Project 1

CMCS 22620/32620, Spring 2004

Assigned: April 7, 2004
Due: April 19, 2004

1 Introduction

The purpose of the first project is twofold. The main goal is to get a working implementation of the *translate* module. This module contains functions which get invoked by the type checker / semantic analyzer. Their purpose is to generate expressions and statements in our `Tree` language which correspond to a given Minijava program. The second goal is to (as a side effect of writing the translator) familiarize yourself with the structure of the type checker itself.

2 Instructions

2.1 Files

Download the file `mj-project1.tgz` from the course web page. This is a compressed tarball containing the Minijava compiler's frontend as well as a template for implementing the *translate* module. Here is a roadmap for understanding the purpose of each file:

`symbol.sml` Definition of abstract type representing symbols (i.e., Minijava identifiers).

`ast.sml` Type definitions for abstract syntax trees. Those trees are generated by the parser.

`minijava.grm` The `ml-yacc` grammar for Minijava.

`minijava.lex` The `ml-lex` specification for the Minijava lexer.

`parse.sml` Glue code putting together lexer and parser. For testing purposes this file currently also contains the invocation of the semantic analyzer.

`types.sml` Definitions of the types of the values that the compiler uses to internally represent Minijava types. Such values are used by the semantic analysis phase.

`label.sml` Definition of an abstract type representing labels in assembly code.

`temp.sml` Definition of an abstract type representing an infinite supply of "virtual registers" called *temporaries*. Temporaries will later get mapped to a finite set of physical registers by the register allocator.

`semant.sml` This file contains the semantic analyzer. Semantic analysis involves type checking as well as verifying other static constraints that valid Minijava programs have to satisfy. (For example, `continue` statements must occur within loops, each reference to a Minijava label must be within the so-labeled statement, variable references must be within scope, . . .) The semantic analyzer also contains *translation stubs*—calls of functions provided by the *translate* module. Their purpose is to generate the intermediate representation of the program which is used by the backend. (From the point of view of `semant.sml`, the intermediate form is abstract: a program consists of a list of *fragments*, each fragment representing the code of one method or the runtime data for one class.)

`tree.sml` This file defines the `Tree` language. Trees are used to represent executable code.

`layout.sml` Definition of some "magic numbers" related to memory layout and value representation. These numbers are used by the *translate* module.

translate.sml This is the *translate* module. The file, as given, contains a template which is to be filled in by you. Find expressions of the form **raise Fail "fill this in. . ."**, read the comments, and replace the expressions with whatever it takes to implement the required functionality. In addition to that you should also try and improve the implementation of function `cond` as suggested by the comment that precedes it.

`minijava.cm` This is the CM description file for the Minijava compiler project. It basically lists your source files and the names of modules to be externally visible. In addition to that, it specifies which external libraries are being used. In particular, we have

- `$/basis.cm` The Standard ML Basis Library. Think of this as the "`libc` of SML." For detailed information, see `http://www.standardml.org/Basis`.
- `$/ml-yacc-lib.cm` The `ml-yacc` helper library. (The code generated by `ml-yacc` contains references to functionality exported by this library.)
- `$/smlnj-lib.cm` The SML/NJ library contains implementations of a number of useful data structures. In particular, we are using *red-black trees* to represent *finite maps* and *finite sets*. Among other things, finite maps are convenient for implementing *functional environments*. (Look at, e.g., `Symbol.Map` defined in `symbol.sml`.)
- `$smlnj/viscomp/basics.cm` To avoid reinventing certain wheels, we pull in some functionality from the SML/NJ compiler's implementation itself. In particular, we make used of modules dealing with input from source files, tracking of region information, and generating error messages. (The corresponding modules are `Source`, `SourceMap`, and `ErrorMsg`.)

2.2 SML/NJ

To get started, the first step is to see whether you have access to a working SML/NJ installation. After downloading, uncompressing, and untaring said tarball, you should end up with a directory named `minijava` containing the files listed above. Go to that directory and fire up SML/NJ by typing `sml` at the shell prompt. You should see a greeting from SML/NJ and a new input prompt. At this prompt, type `CM.make "minijava.cm";`. The program should compile.

2.3 Testing

After `CM.make` has succeeded in compiling the code you can now proceed to test it. In particular, the function `Parse.parse` will take the name of a source file containing Minijava code, parse it, and send it to the semantic analyzer. In the beginning you will get failure almost immediately. That is because of the missing blanks in `translate.sml`.

Once you think you have completed `translate.sml`, it will be useful to test the compiler in earnest. Write a few sample Minijava programs and feed them to `Parse.parse`. Once this goes through without error, it will be useful to write a little module for pretty-printing trees of the `Tree` language. Add an interface to the *translate* module that lets you look at the trees you generate.

3 Handing it in

You should only have to make changes to file `translate.sml`. To hand in your solution, send this file as an e-mail attachment to both the instructor and the TA using the following e-mail addresses:

instructor	`blume (at) tti (hyphen) c (dot) org`
TA	`cysong (at) cs (dot) uchicago (dot) edu`

If you make other changes, then bundle all your files as a tarball and attach that to your e-mail.