Project 4 – COMP301

SPRING'23

Hüseyin Şenol – Batuhan Arat – Metin Arda Oral

Workload

Strictly equally distributed. We met 3 times to understand the topics and share ideas. The implementation is mostly done while we are together, and missing parts are distributed among the team members.

Part 1 – Display How Many Times

In "data_structures", we constructed "nested-procedure" and "extend-env-rec-nested". As the previous versions are available in the code, all we had to do was just to add the required parameters, which were pretty straightforward implementations.

In "environments", we first complete the part in "extend-env". Simply, we catch the procedures with cases, and then we catch the nested procedures in the second cases. In both "else" conditions in the two cases call, we return val (what we do normally when search-sym = var). "extend-env-rec-nested" part is again very straightforward, we almost do the same implementation as given "extend-env-rec", however this time, instead of "procedure" we use "nested-procedure" and corresponding parameters.

In "lang", everything is usual as we did in previous projects. We implemented the required parts, directly leading the pdf instructions.

In "interp", proc-nested-exp, call-nested-exp, and letrec-nested-exp are implemented. proc and letrec implementations are similar to above. In the call-nested-exp, we also define a counter, which is defined from count, and we cons it with arg to pass to the apply-procedure, to use in that. In the "apply-procedure", we create a case "nested-procedure", and display "name" and "cdr arg" (which is count) with provided "recursive-displayer-name". Then we extend-env two times, for var and count.

In "translator", we again implemented proc-exp and letrec-exp in the same way. This means, translation-of proc-exp is proc-nested-exp with corresponding parameters (var 'count (from env) 'anonym (as indicated) and translation of body. Letrec-exp is similar. In the call-exp, we first check with cases whether the rator is var-exp or not. If it is, we need to add 1, since it has been used before. If not, we simply pass const-exp 1.

Part 2 – Translator Modification

We started by implementing the apply-senv-number. The idea is simple: recursively iterate the senv and check the equivalence of the var with the (car senv), if equal, return 1 + (apply-senv-number (cdr senv) var)). Then, after we have the method to calculate the number of occurrences of a variable in a senv, we implemented the var-exp: calculate the number with the apply-senv-number and return string-append(var+number).

After that, we implemented lett-exp and proc-exp with a similar idea, however since there can be recursive expressions, we again calculated number & call lett-exp/proc-exp again,

```
(lett-exp new-var
 (translation-of exp1 senv)
 (translation-of body (extend-senv var senv)))
```

new-var above refers to the new variable name x1,x2, etc.

The most difficult step for us is to figure out a way to display the shadowing explanations on the **middle** of the scan&parse output. We approached the problem from the wrong direction and spent a couple of hours trying everything we could think of. Then it clicked, and we figured that we can give the explanation as the variable to the lett-exp/proc-exp call. So, we changed the new-var from the new variable with the number of occurrences (x1, x2, etc.) to the whole shadowing explanation instead, if the number was bigger than zero. The desired output followed.