Formal Semantics of Programming Languages
Class Project

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
Wolfgang.Schreiner@risc.jku.at

April 15, 2015

Rather than elaborating the exercises, you may also perform this project to receive a grade for the course. There is no deadline for submission, but if you decide to perform the project, send me an email in which you indicate in which programming language you intend to write the implementation.

The project is to be performed individually (discussions are okay, but the detailed elaboration of the semantics and its implementation have to be performed individually).

Semantics and Interpretation of a Simple Subset of C

Take a simple subset of C/C++ with the following features:

- We can declare variables of type `int` (integers on the stack) or `int*` (pointers to integer arrays on the heap).
- We have integer literals, we can add, subtract, multiply, divide and compare integers.
- We can create a pointer to a new integer array by a call `new int[`*n*`]` and access its contents by indexing.
- We can assign integers to `int` variables and integer pointers to `int*` variables.
- We have sequencing of commands, one- and two-sided conditionals, and while loops.
- We have blocks of commands that can arbitrarily mix declarations and commands.
- We have print statements that can print a single integer.

A program in this language is a command block.

A sample program is

```
{
  int n;  n = 10;
  int* a; a = new int[n];
  int i;  i = 0;
  while (i < n)
  {
    int j; j = i*i;
    a[i] = j;
    i = i+1;
  }
  if (i > 0) printf("%d", a[i-1]);
}
```

Your task is as follows:

1. Give a formal definition of the abstract syntax of this language (syntactic domains and syntax rules).

2. Give a denotational semantics of this language (semantic algebras, signatures of valuation functions, definitions of valuation functions).

   Types are checked at runtime; any error detected at runtime shall let the program abort with a result which contains the output produced up to the error and indicates by a string the source of the problem.

   A valuation function **P** shall exhibit the observable behavior of program *P*, i.e. $\mathbf{P}[\![P]\!]$ shall denote the sequence of numbers printed and whether the program has terminated normally (and, if not, the description of the error).

3. Implement the semantics in a language of your choice, e.g., ML, Haskell, Scala, Java, F#, C#. Apart from the syntactic domains (abstract syntax trees), you have to implement the semantic domains, and the valuation functions.

   The implementation shall correspond as much as possible to the semantics; e.g. there shall exist a function `P` that takes the abstract syntax tree of program *P* and returns the sequence of numbers printed by *P* together with an indication whether an error occurred and, if yes, which one.

The deliverable of this project is a zip/tgz file which includes

1. a report including the semantics and the formatted source code of the implementation, together with the outputs of some test runs (showing programs that run without failure and programs that fail), and any comments/explanations you would like to give.
2. the source code of the implementation.

When you are done, upload the first version of the deliverable to the Moodle site and send me an email; I will shortly check it, and possibly send you some requests for improvements. Then we will have a meeting where we will discuss the semantics and you will demonstrate to me the implementation.

If during the elaboration of this project questions arise, feel free to contact me.