some make nonsense. We further assume that all possible sequences of the *amino acids* may occur (that is, can be coded) and that at every point in the string of letters one can only read "sense" in the correct way. This is illustrated in Figure 3. In other words, any two triplets which make sense can be put side by side, and yet the overlapping triplets so formed must always be nonsense.

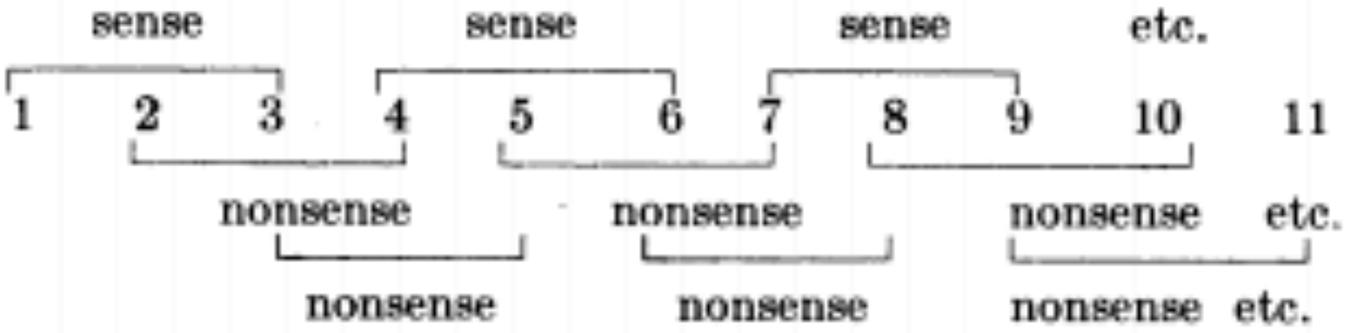


FIG. 3.—The numbers represent the positions occupied by the four letters *A*, *B*, *C*, and *D*. It is shown which triplets make sense and which nonsense.

It is obvious that with these restrictions one will be unable to code 64 different amino acids. The mathematical problem is to find the maximum number that can be coded. We shall show (1) that the maximum number cannot be greater than 20 and (2) that a solution for 20 can be given.

To prove the first point, we consider for the moment the restrictions imposed by placing each amino acid next to itself. Then, clearly, the triplet *AAA* must be nonsense, since, if it corresponded to an amino acid, α, then $\alpha\alpha$ would be *AAAAAA*, and this sequence can be misinterpreted by associating α with the second to fourth, or third to fifth, letters. We can thus reject *AAA*, *BBB*, *CCC*, and *DDD*.

It is easy to see that the 60 remaining triplets can be grouped into 20 sets of three, each set of three being cyclic permutations of one another. Consider as an example *ABC* and its cyclic permutations *BCA* and *CAB*. It is clear that we can choose any one of these, but not more than one. For suppose that we let *BCA* stand for the amino acid β; then $\beta\beta$ is *BCABCA*, and so *CAB* and *ABC* must, by our rules, be nonsense. Since we can choose at the most one triplet from each cyclic set, we cannot choose more than 20. No solution is possible, therefore, which codes more than 20 different amino acids.

```
"Not enough selections taking nucleotides 2 at a time to cover the
aminos but more than enough taken 3 at a time. Call them triples."

(def triples (map string (for [x ACGT y ACGT z ACGT] [x y z])))
triples          ; => ["AAA" "AAC" "AAG" "AAT" "ACA" "ACC" "ACG" "ACT"
;;                     "AGA" "AGC" "AGG" "AGT" "ATA" "ATC" "ATG" "ATT"
;;                     "CAA" "CAC" "CAG" "CAT" "CCA" "CCC" "CCG" "CCT"
;;                     "CGA" "CGC" "CGG" "CGT" "CTA" "CTC" "CTG" "CTT"
;;                     "GAA" "GAC" "GAG" "GAT" "GCA" "GCC" "GCG" "GCT"
;;                     "GGA" "GGC" "GGG" "GGT" "GTA" "GTC" "GTG" "GTT"
;;                     "TAA" "TAC" "TAG" "TAT" "TCA" "TCC" "TCG" "TCT"
;;                     "TGA" "TGC" "TGG" "TGT" "TTA" "TTC" "TTG" "TTT"]

(def rotations
  (fn [s] (let [n (count s)]
            (map string (take n (partition n 1 (cycle s)))))))

(rotations ACGT)                          ; => ("ACGT" "CGTA" "GTAC" "TACG")

(take 7 (map rotations triples))          ; => (("AAA" "AAA" "AAA")
                                          ;     ("AAC" "ACA" "CAA")
                                          ;     ("AAG" "AGA" "GAA")
                                          ;     ("AAT" "ATA" "TAA")
                                          ;     ("ACA" "CAA" "AAC")
                                          ;     ("ACC" "CCA" "CAC")
                                          ;     ("ACG" "CGA" "GAC"))

(set (str ACGT ACGT ACGT))                ; => #{\A \C \G \T}
[((set ACGT) \T) ((set ACGT) \Z)]         ; => [\T nil]

(def codons (set (map (comp set rotations) triples)))
(count codons)                            ; => 24
(take 4 codons)                           ; => (#{"ACC" "CCA" "CAC"}
                                          ;     #{"GGG"}
                                          ;     #{"TTT"}
                                          ;     #{"GCC" "CGC" "CCG"})

(map first (group-by count codons))       ; => (3 1)

(def sense-codons ((group-by count codons) 3))
(count sense-codons)                      ; => 20
```

