Chapter 1:
Language Processor

Mrs. Sunita M Dol (Aher),
Assistant Professor,
Computer Science and Engineering Department,
Walchand Institute of Technology, Solapur, Maharashtra

Chapter 1: Language Processor

- Introduction
- Language Processing Activities
- Fundamentals of Language Processing
- Fundamentals of Language Specification
- Language Processing Development Tools

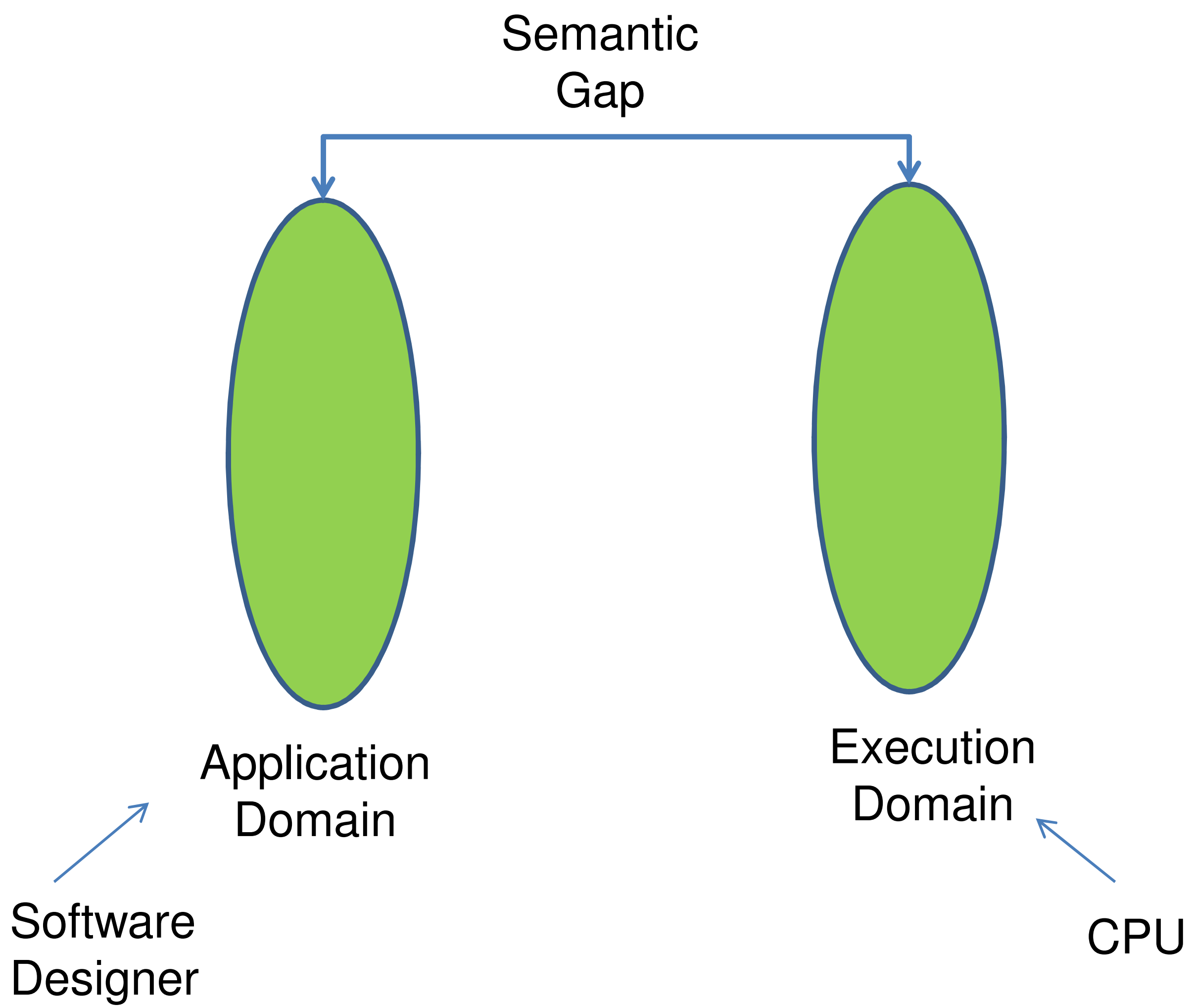
Chapter 1: Language Processor

- Introduction
- Language Processing Activities
- Fundamentals of Language Processing
- Fundamentals of Language Specification
- Language Processing Development Tools

Introduction

- Why Language Processor?
  - Difference between the manner in which software designer describes the ideas (How the s/w should be) and the manner in which these ideas are implemented(CPU).

Introduction

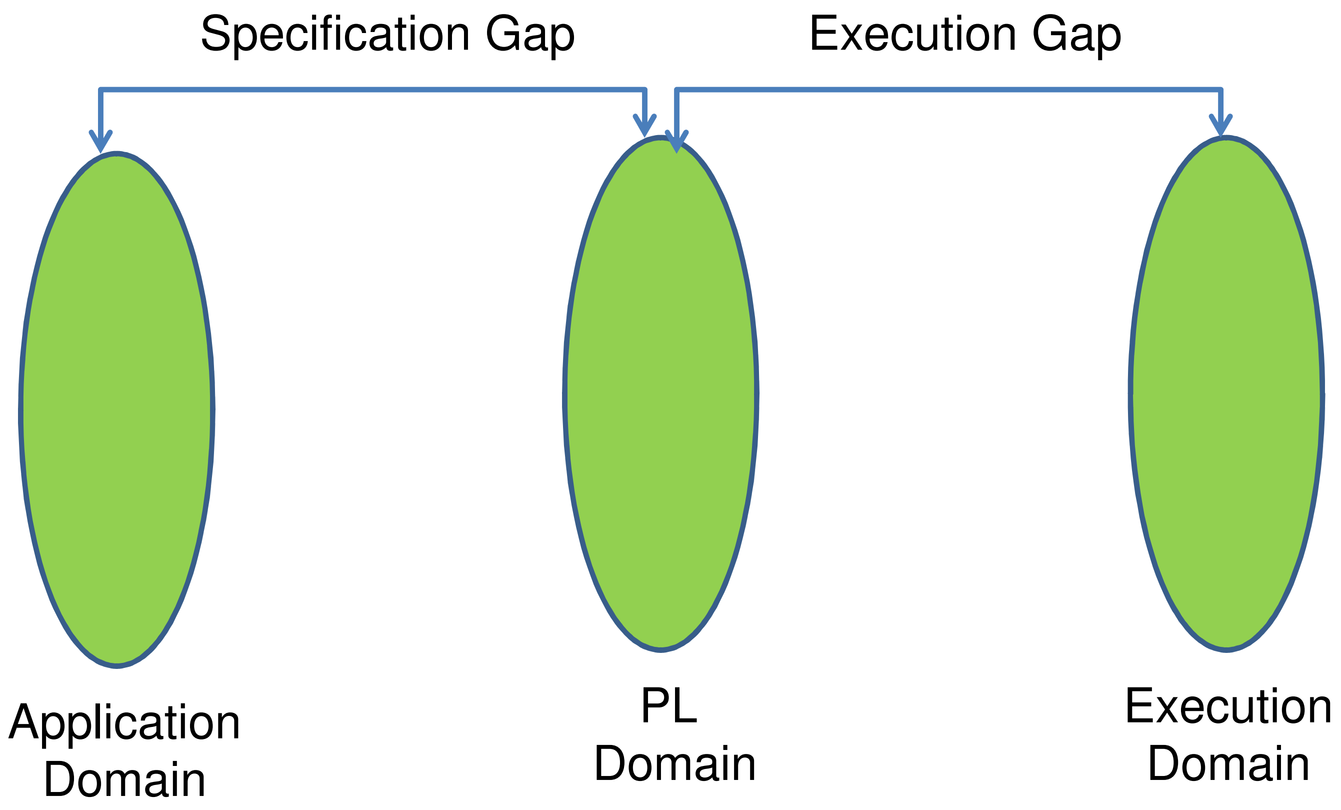


Introduction

- Consequences of Semantic Gap
    1. Large development times
    2. Large development efforts
    3. Poor quality of software

- Issues are tackled by Software Engineering through use of Programming Language.

Introduction



Introduction

- A Language Processor is a software which bridges a specification or execution gap.
- Program formed input to a Language Processor is referred as a Source Program and output as Target Program.
- Languages in which they are written are called as source language and target languages respectively.

A Spectrum of Language Processor

- A language translator bridges an execution gap to the machine language of a computer system.
- A detranslator bridges the same execution gap as the language translator but in reverse direction.
- A Preprocessor is a language processor which bridges an execution gap but is not a language translator.
- A language migrator bridges the specification gap between two PL's.

