Time Complexity of Erlang Programs

Christian Rinderknecht

17th August 2018

Let us note e any Erlang *expression*, i.e., a function call or a constant, that can be rewritten by Erlang clauses. This process is called *computation* and (1) may never end (*loop*), (2) ends with a type error (as `1 + []`), (3) ends with a match error (an expression containing a function call which matches no head), (4) ends with a value (an expression which is a constant).

This procedure can leads us to define the *cost* (or *complexity*) of computing an Erlang expression as the number of rewrite steps until reaching a value. Accordingly, a loop has an infinite cost and a value has a cost of zero. We denote the cost of expression e by $[\![e]\!]$, and the value of an expression e, assuming it exists, by $\mathcal{V}(e)$. Hence, if e is a value, i.e., $e = \mathcal{V}(e)$, then $[\![e]\!] = [\![\mathcal{V}(e)]\!] = 0$.

1 Joining two lists

Consider the following Erlang module:

```
-module(join).
-export([join/2]).
join([],L)    -> L;                        % Clause 1
join([H|T],L) -> [H|join(T,L)].            % Clause 2
```

The cost of a function call like $[\![\texttt{join(}e_1\texttt{,}e_2\texttt{)}]\!]$ is, by definition, the sum of the costs of rewriting e_1 and e_2, plus the cost of rewriting the call with the corresponding values $\mathcal{V}(e_1)$ and $\mathcal{V}(e_2)$ as arguments:

$$[\![\texttt{join(}e_1\texttt{,}e_2\texttt{)}]\!] \triangleq [\![e_1]\!] + [\![e_2]\!] + [\![\texttt{join(}\mathcal{V}(e_1)\texttt{,}\mathcal{V}(e_2)\texttt{)}]\!]$$

We apply this definition to each clause in the definition:

$$[\![\texttt{join([],L)}]\!] = 1$$
$$[\![\texttt{join([H|T],L)}]\!] = 1 + [\![\texttt{join(T,L)}]\!]$$

The "1" in the previous equations are due to the fact that at least one rewrite step is necessary. Here, all Erlang variables, `L`, `H` and `T`, denote values, since their binding takes place after the call. So, $\mathcal{V}(\texttt{L}) = \texttt{L}$ etc. Note that, since there are no rewrite rules for `[H|T]`, the cost of it is 0 and $\mathcal{V}(\texttt{[H|T]}) = \texttt{[H|T]}$.

Let us now define the cost in terms of the size of the input, i.e., the arguments. The function call `join(`e_1`,`e_2`)` has two lists as arguments. The usual way to measure the size of a list is to consider the number of elements it contains, i.e., its length. Here, the size of the input is characterized by a pair (n, m) where $n = \texttt{len(}e_1\texttt{)}$ and $m = \texttt{len(}e_2\texttt{)}$. Let us define the cost of `join/2` in terms of the input's size as

$$\mathcal{C}^{\texttt{join/2}}_{n,m} \triangleq [\![\texttt{join(}e_1\texttt{,}e_2\texttt{)}]\!] \quad \text{where } \texttt{len(}e_1\texttt{)} = n \text{ and } \texttt{len(}e_2\texttt{)} = m$$

Using this definition, the previous equations are equivalent to

$$\begin{aligned} \mathcal{C}^{\texttt{join/2}}_{0,m} &= 1 \\ \mathcal{C}^{\texttt{join/2}}_{n+1,m} &= 1 + \mathcal{C}^{\texttt{join/2}}_{n,m} && \text{where } n, m \geqslant 0 \end{aligned}$$

Notice that the length of the second list, m, does not impact the cost. We need now to express $\mathcal{C}^{\texttt{join/2}}_{n+1,m}$ in terms of n. The trick consists in making the difference of two successive terms in the series and then summing up these differences:

$$\begin{aligned} \mathcal{C}^{\texttt{join/2}}_{n+1,m} - \mathcal{C}^{\texttt{join/2}}_{n,m} &= 1 && \text{where } n \geqslant 0 \\ \sum_{n=0}^{p} (\mathcal{C}^{\texttt{join/2}}_{n+1,m} - \mathcal{C}^{\texttt{join/2}}_{n,m}) &= \sum_{n=0}^{p} 1 && \text{where } p \geqslant 0 \\ \mathcal{C}^{\texttt{join/2}}_{p+1,m} - \mathcal{C}^{\texttt{join/2}}_{0,m} &= p + 1 \end{aligned}$$

