15-312 Assignment 1

Andrew Carnegie
(andrew)

March 14, 2018

1 Introduction

In this paper, we propose a model for deriving asymptotically tight bounds for first order functional programs. We choose a fragment of OCaml as the target language. The abstract and concrete syntax of the language is show below. Note that we only allow first order functions of type $\tau_1 \to \tau_2$, where τ_1 and τ_2 are base types: unit, bool, product, or lists.

BTypes	τ ::=		
	$\mathtt{nat}$	$\mathtt{nat}$	naturals
	$\mathtt{unit}$	$\mathtt{unit}$	unit
	$\mathtt{bool}$	$\mathtt{bool}$	boolean
	$\mathtt{prod}(\tau_1;\tau_2)$	$\tau_1 \times \tau_2$	product
	$\mathtt{list}(\tau)$	$L(\tau)$	list
FTypes	ρ ::=		
	$\mathtt{arr}(\tau_1;\tau_2)$	$\tau_1 \to \tau_2$	first order function
Exp	e ::=		
	x	x	variable
	$\mathtt{nat}[n]$	$\overline{n}$	number
	$\mathtt{unit}$	$()$	unit
	$\mathtt{T}$	$\mathtt{T}$	true
	$\mathtt{F}$	$\mathtt{F}$	false
	$\mathtt{plus}(e_1;e_2)$	$e_1 + e_2$	plus
	$\mathtt{minus}(e_1;e_2)$	$e_1 - e_2$	minus
	$\mathtt{eq}(e_1;e_2)$	$e_1 = e_2$	equality
	$\mathtt{lt}(e_1;e_2)$	$e_1 < e_2$	less-than
	$\mathtt{and}(e_1;e_2)$	$e_1 \wedge e_2$	conjunction
	$\mathtt{or}(e_1;e_2)$	$e_1 \vee e_2$	disjunction
	$\mathtt{not}(e)$	$\neg e$	negation
	$\mathtt{if}(x;e_1;e_2)$	$\mathtt{if}\, x \,\mathtt{then}\, e_1 \,\mathtt{else}\, e_2$	if
	$\mathtt{lam}(x:\tau.e)$	$\lambda\, x:\tau.e$	abstraction
	$\mathtt{ap}(f;x)$	$f(x)$	application
	$\mathtt{tpl}(x_1;x_2)$	$\langle x_1, x_2\rangle$	pair
	$\mathtt{case}(x_1,x_2.e_1)$	$\mathtt{case}\ p\ \{(x_1;x_2) \hookrightarrow e_1\}$	match pair
	$\mathtt{nil}$	$[]$	nil
	$\mathtt{cons}(x_1;x_2)$	$x_1 :: x_2$	cons
	$\mathtt{case}\{l\}(e_1;x,xs.e_2)$	$\mathtt{case}\, l\, \{\mathtt{nil} \hookrightarrow e_1 \mid \mathtt{cons}(x;xs) \hookrightarrow e_2\}$	match list
	$\mathtt{let}(e_1;x:\tau.e_2)$	$\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2$	let
	$\mathtt{share}(x;x_1,x_2.e)$	$\mathtt{share}\ x\ \mathtt{as}\ x_1,x_2\ \mathtt{in}\ e$	share
Val	v ::=		
	$\mathtt{val}(n)$	n	numeric value
	$\mathtt{val}(\mathtt{T})$	$\mathtt{T}$	true value
	$\mathtt{val}(\mathtt{F})$	$\mathtt{F}$	false value
	$\mathtt{val}(\mathtt{Null})$	$\mathtt{Null}$	null value
	$\mathtt{val}(\mathtt{cl}(V;x.e))$	$(V, x.e)$	function value
	$\mathtt{val}(l)$	l	loc value
	$\mathtt{val}(\mathtt{pair}(v_1;v_2))$	$\langle v_1, v_2\rangle$	pair value
State	s ::=		
	$\mathtt{alive}$	$\mathtt{alive}$	live value
	$\mathtt{dead}$	$\mathtt{dead}$	dead value
Loc	l ::=		
	$\mathtt{loc}(l)$	l	location
Var	l ::=		
	$\mathtt{var}(x)$	x	variable

2 Preliminaries

For a finite mapping $f : A \to B$, we write *dom* for the defined values of f. Sometimes we shorten $x \in dom(f)$ to $x \in f$. We write $f[x \mapsto y]$ for the extension of f where x is mapped to y, with the constraint that $x \notin dom(f)$.

Roots represents the set of locations required to compute the continuation *excluding* the current expression. We can think of roots as the heap allocations necessary to compute the context with a hole that will be filled by the current expression.

In order prove soundness of the type system, we need some auxiliary judgements to defining properties of a heap. Below we define $reach : Val \to \{\{\mathsf{Loc}\}\}$ that maps stack values its the root *multiset*, the multiset of locations that's already on the stack.
Next we define reachability of values:

$$\begin{aligned}
&reach_H(\langle v_1, v_2\rangle) = reach_H(v_1) \uplus reach_H(v_2)\\
&reach_H(l) = \{l\} \uplus reach_H(H(l))\\
&reach_H(_) = \emptyset
\end{aligned}$$

For a multiset S, we write $\mu_S : S \to \mathbb{N}$ for the multiplicity function of S, which maps each element to the count of its occurence. If $\mu_S(x) \geq 1$ for a multiset S, then we write $x \in S$ as in the usual set membership relation. If for all $s \in S$, $\mu(s) = 1$, then S is a property set, and we denote it by $\mathsf{set}(S)$. Addtionally, $A \uplus B$ denotes counting union of sets where $\mu_{A \uplus B}(s) = \mu_A(s) + \mu_B(s)$, and $A \cup B$ denotes the usual union where $\mu_{A\cup B}(s) = \max(\mu_A(s), \mu_B(s))$. For the disjoint union of sets A and B, we write $A \sqcup B$.

Next, we define the predicates no_alias, stable, and disjoint:

no_alias(V, H): $\forall x, y \in V,\ x \neq y.$ Let $r_x = reach_H(V(x))$, $r_y = reach_H(V(y))$. Then:

1. $\mathsf{set}(r_x), \mathsf{set}(r_y)$
2. $r_x \cap r_y = \emptyset$

stable(R, H, H'): $\forall l \in R.\ H(l) = H'(l)$.

safe(V, H, F): $\forall x \in V.\ reach_H(V(x)) \cap F = \emptyset$

disjoint$(\mathcal{C})$: $\forall X, Y \in \mathcal{C}.\ X \cap Y = \emptyset$

For a stack V and a heap H, whenever no_alias(V, H) holds, visually, one can think of the situation as the following: the induced graph of heap H with variables on the stack as additional leaf nodes is a forest: a disjoint union of arborescences (directed trees); consequently, there is at most one path from a live variable on the stack V to a location in H by following the pointers.

