15-819: Resource Aware Programming Language

Lecture 19-20: AARA with Multivariate Polynomial Potential

Jan Hoffmann

November 5, 2019

1 Introduction

This lecture introduces AARA with multivariate polynomial potential [HAH11, HDW17]. There are two motivations for extending univariate AARA. First, many algorithms have bounds like $n \cdot m$, which cannot be expressed with univariate AARA. Second, even for algorithms that have univariate bounds like n^2, more complex mixed potential like $n \cdot m$ is often needed for intermediate potential in the program.

Multivariate AARA is an extension of univariate AARA. It preserves the principles of the univariate system while expanding the set of potential functions so as to express a wide range of dependencies between different data structures. Section 3 introduces *resource polynomials*, the multivariate potential functions that I use in this lecture. In Section 4, we see how types can be annotated with resource polynomials. Other than in univariate, potential annotations are now global and there is exactly one annotation per type. Section 6 contains multivariate shift operations as well as type rules that are used to derive annotated type judgments. In Section 7, we state the soundness theorem.

2 Motivating Example

The univariate polynomial potential that we introduced is not as compositional as we might expect, even for programs that only have a single input.

Consider for example the function *quad_inner*.

```
fun id1 l =
  match l with
  | [] → []
  | x::xs →
     let _ = tick 1.0 in
     let xs’ = id1 xs in
     x::xs’

fun map f l =
  match l with
  | [] → []
  | x::xs →
     let x’ = f x in
     let xs’ = map f xs in
     x’::xs’

fun append (l1,l2) =
  match l1 with
  | [] → l2
  | x::xs →
```

```
        let r = append (xs,l2) in
        x::r

   fun quad l =
     match l with
     | [] → []
     | x::xs →
        share xs as xs1, xs2 in
        let xs' = quad xs1 in
        append (xs2, xs')

   fun quad_inner xs =
     let xs' = quad xs in
     map id1 xs'
```

The only cost in the code originates from the function *id1*, which can be assigned the following typing.

$$id1 : \langle L^{(1,0)}(\mathbf{1}), 0\rangle \to \langle L^{(0,0)}(\mathbf{1}), 0\rangle$$

The function *map* is used with argument *id1* and applied to lists of lists. It has the following type.

$$map : (\langle L^{(1,0)}(\mathbf{1}), 0\rangle \to \langle L^{(0,0)}(\mathbf{1}), 0\rangle) \to \langle L^{(0,0)}(L^{(1,0)}(\mathbf{1})), 0\rangle \to \langle L^{(0,0)}(L^{(0,0)}(\mathbf{1})), 0\rangle$$

Now consider the function *quad l* that computes a list of size $\binom{|l|}{2}$ that contains the ith element of l $i-1$ times. Finally, the function *quad_inner* takes a lists of lists of units, applies the function *quad* to the argument and then applies *id1* to each inner list of the result.

Let us now analyze the resource consumption of *quad_inner(l)*, where $l = [v_1, \dots, v_n]$. Since we apply *id1* to each inner list we need potential $|v_i|$ for each inner list v_i, that is, the type of *xs'* should be $L^{(0,0)}(L^{(1,0)}(\mathbf{1}))$. Given that *quad* copies the i-th element of the argument $i-1$ times, we would need a potential like $(i-1)|v_i|$ for every inner list v_i. However, we cannot express such a type in univariate AARA.

In with univariate potential, there seems a mismatch in the way the outer potential and the inner potential is handled during pattern matching. In multivariate AARA, inner and outer potential is treated uniformly and we can derive the aforementioned types.

3 Resource Polynomials

A resource polynomial maps a value of a some type to a non-negative rational number. Potential functions in this section are given by such resource polynomials. The types we consider in this lecture are given as follows.

$$\begin{array}{lcll}
\tau & ::= & \text{unit} & \mathbf{1} \\
 & & \text{arr}(A;B) & A \to B \\
 & & \text{prod}(\tau_1;\tau_2) & \tau_1 \times \tau_2 \\
 & & L(\tau) & L(\tau) \\
 & & & \\
A, B & ::= & \text{pot}(\tau;Q) & \langle \tau, Q\rangle
\end{array}$$

In contrast to univariate AARA there is a global potential annotation $Q = (q_i)_{i \in I}$ that is a family of non-negative rational numbers $q_i \in \mathbb{Q}_{\geq 0}$. We are constructing Q step-by-step and start with the potential functions that we want to represent.

An analysis of typical polynomial computations operating on a list $v = [a_1, \dots, a_n]$ shows that it consists of operations that are executed for every k-tuple $(a_{i_1}, \dots, a_{i_k})$ with $1 \leq i_1 < \cdots < i_k \leq n$. The simplest examples are linear map operations that perform some operation for every a_i. Other examples are common sorting algorithms that perform comparisons for every pair (a_i, a_j) with $1 \leq i < j \leq n$ in the worst case.

The expressions of the language are defined later. We define values with the following grammar as before. Function values are *function closures* that consist of an environment and a (recursive) lambda abstraction. The definition is mutually recursive with the definition of environments.

$$
\begin{array}{rcl}
v & ::= & \langle\rangle \\
 & & [\,] \\
 & & v_1 :: v_2 \\
 & & \langle v_1, v_2\rangle \\
 & & \mathrm{clo}(V; f, x.e) \\
 & & \\
V & ::= & \cdot \\
 & & V, x \mapsto v
\end{array}
$$

We can define the set of values for each type as follows.

$$[\![\tau]\!] = \{v \mid v : \tau\}$$

We do not give the rules for the judgment $v : \tau$. They are similar to the ones for linear AARA.

