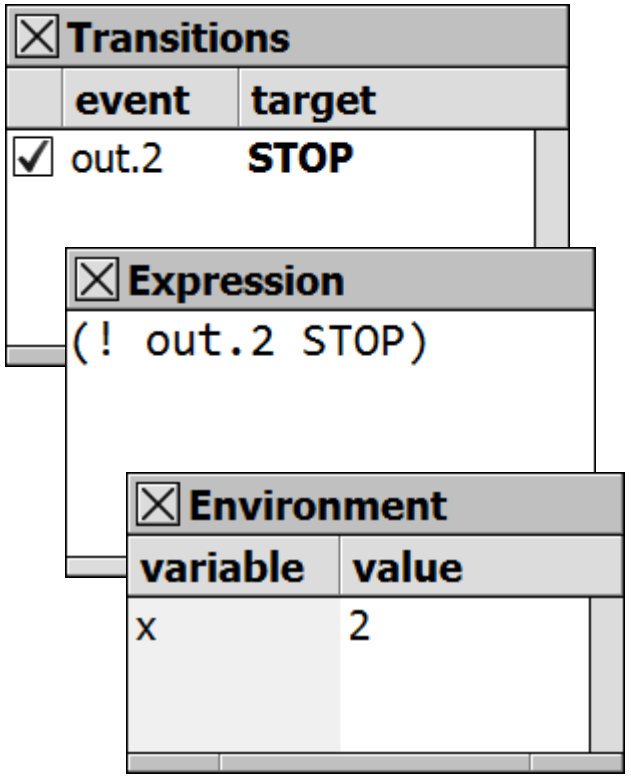
```
(define-process P (? ch (x) (< x 2) STOP))
```

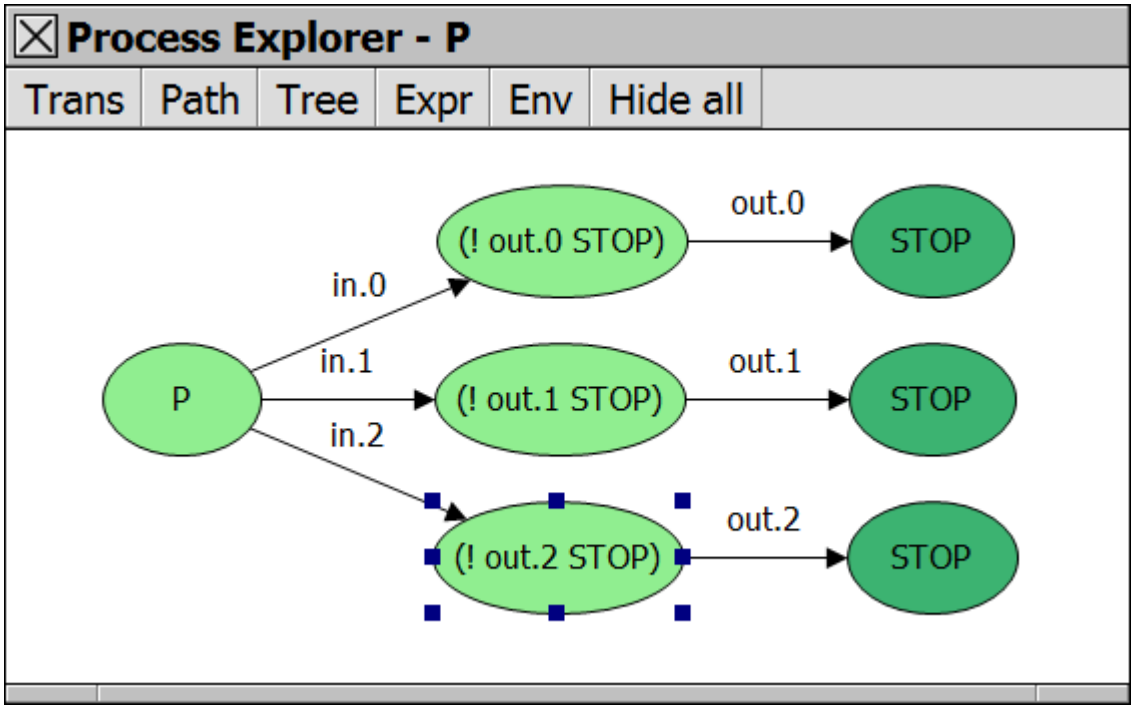




例: チャネル送受信

```
(define-process P
  (? in (x) (! out (x) STOP)))
```





選択 alt

書式

(alt プロセス式$_0$ プロセス式$_1$ ...)

