



# CODES WITHOUT COMMAS

By F. H. C. CRICK, J. S. GRIFFITH, AND L. E. ORGEL

MEDICAL RESEARCH COUNCIL UNIT, CAVENDISH LABORATORY, AND DEPARTMENT OF THEORETICAL  
CHEMISTRY, CAMBRIDGE, ENGLAND

*Communicated by G. Gamow, February 11, 1957*

```
(ns commas
  "CODES WITHOUT COMMAS -- with apologies to F.H.C. Crick, et al"
  (:require [clojure.math.combinatorics :as c]))

"The problem of how a sequence of four things (nucleotides) can
determine a sequence of twenty things (amino acids) is known as
the 'coding' problem."

(comment "http://www.pnas.org/content/43/5/416" is the paper.)

(comment Now is 1957.  What do we know?)

(def nucleotides ["Adenine" "Cytosine" "Guanine" "Thymine"])
nucleotides ;=> ["Adenine" "Cytosine" "Guanine" "Thymine"]

(def ACGT (map first nucleotides))
ACGT ;=> (\A \C \G \T)

(reverse ACGT) ;=> (\T \G \C \A)

(def pair (zipmap ACGT (reverse ACGT)))
pair ;=> {\A \T \C \G \G \C \T \A}

(pair \T) ;=> \A

(rand-nth ACGT) ;=> \G
(rand-nth ACGT) ;=> \C

(def strand (repeatedly (fn [] (rand-nth ACGT))))

(take 7 strand) ;=> (\G \G \C \T \G \C \C)

(def dna (map (fn [b] [b (pair b)]) strand))

(take 6 dna) ;=> ([\G \C] [\C \G] [\T \A] [\T \A] [\A \T] [\A \T])
(take 7 (map first dna)) ;=> (\G \C \T \T \A \A \G)
(take 7 (map second dna)) ;=> (\C \G \A \A \T \T \C)

(def essential {:F "phenylalanine"
               :H "histidine"
               :I "isoleucine"
               :K "lysine"
               :L "leucine"
               :M "methionine"
               :T "threonine"
               :V "valine"
               :W "tryptophan"})

(def conditional {:C "cysteine"
                 :G "glycine"
                 :P "proline"
                 :Q "glutamine"
                 :R "arginine"
                 :Y "tyrosine"})

(def dispensible {:A "alanine"
                  :D "aspartic acid"
                  :E "glutamic acid"
                  :N "asparagine"
                  :S "serine"})

(def amino (merge essential conditional dispensible))

(count amino) ;=> 20

(comment Now ... What can we find out?)
```

```
(c/selections ACGT 2) ;=> ((\A \A)
                          ; (\A \C)
                          ; (\A \G)
                          ; (\A \T)
                          ; (\C \A)
                          ; (\C \C)
                          ; ...
                          ; (\G \G)
                          ; (\G \T)
                          ; (\T \A)
                          ; (\T \C)
                          ; (\T \G)
                          ; (\T \T))

(count (c/selections ACGT 2)) ;=> 16
(count (c/selections ACGT 3)) ;=> 64
(> (count (c/selections ACGT 2)) (count amino)) ;=> false
(> (count (c/selections ACGT 3)) (count amino)) ;=> true

(def string (partial apply str))

(def triples (map string (c/selections ACGT 3)))

(take 7 triples) ;=> ("AAA" "AAC" "AAG" "AAT" "ACA" "ACC" "ACG")
(take 7 (reverse triples)) ;=> ("TTT" "TTG" "TTC" "TTA" "TGT" "TGG" "TGC")
(count triples) ;=> 64

(take (count ACGT) (partition (count ACGT) 1 (cycle ACGT)))
;;=> ((\A \C \G \T) (\C \G \T \A) (\G \T \A \C) (\T \A \C \G))

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

(rotations ACGT) ;=> ("ACGT" "CGTA" "GTAC" "TACG")
(set ACGT) ;=> #{\A \C \G \T}
((set ACGT) \T) ;=> \T
((set ACGT) \Z) ;=> nil
(rotations (take 3 ACGT)) ;=> ("ACG" "CGA" "GAC")

(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"))

(def codons (set (map (comp set rotations) triples)))

(count codons) ;=> 24
(take 7 codons) ;=> ({#"ACC" "CCA" "CAC"}
                    ; {"GGG"}
                    ; {"TTT"}
                    ; {"GCC" "CGC" "CCG"}
                    ; {"CAA" "ACA" "AAC"}
                    ; {"CTC" "CCT" "TCC"}
                    ; {"AGC" "CAG" "GCA"})

(count (group-by count codons)) ;=> 2
(map first (group-by count codons)) ;=> (3 1)

(def sense-codons (filter (fn [g] (= 3 (count g))) codons))

(count sense-codons) ;=> 20 Eureka!
```

```
(take 7 sense-codons) ;=> ({#"ACC" "CCA" "CAC"}
                          ; {"GCC" "CGC" "CCG"}
                          ; {"CAA" "ACA" "AAC"}
                          ; {"CTC" "CCT" "TCC"}
                          ; {"AGC" "CAG" "GCA"}
                          ; {"TAT" "TTA" "ATT"}
                          ; {"GAG" "GGA" "AGG"})

(def sense (map (comp first sort) sense-codons))

(count sense) ;=> 20

(take 7 sense) ;=> ("ACC" "CCG" "AAC" "CCT" "AGC" "ATT" "AGG")

(def nonsense (remove (set sense) (set triples)))

(count nonsense) ;=> 44

(= (count triples)
   (+ (count sense)
      (count nonsense))) ;=> true

(def code (zipmap (map vec (sort sense)) (sort (keys amino)))))

(sort-by second code) ;=> ([[\A \A \C] :A]
                          ; [[\A \A \G] :C]
                          ; [[\A \A \T] :D]
                          ; [[\A \C \C] :E]
                          ; [[\A \C \G] :F]
                          ; [[\A \C \T] :G]
                          ; ...
                          ; [[\C \G \G] :R]
                          ; [[\C \G \T] :S]
                          ; [[\C \T \G] :T]
                          ; [[\C \T \T] :V]
                          ; [[\G \G \T] :W]
                          ; [[\G \T \T] :Y])

(map string (take 7 (partition 3 1 strand)))
;;=> ("TTC" "TCG" "CGG" "GGT" "GTG" "TGA" "GAT")

(def amino-keys (remove nil? (map code (partition 3 1 strand))))

(take 7 amino-keys) ;=> (:R :W :M :W :G :T :D)

(def aminos (map amino amino-keys))

(take 17 aminos) ;=> ("arginine"
                    ; "tryptophan"
                    ; "methionine"
                    ; "tryptophan"
                    ; "glycine"
                    ; "threonine"
                    ; "aspartic acid"
                    ; "alanine"
                    ; "glutamic acid"
                    ; "alanine"
                    ; "glycine"
                    ; "valine"
                    ; "asparagine"
                    ; "alanine"
                    ; "glutamic acid"
                    ; "glutamine"
                    ; "glutamine")

"Note: This is fun programming but (as of 1961) bad biology."
```