;gnu clisp 2.49.60

;строим автомат

(defun CrAut (l res) (cond

((null l) (reverse res))

(T ( CrAut (cdr l) (add\_1 (caaar l) (caddar l) res (cons (caaar l) nil))))))

(defun add\_1(st l res save) (cond

((null l) (add\_2 res save))

((eq (car l) 'l) (add\_1 st (cdr l) (add\_2 res save) (cons st nil)))

(T (add\_1 st (cdr l) res (cons (car l) save)))))

(defun add\_2(res save)(cond

((= (length save) 2) (cons (reverse (cons 'S save)) res ))

(T (cons (reverse save) res))))

; построили

;строим автомат

(defun CrAut\_2 (l res) (cond

((null l) res)

(T ( CrAut\_2 (cdr l) (add\_11 (caaar l) (caddar l) res (cons (caaar l) nil))))))

(defun add\_11(st l res save)(cond

((null l) res)

((eq (car l) 'l) (add\_11 st (cdr l) res (cons st nil)))

((characterp (car l)) (add\_11 st (cdr l) (add\_22 res (cons 'H (cons (car l) save))) (cons st nil)))

(T (add\_11 st (cddr l) (add\_22 res (cons (car l) (cons (cadr l) save))) (cons st nil)))))

(defun add\_22(res save) (cons save res ))

; построили

; теперь детерминируем

; все q для одного p и c

(defun f3(p letter Aut res)

(cond

((null Aut) res)

((and (eq (caar Aut) p) (eq (cadar Aut) letter) (eq (member (caddar Aut) res) nil)) (f3 p letter (cdr Aut) (cons (caddar Aut) res)))

(T (f3 p letter (cdr Aut) res))))

;все q для всех p и одного c

(defun f2(pq letter Aut res)

(cond

((null pq) res)

(T (f2 (cdr pq) letter Aut (f3 (car pq) letter Aut res)))))

; вспомогательная

(defun unt (l m s) (cons l(cons m (cons s nil))))

;все q для всех p и всех c

(defun d2(pq letters Aut res func)

(cond

((null (cdr letters))

(cond

((null func) res)

(T (cons(unt pq (car letters) func) res))))

((null func) (d2 pq (cdr letters) Aut res (f2 pq (cadr letters) Aut nil)))

(T (d2 pq (cdr letters) Aut (cons(unt pq (car letters) func) res) (f2 pq (cadr letters) Aut nil)))))

; вспомогательная

(defun help\_1 (pq letters Aut res) (d2 pq letters Aut res (f2 pq (car letters) Aut nil)))

; вспомогательная

(defun find\_1(l s)

(cond

((null s) nil)

((eq\_1 l (car s)) T)

(T (find\_1 l (cdr s)))))

; вспомогательная

(defun eq\_1(l s)

(cond

((= (length l) (length s)) (eq\_2 l s))

(T nil)))

; вспомогательная

(defun eq\_2(l s)

(cond

((null l) T)

((member (car l) s) (eq\_2 (cdr l) s))

(T nil)))

; детерминируем

; M1 - сохраняем переходы для (car P)

(defun M1(Aut letters Q P res) (M2 Aut letters Q (cdr P) (help\_1 (car P) letters Aut nil) res))

; M2 - дублирование

(defun M2(Aut letters Q P sigma res) (M22 Aut letters Q P sigma sigma res))

; M22 - дополнение Q и P

(defun M22(Aut letters Q P sigma\_1 sigma\_2 res)

(cond

((null sigma\_2) (M3 Aut letters Q P sigma\_1 res))

((find\_1 (caddar sigma\_2) Q) (M22 Aut letters Q P sigma\_1 (cdr sigma\_2) res))

(T (M22 Aut letters (cons (caddar sigma\_2) Q) (cons (caddar sigma\_2) P) sigma\_1 (cdr sigma\_2) res))))

; обЪединяет текущий результат и переходит к слудующему

(defun M3(Aut letters Q P sigma res)

(cond

((null P)

(cond

((null sigma) res)

(T (unt\_2 sigma res ))))

((null sigma) (M1 Aut letters Q P res))

(T (M1 Aut letters Q P (unt\_2 sigma res )))))

; вспомогательная

(defun unt\_2(sigma res)

(cond

((null sigma) res)

(T (unt\_2 (cdr sigma) (cons (car sigma) res)))))

(defun task\_5(l) (cond

((eq(caaar l) 'H) (task\_55(CrAut l nil)))

((eq(caaar l) 'S) (task\_55(CrAut\_2 l nil)))

(T (task\_55(CrAut l nil)))))

(defun task\_55(Aut) (print Aut)

(M1 Aut (set\_letters Aut nil) (set\_St Aut) (set\_St Aut) nil))

(defun set\_letters(Aut res)

(cond

((null Aut) res)

((and (characterp (cadar Aut)) (eq (member (cadar Aut) res) nil)) (set\_letters (cdr Aut) (cons (cadar Aut) res)))

(T (set\_letters (cdr Aut) res))))

(defun set\_St(Aut) (cons (cons (caar Aut) nil) nil))

(print (task\_5 '(((S) = (B #\b l C #\c)) ((B) = (C #\b l #\a)) ((C) = (B #\b l #\a)))))