More Parsing and Scanning

COMP360

“Simplicity is prerequisite for reliability.”

Edsger Dijkstra

Exam Schedule

- The first COMP360 exam scheduled for last Friday has been postponed until Wednesday, February 19, 2020
- You are allowed on 8½ by 11” page of notes
- The exam will cover everything since the start of class
  - Writing a scanner
  - Writing a parser
  - Creating an FSA and BNF
- A sample exam is on Blackboard

Snowflake Parser

- The Parsing programming assignment due date has been moved back to Friday, February 21, 2020
- You may do this assignment in teams of two students
- The BNF for Snowflake is provided
- You may use the provided scanner

Backus-Naur Form

- A BNF is a way of describing the syntax of a language
- BNF is a **metalanguage**, a language used to describe a language
- Terminal symbols are tokens or symbols in the language. They come from the lexical scanner
- Nonterminal symbols are “variables” that represent patterns of terminals and nonterminal symbols

BNF Format

- BNF rules or productions are shown as

  BNFvariable → more BNFvariables and language tokens

- A BNF describes the pattern of punctuation and words in a language
- Exact items from the language (such as punctuation or scanner tokens) are shown in **bold**
- Multiple options are shown separated by a vertical bar

  thing → variable

  | **[** variable **]**

Early BNF

- The idea of describing the structure of language using rewriting rules can be traced back to the work of Pāṇini sometime between the 6th and 4th century BCE
- In elementary school, your teacher may have described English sentences as

  noun part verb part

  noun part is a noun or an adjective noun

  verb part is a verb or a verb noun part

Example BNF for Nonsense Language

1. goat → **a** cat **b**
2. cat → dog
3. | dog cat
4. dog → **c**
5. | **a** cat **b**

- The letters **a, b** and **c** are terminals and appear in the language
- goat, cat and dog are BNF variables or non-terminals

Language Derivation

- You can show that a string of letters is in the language with a series of BNF rules that can create the string
- Showing that **a a c b b** is in the language

a cat **b**	rule 1
a dog **b**	rule 2
a a cat **b b**	rule 5
a a dog **b b**	rule 2
a a c b b	rule 4

1. goat → **a** cat **b**
2. cat → dog
3. | dog cat
4. dog → **c**
5. | **a** cat **b**

Which string is in the language?

1. goat → **a** cat **b**
2. cat → dog
3. | dog cat
4. dog → **c**
5. | **a** cat **b**

A. b a

B. a a c b

C. a c a b c b

D. a c a c b b

E. none of the above

Derivations and Parsing

- A derivation shows that you can build a string that is in the language
- Parsing shows that a given string is in the language
- You can think of
  - Derivation as top down
  - Parsing as bottom up

Reading the Token list

(	5.7	+	dog	)	/	6

- The lexical scanner creates a list of all tokens in the program
- The BNF describes the proper order for the words and punctuation and what is required
- The parser moves sequentially through the list checking if the list matches the BNF

Recursive Descent

- One of the easiest parsing algorithms is recursive descent
- A method is created for each of the BNF productions
- The methods check if the input list of tokens at that point matches the BNF rules
- The methods can return true if the input matches the BNF
- The examples assume the input from the scanner is an array called **`tokens`** indexed by **`curtok`**

Parser for Nonsense Language

- There will be three methods: goat(), cat() and dog()

```
boolean goat() {
        if (tokens[curtok] != 'a') return false;
        curtok++;           // move past 'a'
        if ( !cat() ) return false;
        if (tokens[curtok] != 'b') return false;
        curtok++;           // move past 'b'
        return true;
}
```

1. goat → **a** cat **b**
2. cat → dog
3. | dog cat
4. dog → **c**
5. | **a** cat **b**

dog() Recursive Descent method

```
boolean dog() {
        if (tokens[curtok] != 'c')  {
                curtok++;           // consume C
                return true;
        }
        if (tokens[curtok] != 'a') return false;
        curtok++;                   // consume A
        if ( !cat() ) return false;
        if (tokens[curtok] != 'b') return false;
        curtok++;                   // consume B
        return true;
}
```

1. goat → **a** cat **b**
2. cat → dog
3. | dog cat
4. dog → **c**
5. | **a** cat **b**

Write the cat() Recursive Descent method

1. goat → **a** cat **b**
2. cat → dog
3. | dog cat
4. dog → **c**
5. | **a** cat **b**

