CSC 372, Spring 2025

Lexing and SML Datatypes

Michelle Strout

THE UNIVERSITY
OF ARIZONA

January 30, 2025

Plan

- **Announcements**
  - LA1 will be posted sometime tomorrow and due Friday Feb 14th
- **Last time**
  - Review previous quiz questions
  - SML intro continued and Types intro
  - Syntax and Semantics
- **Today**
  - Grammar clarification
  - Show Anki deck of PL concepts
  - Tokenization/Lexing as a part of syntactic analysis
  - Algebraic datatypes in SML

Grammar and Syntax Rules

- **What is a Grammar?**

- **A grammar defines a language by specifying:**
  - Tokens - The smallest units of a language (e.g., keywords, literals).
  - Rules - How tokens combine into valid statements.

- **Backus-Naur Form (BNF):**
  - Developed by John Backus and Peter Naur (~1960).
  - Defines structure using:
    - Terminals: Basic symbols (tokens).
    - Non-terminals: Higher-level constructs made from terminals.

Example formal grammar in BNF

- **Syntax rules in BNF for SML expressions**

```
<expr> ::= <value> | <expr> <op> <expr>
<value> ::= int | bool
<op> ::= + | * | and | or
```

- **Semantic rules**
  - Integers can be added or multiplied.
  - Booleans can be combined using logical operators.

- **Questions**
  - In the above grammar, how are the terminals denoted?
  - Non-terminals?
  - What does the ::= mean?
  - What does the | mean?

Outline for today

- Tokens exercise
- SML algebraic datatypes
- Regular expressions to specify tokens
- Complications with tokenization/lexing
- Simple tokenization for PA1

Learning by doing in LA1 (show face.svg)

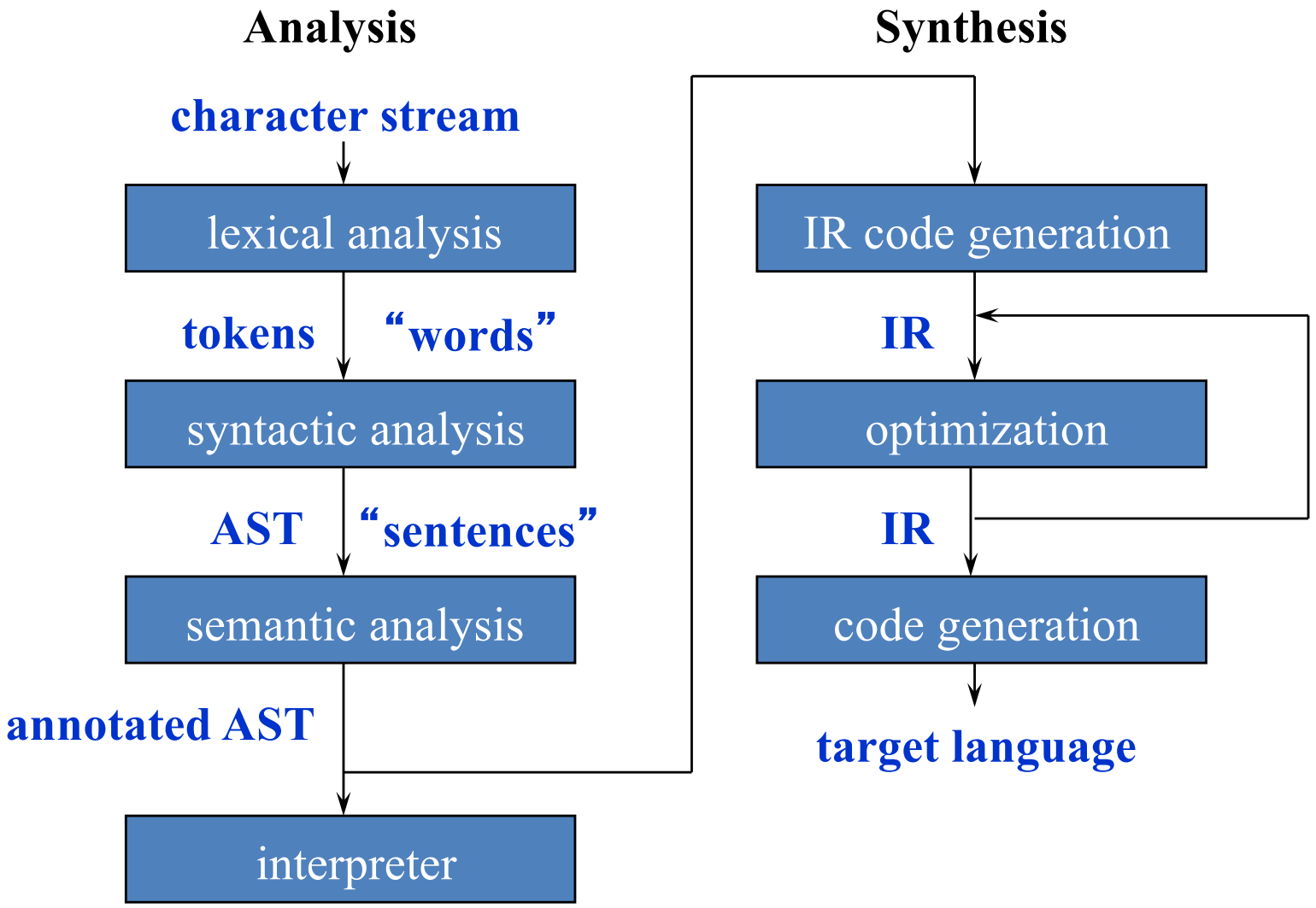
- **Large assignment 1 is a compiler from shapes to svg**

```
CIRCLE 120 150 60 white
```

```
<circle cx="120" cy="150" r="60" fill="white" />
```

