Slovenská technická univerzita
Fakulta informatiky a informačných technológií

Softvérové jazyky

Dokumentácia

Lexikálny a syntaktický analyzátor
simpleURL

Lucia Rapánová , Sofia Shatokhina

LS 2023/2024

Obsah

Zadanie simpleURL

Pre definovanú gramatiku vytvorte tabuľkou riadený syntaktický analyzátor (SA) zhora nadol. Pri tvorbe SA postupujte v nasledujúcich fázach a krokoch:
ANALÝZA A NÁVRH

1. Vytvorte niekoľko príkladov viet daného jazyka.
2. Prepíšte gramatiku z BNF (Backus Naur Form) do gramatických pravidiel s alternatívami. Vyznačte miesta, kde gramatika nespĺňa podmienky pre deterministickú analýzu
3. Transformujte gramatiku tak, aby bola LL(1):
   (a) odstráňte ľavú rekurziu (ak treba)
   (b) urobte ľavú faktorizáciu (ak treba)
4. Nájdite
   (a) množinu FIRST pre každý neterminál v transformovanej gramatike
   (b) množinu FOLLOW pre každý neterminál v transformovanej gramatike
5. Pre transformovanú LL(1).gramatiku vytvorte tabuľku prechodov.
6. Implementujte lexikálny analyzátor rozpoznávajúci potrebné lexikálne jednotky.
7. Vytvorte tabuľkou riadený syntaktický analyzátor (SA), ktorý
   (a) analyzuje postupnosť lexikálnych jednotiek na vstupe
   (b) ak postupnosť zodpovedá vete jazyka, skončí SA prijatím; inak oznámi chybu
   (c) vypisuje protokol o svojej činnosti obsahujúci informáciu o uskutočnených akciách pri parsovaní vstupu
8. A testujte na vetách jazyka vytvorených v 1)

Pravidlá BNF

$$
\begin{aligned}
&url := httpaddress \mid ftpaddress \mid telnetaddress \mid mailtoaddress.\\
&httpaddress := "http://" hostport ["/" path] ["?" search].\\
&ftpaddress := "ftp://" login "/" path.\\
&telnetaddress := "telnet://" login.\\
&mailtoaddress := "mailto::" xalphas "@" hostname.\\
&login := [user [":" password] "@"] hostport.\\
&hostport := hostname [":" port].\\
&hostname := xalphas \{"." xalphas\}.\\
&port := digits.\\
&path := segment \{"/" segment\}.\\
&search := xalphas \{"+" xalphas\}.\\
&user := xalphas.\\
&password := xalphas.\\
&segment := \{xalpha\}.\\
&xalphas := xalpha \{xalpha\}.\\
&xalpha := alpha \mid digit.\\
&digits := digit\{digit\}.\\
&alpha := "A" \mid .. \mid "Z" \mid "a" \mid .. \mid "z".\\
&digit := "0" \mid .. \mid "9".
\end{aligned}
$$

Príklady viet z jazyka

Nižšie sú uvedené korektné príklady viet z jazyka (ktoré prejdú aj bez zotavenia) a nekorektné príklady viet osobitne pre lexikálnu a syntaktickú analýzu. Pri nekoretných príkladoch je uvedené, či daný príklad vie byť akceptovaný pomocou zotavení (čísla zotavení sú uvedené pri konkrétnych príkladoch; ak nie je uvedené číslo, zotavenie tomu nepomôže).
Tieto príklady sa dajú nájsť aj v kóde, konkrétne v súbore `examples.py`.

