MINERVA SCHOOLS AT K.G.I.
Assignment 3
Quang Tran
CS142 Fall 2019

Part 1. An English Language Limited Parser

a. Formal definition of your grammar

$G = (V,\ \sum,\ R,\ S)$ *where* :

1. V is the set of variables and

V={S, $< Prep - Phrase >, < Main - S >,\ < Verb - Phrase >, < Noun - Phrase >,$
$< Cmplx - Noun >,\ < Cmplx - Verb >, < Prep >,\ < Single - Cmplx - Noun >,$
$< Multip - Cmplx - Noun >,\ < a - Article >, < a - Adj >, < Noun >, < an - Article >,$
$< an - Adj >, < THE >, < Adj >, < Pl - Noun >,$ <a-Noun>, <an-Noun>,
$< Other - Cmplx - Noun >$, <Verb>,<Prep> }

2. $\sum$ is the set of terminals and $\sum = \Lambda \cup \{\#,\ ,\}$ where Λ is the set of English alphabet. The # sign represents the white space between the words.
3. R is a finite set of rules
4. S is the start variable

The rules are given below:

$S \rightarrow\ < Prep - Phrase >, \# < Main - S >\ |\ < Main - S > | < Prep - Phrase >< Verb - Phrase >< Noun - Phrase >$

$< Main - S >\ \rightarrow\ < Noun - Phrase >< Verb - Phrase >$

$< Noun - Phrase >\ \rightarrow\ < Cmplx - Noun >\ |\ < Cmplx - Noun >< Prep - Phrase >$

$< Verb - Phrase >\ \rightarrow\ < Cmplx - Verb >\ |\ < Cmplx - Verb >< Prep - Phrase >$

$< Prep - Phrase >\ \rightarrow\ < Prep >< Cmplx - Noun >$

$< Cmplx - Noun >\ \rightarrow\ < Single - Cmplx - Noun >\ |\ < Multip - Cmplx - Noun >$

$< Single - Cmplx - Noun >\ \rightarrow\ < a - Article >< a - Adj >< Noun >\ |\ < an - Article >< an - Adj >< Noun > |$
$| < THE >< Adj >< Pl - Noun > |\ < a - Article >< a - Noun >\ |$
$|\ < an - Article >< an - Noun >$

$< Noun >\ \rightarrow\ < a - Noun >\ |\ < an - Noun >$

$< Adj >\ \rightarrow\ < a - Adj >\ |\ < an - Adj >$

$< a - Adj > \rightarrow\ \varepsilon \mid great\# \mid nice\# \mid tall\#$

$< an - Adj >\ \rightarrow\ \varepsilon\ |\ old\# \mid odd\#$

$< a - Noun > \rightarrow banana\# \mid river\# \mid cat\# \mid car\# \mid home\#$
$< an - Noun > \rightarrow egg\# \mid eye\# \mid actor\# \mid app\# \mid ink\#$
$< Pl - Noun > \rightarrow bananas\# \mid cats\#$

$< a - Article > \rightarrow a\# \mid the\#$
$< an - Article > \rightarrow an\# \mid the\#$
$< THE > \rightarrow the\# \mid \varepsilon$
$< Multip - Cmplx - Noun > \rightarrow < Single - Cmplx - Noun >< Other - Cmplx - Noun >$
$< Other - Cmplx - Noun > \rightarrow \varepsilon \mid ,< Multip - Cmplx - Noun >$
$< Cmplx - Verb > \rightarrow < Verb > \mid < Verb >< Noun - Phrase >$
$< Verb > \rightarrow eats\# \mid goes\# \mid sings\# \mid acts\# \mid plays\#$
$< Prep > \rightarrow in\# \mid on\# \mid at\#$

Properties of the grammar:

1. For the first rule, $< Prep - Phrase >,< Main - S >$ means some sentence with the prepositional phrase at the beginning (e.g., “In the morning, I drank milk”); $< Main - S >$ means some sentence with the simple subject-verb phrase (e.g., “I drank milk"); $< Prep - Phrase >< Verb - Phrase >< Noun - Phrase >$ accounts for a more complex structure of an English sentence where prepositional phrase acts as the subject of the sentence (“Along the street comes a man").
2. The grammar accounts for appropriate article given the noun or noun phrase (e.g., “an apple" but not “a apple")
3. The grammar also allows listing multiple nouns, separated by commas (e.g., “the cat, the dog, the bird take class”)
4. Unfortunately the grammar only allows verbs corresponding to third person regardless of whether the subject is plural or single (e.g., “I plays the piano” but not “I play the piano”)
5. The grammar can be enriched by adding more vocabularies. This is just for demonstration with a few example words.

b. Is your grammar ambiguous?