Eureka!

BROAD
INSTITUTE

商理

3

CODES WITHOUT COMMAS

This is fun programming but (as of 1961) bad biology.

**Biology is not physics and certainly not engineering.
Life's processes are rarely efficient and often messy.
(Crick and Gamow were both degreed in physics.)**

What is this program about? Can you follow it?

**This programming language is as old as this paper.
It was designed to help machines help us think.**

**It is almost the simplest language that can work.
So the program is about the world more than itself.
A programming language can be a tool of discovery
and not just the means of commanding a computer.**

**You can develop and explore a program while it runs.
And see the value of each expression as it's evaluated.
They appear here literally as the computer prints them.
("Notebook systems" like Jupyter return to this idea.)**

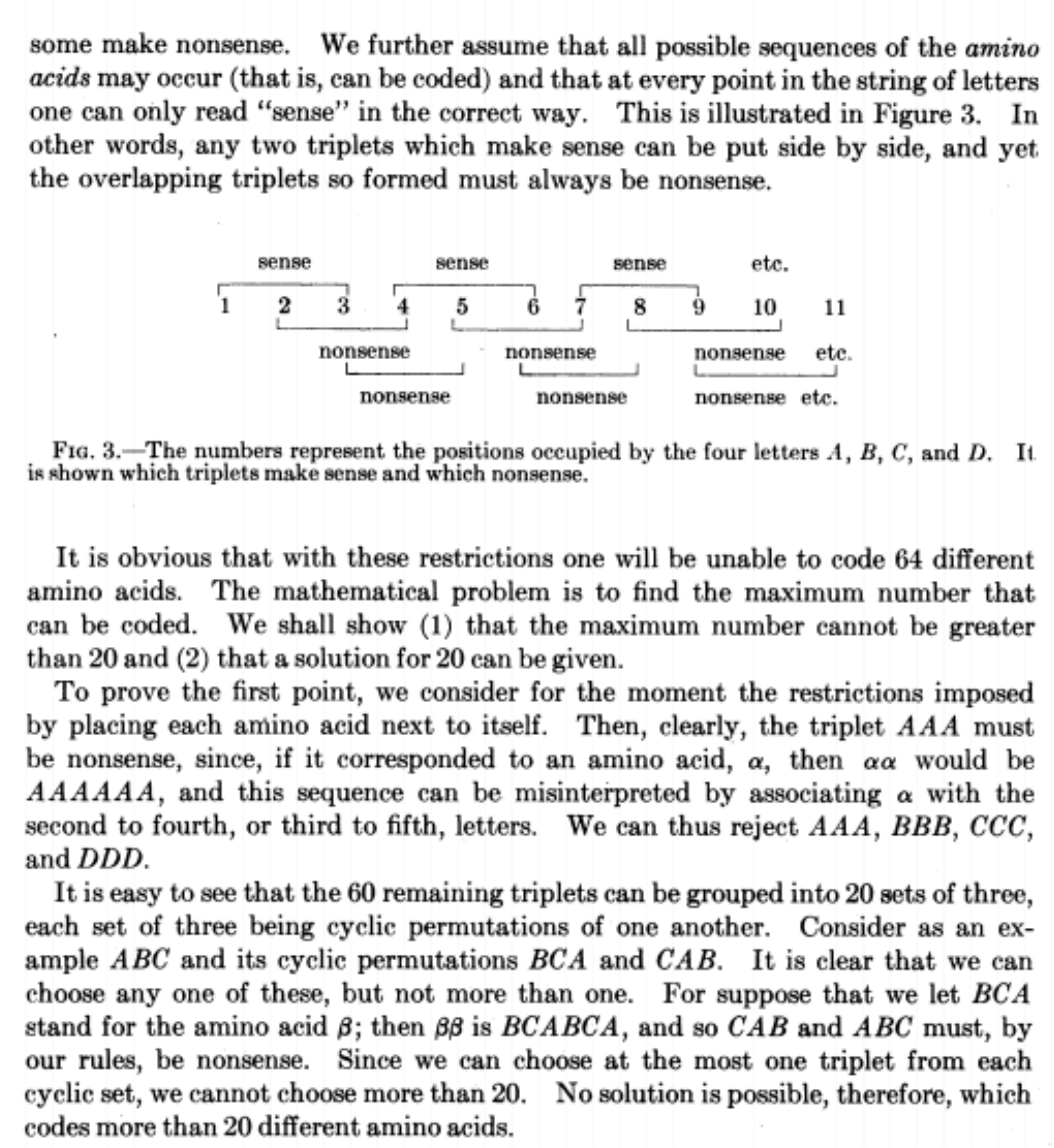
**How do you think with your programming language?
Does it aid or hinder your understanding of the world?**

(-: BTW: Did you notice any commas in this code? :-)

```
(take 5 sense-codons)                   ; => (#{"ACC" "CCA" "CAC"}
                                        ;      #{"GCC" "CGC" "CCG"}
                                        ;      #{"CAA" "ACA" "AAC"}
                                        ;      #{"CTC" "CCT" "TCC"}
                                        ;      #{"AGC" "CAG" "GCA"})

(def sense (map first sense-codons))
(count sense)   ; => 20
(take 7 sense)  ; => ("ACC" "GCC" "CAA" "CTC" "AGC" "TAT" "GAG")
(def nonsense (remove (set sense) (set triples)))
(count nonsense)                        ; => 44

(def code (zipmap sense (keys amino-acids)))
(sort-by second code)                   ; => (["AGC" :A]
                                        ;      ["TAA" :C]