Base Polynomials For each type τ, we define a set $\mathcal{B}(\tau)$ of functions $p : [\![\tau]\!] \to \mathbb{N}$ that map values of type τ to a natural number. The resource polynomials for type τ are then given as non-negative linear combinations of these *base polynomials*. We define $\mathcal{B}(\tau)$ as follows.

$$
\begin{array}{rcl}
\mathcal{B}(\mathrm{unit}) & = & \{\lambda(v)\,1\} \\
\mathcal{B}(A \to B) & = & \{\lambda(v)\,1\} \\
\mathcal{B}(\tau_1 \times \tau_2) & = & \{\lambda(\langle v_1, v_2\rangle)\, p_1(v_1)\cdot p_2(v_2) \mid p_i \in \mathcal{B}(\tau_i)\} \\
\mathcal{B}(L(\tau)) & = & \big\{\lambda([v_1,\ldots,v_n]) \displaystyle\sum_{1\le j_1<\cdots<j_k\le n} \prod_{1\le i\le k} p_i(v_{j_i}) \mid k \in \mathbb{N}, p_i \in \mathcal{B}(\tau)\big\}
\end{array}
$$

For every τ, the set $\mathcal{B}(\tau)$ contains the constant function $\lambda(v)\,1$. In the case of $L(\tau)$ this arises for $k = 0$ (one element sum, empty product).

For example, the function $\lambda(\ell)\binom{|\ell|}{k}$ is in $\mathcal{B}(L(\tau))$ for every $k \in \mathbb{N}$; simply take $p_1 = \ldots = p_k = 1$ in the definition of $\mathcal{B}(L(\tau))$. The function $\lambda(\langle \ell_1, \ell_2\rangle)\binom{|\ell_1|}{k_1}\cdot\binom{|\ell_2|}{k_2}$ is in $\mathcal{B}(L(\tau_1), L(\tau_2))$ for every $k_1, k_2 \in \mathbb{N}$, and the function $\lambda([\ell_1,\ldots,\ell_n]) \sum_{1\le i<j\le n}\binom{|\ell_i|}{k_1}\cdot\binom{|\ell_j|}{k_2}$ is in $\mathcal{B}(L(L(\tau)))$ for every $k_1, k_2 \in \mathbb{N}$.

Example 1. *The resource bound for the function* quad_inner *can be given by the base polynomial* $\lambda([\ell_1,\ldots,\ell_n]) \sum_{1\le i<j\le n}\binom{|\ell_j|}{2}$.

Resource Polynomials A *resource polynomial* $p : [\![\tau]\!] \to \mathbb{Q}_{\ge 0}$ is a non-negative linear combination of base polynomials, that is,

$$p(v) = \sum_{i=1,\ldots,m} q_i \cdot p_i(v)$$

for $q_i \in \mathbb{Q}_{\ge 0}$ and $p_i \in \mathcal{B}(\tau)$. We write $\mathcal{R}(\tau)$ for the set of resource polynomials for type τ.

An instructive, but not exhaustive, example is given by $\mathcal{R}_n = \mathcal{R}(L(\mathrm{unit}) \times \cdots \times L(\mathrm{unit}))$. The set $\mathcal{R}_n$ is the set of linear combinations of products of binomial coefficients over variables $x_1,\ldots,x_n$, that is,

$$\mathcal{R}_n = \{\sum_{i=1}^{m} q_i \prod_{j=1}^{n}\binom{x_j}{k_{ij}} \mid q_i \in \mathbb{Q}_{\ge 0}, m \in \mathbb{N}, k_{ij} \in \mathbb{N}\}\,.$$

These expressions generalize the univariate polynomials of univariate polynomial AARA and meet two conditions that are important to efficiently manipulate polynomials during the analysis. Firstly, the polynomials are non-negative, and secondly, they are closed under the discrete difference operators Δ_i for every i. The discrete derivative $\Delta_i\, p$ is defined through $\Delta_i\, p(x_1,\ldots,x_n) = p(x_1,\ldots,x_i+1,\ldots,x_n) - p(x_1,\ldots,x_n)$.

It can be shown that $\mathcal{R}_n$ is the largest set of polynomials enjoying these closure properties. It would be interesting to have a similar characterisation of $\mathcal{R}(A)$ for arbitrary A. So far, we know that $\mathcal{R}(A)$ is closed under sum and product (see Lemma 1) and are compatible with the construction of elements of data structures in a very natural way (see Lemmas 2 and 3). This provides some justification for their choice and canonicity. An abstract characterization would have to take into account the fact that our resource polynomials depend on an unbounded number of variables, e.g., sizes of inner data structures, and are not invariant under permutation of these variables.

4 Annotated Types

The resource polynomials described in Section 3 are non-negative linear combinations of base polynomials. The rational coefficients of the linear combination are present as type annotations in our type system. To relate type annotations to resource polynomials we systematically *name* base polynomials and resource polynomials for data of a given type.

If one considers only univariate polynomials then their description is straightforward. Every list of size n has a potential of the form $\sum_{1\le i\le k} q_i\binom{n}{i}$. So we can describe the potential function with a vector $\vec{q} = (q_1,\dots,q_k)$ in the corresponding list type $L^{\vec{q}}(\tau)$. Since each annotation refers to one size parameter only, univariate annotated types can be directly composed. For example, an annotated type for a pair of lists has the form $(L^{\vec{q}}(\tau_1), L^{\vec{p}}(\tau_2))$.

In this lecture, we use multivariate potential functions, that is, functions that depend on the sizes of different parts of the input. For a pair of lists of lengths n and m we have, for instance, a potential function of the form $\sum_{0\le i+j\le k} q_{ij}\binom{n}{i}\binom{m}{j}$, which can be described by the coefficients q_{ij}. Potential functions can also refer to the sizes of different lists inside a list of lists, etc. That is why we need to develop a set of indexes $\mathcal{I}(\tau)$ that enumerate the basic resource polynomials p_i and the corresponding coefficients q_i for a type τ.