Yes, it is. To show that the grammar ambiguous, we need to find *one* string that is generated ambiguously by the grammar. And in order to show this, we show that the string is derived in several ways in the grammar.
One such string is “the actor eats the bananas at the river”

$S \rightarrow$ $< Main - S >$ $\rightarrow$ $< Noun - Phrase >< Verb - Phrase >$

$\rightarrow$ $< Cmplx - Noun >< Verb - Phrase >$

$\rightarrow < Single - Cmplx - Noun >< Verb - Phrase >$

$\rightarrow$ $< an - Article >< an - Noun >< Verb - Phrase >$

$\rightarrow$ *the* $< an - Noun >< Verb - Phrase >$

$\rightarrow$ *the actor* $< Verb - Phrase >$

$\rightarrow$ *the actor* $< Cmplx - Verb >$

$\rightarrow$ *the actor* $< Verb >< Noun - Phrase >$

$\rightarrow$ *the actor eats* $< Noun - Phrase >$

$\rightarrow$ *the actor eats* $< Cmplx - Noun >< Prep - Phrase >$

$\rightarrow$ *the actor eats* $< Single - Cmplx - Noun >< Prep - Phrase >$

$\rightarrow$ *the actor eats* $< THE >< Adj >< Pl - Noun >< Prep - Phrase >$

$\rightarrow$ *the actor eats the* $< Adj >< Pl - Noun >< Prep - Phrase >$

$\rightarrow$ *the actor eats the* $< Pl - Noun >< Prep - Phrase >$

$\rightarrow$ *the actor eats the bananas* $< Prep - Phrase >$

$\rightarrow$ *the actor eats the bananas* $< Prep >< Cmplx - Noun >$

$\rightarrow$ *the actor eats the bananas at* $< Cmplx - Noun >$

$\rightarrow$ *the actor eats the bananas at* $< Single - Cmplx - Noun >$

$\rightarrow$ *the actor eats the bananas at* $< a - Article >< a - Noun >$

$\rightarrow$ *the actor eats the bananas at the* $< a - Noun >$

$\rightarrow$ *the actor eats the bananas at the river*

$S \rightarrow$ $< Main - S >$ $\rightarrow$ $< Noun - Phrase >< Verb - Phrase >$