A Spectrum of Language Processor

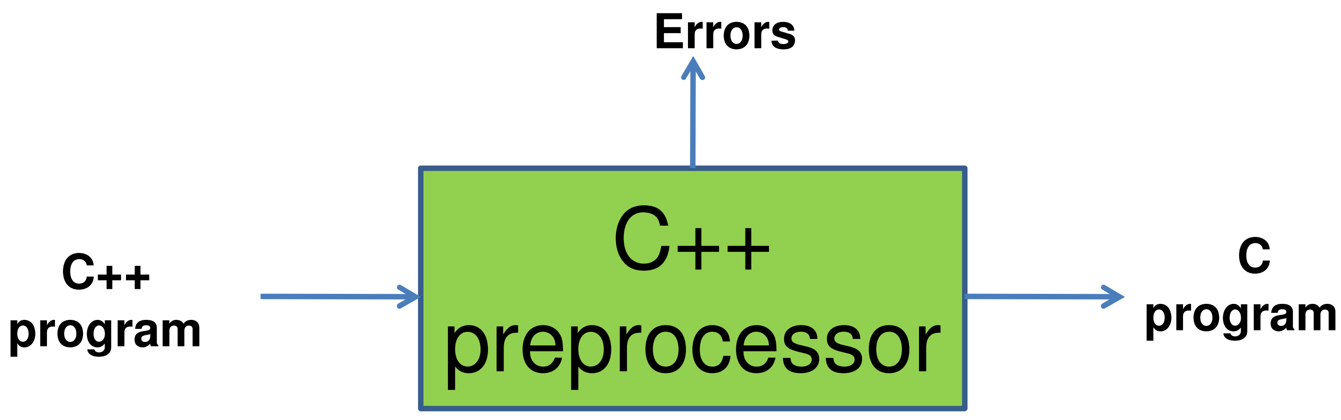


Figure : a

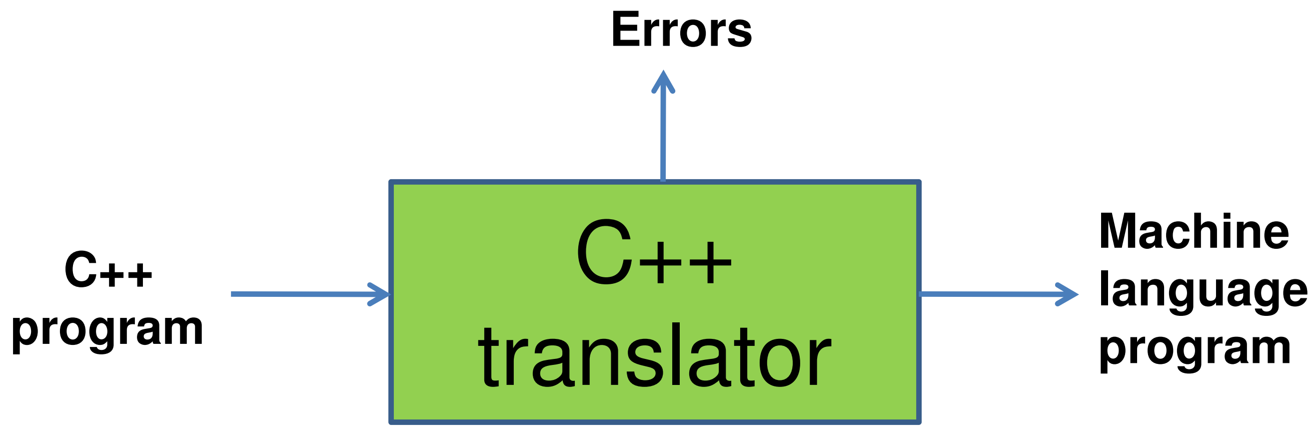
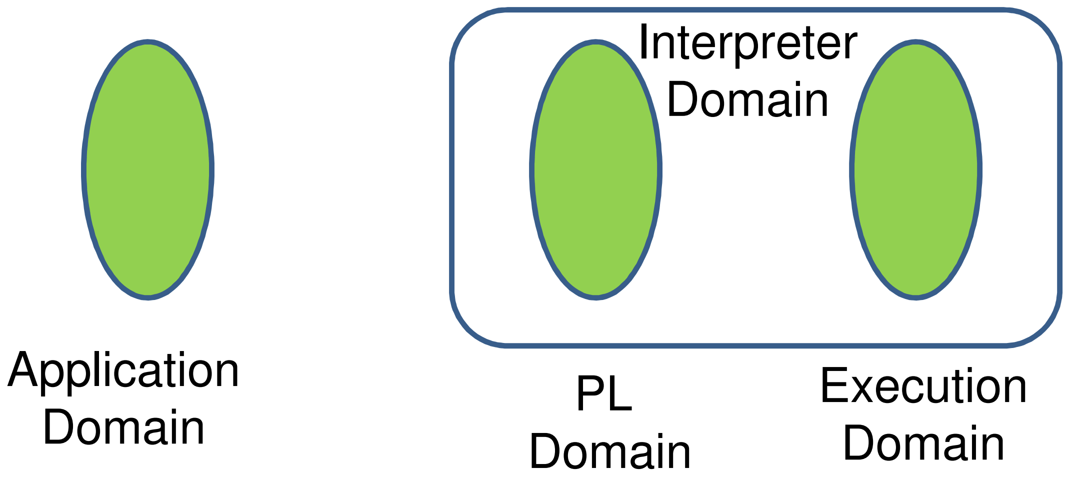


Figure : b

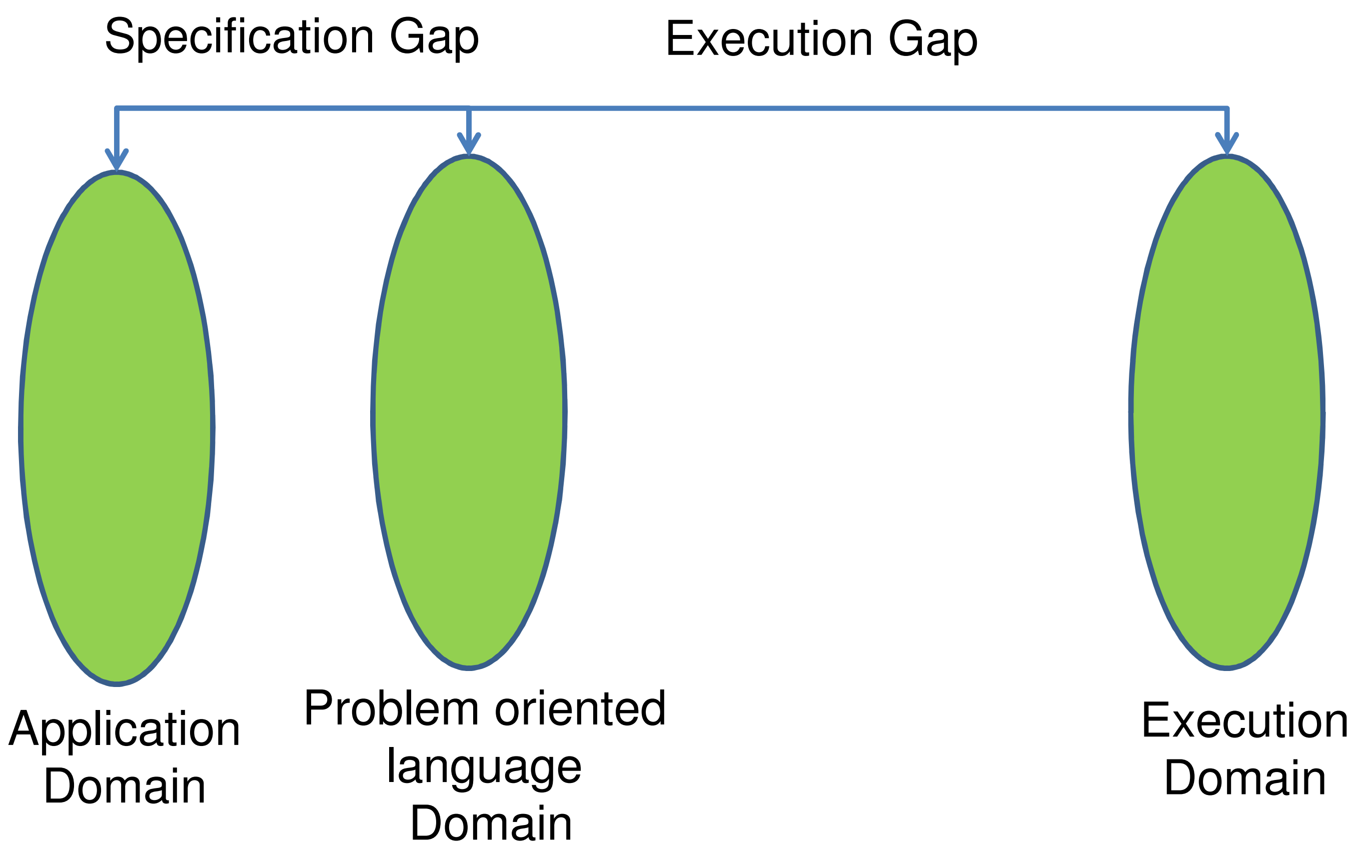
Interpreters

- An interpreter is a language processor which bridges an execution gap without generating a machine language program.
- Here execution gap vanishes totally.



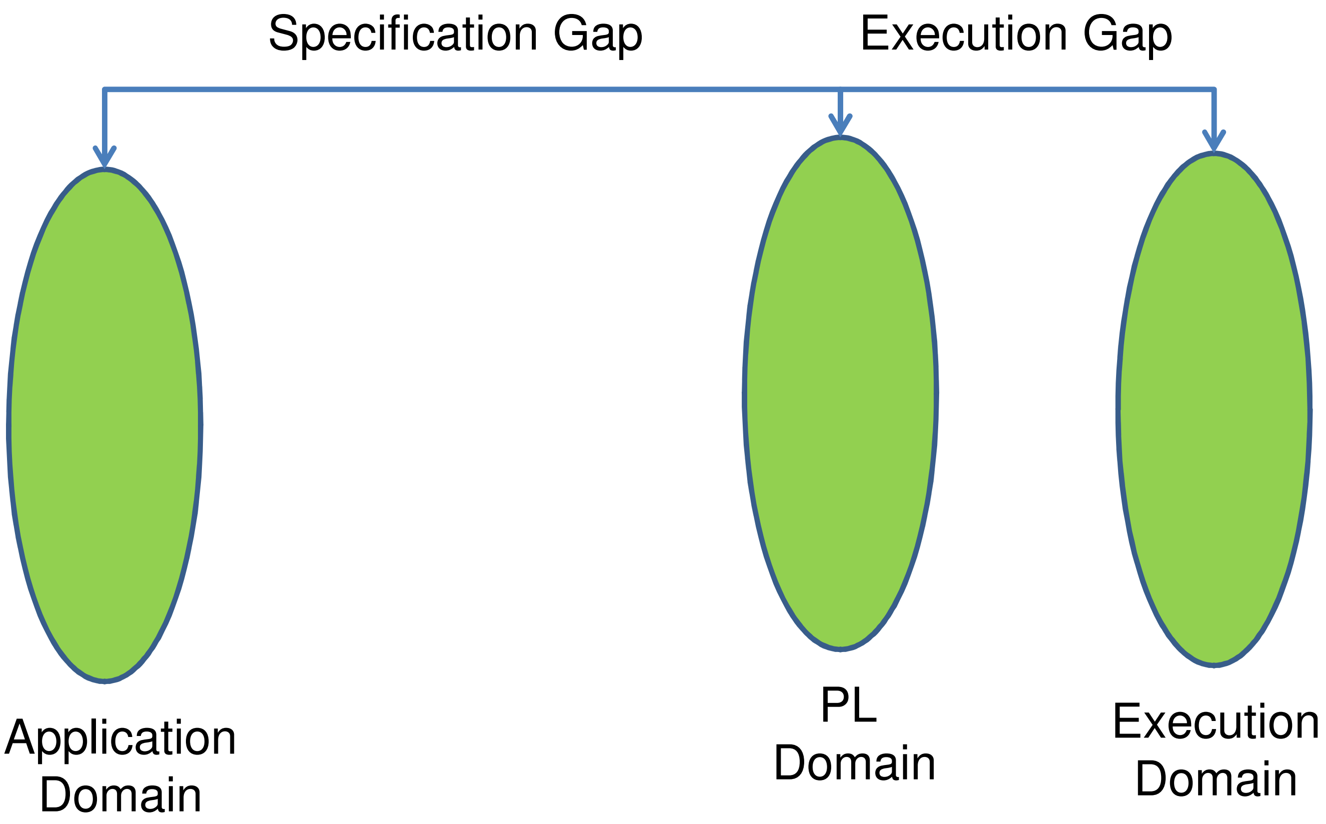
Problem Oriented Language

- These languages are used for specific application



Procedure Oriented Language

- These languages provides general purpose facilities required in most application domain.



