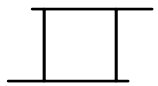
Jonkers compact



cell

chunk

unmarked cell

marked cell

any cell

header

particular cell

pointer into cell storage

pointers = $\mathbb{N}$

flags = { a(tom), m(arked), u(nmarked) }
cells = pointers × flags

$*$: pointers $\longleftrightarrow$ cells : p $\longleftrightarrow$ [π , φ]

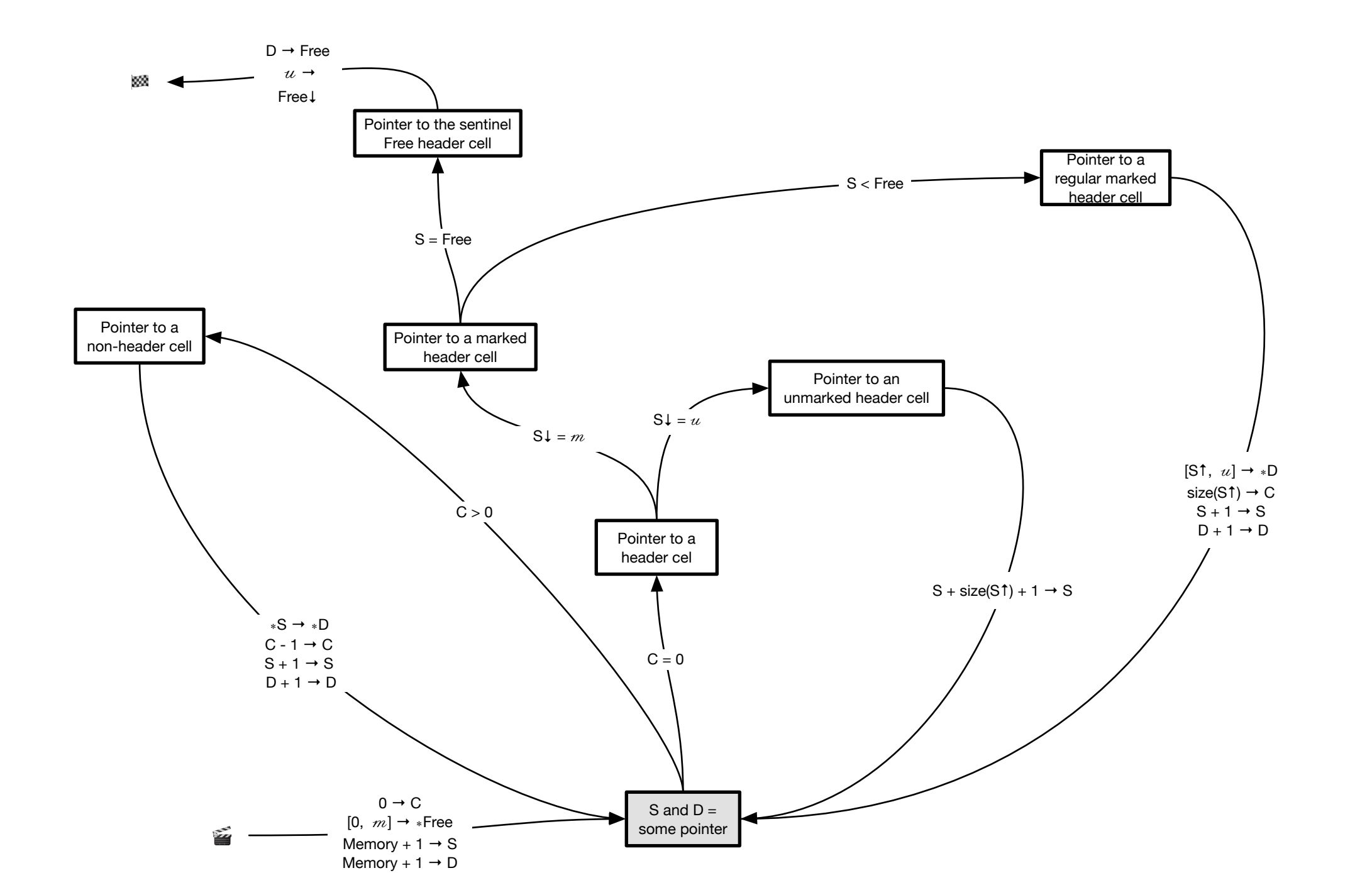
$\uparrow$: pointers $\longrightarrow$ pointers : p $\longmapsto$ p$\uparrow \equiv *\mathrm{p}_{\pi}$

