ON A SOLUTION AND A GENERALIZATION OF THE CIGARETTE SMOKERS' PROBLEM

Nico Habermann

August 1972

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research. This document has been approved for public release and sale; its distribution is unlimited.

ON A SOLUTION AND A GENERALIZATION OF THE CIGARETTE SMOKERS' PROBLEM

Nico Habermann

August 1972

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research. This document has been approved for public release and sale; its distribution is unlimited.

ABSTRACT

The author of the Cigarette Smokers' Problem (S. Patil, MIT) tried to show that the problem could not be solved using Dijkstra's P- and V-operations without conditional statements. D. L. Parnas (CMU) showed that Patil's proof was false by presenting a solution using P- and V-operations, but no conditional statements. This paper presents first a correctness proof of Parnas' solution. This solution leads to an obvious generalization and in connection with it the question of an optimal set of distinguishable codewords is addressed. Finally a more natural approach to the problem is discussed and a solution for this variation is presented with its correctness proof.

INTRODUCTION

The claim that the Cigarette Smokers' Problem has no solution using P- and V-operations as defined by Dijkstra [1] without conditional statements [2,3] is seriously challenged by the solution of the problem given by D. L. Parnas [4]. It will be shown that the solution is indeed correct and so the claim that no solution exists is not justified without adding other restrictions.

The proof is facilitated by representing the processes and ingredients involved as indexed members of an appropriate set rather than naming them explicitly. It turns out that this representation shows how the problem can be generalized quite naturally to more ingredients than just cigarette paper, tobacco and matches. Considerations about the supply of the ingredients lead to a modification of the problem for which still a solution can be found using Dijkstra's P- and V-operations without conditional statements.

REPHRASING THE PROBLEM

Three cyclic processes, called suppliers (together called the agent in Patil's paper [2]), make their moves in a not specified order. The term "move" of a cyclic process means that it is going once through its program. The programs of the suppliers are given and read:

```
process supplier(i) =                    c  i = 0,1,2
   begin sup: P(s);
         V(ingr[(i+1)%3]); V(ingr[(i+2)%3]);
         goto sup
    end
```

The symbol % represents the remainder function. Semaphore s serves to sequence the supplier moves (s-moves for short) and its initial value is one. Semaphores ingr[0:2] represent the three ingredients and the V-operations on those signals represent the arrival of two of the three ingredients. The initial value of the ingredient semaphores is zero.

The objective is to program three user processes, here called addicts (smokers in Patil's paper), which are activated each on a different combination of two ingredients out of the three. In order to avoid confusion, the problem statement is quoted from Patil's paper:

> "The smokers' problem is, then, to define some additional semaphores and processes, if necessary, and to introduce necessary P and V statements in these processes so as to attain the necessary cooperation among themselves required to ensure continued smoking of cigarettes without reaching a deadlock. There is, however, a restriction that the process which supplies the ingredients cannot be changed and that no conditional statements may be used."

As pointed out correctly by Patil, the addict programs:

```
process addict(j) =                    c  j = 0,1,2
   begin ad: P(ingr[(j+1)%3]);
             P(ingr[(j+2)%3]);
             V(s);
             goto ad
   end
```

may cause a deadlock, for instance if ingredients 0 and 1 are supplied and

addict(0) passes P(ingr[1]) before addict(3) does. But the conclusion that no solution without conditional statements would exist is refuted by Parnas' solution [4]. It is here presented in the form of three pushers and three addicts.

```
process pusher(j) =                    c  j = 0,1,2
   begin pu: P(ingr[j]);
         P(mutex);
         t  := t+2↑j; V(a[t]);
         V(mutex);
         goto pu
   end
```

The critical section ensures that pushers will not operate on t simultaneously.

```
process addict(k) =                    c  k = 3,5,6
   begin ad: P(a[k]);
         t := 0;
         V(s);
         goto ad
   end
```

The initial values of semaphore mutex and semaphores a[k] are one and zero, respectively. The initial value of t is zero.

THE CORRECTNESS PROOF

To prove that this solution is deadlock free, it will be shown that a move by a supplier (an s-move) is followed by two pusher moves (p-moves) and one addict move (a-move) and this sequence enables exactly one subsequent s-move.