Names For Base Polynomials To assign a unique name to each base polynomial I define the *index set* $\mathcal{I}(\tau)$ to denote resource polynomials for a given type τ. Interestingly, but maybe coincidentally, $\mathcal{I}(\tau)$ is essentially the semantics of τ with arrow types replaced by unit.

$$\begin{aligned}
\mathcal{I}(\text{unit}) &= \{*\}\\
\mathcal{I}(A \to B) &= \{*\}\\
\mathcal{I}(\tau_1 \times \tau_2) &= \{(i_1,i_2) \mid i_1 \in \mathcal{I}(\tau_1) \text{ and } i_2 \in \mathcal{I}(\tau_2)\}\\
\mathcal{I}(L(\tau)) &= \{[i_1,\dots,i_k] \mid k \ge 0, i_j \in \mathcal{I}(\tau)\}
\end{aligned}$$

The *degree* $\deg(i)$ of an index $i \in \mathcal{I}(\tau)$ is defined as follows.

$$\begin{aligned}
\deg(*) &= 0\\
\deg(i_1,i_2) &= \deg(i_1) + \deg(i_2)\\
\deg([i_1,\dots,i_k]) &= k + \deg(i_1) + \cdots + \deg(i_k)
\end{aligned}$$

Define $\mathcal{I}_k(\tau) = \{i \in \mathcal{I}(\tau) \mid \deg(i) \le k\}$. The indexes $i \in \mathcal{I}_k(\tau)$ are an enumeration of the base polyonomials $p_i \in \mathcal{B}(\tau)$ of degree at most k. For each $i \in \mathcal{I}(\tau)$, we define a base polynomial $p_i \in \mathcal{B}(\tau)$ as follows: If $\tau = \text{unit}$ or $\tau = A \to B$ for some A, B then

$$p_*(v) = 1\,.$$

If $\tau = \tau_1 \times \tau_2$ is a product type and $v = \langle v_1, v_2\rangle$ then

$$p_{(i_1,i_2)}(\langle v_1,v_2\rangle) = p_{i_1}(v_1)\cdot p_{i_2}(v_2)\,.$$

If $\tau = L(\tau')$is a list $v = [v_1,\dots,v_n]$ then

$$p_{[i_1,\dots,i_k]}(v) = \sum_{1\le j_1<\cdots<j_k\le n} p_{i_1}(v_{j_1})\cdots p_{i_k}(v_{j_k})\,.$$

I use the notation 0_τ (or just 0) for the index in $\mathcal{I}(\tau)$ such that $p_{0_A}(v) = 1$ for all v. We have $0_{\text{unit}} = *$ and $0_{\tau_1\times\tau_2} = (0_{\tau_1}, 0_{\tau_2})$ and $0_{L(\tau)} = []$. If $\tau = L(\tau')$ for τ' a type then the index $[0,\dots,0] \in \mathcal{I}(\tau)$ of length n is denoted by just n. We identify the index (i_1,i_2,i_3,i_4) with the index $(i_1,(i_2,(i_3,i_4)))$.

For a list $i = [i_1, \dots, i_k]$ I write $i_0 :: i$ to denote the list $[i_0, i_1, \dots, i_k]$. Furthermore, I write $i i'$ for the concatenation of two lists i and i'.

Recall that $\mathcal{R}(\tau)$ denotes the set of nonnegative rational linear combinations of the base polynomials.

Lemma 1. *Let $p, p' \in \mathcal{R}(\tau)$ be resource polynomials. Then we have $p + p' \in \mathcal{R}(\tau)$, $p \cdot p' \in \mathcal{R}(\tau)$. Moreover,* $\deg(p + p') = \max(\deg(p), \deg(p'))$, *and* $\deg(p \cdot p') = \deg(p) + \deg(p')$.

Proof. By linearity it suffices to show this lemma for base polynomials. For them, the claim follows by structural induction. □

Corollary 1. *For every $p \in \mathcal{R}(\tau, A)$ there exists $p' \in \mathcal{R}(\tau)$ with* $\deg(p') = \deg(p)$ *and $p'(a) = p(a, a)$ for all $a \in [\![\tau]\!]$.*

Proof. The proof follows directly from Lemma 1 noticing that base polynomials $p \in \mathcal{B}(\tau, A)$ take the form $p_i \cdot p_{i'}$. □

Lemma 2. *Let $v \in [\![\tau]\!]$ and $\ell \in [\![L(\tau)]\!]$ be a list. Let furthermore $k \geq 0$ and let $i_0, \dots, i_k \in \mathcal{I}(\tau)$ indexes for type τ. Then we have*

$$\begin{array}{rcl} p_{[i_0, i_1, \dots, i_k]}([]) & = & 0 \\ p_{[i_0, i_1, \dots, i_k]}(v :: \ell) & = & p_{(i_0, [i_1, \dots, i_k])}(\langle v, \ell \rangle) + p_{(0_\tau, [i_1, \dots, i_k])}(\langle v, \ell \rangle) . \end{array}$$

Proof. Let $\ell = [v_1, \dots, v_n]$. Writing v_0 for v we compute as follows.

$$\begin{array}{rcl} p_{[i_0, i_1, \dots, i_k]}(v :: \ell) & = & \displaystyle\sum_{0 \leq j_0 < j_1 < \cdots < j_k \leq n} p_{i_0}(v_{j_0}) \cdot p_{i_1}(v_{j_1}) \cdots p_{i_k}(v_{j_k}) \\ & = & \displaystyle\sum_{1 \leq j_1 < \cdots < j_k \leq n} p_{i_0}(v_0) \cdot p_{i_1}(v_{j_1}) \cdots p_{i_k}(v_{j_k}) \\ & + & \displaystyle\sum_{1 \leq j_0 < j_1 < \cdots < j_k \leq n} p_{i_0}(v_{j_0}) \cdot p_{i_1}(v_{j_1}) \cdots p_{i_k}(v_{j_k}) \\ & = & p_{i_0}(v) \cdot \displaystyle\sum_{1 \leq j_1 < \cdots < j_k \leq n} p_{i_1}(v_{j_1}) \cdots p_{i_k}(v_{j_k}) \\ & + & \displaystyle\sum_{1 \leq j_0 < j_1 < \cdots < j_k \leq n} p_{i_0}(v_{j_0}) \cdot p_{i_1}(v_{j_1}) \cdots p_{i_k}(v_{j_k}) \\ & = & p_{i_0}(v) \cdot p_{[i_1, \dots, i_k]}(\ell) + p_0(v) \cdot p_{[i_0, i_1, \dots, i_k]}(\ell) \end{array}$$

The statement $p_{[i_0, i_1, \dots, i_k]}([]) = 0$ is obvious as the sum in the definition of the corresponding base polinomial is over the empty index set. □

Lemma 3 characterizes concatenations of lists (written as juxtaposition) as they could occur in the construction of tree-like data.