Chapter 1: Language Processor

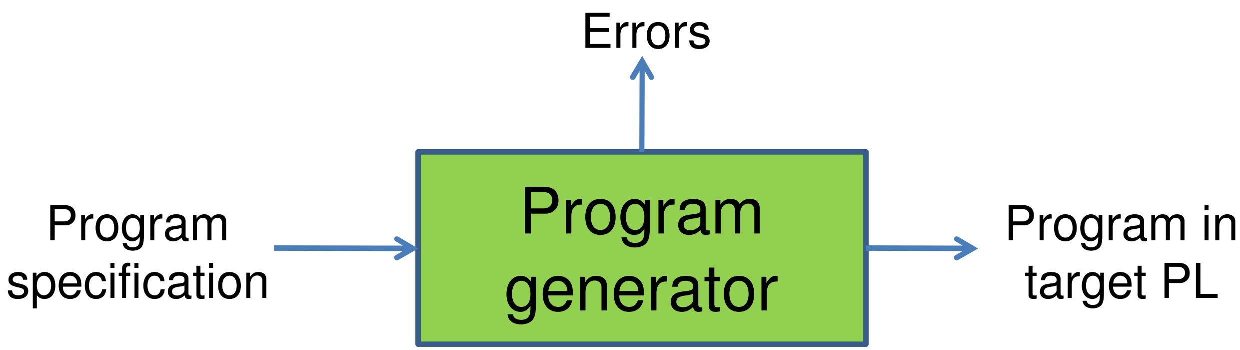
- Introduction
- Language Processing Activities
- Fundamentals of Language Processing
- Fundamentals of Language Specification
- Language Processing Development Tools

Language Processing Activities

- Language processing activities are divided into those that bridge the specification gap and those that bridge the execution gap.
  1. Program generation activities.
  2. Program execution activities.

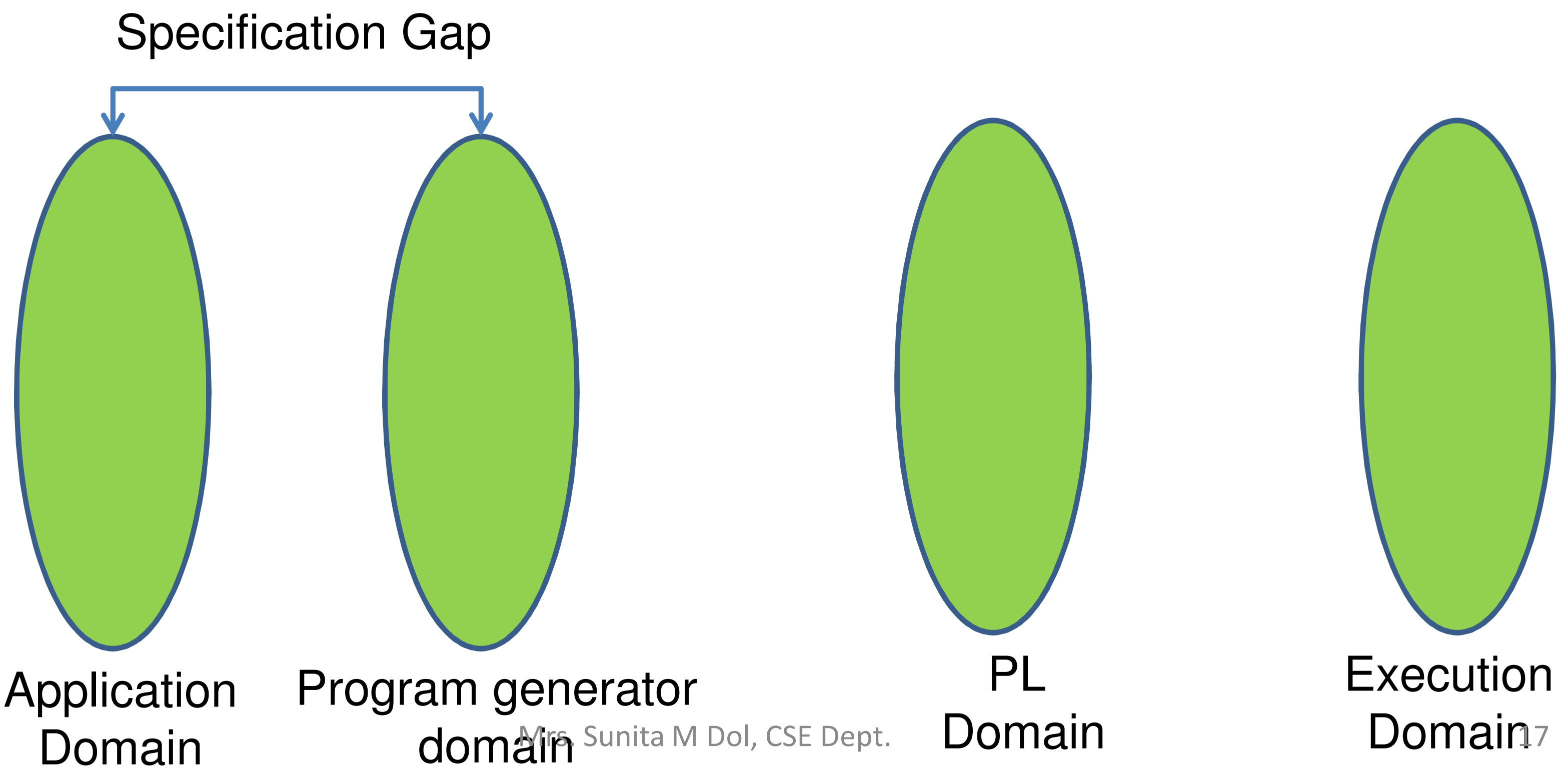
Language Processing Activities

- Program Generation:
  - It is a software which accepts the specification of a program to be generated and generates a program in the target PL.
  - Program generator introduces a new domain between the application and PL domain.



Language Processing Activities

- Program Generation:
  - –Program generator introduces a new domain between the application and PL domain.



Language Processing Activities

- Program Generation Example:
  - A screen handling Program
    - Specification is given as below

```
Employee name : char : start(line=2,position=25)
                       end(line=2,position=80)

Married       : char : start(line =10, position=25)
                       end(line=10,position=27)
                       default('Yes')
```

Language Processing Activities

- Program Generation Example:

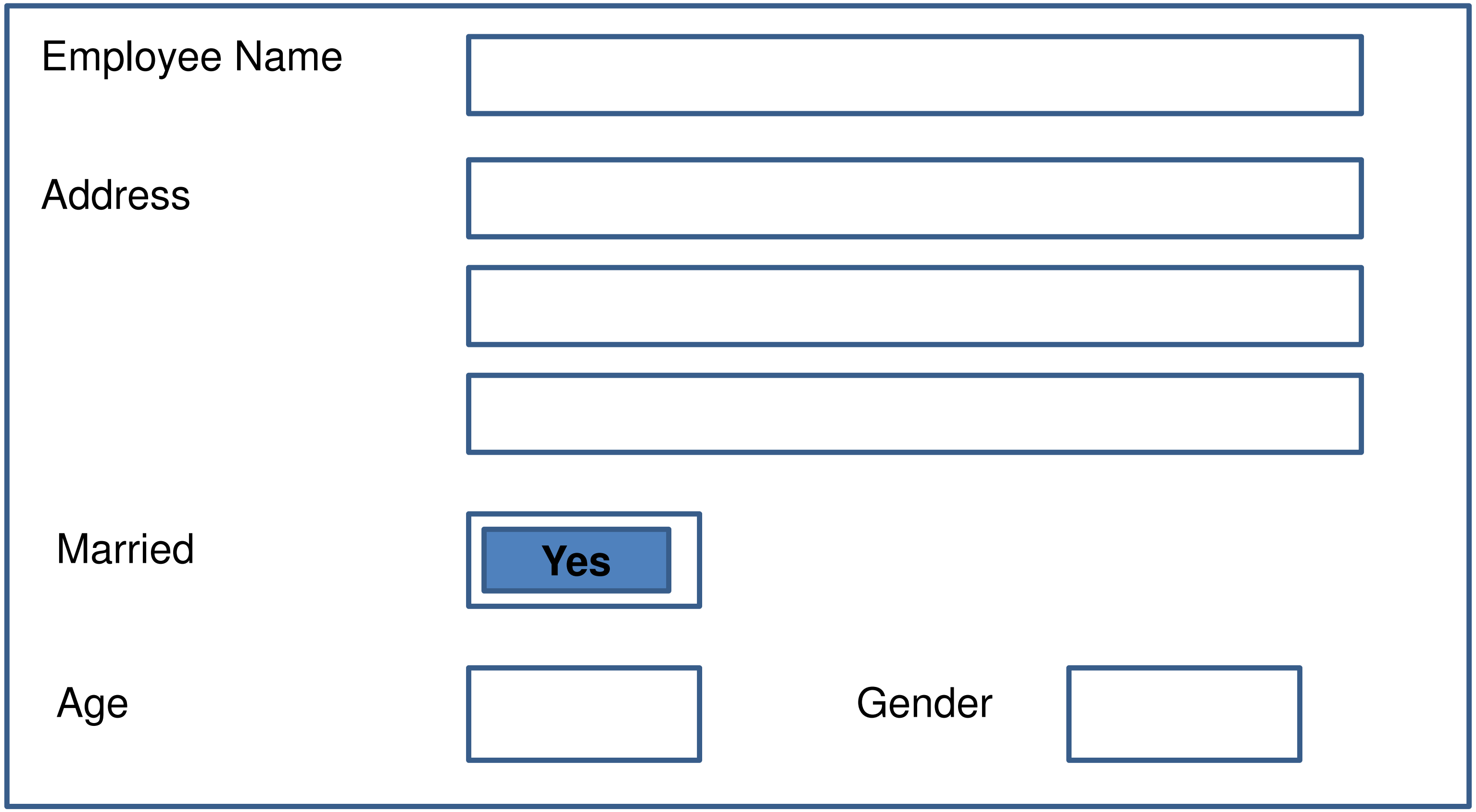


Figure: Screen displayed by a screen handling program

Language Processing Activities

- Program Execution
  - Two models of Program Execution
    - Program translation
    - Program interpretation

Language Processing Activities

- Program Translation:
  - Program translation model bridges the execution gap by translating a program written in PL i.e Source Program into machine language i.e Target Program.