From a paper on correctness proofs of synchronizing processes [5] we borrow the result

$$m(sem) = MIN(p(sem), c+v(sem))$$

which says that the number of times [m(sem)] processes moved on passed an execution of P(sem) equals the minimum [MIN] of the number of times P(sem) was executed [p(sem)], and the number of times V(sem) was executed [v(sem)] incremented by a given initial constant [c]. The equation is an invariant for P- and V-operations. It is valid if sem represents a semaphore, but also if sem represents a group of semaphores, in particular a semaphore array.

Applied to this particular example, we have

$$m(s) = MIN(p(s), 1+v(s))$$
$$m(ingr) = MIN(p(ingr), v(ingr))$$
$$m(a) = MIN(p(a), v(a))$$

where "a" represents the semaphore set {a[i]|i =* 3,5,6*} and "ingr" represents {ingr[i]|i - 0,1,2}. Initially v(s) = v(ingr) » v(a) = 0. It is assumed that none of the processes stops deliberately, since it should be shown that no process will be stopped unintentionally if all are eager to move, i.e., if the numbers p(s), p(ingr) and p(a) are maximal.

Let a_1 represent the action of starting the system.

Lemma 1. Starting the system results in exactly one supplier move.

Proof. Given the initial condition v(s) = v(ingr) = v(a) = 0, the system is started by allowing all processes an attempt to move, thus

p(s) = p(ingr) = p(a) = 3.

→ m(s) = MIN(3,1) = 1
m(ingr) = MIN(3,0) = 0
m(a) = MIN(3,0) = 0

Hence, exactly one supplier will move.

Let s_i, p_i and a_i represent the ith s-move, p-move and a-move, respectively. The result of lemma 1 is then $a_0 < s_1 < p_1$ and $a_0 < s_1 < a_1$, where the symbol < represents ordering in time.

Lemma 2. Move s_i is caused by move a_{i-1} (if any) for i = 1,2,...

Proof. Let i = 1; move s_1 is caused by action a_0 (lemma 1).

Let i > 1;

attempt to move s_i means p(s) ≥ i

before move a_{i-1} v(s) = i-2

→ m(s) = MIN(p(s), 1+v(s)) = MIN(p(s), 1+i-2) = i-1

after move a_{i-1} v(s) = i-1

and so, move a_{i-1} causes move s_i and $a_{i-1} < s_i$

Lemma 3. Move s_i (if any) causes moves p_{2i-1} and p_{2i} for i = 1,2,...

Proof. Attempts to moves p_{2i-1} and p_{2i} mean p(ingr) ≥ 2i

before move s_i v(ingr) = 2(i-1)

→ m(ingr) = MIN(p(ingr), v(ingr)) = MIN(p(ingr), 2(i-1)) = 2(i-1)

after move s_i v(ingr) = 2i

→ m(ingr) = MIN(p(ingr), 2i) = 2i

and so, move s_i causes moves p_{2i-1} and p_{2i}

and $s_i < p_{2i-1}, p_{2i}$.

Lemma 4. The second s-move is preceded by the first pair of p-moves and the first a-move in that order.

Proof.

a) s_1 is the first move to occur (lemma 1). It causes p_1 and p_2 (lemma 3) $\rightarrow s_1 < p_1, p_2$.

b) $\left.\begin{array}{l} a_1 < s_2 \quad \text{(lemma 2)} \\ s_2 < p_3, p_4 \quad \text{(lemma 3)} \end{array}\right\} \rightarrow a_1 < p_3, p_4$

Thus, if move a_1 occurs, it is at most preceded by moves s_1, p_1 and p_2.

c) initial value of $t = 0$

move a_1 occurs when $t = 2\uparrow x + 2\uparrow y$

where $x \neq y$ and $x, y \in \{0,1,2\}$

This is iff $p_1, p_2 < a_1$ (see programs).

d) move a_1 causes move s_2 (lemma 2)

Conclusion: $s_1 < p_1, p_2 < a_1 < s_2$

Note that the proof is constructive; it shows not only that the moves could not occur in any other order, but also that the moves will occur in that order.