Lemma 3. *Let $\ell_1, \ell_2 \in [\![L(\tau)]\!]$ be lists of type τ. Then we have $\ell_1 \ell_2 \in [\![L(\tau)]\!]$ and $p_{[i_1, \dots, i_k]}(\ell_1 \ell_2) = \sum_{t=0}^{k} p_{[i_1, \dots, i_t]}(\ell_1) \cdot p_{[i_{t+1}, \dots, i_k]}(\ell_2)$ for all indexes $i_j \in \mathcal{I}(\tau)$.*

This can be proved by induction on the length of ℓ_1 using Lemma 2 or else by a decomposition of the defining sum according to which indices hit the first list and which ones hit the second.

Annotated Types and Potential Functions A *type annotation* for a type τ is defined to be a family

$$Q_\tau = (q_i)_{i \in \mathcal{I}(\tau)} \text{ with } q_i \in \mathbb{Q}_{\geq 0} .$$

We say that Q_τ is of *degree (at most) k* if $q_i = 0$ for every $i \in \mathcal{I}(\tau)$ with $deg(i) > k$. An *annotated type* is a pair $\langle \tau, Q_\tau \rangle$ of a type τ and a type annotation Q_τ.

Let $v : \tau$ be a value of type τ. Then the type annotation Q_τ defines the *potential*

$$\Phi(v{:}\langle \tau, Q_\tau \rangle) = \sum_{i \in \mathcal{I}(\tau)} q_i \cdot p_i(v) .$$

I usually define type annotations Q_τ by only stating the values of the non-zero coefficients q_i. However, it is sometimes handy to write annotations $(q_0, \dots, q_n)$ for a list of atomic types just as a vector. Similarly, I write annotations $(q_0, q_{(1,0)}, q_{(0,1)}, q_{(1,1)}, \dots)$ for pairs of lists of atomic types sometimes as a triangular matrix.

Examples

Unit The simplest annotated types are those for atomic types like unit. The indexes for unit are $\mathcal{I}(\text{unit}) = \{*\}$ and thus each type annotation has the form (unit, q_0) for a $q_0 \in \mathbb{Q}_{\geq 0}$. It defines the constant potential function $\Phi(v{:}\langle\text{unit}, q_0\rangle) = q_0$.

Simple Products Similarly, tuples of atomic types feature a single index of the form $(*, \dots, *)$ and a constant potential function defined by some $q_{(*,\dots,*)} \in \mathbb{Q}_{\geq 0}$.

Unit Lists More interesting examples are lists of atomic types like, that is, $L(\text{unit})$. The set of indexes of degree k is then

$$\mathcal{I}_k(L(\text{unit})) = \{[], [*], [*,*], \dots, [*,\dots,*]\}$$

where the last list contains k unit elements. Since we identify a list of i unit elements with the integer i we have $\mathcal{I}_k(L(\text{unit})) = \{0, 1, \dots, k\}$. Consequently, annotated types have the form $(L(\text{unit}), (q_0, \dots, q_k))$ for $q_i \in \mathbb{Q}_{\geq 0}$. The defined potential function is

$$\Phi([v_1, \dots, v_n]{:}\langle L(\text{unit}), (q_0, \dots, q_k)\rangle = \sum_{0 \leq i \leq k} q_i \binom{n}{i}$$

.

Pairs of Unit Lists The next example is the type $prodTL(\text{unit})li\text{unit}$ of pairs of unit lists. The set of indexes of degree k is

$$\mathcal{I}_k(L(\text{unit}) \times L(\text{unit})) = \{(i,j) \mid i + j \leq k\}$$

if we identify lists of unit indexes with their lengths i and j as usual. Annotated types are then of the from $(L(\text{unit}) \times L(\text{unit}), Q)$ for a triangular $k \times k$ matrix Q with non-negative rational entries. If $\ell_1 = [a_1, \dots, a_n]$, $\ell_2 = [b_1, \dots, b_m]$ are two lists then the potential function is

$$\Phi(\langle \ell_1, \ell_2 \rangle), \langle L(\text{unit}) \times L(\text{unit}), (q_{(i,j)})\rangle) = \sum_{0 \leq i+j \leq k} q_{(i,j)} \binom{n}{i}\binom{m}{j}.$$

Lists of Lists Finally, consider the type $\tau = L(L(\text{unit}))$ of lists of lists of units. The set of indexes of degree k is then

$$\mathcal{I}_k(L(L(\text{unit}))) = \Big\{[i_1, \dots, i_m] \mid m \leq k, i_j \in \mathbb{N}, \sum_{j=1,\dots,m} i_j \leq k - m\Big\}.$$

Thus we have $\mathcal{I}_k(L(L(\text{unit}))) = \{0, \dots, k\} \cup \{[1], \dots, [k-1]\} \cup \{[0,1], \dots\} \cup \cdots$. Let for instance $\ell = [[a_{11}, \dots, a_{1m_1}], \dots, [a_{n1}, \dots, a_{nm_n}]]$ be a list of lists and $Q = (q_i)_{i \in \mathcal{I}_k(L(L(\text{unit})))}$ be a corresponding type annotation. The defined potential function is then

$$\Phi(\ell, \langle L(L(\text{unit})), Q\rangle) = \sum_{[i_1,\dots,i_l] \in \mathcal{I}_k(\tau)} \sum_{1 \leq j_1 < \dots < j_l \leq n} q_{[i_1,\dots,i_l]} \binom{m_{j_1}}{i_1} \cdots \binom{m_{j_l}}{i_l}.$$

In practice the potential functions are usually not very complex since most of the q_i are zero.

