1 Program Example

Airline ticket agent example from [1]:

```
var seats;
var agent1;
var agent2;
seats=3;
agent1=1;
agent2=1;
par{
   while (agent1==1) do
       if (seats>0) then
           seats=seats-1;
       else
           agent1=0;
       fi;
   od
}{
   while (agent2==1) do
       if (seats>0) then
           seats=seats-1;
       else
           agent2=0;
       fi;
   od
}
remove agent2;
remove agent1;
remove seats;
```

2 Forward/backward stack machine code

2.1 Forward stack machine code

Following stack machine codes are generated by the translator. We show in the form : Program counter | command, operand

```
 1| alloc     0         23| label    64         45| ipush     0
 2| alloc     1         24|  load     0         46|    op     3
 3| alloc     2         25| ipush     1         47|   jpc    49
 4| ipush     3         26|    op     2         48|   jmp    55
 5| store     0         27| store     0         49| label    64
 6| ipush     1         28|   jmp    32         50|  load     0
 7| store     1         29| label    64         51| ipush     1
 8| ipush     1         30| ipush     0         52|    op     2
 9| store     2          31| store     1          53| store     0
10|   par     0          32| label    64          54|   jmp    58
11| label    64          33|   jmp    11          55| label    64
12|  load     1          34| label    64          56| ipush     0
13| ipush     1          35|   par     1          57| store     2
14|    op     4          36|   par     0          58| label    64
15|   jpc    17          37| label    64          59|   jmp    37
16|   jmp    34          38|  load     2          60| label    64
17| label    64          39| ipush     1          61|   par     1
18|  load     0          40|    op     4          62|  free     2
19| ipush     0          41|   jpc    43          63|  free     1
20|    op     3          42|   jmp    60          64|  free     0
21|   jpc    23          43| label    64
22|   jmp    29          44|  load     0
```

2.2 Backward stack machine code

(Program counter, command, operand)

```
 1|   alloc     0         23|   label     0         45|     nop     0
 2|   alloc     1         24|   label     0         46|     nop     0
 3|   alloc     2         25|     nop     0         47|     nop     0
 4|     par     0         26|     nop     0         48|    rjmp     0
 5|    rjmp     0         27|     nop     0         49|   label     0
 6|   label     0         28|    rjmp     0         50|   label     0
 7|    rjmp     0         29|     par     1         51|     nop     0
 8| restore     2         30|     par     0         52|     nop     0
 9|     nop     0         31|    rjmp     0         53|     nop     0
10|    rjmp     0         32|   label     0         54|     nop     0
11|   label     0         33|    rjmp     0         55|     par     1
12| restore     0         34| restore     1         56| restore     2
13|     nop     0         35|     nop     0         57|     nop     0
14|     nop     0         36|    rjmp     0         58| restore     1
15|     nop     0         37|   label     0         59|     nop     0
16|    rjmp     0         38| restore     0         60| restore     0
17|   label     0         39|     nop     0         61|     nop     0
18|   label     0         40|     nop     0         62|    free     2
19|     nop     0         41|     nop     0         63|    free     1
20|     nop     0         42|    rjmp     0         64|    free     0
21|     nop     0         43|   label     0
22|    rjmp     0         44|   label     0
```

3 Step-mode execution

forward execution mode selection

```
mode   1:auto 2:select >> 2
```

execute a sequencial process

```
~~~~~~~~Process0 execute~~~~~~~~
pc = 1   command = alloc   operand = 0
executing stack:       [0]
shared variable stack: [0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 2   command = alloc   operand = 1
executing stack:       [0, 0]
shared variable stack: [0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 3   command = alloc   operand = 2
executing stack:       [0, 0, 0]
shared variable stack: [0, 0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 4   command = ipush   operand = 3
executing stack:       [0, 0, 0, 3]
shared variable stack: [0, 0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 5   command = store   operand = 0
executing stack:       [3, 0, 0]
shared variable stack: [3, 0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 6   command = ipush   operand = 1
executing stack:       [3, 0, 0, 1]
shared variable stack: [3, 0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 7   command = store   operand = 1
executing stack:       [3, 1, 0]
shared variable stack: [3, 1, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 8   command = ipush   operand = 1
executing stack:       [3, 1, 0, 1]
shared variable stack: [3, 1, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 9   command = store   operand = 2
executing stack:       [3, 1, 1]
shared variable stack: [3, 1, 1]
```

select a process and execute parallel processes

```
>> 1
~~~~~~~~Process1 execute~~~~~~~~
pc = 10   command =   par   operand = 0
executing stack:       [3, 1, 1]
```

```
shared variable stack: [3, 1, 1]
>> 2
~~~~~~~~Process2 execute~~~~~~~~
pc = 36   command =   par   operand = 0
executing stack:       [3, 1, 1]
shared variable stack: [3, 1, 1]
>> 1
~~~~~~~~Process1 execute~~~~~~~~
pc = 11   command = label   operand = 64
executing stack:       [3, 1, 1]
shared variable stack: [3, 1, 1]
>> 1
~~~~~~~~Process1 execute~~~~~~~~
pc = 12   command =  load   operand = 1
executing stack:       [3, 1, 1, 1]
shared variable stack: [3, 1, 1]
```

end of all parallel processes

```
>> esc
```

execute sequencial process

```
~~~~~~~~Process0 execute~~~~~~~~
pc = 62   command =   free   operand = 2
executing stack:       [0, 0]
shared variable stack: [0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 63   command =   free   operand = 1
executing stack:       [0]
shared variable stack: [0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 64   command =   free   operand = 0
executing stack:       []
shared variable stack: []
```

backward execution mode selection

```
 mode   1:auto 2:select >> 2
```

execute a sequencial process

```
pc = 1   command =   alloc   operand = 0
shared variable stack: [0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 2   command =   alloc   operand = 1
shared variable stack: [0, 0]
~~~~~~~~Process0 execute~~~~~~~~
```

```
pc = 3   command =   alloc   operand = 2
shared variable stack: [0, 0, 0]
~~~~~~~~Process0 execute~~~~~~~~
```

execute parallel processes step-by-step(input is only enter key)

```
process 1
~~~~~~~~Process1 execute~~~~~~~~
pc = 30   command =     par   operand = 1
shared variable stack: [0, 0, 0]
process 1
~~~~~~~~Process1 execute~~~~~~~~
pc = 31   command =    rjmp   operand = 0
shared variable stack: [0, 0, 0]
process 1
~~~~~~~~Process1 execute~~~~~~~~
pc = 49   command =   label   operand = 0
shared variable stack: [0, 0, 0]
```

end of all parallel processes(input esc)

```
esc
```

execute a sequencial process

```
~~~~~~~~Process0 execute~~~~~~~~
pc = 56   command = restore   operand = 2
shared variable stack: [3, 1, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 57   command =     nop   operand = 0
shared variable stack: [3, 1, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 58   command = restore   operand = 1
shared variable stack: [3, 0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 59   command =     nop   operand = 0
shared variable stack: [3, 0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 60   command = restore   operand = 0
shared variable stack: [0, 0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 61   command =     nop   operand = 0
shared variable stack: [0, 0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 62   command =    free   operand = 2
shared variable stack: [0, 0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 63   command =    free   operand = 1
```

```
shared variable stack: [0]
~~~~~~~~Process0 execute~~~~~~~~
pc = 64   command =    free   operand = 0
shared variable stack: []
```

References

[1] Hoey, J., Ulidowski, I.: Reversible imperative parallel programs and debugging. In: RC 2019. Lecture Notes in Computer Science, vol. 11497, pp. 108–127 (2019)