First, we define $FV^\star(e)$, the multiset of free variables of e. As the usual FV, it is defined inductively over the structure of e; the only unusual thing is that multiple occurences of a free variable x in e will be reflected in the multiplicity of $FV^\star(e)$.

Next, we define $locs_{V,H}$ using the previous notion of reachability.

$$locs_{V,H}(e) = \bigcup_{x \in FV(e)} reach_H(V(x))$$

size calculates the *literal size* of a value, e.g. the size to store its address.

$$\begin{aligned}
&size(\langle v_1, v_2\rangle) = size(v_1) + size(v_2)\\
&size(_) = 1
\end{aligned}$$

Let $card(S)$ denote the number of unique elements, e.g. the cardinality of a multiset S. We write $\|v\|_H$ for $card(reach_H(v))$
As usual, we extend it to stacks V: $\|V\|_H = \sum_{V(x)=v} \|v\|_H$

Let $copy(H, L, v, H', v')$ be a 5-place relation on $\mathsf{Heap} \times \mathsf{Loc} \times \mathsf{Val} \times \mathsf{Heap} \times \mathsf{Val}$. We write this as $H', v = copy(H, L, v)$ to signify the intended mode for this predicate: $(+, +, +, -, -)$.

$$\frac{v \in \{n, \mathtt{T}, \mathtt{F}, \mathtt{Null}\}}{H, v = copy(H, L, v)} \qquad \frac{l' \in L \qquad H', v = copy(H, L \setminus \{l'\}, H(l))}{H'\{l' \mapsto v\}, l' = copy(H, L, l)}$$

$$\frac{L_1 \sqcup L_2 \subseteq L \qquad |L_1| = \|v_1\|_H \qquad |L_2| = \|v_2\|_H \qquad H_1, v_1' = copy(H, L_1, v_1) \qquad H_2, v_2' = copy(H_1, L_2, v_2)}{H_2, \langle v_1', v_2' \rangle = copy(H, L, \langle v_1, v_2 \rangle)}$$

3 Garbage collection semantics

The garbage collection operation semantics consists of judgement of the form:

$$\boxed{\mathcal{C} \vdash_{P:\Sigma} e \Downarrow v, H', F'}$$

Where $\mathcal{C}$ is a *configuration*, consisting of a 4-tuple in $\mathsf{Stack} \times \mathsf{Heap} \times \{\mathsf{Loc}\} \times \{\mathsf{Loc}\}$, usually written as V, H, R, F. P is a program with signature $\Sigma : \mathsf{Var} \to \mathsf{FTypes}$. This can be read as: under stack V, heap H, roots R, freelist F, and program P with signature Σ, the expression e evaluates to v, and engenders a new heap H' and freelist F'. Here, Stack is defined as the set of finite mappings $\mathsf{Var} \to \mathsf{Val}$, and Heap is defined as the set of finite mappings $\mathsf{Loc} \to \mathsf{Val}$.

A *program* is then a Σ indexed map P from Var to pairs $(y_f, e_f)_{f \in \Sigma}$, where $\Sigma(y_f) = A \to B$, and $\Sigma; y_f : A \vdash e_f : B$ (typing rules are discussed in 7). We write $P : \Sigma$ to mean P is a program with signature Σ. Because the signature Σ for the mapping of function names to first order functions does not change during evaluation, we drop the subscript Σ from $\vdash_\Sigma$ when the context of evaluation is clear. It is convenient to think of the evaluation judgement $\vdash$ as being indexed by a family of signatures Σ's, each of which is a set of "top-level" first-order declarations to be used during evaluation.

$$\frac{V(x) = v}{V, H, R, F \vdash x \Downarrow v, H, F}(\mathrm{S}_1) \qquad \frac{}{V, H, R, F \vdash \overline{n} \Downarrow \mathtt{val}(n), H, F}(\mathrm{S}_2) \qquad \frac{}{V, H, R, F \vdash \mathtt{T} \Downarrow \mathtt{val}(\mathtt{T}), H, F}(\mathrm{S}_3)$$

$$\frac{}{V, H, R, F \vdash \mathtt{F} \Downarrow \mathtt{val}(\mathtt{F}), H, F}(\mathrm{S}_4) \qquad \frac{}{V, H, R, F \vdash () \Downarrow \mathtt{val}(\mathtt{Null}), H, F}(\mathrm{S}_5)$$

$$\frac{V = V'[x \mapsto \mathtt{T}] \qquad g = \{l \in H \mid l \notin F \cup R \cup locs_{V,H}(e_1)\} \qquad V', H, R, F \cup g \vdash e_1 \Downarrow v, H', F'}{V, H, R, F \vdash \mathtt{if}(x; e_1; e_2) \Downarrow v, H', F'}(\mathrm{S}_6)$$

$$\frac{V = V'[x \mapsto \mathtt{F}] \qquad g = \{l \in H \mid l \notin F \cup R \cup locs_{V,H}(e_2)\} \qquad V', H, R, F \cup g \vdash e_2 \Downarrow v, H', F'}{V, H, R, F \vdash \mathtt{if}(x; e_1; e_2) \Downarrow v, H', F'}(\mathrm{S}_7)$$

$$\frac{\begin{array}{c} V(x) = v' \\ P(f) = (y_f, e_f) \qquad g = \{l \in H \mid l \notin F \cup R \cup locs_{V,H}(e_f)\} \qquad [y_f \mapsto v'], H, R, F \cup g \vdash e_f \Downarrow v, H', F' \end{array}}{V, H, R, F \vdash f(x) \Downarrow v, H', F'}(\mathrm{S}_8)$$

$$\frac{V(x_1) = v_1 \qquad V(x_2) = v_2}{V, H, R, F \vdash \langle x_1, x_2 \rangle \Downarrow \langle v_1, v_2 \rangle, H, F}(\mathrm{S}_9)$$

$$\frac{\begin{array}{c} V = V'[x \mapsto \langle v_1, v_2 \rangle] \\ g = \{l \in H \mid l \notin F \cup R \cup locs_{V,H}(e)\} \qquad V'' = V'[x_1 \mapsto v_1, x_2 \mapsto v_2] \qquad V'', H, R, F \cup g \vdash e \Downarrow v, H', F' \end{array}}{V, H, R, F \vdash \mathtt{case}\ x\ \{(x_1; x_2) \hookrightarrow e\} \Downarrow v, H', F'}(\mathrm{S}_{10})$$

$$\frac{}{V, H, R, F \vdash \mathtt{nil} \Downarrow \mathtt{val}(\mathtt{Null}), H, F}(\mathrm{S}_{11}) \qquad \frac{v = \langle V(x_1), V(x_2) \rangle \qquad l \in F \qquad H' = H\{l \mapsto v\}}{V, H, R, F \vdash \mathtt{cons}(x_1; x_2) \Downarrow l, H', F \setminus \{l\}}(\mathrm{S}_{12})$$