- **Going to specify the input with a context free grammar and tokens with regular expressions**
- **Going to implement a lexer/tokenization, parser, AST (abstract syntax tree), and "codegen" in SVG**

Structure of a Typical Compiler



Tokens and tokenization

- We can define a token as the smallest unit in a language that has meaning.
- The process of taking a string of input and dividing it up into tokens is called tokenization.
- It's not just about determining what the tokens are—we also want to categorize them into kinds of tokens so that we can then use them to determine legal structures.

Identifying Tokens Exercises

- **At your tables, list out the different tokens in the below "shapes" file using the provided SML datatype**

```
CIRCLE 120 150 60 white
LINE 0 0 300 300 black
RECTANGLE 30 20 300 250 blue
```

```
(* Token datatype *)
datatype token =
    TokenCIRCLE
  | TokenLINE
  | TokenRECTANGLE
  | TokenNUM of int
  | TokenCOLOR of string
```

Foundation: Data/Values in SML

- **Base types: int, real, bool, char, string**
- **Functions**
- **Constructed data**
  - **Tuples: pairs, triples, etc.**
  - **(Records with named fields)**
  - **Lists and other algebraic data types**

Slide Content Credits for slides 10-15: Tufts Comp105
by Norman Ramsey and Kathleen Fisher

Algebraic Datatypes in SML

- **Enumerated types**
  - Datatypes can define an enumerated type and associated values

```
datatype suit = heart | diamond | spade | club
```

- **"suit" is the name of a new type**
- **Data constructors heart, diamond, space, and club are values of that type**
- **Data constructors are separated by vertical bars**

Algebraic Datatypes

- **Pattern matching**
  - Datatypes are deconstructed using pattern matching

```
fun toString heart = "heart"
  | toString diamond = "diamond"
  | toString spade = "spade"
  | toString club = "club"

val suitName = toString heart
```

Write a toString function for token datatype

- **At your tables, write a string function on the whiteboards**

```
datatype token =
    TokenCIRCLE
  | TokenLINE
  | TokenRECTANGLE
  | TokenNUM of int
  | TokenCOLOR of string
```

Data constructors can take arguments!


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

- What is the name of the new type?
- What data constructors are in the above?
- What is the parameter to the Node data constructor?
- What are some example values made with the data constructors?
- What are the type(s) of those values?

Tree Example


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

- ➔ **Tree diagram**

- **Questions**
  - What is the in-order traversal of t3?
  - What is the pre-order traversal of t3?

