# Cheney

cell

paired cells

broken heart


a car cell


a cdr cell

pointer into memory

pointers = $\mathbb{N}$

types = { $a$(tom), $p$(ointer), $b$(roken heart) }
cells = pointers × types

$*$: pointers $\longrightarrow$ cells : p $\longmapsto$ $[\pi, \tau]$

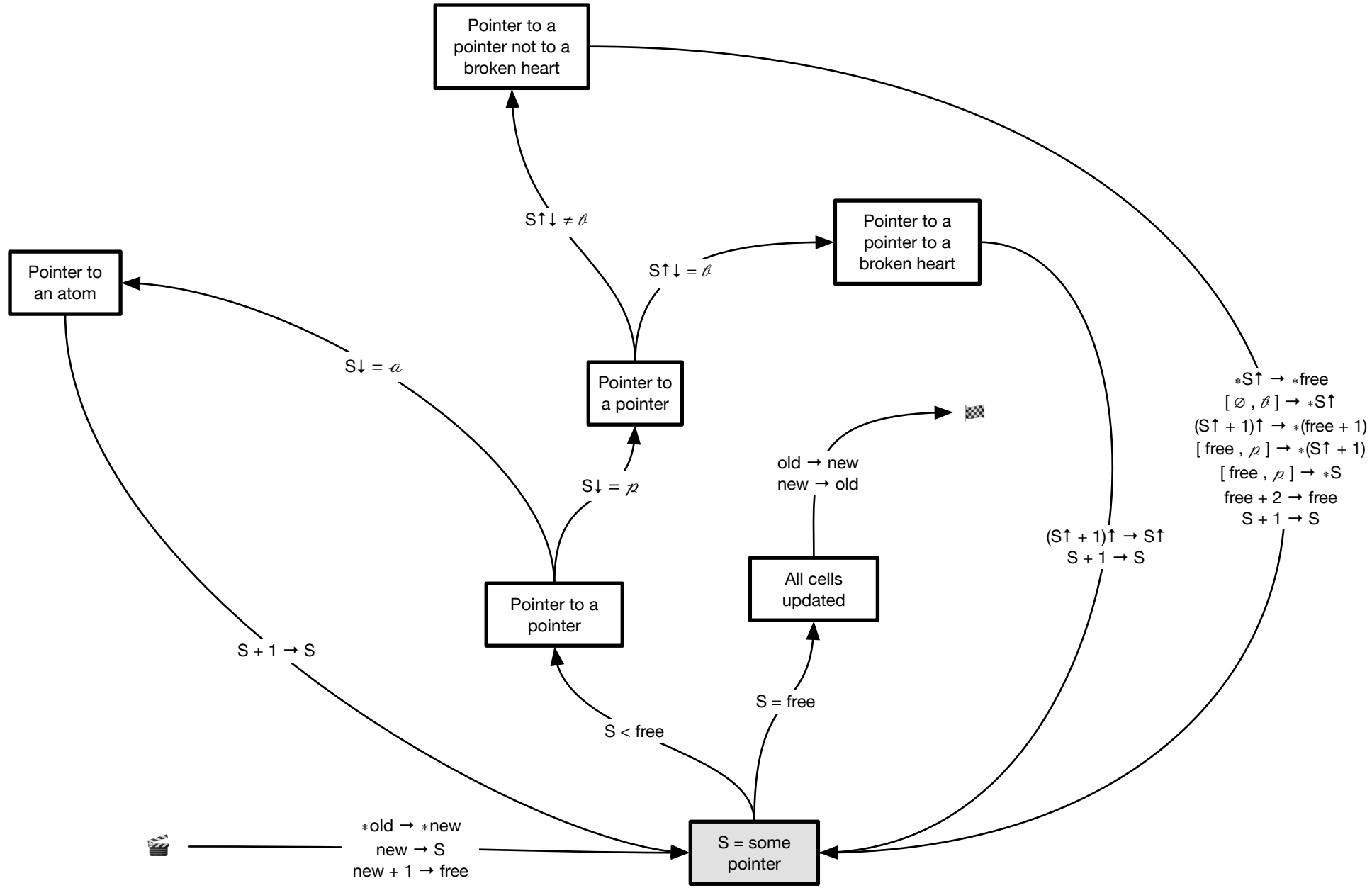
$\uparrow$: pointers $\longrightarrow$ pointers : p $\longmapsto$ p$\uparrow \equiv *\mathrm{p}_{\pi}$