Let w_i be the set of moves $\{a_{i-1}, s_i, p_{2i-1}, p_{2i}\}$ for $i = 1,2,\ldots$ (where a_0 represents getting the system started).

w_i is partially ordered:

$$a_{i-1} < s_i < p_{2i-1}, p_{2i} \qquad \text{(lemma 2,3)}$$

Lemma 5. If $w_1,\ldots,w_k$ ($k \geq 2$) are performed strictly in sequence, this sequence causes move a_k.

Proof.

a) move s_ℓ ($\ell \in \{1,2,\ldots,k\}$) is preceded by an a-move and so the variable t = 0 (see program), when move s_ℓ starts

b) $s_j < p_{2j-1}, p_{2j} < a_j$ for j = 1,...,k-1 because of the strict sequencing.

The sequence $s_j < p_{2j-1}, p_{2j}$ causes v(a) := v(a)+1 (see program and lemma 4,c).

→ at move a_{k-1} v(a) = k-1 = m(a), because v(a) started at zero and was k-1 times incremented.

c) When move a_k is attempted, p(a) ≥ k.

move a_k can be performed iff MIN(p(a),v(a)) = k

→ move a_k requires v(a) := k

(b) in conjunction with (c) implies that move a_k will occur then, when v(a) is incremented by one after move a_{k-1}. But, according to (a), v(a) is incremented by one after move a_{k-1} not until both moves p_{2k-1} and p_{2k} have been completed (see also lemma 4,c).

Lemma 6. $w_i < a_i$ and $w_i < w_{i+1}$ for i = 1,2,... (w_i and w_{i+1} are performed strictly in sequence).

Proof.

a) It is sufficient to prove $w_i < a_i$ for i = 1,2,... because a_i

is the first move of w_{i+1} (see definition) and so relation $w_i < a_i$ implies $w_i < w_{i+1}$.

b) (induction)

$w_1 < a_1$ is true (lemma 4)

Suppose $w_i < a_i$ for $i = 1,\ldots,k-1$.

It remains to be proved that this implies that $w_i < a_k$ is also true.

The assumption implies $w_1 < w_2 <\ldots< w_k$, i.e., $w_1,\ldots,w_k$ is performed strictly in sequence. But this sequence causes move a_k (lemma 5) and so $w_k < a_k$.

The correctness proof of Parnas' solution is now simple:

Theorem. Parnas' solution to the Cigarette Smokers' Problem is deadlock free.

Proof. The proof of lemma 6 showed constructively that suppliers, pushers and addicts will move after starting the system in a sequence that allows concurrency of pushers only and that can be described as

$$(s < (p,p) < a)^*$$

GENERALIZATIONS OF THE PROBLEM

The representation used in the preceding sections shows that it is not very important that the total number of ingredients is fixed to three. Problem and solution can easily be generalized to n suppliers ($n \geq 2$),

each producing a different set of n-1 ingredients:

```
process supplier(i) =          c  i = 0,1,2,...,n-1
   begin integer ℓ;
         sup: P(s);
         for ℓ := 0 until i-1, i+1 until n-1 do V(ingr[ℓ]);
         goto sup
   end
```

Note that the use of the artificial for statement does not really violate the restriction that no conditional statements may be used. Variable ℓ is a local, not operated upon by any other process than supplier(i). The notation describes that supplier(i) performs a V-operation on all ingredient semaphores but ingr[i]. Thus, the sequence of V-operations does not depend on any external operation and the for statement could be replaced by the fixed sequence of V-operations.

As before, the initial values of semaphores s and ingr[0:n-1] are one and zero, respectively. The generalized problem is to write programs for a set of n addicts that should be activated each on the occurrence of a different subset of n-1 ingredients as produced by the suppliers. Parnas' solution can be generalized very easily to solve this problem: the programs of pushers and addicts stay exactly the same! The indices of the pushers are j = 0,...,n-1 and of the addicts:

$$\{k \mid k = 2\uparrow(n+1) - 1 - 2\uparrow i,\ i = 0,\ldots,n-1\}$$

i.e., the binary representation of k is a number of n bits with a zero in position i.

Another generalization is not to fix the number of ingredients produced by the suppliers at n-1, but let them supply one of the $\binom{n}{q}$ different

subsets of q ingredients $0 < q < n$. There are now $\binom{n}{q}$ suppliers, again n pushers, and $\binom{n}{q}$ addicts.

