```clojure
(ns n-gram.words.word-maker (:require [n-gram.misc.misc-functions :refer :all]))

(defn split-words [text] "Vector of all words in text" (clojure.string/split text
#"\s+"))

(def split-words-memo "Memoized split-words" (memoize split-words))

(def unknown "<unk>")

(defn replace-first-word [text distinct-words] "Replaces first occurrence of each word
in text with <unk>"
  (if (< 0 (count distinct-words))
            (clojure.string/replace-first (replace-first-word text (rest distinct-
words))
                                          (re-pattern (str " " (first distinct-words) "
")) (str " " unknown" ")) text))

(def replace-first-word-memo "Memoize replace-first-word" (memoize replace-first-word))

(defn make-words "Creates a sequence of all the words from the input"
  [theWords] (if (> (count theWords) 1)
     (cons(take 1 theWords)(lazy-seq(make-words (rest theWords))))
     (cons (take 1 theWords) "")))

(def make-words-memo "Memoized make-words" (memoize make-words))

(defn make-pairs "Creates a sequence of all the word pairs from the input"
  [theWords] (if (> (count theWords) 2)
     (cons(take 2 theWords)(lazy-seq(make-pairs (rest theWords))))
     (cons (take 2 theWords) "")))

(def make-pairs-memo "Memoized make-pairs" (memoize make-pairs))

(defn make-trios "Creates a sequence of the word trios from the input"
  [theWords] (if (> (count theWords) 3)
     (cons(take 3 theWords)(lazy-seq(make-trios (rest theWords))))
     (cons (take 3 theWords) "")))

(def make-trios-memo "Memoized make-trios" (memoize make-trios))

(defn make-4s "Creates a sequence of the word 4s from the input"
  [theWords] (if (> (count theWords) 4)
     (cons(take 4 theWords)(lazy-seq(make-4s (rest theWords))))
     (cons (take 4 theWords) "")))

(def make-4s-memo "Memoized make-4s" (memoize make-4s))
```