Possible cat() Recursive Descent method

```
boolean cat() {
        if ( !dog() ) return false;
        cat();          // may be true or false
        return true;
}
```

1. goat → **a** cat **b**
2. cat → dog
3. | dog cat
4. dog → **c**
5. | **a** cat **b**

Snowflake Parser

- For the Snowflake parsing assignment, you need to write ten boolean methods

Possible Snowflake BNF

1. SF → parmstmt code ret
2. parmstmt → **parm** varlist **;**
3. varlist → **var**
   | **var** varlist
4. varclist → **varconst**
   | **varconst** varclist
5. ret → **return var ;**
6. pattern → **varconst**
   | **varconst | pattern**

Possible Snowflake BNF (cont)

7. code → line
   | line code
8. line → assign
   | loop
9. assign → **var** **=** varclist **;**
   | **var** pattern **=** varclist **;**
10. loop → **while** **varconst** pattern **{** code **}**

Write the Snowflake code() method

7. code → line
| line code

Possible Snowflake code() method

7. code → line
| line code

```
boolean code() {
        if ( !line() ) return false;   // must start with a  line
        code();                 // may or may not be another code
        return true;
}
```

Beyond reading the assignment when did you start the Snowflake scanner program?

			Wed, Feb 5	Thur, Feb 6 A	Fri, Feb 7	Sat, Feb 8
Sun, Feb 9 B	Mon, Feb 10	Tues, Feb 11 C	Wed, Feb 12 D			

E. I did not do the assignment

The Snowflake scanner program was difficult because

A. I don’t know what a scanner does
B. I could not create an FSA graph
C. I don’t understand how to program an FSA
D. I ran out of time
E. I did not try the assignment

Beyond reading the assignment, have you started the Snowflake parser program?

A. Yes
B. No
C. I do not intend to do the assignment

Stages of a Compiler

- Source preprocessing
- Lexical Analysis (scanning)
- Syntactic Analysis (parsing)
- Semantic Analysis
- Optimization
- Code Generation
- Link to libraries