This version can still be generalized without changing the programs of pushers and addicts. Let us call the set of ingredient semaphore indices, on which a supplier performs a V-operation, a codeword and the set of all codewords the code. The last generalization can then be characterized as a codeset of n signals with a fixed codeword length of q signals $(0 < q < n)$.

A further generalization would be not to fix the codeword length q: instead, the suppliers transmit codewords of various length with the restriction that none of the codewords is a proper subset of any of the others. This restriction is necessary to avoid ambiguity, for, if one supplier, for instance, transmits codeword (a,b,c) and another one codeword (a,b), an addict waiting for codeword (a,b) could snatch this one away from the addict waiting for codeword (a,b,c).

It can be shown, however, that the last generalization viz. to allow variable codeword length, is not very useful. To leave a particular signal (ingredient) out of a codeword should make a difference: let A,B,... be the codewords produced by the suppliers; ingredient $a \in A$ distinguishes codeword A properly from codeword B if $a \notin B$, but $A - \{a\} \subset B$. Since $B \subset A$ creates an ambiguity, there is also a $b \in B$ such that $b \notin A$.

Codeword A is distinguishable from all other codewords if A is not empty and all its elements (ingredients, signals) distinguish A properly from all other codewords. For convenience we will say that codeword A is distinguishable in case the code consists of one signal only and A is that signal.

It seems as if the last generalization, to allow variable codeword length, increases the number of distinguishable codewords in a code of n signals. But the theorem below states that this is not true and so the word length might as well be fixed.

Theorem. The maximum number of distinguishable codewords in a code of n signals is $\binom{n}{q}$, where q = ENTIER(n/2), n ≥ 1.

Proof. A codeword A can be represented by its characteristic function x on the ordered set of n signals: suppose the code set = {u,v,w,x,y,z} and codeword A = (u,y), then x(A) = (100010).

Let Γ be the codeset. If Γ consists of signal a only, there is only one distinguishable set, viz. A = {a} and x(A) = (1). If Γ = {a,b}, there are two distinguishable sets (see definition), viz. A = {a} and B = {b} and so x(A) = (1 0) and x(B) = (0 1). Hence, the statement is true for n=1 and n=2. Let n > 2 and let A be a distinguishable codeword and a ∈ A a (distinguishing) signal. Signal a distinguishes codeword A from a codeword B, i.e., there is a codeword B and a signal b ∈ B such that a ∉ B, b ∉ A and A - {a} ⊂ B. The codeset Γ can be ordered such that signals a and b are the first two elements: Γ = {a,b,...}. Hence, x(A) = (1 0...) and x(B) = (0 1...).

In order to obtain the maximum number of codewords the remainder of codeword A must distinguish A from all other codewords whose characteristic function also begins with 1 0, i.e., from all other codewords that also contain signal a but not signal b. Let Γ* be the codeset derived from Γ by deleting signals a and b. The remainder of codeword A must be a

distinguishable codeword in code Γ^* with $n^* = n-2$.

If $n^* = 1$ then $A^* = \Gamma^* = \{c\}$ and $x(A) = (1\ 0\ 1)$.

If $n^* = 2$ then $x(A^*) = (1\ 0)$ after ordering Γ^* properly and so

$x(A) = (1\ 0\ 1\ 0)$.

If $n^* > 2$ then $x(A^*) = (1\ 0 \ldots)$ after ordering Γ^* and so $x(A) = (1\ 0\ 1\ 0\ \ldots)$. The remainder of A^* can be broken down and this process ends (because n is finite number) when the reduction comes to a codeset of only one or two signals.

The consequence is that the characteristic function of a codeword has the same number of ones as zeros if n is even, and one more one than zeros if n is odd. But the number of codewords with that property equals $\binom{n}{q}$ where $q = n/2$ if n is even and $q = (n+1)/2$ if n is odd. In the latter case $\binom{n}{q} = \binom{n}{q-1}$ as is well known and so the statement follows.

