# The mathematics behind the Banker's Algorithm.

(I recently showed at my lectures the so-called "Banker's Algorithm" as an example of a method for deadlock prevention. Because my informal justification left my students visibly unconvinced, I designed a more explicit one while preparing my next week's lectures. This note is written because I think the argument I developed at that occasion rather nice; it is not a symptom of any revival of my interest in the Banker's Algorithm as a scheduling strategy.)

We consider a non-empty set P of processes p , each of them engaged on a finite transaction for the completion of which it may need a (varying but bounded) number of units of some shared resource at its exclusive disposal. (The units are all equivalent, say: pages of store.)

A process may "borrow" one or more units, which are then added to its current "loan", it may "return" one or more units, which are then subtracted from its current loan. The act of borrowing is restricted by the condition that, for each process, the loan will never exceed a pre-stated "need", i.e. the maximum number of units that may be simultaneously needed by that process for the completion of its transactions. The act of returning is restricted by the (obvious) constraint that for no process the loan can ever become negative; upon completion of a transaction, the corresponding loan returns to zero.

If there are "cap" units in the system, the sum of the loans cannot exceed cap . More precisely, if we define

$$\text{cash} = \text{cap} - \text{sum}(p\ \underline{\text{from}}\ P\text{: loan}[p]) \qquad (1)$$

then "cash" represents the number of unallocated units and must satisfy

$$0 \le \text{cash} \le \text{cap} \quad . \qquad (2)$$

For each process p we have

$$0 \le \text{loan}[p] \le \text{need}[p] \le \text{cap} \quad . \qquad (3)$$

A simple example shows that the danger of deadlock is present. Consider with two processes the following pattern of loans and needs:

$$\text{cap} = 4 \text{ , need}[0] = \text{need}[1] = 3 \text{ , loan}[0] = \text{loan}[1] = 2 \text{ , cash} = 0 \quad .$$

Because for each process $loan < need$ still holds, each process is entitled to request a further unit before returning units; because, however, $cash = 0$ , deadlock would result if they both do so.

The act of borrowing is, therefore, split into two parts. The process requests the units to be borrowed from a banker and waits until the banker has granted this request.

Definition. A "pattern" (of loans and needs) is "safe" if a granting strategy exists such that it can be guaranteed that all (current and future) requests can be granted within a finite period of time. (End of definition.)

It is the function of the banker to keep the pattern safe. The banker does so by inspecting for each request, whether the pattern that would result from granting that request is safe or not. If it is safe, the request can be granted immediately --and we assume that then the banker does so-- . If it is not safe, the banker postpones the granting of that request until a more favourable moment: because the postponement has not changed the pattern of loans and needs, which is therefore still safe, that moment will come within a finite period of time. It is the purpose of the so-called "Banker's Algorithm" to investigate, whether a given pattern of loans and needs is safe or not.

* * *

For each process p we introduce as abbreviation

$$claim[p] = need[p] - loan[p] \quad ;$$

for each process p the current claim[p] thus represents the maximum number of units it may need to borrow before it returns any units. Suppose that P consists of N processes, and that

$$p[0],\ p[1],\ \dots\ ,\ p[N-1]$$

represents a permutation of the process numbers such that

$$(\underline{A}\ i:\ 0 \le i < N:\ claim[p[i]] \le cash + sum(0 \le j < i:\ loan[p[j]])) \qquad .(4)$$

Lemma 1. Relation (4) implies that the pattern is safe. (End of lemma 1.)
Proof. The existence of a granting strategy such as required for safety is

shown by the strategy of <u>only</u> granting (<u>all</u>) requests from process $p[i]$ , provided that all processes $p[j]$ for $0 \leq j < i$ have terminated their transactions. Relation (4) then implies that for $i = 0, 1, \ldots, N-1$ in succession, cash will be sufficient to grant all requests from process $p[i]$ without violating (2) . Within a finite period of time, process $p[i]$ will have terminated its transaction and $i$ can be increased by 1. (End of proof.)

