UNIVERSITY OF NICOSIA
ΠΑΝΕΠΙΣΤΗΜΙΟ ΛΕΥΚΩΣΙΑΣ

COMP-421 Compiler Design

Presented by
Dr Ioanna Dionysiou

Administrative

- Any questions about the syllabus?
- Course Material available at
  - www.cs.unic.ac.cy/ioanna
- Next time reading assignment
  - [ALSU07] Chapters 1, 2

Lecture Outline

- [ASU07] Chapter 2 - A simple syntax-directed translation
  - Introduction to the material presented in [ALSU07] Chapters 3 through 6
    - Number of basic compiling techniques
    - Emphasis on the front-end of the compiler
  - Example presented is a program that translates infix expressions into postfix form

Infix expression --> Postfix Expression

- Translate infix expression into a postfix expression

9 - 5 + 2 ==> 9 5 - 2 +

What is the postfix expression of 8 * 2 + 2 - 1?

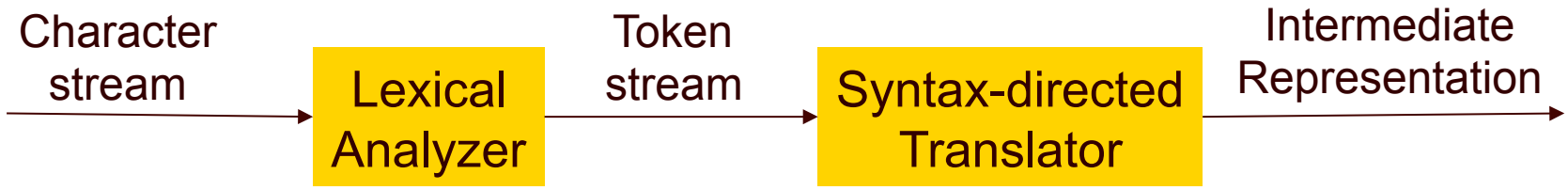
Lecture Outline

- Overview
- Syntax Definition
- Syntax-directed translation
- Parsing
- Translator for simple expressions
  - Lexical analysis, symbol table, parsing

Overview

- Language definition
  - What its programs look like
    - Syntax of the language
      - Context-free grammars (BNF)
  - What its programs mean
    - Semantics of the language
      - Informal descriptions
- Construct a compiler that translates infix expressions into postfix expressions
  - Operand operator operand
  - Operand operand operator

Overview



Is there something missing here?

Lecture Outline

- Overview
- Syntax Definition
- Syntax-directed translation
- Parsing
- Translator for simple expressions
  - Lexical analysis, symbol table, parsing

Syntax Definition

- Introduce a notation
  - Context-free grammar (or grammar)
    - Specify the syntax of a language
- A grammar describes the hierarchical structure of many programming language constructs
  - e.g. an if-else C statement
    - *if (expression) statement else statement*
      - Concatenation of ????

Syntax Definition

- `if (expression) statement else statement`
  - keyword if
  - Opening parenthesis (
  - expression
  - Closing parenthesis)
  - statement
  - Keyword else
  - statement
- Rewrite the above statement

  `stmt → if ( expr) stmt else stmt`

Syntax Definition

```
stmt → if ( expr) stmt else stmt
```

This is a rule called production

→ means “can have the form”

if, else, (,) are called tokens (terminals)

stmt, expr are called nonterminals (sequence of tokens)

Context-free grammar components

- It has 4 components
  - A set of tokens, known as terminal symbols
  - A set of nonterminals
  - A set of productions where each production consists of
    - A nonterminal called the left side of the production
    - An arrow
    - A sequence of tokens and/or nonterminals called the right side of the production
  - A designation of one of the nonterminals as the start symbol

Context-free grammar

stmt → **if** (expr) stmt **else** stmt

expr → **0**

expr → **1**

How many productions?

Left side, right side, terminals, nonterminanls, start symbol?

Context-free grammar

- Conventions
  - Productions for the start symbol listed first
  - Productions with the same nonterminal on the left can be grouped together
    - (separated by the symbol |)
    - We will use this convention only when the right side consists of terminals

In-class Exercise

- Derive a grammar for expressions

  9 - 5 + 2

  3 -1

  7

  Such expressions are “lists of digits separated by plus or minus signs” or a list of one digit

Solution

list → list **+** digit

list → list **-** digit

list → digit

digit → **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

A grammar derives strings by beginning with the start symbol and repeatedly replacing a nonterminal by the right side of a production for that nonterminal.

