Lecture Notes on Resource Analysis

Part 1

JIAWEN LIU, JÉRÉMY THIBAULT, and SHAOWEI ZHU

These are lecture notes taken during Jan Hoffmann's course on resource analysis at the Oregon Programming Language Summer School 2019.

1 MOTIVATION OF RESOURCE ANALYSIS

Consider two implementations of the same (mathematical) function. How do we compare these two implementations? One solution is to compare their *efficiency* in terms of:

- time complexity;
- memory usage;
- power consumption;
- etc.

For each of the above notions of efficiency, one can also consider two kinds of analyses:

- worst-case analysis;
- average-case analysis

Traditionally, these algorithm analyses are performed as follows:

(1) Define the algorithm in "pseudo-code". However, this comes with several limitations. First, the algorithm description's relation to the actual implementation is unclear. For instance, the pseudo-code might assume the existence of a data structure representing sets and make simplified assumptions on its implementation. Secondly, the cost model is often not specified. Finally, the representation of the input is not necessarily explicit.
(2) The complexity is analyzed using Big-O notation.

Definition 1.1 (Big-O)**.** Let f and g be functions from $\mathbb{N}$ to $\mathbb{N}$. Then,

$$f \in O(g) \triangleq \exists n_0, c, \forall n \geq n_0, f(n) \leq cg(n)$$

The notion of big-O also has limitations: algorithms might have large constants making them slow in practice; big-O is an asymptotic notion, and it is impossible to compare programs on "small" inputs; it is also impossible to compare programs that have the same asymptotic behavior; and finally, it is hard to generalize big-O to multiple inputs ($O(mn)$ has no clear meaning).

2 RESOURCE ANALYSIS

We are going to analyze programs as *mathematical objects*. For this, we need to define

- syntax and static semantics;
- cost semantics.

Authors' address: Jiawen Liu, jliu223@buffalo.edu; Jérémy Thibault, jeremy.thibault@inria.fr; Shaowei Zhu, shaoweiz@cs.princeton.edu.

In order to take into account the constant factors, we are going to define tools to help us: proof rules, type systems, and automation techniques that will help us figure the exact constants.

Finally, we want to obtain bounds that come with *certificates*: proofs (proof derivations) that show how the bounds are obtained.

This course series focuses on functional languages, worst-case bounds, and automatic bound inference.

3 A SIMPLE LANGUAGE

3.1 Syntax

We consider a simple language, for which we will define several kinds of cost semantics. Types and expressions are presented in the following chart consisting of abstract syntax, concrete syntax, and explanations:

<*Types*> τ ::=	$\texttt{arr}(\tau_1, \tau_2)$	$\tau_1 \to \tau_2$	type for functions from τ_1 to τ_2
|	$\texttt{unit}$	$\mathbf{1}$	unit type
<*Expressions*> e ::=	x	x	variable
|	$\texttt{lam}\{\tau\}(x.e)$	$\lambda(x:\tau).e$	lambda function
|	$\texttt{triv}$	$\langle\rangle$	empty expression

Values are inductively defined as follows:

$$\frac{}{\texttt{triv val}} \qquad\qquad \frac{}{\lambda\,(x:\tau)\,.\,e\ \text{val}}$$

We are now ready to describe several cost semantics for this language.

3.2 First Option: Structural Dynamics (or Small-Step Semantics)

We define the following judgment: $e \mapsto e'$ meaning expression e evaluates to e' in one step. The definition of $e \mapsto e'$ is given by the following rules:

$$\frac{e_1 \mapsto e_1'}{e_1(e_2) \mapsto e_1'(e_2)} \qquad \frac{e_1\ \text{val} \qquad e_2 \mapsto e_2'}{e_1(e_2) \mapsto e_1(e_2')} \qquad \frac{e_2\ \text{val}}{(\lambda(x:\tau)e)e_2 \mapsto [e_2/x]e}$$

Definition 3.1 (Evaluation cost in the small-step semantics)**.** Let $e : \tau$ be an expression. The *evaluation cost* of e is:

- n, if $e \mapsto^n v$
- ∞, otherwise

Note that here the evaluation is deterministic and unique, so the cost definition makes sense.

