[bana]

Analysis of F# Programming

Research goals of this project:

- Explore typical use cases, best practices,...
- Describe the properties of F# regarding parallel, concurrent execution
- Syntactical refactoring tool
- Code analyzer implementing syntactical software metrics:
  - line of code
  - usage of data structures and different programming constructs
  - **resource analysis**

Simplified phases of resource analysis

- F# source → Core program → Function group (component) → Mathematical equations → Math Formulas

Exact cost calculation

- $\mathbb{R}_\infty = [0, \infty]$ (a closed interval)
- $\kappa_0, \kappa_1, \kappa_2, \ldots, \pi, \eta, \alpha_1, \alpha_2, \zeta_1, \zeta_2, \ldots \in [0, \infty]$

  $[\![e]\!] \in \mathbb{R}_\infty$

Cost calculation – example 1

The cost of a saturated constructor application is κ_n where n is the arity of the constructor:

$$[\![C \quad e_1 \ldots \quad e_n]\!] = \kappa_n.$$

κ_n can be explained as the cost of the creation of a constructor with n parameters in the memory.

Cost calculation – example 2

Case Expressions

Let π denote the cost of pattern matching. Then the cost of case distinction is:

- $[\![\texttt{case}\ \ (C\ f_1\ \ldots\ f_m)\ \texttt{of}\ p_1 \to e_1; \ldots; p_n \to e_n]\!] = \pi + [\![e_i[v_1 := f_1, \ldots, v_m := f_m]]\!]$
  if $(p_i = C\ v_1\ \ldots\ v_m)$ is the first matching pattern;
- $[\![\texttt{case}\ \ e\ \texttt{of}\ p_1 \to e_1; \ldots; p_n \to e_n]\!] = \pi + [\![e_n]\!]$
  if $p_n = _$ and the previous rule is not applicable.

Example: $[\![\texttt{case} \sim[]\ \texttt{of}\ [] \to \text{True}; _ \to \text{False}]\!] = \pi + [\![\text{True}]\!] = \pi + \kappa$

let expressions, nativ functions applications, currying,....

```
length l = case l of
     []    → 0
     x:xs → 1 + length xs
```

$$[\![\text{length List}_0]\!] =$$
$$\alpha_1 + [\![\textbf{case List}_0 \textbf{ of } [] \to 0;\ \text{x:xs} \to 1 + \text{length xs}]\!] =$$
$$\alpha_1 + \pi + [\![0]\!] =$$
$$\alpha_1 + \pi + \kappa.$$

$$[\![\text{length List}_{n+1}]\!] =$$
$$\alpha_{1|} + [\![\textbf{case List}_{n+1} \textbf{ of } [] \to 0;\ \text{x:xs} \to 1 + \text{length xs}]\!] =$$
$$\alpha_1 + \pi + [\![1 + \text{length List}_n]\!] =$$
$$\alpha_1 + \pi + \kappa + \delta + [\![\text{length List}_n]\!].$$

$$f(0) = \alpha_1 + \pi + \kappa;$$
$$f(n+1) = \alpha_1 + \pi + \kappa + \delta + f(n)$$

```
reverse l = case l of
    []  → l
    (x:xs) → reverse xs ++ [x]
```

.....

$$[\![\text{reverse List}_n]\!] = n(2\alpha_2 + 2\pi + \kappa_2 + \kappa + n(\alpha_2 + \pi + \kappa_2)) + \alpha_1 + \pi$$

From this we can see the reverse algoritm is $O(n^2)$

```
StartsWith::String->String->Bool
StartsWith s p = case p of
    [] → True
    x::xs → case s of
      [] → False
      y::ys → x == y && ys 'StartsWith' xs
```

F(_ , 0) = a + b
F(0 , _) = a + b + c
F(n , m) = 2a + 2d + k + F(n-1,m-1)

By hand-calculating we know that the solution is:
F(n,m) = a+b+c + (2a+2d+k) * **min(n,m)**

In 2 or more parameters case ?

Mathematica usually can't solve these.

Maybe it doesn't need directly to solve these

because … tricks...like min

and we can eleminate the more parameters ?

To get to know better the Theorema ?

With other add-in package ?