Date : 1400 / 11 / 16

Books: Language Implementation Pattern, by Terence Parr

Compiler Principles, Techniques & tools, by Alfred V. Aho

Phases of a compiler: (Page 7 Aho's book)

1. Lexical Analysis — تحلیل لغوی

converts a sequence of characters into words, or tokens

کاراکتر کاراکتر می‌خواند. (کلمات را درمتن یک برنامه تشکیل دهد)

2. Syntax Analysis:

converts a sequence of tokens into a parse tree

استخراج یک جمله شرطی داریم. از ترتیب‌ها چک می‌کند) (گرامر باید درست باشد ولی معنی مهم نیست.)

3. Semantic Analysis:
(type checking)

Manipulates parse tree to verify symbol and type information

معنی را چک می‌کند. درست کاری کند یا خیر. معنی باید درست باشد. (int ≠ float : ex)

4. Intermediate Code Generation:

converts parse tree into a sequence of intermediate code instructions

تبدیل کد به یک level که نزدیک به زبان ماشین است.

5. Optimization:

Manipulates intermediate code to produce a more efficient program.

6. Final Code Generation:

Translate intermediate code into final (machine / assembly) code

ex: position = initial + rate * 60

↓

[Lexical Analyzer] — نوع کلمات را تشخیص می‌دهد

↓

<id, 1> <=> <id, 2> <+> <id, 3> <*> <60>

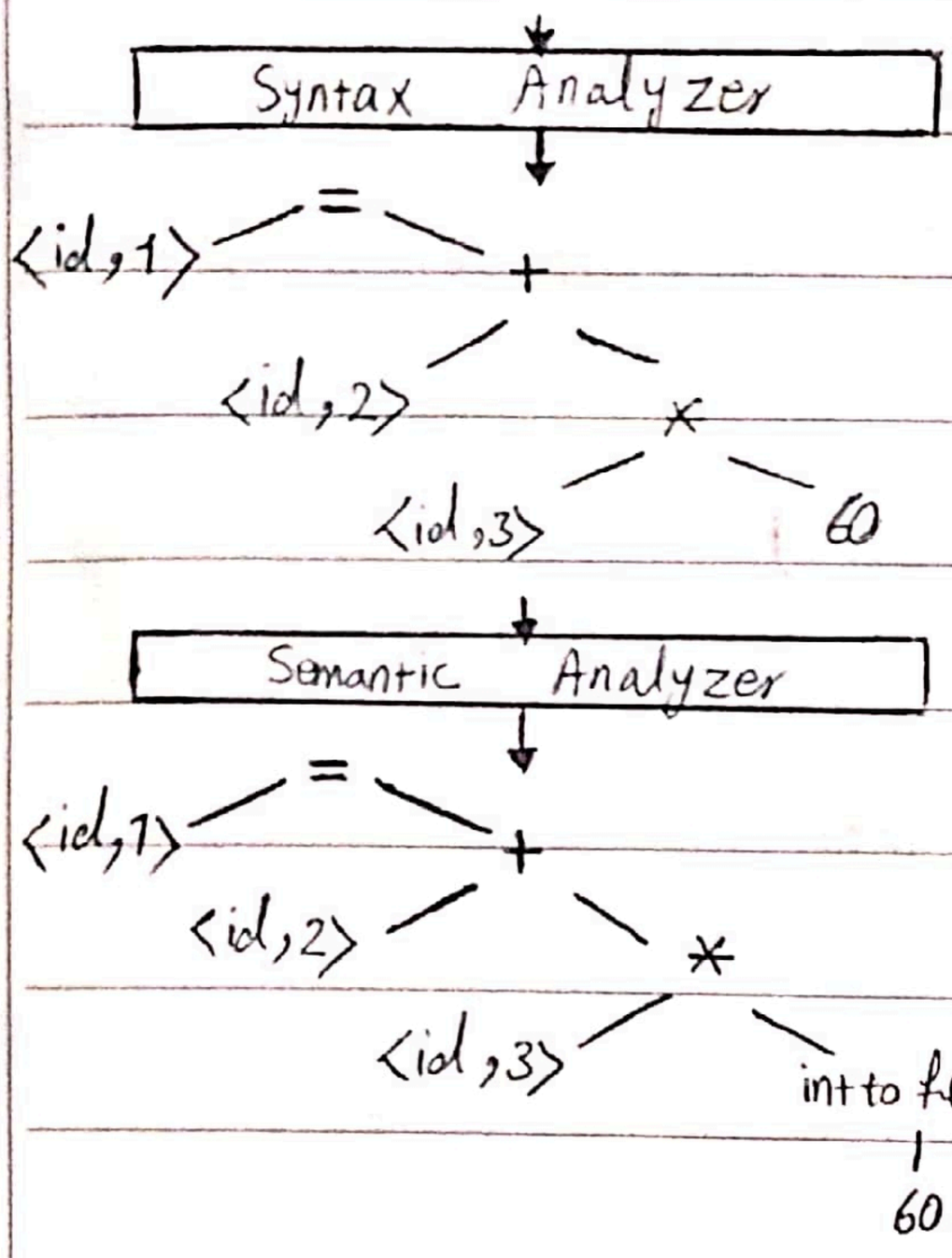
⋮

Date : 1400 / 11 / 16

Introduction

Syntax Analyzer

درخت تجزیه تبدیل می‌کند



Intermediate Code Generator

$t_1 = int\ to\ float\ (60)$

$t_2 = id_3 * t_1$

$t_3 = id_2 + t_2$

$id_1 = t_3$

Code Optimizer