                                        ;      ["TGT" :D]
                                        ;      ["TCA" :E]
                                        ;      ["TAT" :F]
                                        ;      ["TAG" :G]
                                        ;      ["CTG" :H]
                                        ;      ["CAA" :I]
                                        ;      ["ACG" :K]
                                        ;      ["ACC" :L]
                                        ;      ["GCC" :M]
                                        ;      ["GAA" :N]
                                        ;      ["CTA" :P]
                                        ;      ["GAT" :Q]
                                        ;      ["CTC" :R]
                                        ;      ["TCG" :S]
                                        ;      ["CTT" :T]
                                        ;      ["GTG" :V]
                                        ;      ["GAG" :W]
                                        ;      ["GGC" :Y])

(map string (take 3 (partition 3 1 strand))) ; => ("TTA" "TAA" "AAT")

(def amino-keys (map (comp code string) (partition 3 1 strand)))
(take 13 amino-keys) ; => (nil :C nil nil nil :Y nil nil :M nil nil :Y :M)

(def aminos (map amino-acids (remove nil? amino-keys)))
(take 4 aminos) ;=> ("cysteine" "tyrosine" "methionine" "tyrosine")
```

BROAD INSTITUTE

CODES WITHOUT COMMAS



some make nonsense. We further assume that all possible sequences of the *amino acids* may occur (that is, can be coded) and that at every point in the string of letters one can only read "sense" in the correct way. This is illustrated in Figure 3. In other words, any two triplets which make sense can be put side by side, and yet the overlapping triplets so formed must always be nonsense.