Replacing $p + 1$ by n we finally get

$$\begin{aligned} \mathcal{C}^{\texttt{join/2}}_{n,m} - \mathcal{C}^{\texttt{join/2}}_{0,m} &= n && \text{where } n > 0 \\ \mathcal{C}^{\texttt{join/2}}_{n,m} &= n + 1 \end{aligned}$$

Since that replacing n by 0 in the latter formula gives

$$\mathcal{C}^{\texttt{join/2}}_{0,m} = 0 + 1 = 1$$

it is thus possible to have only one furmula for the cost:

$$\mathcal{C}^{\texttt{join/2}}_{n,m} = n + 1 \quad \text{where } n, m \geqslant 0$$

Let us take an example:

$$\begin{aligned} \texttt{join([1,2,3], [4,5])} &\xrightarrow{2} \texttt{[1|join([2,3],[4,5])]} \\ &\xrightarrow{2} \texttt{[1|[2|join([3],[4,5])]]} \\ &\xrightarrow{2} \texttt{[1|[2|[3|join([],[4,5])]]]} \\ &\xrightarrow{1} \texttt{[1|[2|[3|[4,5]]]]} \end{aligned}$$

$$= \texttt{[1,2,3,4,5]}$$

The formula predicted $\mathcal{C}^{\texttt{join/2}}_{\texttt{len([1,2,3]),len([4,5])}} = \mathcal{C}^{\texttt{join/2}}_{3,2} = 3 + 1 = 4$ and we indeed had 4 rewrite steps until the result.

Futhermore, it is important to have a clear understanding of the behaviour of the cost when the input becomes sufficiently big. This insight is provided by considering an equivalent function[1] when the variable is arbitrarily big:

$$\mathcal{C}^{\texttt{join/2}}_{n,m} \sim n \quad \text{where } n \to +\infty$$

These kinds of asymptotical expressions are also very useful when comparing the cost of two Erlang functions, perhaps two versions of the same function.

The idea behind our concept of cost is that the time spent in computing an expression is proportional to the cost. So, here, it means that the time necessary to compute the joining of two lists is proportional to the length of the first list, when the list grows very long. In particular, doubling the size of the first list will approximatively double the time to get the result, whereas doubling the length of the second list will have no impact.

2 Reversing a list

2.1 A naive version

Consider the following Erlang module:

```
-module(rev).
-export([rev/1]).
join([],L)     -> L;
join([H|T],L)  -> [H|join(T,L)].

rev([])    -> [];                       % Clause 3
rev([H|T]) -> join(rev(T),[H]).         % Clause 4
```

The cost of a function call like $[\![\texttt{rev(}e\texttt{)}]\!]$ is, by definition, the sum of the costs of rewriting e and the cost of rewriting the call with the corresponding value $\mathcal{V}(e)$ as argument.

$$[\![\texttt{rev(}e\texttt{)}]\!] \triangleq [\![e]\!] + [\![\texttt{rev(}\mathcal{V}(e)\texttt{)}]\!]$$

Considering each clause separately, we can therefore write the equations defining the costs

$$\begin{aligned} [\![\texttt{rev([])}]\!] &= 1 \\ [\![\texttt{rev([H|T])}]\!] &= 1 + [\![\texttt{join(rev(T),[H])}]\!] \end{aligned}$$

[1]Two functions $f(x)$ and $g(x)$ are equivalent at $+\infty$ if $\lim_{x\to+\infty} f(x)/g(x) = 1$.

$$= 1 + ([\![\texttt{rev(T)}]\!] + [\![\texttt{join(}\mathcal{V}\texttt{(rev(T)),[H])}]\!])$$