$$\frac{V = V'[x \mapsto v'] \qquad g = reach_H(v') \qquad V', H, R, F \cup g \vdash e \Downarrow v, H', F'}{V, H, R, F \vdash \mathtt{drop}(x; e) \Downarrow v, H', F'}(\mathrm{S}_{13})$$

$$\frac{\begin{array}{c} V = V'[x \mapsto v'] \qquad L \subseteq F \\ |L| = \left\|v'\right\|_H \qquad H', v'' = copy(H, L, v') \qquad V_2 = V'[x_1 \mapsto v', x_2 \mapsto v''] \qquad V_2, H', R, F \setminus L \vdash e \Downarrow v, H'', F' \end{array}}{V, H, R, F \vdash \mathtt{shareCopy}\ x\ \mathtt{as}\ x_1, x_2\ \mathtt{in}\ e \Downarrow v, H'', F'}(\mathrm{S}_{14})$$

$$\frac{\begin{array}{c} V(x) = \mathtt{Null} \qquad V' \subseteq V \\ dom(V') = FV(e_1) \qquad g = \{l \in H \mid l \notin F \cup R \cup locs_{V,H}(e_1)\} \qquad V', H, R, F \cup g \vdash e_1 \Downarrow v, H', F' \end{array}}{V, H, R, F \vdash \mathtt{case}\, x\, \{\mathtt{nil} \hookrightarrow e_1 \mid \mathtt{cons}(x_h; x_t) \hookrightarrow e_2\} \Downarrow v, H', F'}(\mathrm{S}_{15})$$

$$\frac{\begin{array}{c} V(x) = l \qquad H(l) = \langle v_h, v_t \rangle \qquad V' \subseteq V \qquad dom(V') = FV(e_2) \setminus \{x_h, x_t\} \\ V'' = V'[x_h \mapsto v_h, x_t \mapsto v_t] \qquad g = \{l \in H \mid l \notin F \cup R \cup locs_{V'',H}(e_2)\} \qquad V'', H, R, F \cup g \vdash e_2 \Downarrow v, H', F' \end{array}}{V, H, R, F \vdash \mathtt{case}\, x\, \{\mathtt{nil} \hookrightarrow e_1 \mid \mathtt{cons}(x_h; x_t) \hookrightarrow e_2\} \Downarrow v, H', F'}(\mathrm{S}_{16})$$

$$\frac{\begin{array}{c} V = V_1 \sqcup V_2 \qquad dom(V_1) = FV(e_1) \\ dom(V_2) = FV(\mathtt{lam}(x : \tau.e_2)) \qquad R' = R \cup locs_{V_2,H}(\mathtt{lam}(x : \tau.e_2)) \qquad V_1, H, R', F \vdash e_1 \Downarrow v_1, H_1, F_1 \\ V_2' = V_2[x \mapsto v_1] \qquad g = \{l \in H_1 \mid l \notin F_1 \cup R \cup locs_{V_2',H_1}(e_2)\} \qquad V_2', H_1, R, F_1 \cup g \vdash e_2 \Downarrow v_2, H_2, F_2 \end{array}}{V, H, R, F \vdash \mathtt{let}(e_1; x : \tau.e_2) \Downarrow v_2, H_2, F_2}(\mathrm{S}_{17})$$

4 Operational semantics

In order to prove the soundess of the type system, we also define a simplified operational semantics that does not account for garbage collection.

$$\boxed{\mathcal{S} \vdash e \Downarrow v, H'}$$

Where $\mathcal{S}$ is a *context*, consisting of a tuple in $\mathsf{Stack} \times \mathsf{Heap}$, and usually written as (V, H). This can be read as: under stack V, heap H the expression e evaluates to v, and engenders a new heap H'. We write the representative rules, since the rest are derived in the obvious way from the garbage collection semantics.

$$\frac{l \notin dom(H) \qquad v = \langle V(x_1), V(x_2)\rangle \qquad H' = H\{l \mapsto v\}}{V, H \ \vdash \texttt{cons}(x_1; x_2) \Downarrow l, H'}(\mathrm{S}_{18})$$

$$\frac{\begin{array}{c} V(x) = l \qquad H(l) = \langle v_h, v_t\rangle \\ V' \subseteq V \qquad dom(V') = FV(e_2) \setminus \{x_h, x_t\} \qquad V'' = V'[x_h \mapsto v_h, x_t \mapsto v_t] \qquad V'', H \ \vdash e_2 \Downarrow v, H' \end{array}}{V, H \ \vdash \texttt{case}\, x\, \{\texttt{nil} \hookrightarrow e_1 \mid \texttt{cons}(x_h; x_t) \hookrightarrow e_2\} \Downarrow v, H'}(\mathrm{S}_{19})$$

$$\frac{\begin{array}{c} V = V_1 \sqcup V_2 \qquad dom(V_1) = FV(e_1) \\ dom(V_2) = FV(\texttt{lam}(x : \tau.e_2)) \qquad V_1, H \vdash e_1 \Downarrow v_1, H_1 \qquad V_2' = V_2[x \mapsto v_1] \qquad V_2', H_1 \vdash e_2 \Downarrow v_2, H_2 \end{array}}{V, H \ \vdash \texttt{let}(e_1; x : \tau.e_2) \Downarrow v_2, H_2}(\mathrm{S}_{20})$$

$$\frac{\begin{array}{c} V = V'[x \mapsto v'] \\ L \cap dom(H) = \emptyset \qquad |L| = \left\|v'\right\|_H \qquad H', v'' = copy(H, L, v') \qquad V'[x_1 \mapsto v', x_2 \mapsto v''], H' \ \vdash e \Downarrow v, H'' \end{array}}{V, H \ \vdash \texttt{shareCopy}\ x\ \texttt{as}\ x_1, x_2\ \texttt{in}\ e \Downarrow v, H''}(\mathrm{S}_{21})$$

5 Well Defined Environments

In order to define the potential for first-order types, we need a notion of well-define environments, one that relates heap values to semantic values of a type. We first give a denotational semantics for the first-order types:

$$\begin{aligned}
() &\in [\![\texttt{unit}]\!] \\
\bot &\in [\![\texttt{bool}]\!] \\
\top &\in [\![\texttt{bool}]\!] \\
0 &\in [\![\texttt{nat}]\!] \\
n+1 &\in [\![\texttt{nat}]\!] \text{ if } n \in [\![\texttt{nat}]\!] \\
\langle a_1, a_2\rangle &\in [\![A_1 \times A_2]\!] \text{ if } a_1 \in [\![A_1]\!] \text{ and } a_2 \in [\![A_2]\!] \\
[] &\in [\![L(A)]\!] \\
\pi(a, l) &\in [\![L(A)]\!] \text{ if } a \in [\![A]\!] \text{ and } l \in [\![L(A)]\!]
\end{aligned}$$

Where semantic set for each type is the least set such that the above holds. Note $\pi(x, y)$ is the usual set-theoretic pairing function, and write $[a_1, ..., a_n]$ for $\pi(a_1, ..., \pi(a_n, []))$.