It is interesting to note that, due to $\binom{n}{q} = \binom{n}{q-1}$ for odd n and $q = (n+1)/2$, it is not necessary to fix the codeword length at $(n+1)/2$: the same number of distinguishable codewords is obtained with $q = (n-1)/2$. In case $n=3$, for instance, and $\Gamma = \{a,b,c\}$ there are at most three distinguishable codewords A, B and C and either

$$x(A) = (1\ 0\ 0),\ x(B) = (0\ 1\ 0),\ x(C) = (0\ 0\ 1)$$

or $$x(A) = (0\ 1\ 1),\ x(B) = (1\ 0\ 1),\ x(C) = (1\ 1\ 0)$$

But a mixture of these codewords does not constitute a distinguishable set.

This theorem has found an application in the ALGOL 60 Compiler for the PDP-10. The problem was to code nine types in a field preferably smaller than nine bits, but in such a manner that the type information can be extracted by means of one test instruction. The PDP-10 allows a

selective test for all zeros (unfortunately not a test for all ones). A nine bit field is certainly wide enough to code the types by means of exactly one 1 in a particular field. The test can then be either a non-zero test on a particular bit or an all zero test on the remaining bits. But the theorem above says that a codeset of 5 signals with a codeword length of two or three will suffice, i.e., a 5-bit field and all patterns with exactly three ones gives a set of 10 distinguishable codewords. To play it safe, we reserved a 6-bit type field and coded the types as patterns with three ones and three zeros. This allows for a set of (^) 20 types. We added three new types later on bringing the total number to 12 and so we were very happy after all with the earlier decision to reserve a 6-bit type field.

The programs need hardly to be changed to implement the m - (^) suppliers, n pushers and m addicts, where n is the number of ingredients and q the number of ingredients on which a supplier performs a V-operation. Each supplier program has a sequence of q V-operations that distinguishes it from all other suppliers. The function of the pushers is essentially to assemble a codeword, i.e., the characteristic function of the sequence of V-operations performed by a supplier. The indices of the addicts are the numbers whose binary representation is one of the codewords.

There are only minor changes in the correctness proof. In lemma 3: move s_1 causes moves $p_{(i-1)q+1}\, p_{(i-1)q+2}$ lemma 4 should read: the second s-move is preceded by the first q p-moves and the first a-move in that order. The changes are straightforward and so a repetition of the complete proof is omitted.

VARIATION ON PROBLEM AND SOLUTION

Going back to the original problem statement, we quote from Patil:

> ".....On the table in front of them, two of the three ingredients will be placed, and....."

and further on:

> "....To inform the smokers about the ingredients which are placed on the table, three semaphores a, b and c representing tobacco, paper and match, respectively are provided. On placing an ingredient on the table, the corresponding semaphore is incremented by performing a V-operation."

There is nothing in the problem statement that suggests or requires that the two ingredients should be placed on the table by one process. But Patil's agent and the suppliers discussed in the preceding sections do exactly that. It seems, however, more natural to assume that the ingredients are provided more or less independently of each other. In terms of the last generalization: it is more natural to assume n (instead of m) suppliers, who each signal one ingredient. Suppliers should cooperate to the extent that providing an ingredient should be stopped after a set of q until the addicts are ready to accept another set. Also, such a set should not contain more than one ingredient from any one of the suppliers. The problem is then to program the suppliers, the addicts, each reacting on a different set of q ingredients, and additional processes, if needed, together forming a deadlock free system.

It seems that phrasing the problem this way is far off from Patil's original problem statement, especially because he requires from any solution that it does not need to alter his agent. But taking a closer look, it appears that the only essential thing about the agent that should not be allowed to be changed is that an ingredient is provided by means of a V-operation on the corresponding semaphore. If this restriction is removed, too, then a trivial solution would exist, in which a V-operation by a supplier is replaced by an addition of 2↑j to the variable t of Parnas' solution, and take over in this way all the work of the pushers. So, the sting of the problem is still there, since an ingredient is provided by means of a V-operation.