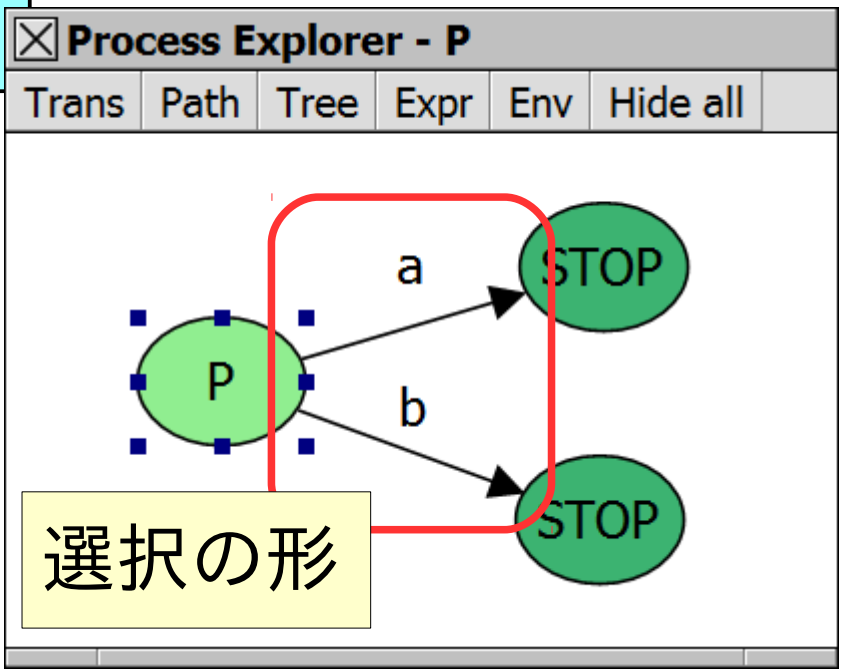
例

```
(define-process P
  (alt
    (! a STOP)
    (! b STOP) ))
```

