Warm-up Project

This assignment is based on the project described in *Modern Compiler Implementation in ML*, section **PROGRAM**, pages 10-12. The source code mentioned in that section (as being available from `$Tiger/chap1`) should be taken from the Documents (https://bb.au.dk/webapps/blackboard/content/listContent.jsp?course_id=_43271_1&content_id=_355480_1&mode=reset) section of this website, as should source code used as the starting point for all projects in this course. **Additional requirement:** In addition to the description of tasks on pages 10-12, you should create 5 expressions, specify them as abstract syntax (like the definition of `prog` in `warmup.sml`), interpret them, and check that they behave as expected.

Note that the code in `warmup.sml` uses a function `buildEnv` (whose implementation is omitted), which hints at an alternative approach. In the approach described in the book, the environment is empty at first and every assignment at runtime causes the environment to be extended with a new binding; lookup will always select the most recently added binding, so this works as expected for assignment (although it will waste memory if the same variable is assigned a new value many times). In the alternative approach, there is a separate ("compilation") phase first where the program is analyzed to discover the exact set of variables that may be needed, and only then is the program executed (in a separate phase which would typically be called "runtime"). In this approach the environment should never grow, it always contains k bindings for a program using k different variables; to implement assignment the environment is rebuilt with the new value. For example, `[("a",3),("c",4)]` becomes `[("a",3),("c",7)]` when an assignment to `c` changes it from 4 to 7.

Very dynamic languages (e.g., JavaScript) require the approach where the environment is modified at runtime, but for compiled languages it is typical to have environments that are fixed during compile time and valid for all executions. The fixed format allows for optimizations and enables certain correctness guarantees to be established, e.g., in the shape of statically checked types.

You may choose to do strictly as described in the book (hence deleting the reference to buildEnv), and you may choose to implement two separate phases (which includes implementing buildEnv), or you may choose to handle both approaches. In any case the report should describe what you have done. It is also worth mentioning that a functional style can be used (avoiding `ref`) for both approaches.

The source code in your solution must be compilable and runnable on the standard platform of the course, and when interpreting the expression from Fig.1.4 (i.e., `prog`) it should output the following:

```
8 7
80
```

Notice there is a difference between call-by-value and call-by-name execution. For this small language you should use call-by-value.

Deliverables

- *Source code to your solution.* The source code must be compilable and runnable on the standard platform of the course. The solution must also include the source to `maxArgs` function.
- *5 expressions that you created.* This should also be included in the report (see the point below).
- *Project report.* The project report should be approximately 1-2 pages. It may be formatted using LaTeX or other document processing systems and delivered as a PDF file (the Documents section of the site includes sample templates). Use the report to describe your approach. Highlight the differences from the project description, problems encountered during the assignment, and/or interesting experiments. It must contain the 5 expressions you created, as source code and as abstract syntax, along with a short description of their

expected behavior during interpretation.

Submission instructions

TBD

Deadline

Wednesday, September 9th, 14:00.

Content on this page is maintained by Aslan Askarov.