Let n be the number of ingredients (signals) and q the codeword length (the number of ingredients wanted by addicts). Let $m = \binom{n}{q}$ and $z = 2\uparrow n - 2\uparrow(n-q)$. semaphore array s[0:n-1] serves to prevent a supplier from providing its ingredient twice when a series of q is being collected. The initial value of these semaphores is one. semaphore array ingr[0:n-1] represents the ingredients; the initial value of the semaphores is zero. semaphore array a[1:z] represents the set of semaphores of which only a subset is used corresponding to the desired codewords. (As Parnas observed, if overflow is a problem, we can introduce dummy addicts for all the irrelevant semaphores. Their program is nothing more than a repetition of a P-operation on such an irrelevant a-semaphore. The initial value of the a semaphore is zero. semaphore quant indicates the number of ingredients that must be provided to complete a codeword. Its initial value is q. semaphore mutex serves to ensure mutual exclusion when operations with variable t are performed. Its initial value is one. integer t is a

variable in which a codeword is assembled; its initial value is zero.

```
process supplier(i) =                    c  i = 0,1,...,n-1
   begin sup:  P(s[i]);
               P(quant);
               V(ingr[i]);
               goto sup
   end
```

The function of the first P-operation is to prevent that ingredient[i] is produced again before an addict has signaled the acceptance of the preceding instance. The subsequent P-operation stops the suppliers after a group of q ingredients (a codeword) have been produced. The intention is that eventually one addict reacts on this codeword and this addict is supposed to perform q V-operations on semaphore quant to allow production of the next codeword. The order of the two P-operations is significant; with the order reversed, a deadlock would arise if a supplier passed P(quant) twice and it was stopped in P(s[i]). When quant gets zero in that situation, at most q-1 ingredients have been produced and so no addict will be activated and do V(quant) to release the suppliers.

```
process pusher(j) =                      c  j = 0,1,...,n-1
   begin pu:  P(ingr[j]);
              P(mutex);
              t := t+2↑j; V(a[t]);
              V(mutex);
              goto pu
   end
```

The pusher program is still the same as in the preceding sections. The function of the pushers is to assemble the codeword in t, which results

eventually in waking up one of the addicts.

```
process addict(k) =      c  k = Σ(i=0 to n-1) c_i 2^i, c_i=0 or 1, Σ(i=0 to n-1) c_i=q
   begin integer array c[1:q];
         integer i,j; j := 1;
         for i := 0 until n-1 do
            if k%2↑(i+1) ≥ 2↑i then
               begin c[j] := i; j := j+1 end;

    ad:  P(a[t]);
         t := 0;
         for i := 1 until q do
            begin V(quant); V(s[c[i]]) end;
         goto ad
   end
```

The first for statement in the addict program initializes array c with the set of ingredient indices that is typical for the kth addict. Array c and both the for statements serve the purpose of representing all addict programs in one for all possible n and q. When a particular n, q and k are given, array c and the first for statement can be left out entirely. The second for statement can then be replaced explicitly by the characteristic (and fixed) sequence of V-operations typical for that addict.

The idea of the statement P(a[t]) is, as in the preceding sections, to activate an addict right after a codeword has been assembled. It is apparently assumed that, when an addict makes its move, no pusher is active, for variable t is reset to zero with the idea that t is still zero when the next set of ingredients is going to be produced. The subsequent for statement enables the production of the next set of q ingredients and releases the suppliers which contributed to the current codeword.

CORRECTNESS PROOF OF THE VARIATION

We recall the equation m(sem) = MIN(p(sem), c+v(sem)), where m(sem) represents the number of times a P-operation on sem has been passed successfully. Application leads to:

$$m(s) = MIN(p(s), n+v(s)) \quad (1)$$
$$m(quant) = MIN(p(quant), q+v(quant)) \quad (2)$$
$$m(ingr) = MIN(p(ingr), v(ingr)) \quad (3)$$
$$m(a) = MIN(p(a), v(a)) \quad (4)$$

The initial value of v(s), v(quant), v(ingr) and v(a) is zero.

a) according to (1) n suppliers are able to begin when the system is started

b) according to (2) not more than q suppliers can complete their move until v(quant) is incremented in move $a_{j'} \rightarrow a_j \leq s_{q+1}$

c) according to (3) a pusher will move when a supplier performs v(ingr), hence, the first q suppliers activate the first q pushers $\rightarrow s_i < p_i$ for all i

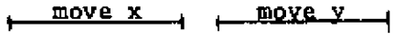
d) an addict cannot move until a pusher performs the right V(a[t]) according to (4)
("Right" means that the codeword $t = \sum_{i=0}^{n-1} c_i 2^i$, where $c_i = 0$ or 1 and $\sum_{i=0}^{n-1} c_i = q$.) This implies, since variable t = 0 initially, that move a_j is preceded by at least q p-moves.)