کمینه کردن

$t_1 = id_3 * 60.0$

$id_1 = id_2 + t_1$

Code Generator

تبدیل به کد اسمبلی

```
LDF  R2, id3
MULF R2, R2, #60.0
LDF  R1, id2
ADDF R1, R1, R2
STF  id1, R1
```

ex: position = initial + rate * 60 (30 chars)

1. Lexical Analysis → The characters are transformed into a sequence of 7 tokens.

Token 1. name = "position"

Token 2. name = "="

Token 3. name = "initial"

Token 4. name = "+"

Token 5. name = "rate"

Token 6. name = "*"

Token 7. name = "60"

2. Syntax Analysis → build a tree of height 4

G1:

assignmentSt::= identifier = expression

expression ::= expression + term | expression - term | term

term::= term * factor | term / factor | factor

factor::= identifier | number | (expression)

3. Semantic Analysis → transform the tree into one of height 5, that includes a type conversion necessary for real addition on an integer operand.

4. Intermediate code generation → uses a simple traversal algorithm to linearize the tree back into a sequence of machine-independent three-address-code instructions.

```
t1 = inttoreal (60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

5. Optimization → reduce the four instructions to two machine-independant instructions.

```
t1 = id3 * 60.0
id1 = id2 + t1
```

7. Final code generation → implement these two instructions using 5 machines instructions, in which the actual registers & addressing modes of the CPU are utilized.