The Potential of a Context For use in the type system, we have to extend the definition of resource polynomials to typing contexts. We treat a context like a product type.

Let $\Gamma = x_1{:}\tau_1, \ldots, x_n{:}\tau_n$ be a typing context and let $k \in \mathbb{N}$. The index set $\mathcal{I}(\Gamma)$ is defined through

$$\mathcal{I}_k(\Gamma) = \big\{(i_1, \ldots, i_n) \mid i_j \in \mathcal{I}_{m_j}(\tau_j)\big\} .$$

The index set $\mathcal{I}_k(\Gamma)$ is defined through

$$\mathcal{I}_k(\Gamma) = \big\{(i_1, \ldots, i_n) \mid i_j \in \mathcal{I}_{m_j}(\tau_j), \sum_{j=1,\ldots,n} m_j \le k\big\} .$$

A *type annotation* Q of degree k for Γ is a family

$$Q = (q_i)_{i \in \mathcal{I}_k(\Gamma)} \text{ with } q_i \in \mathbb{Q}_{\ge 0} .$$

We denote a *resource-annotated context* with $\Gamma; Q$. Let $V : \Gamma$ be a well-formed environment with respect to Γ. The potential of V under $\Gamma; Q$ is

$$\Phi_V(\Gamma; Q) = \sum_{(i_1,\ldots,i_n) \in \mathcal{I}_k(\Gamma)} q_{\vec{\imath}} \prod_{j=1}^{n} p_{i_j}(V(x_j))$$

In particular, if $\Gamma = \emptyset$ then $\mathcal{I}_k(\Gamma) = \{()\}$ and $\Phi_V(\Gamma; q_{()}) = q_{()}$. I sometimes also write q_0 for $q_{()}$.

5 Syntax and Cost Semantics

The expressions we consider are defined as follows.

e	$::=$	x	x
		triv	$\langle\rangle$
		$\text{app}(x_1; x_2)$	$x_1(x_2)$
		$\text{fun}(f, x.e)$	$\mathsf{fun}\ f\, x = e$
		nil	$[]$
		$\text{cons}(x_1; x_2)$	$x_1 :: x_2$
		$\text{mat}_{\text{L}}\{e_0; x_1, x_2.e_1\}(x)$	$\mathsf{case}\ x\ \{\mathsf{nil} \hookrightarrow e_0 \mid \mathsf{cons}(x_1, x_2) \hookrightarrow e_1\}$
		$\text{tick}\{q\}$	$\mathsf{tick}\ q$
		$\text{let}(e_1; x.e_2)$	$\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$
		$\text{share}(x; x_1, x_2.e)$	$\mathsf{share}\ x\ \mathsf{as}\ x_1, x_2\ \mathsf{in}\ e$
		$\text{pair}(x_1; x_2)$	$\langle x_1, x_2\rangle$
		$\text{letp}(x; x_1, x_2.e)$	$\mathsf{letp}\ \langle x_1, x_2\rangle = x\ \mathsf{in}\ e$

The cost dynamics with the judgment

$$V \vdash e \Downarrow v \mid (q, q')$$

can be defined in the same way as before.

6 Static Semantics

Type Judgments The declarative type rules define a *multivariate resource-annotated typing judgment* of the form

$$\Gamma; Q \vdash e {:} \langle\tau, Q'\rangle$$

where e is an expression, $\Gamma; Q$ is a resource-annotated context, and $\langle\tau, Q'\rangle$ is a resource-annotated type. The intended meaning of this judgment is that if there are more than $\Phi(\Gamma; Q)$ resource units available then this is sufficient to pay for the cost of the evaluation e. In addition, there are more than $\Phi(v{:}(\tau, Q'))$ resource units left if e evaluates to a value v.

The type rules are given in Figure 1.

$$\frac{}{x{:}\tau; Q \vdash x : \langle \tau, Q \rangle}\;(\textsc{M:Var}) \qquad \frac{}{\cdot; Q \vdash \mathrm{triv} : \langle \mathrm{unit}, Q \rangle}\;(\textsc{M:Unit})$$

$$\frac{A = \langle \tau, P \rangle \qquad P = \pi_0^{x_2:\tau}(Q)}{x_1 : A \to B, x_2 : \tau; Q \vdash \mathrm{app}(x_1; x_2) : B}\;(\textsc{M:App})$$

$$\frac{A = \langle \tau, P \rangle \qquad \pi_{\vec{0}}^{x:\tau}(Q) = P \qquad \forall\, i \neq \vec{0} : \pi_i^{x:\tau}(Q) = 0 \qquad \Gamma, f : A \to B, x : \tau; Q \vdash e : B}{\Gamma; 0 \vdash \mathrm{fun}(f, x.e) : \langle A \to B, 0 \rangle}\;(\textsc{M:Fun})$$

$$\frac{\Gamma, x_1{:}\tau_1, x_2{:}\tau_2; Q \vdash e : B}{\Gamma, x{:}\tau_1 \times \tau_2; Q \vdash \mathrm{letp}(x; x_1, x_2.e) : B}\;(\textsc{M:LetP})$$

$$\frac{}{x_1{:}\tau_1, x_2{:}\tau_2; Q \vdash \mathrm{pair}(x_1; x_2) : \langle \tau_1 \times \tau_2, Q \rangle}\;(\textsc{M:Pair}) \qquad \frac{q_0 = q'_{\vec{0}}}{\emptyset; Q \vdash \mathrm{nil} : (L(\tau), Q')}\;(\textsc{M:Nil})$$

$$\frac{Q = \lhd_L(Q')}{x_1{:}\tau, x_2{:}L(\tau); Q \vdash \mathrm{cons}(x_1; x_2) : (L(\tau), Q')}\;(\textsc{M:Cons})$$