$\rightarrow$ $< Cmplx - Noun >< Verb - Phrase >$

$\rightarrow < Single - Cmplx - Noun >< Verb - Phrase >$

$\rightarrow$ $< an - Article >< an - Noun >< Verb - Phrase >$

$\rightarrow$ *the* $< an - Noun >< Verb - Phrase >$

$\rightarrow$ *the actor* $< Verb - Phrase >$

$\rightarrow$ *the actor* $< Cmplx - Verb >< Prep - Phrase >$

$\rightarrow$ *the actor* $< Verb >< Noun - Phrase >< Prep - Phrase >$

$\rightarrow$ *the actor eats* $< Noun - Phrase >< Prep - Phrase >$

$\rightarrow$ *the actor eats* $< Cmplx - Noun >< Prep - Phrase >$

$\rightarrow$ *the actor eats* $< Single - Cmplx - Noun >< Prep - Phrase >$

$\rightarrow$ *the actor eats* $< THE >< Adj >< Pl - Noun >< Prep - Phrase >$

$\rightarrow$ *the actor eats the* $< Adj >< Pl - Noun >< Prep - Phrase >$

$\rightarrow$ *the actor eats the* $< Pl - Noun >< Prep - Phrase >$

$\rightarrow$ *the actor eats the bananas* $< Prep - Phrase >$

$\rightarrow$ *the actor eats the bananas* $< Prep >< Cmplx - Noun >$

$\rightarrow$ *the actor eats the bananas at* $< Cmplx - Noun >$

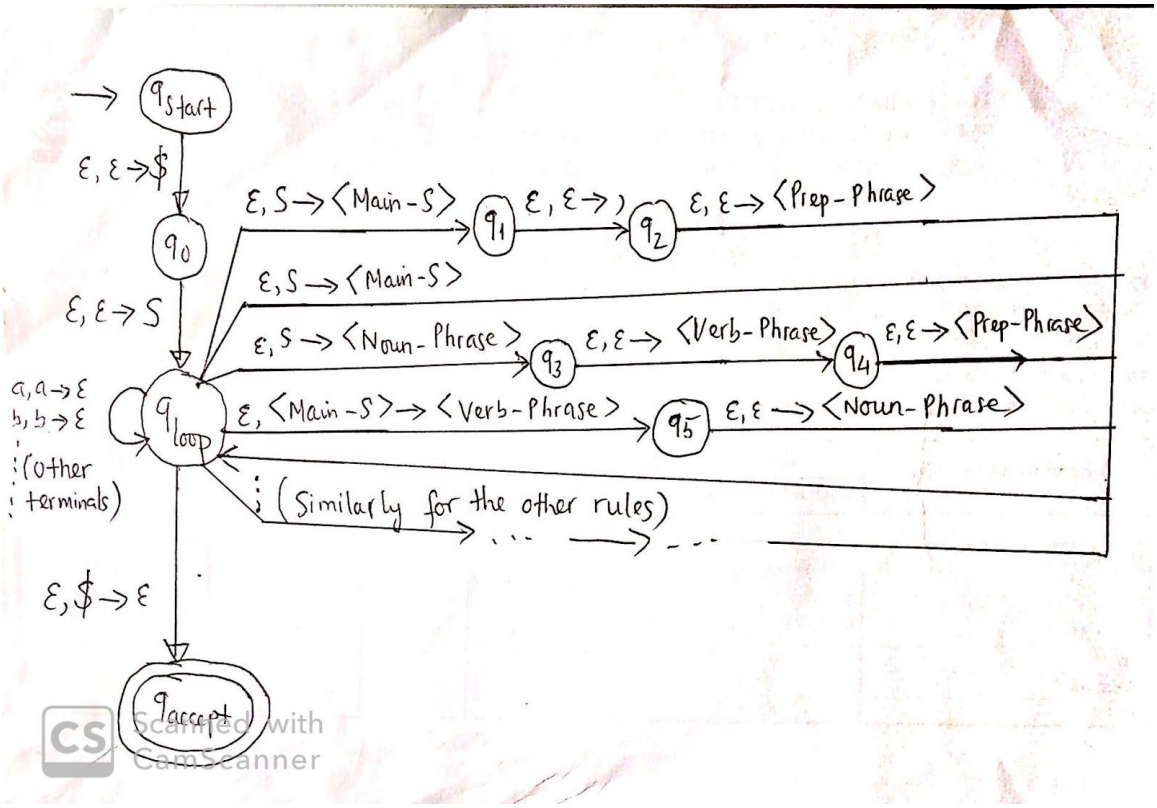
$\rightarrow$ *the actor eats the bananas at* $< Single - Cmplx - Noun >$
$\rightarrow$ *the actor eats the bananas at* $< a - Article >< a - Noun >$
$\rightarrow$ *the actor eats the bananas at the* $< a - Noun >$
$\rightarrow$ *the actor eats the bananas at the river*

We see that there are at least two different **left-most derivations** for the string. Therefore, the grammar is ambiguous.
In the first derivation, the meaning of the sentence is that the actor eats the bananas and these bananas are at the river.
In the second derivation, the meaning of the sentence is that the actor eats the bananas and this action is taken at the river.

c. Formal definition of PDA **Parser1.**

The Parser1 is defined by $(Q, \sum, \Gamma, \delta, q_{start}, F)$ where:

1. Q = {q_start, q_loop, q_accept, q0, q1, q2, q3,...}
2. $\sum = \Lambda \cup \{\#, ,\}$ is the input alphabet
3. $\Gamma = \{\$\} \cup V \cup \sum$ with V being the set of variables of the grammar G in question a. This is the stack alphabet
4. δ transition function, depicted in the diagram below.
5. q_{start} : the start state
6. $F = \{q_{accept}\}$ is the set of accept states.



d. **Python implementation**

Part 2. Word by word translator

Because the machine has to output a string, I choose the finite state transducer.