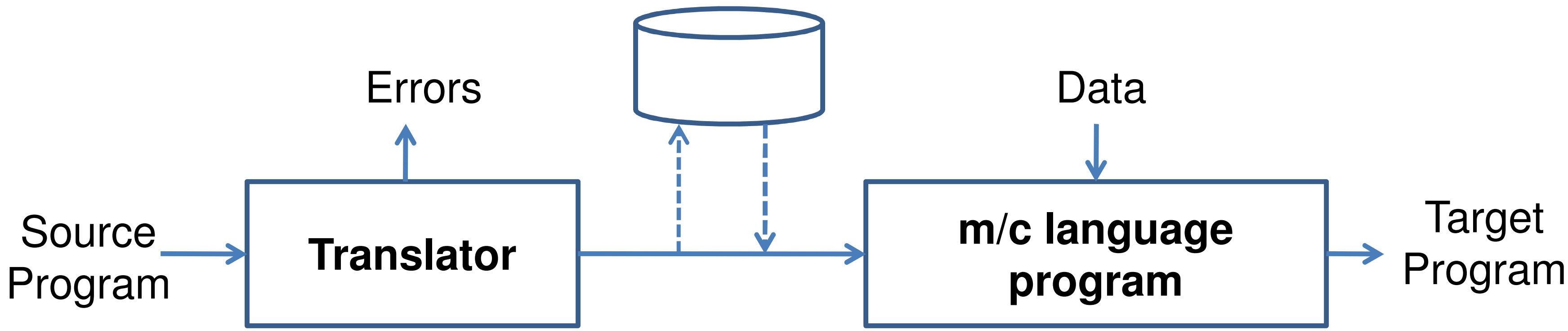


Figure: Program translation model

Language Processing Activities

- Program Interpretation
  - Interpreter reads the source program and stores it in its memory.
  - During interpretation it determines the meaning of the statement.

Language Processing Activities

- Program Interpretation:
  - The CPU uses a program counter (PC) to note the address of the next address.
  - Instruction Execution Cycle
    1. Fetch the instruction
    2. Decode the instruction and determine the operation to be performed.
    3. Execute the instruction.

Language Processing Activities

- Program Interpretation:
  - Interpretation Cycle consists of—
    1. Fetch the statement.
    2. Analyze the statement and determine its meaning.
    3. Execute the meaning of the statement.

Language Processing Activities

- Program Interpretation

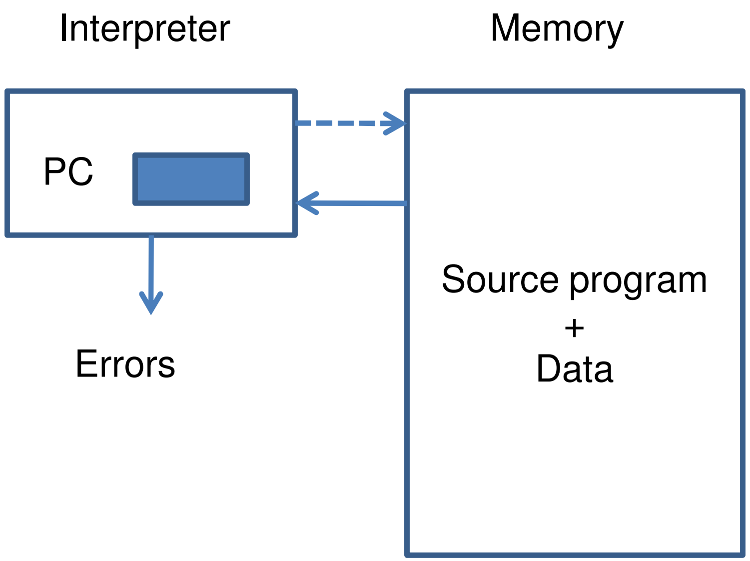


Figure (a) : Interpretation

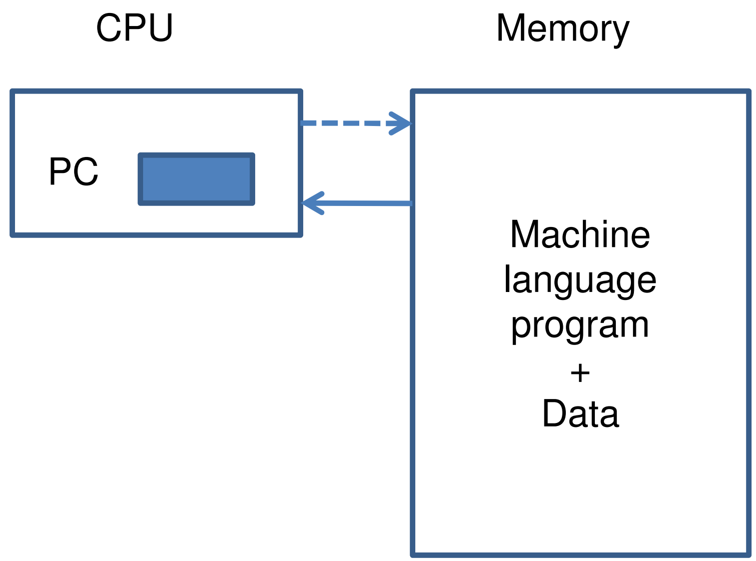


Figure (b) : Program Execution

Language Processing Activities

- Program Interpretation
  - Characteristics of interpretation
    1. Source program is retained in the source form itself i.e. no target program.
    2. A statement is analyzed during its interpretation.

Chapter 1: Language Processor

- Introduction
- Language Processing Activities
- Fundamentals of Language Processing
- Fundamentals of Language Specification
- Language Processing Development Tools

Fundamentals of Language Processing

- Language Processing = Analysis of Source Program
  + Synthesis of Target Program

- Analysis consists of three steps
  1. Lexical rule identifies the *valid lexical units*
  2. Syntax rules identifies the *valid statements*
  3. Semantic rules associate meaning with *valid statement*.

Fundamentals of Language Processing

Example:-

percent_profit := (profit * 100) / cost_price;

Lexical analysis identifies-------

:=, * and / as operators

100 as constant

Remaining strings as identifiers.

Syntax analysis identifies the statement as the assignment statement.

Semantic analysis determines the meaning of the statement as

$$\frac{\text{profit} \times 100}{\text{cost\_price}}$$ to percent_profit

Fundamentals of Language Processing

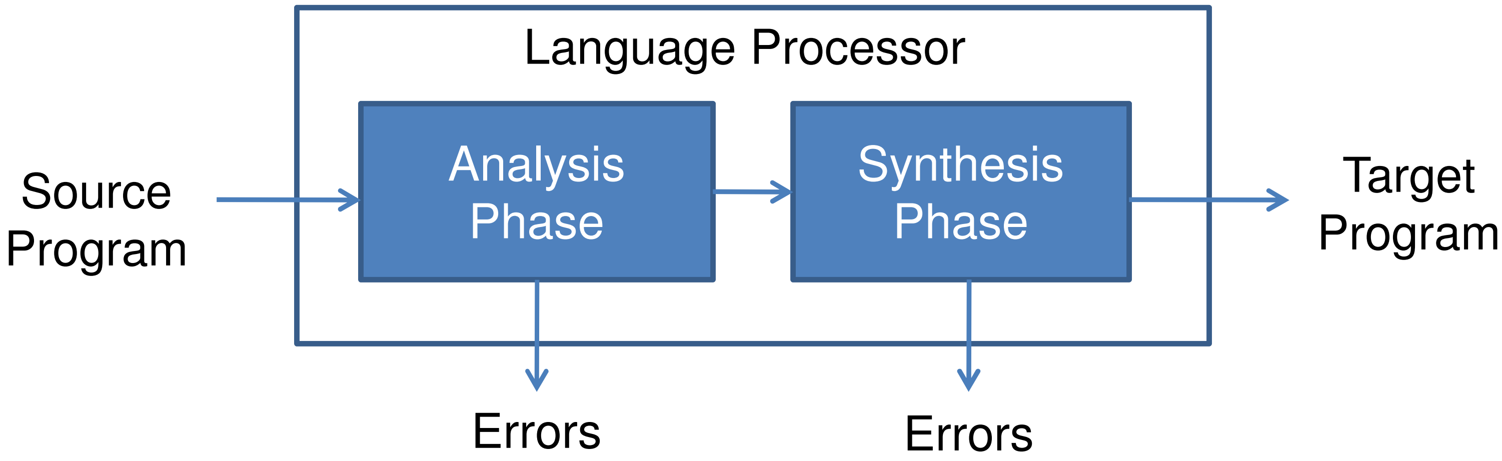
- Synthesis Phase
  1. Creation of Data Structure
  2. Generation of target code

  Referred as memory allocation and code generation, respectively.

```
MOVER            AREG, PROFIT
MULT             AREG, 100
DIV              AREG, COST_PRICE
MOVEM            AREG, PERCENT_PROFIT
-------------------
PERCENT_PROFIT   DW    1
PROFIT           DW    1
COST_PRICE       DW    1
```

Fundamentals of Language Processing

- Phases and Passes of Language Processor
  - Analysis Phase and Synthesis phase is not feasible due to
    1. Forward References
    2. Memory Requirement



Schematic of Language Processor

Fundamentals of Language Processing

- Forward Reference
  - It is a reference to the entity which precedes its definition in the program.

```
percent_profit := (profit * 100) / cost_price;
........
........
long profit;
```

