dOvs - Final Code

Group 9
Miran Hasanagić - 20084902
Jakob Graugaard Laursen - 20093220
Steven Astrup Sørensen - 201206081

December 3, 2015

1 Introduction

This report contains the final code for the implemented compiler. In this phase we looked at the feedback obtained for the different phases, or components, and fix them in order to have fully working compiler.

The rest of this report is structured the following way. The following sections addresses in general terms the different error that have been fixed according to the feedback from the instructor. Afterwards, five tiger programs are presented, which show tiger programs which work correctly after the respective errors have been fixed. Finally, the experience for the whole compiler course is summed up and concluding remarks of the final code are given.

2 Compiler components

In this section the respective components with their error fixes are given.

2.1 Lexer

In this phase there where only minor errors, Mainly 4 errors were observed. 1. A missing comma between states, 2. An escape character could be used at any place, 3. Form Feed wasn't available for use inside the \f_f \sequence. And 4. We didn't have support for all Control Characters.

The Fixes were as following: 1. Insert the comma. 2. Remove the \Character from the list Printable, and make sure to trim one \character away when inserting them into the code again. 3. insert the correct escape sequence into the multi-line states. 4. A function was created that takes as input the Letter of the control character, and takes the ASCII code of that -64 (Which corresponds to the control character for that letter), and convert the into into the character. A special note however is the DEL control character which this doesn't apply to, so that was made a special case.

2.2 Parser

Now the let-expression does not required to at least have one declaration. Additionally, it is not allowed to have a extra comma in function calls, and semicolon in sequences. Finally, having the same precedes twice was removed, since it was unnecessary.

2.3 Semantic Analysis

Now it is not allowed to have a break inside a procedure, which are inside loops. Also it is allowed that a function can return a record with the value nil. Finally, it is not allowed to assign a variable, which has the same name as a for-variable, even though they are not the same variable.

2.4 IR-generation

There was an error with assignments with respect to look up in arrays, e.g. reassigning a arrays element, which has been fixed. Also a runtime error is given, if a lookup/assignment of field for nil.

2.5 Code Generation

A minor error has been fix with respect to function calls in a sequence, in which an assignment of a sequence to a variable.

3 5 tiger programs

This section presents 5 tiger test programs, which illustrated some of the fixes that have been made in the final code.

3.1 test01.tig

```
/*
Test Array, reassignment
*/
let
        type intarray = array of int
        var c := intarray[3] of 43
in
        c[0] := 2;
        c[0] + c[2]
end
```

3.2 test02.tig

```
/* checking that we are doing side effect correctly */
let
        var c := 0
        function increment():int = (c := c+1;1)
        function multiply():int = (c := 2*c;2)
        function test(a:int, b:int):int = c
in
        test(increment(),multiply())
end
```

3.3 test03.tig

```
/*
Checks combination of arrays and record
*/
let
type rectype = {name: string, address: string, id: int, age: int}
type arrtype = array of rectype
var arr := arrtype [5] of
                  rectype{name="aname", address="somewhere", id=0, age=0}
in
```

```
    arr[3].name := "kati";
    arr[1].age := 23;
    arr[1].age
end
```

3.4 test04.tig

```
/*      exponentiation considers only the exponents absolute value
 *      8 + 3 + 1 + 1 + 1 + 0 + 512 = 526
*/
2^3 + 3^4^0 + 0^0 + (-1)^(-2) + 1^(-234) + (-3)^(-43344) + 2^9 = 526
```

3.5 test05.tig

```
/* test for recursive functions
 * current version has an infinite loop for negative cases
*/
let
        function even(i:int):int = if i = 0 then 1
                else if i = 1 then 0
                else odd(i-1)
        function odd(i:int):int = even(i-1)
in
        even(4) = odd(11)
end
```

4 Experience Gain and Concluding Remarks

In general it has been a great experience to develop a full compiler. Furthermore the approach of having the different compiler phases separated into different module made the integration straightforward. Also using SML for programming seems like a natural choice, especially since it is functional and has pattern matching. Also it has been interesting to see how we affect the compiler by the choices we make during the different compiler phases. So it has been nice to see how we actually go from a program in text all the way to the machine code, which can be executed.