プロセスが提示する最初のイベントで選択を行う

Transitions

	event	target
☑	a	STOP
☑	b	STOP



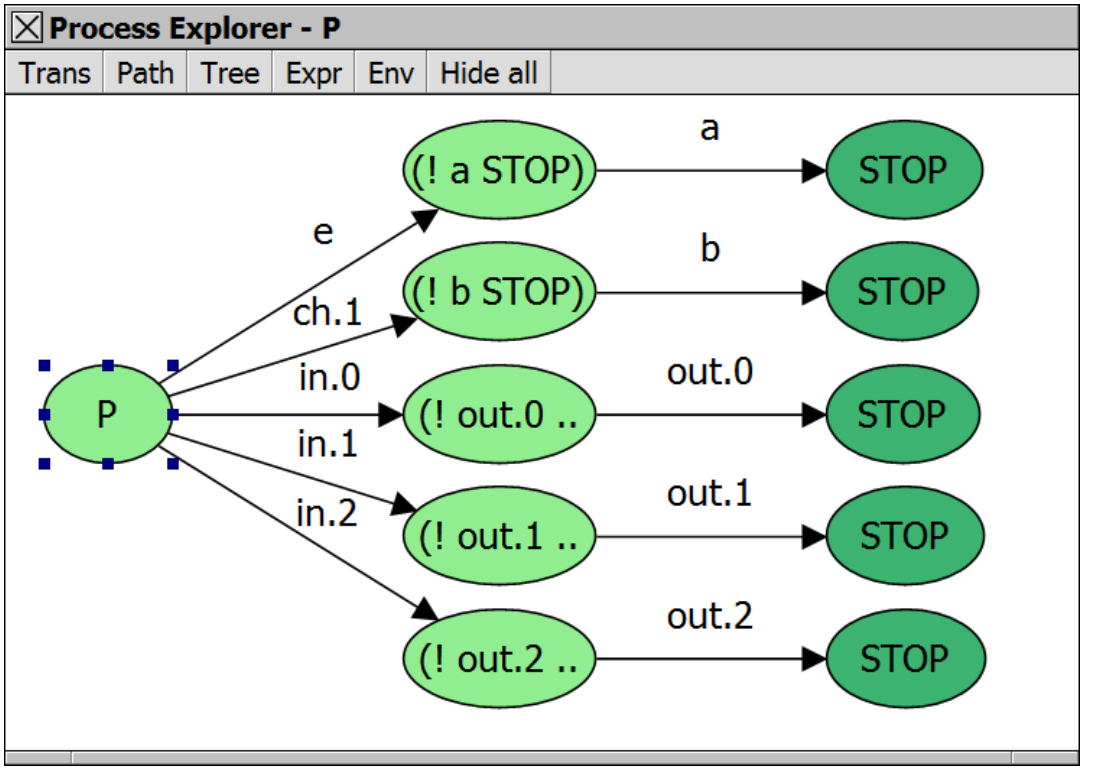
例: 選択における送受信混在

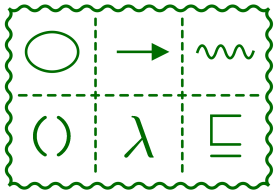
```
(define-process P
  (alt
    (! e      (! a STOP))
    (! ch (1) (! b STOP))
    (? in (x) (! out (x) STOP)) ))
```

どれで同期したかわかるように異なるイベントを出す

Transitions

event	target
e	(! a STOP)
ch.1	(! b STOP)
in.0	(! out.0 STOP)
in.1	(! out.1 STOP)
in.2	(! out.2 STOP)





