1972

On a solution and a generalization of the cigarette smokers' problem

A. Nico Habermann

Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/compsci
NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making
of photocopies or other reproductions of copyrighted material. Any copying of this
document without permission of its author may be prohibited by law.
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.
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:

\[
\text{process } \text{supplier}(i) = \\
\begin{align*}
\text{begin} & \sup: P(s); \text{V(ingr[(i+1)\%3])}; \text{V(ingr[(i+2)\%3])}; \\
\text{goto} & \sup \\
\text{end}
\end{align*}
\]

\(c \quad i = 0,1,2\)
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 $\text{ ingr} [0:2]$ 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:

\[
\text{process addict}(j) = \begin{cases} 
\text{ad} & \text{if } j = 0,1,2 \\
\text{begin ad: } & \text{P(ingr[(j+1)\%3])}; \\
& \text{P(ingr[(j+2)\%3])}; \\
& \text{V(s)}; \\
& \text{goto ad} \\
\end{cases}
\]

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) =
begin
pu: P(ingr[j]);
P(mutex);
t := t+2tj; V(a[t]);
V(mutex);
goto pu
end

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

process addict(k) =
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

\[ m(\text{sem}) = \text{MIN}(p(\text{sem}), c+v(\text{sem})) \]

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

Applied to this particular example, we have

\[
\begin{align*}
  m(s) &= \text{MIN}(p(s), 1+v(s)) \\
  m(\text{ingr}) &= \text{MIN}(p(\text{ingr}), v(\text{ingr})) \\
  m(a) &= \text{MIN}(p(a), v(a))
\end{align*}
\]

where "a" represents the semaphore set \( \{a[i] | i = 3,5,6\} \) and "ingr" = \( \{\text{ingr}[i] | i = 0,1,2\} \). Initially \( v(s) = v(\text{ingr}) = v(a) = 0 \). 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 \( p(s), p(\text{ingr}) \) and \( p(a) \) are maximal.

Let \( a_0 \) represent the action of starting the system.

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

**Proof.** Given the initial condition \( v(s) = v(\text{ingr}) = v(a) = 0 \), the system is started by allowing all processes an attempt to move, thus
\[ p(s) = p(\text{ingr}) = p(a) = 3. \]

\[ \rightarrow m(s) = \text{MIN}(3,1) = 1 \]
\[ m(\text{ingr}) = \text{MIN}(3,0) = 0 \]
\[ m(a) = \text{MIN}(3,0) = 0 \]

Hence, exactly one supplier will move.

Let \( s_i, p_i \) and \( a_i \) represent the \( i \)-th \( s \)-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 \( a_{i-1} \)

\[ v(s) = i-2 \]

\[ \rightarrow m(s) = \text{MIN}(p(s), 1+v(s)) = \text{MIN}(p(s), 1+i-2) = i-1 \]

\[ \text{after move } a_{i-1}, \ v(s) = i-1 \]

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 \( p(\text{ingr}) > 2i \)

before move \( s_i \)

\[ v(\text{ingr}) = 2(i-1) \]

\[ \rightarrow m(\text{ingr}) = \text{MIN}(p(\text{ingr}), v(\text{ingr})) = \text{MIN}(p(\text{ingr}), 2(i-1)) = 2(i-1) \]

\[ \text{after move } s_i, \ v(\text{ingr}) = 2i \]

\[ \rightarrow m(\text{ingr}) = \text{MIN}(p(\text{ingr}), 2i) = 2i \]

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) $a_1 < s_2$ (lemma 2) $\Rightarrow a_1 < p_3, p_4$ (lemma 3)

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:

$$a_{i-1} < s_i < p_{2i-1}, p_{2i}$$ (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 \( s_\ell \) \((\ell \in \{1, 2, \ldots, k\})\) is preceded by an \( a \)-move and so the variable \( t = 0 \) (see program), when move \( s_\ell \) starts.

b) \( s_j < p_{2j-1}, p_{2j} < a_j \) for \( j = 1, \ldots, k-1 \) 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, \( p(a) \geq k \).

move \( a_k \) can be performed \( \text{iff} \) \( \text{MIN}(p(a), v(a)) = k \)

- 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 \( i = 1, 2, \ldots \) \((w_i \) and \( w_{i+1} \) are performed strictly in sequence). 

