The Scholten/Dijkstra Pebble Game Played Straightly, Distributedly, Online and Reversed

Wolfgang Reisig

Department of Computer Science, Humboldt-Universität zu Berlin

With pleasure I remember the visit of Boaz at GMD in Bonn in the early 1980s and at TU Munich in late 1980s as well as long discussions with him in Dagstuhl, and several meetings in Tel Aviv. What a rich source of inspiration!

Abstract. The Scholten/Dijkstra "Pebble Game" is re-examined. We show that the algorithm lends itself to a *distributed* as well as an *online* version, and even to a *reversed* variant.

Technically this is achieved by exploiting the local and the reversible nature of Petri Net transitions. Furthermore, these properties allow to retain the verification arguments of the algorithm.

1 Introduction

University Video Communications [1] distributes a "Distinguished Lecture Series" of "Leaders in Computer Science and Electrical Engineering" where in an "Academic Honour Presentation" Edsger W. Dijkstra talked about *Reasoning about programs.* As an example, Dijkstra presents a "Pebble Game" as an example of a nondeterministic algorithm. Gries in [2] refers the problem to Carl Scholten, due to a letter from Dijkstra in fall 1979. Scholten plays the game with black and white beans in a coffee can. Dijkstra models this algorithm as a guarded command program and proves its decisive properties.

Section 2.1 of this paper recalls Dijkstra's oral presentation of the algorithm as well his program. A Petri Net model of the algorithm is given in Sect. 2.2, and verified in Sect. 2.3.

Part 3 of this paper presents three variants that provide more insight into the nature of the algorithm. In particular, a *distributed* and an *online* version exploit the *local* nature of the algorithm's steps. Furthermore, it is shown that the algorithm can be played "backwards", exploiting the *reversible* nature of the algorithm's steps. Interesting enough, the decisive verification arguments remain valid in all three variants.

2 The Algorithm's Basic Version

2.1 Dijkstra's Algorithm and Model

We quote Dijkstra's oral presentation of [1]:

A. Avron et al. (Eds.): Trakhtenbrot/Festschrift, LNCS 4800, pp. 589–595, 2008.
© Springer-Verlag Berlin Heidelberg 2008

> "... a one person game is played with a big urn full of pebbles where each pebble is white or black. We don't start with an empty urn. ...
>
> The rule is that one goes on playing as long as moves are possible. For a move there have to be at least two pebbles in the urn because a move is the following: What one does is: One shakes the urn, and then looks in the opposite direction, puts one hand in the urn, picks up two pebbles, looks at their color and depending on the color of the two pebbles taken out one puts a pebble in the urn again ...
>
> The idea is that if we take out two pebbles of a different color, we put back the white one. However, if we take out two pebbles of equal color, we put a black one into the urn. (If we take out two white ones, we have to have a sufficient supply of black pebbles) ...
>
> Given the initial content of the urn, what can be said about final pebble?"

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

Dijkstra suggests to annotate this program by assertions, in particular by a loop invariant, thus showing that the final pebble is white if and only if W is odd.

2.2 A Petri Net Model of the Algorithm

To model the algorithm, we start with the pebbles: Initially as well as at any reachable state, the urn contains finitely many white and black pebbles. As a mathematical structure, they form a *finite multiset* (also called a *bag*). Let PEBBLES denote the bag of pebbles initially in the urn.

Next we turn to the urn: In the context of the algorithm, the urn is an item with two properties:

- the urn can contain any bag of white and black pebbles;
- actions can affect the urn, where an action may remove some of the pebbles available in the urn, and may add pebbles to the urn.

A *place* of a (high-level) Petri Net has exactly these properties. So, we model the urn as a Petri Net place.

Finally, we turn to the three actions: Each action is to remove two pebbles with a specific choice of colors from the urn. Occurrence of an action then returns a pebble to the urn. The returned pebbles's color is specified by the rules of the game. This is exactly what a Petri Net transition with corresponding arc inscriptions describes. So, we model each action as a Petri Net transition.

Figure 2 shows the corresponding Petri net. Its steps are the steps of the pebble game and its sequences of steps are the runs of the pebble game. All together, the Petri net of Fig. 2 models the pebble game.

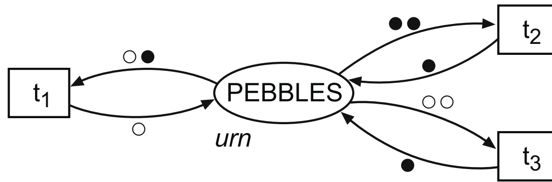


Fig. 2. The basic version of the algorithm

Petri Nets would also allow to model a 2-elementary bag of pebbles be taken out of the urn, instead of two pebbles.