All the strings that can be derived from the start symbol form the language defined by the grammar

In-class Exercise

- Deduce that 9 - 5 + 2 belongs to the language defined by the previous grammar.
- Does 9 belong to the language?
- Does 3 * 1 belong to the language?

Solution

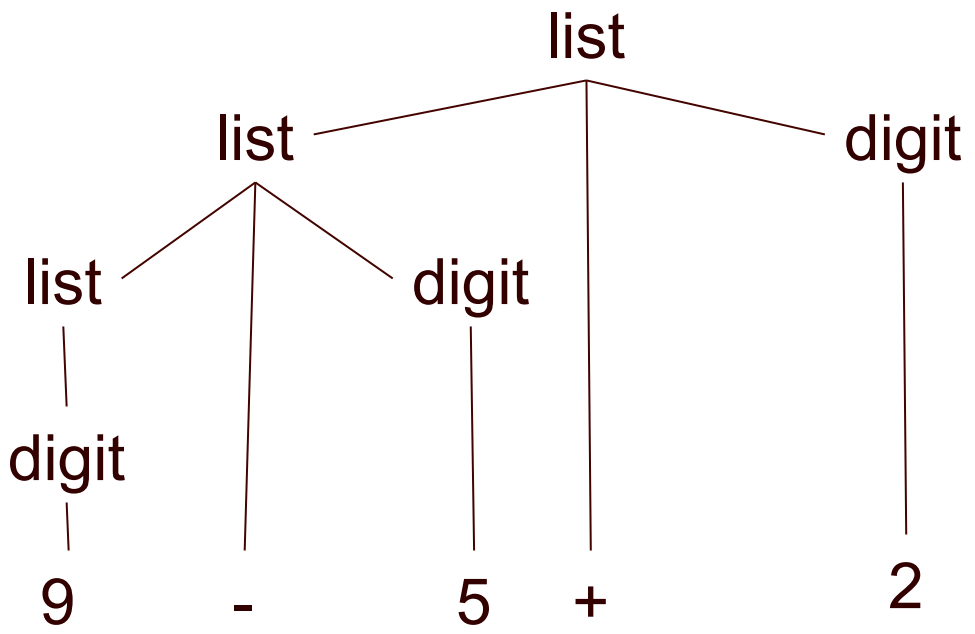
- 9 is a list
  - by production #3
    - 9 is a digit
- 9 - 5 is a list
  - by production #2
    - 9 is a list and 5 is a digit
- 9 - 5 + 2 is a list
  - by production #1
    - 9 - 5 is a list and 2 is a digit

Parse Tree for 9-5+2

A node is labeled by a grammar symbol

Interior Node and its children refer to a production

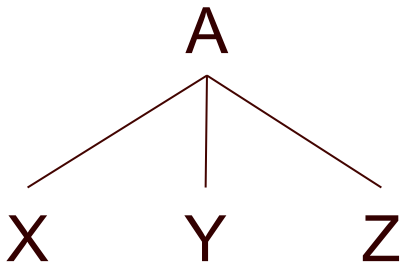
- interior node refers to the left side
- children nodes refer to right side



Parse Trees

- It pictorially shows how the start symbol of a grammar derives the string in the language
  - Suppose that we have production

    $A \rightarrow X\ Y\ Z$



Parse Trees

- Formally, given a context-free grammar, a parse tree is a tree with the following properties:
  - The root is labeled by the start symbol
  - Each leaf is labeled by a token or by ε (empty string)
  - Each interior node is labeled by a nonterminal
  - If
    - A is the nonterminal labeling some interior node and
    - $X_1, X_2, \ldots, X_n$ are the labels of the children of that node from left to right
  - then
    - $A \rightarrow X_1 X_2 \ldots X_n$ is a production
    - Here $X_1, X_2, \ldots, X_n$ stand for a symbol that is either a terminal or a nonterminal.

Parse tree

- The leaves of a parse tree read from left to right form the string generated or derived from the nonterminal at the root of the parse tree.