Let us now define the cost in terms of the size of the input, i.e., the argument. The function call `rev(`e`)` has one list as argument. The usual way to measure the size of a list is to consider the number of elements it contains, i.e., its length. Here, the size of the input is characterized by an integer n where $n = \texttt{len(}e\texttt{)}$. Let us define the cost of `rev/1` in terms of n:

$$\mathcal{C}_n^{\texttt{rev/1}} \triangleq [\![\texttt{rev(}e\texttt{)}]\!] \quad \text{where } \texttt{len(}e\texttt{)} = n$$

Using this definition, the previous equations are equivalent to

$$\begin{aligned}
\mathcal{C}_0^{\texttt{rev/1}} &= 1 \\
\mathcal{C}_{n+1}^{\texttt{rev/1}} &= 1 + \mathcal{C}_n^{\texttt{rev/1}} + \mathcal{C}_{n,1}^{\texttt{join/2}} && \text{since } \texttt{len(rev(T))} = \texttt{len(T)} \\
&= 1 + \mathcal{C}_n^{\texttt{rev/1}} + (n+1) && \text{where } n \geqslant 0 \\
&= 2 + n + \mathcal{C}_n^{\texttt{rev/1}}
\end{aligned}$$

Replacing n by $n-1$, we get

$$\mathcal{C}_n^{\texttt{rev/1}} = 1 + n + \mathcal{C}_{n-1}^{\texttt{rev/1}} \qquad \text{where } n > 0$$

We would like rather to express $\mathcal{C}_n^{\texttt{rev/1}}$ in terms of n only. The trick consists in making the difference of two successive terms in the series and then summing up these differences:

$$\begin{aligned}
\mathcal{C}_n^{\texttt{rev/1}} - \mathcal{C}_{n-1}^{\texttt{rev/1}} &= 1 + n && \text{where } n > 0 \\
\sum_{n=1}^{p} (\mathcal{C}_n^{\texttt{rev/1}} - \mathcal{C}_{n-1}^{\texttt{rev/1}}) &= \sum_{n=1}^{p} (1+n) && \text{where } p > 0 \\
\mathcal{C}_p^{\texttt{rev/1}} - \mathcal{C}_0^{\texttt{rev/1}} &= \sum_{n=2}^{p+1} n \\
\mathcal{C}_p^{\texttt{rev/1}} - 1 &= \sum_{n=2}^{p+1} n \\
\mathcal{C}_p^{\texttt{rev/1}} &= \sum_{n=1}^{p+1} n
\end{aligned}$$

Let us double each side of the equality:

$$2 \times \mathcal{C}_p^{\texttt{rev/1}} = 2 \times \sum_{n=1}^{p+1} n = \sum_{n=1}^{p+1} n + \sum_{n=1}^{p+1} n$$

Let us change n into $p - n + 2$ in the second sum:

$$\begin{aligned}
2 \times \mathcal{C}^{\texttt{rev/1}}_{p} &= \sum_{n=1}^{p+1} n + \sum_{n=1}^{p+1} (p - n + 2)\\
&= \sum_{n=1}^{p+1} (n + (p - n + 2))\\
&= \sum_{n=1}^{p+1} (p + 2) = (p + 1)(p + 2)
\end{aligned}$$

By changing p into n we finally get

$$\mathcal{C}^{\texttt{rev/1}}_{n} = (n + 1)(n + 2)/2 \qquad\qquad \text{where } n > 0$$

Since that replacing n by 0 in the latter formula gives

$$\mathcal{C}^{\texttt{rev/1}}_{0} = (0 + 1)(0 + 2)/2 = 1$$

we can therefore gather the two cases into one as

$$\mathcal{C}^{\texttt{rev/1}}_{n} = (n + 1)(n + 2)/2 \quad \text{where } n \geqslant 0$$

Let us take an example:

$$\begin{aligned}
\texttt{rev([1,2,3])} &\xrightarrow{4} \texttt{join(rev([2,3]),[1])}\\
&\xrightarrow{4} \texttt{join(join(rev([3]),[2]),[1])}\\
&\xrightarrow{4} \texttt{join(join(join(rev([]),[3]),[2]),[1])}\\
&\xrightarrow{3} \texttt{join(join(join([],[3]),[2]),[1])}\\
&\xrightarrow{1} \texttt{join(join([3],[2]),[1])}\\
&\xrightarrow{2} \texttt{join([3|join([],[2])],[1])}\\
&\xrightarrow{1} \texttt{join([3|[2]],[1])}\\
&= \texttt{join([3,2],[1])}\\
&\xrightarrow{2} \texttt{[3|join([2],[1])]}\\
&\xrightarrow{2} \texttt{[3|[2|join([],[1])]]}\\
&\xrightarrow{1} \texttt{[3|[2|[1]]]}\\
&= \texttt{[3,2,1]}
\end{aligned}$$

