Chapter 3b: Syntax

Review: Syntax Diagrams

- **Remember! One** syntax diagram **PER** rule.
- **“Branching out”** will occur if there are **choices.**
- Considering the following grammar:

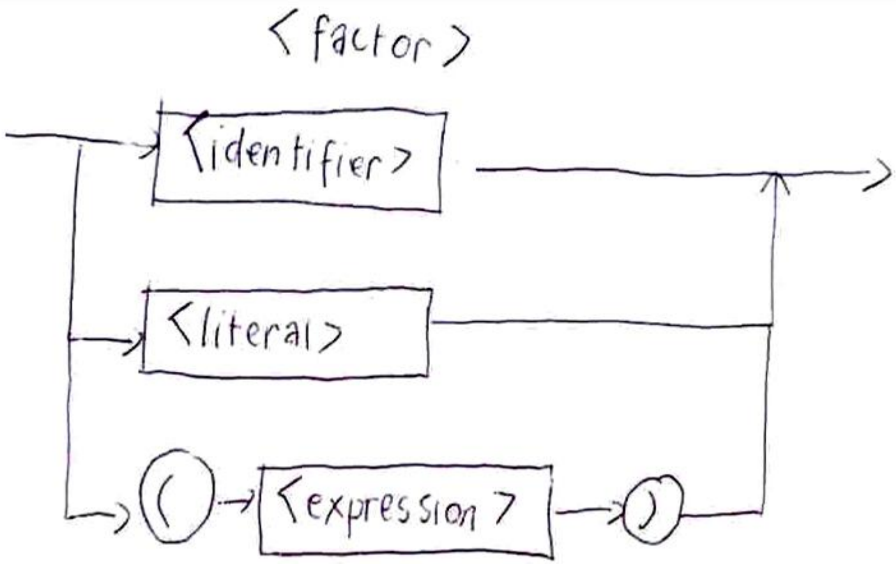
  <factor> ::= <identifier> | <literal> | (<expression>)

  <addop> ::= + | - | or

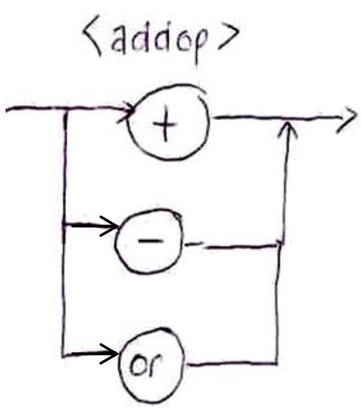
  <term> ::= <term><expression>

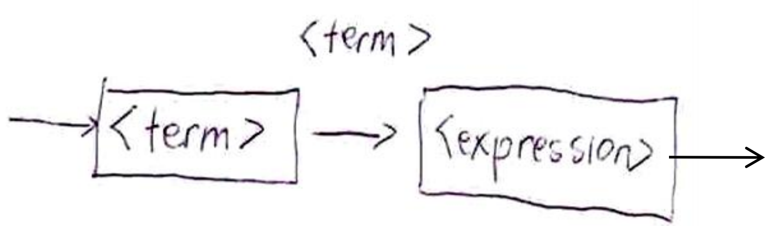
Chapter 3b: Syntax

Review: Syntax Diagrams



<factor> ::= <identifier> | <literal>
| (<expression>)
<addop> ::= + | - | or
<term> ::= <term> <expression>





The three syntax diagrams, handwritten by Sir Rein

Chapter 3c: Semantics

CMSC 124, 1st Semester, AY 2009-10

Chapter 3c: Semantics

Defining Semantics

- **Semantics** defines the meaning of syntactically correct programs.

- It is usually defined informally by attaching explanations and examples to syntax rules in a grammar for the language

Chapter 3c: Semantics

Defining Semantics

Static Semantics

- Meaning of the program at compile time.
- Eg:
    - Type-checking
    - <int> = <float> is illegal but opposite is legal.

Dynamic Semantics

- Meaning (or behaviour) of the program at runtime.
- Eg:
    - Behaviour of a certain while-loop
- Under dynamic semantics:
    - Operational Semantics
    - Denotational Semantics
    - Axiomatic Semantics

Operational Semantics

- It describes how a valid program is interpreted as sequences of computational steps.
  - These sequences then are the meaning of the program.

- It tells how a computation is performed by defining how to simulate the execution of the program.

- It is usually used when teaching a programming language and by compiler writers.

