CS383 Tutorial 1
Course Project Introduction
Yu Gong
Before we start ...

How many of you have used ...

• Java
• Eclipse
• A functional language, e.g. Lisp / ML / Haskell
• Yacc / flex / bison / Jflex / Java CUP
Introduction

• SimPL (pronounced simple)
  – simplified dialect of ML
  – both functional and imperative

• Write an interpreter in Java
Phases

1. Lexical Definition
2. Syntactic analysis
3. Type inference
4. Semantics
5. Implementation
Lexical Definition

• The lexical definition of SimPL consists of four aspects:
  1. Comments
  2. Atoms
  3. Keywords
  4. Operators
Lexical Definition

1. Comments

- Comments in SimPL are enclosed by pairs of (*) and *).
- Comments are nestable, e.g. (* (* *) *) is a valid comment.
- A comment can spread over multiple lines.
- Comments and whitespaces (spaces, tabs, newlines) should be ignored and not evaluated.
Lexical Definition

1. Atoms
   - Atoms are either integer literals or identifiers.
   - Integer literals are matched by regular expression \([0-9]+\).
   - Integer literals only represent non-negative integers less than \(2^{31}\).
   - Integer literals are in decimal format, and leading zeros are insignificant e.g. 0123 represent the integer 123.
   - Identifiers are matched by regular expression \([_a-z][_a-zA-Z0-9']\)*.
Lexical Definition

1. Keywords
   • All the following identifiers are keywords.
     – ref
     – fn rec
     – let in end
     – if then else
     – while do
     – true false
     – not and also or else
Lexical Definition

1. Operations

• All the following identifiers are operations.
  • +      -      *      /      %      ~
  • =      <>     <      <=     >      >=
  • ::     ()     =>
  • :=     !
  • ,      ;      ()
Syntactic analysis

• All SimPL programs are expressions. $\text{Exp}$ is the set of all expressions.
• Names are non-keyword identifiers. $\text{Var}$ is the set of all names.
• Expressions, names, and integer literals are denoted by meta variables $e$, $x$, and $n$.
• Unary operator $uop \in \{\sim, \text{not}, !\}$
• Binary operator $bop \in \{+, -, *, /, \%, =, <>, <, <=, >, >=, \text{andalso}, \text{orelse}, ::, :=, ;\}$
# Syntactic analysis

## Operator Precedence

<table>
<thead>
<tr>
<th>Priority</th>
<th>Operator(s)</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>;</td>
<td>Left</td>
</tr>
<tr>
<td>2</td>
<td>:=</td>
<td>None</td>
</tr>
<tr>
<td>3</td>
<td>orelse</td>
<td>Right</td>
</tr>
<tr>
<td>4</td>
<td>andalso</td>
<td>Right</td>
</tr>
<tr>
<td>5</td>
<td>= &lt;&gt; &lt; &lt;= &gt; &gt;=</td>
<td>None</td>
</tr>
<tr>
<td>6</td>
<td>::</td>
<td>Right</td>
</tr>
<tr>
<td>7</td>
<td>+ -</td>
<td>Left</td>
</tr>
<tr>
<td>8</td>
<td>* / %</td>
<td>Left</td>
</tr>
<tr>
<td>9</td>
<td>(application)</td>
<td>Left</td>
</tr>
<tr>
<td>10</td>
<td>~ not !</td>
<td>Right</td>
</tr>
</tbody>
</table>
Typing and Semantic

• You can refer to the project specification.
• And get the details.
Implementation

• Code in Java 7 and Eclipse
• Submit a runnable JAR file, e.g. SimPL.jar
  
  java -jar SimPL.jar program.spl

• Your interpreter is started by using
  
  – java -jar SimPL.jar program.spl

• Output the result of the execution to the standard output (System.out)

• Case time limit: 5000ms
How to run

```java
void run(String filename) {
    try (InputStream inp = new FileInputStream(filename)) {
        Parser parser = new Parser(inp);
        java_cup.runtime.Symbol parseTree = parser.parse()
        Expr program = (Expr) parseTree.value;
        program.typecheck(new DefaultTypeEnv());
        System.out.println(program.eval(new InitialState()));
    }
    catch (SyntaxError e) {
        System.out.println("syntax error");
    }
    catch (TypeError e) {
        System.out.println("type error");
    }
    catch (RuntimeError e) {
        System.out.println("runtime error");
    }
}
```
Examples

```plaintext
let
    addFive = fn x => x + 5
in
    addFive 2
end

Functions are first-class values
Examples

let

    add = fn x => fn y => x + y

in

    add 1 2

end
Examples

let
factorial = rec f =>
  fn x => if x=1
    then 1
    else x * (f (x - 1))
in
  factorial 4
end

(* this is a comment *)
Examples

let

p = ref 100
(* p = new int(100); *)

in

p := 200;
(* *p = 200; *)

!p
(* return *p; *)

end
Conclusion

• Fully understand the specification before you write the code

• Resources are available online at
  http://www.cs.sjtu.edu.cn/~kzhu/cs383
  http://adapt.seiee.sjtu.edu.cn/~gongyu/cs383

• Send feedback to gy910210@163.com