選択 xalt

書式

```
(xalt 変数 リスト プロセス式)
```

変数の値をリストの各要素としたときのプロセス式で表されるプロセスから選択を行う．リストは評価される

例

```
(xalt k '(0 1 2) (P k))
```

＝

```
(alt (P 0) (P 1) (P 2))
```

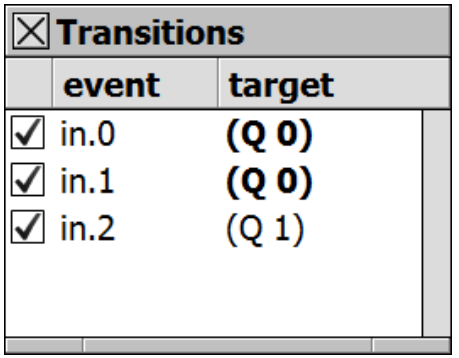
プロセス呼び出し

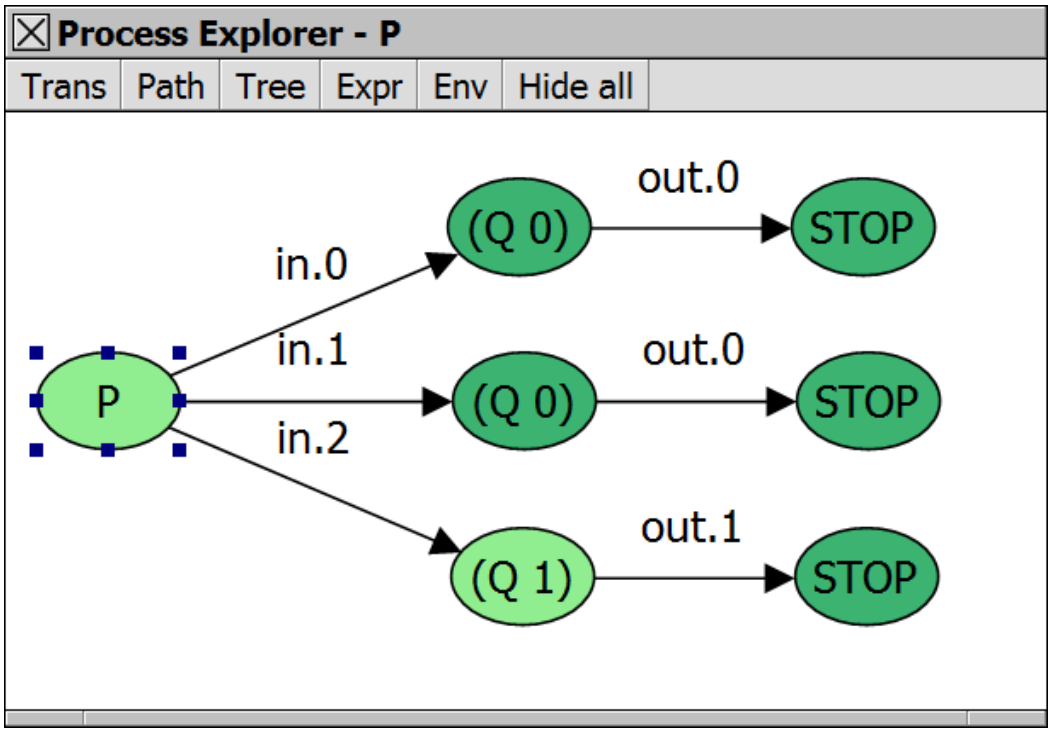
書式

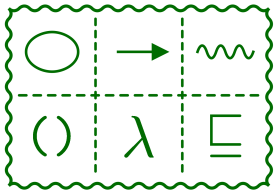
(プロセス名 式$_0$ 式$_1$ ...)

例

```
(define-process P (? in (x) (Q (div x 2))))
(define-process (Q z) (! out (z) STOP))
```