A high-level description of this FST

For this, we make two major assumptions:

1. Each English word in our dictionary can be treated/encoded as one symbol. To denote this, the symbol for a word, say, “actor”, will be <actor>.
2. THe output will also be printed out symbol by symbol and we can decode that symbol into a full German word. For example, one output symbol can be <hallo>, which is “hallo” in German.
3. We only work with symbols, which means the decoding and encoding are not tasks of the FST itself.

We only need one state for this transducer, and this one state can be the accept state. We do not really care about acceptance here as the job of this machine is only to output the translated string.

Reading an input, which is a symbol corresponding to a word, from the state, we stay at the state, and output the word translated into German.

Formal definition

We define it by a 7-tuple $(\sum, \Gamma, Q, q_0, \delta, \sigma, F)$ where:

1. $\sum$ is the input alphabet and

$$\sum = \{'\#', \ < great >, \ < nice >, < tall >, < old >, < odd >, < banana >,$$
$$< river >, < cat >, < car >, < home >, < egg >, < eye >, < actor >, < app >,$$
$$< ink >, < bananas >, < cats >, < a >, < the >, < an >, < eats >, < goes >,$$
$$< sings >, < acts >, < plays >, < in >, < on >, < at >\}$$

where the '#' represents the white space that separate two words in a sentence .

2. Γ is the output alphabet and

$$\Gamma = \{< groß >, \ < nett >, \ < hoch >, \ < alt >, \ < ungerade >, \ < banane >, < fluss >, < katze >, < wagen >,$$
$$< zuhausem >, < ei >, < auge >, < darsteller >, < app >, < tinte >, \ < bananen >, \ < katzen >, \ < ein >,$$
$$< der >, < ein >, < isst >, < geht >, < singt >, < handelt >, < spielt >, < in >, < auf >, < bei >\}$$

3. Q is the set of states and $Q = \{q_0\}$
4. F is the set of finite states and $F = \{q_0\}$
5. δ is the state transition function and $\delta(q, s) = q_0$ for all $(q, s) \in Q \times \sum$, which means it always stays at q_0.
6. σ is the output function which is described below:

State	Input symbol	Output symbol
q_0	#	#
q_0	<great>	<groß>
q_0	<nice>	<nett>
q_0	<tall>	<hoch>
q_0	<old>	<alt>
q_0	<odd>	<ungerade>

q_0	<banana>	<banane>
q_0	<river>	<fluss>
q_0	<cat>	<katze>
q_0	<car>	<wagen>
q_0	<home>	<zuhause>
q_0	<egg>	<ei>
q_0	<eye>	<auge>
q_0	<actor>	<darsteller>
q_0	<app>	$< app >$
q_0	<ink>	$< tinte >$
q_0	<bananas>	$< bananen >$
q_0	<cats>	$< katzen >$
q_0	<a>	$< ein >$
q_0	<the>	$< der >$
q_0	<an>	$< ein >$
q_0	<eats>	$< isst >$
q_0	<goes>	$< geht >$
q_0	<sings>	$< singt >$
q_0	<acts>	$< handelt >$
q_0	<plays>	$< spielt >$
q_0	<in>	$< in >$
q_0	<on>	$< auf >$
q_0	<at>	$< bei >$

Part 3. Auto-Code Function generator

a. Formal definition of the grammar.

This grammar only allows two possible arguments to a function: x and y.

$G = (V, \sum, R, S)$ *where* :

1. V is the set of variables and

V={S, <Argument>, <Func>, <Exp>, <Operand>, <Operator>, <Num>, <Pos-Num>, <Int>, <Digit> }

2. $\sum$ is the set of terminals and

$$\sum = \{x, y, f, (,), =, +, -, *, /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., s, i, n, c, o, e, p, q, r, t, l.o, g, , \}$$

3. R is a finite set of rules
4. S is the start variable

The rules are given below:

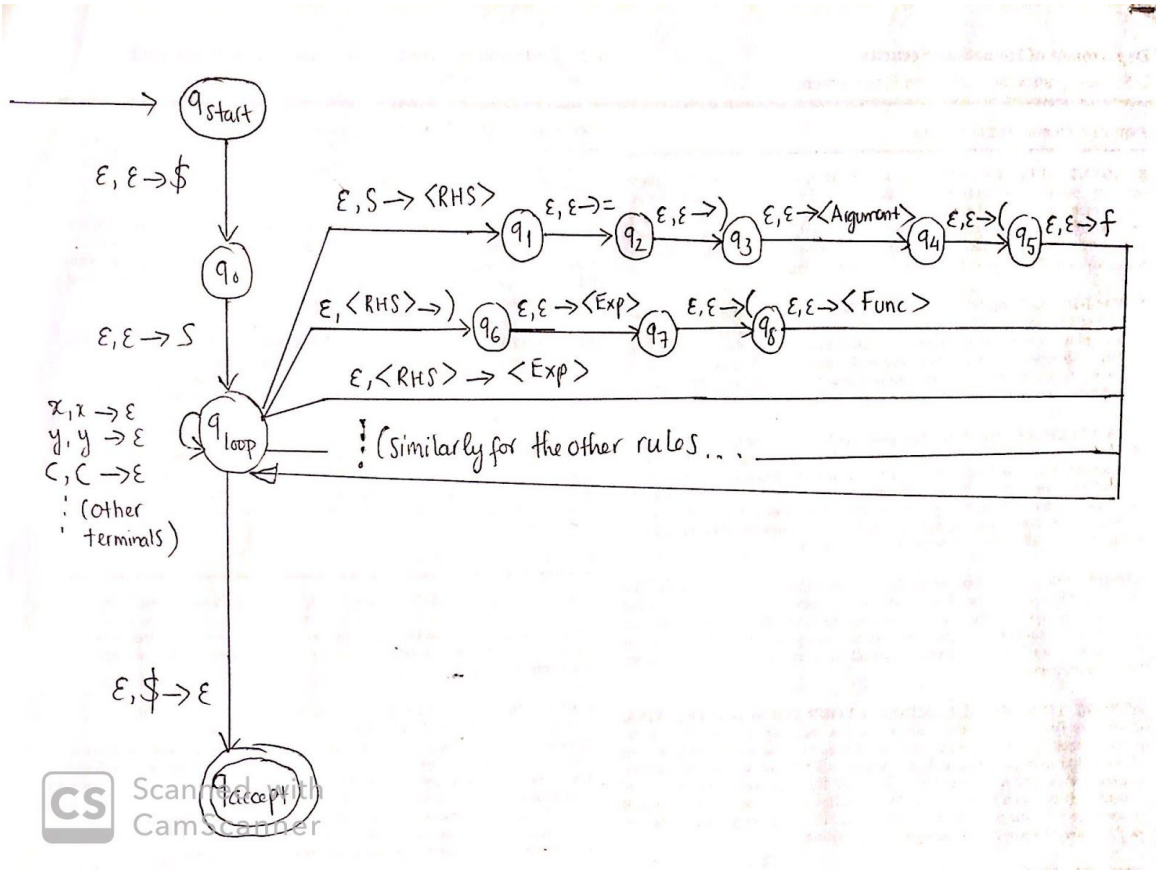
S → f(<Argument>)=<RHS>
<RHS> → <Func>(<Exp>) | <Exp> | <RHS><Operator><RHS>

<Argument> → x | y | x,y | ε
<Func> → sin | cos | exp | sqrt | log
<Exp> → <Operand> | <Operand><Operator><Exp> | <Func>(<Exp>)
<Operator> → + | - | * | /
<Operand> → x | y | <Num>
<Num> → +<Pos-Num> | -<Pos-Num> | <Pos-Num>
<Pos-Num> → <Int> | <Int>.<Int>
<Int> → <Digit> | <Digit><Int>
<Digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

b. Formal definition of the machine

The PDA ParseFunc is defined by $(Q, \sum, \Gamma, \delta, q_{start}, F)$ where:

1. Q = {q_start, q_loop, q_accept, q0, q1, q2, q3,...}
2. $\sum$ is the input alphabet, the same as $\sum$ in the formal definition of the CFG G above.
3. Γ = {\$} $\cup$ V $\cup \sum$ with V being the set of variables of the grammar G in question a.
4. δ transition function, depicted in the diagram below.
5. q_{start} : the start state
6. F = $\{q_{accept}\}$ is the set of accept states.



c. Python code for ParseFunc

Please see my attached notebook

d. Python code for Python code generator

Please see my attached notebook