Now we give the judgements relating heap values to semantic values, in the form $\boxed{H \vDash v \mapsto a : A}$, which can be read as: under heap H, heap value v defines the semantic value $a \in [\![A]\!]$.

$$\frac{n \in \mathbb{Z}}{H \vDash n \mapsto n : \texttt{nat}}\text{(V:ConstI)} \qquad \frac{}{H \vDash \texttt{Null} \mapsto n : \texttt{unit}}\text{(V:ConstI)} \qquad \frac{A \in \mathsf{BType}}{H \vDash \texttt{Null} \mapsto n : L(A)}\text{(V:Nil)}$$

$$\frac{}{H \vDash \texttt{T} \mapsto \top : \texttt{bool}}\text{(V:True)} \qquad \frac{}{H \vDash \texttt{F} \mapsto \bot : \texttt{bool}}\text{(V:False)} \qquad \frac{H \vDash v_1 \mapsto a_1 : A_1 \qquad H \vDash v_2 \mapsto a_2 : A_2}{H \vDash \langle v_1, v_2 \rangle \mapsto \langle a_1, a_2 \rangle : A_1 \times A_2}\text{(V:Pair)}$$

$$\frac{l \in \mathsf{Loc} \qquad H(l) = \langle v_h, v_t \rangle \qquad H \vDash v_h \mapsto a_1 : A \qquad H \vDash v_t \mapsto [a_2, \ldots, a_n] : L(A)}{H \vDash l \mapsto [a_1, \ldots, a_n] : L(A)}\text{(V:Cons)}$$

6 Stack vs Heap Allocated Types

In order to share variables, we need to distinguish between types that are allocated on the stack and the heap. We write $\boxed{\texttt{stack}(A)}$ to denote that values of type A will be allocataed *entirely* on the stack at run time (no references into the heap).

$$\frac{A \in \{\texttt{unit}, \texttt{bool}, \texttt{nat}\}}{\texttt{stack}(A)}\text{(S:Const)} \qquad \frac{\texttt{stack}(A_1) \qquad \texttt{stack}(A_2)}{\texttt{stack}(A_1 \times A_2)}\text{(S:Product)}$$

7 Linear Garbage Collection Type Rules

The linear version of the type system takes into account of garbaged collected cells by returning potential locally in a match construct. Since we are interested in the number of heap cells, all constants are assumed to be nonnegative. The second let rule expresses the fact that since stack types don't reference heap cells, any heap cells used in the evaluation of e_1 can be deallocated, as there are no longer references to them in v_1.

$$\frac{n \in \mathbb{Z}}{\Sigma; \emptyset \left|\frac{q}{q}\right. n : \mathtt{nat}}(\text{L:ConstI}) \qquad \frac{}{\Sigma; \emptyset \left|\frac{q}{q}\right. () : \mathtt{unit}}(\text{L:ConstU}) \qquad \frac{}{\Sigma; \emptyset \left|\frac{q}{q}\right. \mathtt{T} : \mathtt{bool}}(\text{L:ConstT})$$

$$\frac{}{\Sigma; \emptyset \left|\frac{q}{q}\right. \mathtt{F} : \mathtt{bool}}(\text{L:ConstF}) \qquad \frac{}{\Sigma; x : B \left|\frac{q}{q}\right. x : B}(\text{L:Var}) \qquad \frac{\Sigma(f) = A \xmapsto{q/q'} B}{\Sigma; x : A \left|\frac{q}{q'}\right. f(x) : B}$$

$$\frac{\Sigma; \Gamma \left|\frac{q}{q'}\right. e_t : B \qquad \Sigma; \Gamma \left|\frac{q}{q'}\right. e_f : B}{\Sigma; \Gamma, x : \mathtt{bool} \left|\frac{q}{q'}\right. \mathtt{if}\, x\, \mathtt{then}\, e_t\, \mathtt{else}\, e_f : B}(\text{L:Cond}) \qquad \frac{}{\Sigma; x_1 : A_1, x_2 : A_2 \left|\frac{q}{q}\right. \langle x_1, x_2 \rangle : A_1 \times A_2}(\text{L:Pair})$$

$$\frac{\Sigma; \Gamma, x_1 : A_1, x_2 : A_2 \left|\frac{q}{q'}\right. e : B}{\Sigma; \Gamma, x : (A_1, A_2) \left|\frac{q}{q'}\right. \mathtt{case}\ x\ \{(x_1; x_2) \hookrightarrow e\} : B}(\text{L:MatP}) \qquad \frac{}{\Sigma; \emptyset \left|\frac{q}{q}\right. \mathtt{nil} : L^p(A)}(\text{L:Nil})$$

$$\frac{}{\Sigma; x_h : A, x_t : L^p(A) \left|\frac{q+p+1}{q}\right. \mathtt{cons}(x_h; x_t) : L^p(A)}(\text{L:Cons})$$

$$\frac{\Sigma; \Gamma \left|\frac{q}{q'}\right. e_1 : B \qquad \Sigma; \Gamma, x_h : A, x_t : L^p(A) \left|\frac{q+p+1}{q'}\right. e_2 : B}{\Sigma; \Gamma, x : L^p(A) \left|\frac{q}{q'}\right. \mathtt{case}\, x\, \{\mathtt{nil} \hookrightarrow e_1 \mid \mathtt{cons}(x_h; x_t) \hookrightarrow e_2\} : B}(\text{L:MatL})$$

$$\frac{\Sigma; \Gamma_1 \left|\frac{q}{p}\right. e_1 : A \qquad \Sigma; \Gamma_2, x : A \left|\frac{p}{q'}\right. e_2 : B}{\Sigma; \Gamma_1, \Gamma_2 \left|\frac{q}{q'}\right. \mathtt{let}(e_1; x : \tau.e_2) : B}(\text{L:Let}) \qquad \frac{\Sigma; \Gamma \left|\frac{q}{q'}\right. e : B}{\Sigma; \Gamma, x : A \left|\frac{q}{q'}\right. \mathtt{drop}(x; e) : B}(\text{L:Drop})$$

$$\frac{A \curlyvee A_1, A_2, 1 \qquad \Sigma; \Gamma, x_1 : A_1, x_2 : A_2 \left|\frac{q}{q'}\right. e : B}{\Sigma; \Gamma, x : A \left|\frac{q}{q'}\right. \mathtt{shareCopy}\ x\ \mathtt{as}\ x_1, x_2\ \mathtt{in}\ e : B}(\text{L:ShareCopy})$$

Where $A \curlyvee A_1, A_2, n$ is the sharing relation defined as:

$$\begin{aligned}
&L^p(A) \curlyvee L^q(A_1), L^r(A_2), n && \text{if } p = q + r + n \text{ and } A \curlyvee A_1, A_2, n \\
&A \times B \curlyvee A_1 \times A_2, B_1 \times B_2, n && \text{if } A \curlyvee A_1, A_2, n \text{ and } B \curlyvee B_1, B_2, n \\
&A \curlyvee A, A, n && \text{if } A \in \{\mathtt{unit}, \mathtt{bool}, \mathtt{nat}\}
\end{aligned}$$

Now if we take $\dagger : L^p(A) \mapsto L(A)$ as the map that erases resource annotations, we obtain a simpler typing judgement $\boxed{\Sigma^\dagger; \Gamma^\dagger \vdash e : B^\dagger}$.