```
MOVF    id3 , R2
MULF    #60.0 , R2
MOVF    id2 , R1
ADDF    R2 , R1
MOVF    R1 , id1
```

Date : 1400 / 11 / 23

Intraduction

Score Allocation :

پروژه پایانی	x 7	7
میان ترم	x 7	3
پایان ترم	x 7	4
تمرین ها	x 4	4
کوییزهای کلاسی	x 4	2

Compiler construction tools

1. Parser Generators parser برای ما به آن می دهیم و برای ما context free grammer را پیاده می کند.

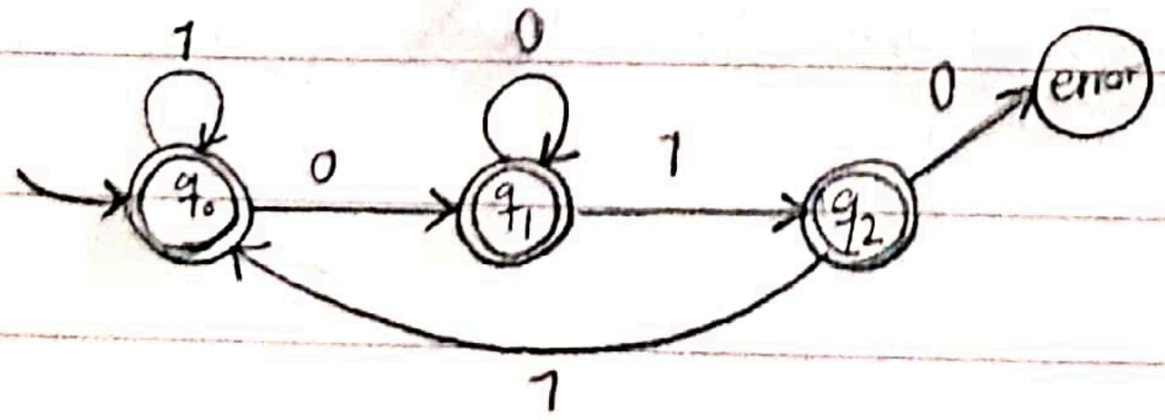
2. Scanner Generators تحلیلگر لغوی می سازد.

lexical Analysis

tokenizer = تحلیلگر لغوی

1400 / 11 / 25 Lexical Analysis.

Ex: $L(M) = \{w \mid$ does not include 010$\}$, w can be combination of 0's and 1's.



<> → مخالف

Slide 24 :

Nextword = کلمه‌ای که پیش‌بینی می‌شود

Length = طول کلمه‌ای که می‌خوانیم

Slide 23 :

PC : Program Counter

BP : Base Pointer

SP : Stac Pointer

Temporaris : یک سری پارامتر موقت که خود مترجم می‌سازد

Slide 42

fragment : چیزی که جای دیگر استفاده شده

1400 / 11 / 30

Compilation unit → شروع برنامه → Start

? → می‌تونه بیاد، می‌تونه نیاد

. → هرکاراکتری

terminal → داخل گرامر

Channel (Hidden) → white space ها را برگشت این یعنی کامپایلر جزء کد متن یا خروجیت rewrite نمی‌کند، white space ها را برمی‌دارد.

حالت عادی → Skip

WS : White Space

Date : 1400 / 11 / 30

Comment : /*

Channel (HIDDEN) → توکن نمی‌شود

Start symbal : English statement program java (compilation unit or start)

Non_terminal symbols : داخل " ' نمی‌زاریم ; مثل جمله خبری، امری، پرسشی ...
↳ terminal : داخل " " حسن، حسین

Non-terminal symbals : Statement : ... ;

ex: Id : LETTER (LETTER | DIGIT)* ;
a b123 def123

expression : در آن حاصل جمع و تفضیل است .
term : در آن حاصل جمع و تفضیل نیست .

1400 / 12 / 2

نکته خانمی مهسم : علامت سوال هم قبول می‌کنه . :)
underline هم قبول می‌کنه :))

Date : 1400 / 12 / 7

Languages & grammers

Backus-Naur Form (BNF) notation

- a formal notation for encoding grammers intended for human consumption

یک زبان برای توصیف grammer زبان است.

- structure : name ::= expansion

or

name → expansion → name : non-terminal

- antlr : name : expansion ;

symbols ⟨ terminal symbols — ex: 'if', 'then', '(', ...
