CSC 372, Spring 2025

Recursive Descent Parsing and Polymorphic Types

Michelle Strout

THE UNIVERSITY OF ARIZONA

February 6, 2025

Plan

- **Announcements**
  - SA3 MT1 review is posted and due Wednesday Feb 12th
  - LA1 parser/shapes2svg is posted and due Friday Feb 14^{th}
- **Last time**
  - TopHat Questions and ICA4: Quiz on SML concepts from SA2
  - Tokenization/Lexing as a part of syntactic analysis
  - Recursive descent parsing
- **Today**
  - TopHat Questions from ICA4: Quiz on SML concepts from SA2
  - ICA5: participation quiz that is a midterm warmup
  - Recursive descent parsing
  - Polymorphism in SML

TopHat Questions

- **SML vs. Lisp**
  – What is Lisp lacking such that it has to check if x is null to see if the list is empty versus the SML example shown?

```
fun length [] = 0
  | length (x::xs) = 1 + length xs;
```

```
(define (length x)
  (if (null? x)
      0
      (+ 1 (length (cdr x)))))
```

TopHat Questions cont …

- Sum of squares of even numbers
  - Recursive version
  - foldl version
- Ways to practice and learn more about SML
  - https://exercism.org/tracks/sml
  - https://github.com/i4ki/awesome-sml

ICA5: Participation Quiz, MT1 warmup

- Read the instructions on the quiz

Outline for rest of today

- Recursive descent parsing for PA1
- SML hints: debugging with print, patterns in val bindings
- Abstract Syntax Trees
- Polymorphic Types in SML
- More SML info

Predictive Parsing

- Predictive parsing, such as recursive descent parsing, creates the parse tree TOP DOWN, starting at the start symbol, and doing a LEFT-MOST derivation.
- For each non-terminal N there is a function recognizing the strings that can be produced by N, with one (case) clause for each production.

- Consider:

```
start      -> stmts EOF
stmts      -> ε | stmt stmts
stmt       -> ifStmt | whileStmt | ID = NUM
ifStmt     -> IF id { stmts }
whileStmt  -> WHILE id { stmts }
```

- Draw parse tree on white board for this

```
WHILE x { IF y { z = 42 } }
```

Predictive Parser: Recursive Descent

THE UNIVERSITY OF ARIZONA
Computer Science

```
start      -> stmts EOF
stmts      -> ε | stmt stmts
stmt       -> ifStmt | whileStmt
ifStmt     -> IF id { stmts }
whileStmt  -> WHILE id { stmts }
```

```
void start() { switch(m_lookahead) {
   case IF, WHILE, EOF: stmts(); match(Token.Tag.EOF); break;
   default:   throw new ParseException(…);
}}
void stmts() { switch(m_lookahead) {
   case IF,WHILE:  stmt(); stmts(); break;
   case EOF:      break;
   default:       throw new ParseException(…);
}}
void stmt() { switch(m_lookahead) {
   case IF:   ifStmt();break;
   case WHILE: whileStmt(); break;
   default:   throw new ParseException(…);
}}
void ifStmt() {switch(m_lookahead) {
   case IF: match(id); match(OPENBRACE);
          stmts(); match(CLOSEBRACE); break;
   default: throw new ParseException(…);
}}
```

Recursive Descent Parsing

- **Each non-terminal becomes a function**
  - that mimics the RHSs of the productions associated with it
  - and choses a particular RHS:
    - an alternative based on a look-ahead symbol
  - and throws an exception if no alternative applies
- **When does this NOT work?**
  - <E> ::= <E> + <E> | <E> * <E>

Determine Look Ahead per grammar rule

Grammar Rule	Lookahead Token
start -> stmts EOF	// no lookahead
stmts -> ε	EOF, }
stmts -> stmt stmts	IF, WHILE, ID
stmt -> ifStmt	IF
stmt -> whileStmt	WHILE
stmt -> ID = NUM	ID
ifStmt -> IF id { stmts }	IF
whileStmt -> WHILE id { stmts }	WHILE

Look at LA1 starter code

- Observations

Outline for rest of today

THE UNIVERSITY OF ARIZONA
Computer Science

- Recursive descent parsing for PA1
- SML hints: debugging with print, patterns in val bindings
- Abstract Syntax Trees
- Polymorphic Types in SML
- More SML info

SML hints: debugging with print

THE UNIVERSITY OF ARIZONA
Computer Science

- Sequence expressions
  - Value of whole expression is value of last item in a semicolon separated sequence
  - All expressions but the last need to be of type unit
- Combined with print, this can be used for debugging

```
let val x=3
in (print "x="; print (Int.toString x); print "\n"; x)
end;
```

- What is the type of 'print' in SML?

SML hints: patterns in val bindings

- **Patterns can be used in ...**
  - Function clauses
  - Case expressions
  - And Val bindings

```
val (x,y) = partition (fn x => true) [5,4,3]
```

Outline for rest of today

- Recursive descent parsing for PA1
- SML hints: debugging with print, patterns in val bindings
- Abstract Syntax Trees
- Polymorphic Types in SML
- More SML info

Abstract Syntax Trees

- **Definition**
  - A tree representation of the structure of source code, abstracting away syntactic details like parentheses and punctuation.
  - Each node represents a construct (e.g., expressions, statements, functions).
  - Inner nodes represent operators or control structures, while leaves represent literals or variables.
- **Purpose**
  - Used in **compilers**, **interpreters**, and **static analysis** to represent and manipulate and process code.
  - More compact than concrete syntax trees (parsing trees).