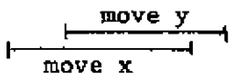
e) a supplier cannot move a second time until an addict has moved.

(d) in conjunction with (e) implies that the first q s-moves produce different ingredients and activate q different pushers. Hence, the first q p-moves produce a codeword and so the first q

p-moves cause the first a-move. → $s_j < p_j < a_1$ for all $j = 1,\ldots,q$.

Let the notation $x < y$ mean that move x causes move y, and x strictly precedes y in time; let the notation $x \leq y$ mean that move x causes move y, and x is all done before y is completed.

$x < y$ 

$x \leq y$ 

Let $c_i = \{s_k, p_k \mid s_k < p_k,\ k = i*q+1,\ldots,i*q+q\}$, for $i = 0,1,\ldots$. Thus, c_i represents the ith set of q s-moves and q p-moves. This set is not ordered, but its first element is s_{i*q+1} and it is completed when all its p-moves are completed.

If a_0 represent getting the system started, the result of the analysis above is

$$a_0 \leq c_0 < a_1$$

The following theorem implies that the system runs deadlock free:

Theorem. If P-operations are performed when possible, suppliers, pushers and addicts move in the order $a_0 \leq c_0 < a_1 \leq c_1 < a_2 \leq c_2 < \ldots\ldots\ldots$

Proof. The statement is true for the sequence $a_0 \leq c_0 < a_1$ (see above). Note that c_0 not just precedes a_1 in time, but it actually causes a_1, i.e., the compound move c_0 incremented v(a) by one (see (4)). Suppose it had been proven that the programs generate the sequence $a_0 \leq c_0 < a_1 \ldots a_{i-1} \leq c_{i-1}$ and that c_j results in incrementing v(a) by one

for j = 0,1,...,i-1. It must be shown that this assumption implies that, if possible P-operations are performed, the sequence

$$a_0 \leq c_0 < a_1 \ldots a_{i-1} \leq c_{i-1} < a_i \leq c_i$$

is generated and that c_i also results in v(a) := v(a)+1.

a) When c_{i-1} is completed, i-1 a-moves have been completed not counting a_0. Hence, the number of possible P-operations on the a-semaphores is $p(a) \geq i-1 + \binom{n}{q} \geq i$. v(a) is initialized at zero and incremented i times → v(a) = i. Hence, m(a) = MIN(p(a),v(a)) = MIN(p(a),i) = i and so c_{i-1} causes a_i and $c_{i-1} < a_i$.

b) It will be shown next that move a_i causes s-moves $s_{i*q+1},\ldots,s_{i*q+q}$, which cause in turn p-moves $p_{i*q+1},\ldots,p_{i*q+q}$. When variable t is reset to zero in move a_i, v(quant) = (i-1)*q (for, i-1 a-moves have been completed). But $m(quant) \leq q + v(quant) \leq i*q$ (see (2)) → move s_{i*q+1} does not start before move a_i. On the other hand, $m(quant) \geq i*q$, because each compound move c_j (j = 0,...,i-1) contains exactly q s-moves.
→ m(quant) = q + v(quant) = i*q and so, move s_{i*q+1} is the one that will occur the next time v(quant) is incremented.
Move a_{i-1} has been completed; this can be derived from the assumption and part (a). Since t = 0, move a_{i+1} cannot start until at least q p-moves are completed. The next q p-moves are $p_{i*q+1},\ldots,p_{i*q+q}$, because s_{i*q+1} is the next s-move and $s_j < p_j$ for all j.

CONCLUSION

It has been shown that Parnas' solution to the Cigarette Smokers' Problem is deadlock free. The particular choice of three ingredients happens to be rather unfortunate, because it does not reveal anything about the distinguishable codewords that can be produced. A generalization of the problem showed that Parnas' solution also applies to the production of codewords of length q out of a set of n ingredients ($0 < q < n$). It was also found that the largest set of distinguishable codewords in a code with n ingredients has $\binom{n}{q}$ elements, where q = ENTIER(n/2) and so nothing is gained by letting the codeword length vary.