What is the operational semantics of:

```
if (x>0) then x=1;
else x=-1;
```

We evaluate x>0 and if true, it assigns the value 1 to x, otherwise it assigns the value -1 to x

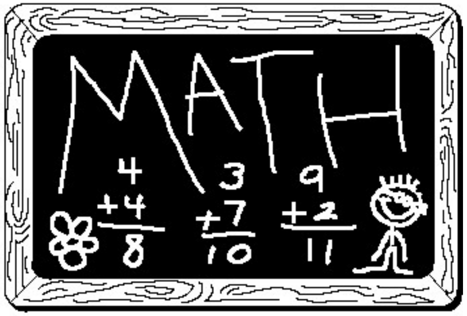
Denotational Semantics

- AKA mathematical semantics.

- A programming language is defined by a valuation function that maps programs (phrases) into mathematical objects considered as their **denotation** (meaning).

- It works as follows: a function is defined that maps a valid expression onto some mathematical objects.

- Mathematical objects include integers, truth values, tuples of values.



Chapter 3c: Semantics

Denotational Semantics

- Programs and objects manipulated are **symbol realizations** of abstract mathematical objects.
  - Strings of digits **realize** numbers.
  - Function subprograms **realize** (approximate) mathematical functions.

- **The Idea:** Associate an appropriate mathematical object (such as a number, tuple, function) with each phrase of the language.

Eg:

- Sample Phrases:
  - "2+4"
  - "(5+3)"
  - "008"
- Notation:
  - meaning[2+4] = 8
  - meaning[(5*3)] = 8
  - meaning[008] = 8

Chapter 3c: Semantics

Denotational Semantics Specification

1. **Syntactic Categories/Domain**
   - C: Command
   - E: Expression
   - N: Numeral
   - I: Identifier

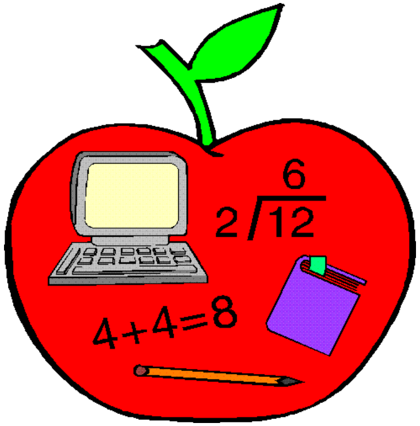
2. **Abstract Production Rules**
   - Describe ways syntactic categories may be combined in accordance with the BNF definition of the language.

Denotational Semantics Specification

3. **Semantic Domains**
   - "Sets" of mathematical objects.
   - Eg:
     Boolean = {true, false}
     Integer = {…,-2,-1,0,1…}

4. **Semantic Functions**
   - Maps the syntactic domains with the semantic domains.
   - Eg:
     evaluate: expression->values

5. **Semantic Equations**
   - Specify how the functions act on each pattern in the syntactic definition of the language phrases.
   - Eg:
     evaluate[E1+E2] =
     plus(evaluate[E1],evaluate[E2])



Denotational Semantics Specification: Example

Simple Language of Non-Negative Integer Numerals

1. Syntactic Domains
   - N: Numeral (non-negative numerals)
   - D: Digit (decimal digits)

2. Abstract Production Rules
   - Numeral ::= Digit | Numeral Digit
   - Digit ::= 0 | 1 | 2 | 3 | … | 9

3. Semantic Domain
   - Number = {0, 1, 2, 3, …..} (natural numbers)

4. Semantic Functions
   - value: Numeral -> Number
   - digit: Digit -> Number

Denotational Semantics Specification: Example

Simple Language of Non-Negative Integer Numerals

5. Semantic Equations
   - value[ND] = plus(times(10, value[N]), digit[D])
   - value[D] = digit[D]
   - digit[0] = 0 digit[5] = 5