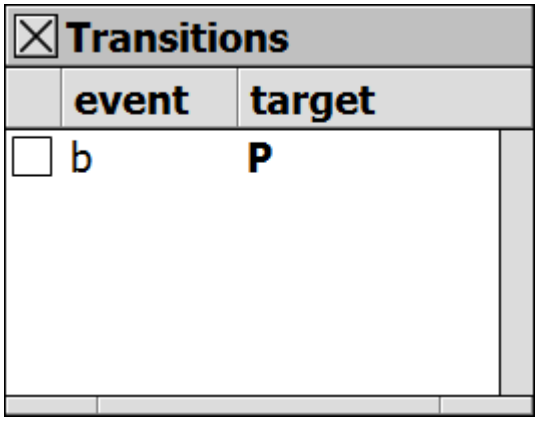


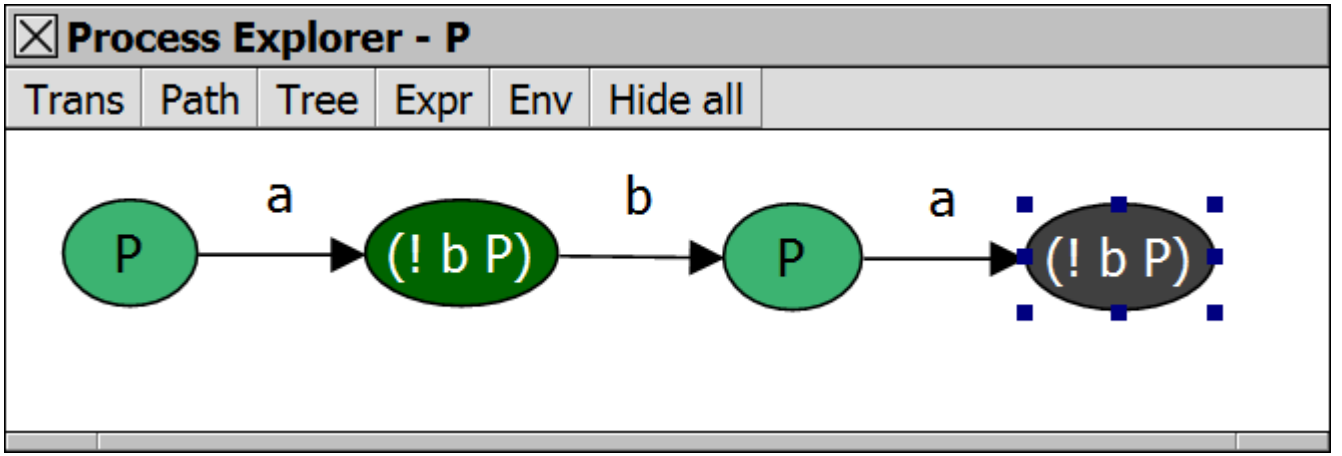
再帰

- 定義しているプロセス自分を呼び出すことでループを記述することができる

例

```
(define-process P (! a (! b P)))
```

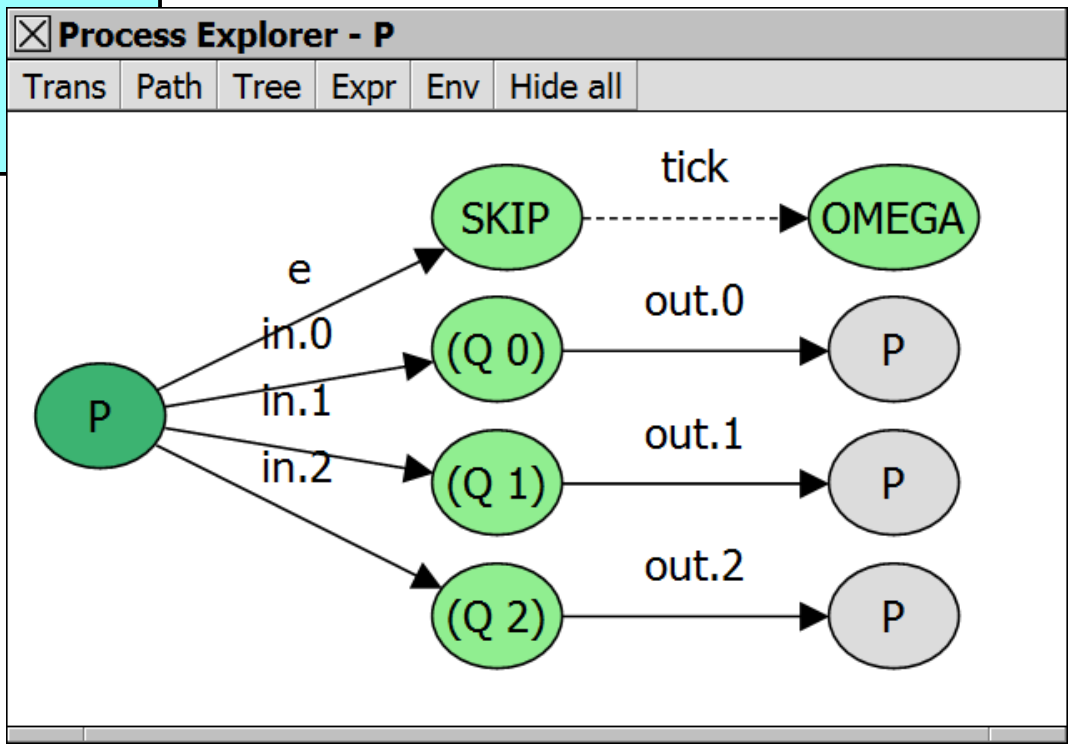


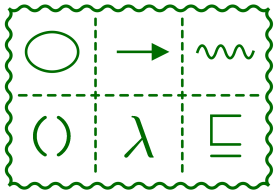


相互再帰

```
(define-process P
  (alt
    (! e SKIP)
    (? in (x) (Q x)) ))

(define-process (Q x)
  (! out (x) P))
```





