Pebble Game

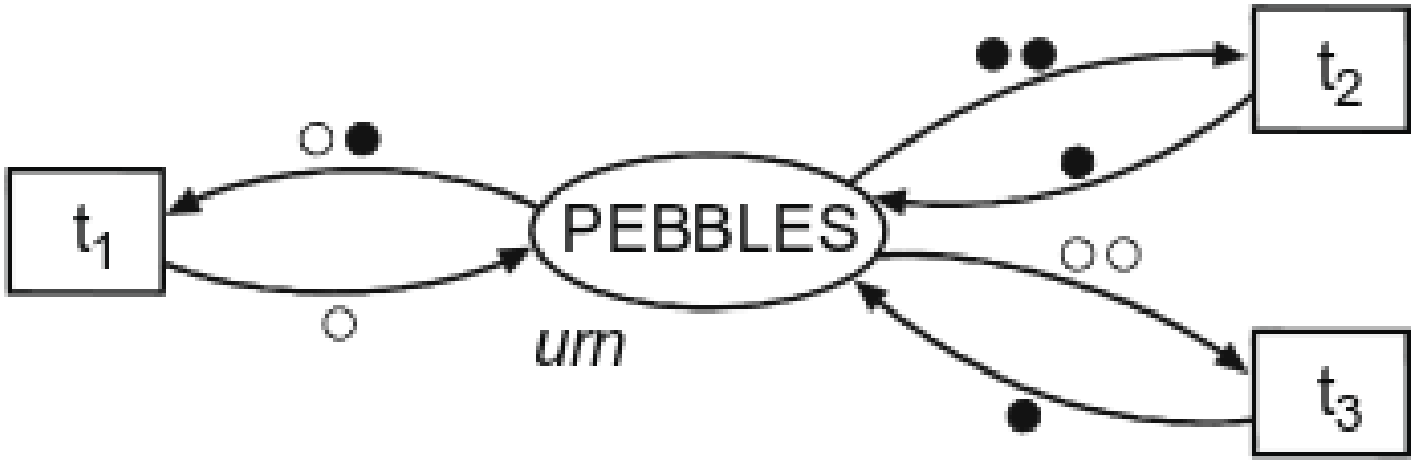

Fig. 2. The basic version of the algorithm

Figure 1 represents the algorithm as a nondeterministic guarded command program. B and W are the number of white and black pebbles in the initial state.

$$
\begin{array}{l}
b := B;\ w := W;\\
\\
\underline{\text{do}}\ w \geq 1 \wedge b \geq 1 \rightarrow b := b - 1\\
\square\ \ b \geq 2 \rightarrow\\
\qquad\qquad b := b - 1\\
\square\ \ w \geq 2 \rightarrow\\
\qquad\qquad w := w - 2;\ b := b + 1\\
\underline{\text{od}}
\end{array}
$$

Fig. 1. Dijkstra's solution to the pebble game

pebble0.pml

```
active proctype P() {
    /* initial numbers of black and white pebbles */
    unsigned B : 31 = 34, W : 31 = 36
    /* actual  numbers of black and white pebbles */
    unsigned b : 31 = B,  w : 31 = W

    do
    :: w >= 1 && b >= 1 -> b--
    :: b >= 2           -> b--
    :: w >= 2           -> w=w-2; b++
    :: else             -> break
    od

    if
    :: b == 0 -> printf("The last pebble is white\n")
    :: else   -> printf("The last pebble is black\n")
    fi
}
```

```
$ spin pebble0.pml
      The last pebble is black
1 process created
```

pebble1.pml

Cambiando el valor inicial de **W** (**36 -> 37**):

```
$ spin pebble1.pml
      The last pebble is white
1 process created
```

¿Cuáles son los datos de entrada?

¿Cuál es la precondición?

¿Qué resultado produce el programa?

¿Cuál es la postcondición?

Los datos de entrada son las variables iniciales **B** y **W**.

La precondición: **B ≥ 0 && W ≥ 0 && B+W > 0**

El resultado está en las variables **b** y **w**.

La postcondición: **b+w = 1**

Si la postcondición se cumple, el programa es *parcialmente* correcto.

El programa será *completamente* correcto si termina.

¿Cómo demostrar que el programa termina?

Se necesita una función de cota (*bound*), en naturales, tal que decrezca estrictamente en cada iteración del bucle.

