CMSC 22610 Winter 2007	Implementation of Computer Languages	Project 2 January 18, 2007

MinML parser

Due: February 2, 2007

1 Introduction

Your second assignment is to implement a parser for MinML. You may either use ML-Yacc or ML-Antlr to generate your parser from an specification. ML-Yacc is described in Chapter 3 of Appel's book and there is a link to the manual on the course web page. ML-Antler is a new tool that supports LL(k) parsing. Its manual is also available from the course web page. The actions of this parser will construct a *parse tree* representation for an MinML program. We will provide an ML-Lex based scanner and the definition of the parse-tree representation, or you may use your own scanner.

2 The MinML grammar

The concrete syntax of MinML is specified by the grammar given in Figures 1 and 2.

As written, this grammar is ambiguous. To make this grammar unambiguous, the precedence of operators must be specified. The precedence of the binary operators are (from weakest to strongest):

```
orelse
andalso
= <= <
@ ::
+ -
* div mod
```

All binary operators are left associative except "`@`" and "`::`," which are right associative. The next highest precedence is function application, which associates to the left. Here are some examples:

```
a + b * c + d      ≡     (a + (b * c)) + d
a + 1 :: b :: []   ≡  (a + 1) :: (b :: [])
hd l x y           ≡           ((hd l) x) y
```

3 Requirements

Your implementation should consist of the following five files:

Prog
::= (*TopDecl* **;**)* *Exp*

TopDecl
::= **type** *TypeParams*^{opt} tyid **=** *Type*
| **datatype** *TypeParams*^{opt} tyid **=** *ConsDecl* (**|** *ConsDecl*)*
| *ValueDecl*

TypeParams
::= *tyvar*
| **(** *tyvar* (**,** *tyvar*)* **)**

Type
::= *TupleType* **->** *Type*
| *TupleType*

TupleType
::= *AtomicType* (***** *AtomicType*)*

AtomicType
::= tyid
| tyvar
| *AtomicType tyid*
| **(** *Type* (**,** *Type*)* **)** tyid
| **(** *Type* **)**

ConsDecl
::= conid (**of** *Type*)^{opt}

ValueDecl
::= **val** *TuplePat* **=** *Exp*
| **fun** *FunDef* (**and** *FunDef*)*

FunDef
::= vid *TuplePat* **=** *Exp*

Figure 1: The concrete syntax of MinML (A)

Exp
::= **let** *ValueDecl*$^+$ **in** *Exp* (**;** *Exp*)* **end**
| **if** *Exp* **then** *Exp* **else** *Exp*
| **case** *Exp* **of** *Match* (**|** *Match*)*
| *Exp* **andalso** *Exp*
| *Exp* **orelse** *Exp*
| *Exp* **=** *Exp*
| *Exp* **<=** *Exp*
| *Exp* **<** *Exp*
| *Exp* **::** *Exp*
| *Exp* **@** *Exp*
| *Exp* **+** *Exp*
| *Exp* **-** *Exp*
| *Exp* ***** *Exp*
| *Exp* **div** *Exp*
| *Exp* **mod** *Exp*
| *Exp Exp*
| **~** *Exp*
| *Const*
| vid
| **(** *Exp* (**,** *Exp*)* **)**
| **(** *Exp* (**;** *Exp*)* **)**

Match
::= *Pat* **=>** *Exp*

Pat
::= *Const*
| conid *TuplePat*
| *TuplePat*

TuplePat
::= *AtomicPat*
| **(** *AtomicPat* (**,** *AtomicPat*)* **)**

AtomicPat
::= vid
| _

Const
::= num
| str
| conid

Figure 2: The concrete syntax of MinML (B)

`MinML.cm` — a CM sources file for compiling your project.

`main.sml` — An SML source file containing the definition a structure `Main`, that defines a function

```
val parseFile : string -> ParseTree.program
```

where `ParseTree.program` is the type of program parse trees. This function should open the named source file, parse it, and return the resulting tree.

`parse-tree.sml` — An SML file containing a module `ParseTree` that defines the parse-tree representation of MinML programs. We will provide this file.

either `MinML.y` or `MinML.grm` — An parser specification file for parsing MinML programs. If you use ml-yacc, then your file should be called `MinML.y` and if you use ml-antlr, then it should be called `MinML.grm`. The actions of this parser should construct parse tree nodes.

either `MinML.l` or `MinML.lex` — An lexer specification file for lexing MinML. If you use ml-yacc for your parser, then you should use the ml-lex specification (`MinML.l`), while if you use ml-antlr, then you should use the ml-ulex specification (`MinML.lex`). We will provide these files, but you may modifiy it to match the terminals in your parser. You may also choose to use a modified version of the lexer you wrote for Part 1 of the project, but it will require some restructuring of the interface.

We will set up an SVN project for each student on the gforge server. This project will be seeded with the files mentioned above. You should use this repository to hold the source for your project. We will collect the projects at 10pm on Monday January 29th from the SVN repositories, so make sure that you have committed your final version before then.

4 Document history

January 23, 2007 Changed due date to February 2.

January 18, 2007 Original version.