Čísla zotavení:

- 1 – Preskočenie nesprávnych znakov
- 2 – Vloženie očakávaného symbolu
- 3 – Panic Mode Recovery
- 4 – Level Phrase Recovery

(viac o zotaveniach je rozpísané v sekcii Implementácia)

Korektné vstupy – prejdú aj bez zotavenia

- `http://google`
- `http://google.com`
- `http://123sj.sk/path/file?search+hello`
- `http://fiit.stuba.sk`
- `http://abc.abc/whatever/whatever?hladanie+hladanie+zase`
- `http://youtu.be/dQw4w9WgXcQ`
- `http://hostname:12345`
- `http://softverovejazyky:2324/su/najlepsi/predmet/na/svete/frfr`
- `http://example.com/page1/page2`
- `http://example.com:8080/page`
- `ftp://lucien@vanhohenheim.fr/islands`
- `ftp://lucien:password@hostport/why`
- `ftp://lucien@vanhohenheim.fr/islands/`
- `ftp://username:password@ftp.example.com/`
- `ftp://username:password@ftp.example.com:21`
- `telnet://lucia:password@gmail.com`
- `telnet://lucien:password@vanhohenheim:13111988`
- `telnet://username:password@example.com`
- `mailto::sofia@shatokhina`
- `mailto::lucia@rapanova.com`
- `mailto::example@example.com`

Nekorektné vstupy – padnú počas lexikálnej analýzy

- `http:/`
  - chýbajúci znak v protokole
  - 2 – doplní `/` do tokenu protokolu
- `http:/google.com`
  - chýbajúci znak v protokole
  - 2 – doplní `/` do tokenu protokolu
- `akjsdkfj`
  - neobsahuje protokol
- `°⁺✧ヽ(⁰▽⁰)ノ✧⁺° ٩(｡•́‿•̀｡)۶ *:･ﾟ✧'`
  - obsahuje neplatné znaky, neobsahuje protokol
- `telnet::lucien`
  - chýbajúci znak (`//`) v protokole
  - 2 – doplní `//` do tokenu protokolu (výsledné tokeny: `telnet://`, `:`, `l`, `u`, `c`, `i`, `e`, `n`)
- `mail@to///::lucien`
  - znaky navyše v protokole
  - 1 – preskočí `@` a `///` (výsledné tokeny: `mailto::`, `l`, `u`, `c`, `i`, `e`, `n`)
- `ft*p://login`
  - neexistujúci znak navyše v protokole
  - 1 - preskočí `*` (výsledné tokeny: `ftp://`, `l`, `o`, `g`, `i`, `n`)

Nekorektné vstupy – padnú počas syntaktickej analýzy

- `http://google.`
  - za bodkou musí byť aspoň jeden alfanumerický znak (token)
- `http://example.com:80:8080`
  - token navyše (viac portov po sebe nejde)
  - 3 – preskočí `:` (akceptovaný vstup: `http://example.com:808080`)
- `http://username:password@example.com`
  - v http nie je ziadny login, `:` je posledný token ktorý bol akceptovaný, lebo to môže očakávať iba port (digits)
- `http://website.com?search/wrongorder`
  - po `?search` nesmie nasledovať `/path`
  - 3 – preskočí `/wrongorder`, pretože očakáva `+` alebo `eps` na vstupe (akceptovaný vstup je `http://website.com?search`)
  - 4 - stále zamietnutý vstup, nič sa nedoplní, nevie rozpoznať `/`
- `http://w.c?s/a+b`
  - po `?search` nesmie nasledovať `/path`
  - 3 – preskočí `/`, natrafí na `+`, ktoré už vie akceptovať (akceptovaný vstup je `http://w.c?s+b`)
  - 4 – zamietne vstup, lebo nepozná `/`
- `http://hostname:port`
  - token musí byť numericky (digits, nie letters)