$$\frac{R = \pi_0^{\Gamma}(Q) \qquad \Gamma; R \vdash e_1 : B \qquad P = \lhd_L(Q) \qquad \Gamma, x_1{:}\tau, x_2{:}L(\tau); P \vdash e_2 : B}{\Gamma, x{:}L(\tau); Q \vdash \mathrm{mat_L}\{e_1; x_1, x_2.e_2\}(x) : B}\;(\textsc{M:MatL})$$

$$\frac{\begin{array}{l} P = \pi_{\vec{0}}^{\Gamma_1}(Q) \qquad \Gamma_1; P \vdash e_1 : (\tau, P') \qquad \Gamma_2, x{:}\tau; R \vdash e_2 : B \qquad P' = \pi_{\vec{0}}^{x:\tau}(R) \\ \forall \vec{0} \neq j \in \mathcal{I}(\Gamma_2){:}\;\; \Gamma_1; P_j \overset{\mathrm{cf}}{\vdash} e_1 : (\tau, P'_j) \qquad P_j = \pi_j^{\Gamma_1}(Q) \qquad P'_j = \pi_j^{x:\tau}(R) \end{array}}{\Gamma_1, \Gamma_2; Q \vdash \mathrm{let}(e_1; x.e_2) : B}\;(\textsc{M:Let})$$

$$\frac{\Gamma, x_1 : \tau_1, x_2 : \tau_2; P \vdash e : B \qquad Q = \curlyvee(P)}{\Gamma, x : \tau; Q \vdash \mathrm{share}(x; x_1, x_2.e) : B}\;(\textsc{M:Share}) \qquad \frac{\Gamma; \pi_{\vec{0}}^{\Gamma}(Q) \vdash e : B}{\Gamma, x{:}\tau; Q \vdash e : B}\;(\textsc{M:Weak})$$

$$\frac{\Gamma; P \vdash e : \langle \tau, P' \rangle \qquad Q \geq P \qquad Q' \leq P'}{\Gamma; Q \vdash e : \langle \tau, Q' \rangle}\;(\textsc{M:Relax})$$

$$\frac{\Gamma; P \vdash e : \langle \tau, P' \rangle \qquad Q = P + c \qquad Q' = P' + c}{\Gamma; Q \vdash e : \langle \tau, Q' \rangle}\;(\textsc{M:Offset})$$

Figure 1: Type rules for annotated types.

Notations Families that describe type and context annotations are denoted with upper case letters $Q, P, R, \ldots$ with optional superscripts. I use the convention that the elements of the families are the corresponding lower case letters with corresponding superscripts, that is, $Q = (q_i)_{i\in I}$, $Q' = (q'_i)_{i\in I}$, and $Q^x = (q^x_i)_{i\in I}$.

Let Q, Q' be two annotations with the same index set I. I write $Q \le Q'$ if $q_i \le q'_i$ for every $i \in I$. For $K \in \mathbb{Q}$ I write $Q = Q' + K$ to state that $q_{\vec{0}} = q'_{\vec{0}} + K \ge 0$ and $q_i = q'_i$ for $i \ne \vec{0} \in I$. Let $\Gamma = \Gamma_1, \Gamma_2$ be a context, let $i = (i_1, \ldots, i_k) \in \mathcal{I}(\Gamma_1)$ and $j = (j_1, \ldots, j_l) \in \mathcal{I}(\Gamma_2)$. I write (i, j) to denote the index $(i_1, \ldots, i_k, j_1, \ldots, j_l) \in \mathcal{I}(\Gamma)$.

Let Q be an annotation for a context Γ_1, Γ_2. For $j \in \mathcal{I}(\Gamma_2)$ I define the *projection* $\pi^{\Gamma_1}_j(Q)$ of Q to Γ_1 to be the annotation Q' with $q'_i = q_{(i,j)}$. The essential properties of the projections are stated by Proposition 1. It shows how the analysis of juxtaposed functions can be broken down to individual components.

Proposition 1. *Let $\Gamma, x{:}\tau; Q$ be an annotated context and $V : \Gamma, x{:}\tau$. Then*

$$\Phi_V(\Gamma, x{:}\tau; Q) = \sum_{j\in\mathcal{I}(\tau)} \Phi_V(\Gamma; \pi^{\Gamma}_j(Q)) \cdot p_j(V(x)) \,.$$

Additive Shift A key notion in the type system is the multivariate *additive shift* that is used to assign potential to typing contexts that result from a pattern match on lists.

Let $\Gamma, y{:}L(\tau)$ be a context and let $Q = (q_i)_{i\in\mathcal{I}(\Gamma, y:L(\tau))}$ be a context annotation of degree k. The *additive shift for lists* $\lhd_L(Q)$ of Q is an annotation $\lhd_L(Q) = (q'_i)_{i\in\mathcal{I}(\Gamma, x_1:\tau, x_2:L(\tau))}$ of degree k for a context $\Gamma, x_1{:}\tau, x_2{:}L(\tau)$ that is defined through

$$q'_{(i,j,\ell)} = \begin{cases} q_{(i,j::\ell)} + q_{(i,\ell)} & j = 0 \\ q_{(i,j::\ell)} & j \ne 0 \end{cases}$$

Let us first look that the additive shift for some examples.

- Consider for instance a context $\ell{:}L(\text{unit})$ with a single integer list that features an annotation $Q = (q_0, \ldots, q_k) = (q_{[]}, \ldots, q_{[0,\ldots,0]})$. The shift operation $\lhd_L$ for lists produces an annotation for a context of the form $x_1{:}\text{unit}, x_2{:}L(\text{unit})$, namely

  $$\lhd_L(Q) = Q' = (q'_{(0,0)}, \ldots, q'_{(0,k)}) \text{ such that } q'_{(0,i)} = q_i + q_{i+1} \text{ for all } i < k \text{ and } q_{(0,k)} = q_k.$$

  This is exactly the additive shift that used in univariate AARA. Like in the univariate system, we use it in a context where ℓ points to a list of length $n+1$ and x_2 is the tail of ℓ. It reflects the fact that $\sum_{i=0,\ldots,k} q_i\binom{n+1}{i} = \sum_{i=0,\ldots,k-1} q_{i+1}\binom{n}{i} + \sum_{i=0,\ldots,k} q_i\binom{n}{i}$.

