;; [問題] あるお百姓さんが、大八車を引きながら村はずれの川の西の岸にやってきた
;; 大八車には、荒縄でくくられたキャベツと背負い駕篭に入ったニワトリが積まれ
;; ていた。明日は隣村に市が立つので、その市でキャベツとニワトリを売ろうとい
;; うわけである。その後を、お百姓さんにすっかりなついてしまったキツネのコン
;; 太が、いかにもニワトリを食べたそうなそぶりて、着いて来ていた。
;; さて、川のほとりには一艘の小さな船がつないであった。この船で、お百姓さん
;; はニワトリ、キャベツ、キツネのすべてを川の東の岸へ運ばなければならない。
;; 困ったことに、この船は大変小さかったので、船をあやつるお百姓さん以外に
;; 一度に運べるのは、ニワトリ、キャベツ、キツネのうちのどれか一つだけだった。
;; ところが、腹のへったニワトリは、お百姓さんがいなくなるとキャベツを食べて
;; しまい、さらにキツネのコン太はお百姓さんが目を離すと、ニワトリを食べてし
;; まう。
;; どうやったら、お百姓さんは首尾良くニワトリ、キャベツ、キツネのすべてを東
;; 側の岸に運べるだろうか？

;; FARMER+HEN.SCM BY KAZUO AMANO
;; EAST-SIDE-STATE IS REPRESENTED BY LIST $(w\ x\ y\ z)$ WHERE EACH $w, x, y, z = 0$ OR 1
;; EXAMPLE:

$(1\ 1\ 1\ 1)$ = (farmer hen cabbage fox) − state
$(0\ 1\ 0\ 1)$ = (none hen none fox) − state
$(1\ 0\ 1\ 0)$ = (farmer none cabbage none) − state

;; INITIAL-STATE = $(1\ 1\ 1\ 1)$
;; STATE-SEQUENCE = (... STATE2 STATE1 INITIAL-STATE)
;; STATE-TREE = (... STATE-SEQUECE2 STATE-SEQUENCE1 STATE-SEQUENCE0)

```
;  西から東への移動
(define (west->east x seq)
  (let* ((y (car seq)) (fa (car y)) (he (cadr y)) (ca (caddr y)) (fo (cadddr y)))
    (cond ((= fa 0) (cons '() seq))
      (else (cond ((and (equal? x 'hen) (= he 1)) (cons (list 0 0 ca fo) seq))
                  ((and (equal? x 'cabbage) (= ca 1)) (cons (list 0 he 0 fo) seq))
                  ((and (equal? x 'fox) (= fo 1)) (cons (list 0 he ca 0) seq))
                  (else (cons '() seq)))))))

;  東から西への移動
(define (west<-east x seq)
  (let* ((y (car seq)) (fa (car y)) (he (cadr y)) (ca (caddr y)) (fo (cadddr y)))
    (cond ((= fa 1) (cons '() seq))
      (else (cond ((and (equal? x 'hen) (= he 0)) (cons (list 1 1 ca fo) seq))
                  ((and (equal? x 'cabbage) (= ca 0)) (cons (list 1 he 1 fo) seq))
                  ((and (equal? x 'fox) (= fo 0)) (cons (list 1 he ca 1) seq))
                  ((equal? x 'none) (cons (list 1 he ca fo) seq))
                  (else (cons '() seq)))))))
```

```
;  終了の判定をする関数
(define (finished? tree)
  (let finished1? ((x tree))
    (cond ((null? x) #f)
      ((equal? (caar x) '(0 0 0 0)) #t)
      (else (finished1? (cdr x))))))

;  不適当な branch を切り落とす関数
(define (rm-bad-seq tree)
  (let rm-bad-seq1 ((x tree) (y '()))
    (cond ((null? x) y)
      ((null? (caar x)) (rm-bad-seq1 (cdr x) y))
      ((equal? (caar x) '(1 0 1 0)) (rm-bad-seq1 (cdr x) y))
      ((equal? (caar x) '(1 0 0 1)) (rm-bad-seq1 (cdr x) y))
      ((equal? (caar x) '(0 1 1 0)) (rm-bad-seq1 (cdr x) y))
      ((equal? (caar x) '(0 1 0 1)) (rm-bad-seq1 (cdr x) y))
      ((equal? (caar x) '(1 1 1 1)) (rm-bad-seq1 (cdr x) y))
      (else (rm-bad-seq1 (cdr x) (cons (car x) y))))))

;  branch を成長させる関数
(define (mkseq seq)
    (rm-bad-seq
     (list (west->east 'hen seq) (west->east 'cabbage seq) (west->east 'fox seq)
       (west<-east 'hen seq) (west<-east 'cabbage seq) (west<-east 'fox seq)
       (west<-east 'none seq))))

;  手順の tree を作る関数
(define (mktree initial-state)
  (let mktree1 ((x (list (list initial-state))))
    (cond ((finished? x) x)
      (else (mktree1 (let mktree2 ((x1 x) (x2 '()))
               (cond ((null? x1) x2)
                 (else (mktree2 (cdr x1) (append x2 (mkseq (car x1))))))))))))

;  main 関数
(define (main-func)
  (let display-answer ((x (mktree '(1 1 1 1))))
    (cond ((null? x) (newline))
      (else (cond ((equal? (caar x) '(0 0 0 0)) (display (car x)) (newline)))
        (display-answer (cdr x))))))
```