- `ftp://lucien:@hostport`
  - token navyše (`@` po `:`), chýbajúci token (`/` na konci)
  - 3 – preskočí `@` (keďže očakáva password po `:`), dočíta hostport ale zamietne vstup, lebo chýba `@hostport/`)
  - 4 – zamietne (mohlo by doplniť `/`, avšak je tu token navyše, takže to nespraví)
- `ftp://lucien:@hostport@nieco/`
  - token navyše (`@` po `:`)
  - 3 – preskočí `@` a akceptuje vstup (`ftp://lucien:hostport@nieco/`)
- `ftp://lucien:password`
  - chýbajúci token (`@hostport`)
  - 4 – doplní `@`, ale stále *zamietne* vstup (lebo nebude doplňať ďalej alfanumerické tokeny (znaky))
- `ftp://:lucien@gmail.com`
  - token navyše (`:` pred `lucien`)
  - 3 – preskočí `:`, avšak chýba `/`, takže zamietne
  - 4 – zamietne, lebo sa nevie dostať cez `:`, ktoré je navyše
- `ftp://cantsearchthis.com/donttrythisathome?howtopassschoolandnotpassaway`
  - chýbajúci token (buď `:password@hostport` alebo `@hostport`)
- `ftp://lucien@vanhohenheim.fr`
  - chýbajúci token (`/` na konci)
  - 4 – doplní `/`, akceptuje vstup (`ftp://lucien@vanhohenheim.fr`)
- `ftp://lucien:password@hostport`
  - chýbajúci token (`/` na konci)
  - 4 – doplní, `/` akceptuje vstup (`ftp://lucien:password@hostport/`)
- `telnet://vanhohenheim`
  - chýba `@hostport` z loginu, recovery nebude
- `telnet://lucien:password@vanhoheinheim:1311.1988`
  - token navyše (`.` v porte)
  - 3 – odstráni `.` z portu
- `telnet://example.com:23`
  - chýbajúci token (telnet potrebuje login a v logine je vzdy `@hostname`)
- `telnet://example.com`
  - chýbajúci token (telnet potrebuje login a v logine je vzdy `@hostname`)
- `mailto::lucien`
  - v mailto musí byť `@`
  - 4 – doplní `@,` ale vstup bude stále zamietnutý, lebo chýba alfanumerický znak (token)
- `mailto::example@example.com?subject`
  - token navyše (v mailto nie je `?search`)

Nekorektné vstupy – opravované dvoma zotaveniami

- `htttp://google.com:abc80`
  - 1 – odstráni znak navyše (`t` v `htttp://`)
  - 3 – preskočí alfabetické znaky v porte (akceptovaný vstup: `http://google.com:80`)
- `fttp://lucien@vh.fr`
  - 1 – odstráni znak navyše (`t` v `fttp://`)
  - 4 – doplní chýbajúci token / na konci (akceptovaný vstup: `ftp://lucien@vh.fr/`)
- `mailto:lucia@rapanova@.com`
  - 2 – doplní chýbajúci znak (`:` do `mailto:`)
  - 3 – preskočí token navyše (`@` pred `.com`, akceptovaný vstup: `mailto::lucia@rapanova.com`)
- `ftp/examples@py`
  - 2 – doplní chýbajúci znak (`:/` do `ftp/`)
  - 4 – doplní chýbajúci token / na konci (akceptovaný vstup: `ftp://examples@py/`)

Prevod z BNF

Ako prvé sme prepísali najmä zátvorky, ktoré znamenali počty výskytov.

```
url → httpaddress | ftpaddress | telnetaddress | mailtoaddress

httpaddress → “http://” hostport“/”path”?”search
httpaddress → “http://” hostport“/”path
httpaddress → “http://” hostport”?”search
httpaddress → “http://” hostport

ftpaddress → “ftp://”login”/”path
telnetaddress → “telnet://”login
mailtoaddress → “mailto::”xalphas”@”hostname

login → user”:”password”@”hostport
login → user”@”hostport
login → hostport

hostport → hostname”:”port
hostport → hostname

hostname → xalphas
hostname → xalphas”.”hostname

port → digits

path → segment
path → segment”/”path

search → xalphas
search → xalphas”+”search

user → xalphas
password → xalphas

segment → xalpha segment
segment → eps

xalphas → xalpha
xalphas → xalpha xalphas


xalpha → alpha
xalpha → digit

digits → digit
digits → digit digits
```

```
alpha → "A" | .. | "Z" | "a" | .. | "z"
digit → "0" | .. | "9"
```