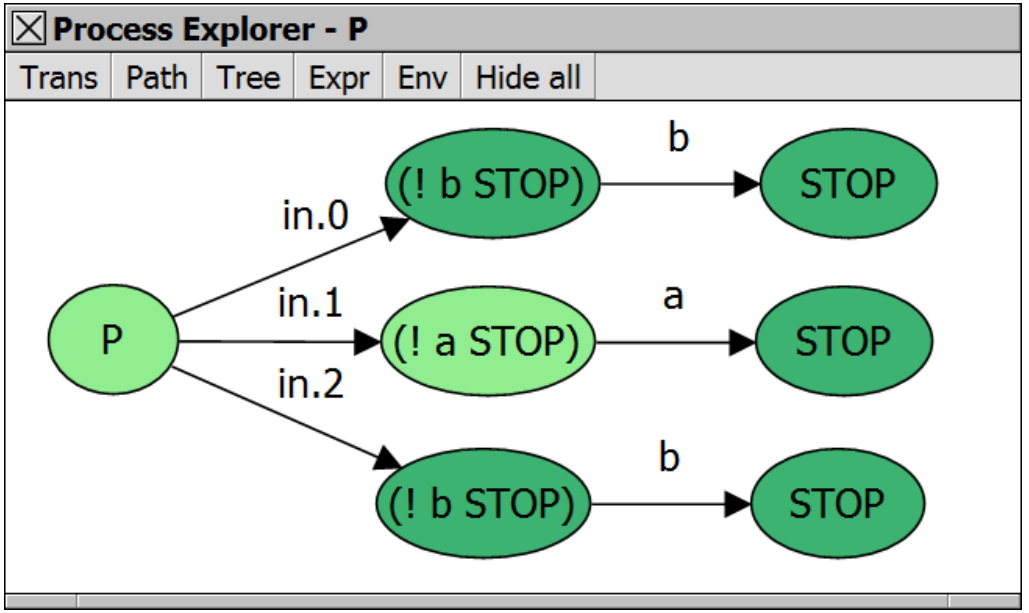
場合分け

書式

(if 条件式 プロセス式$_0$ プロセス式$_1$)

例

```
(define-process P
  (? in (x)
    (if (= x 1)
        (! a STOP)
        (! b STOP) )))
```



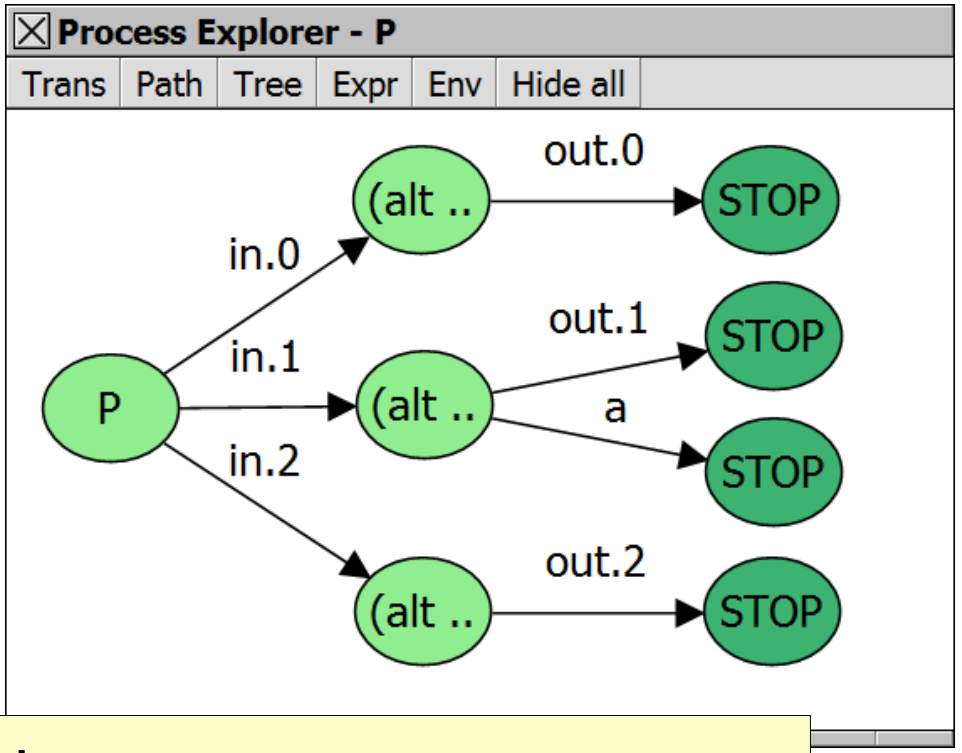
if + STOP によるガード

パターン

(if 条件式 プロセス式 STOP)