Remark. We also assume that all steps cost the same as a simplification, but in reality this is not necessarily the case. For instance, the substitution operation traverses the whole expression, and it would be legitimate to give it a higher cost.

3.3 Second Option: Evaluation Dynamics (Also Known as Big-Step Semantics)

The traditional way of defining big-step semantics, $e \Downarrow v$, does not capture cost. The first idea is to define a big-step semantics with a new component $q \in \mathbb{Q}$, that vary during the execution and represents the cost of the evaluation:

$$e \Downarrow^q v$$

This judgment reads "expression e evaluates to value v, with cost q".

In fact, we will go a step further and add a *metric* M:

$$V \vdash_M e \Downarrow^q v$$

This judgment means that under environment $V : \text{Vars} \to \text{Vals}$, with cost metric M, expression e evaluates to value v with cost $q \in \mathbb{Q}$. (The V is not necessary, could also do $\dot{\vdash}_M e \Downarrow^q v$. But will use V later in soundness)

Here, M is a cost metric that maps each operation to a rational cost value.

$$M : \{\texttt{unit}, \texttt{app}, \texttt{lam}, \texttt{var}\} \to \mathbb{Q}$$

Costs could be negative. This can be used to indicate that some resources are becoming available.

The rules of evaluation dynamics are:

$$\frac{}{V \vdash_M x \Downarrow^{M(\texttt{var})} V(x)} \qquad \frac{}{V \vdash_M \texttt{triv} \Downarrow^{M(\texttt{unit})} \texttt{triv}} \qquad \frac{}{V \vdash_M \lambda(x:\tau).\, e \Downarrow^{M(\texttt{lam})} \lambda(x:\tau).\, e}$$

$$\frac{V \vdash_M e_1 \Downarrow^{c_1} \lambda(x:\tau)e \qquad V \vdash_M e_2 \Downarrow^{c_2} v_2 \qquad V[x \mapsto v_2] \vdash_M e \Downarrow^{c} v}{V \vdash_M e_1(e_2) \Downarrow^{c+c_1+c_2+M(\texttt{app})} v}$$

Based on knowledge of β-reduction, the following definition of metric S seems reasonable if we want the big-step semantics to be consistent with small-step semantics:

$$S(\texttt{app}) = 1 \tag{1}$$

$$S(\texttt{unit}) = 0 \tag{2}$$

$$S(\texttt{var}) = 0 \tag{3}$$

$$S(\texttt{lam}) = 0 \tag{4}$$

Indeed, under this cost metric S we can prove the following theorem:

Theorem 3.1 (Equivalence between big-step and small-step cost semantics). *For any expression $e : \tau$,*

$$e \mapsto^n v \iff \vdash_S e \Downarrow^n v$$

High-water mark. Suppose that we want to analyze the resource consumption of a stack-based process, and we define the cost to be the number of stack cells that is currently occupied. Clearly, the number of occupied stack cells changes as time goes by, and in the end everything is deallocated with the effect that the cost goes back to 0. Thus knowing the "net cost" by comparing the final state to the initial state does not seem to be very useful. We are instead more interested in knowing the maximum amount of stack that was used at any point in the execution: this is the *high-water mark*. We cannot use the previous two cost semantics to study the high-water mark. The next two subsections are dedicated to describing cost semantics that can be used to study this high-water mark notion for resources.

3.4 Third Option: Resource Safety

The judgment for resource safety,

$$V_M \vdash_{q'}^{q} e \Downarrow v,$$

means that under environment V, metric M, and given that q resources are available, expression e evaluates to value v, and q' resource are available afterwards (we always require $q, q' \geq 0$; otherwise, it doesn't make sense).