- Parsing is the process of finding a parse tree for a given string of tokens

Parse Trees and Ambiguity

- While it is clear that each parse tree derives exactly one string (read off its leaves), a grammar can have more than one parse tree generating a given string of tokens
  - Such a grammar is said to be ambiguous
    - All we need to do is find a token string that has more than one parse tree

In-class Exercise

string → string **+** string

string → string **-** string

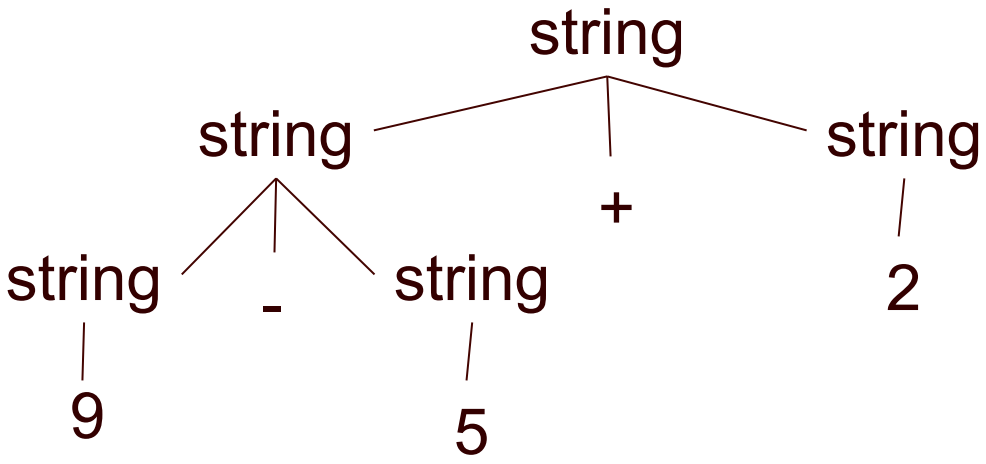
string → **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

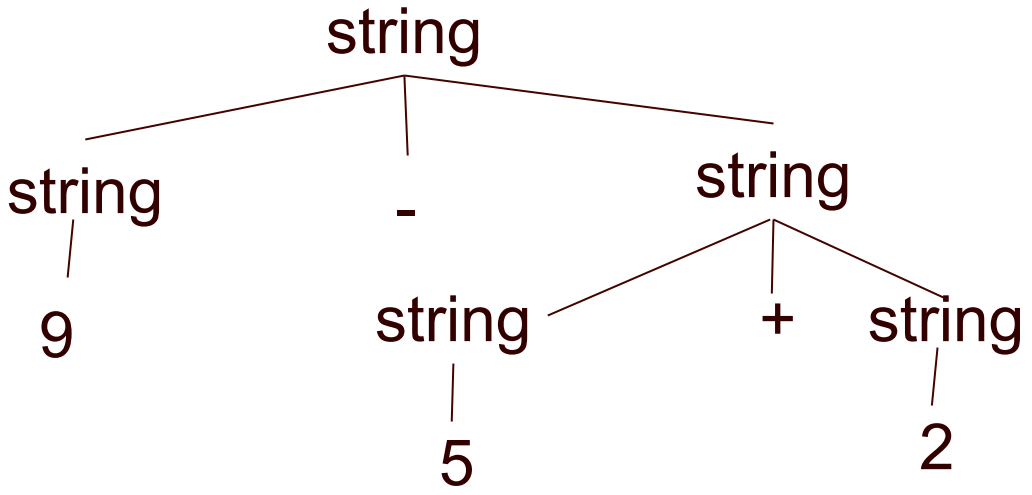
Can you derive more than one parse trees for the string 9 - 5 + 2?

Solution

- Yes, there are two parse trees depending on how we parenthesize the expression
  - (9-5) + 2
  - 9 - (5 + 2)
  - Note: the previous grammar did not allow the second interpretation

Solution





Associativity of Operators

- 9 + 2 + 1 is equivalent to (9 + 2) + 1
- 9 - 2 - 1 is equivalent to (9 - 2) - 1

- When an operand has two operators to its left and its right, we need to decide which operator takes that operand
  - Operators +, -, *, \ are left-associative
    - E.g operand with the + signs on both sides of it is taken by the operator to its left

Precedence of Operators

- Consider 9 + 5 * 2
  - Two possible interpretations
    - (9+5)*2
    - 9+(5*2)
  - The Associativity of + and * do not resolve this ambiguity
  - We need to know the relative precedence of operators when more than one kind of operator is present

Precedence of Operators

- * has higher precedence than +
  - It takes its operands before + does

- [ASU07] page 49
  - Grammar for arithmetic expressions that follow the Associativity and precedence rules.