non-terminal symbols — ex: if_statement, ...
start symbols — ex: c-program
↳ compilation unit

non-terminal symbols : آن هایی که با یک expansion جواب دارند

Extended BNF (EBNF) notation

1. Grouping operator : (...) ex: ("+" | "-")
2. Optional operator: [...] ex: ["+" | "-"]
3. Repetition operator (zero or more) : {...} ex: {character}

BNF example:

⟨expr⟩ ::= ⟨expr⟩ "+" ⟨term⟩
| ⟨expr⟩ "-" ⟨term⟩ | ⟨term⟩

⟨term⟩ ::= ⟨term⟩ "*" ⟨factor⟩
| ⟨term⟩ "/" ⟨factor⟩

⟨factor⟩ ::= "(" ⟨expr⟩ ")"
| number | identifier

In ANTLR : expr :

expr ('+' | '-') term

| term ;

term :

term ('*' | '/') factor

| factor ;

factor :

'(' expr ')' | Signed | String | Identifier

;

String : character + ;

EBNF example : expr ::= <expr> ("+" | "-") <term>

|<term>

<term> ::= <term> ("*" | "/") <factor>

| <factor>

<factor> ::= "(" <expr> ")"

| <Signed> | <string> | identifier

<Signed> ::= ["+" | "-"] number

<string> ::= """ character {character} """

How to write parser ? (slide 15, languages & grammers)

تجزیه بالا به پایین : از سمت چپ آغاز به ترم های پایانی روی علم داده شده خاتمه می یابد.
(Top-down parsing)
Left most derivation begining from the start Symbol

تجزیه پایین به بالا : در مقابل تجزیه بالا به پایین قرار دارد. از ترم های پایانی روی علم داده شده شروع شده به سمت ترم می رسد.
(Bottom-Up Parsing)
Right most derivation

Eg. $S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

Input: abbcde

A
↑
abbcde

aAbcde

aAde

aABe

S

* مراحل اشتقاق، برعکس مراحل parse می باشد.

Bottom_Up parsing → shift_reduce parsing, Left to Right (LR) family or Precedence parsing

اگر در گرامر برای یک ترم میانی، قاعده خودبازگشتی به سمت چپ و قاعده خودبازگشتی به سمت راست داشته باشیم، گرامر ٪100 مبهم می باشد.

IfSt → if Condition then Statement ElsePart → مبهم

ElsePart → else statement | λ

↳ ex. There are two parse trees with two different semantics for the statement:

if a>5 then if a<7 then writeln('a=6') else writeln('a>=7')

- Arithmetic operators : +, -, *, / are left associative.
- The power operator ^ is right associative.

1400 / 2 / 21 Languages & Grammer

Operator association