$\downarrow$: pointers $\longrightarrow$ types : p $\longmapsto$ p$\downarrow \equiv *\mathrm{p}_{\tau}$

new : new memory
old : old memory

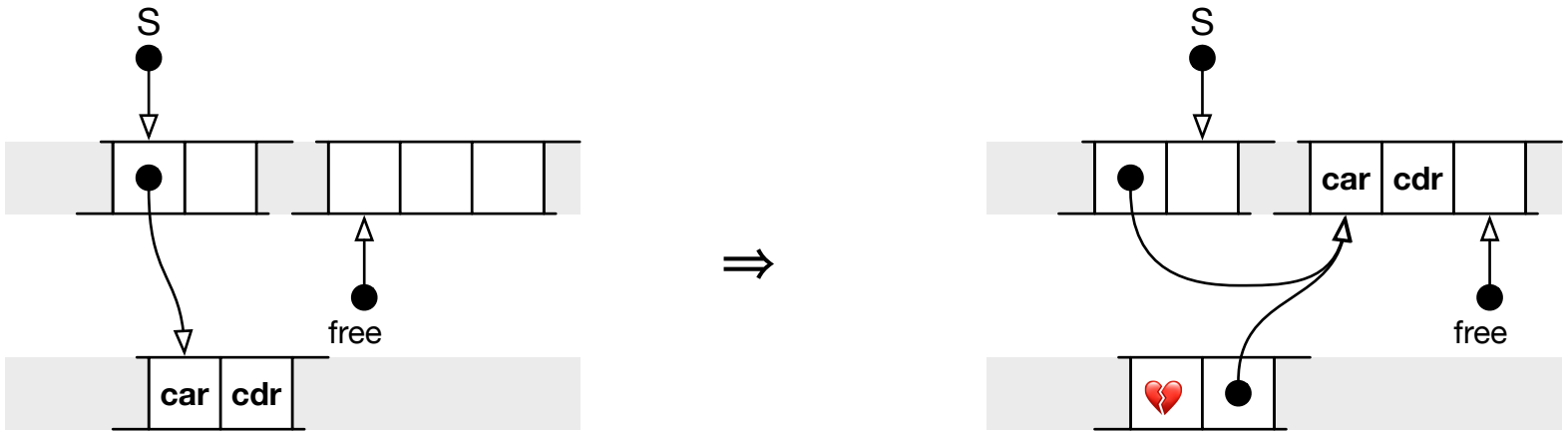
root : root pointer
free : free pointer

S : scan pointer


Pointer to a pointer not to a broken heart
S↑↓ ≠ ℓ
S↑↓ = ℓ
Pointer to a pointer to a broken heart
Pointer to an atom
S↓ = a
Pointer to a pointer
S↓ = p
old → new
new → old
All cells updated
Pointer to a pointer
S + 1 → S
S < free
S = free
(S↑ + 1)↑ → S↑
S + 1 → S
∗S↑ → ∗free
[ ∅ , ℓ ] → ∗S↑
(S↑ + 1)↑ → ∗(free + 1)
[ free , p ] → ∗(S↑ + 1)
[ free , p ] → ∗S
free + 2 → free
S + 1 → S
∗old → ∗new
new → S
new + 1 → free
S = some pointer