The formula for the cost stated $\mathcal{C}^{\texttt{rev\_join/2}}_{\texttt{len([1,2,3])}} = \mathcal{C}^{\texttt{rev\_join/2}}_{3} = (3+1)(3+2)/2 = 10$, and we indeed made 10 rewrites until reaching the value `[3,2,1]`.

Asymptotically, we have the equivalence

$$\mathcal{C}_n^{\texttt{rev/1}} \sim \frac{1}{2}n^2 \quad \text{where } n \to +\infty$$

The idea behind our concept of cost is that the time spent in computing an expression is proportional to the cost. So, here, it means that the time needed to reverse a list by `rev/1` is proportional to half of the square of the list length, when the list is big enough. In particular, multiplying by 10 the size of a big input list will multiply at least by 50 the computing time.

2.2 A better version

Consider the following list reversing using an *accumulator*:

```
-module(rev_bis).
-export([rev_bis/1]).
rev_bis(L)        -> rev_join(L,[]).       % Clause 5

rev_join([],A)    -> A;                    % Clause 6
rev_join([H|T],A) -> rev_join(T,[H|A]).    % Clause 7
```

The cost of a function call like $[\![\texttt{rev\_bis(}e\texttt{)}]\!]$ is, by definition, the sum of the costs of rewriting e and the cost of rewriting the call with the corresponding value $\mathcal{V}(e)$ as argument. The same idea applies to $\texttt{rev\_join(}e_1\texttt{,}e_2\texttt{)}$.

$$[\![\texttt{rev\_bis(}e\texttt{)}]\!] \triangleq [\![e]\!] + [\![\texttt{rev\_bis(}\mathcal{V}(e)\texttt{)}]\!]$$
$$[\![\texttt{rev\_join(}e_1\texttt{,}e_2\texttt{)}]\!] \triangleq [\![e_1]\!] + [\![e_2]\!] + [\![\texttt{rev\_join(}\mathcal{V}(e_1)\texttt{,}\mathcal{V}(e_2)\texttt{)}]\!]$$

Considering each clause separately, we can therefore write the equations defining the costs

$$[\![\texttt{rev\_bis(L)}]\!] = 1 + [\![\texttt{rev\_join(L,[])}]\!]$$
$$[\![\texttt{rev\_join([],A)}]\!] = 1$$
$$[\![\texttt{rev\_join([H|T],A)}]\!] = 1 + [\![\texttt{rev\_join(T,[H|A])}]\!]$$

Let us now define the cost in terms of the size of the input, i.e., the argument. The function call $\texttt{rev\_bis(}e\texttt{)}$ has one list as argument. The usual way to measure the size of a list is to consider the number of elements it contains, i.e., its length. Here, the size of the input is characterized by an integer n where $n = \texttt{len(}e\texttt{)}$. The same line of thought applies to `rev_join/2`. Let us define the cost of `rev_bis/1` and `rev_join/2` in terms of the input size:

$$\mathcal{C}_n^{\texttt{rev\_bis/1}} \triangleq [\![\texttt{rev\_bis(}e\texttt{)}]\!] \qquad \text{where } \texttt{len(}e\texttt{)} = n$$
$$\mathcal{C}_{n,m}^{\texttt{rev\_join/2}} \triangleq [\![\texttt{rev\_join(}e_1\texttt{,}e_2\texttt{)}]\!] \quad \text{where } \texttt{len(}e_1\texttt{)} = n \text{ and } \texttt{len(}e_2\texttt{)} = m$$