**Proof.**

a) It is sufficient to prove \( w_i < a_i \) for \( i = 1, 2, \ldots \) because \( a_i \)
is the first move of \( w_{i+1} \) (see definition) and so relation
\[
\text{w}_i < a_i \text{ implies } \text{w}_i < \text{w}_{i+1}.
\]

b) (induction)

\[
\text{w}_1 < a_i \text{ is true (lemma 4)}
\]
Suppose \( \text{w}_i < a_i \) for \( i = 1, \ldots, k-1 \).
It remains to be proved that this implies that \( \text{w}_i < a_k \) is also true.

The assumption implies \( \text{w}_1 < \text{w}_2 < \ldots < \text{w}_k \), i.e., \( \text{w}_1, \ldots, \text{w}_k \) is performed strictly in sequence. But this sequence causes move \( a_k \) (lemma 5) and so \( \text{w}_k < a_k \).

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:

\[
\text{process } \text{supplier}(i) = \quad c \ i = 0,1,2,\ldots,n-1
\begin{align*}
\text{begin integer } & \ell; \\
\text{sup: } & P(s); \\
\text{for } & \ell := 0 \text{ until } i-1, i+1 \text{ until } n-1 \text{ do } V(\text{ingr}[\ell]); \\
\text{goto sup} \\
\text{end}
\end{align*}
\]

Note that the use of the artificial for statement does not really violate
the restriction that no conditional statements may be used. Variable \( \ell \)
is a local, not operated upon by any other process than \text{supplier}(i).
The notation describes that \text{supplier}(i) performs a \( V \)-operation on all
ingredient semaphores but \( \text{ingr}[i] \). 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 \( \text{ingr}[0:n-1] \) 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 \( j = 0,\ldots,n-1 \) and of the addicts:

\[
\{ k \mid k = 2^t(n+1) - 1 - 2^t, i = 0,\ldots,n-1 \}
\]
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 pro-
duced 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 \( A, B, \ldots \) be the codewords produced by the suppliers; ingredient \( a \in A \) distinguishes codeword \( A \) properly from codeword \( B \) if \( a \notin B \), but \( A - \{a\} \subseteq B \). Since \( B \subseteq A \) 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 \( q = \text{ENTIER}(n/2) \), \( n \geq 1 \).

**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 \( \Gamma \) be the codeset. If \( \Gamma \) consists of signal \( a \) only, there is only one distinguishable set, viz. \( A = \{a\} \) and \( x(A) = (1) \). If \( \Gamma = \{a,b\} \), 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 \( a \in A \) a (distinguishing) signal. Signal \( a \) distinguishes codeword \( A \) from a codeword \( B \), i.e., there is a codeword \( B \) and a signal \( b \in B \) such that \( a \notin B, \ b \notin A \) and \( A - \{a\} \subseteq B \). The codeset \( \Gamma \) can be ordered such that signals \( a \) and \( b \) are the first two elements: \( \Gamma = \{a,b,...\} \). 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 \( \Gamma^* \) be the codeset derived from \( \Gamma \) by deleting signals \( a \) and \( b \). The remainder of codeword \( A \) must be a
distinguishable codeword in code $\Gamma^*$ with $n^* = n-2$.

If $n^* = 1$ then $A^* = \Gamma^* = \{c\}$ and $x(A) = (101)$.

If $n^* = 2$ then $x(A^*) = (10)$ after ordering $\Gamma^*$ properly and so
$x(A) = (1010)$.

If $n^* > 2$ then $x(A^*) = (10\ldots)$ after ordering $\Gamma^*$ and so $x(A) = (1010\ldots)$. 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

$$x(A) = (100), \quad x(B) = (010), \quad x(C) = (001)$$

or

$$x(A) = (011), \quad x(B) = (101), \quad x(C) = (110)$$

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 pat­
terns with three ones and three zeros. This allows for a set of \( \binom{6}{3} = 20 \)
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 \( m = \binom{n}{q} \) sup­
pliers, \( n \) 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 \( s_i \) causes moves \( p(i-1)q+1, p(i-1)q+2, \ldots, p_i \), and 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 2tj 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 \( z = 2t^{n-1}q(n-q) \). 

