Script generated by TTT

Title: Seidl: Functional Programming and Verification (18.01.2019)

Date: Fri Jan 18 08:29:47 CET 2019

Duration: 90:37 min

Pages: 7

Discussion

- The lemma tells us that in every context, all occurrences of the expression e_1 can be replaced by the expression e_2 — whenever e_1 and e_2 represent the same values.
- The lemma can be proven by induction on the depth of the required derivations (which we omit).
- The exchange of expressions proven equal, allows us to design a calculus for proving the equivalence of expressions ...

Rule for pattern matching

$$\frac{e_0 = []}{\texttt{match}\ e_0\ \texttt{with}\ [] \texttt{->} e_1 \mid \ldots \mid p_m \texttt{->} e_m \ = \ e_1}$$

$$\frac{e_0 \text{ terminates} \qquad e_0 = e_1' :: e_2'}{\texttt{match}\ e_0\ \texttt{with}\ [] \texttt{->} e_1 \mid x :: xs \texttt{->} e_2 \ = \ e_2[e_1'/x, e_2'/xs]}$$

Structured values

$$\frac{e_1 = e_1' \ \ldots \ e_k = e_k'}{(e_1, \ldots, e_k) = (e_1', \ldots, e_k')}$$

$$\frac{e_1 = e_1' \qquad e_2 = e_2'}{e_1 :: e_2 = e_1' :: e_2'}$$

Functions

$$\frac{e_1[v/x_1] = e_2[v/x_2] \quad \text{for all} \quad v}{\texttt{fun}\ x_1\ \texttt{->}\ e_1 \ = \ \texttt{fun}\ x_2\ \texttt{->}\ e_2}$$

$\Longrightarrow$ extensional equality

Analogously we proceed for assertion (2) ...

$n = 0$ Then: `x = []`

We deduce:

```
app x (app y z)  =  app [] (app y z)
                 =  match [] with [] -> app y z | h::t -> ...
                 =  app y z
                 =  app (match [] with [] -> y | ...) z
                 =  app (app [] y) z
                 =  app (app x y) z
```


$n > 0$ Then `x = h::t` where `t` has length $n - 1$.

We deduce:

```
app x (app y z)  =  app (h::t) (app y z)
                 =  match h::t with [] -> app y z
                        | h::t -> h :: app t (app y z)
                 =  h :: app t (app y z)
                 =  h :: app (app t y) z   by induction hypothesis
                 =  app (h :: app t y) z
                 =  app (match h::t with [] -> []
                          | h::t -> h :: app t y) z
                 =  app (app (h::t) y) z
                 =  app (app x y) z
```


$n > 0$ Then `x = h::t` where `t` has length $n - 1$.

We deduce:

```
app x (app y z)  =  app (h::t) (app y z)
                 =  match h::t with [] -> app y z
                        | h::t -> h :: app t (app y z)
                 =  h :: app t (app y z)
                 =  h :: app (app t y) z   by induction hypothesis
                 =  app (h :: app t y) z
                 =  app (match h::t with [] -> []
                          | h::t -> h :: app t y) z
                 =  app (app (h::t) y) z
                 =  app (app x y) z
```


Discussion

- For the correctness of our induction proofs, we require that all occurring function calls terminate.
- In the example, it suffices to prove that for all `x, y`, there exists some v such that:

  `app x y` $\Rightarrow v$

  ... which we have already proven, as usual, by induction.