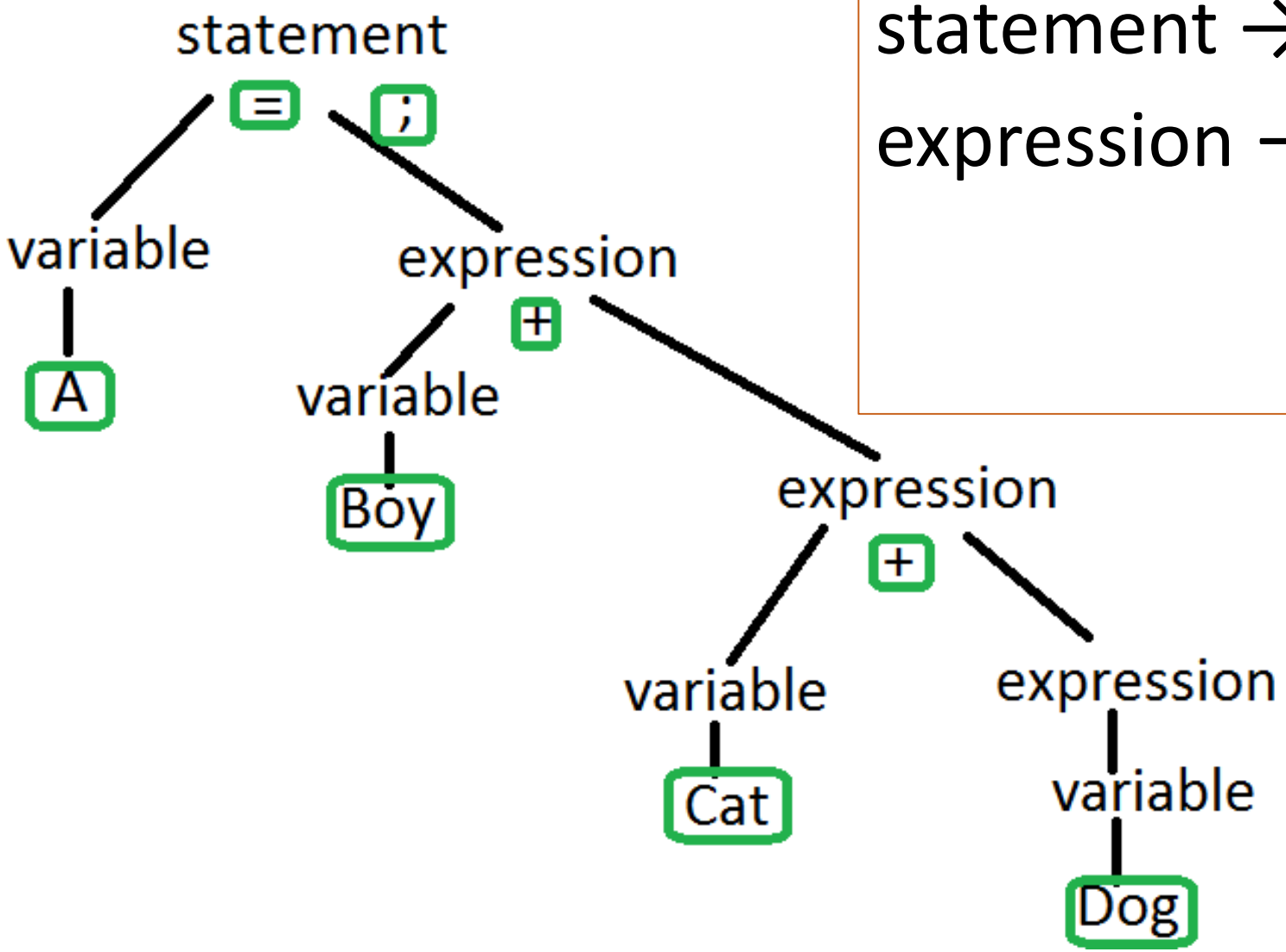
Semantic Analysis

- The Semantic Analysis inputs the parse tree from the parser
- Semantic Analysis checks that the operations are valid for the given operands *(e.g. cannot multiply a String)*
- This stage determines what the program is to do
- The output of the Semantic Analysis is an intermediate code. This is similar to assembler language, but may include higher level operations

Creating a Parse Tree

- The goal of the parser is to create a parse tree whose leaves are the tokens from the lexical scanner when traversed left to right
- In addition to proving that a tree does or does not exist, the parser should actually build the tree
- The semantic analyzer will use the tree to create executable code

Parsing A = Boy + Cat + Dog;



statement → **variable =** expression ;

expression → **variable +** expression

| **variable**

Semantic Analysis

- Intermediate code defines a series of simple steps that will execute the program

Operation	First Operand	Second Operand	Destination
add	Boy	Cat	temp1
add	temp1	Dog	temp2
copy	temp2		A

Simple Machine Language

- Load register with B
- Add C to register
- Store register in Temp1
- Load register with Temp1
- Add D to register
- Store register in Temp2
- Load register with Temp2
- Store register in A

Optimized Machine Language

- Load register with B
- Add C to register
- Store register in Temp1
- Load register with Temp1
- Add D to register
- Store register in Temp2
- Load register with Temp2
- Store register in A