     digit[1] = 1 digit[6] = 6

     digit[2] = 2 digit[7] = 7

     digit[3] = 3 digit[8] = 8

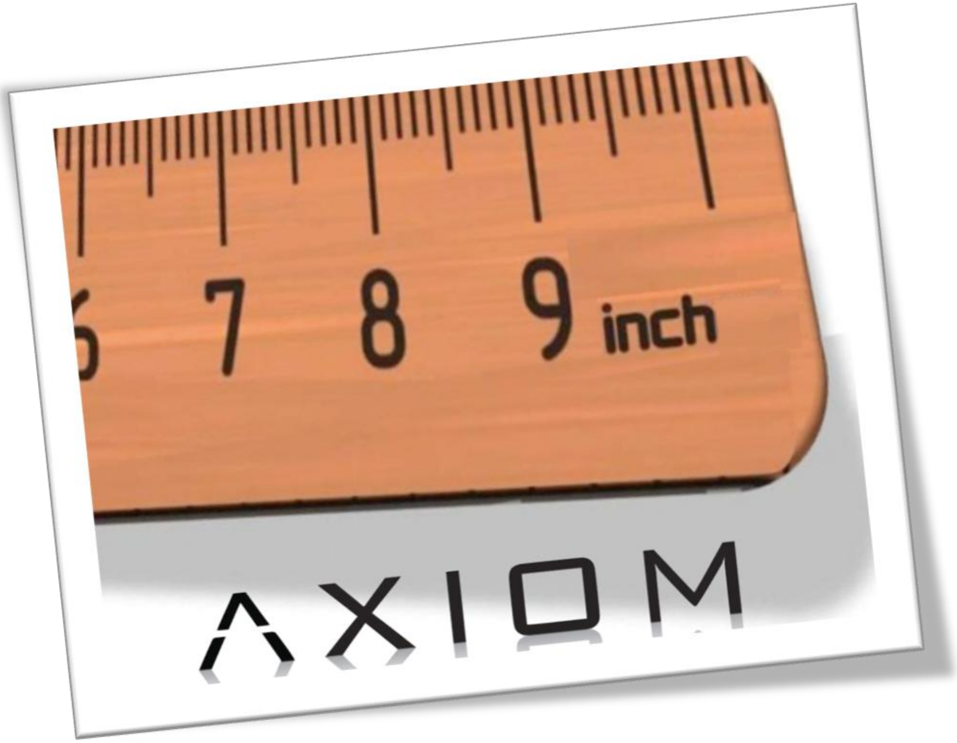
     digit[4] = 4 digit[9] = 9

Now, evaluate numeral 65 using the denotational definition.

Chapter 3c: Semantics

Axiomatic Semantics

- Based on methods of logical deduction from predicate logic.
- More abstract than denotational and operational.
- It is defined by correctness assertions that describe how to draw conclusions about the I/O interface of a program.
- It uses predicate calculus as a notation to describe constraints.



Axiomatic Semantics

- **Notation**

  {P} S {Q}

  where
  P – precondition
  Q – postcondition
  S – statement
  P, Q - logical expressions called predicates or assertions

- If S is executed in a state in which assertion P is satisfied and S terminates, then S terminates in a state in which assertion Q is satisfied.

- Eg:
  {a>=0 ^ b>=0} c = a*b
  {c = a * b}
  - Pre-condition is {a>=0 ^ b>=0}.
  - Post-condition is {c = a * b}.

Axiomatic Semantics: Example on Simple Assignment Statements

- Prove the correctness of
  sum = 2 * x + 1 {sum > 1}

  Question: What is {P}?

  - {x > 10} ?
  - {x > 1000} ?
  - {x > 0} ? → **"Weakest Precondition"**
  - {x >= 0} ?

  Answer: The first three are possible preconditions but the last one is not.

Axiomatic Semantics: Example on Simple Assignment Statements

- Prove the correctness of
  x = 2 * y – 3 {x > 25}
  Question: What is weakest precondition {P}?
  Answer: {y > 14}

GET 1/4

- **Exercise @ Home:** Prove the correctness of
  x = x + y – 3 {x > 10}
  Question: What is weakest precondition {P}?

Axiomatic Semantics: Example on Sequences

- Prove the correctness of
  y = 3 * x + 1
  x = y + 3
  {x < 10}

  Question: What are the weakest precondition/s?

- Answers/Complete Notation
  {x < 2}
  y = 3 * x + 1
  {y < 7}
  x = y + 3
  {x < 10}

Axiomatic Semantics: Example on Selections

- Prove the correctness of

  if (x > 0) y = y -1
  else y = y + 1
  {y > 0}

  Question: What is weakest precondition {P}?

  - Precondition 1 (if)

    y = y – 1
    {y > 0}

    Precondition 1 is
    {y > 1}

  - Precondition 2 (else)

    y = y + 1
    {y > 0}

    Precondition 2 is
    {y > - 1}

  Which will hold? {P} is {y>1}