CMSC 22610 Winter 2005	Implementation of Computer Languages	Project 2 January 20, 2005

MLR parser

Due: January 31, 2005

1 Introduction

Your second assignment is to implement a parser for MLR. You will use ML-Yacc to generate a parser from an LALR(1) specification (see Chapter 3 of Appel's book). The actions of this parser will construct a *parse tree* representation for an MLR program. We will provide an ML-Lex based scanner and the definition of the parse-tree representation.

2 The MLR grammar

The concrete syntax of MLR is specified by the grammar given in Figures 1 and 2. We have extended the grammar to include mutable fields in records.

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):

```
:=
with
orelse
andalso
== <= <
::
+ -
* / %
```

All binary operators except "`::`" (list construction) are left associative. The next highest precedence are the unary operators (`-`, **`not`**, *etc.*) and function application. Function application associates to the left The highest precedence is field selection. Here are some examples:

```
a + b * c + d          ≡     (a + (b * c)) + d
a + 1 :: b :: []       ≡  (a + 1) :: (b :: [])
f x . m + 1            ≡       (f (x . m)) + 1
f x with { x = 2 }     ≡  (f x) with { x = 2 }
hd l x y               ≡           ((hd l) x) y
```

Prog
::= *TopDecl*$^{+}$

TopDecl
::= **type** tid **=** *Type*
| *Function*

Type
::= *AtomicType* **->** *Type*
| *AtomicType*

AtomicType
::= **bool**
| **int**
| **string**
| tid
| *AtomicType* **list**
| **{** *RowType*^{opt} **}**
| **(** *Type* **)**

RowType
::= *FieldType* (**,** *FieldType*)*

FieldType
::= lid **:** *Type*
| lid **!** *Type*

Function
::= **fun** vid *Param*$^{+}$ **:** *Type* **=** *Exp*

Param
::= **(** vid **:** *Type* **)**

Figure 1: The concrete syntax of MLR (A)

Exp
::= **let** vid **:** *Type* = *Exp* **in** *Exp*
| *Function* **in** *Exp*
| **if** *Exp* **then** *Exp* **else** *Exp*
| *Exp* **with** **{** *RowExp* **}**
| *Exp* **orelse** *Exp*
| *Exp* **andalso** *Exp*
| *Exp* **==** *Exp*
| *Exp* **<=** *Exp*
| *Exp* **<** *Exp*
| *Exp* **::** *Exp*
| *Exp* **+** *Exp*
| *Exp* **-** *Exp*
| *Exp* ***** *Exp*
| *Exp* **/** *Exp*
| *Exp* **%** *Exp*
| **-** *Exp*
| **not** *Exp*
| **isnull** *Exp*
| **hd** *Exp*
| **tl** *Exp*
| *Exp Exp*
| *Exp* **.** lid
| *Exp* **!** lid
| *Exp* **!** lid **:=** *Exp*
| **true**
| **false**
| num
| str
| vid
| **(** *Exp* (**;** *Exp*)* **)**
| **[** (*Exp* (**,** *Exp*)*)^{opt} **]**
| **{** *RowExp*^{opt} **}**

RowExp
::= *FieldExp* (*FieldExp*)*

FieldExp
::= lid **=** *Exp*
| lid **!=** *Exp*

Figure 2: The concrete syntax of MLR (B)

3 Requirements

Your implementation should consist of the following five files:

`mlr.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 MLR programs. We will provide this file.

`mlr.y` — An ML-Yacc specification file for parsing MLR programs. The actions of this parser should construct parse tree nodes.

`mlr.l` — An ML-Lex specification file for lexing MLR. We will provide this file, 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 a CVS repository for each student on the Computer Science server. This repository will be seeded with CVS modules for each of the projects. For this project, the module is named `project-2` and contains the files mentioned above. You should use this repository to hold the source for your project. We will collect the projects at 12pm on Monday January 31st from the repositories, so make sure that you have committed your final version before then.

4 Document history

Jan. 24 Updated discussion of precedence and associativity and updated submission guidelines. Also, changed the name of `MLRParseTree` to `ParseTree`.

Jan. 20 Original version.