8 Soundness for Linear GC

Definition 8.1 (Well-formed computation)**.** Given a configuration $\mathcal{C} = (V, H, R, F)$ and an expression e, we say the 5-tuple $(\mathcal{C}, e)$ is a *computation*; it is a *well-formed computation* given the following:

1. $dom(V) = FV(e)$
2. $\mathsf{no\_alias}(V, H)$, and
3. $\mathsf{disjoint}(\{R, F, locs_{V,H}(e)\})$

And we write $\mathsf{wfc}(V, H, R, F, e)$ to denote this fact.

Lemma 1.1. *If* $\Sigma; \Gamma \left|\frac{q}{q'}\right. e : B$*, then* $\Sigma^\dagger; \Gamma^\dagger \vdash e : B^\dagger$*.*

Lemma 1.2. *If* $\Sigma; \Gamma \left|\frac{q}{q'}\right. e : B$*, then* $\mathsf{set}(FV^\star(e))$ *and* $dom(\Gamma) = FV(e)$*.*

Proof. Induction on the typing judgement. □

Lemma 1.3. *Let* $H \vDash v \mapsto a : A$*. For all sets of locations* R*, if* $reach_H(v) \subseteq R$ *and* $\mathsf{stable}(R, H, H')$*, then* $H' \vDash v \mapsto a : A$ *and* $reach_H(v) = reach_{H'}(v)$*.*

Proof. Induction on the structure of *cst*. □

Corollary 1.3.1. *Let* $H \vDash V : \Gamma$*. For all sets of locations* R*, if* $\bigcup_{x \in V} reach_H(V(x)) \subseteq R$ *and* $\mathsf{stable}(R, H, H')$*, then* $H' \vDash V : \Gamma$*.*

Proof. Follows from Lemma 1.3. □

Lemma 1.4. *Let* $H \vDash v \mapsto a : A$*. If* $\mathtt{stack}(A)$*, then* $\Phi_H(v : A) = 0$*.*

Proof. Induction on $H \vDash v \mapsto a : A$. □

Lemma 1.5 (stability of copying)**.** *Let* $H', v' = copy(H, L, v)$*. For all* $l \in H$*, if* $l \notin L$*, then* $H(l) = H'(l)$*. Further,* $reach_{H'}(v') \subseteq L$*.*

Lemma 1.6 (copy is copy)**.** *Let* $H', v' = copy(H, L, v)$*. If* $H \vDash v \mapsto a : A$*, then* $H' \vDash v' \mapsto a : A$*.*

Lemma 1.7 (main lemma)**.** *For all stacks* V *and heaps* H*, let* $V, H, R, F \vdash e \Downarrow v, H', F'$ *and* $\Sigma; \Gamma \vdash e : B$*. Then given that* $\mathsf{wfc}(V, H, R, F, e)$*, we have the follwoing:*

1. $\mathsf{set}(reach_{H'}(v))$
2. $\mathsf{disjoint}(\{R, F', reach_{H'}(v)\})$*, and*
3. $\mathsf{stable}(R, H, H')$

To formally state the soundness theorem (and later the equivalence of free and copy semantics), we need the notion of context equivalence. Here we define it for contexts, which consisting of only the stack and heap. Later, we extend it the the full configuration. First, define *value* equivalence:

Definition 8.2 (Value Equivalence)**.** Two values v_1, v_2 are equivalent (with the presupposition that they are well-formed w.r.t heaps H_1, H_2), iff $H_1 \vDash v_1 \mapsto a : A$ and $H_2 \vDash v_2 \mapsto a : A$. Write value equivalence as $v_1 \sim^{H_1}_{H_2} v_2$.

Definition 8.3 (Context Equivalence)**.** Two simple contexts $(V_1, H_1), (V_2, H_2)$ are equivalent (with the presupposition that both are well-formed contexts) iff $dom(V_1) = dom(V_2)$ and for all $x \in dom(V_1)$, $V_1(x) \sim^{H_1}_{H_2} V_2(x)$. Write context equivalence as $(V_2, H_2) \sim (V_2, H_2)$

Stated simply, two contexts are equivalent when they have the same domain and equal variables bind equal semantic values.

Task 1.8 (Soundness)**.** *let* $H \vDash V : \Gamma$*,* $\Sigma; \Gamma \left|\frac{q}{q'}\right. e : B$*,* $V, H \ \vdash e \Downarrow v, H'$*, and* $H' \vDash v \mapsto a : A$*. Then* $\forall C \in \mathbb{Q}^+$ *and* $\forall F, R \subseteq \mathsf{Loc}$*, given the following (existence lemma):*

1. $\mathsf{wfc}(V, H, R, F, e)$
2. $|F| \geq \Phi_{V,H}(\Gamma) + q + C$

then there exists a context (W, Y)*, a value* w*, and a freelist* F' *s.t.*

1. $(W, Y) \sim (V, H)$
2. $W, Y, R, F \vdash e \Downarrow w, Y', F'$
3. $v \sim^{H'}_{Y'} w$
4. $|F'| \geq \Phi_{H'}(v : B) + q' + C$

Proof. Nested induction on the evaluation judgement and the typing judgement.

Case 1: E:Var

$V, H \ \vdash x \Downarrow V(x), H$ (admissibility)

$\Sigma; x : B \left|\frac{q}{q}\right. x : B$ (admissibility)

Let $C \in \mathbb{Q}^+, F, R \subseteq \mathsf{Loc}$ be arb.

Suppose this eval-config is well-formed, and further, $|F| \geq \Phi_{V,H}(x : B) + q + C$

Let $F' = F$. Then

$V, H, R, F \vdash e \Downarrow V(x), H, F'$ (E:Var)

And we have $F' = F \geq \Phi_{V,H}(x : B) + q + C$

$= \Phi_H(V(x) : B) + q + C$ (definition of Φ)

Case 2: E:Const* Due to similarity, we show only for E:ConstI

$|F| - |F'| = |F| - |F|$ (ad.)

$= 0$

$\Phi_{V,H}(\Gamma) + q - (\Phi_{H'}(v : B) + q') = \Phi_{V,H}(\emptyset) + q - (\Phi_H(v : int) + q)$ (ad.)