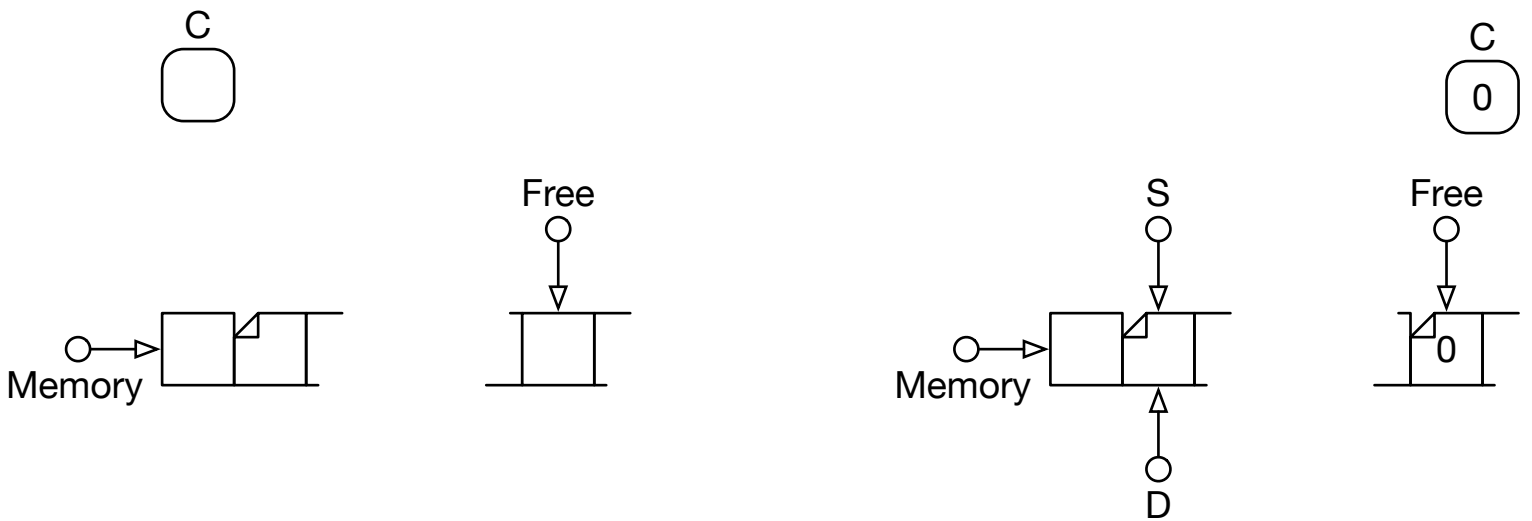
$\downarrow$: pointers $\longrightarrow$ flags : p $\longmapsto$ p$\downarrow \equiv *\mathrm{p}_{\varphi}$

regular? : cells $\longrightarrow$ boolean
raw? : cells $\longrightarrow$ boolean
size : cells $\longrightarrow \mathbb{N}$

