Program Optimization

Exercise Sheet 12
24.01., due 30.01. 12:30

Exercise 1: Functional Inlining

Perform function inlining on the following program.

```ocaml
let f y = 
  let z = y + 1 in 
    5 + y 
  in 
let x = 5 in 
  f x + f 7
```

**Solution:** We first remove useless constant definitions to obtain:

```ocaml
let f y = 5 + y in 
let x = 5 in 
  f x + f 7
```

Then we perform the inlining of f and obtain the following program:

```ocaml
let x = 5 in 
  (let y = x in 5 + y) + 
  (let y = 7 in 5 + y)
```

Exercise 2: Specialization

Consider the following program:

```ocaml
let rec fold = fun f -> fun a -> fun l ->
  match l with
  | [] => a
  | x :: xs => f x (fold f a xs)
in 
let f = fun x -> fun s -> x + s in 
let sum = fold f 0
```

Perform function specialization and inlining.

**Solution:** We first specialize the first argument \( f \). After inlining we obtain:

```ocaml
let rec fold_f = fun a -> fun l ->
```
match l with
  [] => a
  | x::xs => x + (fold_f a xs)

in
let sum = fold_f 0

Next, we also specialize the second argument. After inlining we obtain:

let rec fold_f_0 = fun l ->
  match l with
  [] => 0
  | x::xs => x + (fold_f_0 xs)

in
let sum = fold_f_0

We also notice that sum is an alias for fold_f_0, so we can perform one more simplification:

let rec sum = fun l ->
  match l with
  [] => 0
  | x::xs => x + (sum xs)

Exercise 3: Deforestation
Let comp, id, map, foldl, and tabulate be defined as in the lecture. Consider the following code:

let sq x = x \times x
let sumSquares = comp (foldl (+) 0) (comp (map sq) (tabulate id))

Simplify the function sumSquares as much as possible.
Solution:

\[
\begin{align*}
\text{comp} (\text{foldl} (+) 0) (\text{comp} (\text{map} \ sq) (\text{tabulate} \ \text{id})) \\
&\equiv \text{by} \ \{ \text{comp} (\text{map} \ f) (\text{tabulate} \ g) = \text{tabulate} (\text{comp} \ f \ g) \} \\
\text{comp} (\text{foldl} (+) 0) (\text{tabulate} (\text{comp} \ sq \ id)) \\
&\equiv \text{by} \ \{ \text{comp} \ f \ id) = f \} \\
\text{comp} (\text{foldl} (+) 0) (\text{tabulate} \ sq) \\
&\equiv \text{by} \ \{ \text{comp} (\text{foldl} \ f \ a) (\text{tabulate} \ g) = \text{loop} (\text{comp}_{2} \ f \ g) \ a \} \\
\text{loop} (\text{comp}_{2} (+) \ sq) 0
\end{align*}
\]

Exercise 4: Strictness Analysis
Consider the following algorithm:

let rec ea x y =
  if x = 0 then y else ea (y mod x) x
Use the strictness analysis to detect if the function is strict on both arguments. 

**Solution:** The equation for $ea$ is:

$$ea \equiv \begin{cases} b_1 \land 1 & \land (b_2 \lor ea \land b_1) \\ \equiv b_1 \land (b_2 \lor ea \land b_1) \land b_1 \end{cases}$$

The fixpoint iteration is as follows:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$ea$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>fun $b_1 \rightarrow$ fun $b_2 \rightarrow 0$</td>
</tr>
<tr>
<td>1</td>
<td>fun $b_1 \rightarrow$ fun $b_2 \rightarrow b_1 \land b_2$</td>
</tr>
<tr>
<td>2</td>
<td>fun $b_1 \rightarrow$ fun $b_2 \rightarrow b_1 \land b_2$</td>
</tr>
</tbody>
</table>

Thus, $ea$ is strict in both its arguments.