Lecture Outline

- Overview
- Syntax Definition
- Syntax-directed translation
- Parsing
- Translator for simple expressions
  - Lexical analysis, symbol table, parsing

Syntax-directed Translation

- To translate a programming construct the compiler may need to know
  - Type of the construct
  - Location of the first instruction in the target code
  - Number of instructions generated

- These are the attributes associated with constructs
  - An attribute may represent any quantity
    - String, Type, Memory location
    - Etc…

Syntax-directed definition

- A formalism for specifying translations for programming language constructs
  - Uses a context-free grammar to specify syntactic structure of input
    - With each grammar symbol it associates a set of attributes
  - With each production it associates a set of semantic rules for computing values of the attributes associated with the symbols appearing in the production

Annotated Parse Tree

- A translation is an input-output mapping
  - Suppose we have input x
  - First, construct parse tree for input x
  - Suppose that a node n in the parse tree is labeled by grammar symbol Y
    - Write Y.a to denote the value of attribute a of Y at that node
    - Value of Y.a at n is computed using the semantic rule for attribute a associated with the Y-production used an node n
- A parse tree showing the attribute values at each node is called an annotated parse tree

Synthesized Attributes

- An attribute is said to be synthesized if its value at a parse-tree node is determined from attributes at the children of the node
  - Advantage: evaluated during a single bottom-up traversal of the parse tree

Example

- Syntax-directed definition for translating expressions consisting of digits separated by plus or minus signs into postfix notation

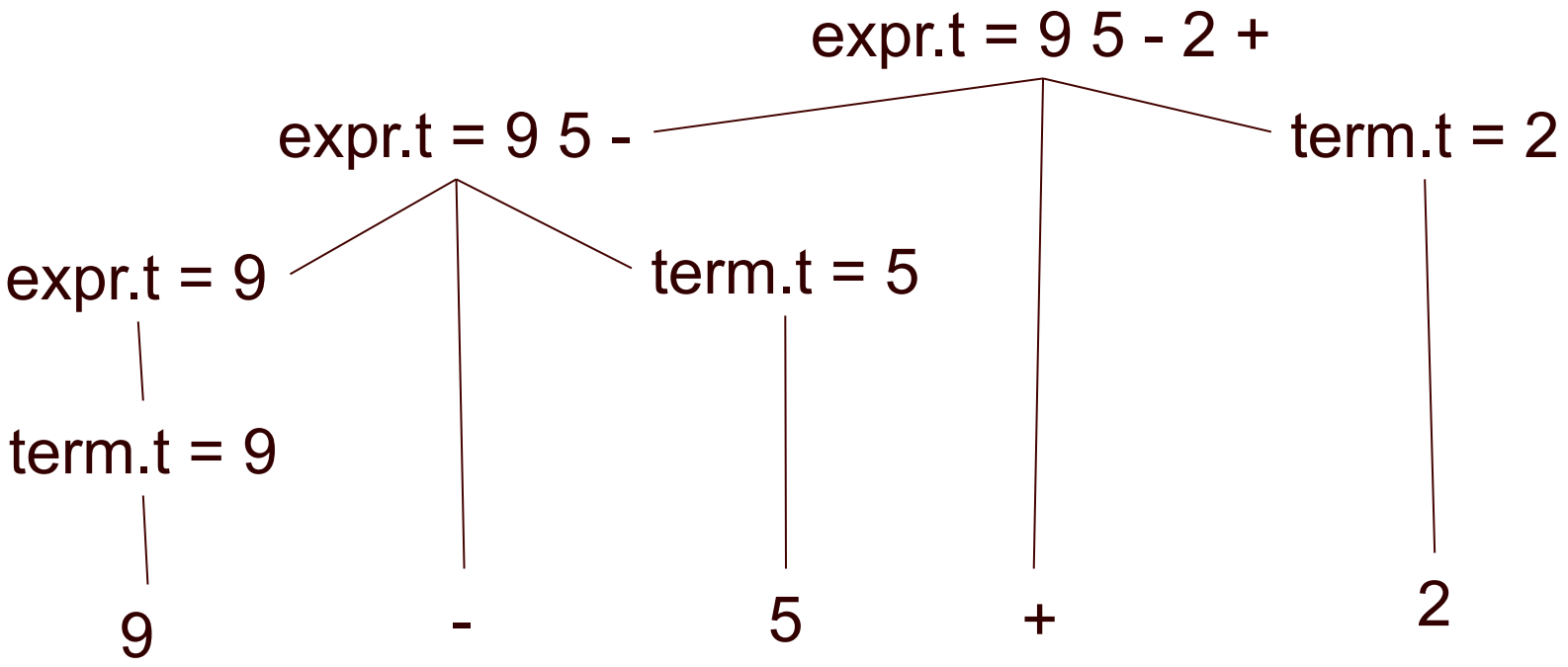
PRODUCTION	SEMANTIC RULE
expr → expr + term	expr.t := expr.t || term.t || '+'
expr → expr - term	expr.t := expr.t || term.t || '-'
expr → term	expr.t := term.t
term → 0	term.t := '0'
term → 1	term.t := '1'
term → 2	term.t := '2'
…	
term → 9	term.t := '9'

