Student's Project for Compiler Construction: Compiling a Fragment of SetlX to *Java* — Task Description —

Prof. Dr. Karl Stroetmann

March 5, 2013

This note specifies the task that is given to those students that in Stuttgart during the summer term. The task is to implement a small *cross language compiler*. This compiler takes as input a program written in the source language SetlX and produces as output *Java* source code, so the target language is *Java*. In order to execute the resulting *Java* programs, these programs would then be compiled with the traditional `javac` command into *Java* class files.

An interpreter for the programming language SetlX is available at

`http://wwwlehre.dhbw-stuttgart.de/~stroetma/SetlX/setlX.php`,

while a tutorial describing SetlX can be found at

`http://wwwlehre.dhbw-stuttgart.de/~stroetma/SetlX/tutorial.pdf`.

Students are only required to implement a small subset of SetlX. In particular, the fragment of SetlX that has to be implemented has to satisfy the following conditions:

1. The following data types have to be supported:

   (a) Integers of arbitrary precision.

   In *Java*, integers of arbitrary precision can be represented using the class `BigInteger`.

   (b) Sets of arbitrary nesting depth. The elements of these sets can be both integers and sets of arbitrary nesting depth. It is a little bit tricky to work with these kinds of sets in Java, as the class

   TreeSet

   does not implement the interface `Comparable` and does not define a method that can be used to compare sets. However, if some class E is given and a set of objects of class E is to be constructed as `TreeSet<`E`>`, then the class E is required to either provide a method `compareTo()` or an object of type `Comparator` that itself provides a method `compare()`.

   Therefore, I have provided a class called `ComparableSet` that is provided at

   http://www.dhbw-stuttgart.de/stroetmann/Compiler/ComparableSet.java

   that overcomes these limitations. The basic functionality of this class is exercised by the class

   http://www.dhbw-stuttgart.de/stroetmann/Compiler/TestSet.java

   The implementation of these classes will be discussed in detail in the lecture.

   While the nesting of sets should be unrestricted, it may be assumed that the sets are *homogeneous*: Therefore, a set either contains only integers or it will contain sets, but it will never contain both integers and sets.

2. For integers, the arithmetical operators

"+", "-", "*", "/", and "**"

have to be supported. Here the operator "**" denotes exponentiation. For comparisons, you have to support the operators

"<", "<=", "==", and "!=".

3. For sets of integers, the operators

   "+", "-", "*", and "**".

   have to be supported. For two sets s and t,

   (a) s + t denotes the union of s and t,
   (b) s - t denotes the difference of s without t, while
   (c) s * t denotes the intersection of s and t.
   (d) 2 ** s denotes the power set of s.

4. Furthermore, the following methods to construct sets have to be supported:

   (a) Construction by explicit enumeration of the elements. For example,

   `{1, 2, 3}`

   is the set containing the integers 1, 2, and 3.

   (b) Construction as a range. For example,

   `{2..7}`

   is the set containing the integers 2, 3, 4, 5, 6, and 7.

   (c) Construction as an *image set*. An image set has the form

   { *expr*: x_1 `in` s_1, $\cdots$, x_n `in` s_n }.

   Here *expr* is an expression containing the variables $x_1, \cdots\ x_n$, while $s_1, \cdots, s_n$ denote sets. The resulting set contains all values of *expr* where the variables x_i have been substituted with values from the sets s_i. For example, the set

   `{ p * q: p in {1..10}, q in {1..10}}`

   contains the set of all products of positive natural numbers p and q such that both p and q are less than 10.

   For comparisons of sets, you have to support the operators

   "<", "<=", "==", and "!=". For given sets s and t, the semantics of these operators is as follows:

   (a) s < t if and only if s is a proper subset of t, i.e. if $s \subset t$ holds.
   (b) s <= t if and only if s is a subset of t, i.e. if $s \subseteq t$ holds.
   (c) s == t if and only if s and t contain the same elements.
   (d) s != t if and only if s and t do not contain the same elements.

   Furthermore, you have to support the binary operator "`in`". The expression

   x `in` s

   is true iff x is an element of s.

5. The following control structures have to be supported:
   (a) `if (test) {` *body* `}`
   (b) `if (test) {` *body*$_1$ `} else {` *body*$_2$ `}`
   (c) `while (test) {` *body* `}`
   (d) Both the definition and the invocation of functions have to be supported.
6. The tests used in the control structures have to be Boolean expressions. Boolean expressions support the operators

   `&&`, `||`, and `!`.

   These operators have the same meaning as in `C` or *Java*.

The compiler has to be able to translate the program shown in Figure 1 into a working *Java* program. This program should compute the prime numbers less than or equal to n. As this program tests only a small number of the required features, your task is to implement additional SetlX programs that test the remaining features.

```
primes := procedure(n) {
    s := { 2 .. n };
    return s - { p * q : p in s, q in s };
};

print(primes(100));
```

Figure 1: A SetlX program to compute the prime numbers less than n.

Deliverables: You should combine all your source files that are needed to build your compiler in one zip file. This file should be named

your-name.zip

where *your-name* has to be replaced by a combination of your name and the name of your partner. For example, the zip file could have the name `fox-meyers.zip` if your name is fox and your partner is called meyers. Unzipping this file has to produce a directory with the name *your-name*. This directory should contain only source files, it must not contain any "`.class`" files. Furthermore, this directory has to contain a `Makefile`. Running the command

`make`

should build the compiler and, furthermore, it should perform a number of tests. These tests have to consist of three steps:

1. In the first step, they have to compile a SetlX source file into a *Java* file.
2. In the second step, the resulting *Java* source file should be translated into a *Java* class file via an invocation of the command `javac`.
3. In the final step, the *Java* class file should be tested using the *java* command.

You are required to test your deliverable. If I unzip your deliverable and discover that running `make` does not work as described above, you have **failed**. Therefore, make sure to test everything in a *Unix* or *Linux* environment. If you are working with windows, you should test your code using the tools known as `cygwin`:

`http://www.cygwin.org`.

You are allowed to share your test files with other groups, but you are **not allowed** to share your scanner, your grammar, or any other *Java* code.