Transformácia na tokeny

Terminály sme transformovali na nasledovné lexikálne jednotky (tokeny). Písmená a čísla sme zatiaľ ponechali ako samostatné tokeny, nevypisujeme všetky v dokumentácii.

```
http: "http://"
ftp: "ftp://"
telnet: "telnet://"
mailto: "mailto::"

slash: "/"
qmark: "?"
at: "@"
colon: ":"
dot: "."
plus: "+"
```

Prepísané pravidlá s tokenmi:
Pri prepisovaní sme kontrolovali aj ľavé rekurzie a ľavé prefixy(ľavý token zvýraznený). Nenašli sme žiadne ľavé rekurzie, ale našli sme niekoľko ľavých prefixov.

```
url → httpaddress | ftpaddress | telnetaddress | mailtoaddress

httpaddress → http hostport slash path qmark search
httpaddress → http hostport slash path
httpaddress → http hostport qmark search
httpaddress → http hostport


ftpaddress → ftp login slash path
telnetaddress → telnet login
mailtoaddress → mailto xalphas at hostname

login → user colon password at hostport
login → user at hostport
login → hostport

hostport → hostname colon port
hostport → hostname

hostname → xalphas
hostname → xalphas dot hostname
```

```
port → digits

path → segment
path → segment slash path

search → xalphas
search → xalphas plus search

user → xalphas
password → xalphas

segment → xalpha segment
segment → eps

xalphas → xalpha
xalphas → xalpha xalphas

xalpha → alpha
xalpha → digit

digits → digit
digits → digit digits

alpha → A | .. | Z | a | .. | z
digit → 0 | .. | 9
```

Odstránenie ľavého prexifu

V nasledujúcich krokoch odstraňujeme ľavý prefix:

```
url → httpaddress |
      ftpaddress |
      telnetaddress |
      mailtoaddress
-----------------------------------------------------------------
httpaddress → http hostport slash path qmark search
httpaddress → http hostport slash path
httpaddress → http hostport qmark search
httpaddress → http hostport

httpaddress → http hostport httpaddr_1

httpaddr_1 → slash path httpaddr_2
httpaddr_1 → qmark search
httpaddr_1 → eps

httpaddr_2 → qmark search
httpaddr_2 → eps
-----------------------------------------------------------------
ftpaddress → ftp login slash path
telnetaddress → telnet login
mailtoaddress → mailto xalphas at hostname
-----------------------------------------------------------------
login → user colon password at hostport
login → user at hostport
login → hostport

login -> user login_1
login -> hostport

login_1 -> colon password at hostport
login_1 -> at hostport
-----------------------------------------------------------------
hostport → hostname colon port
hostport → hostname

hostport -> hostname hostport_1
hostport_1 -> colon port
hostport_1 -> eps
-----------------------------------------------------------------
hostname → xalphas
hostname → xalphas dot hostname

hostname -> xalphas hostname_1
hostname_1 -> dot hostname
```

```
hostname_1 -> eps
-----------------------------------------------------------------
port → digits
-----------------------------------------------------------------
path → segment
path → segment slash path

path -> segment path_1
path_1 -> slash path
path_1 -> eps
-----------------------------------------------------------------
search → xalphas
search → xalphas plus search

search -> xalphas search_1
search_1 -> plus search
search_1 -> eps
-----------------------------------------------------------------
user → xalphas
password → xalphas
segment → xalpha segment
segment → eps
-----------------------------------------------------------------
xalphas → xalpha
xalphas → xalpha xalphas

xalphas -> xalpha xalphas_1
xalphas_1 -> xalphas
xalphas_1 -> eps
-----------------------------------------------------------------
xalpha → alpha
xalpha → digit
-----------------------------------------------------------------
digits → digit
digits → digit digits

digits -> digit digits_1
digits_1 -> digits
digits_1 -> eps
-----------------------------------------------------------------
alpha → A | .. | Z | a | .. | z
digit → 0 | .. | 9
```

