Automated Amortized Resource Analysis

Based on notes by Jan Hoffmann @ CMU

Amortized Analysis

Example: Queue from 2 stacks

push → [3|4|5] - bwd   [4|3|2|1] $\xrightarrow{\text{reverse}}$ [1|2|3|4]

pop ← [1|2] - f   Pop ——→ ?

Cost of one push: 1

Cost of one pop: 1 if fwd list is non-empty

|bwd| if fwd empty

Idea: Give each elt. in bwd stack a "token"

charge to push, use to pay for reverse

Cost of n pushes + n pops: 2n + n

Example: Binary Counter

Cost of inc: # of bit flips

1 0→1 + n 1→0

0	11	110
1	100	111
10	101	1000

Charge 2 to flip 0→1

Every 1 gets a token - used to pay to flip to 0

Cost of n incs: 2n

AARA

$\tau ::= \text{unit} \mid \text{int} \mid \cdots \mid \tau\ \text{list}^q$ ← τ with q tokens

$e ::= x \mid () \mid \cdots \mid \text{nil} \mid \text{cons}(x,x) \mid \text{case } x \text{ of } \{e; h.t.e\}$ ← if nil ← if cons(h,t)

$\mid \text{fix } x = e\ x.f \mid \text{tick } q \mid \text{let } x = e \text{ in } e$

tick q ← some op that costs q tokens

2-stack queue: $\tau\ \text{list}^0 \times \tau\ \text{list}^1$

Typing judgment: $\Gamma; q \vdash e : \tau; q_f$

q ← tokens at start; q_f ← tokens at end

$x:\text{int}, p:\text{int list}^0 \times \text{int list}^1; 2 \vdash \text{push} : \text{int list}^0 \times \text{int list}^1; 0$

$p:\text{int list}^0 \times \text{int list}^1; 1 \vdash \text{pop} : \text{int} \times (\text{int list}^0 \times \text{int list}^1); 0$

$$\frac{}{\Gamma, x:\tau; 0 \vdash x:\tau; 0}\ (\text{Var}) \qquad \frac{}{\Gamma; 0 \vdash \text{nil} : \tau\ \text{list}^q; 0}\ (\text{list-}I_1)$$

$$\frac{}{\Gamma, x_1:\tau\ x_2:\tau\ \text{list}^q; q \vdash \text{cons}(x_1,x_2) : \tau\ \text{list}^q; 0}\ (\text{list-}I_2)$$

q ← pay for the tokens

$$\frac{\Gamma; q \vdash e_0 : \tau'; q' \qquad \Gamma, h:\tau, t:\tau\ \text{list}^p; q+p \vdash e_1 : \tau'; q'}{\Gamma, x:\tau\ \text{list}^p; q \vdash \text{case } x \text{ of } \{e_0; h.t.\ e_1\} : \tau'; q'}\ (\text{list-}F)$$

$$\frac{\Gamma_1; q \vdash e_1 : \tau; q' \qquad \Gamma_2, x:\tau; q' \vdash e_2 : \tau; q''}{\Gamma_1, \Gamma_2; q \vdash \text{let } x = e_1 \text{ in } e_2 : \tau; q''}\ (\text{let}) \qquad \frac{}{\Gamma; q \vdash \text{tick } q : \text{unit}; 0}$$

Γ_1, Γ_2 ← will explain later push:

② let _ = tick 1 in (⓪ fst $q^{:\,\text{int list}^0}$, cons ($x^{:\,\text{int}}$, snd $q^{:\,\text{int list}^1}$))

pop: ① let _ = tick 1 in ⓪
case fst q of
{ case rev (snd q) of
{ error;
h. t. (h, (t, nil))
};
h. t. (t, snd q)
}

$\text{rev} : \tau \text{ list}^1 \to \tau \text{ list}^0$

Soundness:

Cost semantics: $e \Downarrow v ; q$ e evals. to v w/ cost q

$$\frac{}{v \Downarrow v ; 0} \qquad \frac{}{\text{tick } q \Downarrow () ; q} \qquad \frac{e_1 \Downarrow v_1 ; q_1 \quad e_2 \Downarrow v_2 ; q_2}{(e_1, e_2) \Downarrow (v_1, v_2) ; q_1 + q_2}$$

Thm: If $\cdot ; q \vdash e : \tau ; q'$ then $e \Downarrow v ; p$ and $p \leq q - q'$.

What's to stop us from doing this?

① let $l^{:\,\text{unit list}^1}$ = cons ((), nil) in
case l of { ... ;
_ . _ . case l of { ... ;
_ . _ . case l of { ... ;
_ . _ . e
}
}
}

Linear (Affine) Type Systems

Can use variables from Γ (at most) once

Affine:

$$\frac{\Gamma_1 \vdash e_1 : \tau \qquad \Gamma_2, x:\tau \vdash e_2 : \tau'}{\underbrace{\Gamma_1, \Gamma_2}_{} \vdash \text{let } x = e_1 \text{ in } e_2 : \tau'}$$

split into 2 disjoint ctxs: can't share

Contraction: $\Gamma, x:\tau, x:\tau = \Gamma, x:\tau$ $\cancel{\checkmark}$

Linear: Weakening $\cancel{\checkmark}$

$$\frac{}{x:\tau \vdash x:\tau}\,(\text{var})$$

ctx must have only x