2.3 Verification of the Algorithm

As described by Dijkstra, the algorithm has two decisive properties. Firstly,

$$\text{the algorithm terminates with one pebble, } p\text{, remaining in the urn.} \tag{1}$$

Furthermore, the color of p depends only on the number W of white pebbles in PEBBLES:

$$p \text{ is white if and only if } W \text{ is odd} \tag{2}$$

Property (1) follows immediately from the observation that each occurrence of each transition of the Petri net in Fig. 2 reduces the magnitude of the bag in the urn by one, and that at least one of the transitions is enabled as long as there are at least two pebbles in the urn. As each transition returns a pebble, the process will eventually terminate with one pebble in the urn. Any further formalization of this obvious argument would be a formal overkill.

Proof of property (2) is far less trivial. It is based on a function f that assigns each bag of pebbles one of the numbers 0 or 1. More precisely, if BAG is a bag of pebbles with W white pebbles, let

$$f(BAG) = \begin{cases} 0 & \text{if } W \text{ is even} \\ 1 & \text{if } W \text{ is odd} \end{cases} \tag{3}$$

In particular, for the one-elementary bags $[\bullet]$ and $[\circ]$, $f([\bullet]) = 0$ and $f([\circ]) = 1$.

The decisive argument is now that f is *stable*: For each step $M \xrightarrow{t} M'$ of the algorithm holds:

$$f(M(urn)) = f(M'(urn)) \tag{4}$$

where $M(urn)$ and $M'(urn)$ represent the token load of the place *urn* at the markings M and M', respectively. This rises the question of how to prove (4). Petri Net Theory provides the standard technique of *place invariants* to prove this kind of properties.

The place invariant technique exploits two observations:

- the effect of a transition occurrence boils down to the addition and subtraction of bags;
- the invariant function f as in (4) is linear on bags (with multiset addition) and on its range (in (4), the range is $\{0, 1\}$ with addition modulo 2).

By a simple argument on the iteration of steps, (4) extends apparently to all reachable markings: With M_0 the initial marking and M_ω the final marking of *urn*, we get

$$f(M_\omega(urn)) = f(M_0(urn)). \tag{5}$$

Now it is easy to prove (2): The remaining pebble is white
iff $M_\omega(urn) = [\circ]$
iff $f(M_\omega(urn)) = f([\circ]) = 1$
iff $f(M_0(urn)) = 1$
iff K is odd.
The intuition behind this proof in fact resembles Dijkstra's loop invariant.

Dijkstra's annotations for the program in Fig. 1 may be mirrored in Petri Nets by help of additional places, where an invariant property corresponds to a constant token load. Verification then reduces to problems of transition enabling, and can be attacked by place invariants just as program verification is based on loop invariants. This is merely a matter of convention and syntactic sugar.

3 Variants of the Algorithm

3.4 A Distributed Version of the Algorithm

Dijkstra assumes *one* player to execute the steps of the algorithms in a sequential order. There is no reason to do so. Many players may concurrently remove pairs of pebbles from the urn. To illustrate the most general case, assume a group of children with each child representing a white or a black pebble. The children are collected in a large circle (representing the urn) on the floor of the kindergarden. Any two children may decide to leave the circle together, with one of them returning, colored according to the rule's game.

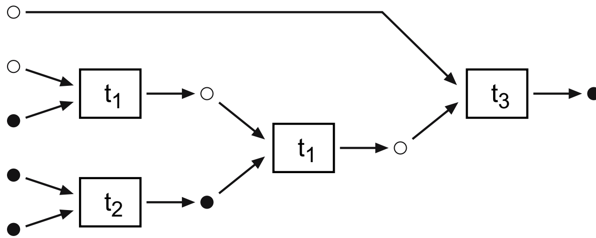
How model this behavior? The initial count of n children yields $n(n-1)$ enabled actions, of which no more than $\lfloor n/2 \rfloor$ occur concurrently. Each action

occurrence reduces n by one. A sequence of sets of actions would misrepresent this kind of behavior, because nothing in the algorithm enforces lockstep behavior. Sequential subprocesses likewise don't shine up.

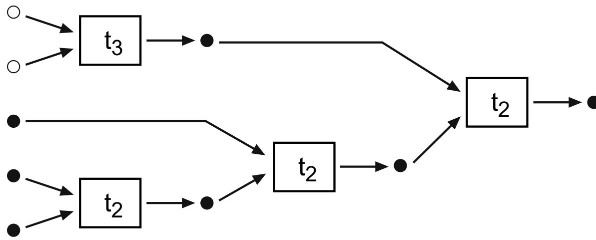
From the very beginning, Petri Nets came with the notion of *distributed runs*: A distributed run is a partially ordered set of *transition occurrences*. An occurrence of a transition t is technically represented as an instance of t, with ingoing and outgoing arcs from and to the tokens that are consumed and produced by an occurrence of t. For example, an occurrence of transition t_1 in Fig. 2 reads