FIG. 3.—The numbers represent the positions occupied by the four letters *A*, *B*, *C*, and *D*. It is shown which triplets make sense and which nonsense.

It is obvious that with these restrictions one will be unable to code 64 different amino acids. The mathematical problem is to find the maximum number that can be coded. We shall show (1) that the maximum number cannot be greater than 20 and (2) that a solution for 20 can be given.

To prove the first point, we consider for the moment the restrictions imposed by placing each amino acid next to itself. Then, clearly, the triplet *AAA* must be nonsense, since, if it corresponded to an amino acid, α, then $\alpha\alpha$ would be *AAAAAA*, and this sequence can be misinterpreted by associating α with the second to fourth, or third to fifth, letters. We can thus reject *AAA*, *BBB*, *CCC*, and *DDD*.

It is easy to see that the 60 remaining triplets can be grouped into 20 sets of three, each set of three being cyclic permutations of one another. Consider as an example *ABC* and its cyclic permutations *BCA* and *CAB*. It is clear that we can choose any one of these, but not more than one. For suppose that we let *BCA* stand for the amino acid β; then $\beta\beta$ is *BCABCA*, and so *CAB* and *ABC* must, by our rules, be nonsense. Since we can choose at the most one triplet from each cyclic set, we cannot choose more than 20. No solution is possible, therefore, which codes more than 20 different amino acids.

```
"Not enough selections taking nucleotides 2 at a time to cover the
aminos but more than enough taken 3 at a time. Call them triples."

(def triples (map string (for [x ACGT y ACGT z ACGT] [x y z])))
triples         ; => ["AAA" "AAC" "AAG" "AAT" "ACA" "ACC" "ACG" "ACT"
;;                    "AGA" "AGC" "AGG" "AGT" "ATA" "ATC" "ATG" "ATT"
;;                    "CAA" "CAC" "CAG" "CAT" "CCA" "CCC" "CCG" "CCT"
;;                    "CGA" "CGC" "CGG" "CGT" "CTA" "CTC" "CTG" "CTT"
;;                    "GAA" "GAC" "GAG" "GAT" "GCA" "GCC" "GCG" "GCT"
;;                    "GGA" "GGC" "GGG" "GGT" "GTA" "GTC" "GTG" "GTT"
;;                    "TAA" "TAC" "TAG" "TAT" "TCA" "TCC" "TCG" "TCT"
;;                    "TGA" "TGC" "TGG" "TGT" "TTA" "TTC" "TTG" "TTT"]

(def rotations
  (fn [s] (let [n (count s)]
            (map string (take n (partition n 1 (cycle s)))))))

(rotations ACGT)                        ; => ("ACGT" "CGTA" "GTAC" "TACG")

(take 7 (map rotations triples))        ; => (("AAA" "AAA" "AAA")
                                        ;     ("AAC" "ACA" "CAA")
                                        ;     ("AAG" "AGA" "GAA")
                                        ;     ("AAT" "ATA" "TAA")
                                        ;     ("ACA" "CAA" "AAC")
                                        ;     ("ACC" "CCA" "CAC")
                                        ;     ("ACG" "CGA" "GAC"))

(set (str ACGT ACGT ACGT))              ; => #{\A \C \G \T}
[((set ACGT) \T) ((set ACGT) \Z)]       ; => [\T nil]

(def codons (set (map (comp set rotations) triples)))
(count codons)                          ; => 24
(take 4 codons)                         ; => (#{"ACC" "CCA" "CAC"}
                                        ;     #{"GGG"}
                                        ;     #{"TTT"}
                                        ;     #{"GCC" "CGC" "CCG"})

(map first (group-by count codons))     ; => (3 1)

(def sense-codons ((group-by count codons) 3))
(count sense-codons)                    ; => 20
```

Eureka!