The last variation on the problem does not leap to the obscure conclusion that the ingredients to assemble a codeword have to be produced by one supplier. Instead, a supplier produces only one ingredient at a time and codewords must be assembled out of potentially concurrent suppliers and pushers. The variation has still the same basic difficulty as the original problem, although the supplier programs differ considerably in appearance from Patil's agent. It is interesting to notice that the solution is nevertheless very close to the one given by Parnas.

Throughout the paper the invariant equation that describes the working of Dijkstra's P- and V-operation has shown to be very useful. But the length and most of the phrasing of the proofs show that it is even hard to apply our current proving techniques to what seems rather obvious to the programmer.

ACKNOWLEDGMENT

I am grateful to D. Parnas for bringing his solution to my attention and to Anita Jones for discussing a first attempt to prove the correctness of Parnas' solution.

-* move a^^ causes moves [illegible] > • • • > i*q+ > [illegible] cause moves

i*q+1 • • • i*q+q [illegible] P [illegible] 8 [illegible] > •

c) It remains to be shown that moves i*q+i > • • • > .j*q.q produce different ingredients such that moves p.. [illegible] p [illegible] result in an increment of v(a).

A particular supplier(j) would t>a able to move twice after move a_^ started if it was waiting in P(quant) and move a^ is going to perform a V-operation on s[j]. The latter implies that supplier(j) moved in compound move ^. But until at least move a^ ^ each s-move was followed by a V-operation in an a-move and so m(s[j]) = 1 + v(s[j]) when move a^ starts. But then an attempt to supply ingredient[j] after a^ started cannot succeed until move a^ increments v(s[j]) because of

m(s[j]) = MIN(p(s[j]),1 + v(s[j]))

Hence, such a supplier[j] can only move once after move a^ started and consequently all supplier moves s^q+i > • • • i*q+q are different. Thus, the p-moves P ^ * ^ > • • • >Pi*q+q [illegible] on different indices and so exactly one codeword is the result or v(a) is incremented by one.

It is remarkable that this proof is not a copy of the correctness proof of the first generalization. This is mainly due to the fact that the first one is less complex, because producing all the ingredients for one codeword in one supplier dismisses the problem of proving that different ingredients are produced. However, parts a) and b) of this proof are very similar to parts of the correctness proof.

References

1. Dijkstra, E. W., Cooperating Sequential Processes, Report EWD 123, Technological University, Eindhoven, The Netherlands, 1965. Also published in Programming Languages, F. Genuys, ed., Academic Press, 1968.

2. Patil, S. S., Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination Among Processes, Project MAC, Computational Structures Group Memo 57, February 1971.

3. Project MAC Progress Report, 1970-1971.

4. Parnas, D. L., On a Solution to the Cigarette Smokers' Problem (without conditional statements), Department of Computer Science, Carnegie-Mellon University, July 1972.

5. Habermann, A. N., "Synchronization of Communicating Processes", CACM 15, March 1972.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Computer Science Department Carnegie-Mellon University Pittsburgh, Pa. 15213	UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

ON A SOLUTION AND A GENERALIZATION OF THE CIGARETTE SMOKERS' PROBLEM

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Scientific Final

5. AUTHOR(S) (First name, middle initial, last name)

Nico Habermann

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
August 1972	26	5

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
F44620-70-C-0107 b. PROJECT NO. 9769 c. 61102F d. 681304	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
TECH OTHER	Air Force Office of Scientific Rsch (NM) 1400 Wilson Blvd. Arlington, Va. 22209

13. ABSTRACT

The author of the Cigarette Smokers' Problem (S. Patil, MIT) tried to show that the problem could not be solved using Dijkstra's P- and V-operations without conditional statements. D. L. Parnas (CMU) showed that Patil's proof was false by presenting a solution using P- and V-operations, but no conditional statements. This paper presents first a correctness proof of Parnas' solution. This solution leads to an obvious generalization and in connection with it the question of an optimal set of distinguishable codewords is addressed. Finally a more natural approach to the problem is discussed and a solution for this variation is presented with its correctness proof.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6801

Security Classification