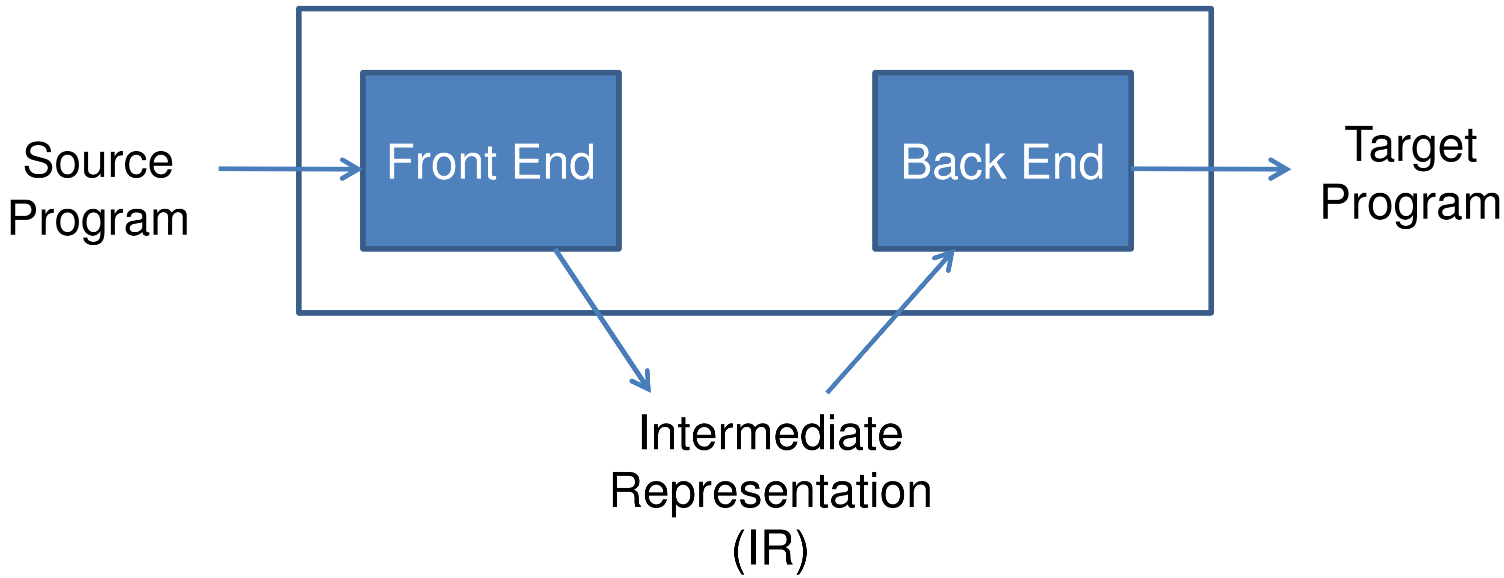
Fundamentals of Language Processing

- Language Processor Pass
  - Pass I : performs analysis of SP and notes relevant information.
  - Pass II: performs synthesis of target program.

Pass I analyses SP and generates IR which is given as input to Pass II to generate target code.

Fundamentals of Language Processing

- Intermediate Representation
  - An IR reflects the effect of some but not all, analysis and synthesis tasks performed during language processing.



Fundamentals of Language Processing

- Properties of Intermediate Representation(IR)
  1. Ease of use.
  2. Processing efficiency.
  3. Memory efficiency.

Fundamentals of Language Processing

- Toy Compiler
  - Front End: Performs lexical, syntax and semantic analysis of SP.
  - Each kind of analysis involves
    1. Determine validity of source stmt.
    2. Determine the content of source stmt.
    3. Construct a suitable representation.

Fundamentals of Language Processing

- Output of Front End
  1. **Tables of information**: The most important table is Symbol Table which contains information concerning all identifiers used in the source program
  2. **Intermediate code (IC):** IC is a sequence of IC units, represents meaning of one action in SP

Fundamentals of Language Processing

- Output of Front End Example

i: integer;
a,b: real;
a := b+i;

Symbol Table

No.	Symbol	Type	Length	Address
1	i	int		
2	a	real		
3	b	real		
4	i*	real		
5	temp	real		

Intermediate code

1. Convert (id, #1) to real, giving (id, #4)
2. Add (id, #4) to (id, #3) giving (id, #5)
3. Store (id, #5) in (Id, #2)

Fundamentals of Language Processing

- Toy Compiler
  - Lexical or Linear Analysis (Scanning)
    - Identifies lexical units in a source statement.
    - Classifies units into different classes e.g. id's, constants, reserved id's etc and enters them into different tables

Fundamentals of Language Processing

- Toy Compiler
  - Token contains
    - Class code and number in class

Code #no e.g. Id #10

- Example

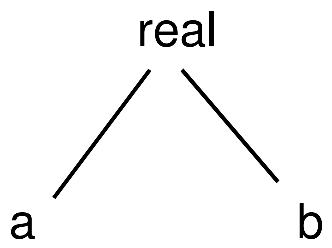
Statement a := b + i;

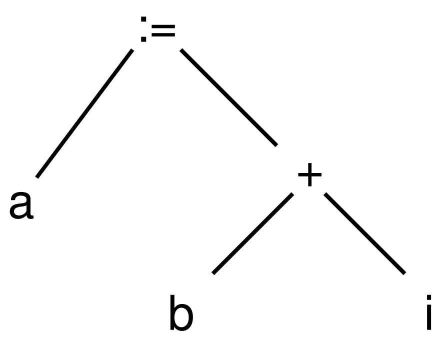
a	:=	b	+	i	;
Id #2	Op #5	Id #3	Op #3	Id #1	Op #10

Fundamentals of Language Processing

- Toy Compiler
  - Syntax or Hierarchical Analysis (Parsing)
    - Determines the statement class such as assignment statement, if stmt etc.

e.g.:- a , b : real; and a = b + I ;





Fundamentals of Language Processing

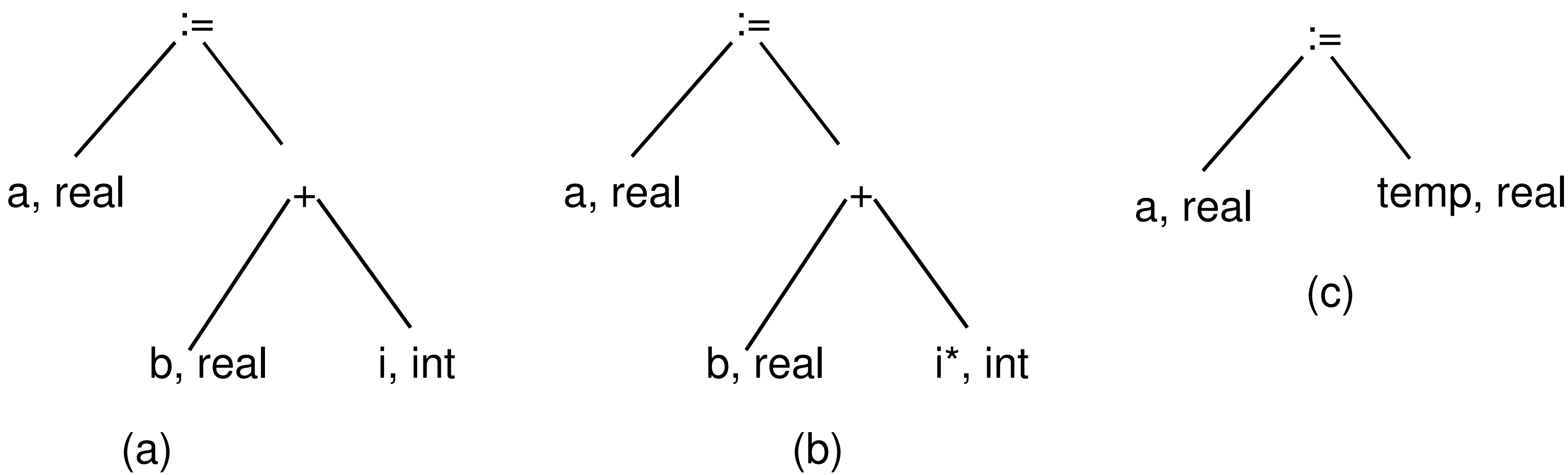
- Toy Compiler
  - Semantic Analysis
    - Determines the meaning of the SP
    - Results in addition of info such as type, length etc.
    - Determines the meaning of the subtree in IC and adds info to the IC tree.

Fundamentals of Language Processing

- Toy Compiler
  - Semantic Analysis
    - Stmt a := b +i; proceeds as
      1. Type is added to the IC tree
      2. Rules of assignment indicate expression on RHS should be evaluated first.
      3. Rules of addition indicate i should be converted before addition
         i. Convert i to real giving i*;
         ii. Add i* to b giving temp.
         iii. Store temp in a.

Fundamentals of Language Processing

- Toy Compiler
  - Semantic Analysis
    - Stmt a := b +i; proceeds as



Fundamentals of Language Processing

- Toy Compiler (Front End)