Memory : Memory pointer
Free : Free pointer
root : root pointer

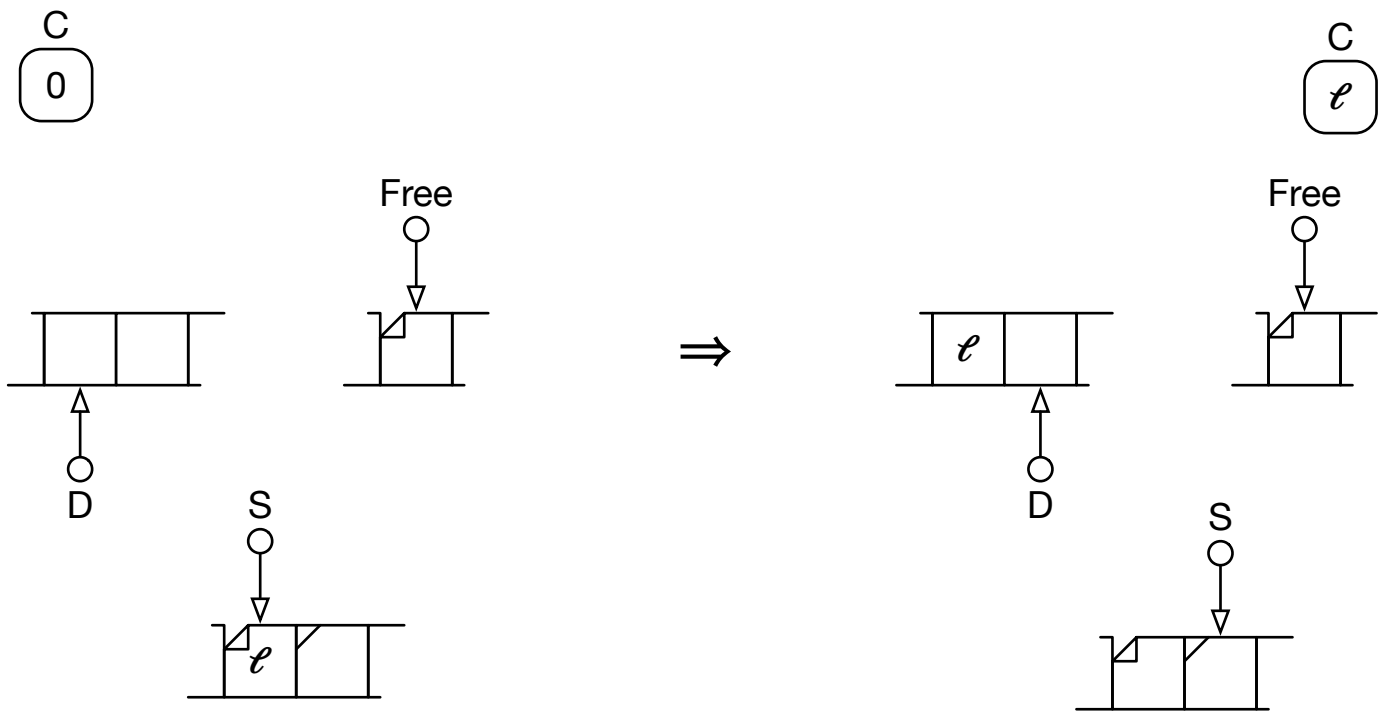
A : anchor pointer
S : source pointer
D : destination pointer


D → Free
u →
Free↓
Pointer to the sentinel Free header cell
S = Free
S < Free
Pointer to a regular marked header cell
Pointer to a non-header cell
Pointer to a marked header cell
Pointer to an unmarked header cell
S↓ = u
S↓ = m
[S↑, u] → *D
size(S↑) → C
S + 1 → S
D + 1 → D
C > 0
Pointer to a header cel
S + size(S↑) + 1 → S
*S → *D
C - 1 → C
S + 1 → S
D + 1 → D
C = 0
0 → C
[0, m] → *Free
Memory + 1 → S
Memory + 1 → D
S and D = some pointer