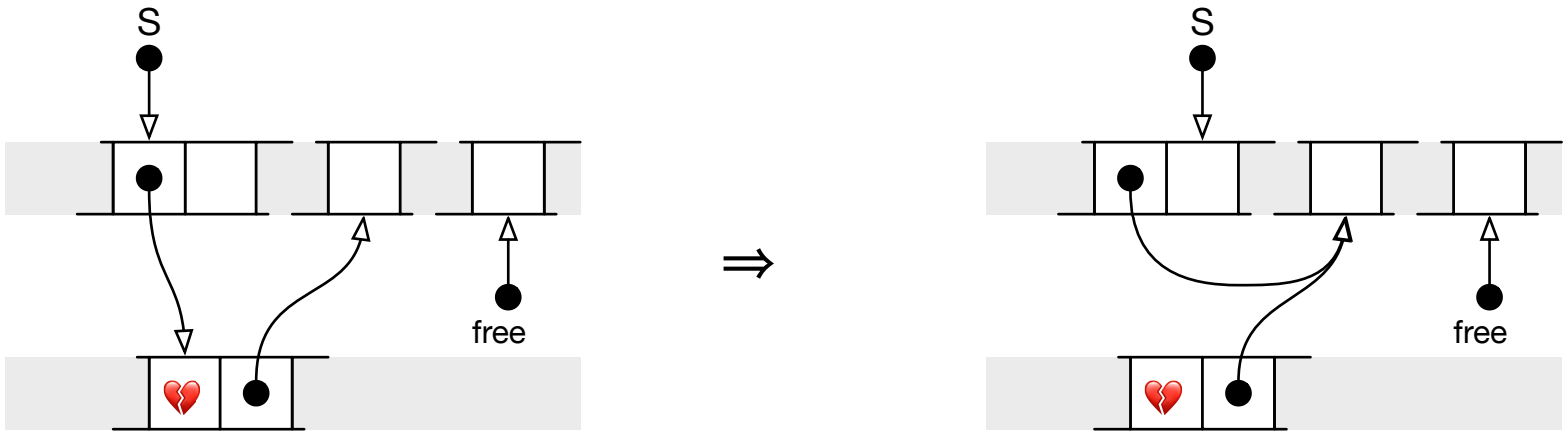

{∗old , new , new + 1 } → { ∗new , S , free }

$(S < free) \wedge (S\downarrow = p) \wedge (S\uparrow\downarrow \neq \emptyset)$



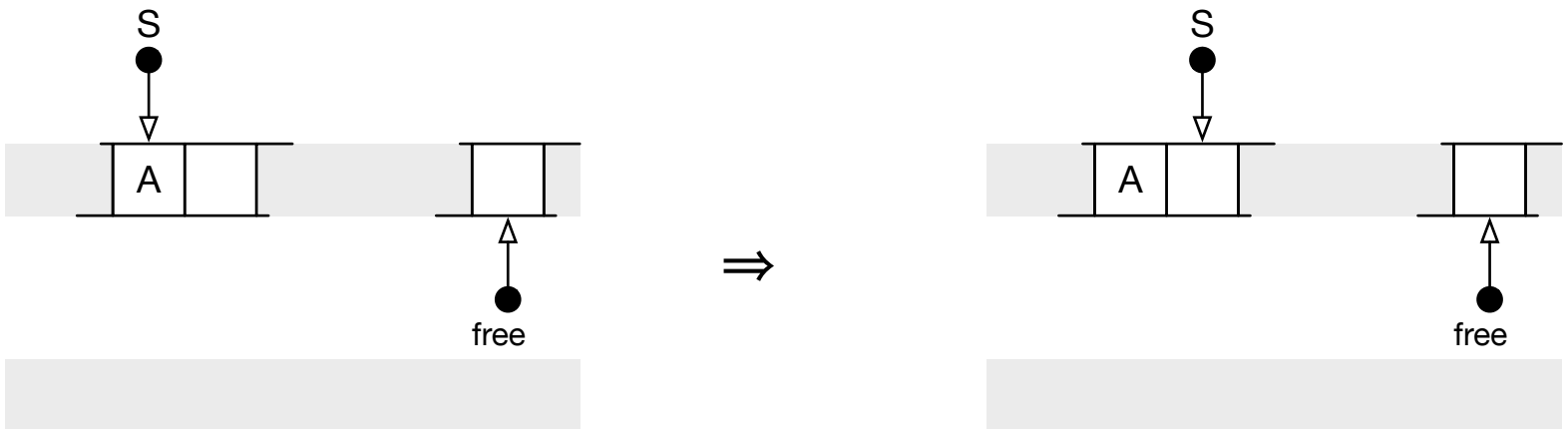

$\{ *S\uparrow , [ \emptyset , \emptyset ] , *(S\uparrow + 1) , [ free , p ] , [ free , p ] , free + 2 , S + 1 \} \rightarrow \{ *free , *S\uparrow , *(free + 1) , *(S\uparrow + 1) , *S, free , S \}$

