CS 131: Programming Languages, Spring 2003

Assignment 4
Pic2ps, Part 2

Due: Wednesday, February 26

Goals for this Assignment

1. Writing an interpreter for a nontrivial imperative language.

Office Hours

The purpose of office hours is to provide you with a chance to discuss class-related topics, to ask questions about material you find particularly interesting or particularly confusing, and to provide comments about the class.

You can drop by Olin 1253 any time to see if the professor is around; times specifically set aside are Tuesday, Thursday, and Friday from 2:30–4pm. Grutors are available in the Terminal Room on Tuesdays 7–9pm and Sundays 2–4pm. Questions can also be sent to `cs131help@cs.hmc.edu` .

Instructions

This assignment requires extending the translator from the previous assignment; you may start with a copy of your own code (recommended if it works), or from the sample solution in the directory `/cs/cs131/src/sol3`. In either case, you should **then** copy in everything from the directory `/cs/cs131/src/4`; this replaces some of the files you were given in Assignment 2 and and adds a few new ones.

Please submit *only* your `pic2ps.sml` file, using `cs131submit`.

As before, if you need more information about *pic* you can look at Bentley's chapter *Little Languages*, or the Pic User Manual. Functions must have the exact names and types given in the assignment (though you are encouraged to define other helper functions as needed), and should be reasonably well-commented in order to help the graders understand your code.

program :
command *program*
eof

command :
primitive *attribute-list* ;
direction ;
var = *exp* ;
{ *command-list* }
`for` *var* = *exp* `to` *exp* `by` *exp* `do` { *command-list* }

primitive : `box` | `circle` | `move` | `arrow` | `line`

direction : `right` | `left` | `up` | `down`

attribute-list :
string *attribute-list*
direction *attribute-list*
ϵ

command-list :
command *command-list*
ϵ

attribute-list :
string *attribute-list*
`right` *optexp* *attribute-list*
`left` *optexp* *attribute-list*
`up` *optexp* *attribute-list*
`down` *optexp* *attribute-list*
ϵ

optexp :
exp
ϵ

exp :
number
var
(*exp* + *exp*)
(*exp* - *exp*)
(*exp* * *exp*)
(*exp* / *exp*)

Figure 1: Concrete Syntax for the Input

1 Abstract Syntax (20%)

The grammar in Figure 1 has been extended to include numbers, expressions, assignment to variables, for-loops, curly-brace-delimited blocks ({...}), and optional lengths for direction specifications. (The grammar requires that most arithmetic expressions be enclosed by parentheses; this makes the grammar for expressions unambiguous and lets us avoid questions of associativity and precedence.)

The lexer and the `Tokens` structure have been extended for you; the new definition of the type `Tokens.token` is

```
datatype token = Left_tok | Right_tok | Up_tok | Down_tok
               | Box_tok | Circle_tok | Move_tok | Arrow_tok | Line_tok
               | String_tok of string | Semi_tok | EOF_tok
               | For_tok | To_tok | By_tok | Do_tok  (* new keywords *)
               | Eq_tok                              (* = *)
               | Lbrace_tok   | Rbrace_tok           (* { and } *)
               | Lparen_tok   | Rparen_tok           (* ( and ) *)
               | Plus_tok | Minus_tok | Times_tok | Divide_tok (* +,-,*,/ *)
               | Num_tok of real                     (* real constants *)
               | Var_tok of string                   (* variables *)
```

In your copy of `pic2ps.sml`, update the definition of the type `command` so that it is suitable for representing the abstract syntax for the enlarged set of possible `pic` programs.

Your changes should include defining a type named `exp` that can be used to represent `pic` arithmetic expressions.

2 Parsing (30%)

Update your recursive-descent parser so that it can handle the entire grammar shown in Figure 1. Your main function should still have type

```
parseProgram : token list -> program
```

where `program` is equivalent to `command list`.

Hints:

- The built-in type `'a option`, defined by

```
datatype 'a option =
   SOME of 'a
 | NONE
```

may be useful.

- Because SML does not like to compare reals for equality using the standard equality operator, and because a value of type `Tokens.token` now might contain a real number, SML will no longer let you compare two tokens with the `=` operator. In case you want to write a function analogous to `eat` in the lecture slides on parsing, a new function `Tokens.eq` has been provided.
- Recall that in SML, mutually-recursive function definitions are connected with `and`.