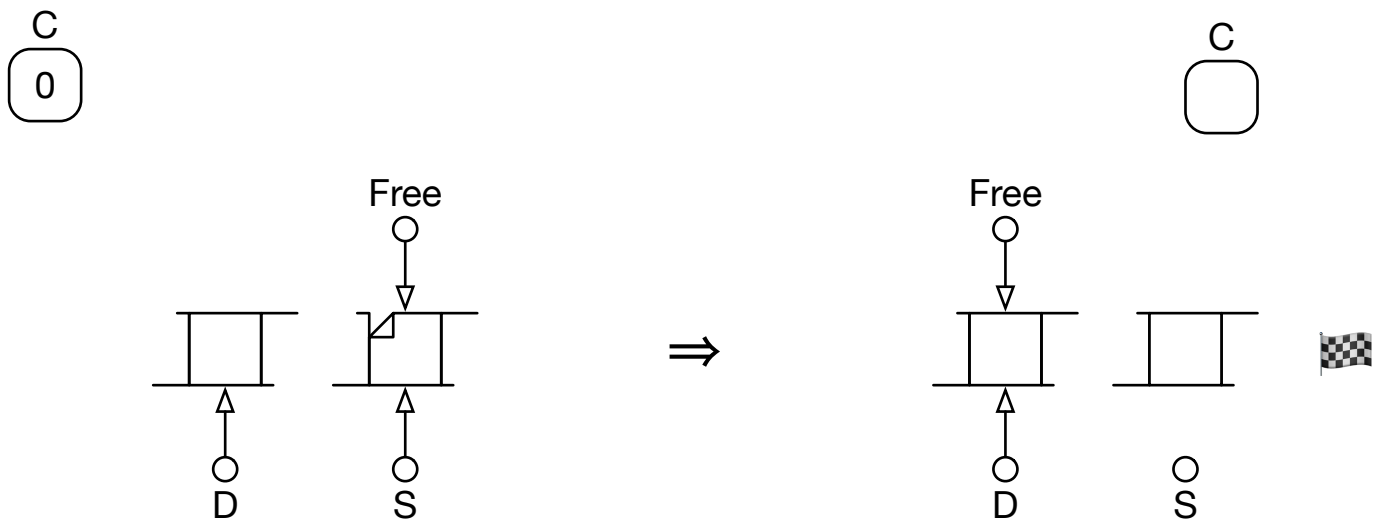
{ 0 , [0, m] , Memory + 1 , Memory + 1 } → { C , ∗Free , S , D }

(C = 0) ∧ (S↓ = m) ∧ (S < Free)



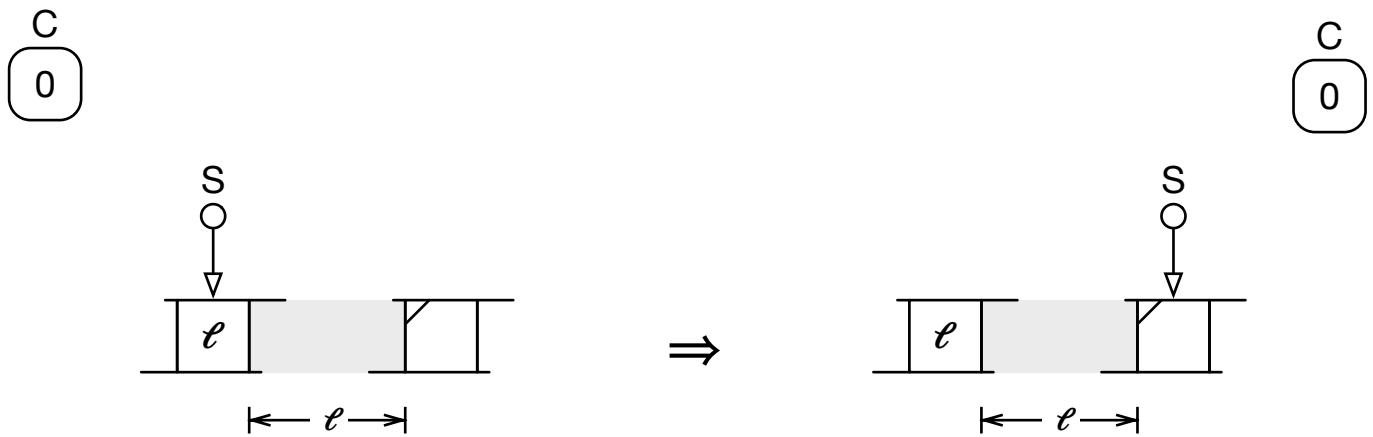
{ [S↑, u] , size(S↑) , S + 1 , D + 1 } → { ∗D , C , S , D }

(C = 0) ∧ (S↓ = m) ∧ (S = Free)