- Now consider a context $\ell_1{:}L(\text{unit}), \ell_2{:}L(\text{unit}); Q$ where $Q = (q_{(i,j)})_{i+j\le k}$, ℓ_1 is a list of length m, and ℓ_2 is a list of length $n+1$. The additive shift results in an annotation for a context of the form $\ell_1{:}L(\text{unit}), x{:}\text{unit}, xs{:}L(\text{unit})$ and the intention is that xs is the tail of ℓ_2, that is, a list of length n. From the definition it follows that $\lhd_L(Q) = (q'_{(i,0,j)})_{i+j\le k}$ where

  $$q_{(i,0,j)} = q_{(i,j)} + q_{(i,j+1)} \text{ if } i+j<k \text{ and } q_{(i,0,j)} = q_{(i,j)} \text{ if } i+j=k.$$

  The soundness follows from the fact that for $i \le k$, we have

  $$\sum_{j=1}^{k-i} q_{(i,j)}\binom{m}{i}\binom{n+1}{j} = \binom{m}{i}\Big(\sum_{j=0}^{k-i-1}(q_{(i,j)} + q_{(i,j+1)})\binom{n}{i} + q_{(i,k-i)}\binom{n}{k}\Big)\,.$$

Lemma 4 states the soundness of the shift operations.

Lemma 4. *Let $\Gamma, x{:}L(\tau); Q$ be an annotated context, $V, x \mapsto v_1 :: v_2 : \Gamma, x{:}L(\tau)$, and let $V' = V, x_1 \mapsto v_1, x_2 \mapsto v_2$. Then $S' : \Gamma, x_1{:}\tau, x_2{:}L(\tau)$ and $\Phi_V(\Gamma, x{:}L(\tau); Q) = \Phi_{V'}(\Gamma, x_1{:}\tau, x_2{:}L(\tau); \lhd_L(Q))$.*

Lemma 4 is a consequence of Lemma 2. One takes the linear combination of instances of its second equation and regroups the right hand side according to the base polynomials for the resulting context.

Lists The rule M:CONS assigns potential to a extended list. The additive shift $\lhd_L(Q')$ transforms the annotation Q' for a list type into an annotation for the context $x_1{:}A, x_2{:}L(\tau)$. Lemma 4 shows that potential is neither gained nor lost by this operation.

The rule M:MATL types pattern matching of lists. The initial potential defined by the annotation Q of the context $\Gamma, x{:}L(\tau)$ has to be sufficient to pay the costs of the evaluation of e_1 or e_2 (depending on whether the matched list is empty or not) and the potential defined by the annotation Q' of the result type. To type the expression e_1 of the *nil* case we use the projection $\pi_0^\Gamma(Q)$ that results in an annotation for the context Γ. Since the matched list is empty in this case no potential is lost by the discount of the annotations $q_{(i,j)}$ of Q where $j \neq 0$. To type the expression e_2 of the *cons* case we rely on the shift operation $\lhd_L(Q)$ for lists that results in an annotation for the context $\Gamma, x_1{:}\tau, x_2{:}L(\tau)$. Again there is no loss of potential (see Lemma 4).

Sharing Let $\Gamma, x_1{:}\tau, x_2{:}\tau; Q$ be an annotated context. The *sharing operation* $\curlyvee(Q)$ defines an annotation for the context $\Gamma, x{:}\tau$. It is used when the potential is split between multiple occurrences of a variable. The following lemma shows that sharing is a linear operation that does not lead to any loss of potential.

Lemma 5. *Let τ be a type. Then there are non-negative rational numbers $c_k^{(i,j)}$ for $i,j,k \in \mathcal{I}(A)$ and $\deg(k) \le \deg(i,j)$ such that the following holds. For every context $\Gamma, x_1{:}\tau, x_2{:}\tau; Q$ and $V, x \mapsto v : \Gamma, x{:}\tau$ we have $V, x_1 \mapsto v, x_2 \mapsto v : \Gamma, x_1{:}\tau, x_2{:}\tau$ and $\Phi_V(\Gamma, x{:}\tau; Q') = \Phi_{V'}(\Gamma, x_1{:}\tau, x_2{:}\tau; Q)$ where $q'_{(\ell,k)} = \sum_{i,j\in\mathcal{I}(\tau)} c_k^{(i,j)} q_{(\ell,i,j)}$.*

Lemma 5 is a direct consequence of Corollary 1. In fact, inspection of the argument of the underlying Lemma 1 shows that the coefficients $c_k^{(i,j)}$, are indeed *natural* numbers and can be computed effectively.

For a context $\Gamma, x_1{:}\tau, x_2{:}\tau; Q$ we define $\curlyvee(Q)$ to be the Q' from Lemma 5.