The semantics is as follows:

$$\frac{}{V_M \vdash_{q'}^{q+M(\mathsf{var})} x \Downarrow V(x)} \qquad \frac{}{V_M \vdash_{q}^{q+M(\mathsf{lam})} \lambda\,(x:\tau)\,.\,e \Downarrow \lambda\,(x:\tau)\,.\,e}$$

$$\frac{V_M \vdash_{p}^{q} e_1 \Downarrow \lambda\,(x:\tau)\,.\,e \qquad V_M \vdash_{r}^{p} e_2 \Downarrow v_2 \qquad V[x \mapsto v_2]_M \vdash_{q'}^{r} e \Downarrow v}{V_M \vdash_{q'}^{q+M(\mathsf{app})} e_1(e_2) \Downarrow v} \qquad \frac{}{V_M \vdash_{q'}^{q+M(\mathsf{unit})} () \Downarrow ()}$$

In these rules, the resources p, q, are always non-negative. Hence, this semantics ensures that the resource consumption never goes below 0, and if $V_M \vdash_{q'}^{q} e \Downarrow v$, q' is an upper bound on the amount of resources used.

We can prove the following results regarding resource safety.

Lemma 3.2. *Let $V_M \vdash_{q'}^{q} e \Downarrow v$ and $V_M \vdash_{p'}^{p} e \Downarrow v'$ then $q - q' = p - p'$ and $v = v'$.*

Informally, this theorem states that the execution is deterministic: the result of an evaluation is always the same, and the amount of resources consumed too.

Theorem 3.3 (Resource Safety Implies Big-Step). *If $V_M \vdash_{q'}^{q} e \Downarrow v$ then $V \vdash_M e \Downarrow^{q-q'} v$.*

Note that the theorem is *not* saying: $V \vdash_M e \Downarrow^{c} v$ implies $V_M \vdash_0^c e \Downarrow v$. Indeed, there are executions accepted by the first big-step semantics, but in fact consume more that c resources at some point of the execution.

3.5 Fourth Option: Resource Monoid

Combining resource analysis is not straightforward: the high-water mark of a sequential composition is neither the sum of the high-water marks, or the second high-water mark.

To handle this, we consider a judgment of the following form:

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

In this judgment, q is the high-water mark and q' is the quantity of left-over resources ($q, q' \geq 0$). Some rules are given below:

$$\frac{M(\mathsf{var}) \geq 0}{V \vdash_M x \Downarrow V(x) \mid (M(\mathsf{var}), 0)} \qquad \frac{M(\mathsf{var}) < 0}{V \vdash_M x \Downarrow V(x) \mid (0, -M(\mathsf{var}))}$$

$$\frac{M(\mathsf{app}) \geq 0 \qquad V \vdash_M e_1 \Downarrow \lambda(x:\tau)e \mid (q, q') \qquad V \vdash_M e_2 \Downarrow v_2 \mid (p, p') \qquad V[x \mapsto v_2] \vdash_M e \Downarrow v \mid (r, r')}{V \vdash_M e_1(e_2) \Downarrow v \mid (M(\mathsf{app}), 0) \cdot (q, q') \cdot (p, p') \cdot (r, r')}$$

$$\frac{M(\mathsf{app}) < 0 \qquad V \vdash_M e_1 \Downarrow \lambda(x:\tau)e \mid (q, q') \qquad V \vdash_M e_2 \Downarrow v_2 \mid (p, p') \qquad V[x \mapsto v_2] \vdash_M e \Downarrow v \mid (r, r')}{V \vdash_M e_1(e_2) \Downarrow v \mid (M(\mathtt{0}, -\mathsf{app})) \cdot (q, q') \cdot (p, p') \cdot (r, r')}$$

The main change in these definitions is the addition of a special operator, $\cdot$, that allows composing pairs consisting of a high-water mark and an amount of leftover resources.

Homework. define the operator $\cdot$: $(p, p') \cdot (q, q')$ so that the following theorem holds.

Theorem 3.4 (Equivalence Between the Resource Monoid and Resource Safety). *The following propositions hold:*

(1) *If* $V \vdash_M e \Downarrow v \mid (q, q')$ *then* $V \vdash^{q}_{q'} e \Downarrow v$.
(2) *If* $V \vdash^{q}_{q'} e \Downarrow v$ *then* $V \vdash e \Downarrow v \mid (p, p')$ *and* $p \leq q$.