Using these definitions, the previous equations are equivalent to

$$\mathcal{C}_n^{\texttt{rev\_bis/1}} = 1 + \mathcal{C}_{n,0}^{\texttt{rev\_join/2}}$$

$$\begin{aligned}
&\mathcal{C}^{\texttt{rev\_join/2}}_{0,m} = 1\\
&\mathcal{C}^{\texttt{rev\_join/2}}_{n+1,m} = 1 + \mathcal{C}^{\texttt{rev\_join/2}}_{n,m+1} \qquad\qquad \text{where } n, m \geqslant 0
\end{aligned}$$

We would like rather to express $\mathcal{C}^{\texttt{rev\_bis/1}}_{n}$, respectively $\mathcal{C}^{\texttt{rev\_join/2}}_{n,m}$, in terms of n only, respectively n and m. The trick consists in making the difference of two successive terms in the series $\{\mathcal{C}^{\texttt{rev\_join/2}}_{n,m}\}_n$ and then summing up these differences:

$$\begin{aligned}
\mathcal{C}^{\texttt{rev\_join/2}}_{n,m} - \mathcal{C}^{\texttt{rev\_join/2}}_{n-1,m+1} &= 1\\
\mathcal{C}^{\texttt{rev\_join/2}}_{n-1,m+1} - \mathcal{C}^{\texttt{rev\_join/2}}_{n-2,m+2} &= 1\\
&\vdots\\
\mathcal{C}^{\texttt{rev\_join/2}}_{1,m+n-1} - \mathcal{C}^{\texttt{rev\_join/2}}_{0,m+n} &= 1
\end{aligned}$$

$$\begin{aligned}
\mathcal{C}^{\texttt{rev\_join/2}}_{n,m} - \mathcal{C}^{\texttt{rev\_join/2}}_{0,m+n} &= n\\
\mathcal{C}^{\texttt{rev\_join/2}}_{n,m} - 1 &= n\\
\mathcal{C}^{\texttt{rev\_join/2}}_{n,m} &= n + 1 \qquad\qquad \text{where } n > 0
\end{aligned}$$

Since that replacing n by 0 in the last formula leads to

$$\mathcal{C}^{\texttt{rev\_join/2}}_{0,m} = 0 + 1 = 1$$

we can gather all the cases into one formula:

$$\mathcal{C}^{\texttt{rev\_join/2}}_{n,m} = n + 1 \qquad\qquad \text{where } n, m \geqslant 0$$

Therefore

$$\mathcal{C}^{\texttt{rev\_bis/1}}_{n} = n + 2 \qquad\qquad \text{where } n \geqslant 0$$

Let us take an example:

$$\begin{aligned}
\texttt{rev\_bis([1,2,3])} &\xrightarrow{5} \texttt{rev\_join([1,2,3],[])}\\
&\xrightarrow{7} \texttt{rev\_join([2,3],[1])}\\
&\xrightarrow{7} \texttt{rev\_join([3],[2,1])}\\
&\xrightarrow{7} \texttt{rev\_join([],[3,2,1])}\\
&\xrightarrow{6} \texttt{[3,2,1]}
\end{aligned}$$

The formula for the cost stated $\mathcal{C}^{\texttt{rev\_bis/1}}_{\texttt{len([3,2,1])}} = \mathcal{C}^{\texttt{rev\_bis/1}}_{3} = 3 + 2 = 5$ and we indeed made 5 rewrites before reaching the result `[3,2,1]`.

Asymptotically, we have the law

$$\mathcal{C}_n^{\texttt{rev\_bis/1}} \sim n \quad \text{where } n \rightarrow +\infty$$

The idea behind our concept of cost is that the time spent in computing an expression is proportional to the cost. So, here, it means that the time needed to reverse a list by `rev_bis/1` is proportional to the length of the list, when the list is long enough. We can also now compare the cost of `rev_bis/1` with `rev/1`:

$$\begin{aligned} \mathcal{C}_n^{\texttt{rev/1}} &= \frac{n+1}{2}\,\mathcal{C}_n^{\texttt{rev\_bis/1}} \\ \mathcal{C}_n^{\texttt{rev/1}} &\sim \frac{n}{2}\,\mathcal{C}_n^{\texttt{rev\_bis/1}} && \text{where } n \rightarrow +\infty \end{aligned}$$

Clearly, `rev_bis/1` is better than `rev/1`.