$= 0$ (def of $\Phi_{V,H}$)

$|F| - |F'| \leq \Phi_{V,H}(\Gamma) + q - (\Phi_{H'}(v : B) + q')$

Case 4: E:App

Case 5: E:CondT

$\Gamma = \Gamma', x : \texttt{bool}$ (ad.)

$H \vDash V : \Gamma'$ (def of W.F.E)

$\Sigma; \Gamma' \left|\frac{q}{q'}\right. e_t : B$ (ad.)

$V, H, R, F \cup g \ \vdash e_t \Downarrow v, H', F'$ (ad.)

$|F \cup g| - |F'| \leq \Phi_{V,H}(\Gamma) + q - (\Phi_{H'}(v : B) + q')$ (IH)

$|F| - |F'| \leq \Phi_{V,H}(\Gamma) + q - (\Phi_{H'}(v : B) + q')$

Case 6: E:CondF Similar to E:CondT

Case 7: E:Let

$V, H \vdash e \Downarrow v_2, H_2$ (case)

$V, H \vdash e_1 \Downarrow v_1, H_1$ (ad.)

$\Sigma; \Gamma_1 \frac{|q}{|p} e_1 : A$ (ad.)

$H \vDash V_1 : \Gamma_1$ (def of W.D.E)

Let $C \in \mathbb{Q}^+, F, R \subseteq \mathsf{Loc}$ be arb.

Suppose $dom(V) = FV(e), \mathsf{no\_alias}(V, H), \mathsf{disjoint}(\{R, F, locs_{V,H}(e)\}),$ and $|F| \geq \Phi_{V,H}(\Gamma) + q + C$

NTF F' s.t.

1.$V, H, R, F \vdash e \Downarrow v_2, H_2, F'$ and

2.$|F'| \geq \Phi_{H_2}(v_2 : B) + q' + C$

Let $R' = R \cup locs_{V,H}(\mathtt{lam}(x : \tau.e_2))$

$\mathsf{disjoint}(\{R', F, locs_{V,H}(e_1)\})$ (Similar to case in Lemma 1.7)

Instantiate IH with $C = C + \Phi_{V_2,H}(\Gamma_2), F = F, R = R',$ we get existence lemma on J_1 :

NTS (1) - (4) to instantiate existence lemma on J_1

(1) $dom(V_1) = FV(e_1)$

(2) $\mathsf{no\_alias}(V_1, H)$

(3) $\mathsf{disjoint}(\{R, F, locs_{V,H}(e)\})$ ((1) - (3) all verbatim as in Lemma 1.7)

(4) $|F| \geq \Phi_{V_1,H}(\Gamma_1) + q + C + \Phi_{V,H}(\Gamma_2)$ ($|F| \geq \Phi_{V,H}(\Gamma) + q + C$ and $\Phi_{V,H}(\Gamma) \geq \Phi_{V_1,H}(\Gamma_1) + \Phi_{V,H}(\Gamma_2)$)

Instantiating existence lemma on J_1, we get F'' s.t.

1.$V, H, R', F \vdash e_1 \Downarrow v_1, H_1, F''$ and

2.$|F''| \geq \Phi_{H_1}(v_1 : A) + p + C + \Phi_{V_2,H_1}(\Gamma_2)$

For the second premise:

$\Sigma; \Gamma_2, x : A \frac{|p}{|q'} e_2 : B$ (ad.)

$H_1 \vDash v_1 : A$ and (Theorem 3.3.4)

$H_1 \vDash V : \Gamma_2$ (???)

$H_1 \vDash V' : \Gamma_2, x : A$ (def of $\vDash$)

$V', H_1 \vdash e_2 \Downarrow v_2, H_2$ (ad.)

Let $g = \{l \in H_1 \mid l \notin F_1 \cup R \cup locs_{V',H_1}(e_2)\}$

Instantiate IH with $C = C, F = F'' \cup g, R = R,$ we get existence lemma on J_2 :

NTS (1) - (4) to instantiate existence lemma on J_1

(1) $dom(V_2') = FV(e_2)$

(2) $\mathsf{no\_alias}(V_2', H_1)$

(3) $\mathsf{disjoint}(\{R, F'' \cup g, locs_{V_2',H_1}(e_2)\})$ ((1) - (3) all verbatim as in Lemma 1.7)

(4) $|F'' \cup g| \geq \Phi_{V_2',H_1}(\Gamma_2, x : (A - 1)) + p + C$

STS $|F'' \cup g| \geq \Phi_{V_2,H_1}(\Gamma_2) + \Phi_{H_1}(v_1 : (A - 1)) + p + C$

$|F'' \cup g| \geq \|V_1\|_H + |F| - \|v_1\|_{H_1}$ (conservation lemma)

$\geq \Phi_{V,H}(\Gamma) + q + C + \|V_1\|_H - \|v_1\|_{H_1}$ ($|F| \geq \Phi_H(V) + q + C$)

STS $\Phi_{V_1,H}(\Gamma_1) + q + C \, \|V_1\|_H - \|v_1\|_{H_1} \geq \Phi_{H_1}(v_1 : (A - 1)) + p + C$

$\Phi_{V_1,H}(\Gamma_1) \geq \Phi_{H_1}(v_1 : (A - 1))$ (lemma about cf typing)

STS $\|V_1\|_H - \|v_1\|_{H_1} + q \geq p$ (done by aux lemma)

Instantiating existence lemma on J_2, we get $F^{(3)}$ s.t.

1.$V_2', H_1, R, F'' \cup g \vdash e_2 \Downarrow v_2, H_2, F^{(3)}$

2.$|F^{(3)}| \geq \Phi_{H_2}(v_2 : B) + q' + C$

Take $F' = F^{(3)}$

$V, H, R, F \vdash e \Downarrow v_2, H_2, F'$ and (E:Let)

$|F'| \geq \Phi_{H_2}(v_2 : B) + q' + C$ (from IH)

Case 8: E:Pair Similar to E:Const*

Case 9: E:MatP Similar to E:MatCons

Case 10: E:Nil Similar to E:Const*

Case 11: E:Cons

$V, H \vdash \mathtt{cons}(x_1; x_2) \Downarrow l, H'$ (case)

Let $C \in \mathbb{Q}^+, F, R \subseteq \mathsf{Loc}$ be arb.

Suppose $dom(V) = FV(e), \mathsf{no\_alias}(V, H), \mathsf{disjoint}(\{R, F, locs_{V,H}(e)\}), |F| \geq \Phi_{V,H}(\Gamma) + q + C$

NTF F' s.t.

1.$V, H, R, F \vdash e \Downarrow v, H', F'$ and

2.$|F'| \geq \Phi_{H'}(v : B) + q' + C$

Let $F' = F$

Case 12: E:MatNil Similar to E:Cond*

Case 13: E:MatCons

$V(x) = (l, \mathtt{alive})$ (ad.)

