CMSC 22610 **Implementation of Computer Languages I** **Handout 2**
Winter 2011 **March 28, 2011**

MinML Syntax

1 The MinML grammar

The concrete syntax of MinML is specified by the grammar given in Figures **??** and **??**.

There are four classes of identifiers: *vid* for value identifiers (variables), *cid* for data constructor identifiers, *tyv* for type variables, and *tyc* for type constructors (including type constants). The identifier classes *vid* and *tyv* consist of alphanumeric identifiers starting with a lower-case letter, while *cid* and *tyc* range over alphanumeric identifier starting with an upper-case letter.

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):

```
        ||
        &&
== <> < <= > >=
        @
       + -
      * / %
```

All binary operators are left associative except "@" (string concatenation) which is 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
f a @ b @ "    ≡   (f a) @ (b @ "")
hd l x y       ≡       ((hd l) x) y
```

The lexical structure of MinML is defined in the ML-Lex specification file `minml.lex`, and the Yacc-style grammar is defined in `minml.grm`.

Prog
::= $($*TopDecl* **`;`**$)^{*}$ *Exp*

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

TypeParams
::= *tyv*
| **`(`** *tyv* $($**`,`** *tyv*$)^{*}$ **`)`**

Type
::= *Type* **`->`** *Type*
| *AtomicType* $($**`*`** *AtomicType*$)^{*}$

AtomicType
::= tyv
| tyc $($**`(`** *Type* $($**`,`** *Type*$)^{*}$ **`)`**$)^{opt}$
| **`(`** *Type* **`)`**

ConsDecl
::= cid $($**`(`** *Type* **`)`**$)^{opt}$

ValueDecl
::= **`val`** *TuplePat* **`=`** *Exp*
| **`fun`** *FunDef* $($**`and`** *FunDef*$)^{*}$

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

Figure 1: The concrete syntax of MinML (A)

Const
::= num
| str
| cid

Pat
::= *Const*
| cid *TuplePat*
| *TuplePat*

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

AtomicPat
::= vid
| _

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

Exp
::= vid
| *Const*
| *Exp* `||` *Exp*
| *Exp* `&&` *Exp*
| *Exp* `==` *Exp*
| *Exp* `<>` *Exp*
| *Exp* `<` *Exp*
| *Exp* `<=` *Exp*
| *Exp* `>` *Exp*
| *Exp* `>=` *Exp*
| *Exp* `@` *Exp*
| *Exp* `+` *Exp*
| *Exp* `-` *Exp*
| *Exp* `*` *Exp*
| *Exp* `/` *Exp*
| *Exp* `%` *Exp*
| `~` *Exp*
| *Exp* *Exp*
| `(` *Exp* (`,` *Exp*)* `)`
| `(` *Exp* (`;` *Exp*)* `)`
| `if` *Exp* `then` *Exp* `else` *Exp*
| `let` *ValueDecl*$^+$ `in` *Exp* (`;` *Exp*)* `end`
| `case` *Exp* `of` *Match* (`|` *Match*)* `end`
| `fn` vid `=>` *Exp*

Figure 2: The concrete syntax of MinML (B)