String-valued attribute t represents the postfix notation for the expression generated by that nonterminal in a parse tree

In-class Exercise

- Derive the annotated parse tree for input 9 - 5 + 2 according to the syntax-directed definition shown at the previous slide

Solution



A syntax-directed definition does not impose any specific order for the evaluation of attributes in a parse tree. The only requirement is that the value of an attribute a is computed after all the other attributes that a depends on are computed

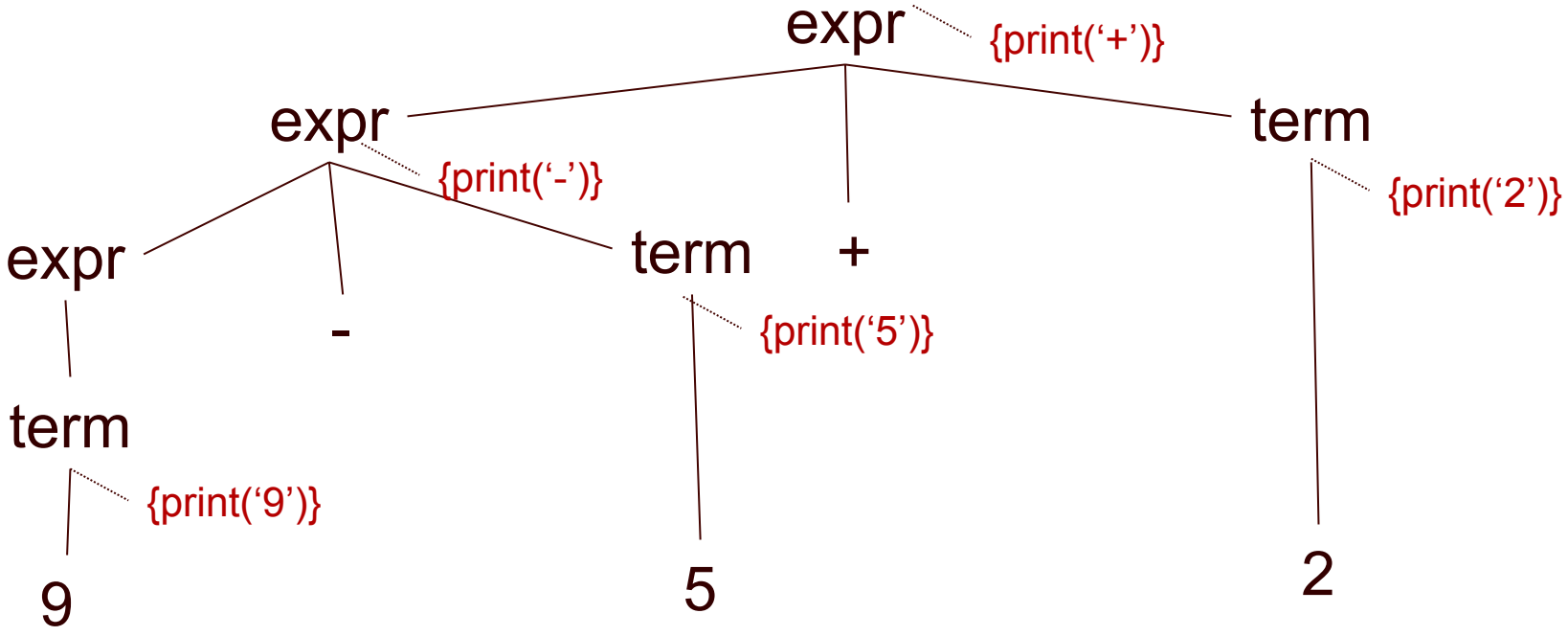
Translation Schemes

- Translation scheme is a context-free grammar in which program fragments called semantic actions are embedded within the right sides of productions
  - Similar to syntax-directed definition
  - Order of evaluation of the semantic rules is explicitly shown