Action Rules

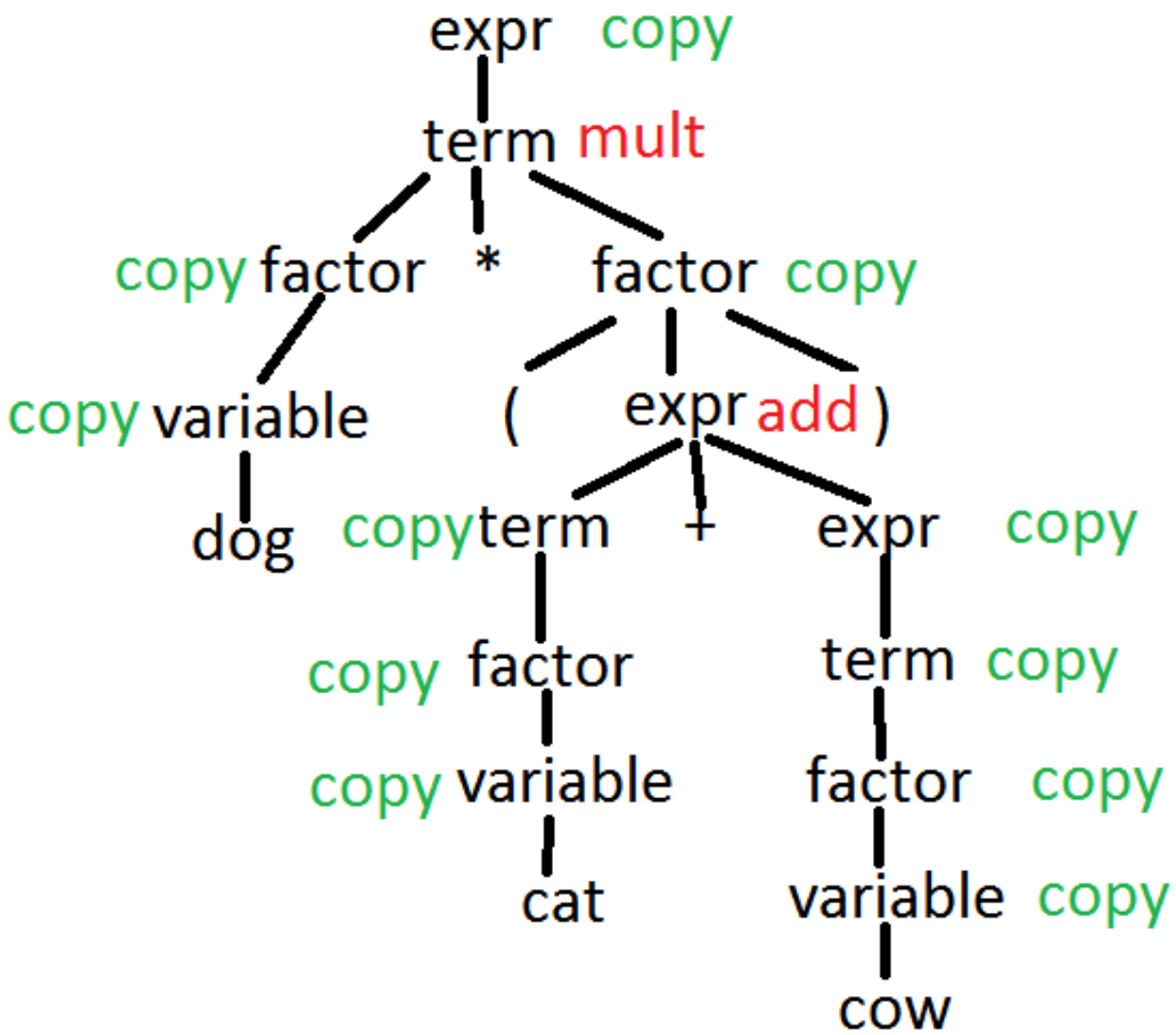
- Action rules are applied to the BNF productions to indicate what action needs to be taken to execute that portion of the program
- Action rules can either copy a value from a lower production in the tree or compute a function on the lower tree values
- Action rules can only involve lower items on the tree

Example Grammar with Actions

1.	expr → term	expr = term
2.	expr → term **+** $expr_2$	expr = sum(term, $expr_2$)
3.	expr → term **-** $expr_2$	expr = diff(term, $expr_2$)
4.	term → factor	term = factor
5.	term → factor * $factor_2$	term = mult(factor,$factor_2$)
6.	term → factor **/** $factor_2$	term = div(factor,$factor_2$)
7.	factor → **variable**	factor = variable
8.	factor → **(** expr **)**	factor = expr

Annotated Parse Tree Example

dog * (cat + cow)



Postfix Traversal

- Traversing the parse tree in a postfix order, taking the specified actions, will correctly execute the program
- After the parser has built a parse tree, the semantic analyzer can traverse the tree in postfix order
- Executable code can be generated in the order found by a postfix traversal

Topics

• Language Paradigms	chapter 1
• Theory of language	section 2.4
• BNF	section 2.1
• Scanning	section 2.2
• Parsing	section 2.3

Likely Exam Questions

- Write FSA for a scanner
- Write a BNF
- Write a recursive descent parser

Exam Schedule

- The first COMP360 exam scheduled for last Friday has been postponed until Wednesday, February 19, 2020
- You are allowed on 8½ by 11” page of notes
- The exam will cover everything since the start of class
  - Writing a scanner
  - Writing a parser
  - Creating an FSA and BNF
- A sample exam is on Blackboard

Snowflake Parser

- The Parsing programming assignment due date has been moved back to Friday, February 21, 2020
- You may do this assignment in teams of two students
- The BNF for Snowflake is provided
- You may use the provided scanner