En la 1ra rama decrece siempre **`b`**.
En la 2da rama también.
Pero en la 3ra rama **`b`** crece en 1, pero **`w`** decrece en 2. Entonces, **`b+w`** decrece en 1.

La función de cota es: **`b+w`**

pebble2.pml

```
active proctype P() {
    unsigned B : 31 = 34, W : 31 = 36
    unsigned b : 31 = B,  w : 31 = W, t : 31 = B+W

    assert(B >= 0 && W >= 0 && B+W > 0)

loop:
    t = b + w
    if
    :: w >= 1 && b >= 1 -> b--; goto loop_fin
    :: b >= 2           -> b--; goto loop_fin
    :: w >= 2           -> w=w-2; b++; goto loop_fin
    :: else             -> goto fin
    fi
loop_fin:
    assert(b+w < t)
    goto loop
fin:
    if
    :: b == 0 -> printf("The last pebble is white\n")
    :: else   -> printf("The last pebble is black\n")
    fi

    assert(b+w == 1)
}
```

¿Cuál es el invariante del bucle?

Al inicio de cada iteración:

La paridad de **`b`** se cambia en cada iteración: de par a impar, de impar a par.

La paridad de **`w`** se mantiene: si es par inicialmente, será par en cada iteración; si es impar inicialmente, se quedará impar en cada iteración.

pebble3.pml

```
active proctype P() {
    unsigned B : 31 = 34, W : 31 = 36
    unsigned b : 31 = B,  w : 31 = W, t : 31 = B+W
    bit W_oddness, B_oddness

    W_oddness = W & 1
    assert(B >= 0 && W >= 0 && B+W > 0)
loop:
    t = b+w
    assert(w & 1 == W_oddness)
    B_oddness = b & 1
    if
    :: w >= 1 && b >= 1 -> b--; goto loop_fin
    :: b >= 2           -> b--; goto loop_fin
    :: w >= 2           -> w=w-2; b++; goto loop_fin
    :: else             -> goto fin
    fi
loop_fin:
    assert(b+w < t)
    assert(b & 1 != B_oddness)
    goto loop
fin:
    if
    :: b == 0 -> printf("The last pebble is white\n")
    :: else   -> printf("The last pebble is black\n")
    fi

    assert(b+w == 1)
}
```

```
$ spin -a pebble3.pml
$ gcc -o pan pan.c
$ ./pan
hint: this search is more efficient if pan.c is compiled -DSAFETY
pan:1: assertion violated (w&(1==W_oddness)) (at depth 2)
pan: wrote pebble3.pml.trail

(Spin Version 6.4.3 -- 16 December 2014)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 36 byte, depth reached 2, errors: 1
        3 states, stored
        0 states, matched
        3 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.000       equivalent memory usage for states (stored*(State-vector + overhead))
    0.290       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage

pan: elapsed time 0.01 seconds
```

pebble3.pml

```
active proctype P() {
    unsigned B : 31 = 34, W : 31 = 36
    unsigned b : 31 = B,  w : 31 = W, t : 31 = B+W
    bit W_oddness, B_oddness

    W_oddness = W & 1
    assert(B >= 0 && W >= 0 && B+W > 0)
loop:
    t = b+w
    assert((w & 1) == W_oddness)
    B_oddness = b & 1
    if
    :: w >= 1 && b >= 1 -> b--; goto loop_fin
    :: b >= 2           -> b--; goto loop_fin
    :: w >= 2           -> w=w-2; b++; goto loop_fin
    :: else             -> goto fin
    fi
loop_fin:
    assert(b+w < t)
    assert((b & 1) != B_oddness)
    goto loop
fin:
    if
    :: b == 0 -> printf("The last pebble is white\n")
    :: else   -> printf("The last pebble is black\n")
    fi

    assert(b+w == 1)
}
```

```
$ spin -a pebble3.pml
$ gcc -o pan pan.c
$ ./pan
hint: this search is more efficient if pan.c is compiled -DSAFETY

(Spin Version 6.4.3 -- 16 December 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 36 byte, depth reached 352, errors: 0
     4178 states, stored
     1512 states, matched
     5690 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.255       equivalent memory usage for states (stored*(State-vector + overhead))
    0.485       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.925       total actual memory usage

unreached in proctype P
        (0 of 30 states)

pan: elapsed time 0.01 seconds
```