Gramatika po aplikovaní úprav:

1. **url → httpaddress**
2. **url → ftpaddress**
3. **url → telnetaddress**
4. **url → mailtoaddress**

5. **httpaddress** → http **hostport httpaddr_1**
6. **httpaddr_1** → slash **path httpaddr_2**
7. **httpaddr_1** → qmark **search**
8. **httpaddr_1** → *eps*
9. **httpaddr_2** → qmark **search**
10. **httpaddr_2** → *eps*
11. **ftpaddress** → ftp **login** slash **path**
12. **telnetaddress** → telnet **login**
13. **mailtoaddress** → mailto **xalphas** at **hostname**
14. ***login → user login_1***
15. ***login → hostport***
16. **login_1** → colon **password** at **hostport**
17. **login_1** → at **hostport**
18. **hostport** → **hostname hostport_1**
19. **hostport_1** → colon **port**
20. **hostport_1** → *eps*
21. **hostname** → **xalphas hostname_1**
22. **hostname_1** → dot **hostname**
23. **hostname_1** → *eps*
24. **port** → **digits**
25. **path** → **segment path_1**
26. **path_1** → slash **path**
27. **path_1** → *eps*
28. **search** → **xalphas search_1**
29. **search_1** → plus **search**
30. **search_1** → *eps*
31. **user** → **xalphas**
32. **password** → **xalphas**
33. **segment** → **xalpha segment**
34. **segment** → *eps*
35. **xalphas** → **xalpha xalphas_1**
36. **xalphas_1** → **xalphas**
37. **xalphas_1** → *eps*
38. **xalpha** → **alpha**
39. **xalpha** → **digit**
40. **digits** → **digit digits_1**
41. **digits_1** → **digits**
42. **digits_1** → *eps*
43. **alpha** → A | .. | Z | a | .. | z
44. **digit** → 0 | .. | 9

Množiny FIRST a FOLLOW

Neterminály:
`url, httpaddress, telnetaddress, ftpaddress, mailtoaddress, hostport, path, search, httpaddr_1, httpaddr_2, login, login_1, password, user, port, hostname, hostport_1, hostname_1, xalphas, digits, segment, path_1, search_1, xalpha, digit, alpha, xalphas_1, digits_1`

Terminály:
`http, ftp, telnet, mailto, slash, qmark, at, colon, dot, plus, A … z, 0 … 9`

Pri spísaní týchto pravidiel sme sa pozerali, či nenastávajú FIRST/FIRST alebo FIRST/FOLLOW konflikty. Farebne sme vyznačili miesta, na ktorých predpokladame, že môžu nastať konflikty.

1. FIRST(**url**) = FIRST(httpaddress) ∪ FIRST(telnetaddress) ∪ FIRST(ftpaddress) ∪ FIRST(mailtoaddress) = {http, telnet, ftp, mailto}
2. FIRST(**httpaddress**) = {http}
3. FIRST(**httpaddr_1**) = {slash, qmark, *eps*}
4. FIRST(**httpaddr_2**) = {qmark, *eps*}
5. FIRST(**telnetaddress**) = {telnet}
6. FIRST(**ftpaddress**) = {ftp}
7. FIRST(**mailtoaddress**) = {mailto}
8. FIRST(**hostport**) = FIRST(hostname) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
9. FIRST(**hostport_1**) = {colon, *eps*}
10. FIRST(**path**) = FIRST(segment) – *eps* ∪ FIRST(path_1) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ∪ {slash, *eps*} = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, slash, *eps*}
11. FIRST(**path_1**) = {slash, *eps*}
12. FIRST(**search**) = FIRST(xalphas) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
13. FIRST(**search_1**) = {plus, *eps*}
14. FIRST(**login**) = FIRST(user) ∪ FIRST(hostport) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ∪ {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9} = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
15. FIRST(**login_1**) = {colon, at}
16. FIRST(**password**) = FIRST(xalphas) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
17. FIRST(**user**) = FIRST(xalphas) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
18. FIRST(**port**) = FIRST(digits) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
19. FIRST(**hostname**) = FIRST(xalphas) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
20. FIRST(**hostname_1**) = {dot, *eps*}
21. FIRST(**xalphas**) = FIRST(xalpha) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
22. FIRST(**xalphas_1**) = FIRST(xalphas) ∪ *eps* = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *eps*}
23. FIRST(**digits**) = FIRST(digit) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