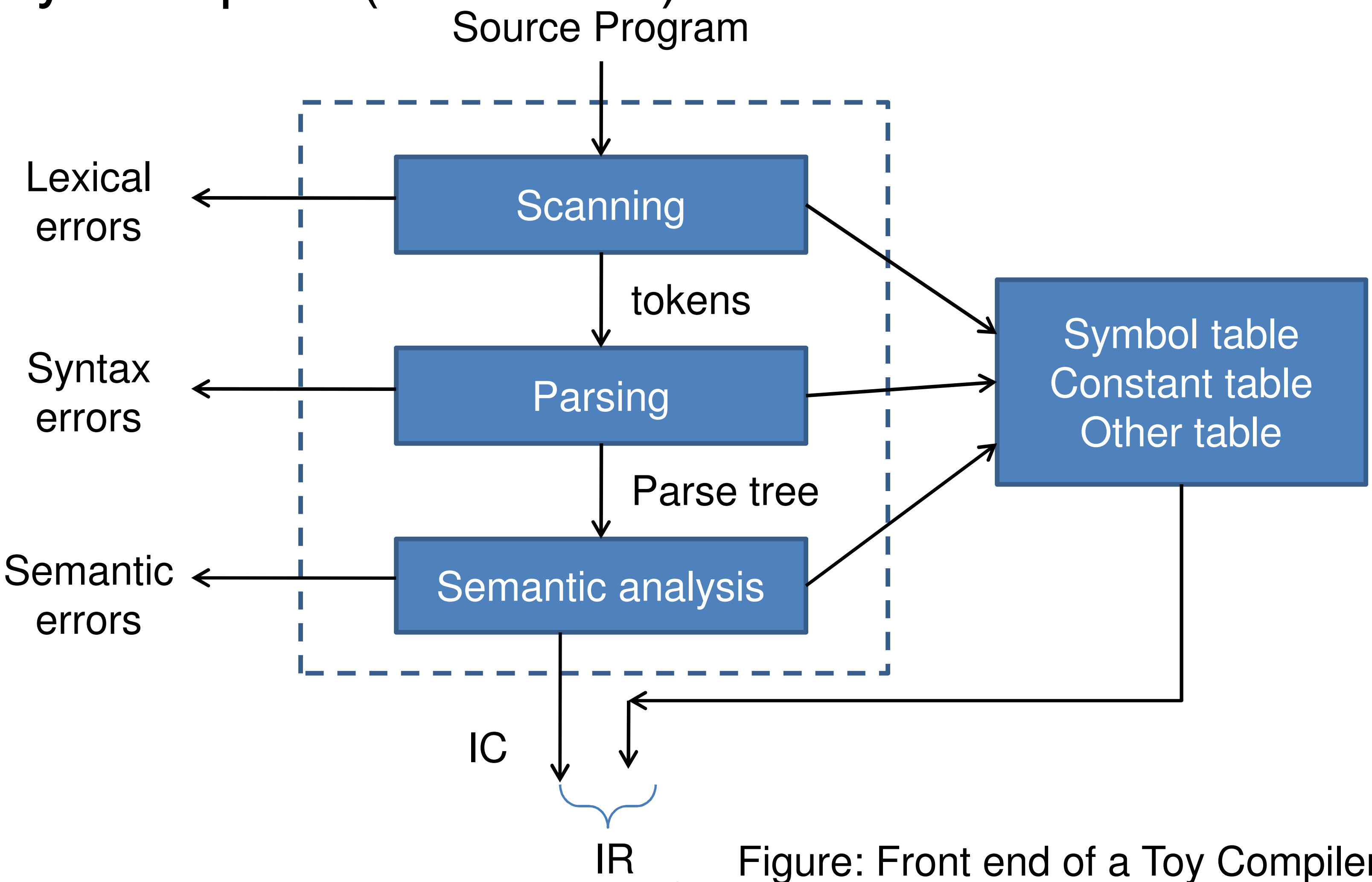


Figure: Front end of a Toy Compiler

Fundamentals of Language Processing

- Back End
  - Memory Allocation: Calculated form its type, length and dimensionality.

No.	Symbol	Type	Length	Address
1	i	int		2000
2	a	real		2001
3	b	real		2002

Fundamentals of Language Processing

- Code generation
  1. Determine places where results should be kept in registers/memory location.
  2. Determine which instruction should be used for type conversion operations.
  3. Determine which addressing modes should be used for accessing variables.

```
CONV_R      AREG, I
ADD_R       AREG, B
MOVEM       AREG, A
```

Figure: Target Code a:= b+ i

Fundamentals of Language Processing

- Toy Compiler (Back End)

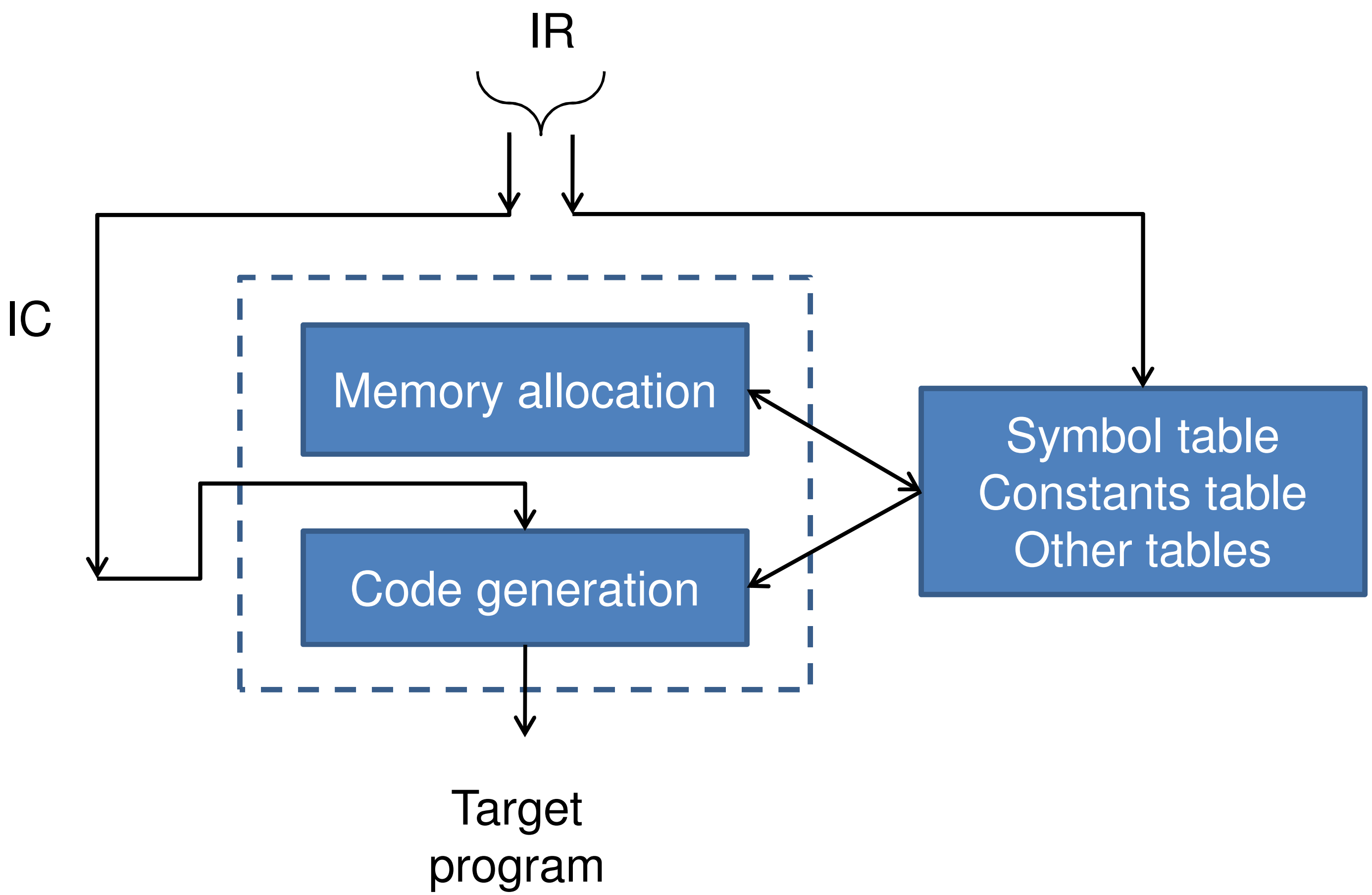


Figure: Back End of the toy compiler

Chapter 1: Language Processor

- Introduction
- Language Processing Activities
- Fundamentals of Language Processing
- Fundamentals of Language Specification
- Language Processing Development Tools

Fundamentals of Language Specification

- Programming Language grammars.

alphabets

↓

Words / Strings

↓

Sentences/ Statements

↓

Language

Terminal symbols, alphabet and strings

- Terminals symbol, alphabet and strings
  - The alphabet of L is represented by a Greek symbol Σ.
  - A symbol in the alphabet is known as a terminal symbol of language L.
  - Σ = {a , b , ….z, 0, 1,…. 9}
  - A string is a finite sequence of symbols.

    α= axy

Terminal symbols, alphabet and strings

- Nonterminal Symbols:
  - A nonterminal symbol is the name of a syntax category of a language.
  - E.g. noun, verb, etc.

Fundamentals of Language Specification

- Production
  - Also called as rewriting rule, is a rule of the grammar.

    A nonterminal symbol ::= String of Terminals and Nonterminals

e.g.:

<Noun Phrase> ::= <Article> <Noun>

<Article> ::= a| an | the

<Noun> ::= boy | apple

Fundamentals of Language Specification

- Grammar
  - A grammar G of a language LG is a quadruple (Σ, SNT, S, P) where
    - Σ is the alphabet
    - SNT is the set of NT's
    - S is the distinguished symbol
    - P is the set of productions

Fundamentals of Language Specification

- Derivation
  - Derivation is used to generate valid strings.
  - Let production P1 of grammar G be of the form

    $$P1 \quad : \quad A ::= \alpha$$

    And let β be such that $\beta = \gamma A \theta$ then replacement of A by α in string β constitute a derivation

    $$\beta = \gamma \alpha \theta$$

Fundamentals of Language Specification

- Derivation Example

<Sentence> :: = <Noun Phrase> <Verb Phrase>
<Noun Phrase> ::= <Article> <Noun>
<Verb Phrase> ::= <Verb> <Noun Phrase>
<Article> ::= a| an| the
<Noun> ::= boy | apple
<Verb> ::= ate

Fundamentals of Language Specification

- Derivation Example
  - Derivation for ‘the boy ate an apple’

    <Sentence>

    <Noun Phrase> <Verb Phrase>

    <Article> <Noun> <Verb Phrase>

    <Article> <Noun> <Verb> <Noun Phrase>

    the <Noun> <Verb> <Article> <Noun>

    the boy <Verb> <Article> <Noun>

    the boy ate <Article> <Noun>

    the boy ate an <Noun>

    the boy ate an apple

Fundamentals of Language Specification

- Reduction
  - Derivation is used to recognize valid string.
  - Let production P1 of grammar G be of the form

    P1 : $A ::= \alpha$

    And let β be such that $\beta = \gamma\ \alpha\ \theta$ then replacement of α by A in string β constitute a derivation

    $$\beta = \gamma\alpha\theta$$

Fundamentals of Language Specification

- Reduction Example

<Sentence> :: = <Noun Phrase> <Verb Phrase>
<Noun Phrase> ::= <Article> <Noun>
<Verb Phrase> ::= <Verb> <Noun Phrase>
<Article> ::= a| an| the
<Noun> ::= boy | apple
<Verb> ::= ate