$H(l) = \langle v_h, v_t \rangle$ (ad.)

$\Gamma = \Gamma', x : L^p(A)$ (ad.)

$\Sigma; \Gamma', x_h : A, x_t : L^p(A) \Big|\frac{q+p+1}{q'}\, e_2 : B$ (ad.)

$V'', H \ \vdash e_2 \Downarrow v, H'$ (ad.)

Let $C \in \mathbb{Q}^+, F, R \subseteq \mathsf{Loc}$ be arb.

$H \vDash V(x) : L^p(A)$ (def of W.D.E)

$H'' \vDash v_h : A,\ H'' \vDash v_t : L^p(A)$ (ad.)

$H \vDash v_h : A,\ H \vDash v_t : L^p(A)$ (???)

$H \vDash V'' : \Gamma', x_h : A, x_t : L^p(A)$ (def of W.D.E)

Suppose $\mathsf{no\_alias}(V, H), \mathsf{disjoint}(\{R, F, locs_{V,H}(e)\}),$ and $|F| \geq \Phi_{V,H}(\Gamma) + q + C$

NTF F' s.t.

1.$V, H, R, F \vdash e \Downarrow v, H', F'$ and

2.$|F'| \geq \Phi_{H'}(v : B) + q' + C$

Let $g = \{l \in H \mid l \notin F \cup R \cup locs_{V'',H}(e_2)\}$

We want to g nonempty, in particular, that $l \in g$

$l \notin F \cup R$ ($\mathsf{disjoint}(\{R, F, locs_{V,H}(e)\})$)

AFSOC $l \in locs_{V'',H}(e_2)$

Then $l \in reach_H(\overline{V}''(x'))$ for some $x' \neq x$

$x' \in \{x_h, x_t\}$ (since $reach_H(\overline{V}(x')) \cap reach_H(\bar{(}Vx)) = \emptyset$ from $\mathsf{no\_alias}(V, H)$)

WLOG let $x' = x_h$

But then $\mu_{reach_H(\overline{V}(x))}(l) \geq 2$ and $\mathsf{set}(reach_(\overline{V}(x)))$ doesn't hold

$l \notin locs_{V'',H}(e_2)$

Hence $l \in g$

Next, we have $\mathsf{no\_alias}(V'', H)$ and $\mathsf{disjoint}(\{R, F \cup g, locs_{V'',H}(e_2)\})$ (similar to case in Lemma 1.2)

By IH with $C' = C, F'' = F \cup g$ and the above conditions, we have: $F^{(3)}$ s.t.

1.$V'', H, R, F \cup g \vdash e_2 \Downarrow v, H', F^{(3)}$

2.$|F^{(3)}| \geq \Phi_{H'}(v : B) + q' + C$

Where we also verify the precondition that $|F''| \geq \Phi_{V'',H}(\Gamma', x_h : A, x_t : L^p(A)) + q + p + 1 + C'$:

$$\begin{aligned}
|F''| &= |F \cup g| \\
&= |F| + |g| && (F \text{ and } g \text{ disjoint}) \\
&\geq \Phi_{V,H}(\Gamma) + q + C + |g| && \text{(Sp.)} \\
&= \Phi_{V,H}(\Gamma', x_h : A, x_t : L^p(A)) + p + q + C + |g| && \text{(Lemma 4.1.1)} \\
&= \Phi_{V,H}(\Gamma', x_h : A, x_t : L^p(A)) + p + q + C + 1 && (g \text{ nonempty})
\end{aligned}$$

Now take $F' = F^{(3)}$

$V, H, R, F \ \vdash e \Downarrow v, H', F'$ (E:MatCons)

$|F'| \geq \Phi_{H'}(v : B) + q' + C$ (From the IH)

Case 14: E:Share

$V, H \vdash e \Downarrow v, H''$ (case)

$V'[x_1 \mapsto v', x_2 \mapsto v''], H' \vdash e' \Downarrow v, H''$ (ad)

$\Sigma; \Gamma, x : A \left|\frac{q}{q'}\right. e : B$ (case)

$A \curlyvee A_1, A_2, 1$ (ad.)

$\Sigma; \Gamma, x_1 : A_1, x_2 : A_2 \left|\frac{q}{q'}\right. e : B$ (ad.)

Let $C \in \mathbb{Q}^+, F, R \subseteq \mathsf{Loc}$ be arb.

Suppose $\mathsf{no\_alias}(V, H), \mathsf{disjoint}(\{R, F, locs_{V,H}(e)\}),$ and $|F| \geq \Phi_{V,H}(\Gamma, x : A) + q + C$

NTF F'' s.t.

1.$V, H, R, F \vdash e \Downarrow v, H'', F''$ and

2.$|F''| \geq \Phi_{H''}(v : B) + q' + C$

We need to show the freelist is sufficient for the subsequent computation to invoke the IH:

Instantiate with $C, F \setminus L, \text{and} R$

STS $|F \setminus L| \geq \Phi_{V_2,H'}(\Gamma, x_1 : A_1, x_2 : A_2) + q + C$

$$\begin{aligned}
&\iff |F| - |L| \geq \Phi_{V_2,H'}(\Gamma) + \Phi_{V_2,H'}(x_1 : A_1) + \Phi_{V_2,H'}(x_2 : A_2) + q + C \\
&\iff |F| \geq \Phi_{V_2,H'}(\Gamma) + \Phi_{V_2,H'}(x_1 : A_1) + \Phi_{V_2,H'}(x_2 : A_2) + \left\|v'\right\|_H + q + C \\
&\iff |F| \geq \Phi_{V_2,H'}(\Gamma) + \Phi_{V,H}(x : A) + q + C && \text{(definition of sharing relation)} \\
&\iff |F| \geq \Phi_{V,H}(\Gamma, x : A) + q + C && \text{(stability of copying)}
\end{aligned}$$

done from assumption

By IH, we get F'' fulfilling the previous two points for the case.

$\square$

9 Copy-free garbage collection semantics

Consider the GC semantics (from now on copy semantics) above, with the share rule replaced with the following:

$$\frac{V = V'[x \mapsto v'] \qquad V'[x_1 \mapsto v', x_2 \mapsto v'], H', R, F \vdash e \Downarrow v, H'', F'}{V, H, R, F \vdash \texttt{share } x \texttt{ as } x_1, x_2 \texttt{ in } e \Downarrow v, H'', F'} \text{(F:Share)}$$

Call this new semantics free semantics for copy-free (all rules are renamed to F:_ for free). It is easy to see that any terminating compuation in copy has a corresponding one in free that can be instantiated with an equal or smaller freelist. Before formalizing this idea, we extend context equivalence to a preorder on configurations:

Definition 9.1. A configuration $\mathcal{C}_1 = (\mathcal{S}_1, R_1, F_1)$ is less than a configuration $\mathcal{C}_2 = (\mathcal{S}_2, R_2, F_2)$ iff

1. $|F_1| \leq |F_2|$
2. $R_1 = R_2$
3. $\mathcal{S}_1 \sim \mathcal{S}_2$

Write this as $\mathcal{C}_1 \leq \mathcal{C}_2$.