Actions translating expressions into postfix notation

Production	Action
expr → expr + term	{ print('+') }
expr → expr - term	{ print('-') }
expr → term	
term → 0	{ print('0') }
term → 1	{ print('1') }
term → 2	{ print('2') }
…	
term → 9	{ print('9') }

Parse Tree for Translation Scheme



Lecture Outline

- Overview
- Syntax Definition
- Syntax-directed translation
- Parsing
- Translator for simple expressions
  - Lexical analysis, symbol table, parsing

Parsing

- Process of determining if a string of tokens can be generated by a grammar.
  - Parse tree!
- A parser can be constructed for any grammar
  - Almost all programming language parsers
    - Make a single left-to-right scan over input
    - Look ahead 1 token at a time

Parsing Methods

- There are two methods, depending on the order in which nodes in the parse tree are constructed
  - Top-down
    - Construction starts at the root and proceeds towards the leaves
  - Bottom-up
    - Construction starts at the leaves and proceeds towards the root

Top-Down Parsing

- [ASU07] Figure 2.17
  - Top-down parsing while scanning the input from left-to-right
  - One lookahead token
    - Select a production for the nonterminal depending on the token read
    - Bakctracking is allowed
      - A production is unsuitable if, after using the production, we cannot complete the tree to match the input string
      - Go back and choose another production
      - EXCEPTION: predictive parsing (not allowed)

Predictive Parsing

- Predictive parsing is a form of recursive-descent parsing
  - Execute a set of recursive procedures to process the input
  - A procedure is associated with each nonterminal of a grammar
  - Lookahead symbol unambiguously determines the procedure to selected for each nonterminal

Lecture Outline

- Overview
- Syntax Definition
- Syntax-directed translation
- Parsing
- Translator for simple expressions
  - Lexical analysis, symbol table, parsing

Translator for infix->postfix

- [ASU07] Section 2.5
  - Using the techniques discussed so far, we can construct a syntax-directed translator (in java) that translates arithmetic expression into postfix form

Enhanced Translator

- [ASU07] Section 2.6
  - Add to the translator a lexical analyzer
    - Eliminate white spaces and comments
    - Recognize identifiers and keywords

Incorporating a Symbol Table

- Symbol table is a data structure
  - Stores information about programming language constructs
    - E.g. during lexical analysis the character string that forms an identifier is saved in a symbol-table entry
- Two main routines
  - Insert(s,t) : return index of new entry for string s, token t
  - Lookup(s) : return index of the entry for string s, or 0 if s is not found

Handling reserved words

- Insert("div",**div**)
  - String div
  - Token **div**
  - Any subsequent call lookup("div") returns the token **div**, so div cannot be used as an identifier

Abstract Stack Machine

Intermediate representation: one option is code for an abstract stack machine

Abstract Stack Machine

- There are three classes of instructions
  - Integer arithmetic
  - Stack manipulation
  - Control flow

Integer arithmetic

- Must implement each operator in the intermediate language
  - Addition, subtraction are supported directly by the abstract machine
  - Assumption: there is an instruction for each arithmetic operator
  - Abstract machine code for an arithmetic expression simulates the evaluation of a postfix representation for that expression using a stack

Example

- Evaluation of 1 3 + 5 *
  - Stack 1
  - Stack 3
  - Add two topmost elements, pop them and stack result 4
  - Stack 5
  - Multiply two topmost elements, pop them and stack result 20
  - Value on top of the stack at the end is the value of the entire expression (in this case 20)

Stack Manipulation

- Instructions

  push v (push v onto stack)

  rvalue l (push contents of data location l)

  lvalue l (push address of data location l)

  pop (remove value from top of the stack)

  := (the r-value on top is placed in the l-value below it and both are popped)

  copy (push a copy of the top value on the stack)

Control Flow

- Control-flow instructions for the stack machine are
  - label l
  - goto l
  - gofalse
  - gotrue
  - halt