Fundamentals of Language Specification

- Reduction Example
  - Reduction for 'the boy ate an apple'

    the boy ate an apple

    the boy ate an <Noun>

    the boy ate <Article> <Noun>

    the boy <Verb> <Article> <Noun>

    the <Noun> <Verb> <Article> <Noun>

    <Article> <Noun> <Verb> <Noun Phrase>

    <Article> <Noun> <Verb Phrase>

    <Noun Phrase> <Verb Phrase>

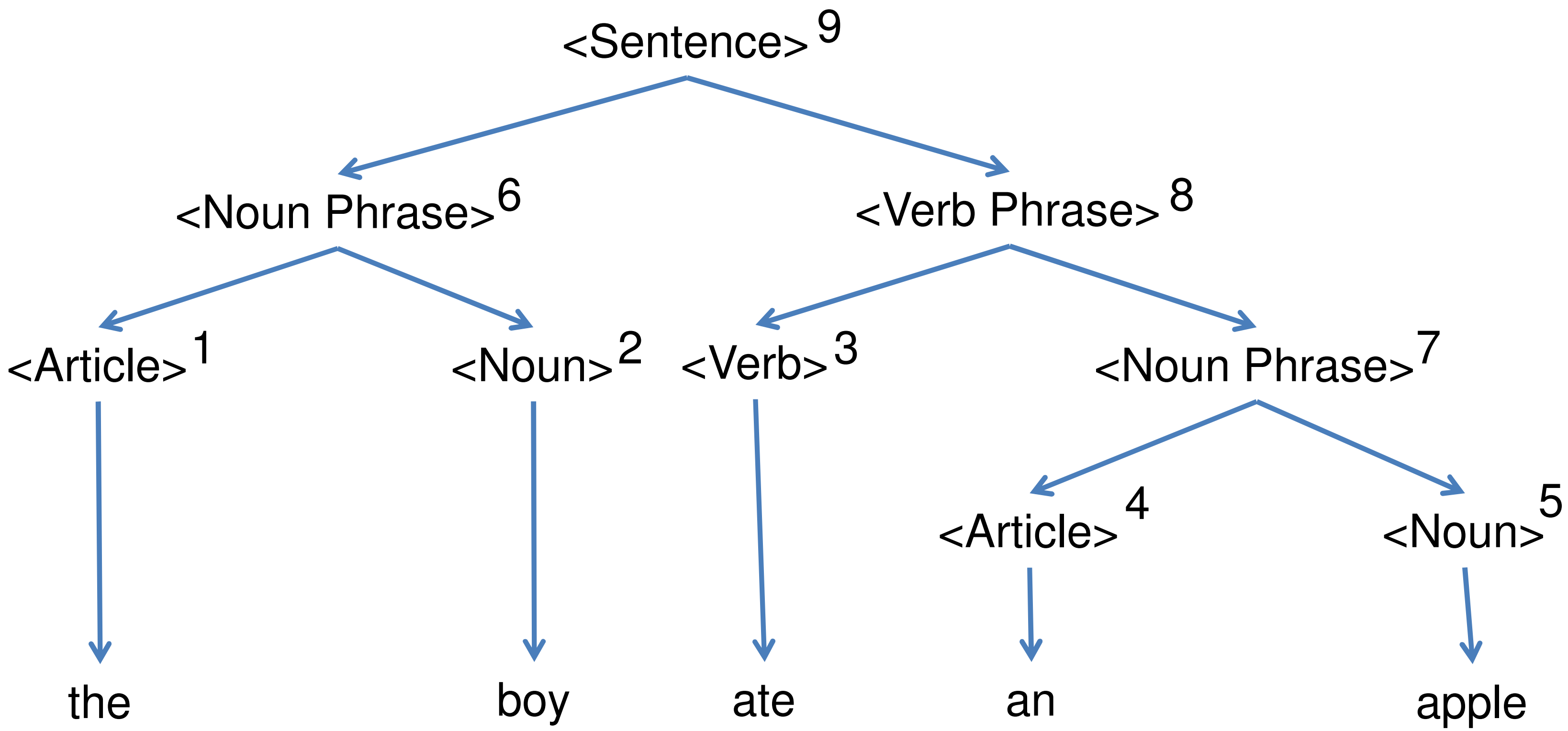
    <Sentence>

Fundamentals of Language Specification

- Parse Tree
  - A sequence of derivation or reduction reveals the syntactic structure of a string with respect to grammar G.
  - We depict the syntactic structure in the form of parse tree.

Fundamentals of Language Specification

- Parse Tree Example
  - Parse Tree for 'the boy ate an apple'



Fundamentals of Language Specification

- Recursive Specification
  - The RHS alternative employing recursion is called a recursive rule.
  - Recursive rules are classified into left recursive and right recursive rules.
  - Indirect recursion occurs when two or more nonterminals are defined in terms of one another.

Fundamentals of Language Specification

- Recursive Specification Example

<exp> ::= <exp> + <term> | <term>

<term> ::= <term> * <factor> | <factor>

<factor> ::= <factor> ↑ <primary> | <primary>

<primary> ::= <id> | <const> | (<exp>)

<id> ::= <letter> | <id>[<letter | digit>]

<const> ::= [+ | -] <digit> | <const> <digit>

<letter> ::= a | b | …. | z

<digit> ::= 0 | 1|….| 9

Fundamentals of Language Specification

- Recursive Specification Example
  - The rule for <id> and <const> are equivalent to the rules

    <id> ::= <letter> | <id> <letter> | <id> <digit>]

    <const> ::= + <digit> | - <digit> | <const> <digit>

  - Controlled recurrence may be specified for <id> as follows

    <id> ::= <letter> $\{$<letter> | <digit>$\}_0^{15}$

Fundamentals of Language Specification

- Classification of Grammars
  - Type-0 grammar or Unstructured or Phrase Structured grammar

    $\alpha ::= \beta$

  - Type-1 grammar or Context Sensitive grammar

    $\alpha A\beta ::= \alpha \pi\beta$

  - Type-3 grammar or Context Free grammar

    $A ::= \pi$

Fundamentals of Language Specification

- Classification of Grammars
  - Type-4 grammar or Linear Grammar

    A ::= tB | t or

    A ::= Bt | t

  - Operator Grammar: a grammar none of whose production contain two or more consecutive nonterminals in any RHS alternative.

    e.g. E ::= E + E | E – E | E * E | E / E | id

Fundamentals of Language Specification

- Ambiguity in grammar specification
  - Ambiguity implies the possibility of different interpretation of a source string.

    E.g. E ::= E + E | E * E | (E) | id
  - An ambiguous grammar should be rewritten to eliminate ambiguity

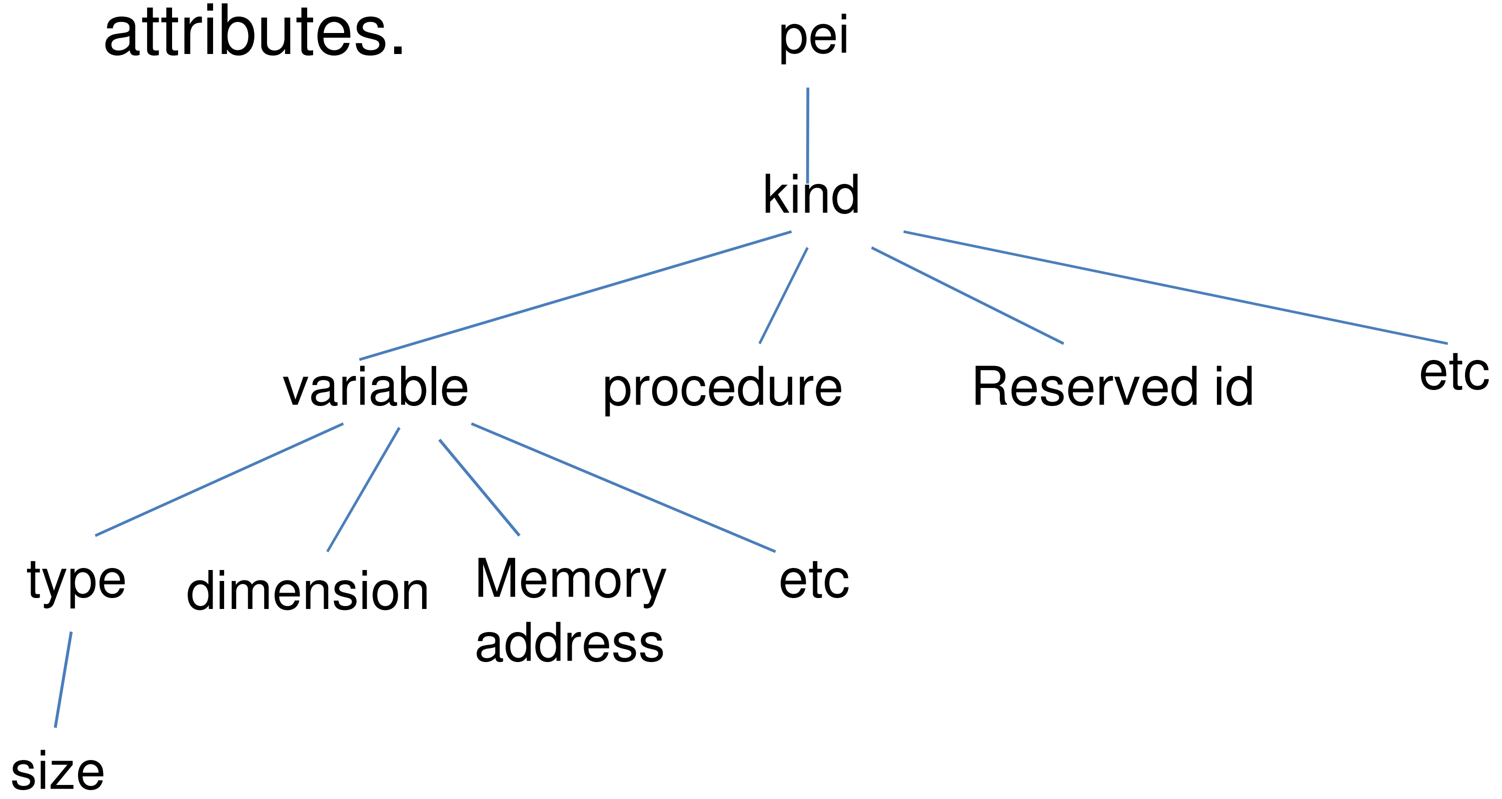
    E ::= E + E | E * E | (E) | id

    After eliminating ambiguity from grammar

    E ::= E + T | T
    T ::= T * F | F
    F ::= (E) | id

Fundamentals of Language Specification

- Binding and Binding Times
  - Program entity pei in program P has a some attributes.