- **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. 

\[ \textbf{integer} \ t \text{ is a} \]
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.

\[ \text{process } \text{addict}(k) = \sum_{i=0}^{n-1} c_i 2^i, c_i = 0 \text{ or } 1, \sum_{i=0}^{n-1} c_i = q \]

\begin{verbatim}
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
\end{verbatim}

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

\[
\begin{align*}
m(s) &= \text{MIN}(p(s), n+v(s)) \quad (1) \\
m(\text{quant}) &= \text{MIN}(p(\text{quant}), q+v(\text{quant})) \quad (2) \\
m(\text{ingr}) &= \text{MIN}(p(\text{ingr}), v(\text{ingr})) \quad (3) \\
m(a) &= \text{MIN}(p(a), v(a)) \quad (4) 
\end{align*}
\]

The initial value of \( v(s), v(\text{quant}), v(\text{ingr}) \) 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 \( v(\text{quant}) \) is incremented in move \( a_i \). \( a_i \leq s_{q+1} \)

c) according to (3) a pusher will move when a supplier performs \( v(\text{ingr}) \), 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 \( a_1 \) 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.

\[
\begin{align*}
\text{move } x & \quad \text{move } y \\
\text{move } y & \quad \text{move } x
\end{align*}
\]

Let \( c_i = \{ s_k, p_k \mid s_k < p_k, k = iq+1, \ldots, iq+q \} \), for \( i = 0, 1, \ldots \).
Thus, \( c_i \) represents the \( i \)-th set of \( q \) s-moves and \( q \) p-moves. This set is
not ordered, but its first element is \( s_{iq+1} \) 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 \( a_0 \leq c_0 < a_1 \leq c_1 < a_2 \leq c_2 < \ldots \).

**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
\( a_0 \leq c_0 < a_1 \ldots a_{i-1} < c_{i-1} \) and that \( c_j \) results in incrementing \( v(a) \) by one
for \( j = 0, 1, \ldots, i-1 \). 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 \( \rightarrow v(a) = i \). Hence, \( m(a) = \text{MIN}(p(a), v(a)) = \text{MIN}(p(a), i) = i \) 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

\[
s_{i\times q+1}, \ldots, s_{i\times q+q}
\]
which cause in turn p-moves

\[
p_{i\times q+1}, \ldots, p_{i\times q+q}
\]
When variable \( t \) is reset to zero in move \( a_i \), \( v(\text{quant}) = (i-1)q \) (for, \( i-1 \) a-moves have been completed).

But \( m(\text{quant}) \leq q + v(\text{quant}) \leq i\times q \) (see (2)) \( \rightarrow \) move \( s_{i\times q+1} \) does not start before move \( a_i \). On the other hand, \( m(\text{quant}) \geq i\times q \), because each compound move \( c_j \) \( (j = 0, \ldots, i-1) \) contains exactly \( q \) s-moves.

\( \rightarrow m(\text{quant}) = q + v(\text{quant}) = i\times q \) and so, move \( s_{i\times q+1} \) is the one that will occur the next time \( v(\text{quant}) \) 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

\[
p_{i\times q+1}, \ldots, p_{i\times q+q}
\]
because \( s_{i\times q+1} \) 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 \( q = \text{ENTIER}(n/2) \) 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 \( s_{i^*q+1} \), \( s_{i^*q+2} \), \( s_{i^*q+q} \), which cause moves \( p_{i^*q+1} \), \( p_{i^*q+2} \), \( p_{i^*q+q} \) (see programs).

c) It remains to be shown that moves \( s_{i^*q+1} \), \( s_{i^*q+2} \), \( s_{i^*q+q} \) produce different ingredients such that moves \( p_{i^*q+1} \), \( p_{i^*q+2} \), \( p_{i^*q+q} \) result in an increment of \( v(a) \).

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

\[
m(s[j]) = \text{MIN}(p(s[j]), 1 + v(s[j]))
\]

Hence, such a supplier \( (j) \) can only move once after move \( a_i \) started and consequently all supplier moves \( s_{i^*q+1} \), \( s_{i^*q+2} \), \( s_{i^*q+q} \) are different. Thus, the \( p \)-moves \( p_{i^*q+1} \), \( p_{i^*q+2} \), \( p_{i^*q+q} \) are also 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 first correctness proof.
References


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

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.