Some Examples

1. Hello World!

```
init {
        printf("Hello World!\n")
}
```

2. Hello World: Multiple Processes

```
active proctype Hello() {
printf("Hello process, my pid is: %d\n", _pid);
}

init {
int lastpid;
printf("init process, my pid is: %d\n", _pid);
lastpid = run Hello();
printf("last pid was: %d\n", lastpid);
}
```

3. Peterson Algorithm

It is a concurrent programming algorithm for mutual exclusion that allows two or more processes to share a single-use resource without conflict, using only shared memory for communication. It was formulated by Gary L. Peterson in 1981. While Peterson's original formulation worked with only two processes, the algorithm can be generalized for more than two.

The algorithm uses two variables, flag and turn. A flag[n] value of true indicates that the process n wants to enter the critical section. Entrance to the critical section is granted for process P0 if P1 does not want to enter its critical section or vice versa.

```
bool flag[2] = {false, false};
int turn;
```

```
P0:       flag[0] = true;
P0_gate: turn = 1;
          while (flag[1] == true &&
turn == 1)
          {
              // busy wait
          }
          // critical section
          ...
          // end of critical section
          flag[0] = false;
```

```
P1:       flag[1] = true;
P1_gate: turn = 0;
          while (flag[0] == true &&
turn == 0)
          {
              // busy wait
          }
          // critical section
          ...
          // end of critical section
          flag[1] = false;
```

The algorithm satisfies the three essential criteria: mutual exclusion, progress, and bounded waiting. In spin:

```
bool turn, flag[2];
byte ncrit;

active [2] proctype user()
{
        assert(_pid == 0 || _pid == 1);
again:
        flag[_pid] = 1;
        turn = 1 - _pid;
        (flag[1 - _pid] == 0 || turn == _pid);

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

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

4. Producer-Consumer Problem

```
mtype = { P,C }; /* symbols used */
mtype turn = P; /* shared variable */

active proctype producer(){
do
:: (turn == P) ->
/* Guard */
printf("Produce\n");
turn = C
od
}

active proctype consumer(){
again:
if
:: (turn == C) ->
/* Guard */
printf("Consume\n");
turn = P;
goto again
fi
}
```

5. Sender and Receiver Process

5.1: Message with one data item

```
chan x = [3] of { int };
int turn = 0;
```

```
active proctype sender()
{
        x!3; x!2; x!1;
 turn = 1;
        printf("sender process\n");
}

active proctype receiver()
{
        if
        :: (turn == 1) -> int var1, var2, var3;
                        x?var1; x?var2; x?var3;
 printf("Receiver Process: %d %d %d\n",var1,var2,var3); fi
}
```

5.2: Message with multiple data items

```
chan x = [1] of { int , int , int };
int turn = 0;

active proctype sender()
{
        x!3,2,1;
 turn = 1;
        printf("sender process\n");
}

active proctype receiver()
{
        if
        :: (turn == 1) -> int var1, var2, var3;
                        x?var1,var2,var3;
 printf("receiver process: %d %d %d\n",var1,var2,var3); fi
}
```

5.2: Other channel operations

```
chan glob = [1] of { chan };

active proctype A(){
chan loc = [1] of { int }
int var;
glob!loc;
loc?var;
printf("Loc = %d\n",var);
}

active proctype B(){
chan who;
```

```
glob?who;
who!1000;
}
```

6. Alternating Bit Protocol

```
mtype = { msg, ack };

chan to_sndr = [2] of { mtype, bit };
chan to_rcvr = [2] of { mtype, bit };

active proctype Sender ()
{ bit seq_out=1, seq_in;

        do
        :: to_rcvr!msg (seq_out) ->
                to_sndr?ack (seq_in);
                if
                :: seq_in == seq_out ->
                        printf("ack received\n"); seq_out = !seq_out
                :: else ->
                        printf("ack not received\n"); skip
                fi
        od
}

active proctype Receiver ()
{ bit seq_in;

        do
        :: to_rcvr?msg (seq_in) ->
                printf("msg received\n"); to_sndr!ack (seq_in)
        :: timeout ->
                printf("time out\n"); to_sndr!ack (seq_in)
        od
}
```