例

```
(define-process P
  (? in (x)
    (alt
      (! out (x) STOP)
      (if (= x 1)
          (! a STOP)
          STOP))))
```



ポイント: (alt P STOP) = P

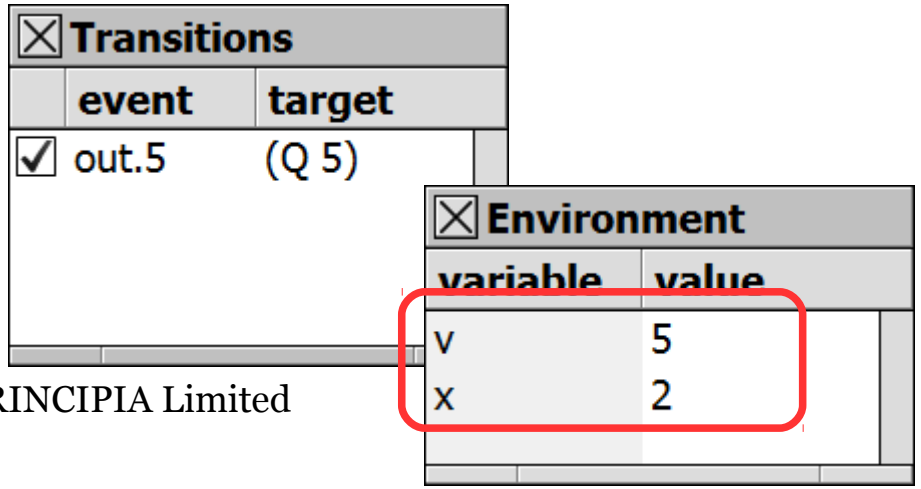
ローカル変数 let

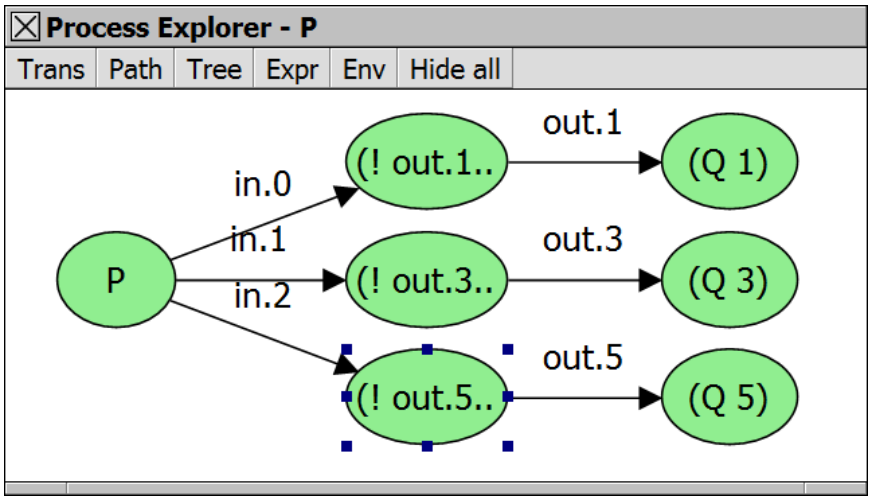
書式

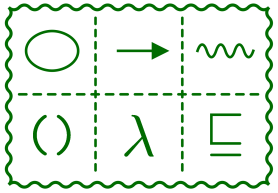
```
(let ((変数0 式0)(変数1 式1) ... )
  プロセス式)
```

例

```
(define-process P
  (? in (x)
    (let ((v (+ (* x 2) 1)))
      (! out (v) (Q v)))))
```