(6)

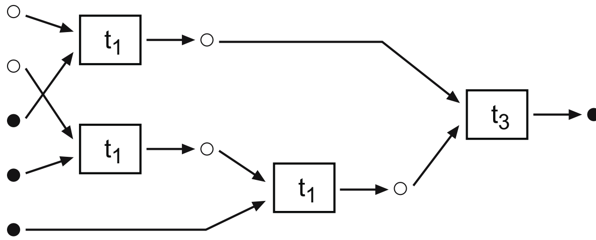
As an example, assume the initial bag PEBBLES of the net in Fig. 2 to contain two white and three black pebbles. Fig. 3 shows three distributed runs with this initial state.



A distributed run of the algorithm



Another distributed run of the algorithm



A distributed run with concurrent occurrences of t_1

Fig. 3. Three distributed runs of the algorithm in Fig. 2

The decisive properties (1) and (2) remain valid in the distributed version. Furthermore, proof of (1) and (2) as given in Sect. 2.3 also apply to the distributed case.

Summing up, the distributed version of the algorithm comes without extra cost. It just employs the notion of distributed runs. This notion is anyway the adequate notion of runs for distributed systems.

3.5 An Online Version of the Algorithm

Both the sequential and the distributed versions of the algorithm assume the initial bag PEBBLE be available before computation starts. There is no reason to do so. The pebbles may be produced elsewhere and be added to the urn *while* the algorithm processes previous pebbles. The urn then serves as an (unbounded) *buffer*. Figure 4 shows this online version of the algorithm.

Property (1) is no longer valid, as we now do not necessarily assume PEBBLES to be a *finite* multiset. In the course of time, infinitely many pebbles may be added to the urn. A weaker property of (2) holds nonetheless:

For each intermediate state S with only one pebble p in the urn, p is white if the number of so far arrived white pebbles is odd. (7)

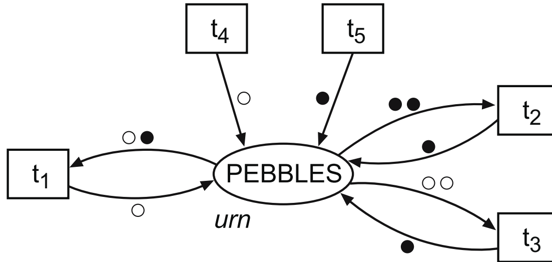


Fig. 4. The online version of the algorithm

3.6 A Reversed Version of the Algorithm

This version reverses the three rules of the game: Each action removes one pebble from the urn and adds two pebbles. The color of the added pebbles depends on the color of the removed pebble: If the removed pebble is white, pebbles with mixed color are added: A white and a black one. If the removed pebble is black, pebbles with equal color are added: Either two black ones, or two white ones. Starting with a single pebble, the algorithm may proceed forever, with unlimited numbers of pebbles collecting in the urn.

To model this behavior, we just reverse the arrow heads in the basic version of the algorithm. Figure 5 shows the result. PEBBLES now consists of only one pebble.

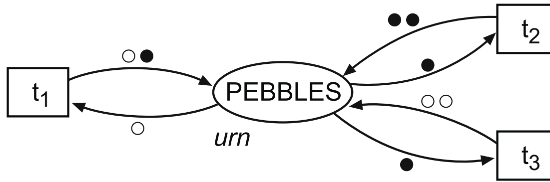


Fig. 5. The reversed version of the algorithm

As an interesting property of this version, in any reachable state S, the number of white pebbles is odd if and only if the color of the initial pebble is white. Proof of this property follows again from the place invariant (4).

One may construct a distributed as well as an online version of the reversed algorithm, in an obvious way.

4 Conclusion

We agree that an adequate invariant decisively supports both intuition and verification of the pebble algorithm. We challenge however that guarded commands was the most adequate modeling technique for this algorithm.

Removing or adding pebbles from or to the urn is *locally confined* to the pebbles affected. Different pairs of pebbles can therefore independently be processed. Both the distributed and the online version of the algorithm exploit locality of the actions.

Removing or adding pebbles from or to the urn are *reversible* actions: Knowing the resulting state and the action that caused the state allows re-tracing the start state. Assignment statements are in general not reversible. The reversed version of the algorithm exploits the reversibility of the actions.

Acknowledgements

H. Völzer brought the pebble game to my attention. G. Goos pointed me at David Gries' book. D. Bjørner gave me some additional hints.

References

1. Dijkstra, E.W.: Reasoning about programs, University Video Communications, Stanford. The Distinguished Lecture Series, Academic Leaders in Computer Science and Electrical Engineering, vol. III (1990)
2. Gries, D.: The Science of Programming, pp. 165–301. Springer, Heidelberg (1981)