Recovering Assignment 2 Functionality

As the next step, it is suggested you get your code back into a working state, so that it handles programs that should have worked in Assignment 2, but using the new abstract syntax. That is, get translation working except for the new parts of the input language (length expressions, assignments, for loops, ...).

3 Evaluating Expressions (5%)

The file `picenv.sml` contains a module `PicEnv` which defines a type `PicEnv.pic_env` of environments (functional lookup tables mapping strings to real numbers), along with functions like `PicEnv.lookup` and `PicEnv.extend`. Use these to add a function

```
evalExp : PicEnv.pic_env * exp -> real
```

to your `pic2ps.sml` file (before the translation functions). This function should evaluate the given expression, using the given environment to find the values of variables.

4 Adding Environments (10%)

Next, we want the translator to follow the official implementation by using the *pic* variables `boxwid` and `boxht` when deciding what size box to draw, instead of relying on the (constant) SML variables of the same name. To do this, the translation functions are likely to need an environment as an extra parameter. (All the relevant variable names are given in the *pic* documentation, and also listed near the end of `picenv.sml`).

To this end, the main `translateCmd` should be modified so that it takes an environment as an extra argument, and so that it returns an environment as an extra result:

```
command * (real * real) * direction * PicEnv.pic_env ->
   string * (real * real) * direction * PicEnv.pic_env
```

The argument environment gives the values of the variables (`boxwid`, `circlerad`, etc.) which provide sizes during the translation. The result argument is a modified copy of the given environment updated with any assignments that occurred when interpreting the given command. None of the actual drawing elements inherited from Assignment 2 change the values of variables, so these cases just return the given environment.

Instead of using SML variables such as `circlerad`, your code should look up these names in the environment.

Important: To get your modified code to compile (since the type of `translateElem` has changed), you'll need to do the following two steps:

1. Delete the `pic2ps` function at the end of `pic2ps.sml`
2. Add the filename `main.sml` to the `sources.cm` file

Then you can test your code by running `Main.pic2ps` instead of `Pic2ps.pic2ps`.

Finally, delete the SML variables `boxwid`, `arrowht`, etc. from your `pic2ps.sml` file.

At this point your code should still work on all the examples from Assignment 2, plus you should be able to specify distances for the `left`, `right`, `up`, and `down` attributes of `move`, `line`, and `arrow`.

5 Element lists (5%)

Write a function

```
translateCmds :
    command list * (real * real) * direction * PicEnv.pic_env ->
    string * (real * real) * direction * PicEnv.pic_env
```

that is just like `translateCmd` except that it takes a list of elements, translates each one sequentially (passing the location, direction and environment resulting from one element as inputs for the translation of the following element), and returns the concatenation of all the postscript code along with the final position, direction, and environment.

6 Adding assignments, blocks, and for-loops (30%)

Finally, change `translateCmd` so that it handles assignments, curly-brace-delimited sequences, and for-loops as described in the *pic* documentation. A few hints:

- An assignment element should not cause `translateElem` to return new PostScript code or change the position or direction. It should cause `translateElem` to return an updated environment where the specified variable has a new value. (This is the first case in which `translateElem` does not return the same environment it was given.) Verify that, for example, putting an assignment to the variable `circlerad` in a *pic* test file causes the sizes of your circles to change.
- In *pic* an element that is just a sequence of elements surrounded by curly braces generates PostScript code as usual, but after the block is finished the current position and direction are restored to the values they had when the sequence was started. This can be implemented easily by having `translateElem` return the position and direction

given rather than any values computed during the translation of this block. (The environment is not restored afterwards, however. *pic* does undo all assignments when leaving a block delimited by `[...]`, but you are not being asked to implement that part of the language.)

- `for` *var* `=` exp_1 `to` exp_2 `by` exp_3 `do` { *command-list* } first sets *var* to exp_1, and then while the value of *var* is less than or equal to that of exp_2, we repeatedly draw *command-list* and increment *var* by the value of exp_3.

  The position, direction, and environment computed at the end of each iteration is used for the beginning of the next iteration, except that the value of the loop index variable is updated in the environment before each iteration. The PostScript returned will by the concatenation of the PostScript codes created by each iteration of the loop.

  Warning: the body of a `for` loop is syntactically required to be bounded by curly braces, but this does *not* signal that the position and direction are ever being reset. If you wanted the position and direction to be reset after every iteration of the loop, the loop body must be inside yet another pair of curly braces. (Compare the files `test4.pic` and `test5.pic`)