```vhdl
-- ROBOT.VHD --

library ieee;
use ieee.std_logic_1164.all;

entity Robot is
    port(reset, clk, athome, findfood, lostfood, closetofood,
    success, aboverestth, abovesearchth, scantimeup: in std_logic;
    rest, search, food: out std_logic);
end Robot;

architecture automate_robot of Robot is

    type States is (IDLE, RESTING, RANDOMWALK, SCANAREA, HOMING, MOVETOFOOD,
    MOVETOHOME, DEPOSIT, GRABFOOD) ;
    Signal state, nextstate : States;
    -- psl default clock is rising_edge(clk);

    -- psl property p1 is always (search = '1' ->  (findfood = '1') before! (state =
    GRABFOOD) );
    -- psl assert p1;

    -- psl property p2 is always (search = '1' ->  (abovesearchth = '1') before!
    (state = HOMING) );
    -- psl assert p2;

    -- psl property p3 is always (state = MOVETOHOME -> state=DEPOSIT before! rest =
    '1');
    -- psl assert p3;

    -- psl property p4 is

    --  always { state = RANDOMWALK and abovesearchth = '0';
    -- (abovesearchth = '0' and findfood = '0' and not(state = IDLE) )[*];
    -- (abovesearchth ='0' and findfood = '1' and not(state = IDLE)) ;
    -- (abovesearchth = '0' and lostfood = '0' and closetofood = '0' and not(state =
    IDLE))[*];
    -- (abovesearchth = '0' and lostfood = '1' and not(state = IDLE)) ;
    -- (abovesearchth = '0' and findfood = '0' and scantimeup = '0' and not(state =
    IDLE))[*];
    -- (abovesearchth = '0' and findfood = '0' and scantimeup = '1' and not(state =
    IDLE)) } |=> {state = RANDOMWALK} ;
    -- psl assert p4;


    -- psl property p5 is always ( {state = RESTING } |=> {[*] ; state = RANDOMWALK
    });
    -- psl assert p5;


begin

    -- Calcul de l'état suivant
    -- Comme on est en std_logic,"elsif ='0'" et non "else", car le signal peux
    avoir d'autre valeur
    process (state, athome, findfood, lostfood, closetofood, success, aboverestth,
    abovesearchth, scantimeup)
    begin
        case state is
            when IDLE => nextstate <= RESTING;
            when RESTING =>
                if aboverestth = '1' then nextstate <= RANDOMWALK;
                else--elsif aboverestth = '0' then
                nextstate <= RESTING;
                end if;

            when RANDOMWALK =>
                if abovesearchth = '1' then nextstate <= HOMING;
                else--  abovesearchth = '0' then
                    if findfood = '1' then nextstate <= MOVETOFOOD;
                    else--elsif findfood = '0' then
                        nextstate <= RANDOMWALK;
                    end if;
```

```
64                  end if;
65
66              when SCANAREA =>
67                  if abovesearchth = '1' then nextstate <= HOMING;
68                  else--elsif  abovesearchth = '0' then
69                      if findfood = '1' then nextstate <= MOVETOFOOD;
70                      else--elsif findfood = '0' then
71                          if scantimeup = '1' then nextstate <= RANDOMWALK;
72                          else--elsif scantimeup = '0' then
73                          nextstate <= SCANAREA;
74                          end if;
75                      end if;
76                  end if;
77              when HOMING => if(athome = '1') then nextstate <= RESTING; else
                nextstate <= HOMING; end if;
78              when MOVETOFOOD =>
79                  if abovesearchth = '1' then nextstate <= HOMING;
80                  else--elsif  abovesearchth = '0' then
81                      if lostfood = '1' then nextstate <= SCANAREA;
82                      else--elsif lostfood = '0' then
83                          if closetofood = '1' then nextstate <= GRABFOOD;
84                          else--elsif closetofood = '0' then
85                              nextstate <= MOVETOFOOD;
86                          end if;
87                      end if;
88                  end if;
89              when GRABFOOD =>
90                  if success = '1' then nextstate <= MOVETOHOME;
91                  else--elsif success = '0' then
92                   nextstate <= GRABFOOD;
93                  end if;
94              when MOVETOHOME =>
95                  if athome = '1' then nextstate <= DEPOSIT;
96                  else--elsif athome = '0' then
97                   nextstate <= MOVETOHOME;
98                  end if;
99              when DEPOSIT =>
100                 if success = '1' then nextstate <= RESTING;
101                 else--elsif success = '0' then
102                  nextstate <= DEPOSIT;
103                 end if;
104         end case;
105     end process;
106
107     -- MISE A JOUR DU REGISTRE D'ETAT
108
109     process(reset, clk)
110     begin
111         -- RESET : asynchrone haut
112         if reset = '1' then state <= IDLE;
113         -- HORLOGE : front montant
114         elsif (clk'event and clk = '1') then
115             state <= nextstate;
116         end if;
117     end process;
118
119
120     -- MISE A JOUR DES OUTPUTS
121     rest <= '1' when (( state = DEPOSIT and success = '1' ) OR (state = IDLE) OR
        (state = HOMING and athome = '1') ) else '0';
122     search <= '1' when (state = RESTING and aboverestth = '1' ) else '0';
123     food <= '1' when  (state = MOVETOFOOD and abovesearchth = '0' and lostfood = '0'
        and closetofood ='1') else '0';
124
125
126
127 end automate_robot;
128
129
```