Now the lemma:

Lemma 1.9. *Let $\mathcal{C}_1 \vdash^{\mathsf{copy}} e \Downarrow v, H', F'$, and $H' \vDash v \mapsto a : A$. Then for all configurations $\mathcal{C}_2$ such that $\mathcal{C}_2 \leq \mathcal{C}_1$, there is exists a triple $(w, Y', M') \in \mathsf{Val} \times \mathsf{Heap} \times \mathsf{Loc}$ s.t.*

1. $\mathcal{C}_2 \vdash^{\mathsf{free}} e \Downarrow w, Y', M'$
2. $v \sim^{H'}_{Y'} w$
3. $|M'| \geq |F'|$

Proof. Induction on the evaluation judgement.

Case 1: E:Var

$V, H, R, F \vdash^{\mathsf{copy}} e \Downarrow V(x), H, F$ (case)

Suppose $H \vDash V(x) \mapsto a : A$. We need to show a triple that satisfies the 3 post-conditions.

Take $(v', H'', F'') = (V(x), H, F)$

(1) $V, H, R, F \vdash^{\mathsf{free}} e \Downarrow v', H'', F''$ (F:Var)

(2) $H'' \vDash v' \mapsto a : A$ (assumption)

(3) $|F''| = |F| \geq |F|$

Case 2: E:Const* Due to similarity, we show only for E:ConstI

Case 4: E:App

Case 5: E:CondT

$V, H, R, F \vdash^{\mathsf{copy}} \texttt{if}(x; e_1; e_2) \Downarrow v, H', F'$ (case)

Suppose $H' \vDash v \mapsto a : A$. We need to show a triple that satisfies the 3 post-conditions.

By IH, we have (v', H'', F'') such that

(1) $V', H, R, F \cup g \vdash^{\mathsf{free}} e_1 \Downarrow v', H'', F''$

(2) $H'' \vDash v' \mapsto a : A$

(3) $|F''| \geq |F'|$

Apply F:CondT to *(1)*, we are done.

Case 6: E:CondF Similar to E:CondT

Case 7: E:Let

Case 8: E:Pair Similar to E:Const*

Case 9: E:MatP Similar to E:MatCons

Case 10: E:Nil Similar to E:Const*

Case 11: E:Cons

Case 12: E:MatNil Similar to E:Cond*

Case 13: E:MatCons

Case 14: E:Share

$V, H, R, F \ \vdash^{\mathsf{copy}} \texttt{shareCopy}\ x\ \texttt{as}\ x_1, x_2\ \texttt{in}\ e \Downarrow v, H'', F'$ (case)

Suppose $H'' \vDash v \mapsto a : A$. We need to show a configuration and a triple that satisfies the 3 post-conditions. By IH, we have (w, K, M) such that

(1) $V_2, H', R, F \setminus L \vdash^{\mathsf{free}} e \Downarrow w, K, M$ (F:Var)

(2) $K \vDash w \mapsto a : A$

(3) $|M| \geq |F'|$

□

10 Examples

For brevity, we write the following examples using an extended syntax which is not in let-normal form, but which is equivalent to the restricted syntax in expressiveness and semantics. The program and signature is populated with top level let declarations.

append:

```
let rec append (l1, l2) =
  match l1 with
    | [] -> l2
    | x::xs -> let r = append (xs, l2) in x::r
```

$P = [\mathsf{append} \mapsto e_{\mathsf{append}}]$

$\Sigma = [\mathsf{append} \xrightarrow{q/q} L^p(A) \times L^p(A) \to \text{Ł}^p(A)]$

Type derivation:

$$\dfrac{\dfrac{}{\Sigma; l_2 : L^p(A) \Big|^{q}_{q}\ l_2 : L^p(A)}\ \text{L:Var} \qquad \dfrac{\dfrac{append : \xrightarrow{q/q} L^p(A) \times L^p(A) \to \text{Ł}^p(A) \in \Sigma}{\Sigma; L_2 : L^p(A), xs : L^p(A) \Big|^{q+p+1}_{q+p+1}\ append(xs, l_2) : L^p(A)}\ \text{L:App} \qquad \dfrac{}{\Sigma; x : A, r : L^p(A) \Big|^{q+p+1}_{q}\ x :: r : L^p(A)}\ \text{L:Cons}}{\Sigma; l_2 : L^p(A), x : A, xs : L^p(A) \Big|^{q+p+1}_{q}\ \texttt{let}\ r = append(xs, l_2)\ \texttt{in}\ x :: r : L^p(A)}\ \text{L:Let}}{\Sigma; l_1 : L^p(A), l_2 : L^p(A) \Big|^{q}_{q}\ \texttt{case}\ l_1\ \{\texttt{nil} \hookrightarrow l_2 \mid \texttt{cons}(x; xs) \hookrightarrow \texttt{let}\ r = append(xs, l_2)\ \texttt{in}\ x :: r\} : L^p(A)}\ \text{L:MatL}$$

This can be read as append takes two lists, each with potential p per element, and a constant potential q, and returns a list with potential p per element and constant potential q. This bound is tight and reflects the fact that append is constructing the concatenation "in place" by collecting the cells in l_1. Thus, append induces no overhead heap cells in addition to its arguments.

Now we show that quicksort is also has no overhead. First, the partition:

```
let rec partition (p, l) =
  match l with
    | [] -> ([],[])
    | x::xs ->
      let (l1,l2) = partition xs in
      let r = x < p in
      if r then
        (x::l1,l2)
      else
        (l1,x::l2)
```

$P = [partition \mapsto e_{partition}]$
$\Sigma = [partition \mapsto \texttt{nat} \times L^p(\texttt{nat}) \to L^p(\texttt{nat}) \times L^p(\texttt{nat})]$
The type derivation for the Cons branch:

$$\frac{}{\Sigma; p : \texttt{nat}, x : \texttt{nat}, xs : L^p(\texttt{nat}) \vdash \texttt{let}\ r = \ \texttt{in}\ x :: r}$$

```
let rec quicksort l =
match l with
  | [] -> []
  | x::xs ->
      let ys, zs = partition (x,xs) in
let l1 = quicksort ys in
let l2 = quicksort zs in
let r = x :: l2 in
      append (l1, r)
```

map:

```
let rec map (f,l) =
match l with
| [] -> []
| x::xs -> let r = f x in r :: map (f, xs)
```

$P = [\texttt{map} \mapsto e_{\texttt{map}}]$
$\Sigma = [\texttt{map} \mapsto A \to B \times L^0(A) \to Ł^0(B)]$

Type derivation: