Resource Analysis: Lecture 1

JAN HOFFMANN

1 MOTIVATION OF RESOURCE ANALYSIS

Consider two implementations of the same (mathematical) function. How do we compare these two implementations? By their efficiency:

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

We 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 makes assumptions on its implementation and the cost of using it. The cost model is also often not specified. Finally, the representation of the input is not necessarily explicit.
(2) The complexity is analyzed using Big-O.

Definition 1.1 (Big-O). Let f and g be a function 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 have 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.

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

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

Author's address: Jan Hoffmann, jhoffmann@cmu.edu.

In this short course series, we focus on functional languages, worst-case bounds, and automatic bound inference.

3 A SIMPLE LANGUAGE

3.1 Syntax

We consider a very simple language, for which we will define several kind of cost semantics. Types and expressions are defined as follows:

$\langle \textit{Types} \rangle\ \tau\ ::= \mathrm{arr}\,(\tau_1, \tau_2)$ // function from τ_1 to τ_2, written as $\tau_1 \to \tau_2$
  | unit

$\langle \textit{Expressions} \rangle\ e\ ::= x$ // variables, written as x
  | $\mathrm{app}\,(e_1, e_2)$ // function application, written as $e_1(e_2)$
  | $\mathrm{lam}\{\tau\}\,(x.e)$ // function with argument x of type τ and body e, written as $\lambda(x : \tau)e$
  | $\langle\rangle$ // empty expression

Values are inductively defined as follows:

$$\frac{}{\langle\rangle\ \mathrm{val}} \qquad\qquad \frac{}{\lambda\,(x : \tau)\,.\,e\ \mathrm{val}}$$

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: $e \mapsto e'$ means expression e evaluates to e' in one step.

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

Fig. 1. Structural dynamics for the simple language.

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.

Here 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, $e \Downarrow v$, do 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, costing 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 : Vars $\longrightarrow$ Vals, with cost metric M, expression e evaluates to value v with cost $q \in \mathbb{Q}$.

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

$$M : \{\text{unit}, \text{app}, \text{lam}, \text{var}\} \longrightarrow \mathbb{Q}$$

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

Rules are given in Fig. 2.

$$\frac{}{V \vdash_M x \Downarrow^{M(\text{var})} V(x)} \qquad \frac{}{V \vdash_M \langle\rangle \Downarrow^{M(\text{var})} \langle\rangle} \qquad \frac{}{V \vdash_M \lambda\,(x:\tau)\,.\,e \Downarrow^{M(\text{var})} \lambda\,(x:\tau)\,.\,e}$$

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

Fig. 2. Evaluation dynamics

Here is an example of metric S we could use:

$$S\,(\text{app}) = 1 \tag{1}$$
$$S\,(\text{unit}) = 0 \tag{2}$$
$$S\,(\text{var}) = 0 \tag{3}$$
$$S\,(\text{lam}) = 0 \tag{4}$$

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

Theorem 3.2 (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. We want to analyze the resource consumption of a stack-based process. The number of occupied stack cells changes as time goes by, but at the end, everything is deallocated, and the cost goes back to 0. 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 may not 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.

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).

Semantics are shown in Fig. 3.

$$\frac{}{V\,{}_M\!\vdash^{q+M(\text{var})}_{q'} x \Downarrow V(x)} \qquad \frac{}{V\,{}_M\!\vdash^{q+M(\text{lam})}_{q} \lambda\,(x:\tau)\,.\,e \Downarrow \lambda\,(x:\tau)\,.\,e}$$

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

Fig. 3. Big-step semantics for resource safety

We can prove the following results regarding resource safety.

Lemma 3.3. *Let* $V\,{}_M\!\vdash^{q}_{q'} e \Downarrow v$ *and* $V\,{}_M\!\vdash^{p}_{p'} e \Downarrow v'$ *then* $q - q' = p - p'$ *and* $v = v'$.

Theorem 3.4 (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 $V\,{}_M\!\vdash^{c}_{0} e \Downarrow v$.

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 ($q, q' \geq 0$).

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

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

Fig. 4. Semantics for resource monoid method. For simplicity we only list the rule for app when its cost is nonnegative. The definition of the operator $\cdot$ is left as an exercise.

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

Theorem 3.5. (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* $p \leq q$.