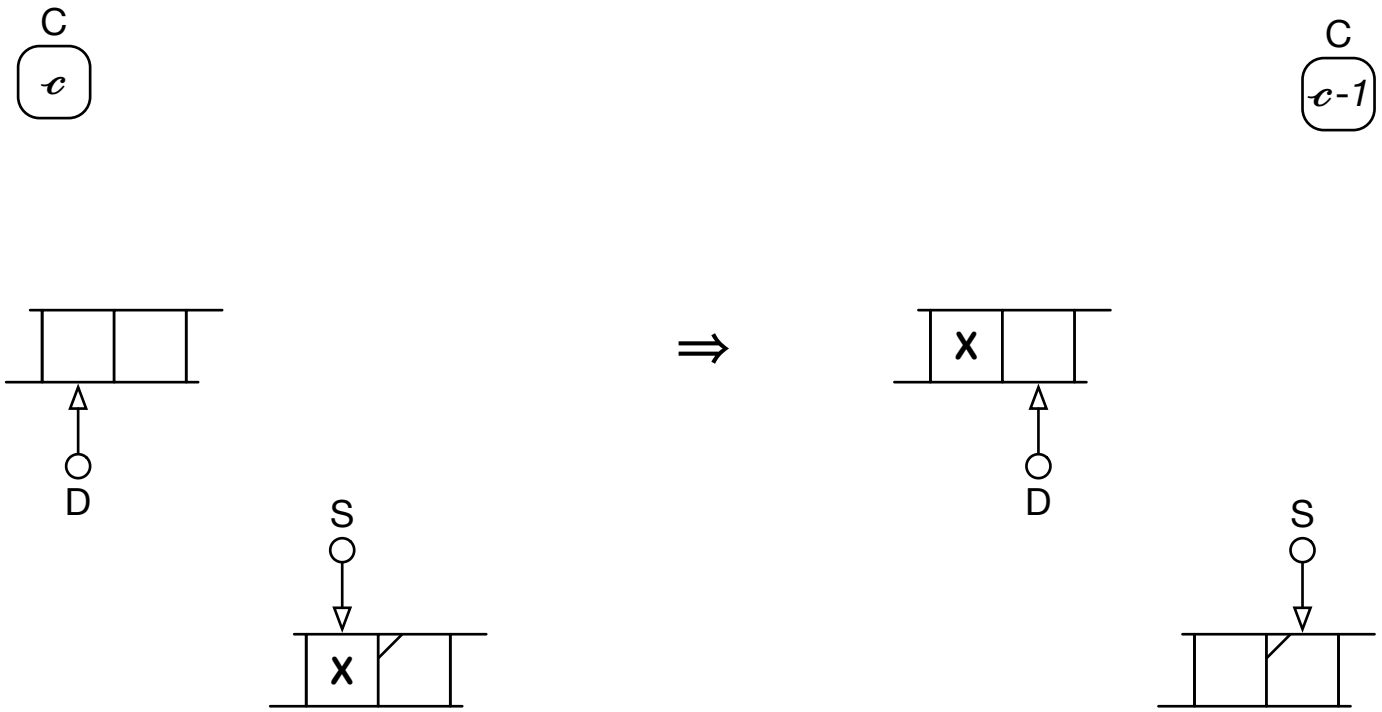
{ D , u } → { Free , Free↓}

$(C = 0) \wedge (S\downarrow = u)$



$\{\ S + \text{size}(S\uparrow) + 1\ \} \rightarrow \{\ S\ \}$

C > 0



{ ∗S , C - 1 , S + 1 , D + 1 } → { ∗D , C , S , D }

```
typedef struct CEL * ptr;
typedef enum {a, m, u} flg;
typedef struct CEL { ptr P; flg F; } cel;

ptr Memory, Free;

unsigned size(ptr);
void unmark(ptr);

void Jonkers_compact(void)
  { ptr D, S, S_;
    unsigned C;
    C = 0;                              // C <- 0
    *Free = (cel){ 0, m };              // *Free = [0, m]
    for (S = D = Memory + 1;;)          // S <- D <- Memory + 1
      { if (C == 0)                     // C = 0
          { S_ = S->P;                  // S^
            if (S->F == m)              // Sv = m
              if (S < Free)             // S < Free
                { *D = (cel){ S_, u };  // *D <- [S^, u]
                  C = size(S_);         // C <- size(S^)
                  S += 1;               // S <- S + 1
                  D += 1; }             // D <- D + 1
              else                      // S = Free
                { Free = D;             // Free <- D
                  unmark(Free);         // (Free)v <- u
                  break; }              // stop
            else                        // Sv = u
              S += size(S_) + 1; }      // S <- S + size(S^) + 1
        else                            // C > 0
          { *D = *S;                    // *D <- *S
            C -= 1;                     // C <- C - 1
            S += 1;                     // S <- S + 1
            D += 1; }}}                 // D <- D + 1
```