$(S < \text{free}) \wedge (S\downarrow = p) \wedge (S\uparrow\downarrow = \emptyset)$



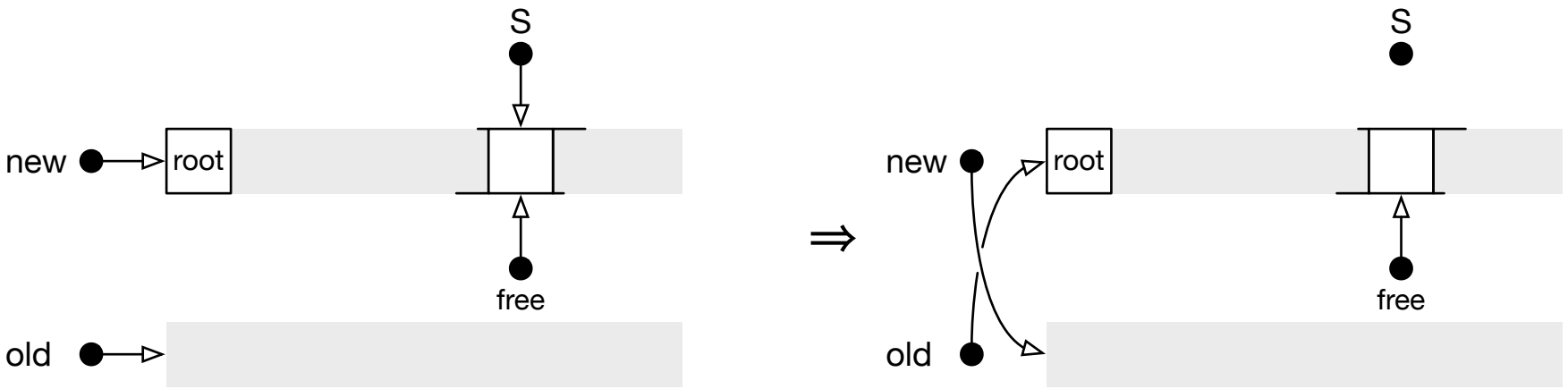

$\{ *(S\uparrow + 1) , S + 1 \} \rightarrow \{ *S , S \}$

$(S < \text{free}) \wedge (S\downarrow = a)$




$\{ S + 1 \} \rightarrow \{ S \}$

S = free




{ old , new } → { new , old } →

```
typedef struct CEL * ptr;
typedef enum {a, p, b} typ;
typedef struct CEL { ptr P; typ T; } cel;

ptr Free, New, Old;

void Cheney(void)
  { ptr S, S_;
    *New = *Old;                          // *New <- *Old
    S = New;                              // S <- New
    for (Free = New + 1;                  // Free <- New + 1
         S < Free;                        // S < Free
         S += 1)                          // S <- S + 1
      if (S->T == p)                      // Sv = p
        { S_ = S->P;                      // S^
          if (S_->T != b)                 // S^v ≠ b
            { *Free = *S_;                // *Free <- *S^
              *S_ = (cel){ 0, b };        // *S^ <- [∅, b]
              *(Free + 1) = *(S_ + 1);    // *(Free + 1) <- *(S^ + 1)
              *(S_ + 1) = (cel){ Free, p }; // *(S^ + 1) <- [Free, p]
              *S = (cel){ Free, p };      // *S <- [Free, p]
              Free += 2; }                // Free <- Free + 2
          else                            // S^v = b
            *S = *(S_ + 1); }             // *S <- *(S^ + 1)
      else;                               // Sv = a
    S = Old;                              // Old
    Old = New;                            // Old <- New
    New = S; }                            // New <- old
```