Fundamentals of Language Specification

- Binding and Binding Times
  - A binding is an association of an attribute of a program entity with a value.
  - Binding time is the time at which binding is performed.

int a;

Fundamentals of Language Specification

- Binding and Binding Times
  1. Language definition time of language L
  2. Language implementation time of language L
  3. Compilation time of program P
  4. Execution init time of procedure proc
  5. Execution time of procedure proc

Fundamentals of Language Specification

```
program bindings(input , output);
     var
          i : integer;
          a,b : real;
    procedure proc (x: real; j: integer);
          var
                    info : array[1…10,1…5] of integer;
                    p :  ↑ integer;
          begin
                    new (p);
          end;
begin
    proc(a,i);
end
```

Fundamentals of Language Specification

- Binding and Binding Times
  - Binding of keywords with its meaning.

    e.g.: **program**, **procedure**, **begin** and **end**.
  - Size of type int is bind to n bytes (Language implementation time)
  - Binding of var to its type. (Compilation time)
  - Memory addresses are allocated to the variables. (Execution init time)
  - Values are binded to memory address. (Execution time of procedure)

Fundamentals of Language Specification

- Importance of binding times
  - The binding time of an attribute of a program entity determines the manner in which a language processor can handle the use of the entity.

    e.g. procedure pl1_proc (x, j, info_size, columns)

    ```
            declare x float;
            declare (j, info_size, columns) fixed;
            declare pl1_info (1: info_size, 1: columns) fixed;
            ….
    end pl1_proc
    ```

  - An early binding provides greater execution efficiency whereas a late binding provides greater flexibility in the writing of a program.

Fundamentals of Language Specification

- Static and dynamic binding
  - Static Binding: Binding is performed before the execution of a program begins.
  - Dynamic Binding:Binding is performed after the execution of a program has begun.

Chapter 1: Language Processor

- Introduction
- Language Processing Activities
- Fundamentals of Language Processing
- Fundamentals of Language Specification
- Language Processing Development Tools

Language Processor Development Tools (LPDT)

- LPDT requires two inputs:
  1. Specification of a grammar of language L
  2. Specification of semantic actions

- Two LPDT's which are widely used
  1. LEX (Lexical analyzer)
  2. YACC (Parser Generator)

Language Processor Development Tools (LPDT)

- LPDT contains translation rules of form:

  <string specification> { <semantic action>}

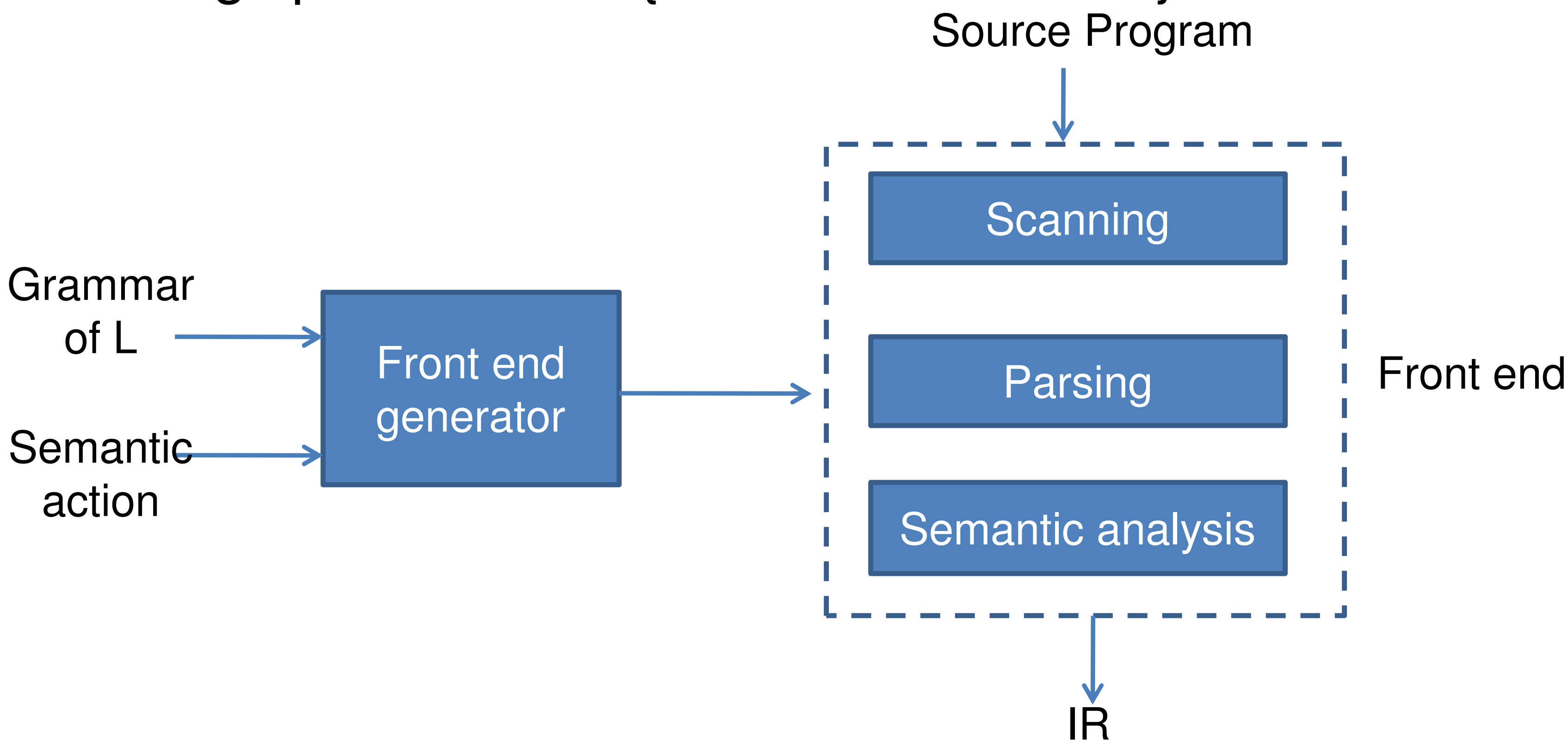


Figure: Language Processor Development Tool

Language Processor Development Tools (LPDT)

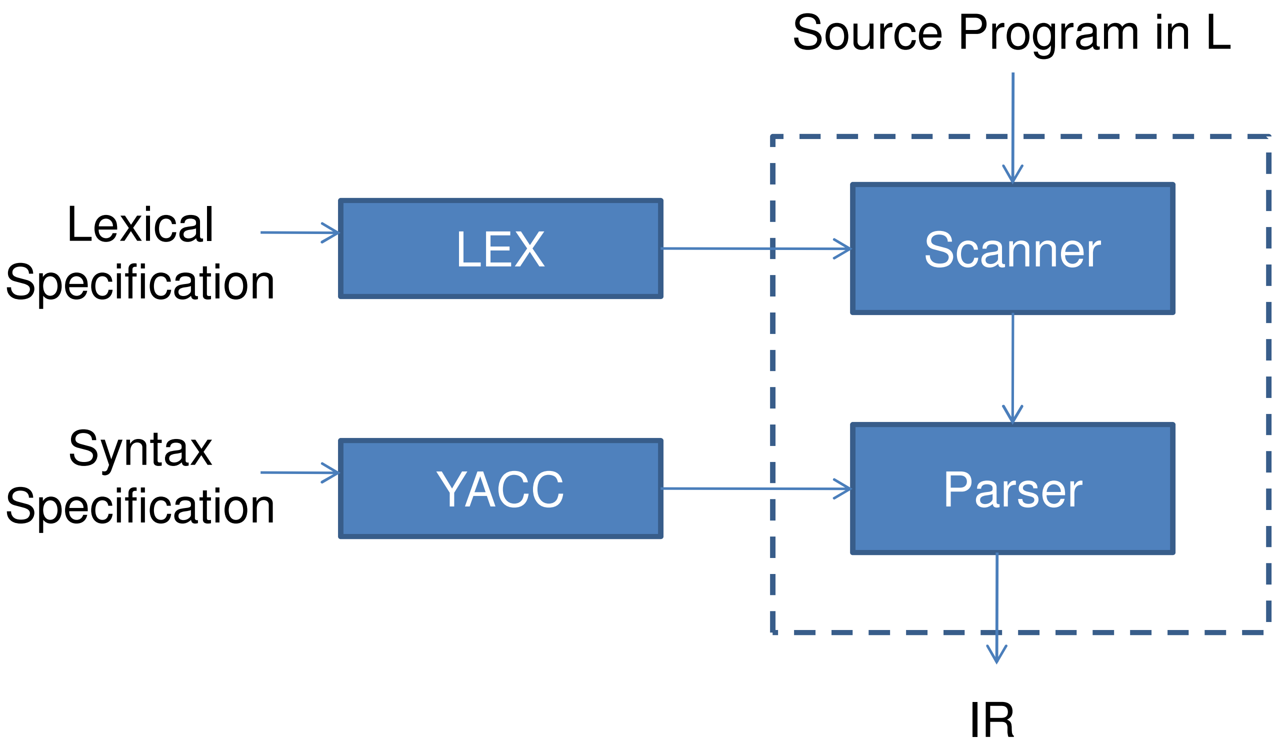


Figure: Using LEX and YACC

Language Processor Development Tools (LPDT)

- LEX
  - LEX consists of two components
    - Specification of strings such as id's, constants etc.
    - Specification of semantic action

Language Processor Development Tools (LPDT)

- LEX

```
%{
        //Symbols definition;        1st section
%}

%%
        //translation rules;         2nd section
        Specification Action
%%

        //Routines;                  3rd section
```

Language Processor Development Tools (LPDT)

- LEX Example

```
%{
        letter          [A-Za-z]
        digit           [0-9]
}%
%%
    begin                           {return(BEGIN);}
    end                             {return(END);}
    {letter} ( {letter}|{digit})*   {yylval = enter_id(); return(ID);}
    {digit}+                        {yylval=enter_num(); return(NUM);}
%%
enter_id() { /*enters id in symbol table*/ }
enter_num() {/* enters number in constabts table */}
```

Language Processor Development Tools (LPDT)

- YACC
  - String specification resembles grammar production.
  - YACC performs reductions according to the grammar.

Language Processor Development Tools (LPDT)

- YACC Example

```
%%
    E    :    E+T    {$$ = gencode('+', $1, $3);}
         |    T      {$$ = $1;}
    T    :    T*V    {$$ = gencode('*', $1, $3);}
         |    V      {$$ = $1;}
    V    :    id     {$$ = gendesc($1);}
%%
gencode (operator, operand_1, operand_2){  }
gendesc(symbol) {   }
```