“Code” Generation Given an AST

- SML data type for the AST for shapes language

```
(* Abstract Syntax Tree (AST) datatype *)
datatype ast =
    Program of ast list
  | StmtCircle of int * int * int * string
  | StmtLine of int * int * int * int * string
  | StmtRectangle of int * int * int * int * string
```

- Function that generates code based on that AST

```
(* codegen function *)
fun svgGen (Program stmts) =
      "<svg xmlns=\"http://www.w3.org/2000/svg\">\n"
      ^ String.concatWith "\n" (List.map svgGen stmts)
      ^ "\n</svg>\n"
  | svgGen _ = "Implement me"
```

Outline for rest of today

- Recursive descent parsing for PA1
- SML hints: debugging with print, patterns in val bindings
- Abstract Syntax Trees
- Polymorphic Types in SML
- More SML info

Recall the Tree Example

```
datatype IntTree = Leaf | Node of int * IntTree * IntTree
```

```
val tempty = Leaf
val t1 = Node (1, tempty, tempty)
val t2 = Node (2, t1, t1)
val t3 = Node (3, t2, t2)
```

Deconstruct values with pattern matching

THE UNIVERSITY OF ARIZONA. Computer Science

```
fun inOrder Leaf = []
  | inOrder (Node (v, left, right)) =
        (inOrder left) @ [v] @ (inOrder right)
val il3 = inOrder t3

fun preOrder Leaf = []
  | preOrder (Node (v, left, right)) =
         v :: (preOrder left) @ (preOrder right)
val pl3 = preOrder t3
```

- Notes
  - IntTree is monomorphic because it has a single type
  - Note though that the inOrder and preOrder functions only care about the structure of the tree, not the payload value
- Questions
  - What if we want to store something other than an int in a tree?

Polymorphic datatypes!

```
datatype 'a tree = Child
                 | Parent of 'a * 'a tree * 'a tree
```

- Notes
  - Polymorphic datatypes are written using type variables that can be instantiated with any type
  - `tree` is a type constructor (written in post-fix notation), which means it produces a type when applied to a type argument
  - Examples:
    - `int tree` is a tree of integers
    - `bool tree` is a tree of booleans
    - `int list tree` is a tree of a list of integers
  - `'a` is a type variable: it can represent any type
- Questions
  - What are the data constructors?
  - Create an example tree with at least two parents.

Different kinds of Polymorphism

- **Ad-hoc Polymorphism (Overloading & Coercion)**
  - Function/operator overloading (e.g., + for ints and reals).
  - Implicit type conversions (e.g., int to float in C/C++).

- **Parametric Polymorphism**
  - Functions and data structures operate on any type.
  - Example: fun identity x = x in SML (‘a -> ‘a)
  - Implemented as generics in languages like Java and Rust.

- **Subtype Polymorphism (Inheritance and Subtyping)**
  - Objects of a class can be used where a superclass is expected.
  - Enables method overriding and dynamic dispatch.
  - Example: Animal superclass, Dog subclass.

Pattern Matching with Polymorphism

```
datatype 'a tree = Child
                 | Parent of 'a * 'a tree * 'a tree

fun inOrder Child = []
  | inOrder
```

- Questions
  - Finish the rest of the above?
  - How is polymorphism different than overloading?

Outline for rest of today

- Recursive descent parsing for PA1
- SML hints: debugging with print, patterns in val bindings
- Abstract Syntax Trees
- Polymorphic Types in SML
- More SML info

Tuple Pattern Matching

- **Question**
  - Indicate what the variables in the pattern will be bound to.

```
val (x,y) = (1,2) (* answer: x=1, y=2 *)

val (n,xs) = (3,[1,2,3]) (* answer: n=3, xs=[1,2,3] *)

val (x::xs) = [1,2,3]  (* answer: x=1, xs=[2,3] *)

val (_::xs) = [1,2,3]  (* answer: xs=[2,3] *)

val (_::xs) = [3]  (* answer: xs=[] *)
```

Case Expressions also use pattern matching

- `case` expression

```
fun length xs =
  case xs
    of []        => 0
     | (x::xs) => 1 + length xs
```

- At top level, `fun` is better than `case`

```
fun length []      = 0
  | length (x:xs) = 1 + length xs
```

Case works for any datatype

- At top level, **`fun`** is better than **`case`**

```
fun toStr t =
    case t
      of Leaf => "Leaf”
       | Node(v,left,right) => "Node"
```

- Question: how do we rewrite case using fun?

```
fun toStr ??
```

Exception Handling

- Syntax
  - `Declaration: exception exn`
  - `Introduction: raise where e: exn`
  - `Elimination: e1 handle pat => e2`
- Informal Semantics
  - Alternative to normal termination
  - Can happen in any expression
  - Tied to function call: if evaluation of body raises exn, call raises exn
  - Handler uses pattern matching

```
e handle pat1 => e1 | pat2 => e2
```

ML traps and pitfalls

- Order of clauses matters

```
fun take n (x::xs) = x :: take (n-1) xs
  | take 0 xs = []
  | take n [] = []
(* what goes wrong? *)
```

- Gotcha – overloading

```
- fun plus x y = x + y;
> val plus = fn : int -> int -> int
- fun plus x y = x + y : real;
> val plus = fn : real -> real -> real
```

- Gotcha – equality types, `''a` is equality type var

```
- (fn (x,y) => x=y);
> val ''a it = fn : ''a * ''a -> bool
```