```
expr: <assoc=right> expr '^' expr
    | INT
    ;
```

از چپ به راست، تبدیل به از راست به چپ می کند.

Top-down parsers

از جمله فارسی به گرامر می رسیم.

نقش parsing : با در دست داشتن یک look a head تشخیص می دهم که term بعدی، چه میانی، چه غیر میانی چه می آید. در واقع در روش parsing، همیشه انتظار یک چیزی را داریم.

Date : 1400 / 2 / 21

Top-down parser

ex: S ::= aBC | bBd ;

B ::= bB | d

C ::= BC | a

S: abbdb

(1) Compute the First set for each non-terminal symbol

(2) Build a top-down parsing table

(3)

First(S) = {a, b} (1)

First(B) = {b, d}

First(C) = First(B) + {a} = {b, d, a}

(2)

	a	b	d
S	aBC	bBd	—
B	—	bB	d
C	a	BC	BC

Parsing stack	Source file	Action
S	abbdb	aBC
CBa	abbdb	delete
CB	bbdb	bB
CBb	bbdb	delete
CB	bdb	bB
CBb	bdb	delete
CB	db	d
Cd	db	delete
C	b	BC
CB	b	bB

← جمله خروجی ایرنیست

Date : 1400 / 2 / 21

Top-down parser

S ::= bSB | AC First(S) = First(A) + b = {a, b, d}

A ::= aB | dC First(A) = {a, d}

B ::= bB | d First(B) = {b, d}

C ::= cB | a First(C) = {c, a}

S: abbda

	a	b	c	d
S	AC	bSB	—	AC
A	aB	—	—	dC
B	—	bB	—	d
C	a	—	cB	—

Parsing stack	Source file	Action
S	abbda	AC
CA	abbda	aB
CBa	abbda	delete
CB	bbda	bB
CBb	bbda	delete
CB	bda	bB
CBb	bda	delete
CB	da	d
Cd	da	delete
C	a	a
a	a	delete

Date : 1400 / 12 / 23

Top-down parser

هر گرامر غیر مبهم به هر شکلی که باشد، می‌توان آن را به فرمت LL1 تبدیل کرد.

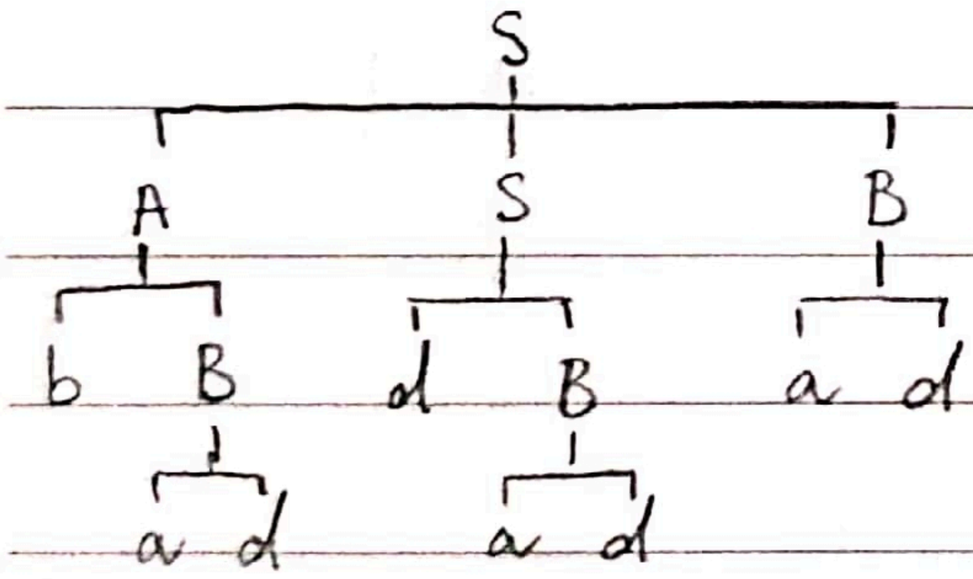
ex: S ::= ASB | dB → First(S) = First(A) + {d} = {a, b, d}

A ::= aAb | bB → First(A) = {a, b}

B ::= bBd | ad → First(B) = {b, a}

baddadadad

	a	b	d
S	ASB	ASB	dB
A	aAb	bB	—
B	ad	bBd	—



Parsing stack	Source file	Action
S	baddadad	ASB
BSA	baddadad	bB
BSBb	baddadad	delete
BSB	addadad	ad
BSda	addadad	delete
BSd	ddadad	delete
BS	dadad	dB
BBd	dadad	delete
BB	adad	ad
Bda	adad	delete
B	ad	ad
da	ad	delete

Date : 1400 / 12 / 23

Top-down parser

Grammer LL1 : اشتراک مجموعه First هیچ دو ترمینال از این Term میانی ها تهی نباشد. یعنی اشتراک مجموعه First ترمینال ها دو به دو تهی باشد.

$$\forall A \to \beta_1 | \beta_2 | \dots | \beta_n \in G$$

$$\Rightarrow \cap (First(B_i), First(B_j)) = \emptyset \;,\; \forall (i,j) \in [1 \dots n] \wedge i \neq j$$

where

$A \to a\beta \Rightarrow First(A) = \{a\}$, $a \in$ Terminal Symbols & β is any string

$A \to B\beta \Rightarrow First(A) \supseteq First(B)$, $B \in$ Nonterminal Symbols & ...

$A \to \beta_1 | \beta_2 | \dots | \beta_n$

$\Rightarrow First(A) = First(\beta_1) \cup First(\beta_2) \cup \dots \cup First(\beta_n)$

ex: $G1 : A \to aB | aA | Db$ → This grammer is not LL1

$G2 : A \to Bb | dB$ → $First(A) = \{d, b\}$

$B \to dB | b$ → $First(B) = \{d, b\}$

$First(Bb) \cap First(dB) = \{b, d\} \cap \{d\} \neq \emptyset$