Proof. The task is to show that for every resource polynomial $p_{(i,j)}((v,v)) = p_i(v)\cdot p_i(v)$ can be written as a sum (possibly with repetitions) of $p_{i'}(v)$'s. We argue by induction on τ. If τ is an atomic type *bool*, *int*, or *unit*, we can simply write $1\cdot 1$ as 1. If τ is a pair $\tau = (\tau_1, \tau_2)$ then we have $p_{(i,j)}((v,w)) \cdot p_{(i',j')}((v,w)) = p_i(v)p_j(w)p_{i'}(v)p_{j'}(w) = (p_i(v)p_{i'}(v))(p_j(w)p_{j'}(w))$. By induction hypothesis, $(p_i(v)p_{i'}(v))$ and $(p_j(w)p_{j'}(w))$ both are sums of elemtary resource polynomials for τ_1 or τ_2, respectively. So the expression is a sum of terms of the form $p_{i''}(v)p_{j''}(w)$, which is $p_{(i'',j'')}((v,w))$. If τ is a list $\tau = L(\tau)'$ we have to consider

$$\begin{aligned} & p_{[i_1,\dots,i_k]}([v_1,\dots,v_n]) \cdot p_{[i'_1,\dots,i'_{k'}]}([v_1,\dots,v_n]) \\ = \; & \Big(\sum_{1\le j_1<\dots<j_k\le n} p_{i_1}(v_{j_1})\dots p_{i_k}(v_{j_k})\Big)\Big(\sum_{1\le j'_1<\dots<j'_{k'}\le n} p_{i'_1}(v_{j'_1})\dots p_{i'_{k'}}(v_{j'_{k'}})\Big) \end{aligned}$$

Using the distributive law, this can be considered as the sum over all possible ways to arrange the $j_1,\dots,j_k$ and $j'_1,\dots,j'_{k'}$ relative to each other respecting their respective orders, including the case that some j_i coincide with some $j'_{i'}$. Each of term in this sum of fixed length (independent of the lists) has the shape

$$\sum_{1\le j''_1<\dots<j''_\ell\le n} q_1(v_{j''_1})\dots q_\ell(v_{j''_\ell})$$

where each $q_r(v_{j_r})$ is either a $p_{i_s}(v_{j_r})$, a $p_{i'_{s'}}(v_{j_r})$ or a product $p_{i_r}(v_{j_r})p_{i'_{s'}}(v_{j_r})$. The latter can, by induction hypothesis, be written as sum of $p_{i''}(v_{j_r})$'s. Again, this presentation is independent of the actual value of v_{j_r}. Using distributivity again, we obtain a sum of expressions of the form

$$\sum_{1\le j''_1<\dots<j''_\ell\le n} p_{i''_1}(v_{j''_1})\dots p_{i''_\ell}(v_{j''_\ell}) = p_{[i''_1,\dots,i''_\ell]}$$

□

Sequential Composition The rule M:LET uses cost-free type judgments

$$\Gamma; Q \overset{\text{cf}}{\vdash} e : (\tau, Q')$$

in which all expressions tick{q} are treated as tick{0} (so that the cost is zero). We use cost-free judgments to assign potential to an extended context in the let rule.

M:LET can be explained in two steps. The first starts with the derivation of the judgment $\Gamma_1; P \vdash e_1 : (\tau, P')$ for the sub-expression e_1. The annotation P corresponds to the potential that is exclusively attached to Γ_1 by the annotation Q, namely $P = \pi_{\vec{0}}^{\Gamma_1}(Q)$. Now we derive the judgment $\Gamma_2, x{:}\tau; R \vdash e_2 : B$. The potential that is assigned by R to $x{:}\tau$ is the potential that resulted from the judgment for e_1, namely $P' = \pi_{\vec{0}}^{x:\tau}(R)$. The potential that is assigned by R to Γ_2 is essentially the potential that is assigned by to Γ_2 by Q, namely $\pi_{\vec{0}}^{\Gamma_2}(Q) = \pi_{0}^{\Gamma_2}(R)$.

The second step of the derivation is to relate the annotations in R that refer to mixed potential between $x{:}\tau$ and Γ_2 to the annotations in Q that refer to potential that is mixed between Γ_1 and Γ_2. To this end we remember that we can derive from a judgment $\Gamma_1; S \vdash e_1 : \langle \tau, S' \rangle$ that $\Phi(\Gamma_1; S) \geq \Phi(v{:}\langle \tau, S' \rangle)$ if e_1 evaluates to v. This inequality remains valid if multiplied with a potential for $\phi_{\Gamma_2} = \Phi(\Gamma_2; T)$, that is, $\Phi(\Gamma_1; S) \cdot \phi_{\Gamma_2} \geq \Phi(v{:}(\tau, S')) \cdot \phi_{\Gamma_2}$. To relate the mixed potential annotations we thus derive a cost-free judgment $\Gamma_1; P_j \overset{\text{cf}}{\vdash} e_1 : (\tau, P'_j)$ for every $\vec{0} \neq j \in \mathcal{I}(\Gamma_2)$. (We use cost-free judgments to avoid paying multiple times for the evaluation of e_1.) Then we equate P_j to the corresponding annotations in Q and equate P'_j to the corresponding annotations in R, that is, $P_j = \pi_j^{\Gamma_1}(Q)$ and $P'_j = \pi_j^{x:\tau}(R)$. The intuition is that j corresponds to ϕ_{Γ_2}.

Note that things are a bit more complex than described here. The problem is that the function types in Γ_1 that are used in the judgments $\Gamma_1; P_j \overset{\text{cf}}{\vdash} e_1 : (\tau, P'_j)$ describe bounds on the cost of the function bodies. However, we really want cost-free function types. So we should in fact again use a set of function types for each function like we did for resource polymorphic recursion. However, we would have cost-free and "normal" function types in the set then can be picked in the respective typing modes.

7 Soundness

We can prove the usual soundness theorem for AARA. The prove follows the same structure as for linear AARA. The additional complexity is largely handled lemmas by the previous lemmas.

Theorem 1 (Soundness of AARA)**.** *Let* $\Gamma; Q \vdash e : A$ *and* $V : \Gamma$*. If* $V \vdash e \Downarrow v \mid (p, p')$ *for some* v *and* (p, p') *then* $\Phi_V(\Gamma; Q) \geq p$ *and* $\Phi_V(\Gamma; Q) - \Phi(v : A) \geq p - p'$.

References

[HAH11] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate Amortized Resource Analysis. In *38th Symposium on Principles of Programming Languages (POPL'11)*, 2011.

[HDW17] Jan Hoffmann, Ankush Das, and Shu-Chun Weng. Towards Automatic Resource Bound Analysis for OCaml. In *44th Symposium on Principles of Programming Languages (POPL'17)*, 2017.