Deconstruct values with pattern matching


```
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 does @ do?
  - How would we implement postOrder? (postOrder left) @ (postOrder right) @ [v]

Outline for today

- Tokens exercise
- SML algebraic datatypes
- **Regular expressions to specify tokens**
- **Complications with tokenization/lexing**
- **Simple tokenization for PA1**

Languages

A language is a set of strings
(sometimes called sentences)

String: A finite sequence of letters

Examples: "cat", "dog", "house", …

Defined over a fixed alphabet:

$$\Sigma = \{a, b, c, \ldots, z\}$$

Empty String

A string with no letters: ε (sometimes λ is used)

Observations: $|\varepsilon| = 0$

$$\varepsilon w = w\varepsilon = w$$

$$\varepsilon abba = abba\varepsilon = abba$$

Regular Expressions

Regular expressions describe regular languages
You have probably seen them in OSs / editors

Example: $(a \mid (b)(c))^*$

describes the language

$$L((a \mid (b)(c))^*) = \{\varepsilon, a, bc, aa, abc, bca, \ldots\}$$

Recursive Definition for Specifying Regular Expressions

Primitive regular expressions: $\varnothing,\ \varepsilon,\ \alpha$

where $\alpha \in \Sigma$, *some alphabet*

Given regular expressions r_1 *and* r_2

$r_1 \mid r_2$

$r_1\ r_2$

$r_1{}^*$

(r_1)

} *Are regular expressions*

Regular operators

choice:	**A | B**	**a string from L(A) or from L(B)**
concatenation:	**A B**	**a string from L(A) followed by a string from L(B)**
repetition:	**A***	**0 or more concatenations of strings from L(A)**
	A^+	**1 or more**
grouping:	**(A)**	

Concatenation has precedence over choice: A|B C vs. (A|B)C

More syntactic sugar, used in scanner generators:

[abc] means a or b or c

[\t\n] means tab, newline, or space

[a-z] means a,b,c, …, or z

Example Regular Expressions and Regular Definitions

Regular definition:

name : regular expression

name can then be used in other regular expressions

Keywords "print", "while"

Operations: "+", "-", "*"

Identifiers:

let : [a-zA-Z] // chose from a to z or A to Z

dig : [0-9]

id : let (let | dig)*

Numbers: dig^+ = dig dig*

What are the regular expressions for “shapes” tokens?

```
CIRCLE 120 150 60 white
LINE 0 0 300 300 black
RECTANGLE 30 20 300 250 blue
```

```
datatype token =
    TokenCIRCLE
  | TokenLINE
  | TokenRECTANGLE
  | TokenNUM of int
  | TokenCOLOR of string
```

```
TokenCIRCLE:
TokenLINE:
TokenRECTANGLE:
TokenNUM:
TokenCOLOR:
```

Outline for today

- Tokens exercise
- SML algebraic datatypes
- Regular expressions to specify tokens
- **Complications with tokenization/lexing**
- **Simple tokenization for PA1**

Complications

1. "1234" is an NUMBER but what about the "123" in "1234" or the "23", etc. Also, the scanner must recognize many tokens, not one, only stopping at end of file.

2. "if" is a keyword or reserved word IF, but "if" is also defined by the reg. exp. for identifier ID. We want to recognize IF.

3. We want to discard white space and comments, most of the time.

4. "123" is a NUMBER but so is "235" and so is "0", just as "a" is an ID and so is "bcd", we want to recognize a token, but add attributes to it.

Complications 1

1. "1234" is an NUMBER but what about the "123" in "1234" or the "23", etc. Also, the scanner must recognize many tokens, not one, only stopping at end of file. So:

recognize the largest string defined by some regular expression, only stop getting more input if there is no more match. This introduces the need to reconsider a character, as it is the first of the next token

e.g. *fname(a,bcd);*

would be scanned as

ID OPEN ID COMMA ID CLOSE SEMI EOF

scanning *fname* would consume (, which would be put back and then recognized as OPEN

Complication 2

2. "if" is a keyword or reserved word IF, but "if" is also defined by the reg. exp. for identifier ID, we want to recognize IF, so

Have some way of determining which token (IF or ID) is recognized.

This can be done using priority, e.g. in scanner generators an earlier definition has a higher priority than a later one.

By putting the definition for IF before the definition for ID in the input for the scanner generator, we get the desired result.

What about the string "ifyouleavemenow"?

Complication 3

3. we want to discard white space and comments and not bother the parser with these. So:

in scanner generators, we can

specify, using a regular expression, white space e.g. [\t\n]

and return no token, i.e. move to the next

specify comments using a (NASTY) regular expression and again return no token, move to the next

Complication 4

4. "123" is a NUMBER but so is "235" and so is "0", just as "a" is an ID and so is "bcd", we want to recognize a token, but add attributes to it. So,

Scanners/Lexers return Symbols, not tokens. Different in SML?

A Symbol is a (token, tokenValue) pair,

e.g. (NUMBER,123) or (ID,"a").

Often more information is added to a symbol, e.g. line number and position.

Outline for today

- Tokens exercise
- SML algebraic datatypes
- Regular expressions to specify tokens
- Complications with tokenization/lexing
- **Simple tokenization for PA1**

Simple Tokenization for PA1

- **LA1: compiler from shapes to SVG**

```
CIRCLE 120 150 60 white
```

```
<circle cx="120" cy="150" r="60" fill="white" />
```

- **Just because something can be complicated doesn't mean it always is**
- **Ask an AI or search engine how to break up a string into substrings using whitespace as a delimeter**
- **What if we were going from SVG to shapes?**