The Banker's Algorithm tries to find such a permutation of the process numbers by keeping

$$(\underline{A}\ i\colon 0 \leq i < k\colon \text{claim}[p[i]] \leq \text{cash} + \text{sum}(0 \leq j < i\colon \text{loan}[p[j]])) \qquad (5)$$

invariant. After having established it (trivially) by means of $k := 0$ , it then tries to increase $k$ by 1 under invariance of (5) until $k = N$ . It does so by not changing $p[0], \ldots, p[k-1]$ , and by searching for an $h$ , such that

$$k \leq h < N\ \underline{\text{and}}\ \text{claim}[p[h]] \leq \text{cash} + \text{sum}(0 \leq j < k\colon \text{loan}[p[j]]) \quad . \qquad (6)$$

If such an $h$ has been found,

"p:swap(h, k); k:= k+1"

increases $k$ by 1 under invariance of (5). If, however, for $k < N$ equation (6) has no solution for $h$ , we say that "the ordering effort has failed". If (6) remains solvable each time, until $k = N$ , we say that "the ordering effort has not failed".

Because an ordering effort that does not fail implies the existence of a permutation satisfying (4), and, hence, on account of lemma 1, that the pattern is safe, we conclude that for a pattern that is not safe, all ordering efforts must fail. Or, with

| | |
|---|---|
| Ass.0: | the pattern of loans and needs is not safe |
| Ass.1: | all ordering efforts must fail |

we have derived

$$\text{Ass.0} \Rightarrow \text{Ass.1} \quad . \qquad (7)$$

With

Ass.2: a failing ordering effort is possible

we conclude (because the set of possible ordering efforts is not empty) that

Ass.1 ⇒ Ass.2 . (8)

Consider next

Ass.3: the non-empty set of processes --or, to be quite precise, the non-empty set P' of process numbers-- can be partioned into A + B , such that B is non-empty and
$(\underline{A}\ b\ \underline{\text{from}}\ B\text{: claim}[b] > \text{cash} + \text{sum}(a\ \underline{\text{from}}\ A\text{: loan}[a]))$ .

We can then conclude that

Ass.2 ⇒ Ass.3 . (9)

Proof. Consider the state as reached by the failng ordering effort that is possible under the assumption of Ass.2 . Choose then

$$A = \{p[j] \mid 0 \le j < k\} \quad ;$$

from which we conclude that

$$\text{cash} + \text{sum}(a\ \underline{\text{from}}\ A\text{: loan}[a]) = \text{cash} + \text{sum}(0 \le j < k\text{: loan}[p[j]]) \quad ;$$

choose furthermore

$$B = \{p[j] \mid k \le j < N\} \quad ;$$

because $k < N$ , B is not empty, and because the ordering effort has failed, (6) has no solution for h , and hence A and B satisfy the criteria that are imposed upon them in Ass.3 . (End of proof.)

Finally we conclude

Ass.3 ⇒ Ass.0 (10)

Proof. Let all processes from B from now on try to borrow until their loans equal their needs, before they return any units. Let all processes from A terminate their activity. In spite of what has been returned, Ass.3 implies that the banker has still not enough in cash to see any process from B through to completion, and, hence, the pattern of loans and needs is not safe. (End of proof.)

Combining (7), (8), (9), and (10), we see

$$\text{Ass.0} \Rightarrow \text{Ass.1} \Rightarrow \text{Ass.2} \Rightarrow \text{Ass.3} \Rightarrow \text{Ass.0}$$

but from this cyclic implication we are allowed to conclude

$$\text{Ass.}0 = \text{Ass.}1 = \text{Ass.}2 = \text{Ass.}3 \qquad . \qquad (11)$$

Conclusion (11) is the important one. While it is obvious that a non-failing ordering effort implies that the pattern is safe, (11) implies that the discovery of a single failing ordering effort allows us to conclude immediately --i.e. without any of the back-tracking that is traditionally involved in the search for permutations satisfying some criterion-- that no such permutation exists and that the pattern is not safe.

From (11) it also follows rapidly that, in order to investigate the safety of the pattern that would result from granting, in a safe situation, a request to process $c$ , the ordering effort can be stopped as soon as $c = p[k]$ , for then safety is already implied. (The credit for this discovery is due to L.Zwanenburg, who made it in the early sixties.)

* * *

In retrospect I am grateful to the puzzled looks on my students' faces. That from a cyclic arrangement of $n$ assertions, each implying the next one, we can conclude that all $n$ assertions are equivalent --or to put it more dramaticly: can conlude all $n(n-1)$ pair-wise implications-- is not unknown at all. But the larger the value of $n$ , the more impressive an example of effective reasoning we have, in particular if --as in this case-- the assertions have been arranged in such an order, that the $n$ antecedents are not difficult to prove.

It is a pity that, probably, the case $n = 2$ is the most common one, for in that case the "gain" --as measured in terms of the number of implications established-- is nihil!

Plataanstraat 5
5671 AL NUENEN
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow

PS. Please note my new postal code!