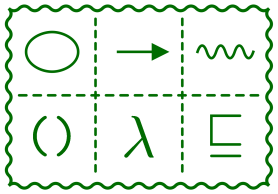






まとめ

プロセス定義	(define-process *pspec pexpr*)
	pspec ::= *name* | (*name param* ...)
停止プロセス	STOP
終了プロセス	SKIP
イベント同期	(! *event pexpr*)
チャネル送信	(! *channel* (*expr* ...) *pexpr*)
チャネル受信	(? *channel* (*param* ...) [*guard*] *pexpr*)
選択	(alt *pexpr* ...)
	(xalt *var lexpr pexpr*)
プロセス呼び出し	(*name expr* ...)
場合分け	(if *bexpr* $pexpr_0$ $pexpr_1$)
ローカル変数	(let ((*var expr*) ...) *pexpr*)



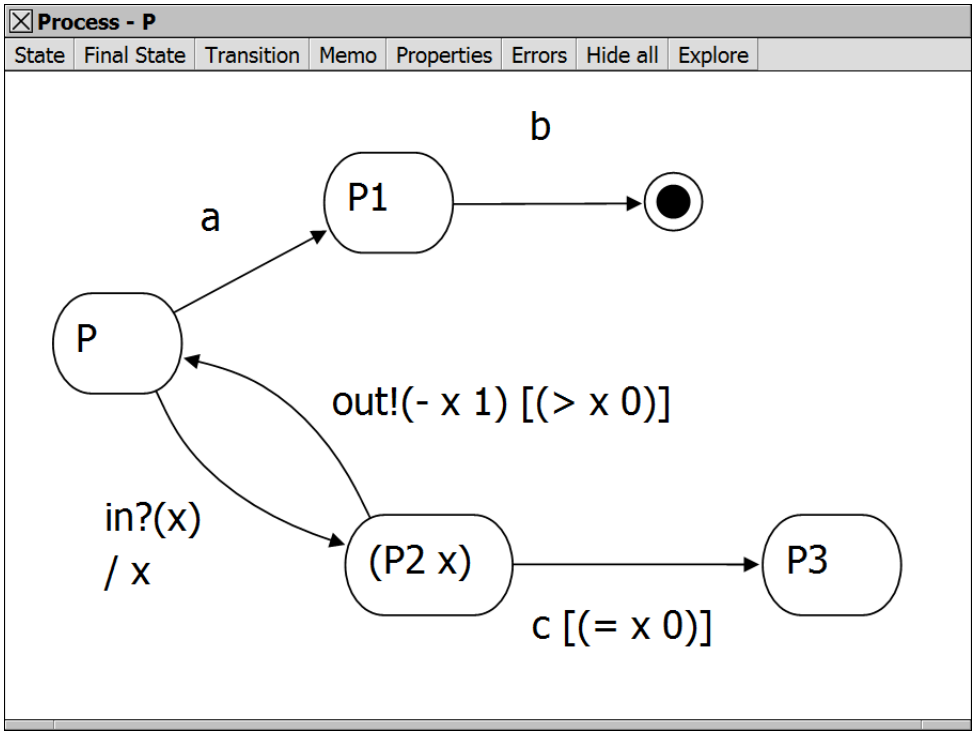
問題

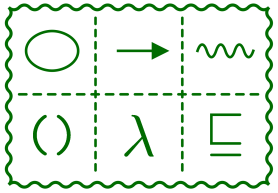
exercise-2-0.ss

- この状態遷移図で表されるプロセスと同じ振る舞いをするプロセス Q を式でモデル化してください
- それぞれの計算木を表示・比較して，同じ振る舞いであることを確認してください

```
(define M 3)
(define I (interval 0 M))
(define D (map list I))

(define-event a)
(define-event b)
(define-event c)
(define-channel in (x) D)
(define-channel out (x) D)
```





解答

```
(define-process Q
  (alt
    (! a (! b SKIP))
    (? in (x)
       (if (= x 0)
           (! c STOP)
           (! out ((- x 1)) Q)))))
```