24. FIRST(**digits_1**) = FIRST(digits) ∪ *eps* = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *eps*}
25. FIRST(**segment**) = FIRST(xalpha) ∪ *eps* = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *eps*}
26. FIRST(**xalpha**) = FIRST(alpha) ∪ FIRST(digit) = {A...z} ∪ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
27. FIRST(**digit**) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
28. FIRST(**alpha**) = {A...z}

Pri kontrole potenciálnych FIRST/FIRST konfliktov sme našli konflikt v **FIRST(login)** (ktorý vychádza z pravidiel 14 a 15).
FIRST(user) ∩ FIRST(hostport) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ∩ {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ≠ ∅

Pravidlá sme upravili tak, aby výsledná gramatika bola typu LL(1), a to tým spôsobom, že sme odstránili jedno z pravidiel spôsobujúcich konflikt (konkrétne pravidlo 15). Nasledujúce FOLLOW pravidlá sú spísané po uprave gramatiky (čiže bez FIRST/FIRST konfliktu).

14. login → user login_1
15. **login_1** → colon **password** at **hostport**
16. **login_1** → at **hostport**

14. FIRST(login) → FIRST(user) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

1. FOLLOW(**url**) = {$}
2. FOLLOW(**httpaddress**) = FOLLOW(url) = {$}
3. FOLLOW(**httpaddr_1**) = FOLLOW(httpaddress) = {$}
4. FOLLOW(**httpaddr_2**) = FOLLOW(httpaddr_1) = {$}
5. FOLLOW(**telnetaddress**) = FOLLOW(url) = {$}
6. FOLLOW(**ftpaddress**) = FOLLOW(url) = {$}
7. FOLLOW(**mailtoaddress**) = FOLLOW(url) = {$}
8. FOLLOW(**hostport**) = FIRST(httpaddr_1) – *eps* ∪ FOLLOW(httpaddress) ∪ FOLLOW(login) ∪ FOLLOW(login_1) = {slash, qmark} ∪ {$} ∪ {slash, $} ∪ {slash, $} = {slash, qmark, $}
9. FOLLOW(**hostport_1**) = FOLLOW(hostport) = {slash, qmark, $}
10. FOLLOW(**path**) = FIRST(httpaddr_2) – *eps* ∪ FOLLOW(httpaddr_1) ∪ FOLLOW(ftpaddress) ∪ FOLLOW(path_1) = {qmark} ∪ {$} ∪ {$} ∪ ~~FOLLOW(path)~~ = {qmark, $}
11. FOLLOW(**path_1**) = FOLLOW(path) = {qmark, $}
12. FOLLOW(**search**) = FOLLOW(httpaddr_1) ∪ FOLLOW(httpaddr_2) ∪ FOLLOW(search_1) = {$} ∪ {$} ∪ ~~FOLLOW(search)~~ = {$}
13. FOLLOW(**search_1**) = FOLLOW(search) = {$}
14. FOLLOW(**login**) = {slash} ∪ FOLLOW(telnetaddress) = {slash, $}
15. FOLLOW(**login_1**) = FOLLOW(login) = {slash, $}

16. FOLLOW(**password**) = {at}
17. FOLLOW(**user**) = FIRST(login_1) = {colon, at}
18. FOLLOW(**port**) = FOLLOW(hostport_1) = {slash, qmark, $}
19. FOLLOW(**hostname**) = FOLLOW(mailtoaddress) ∪ FIRST(hostport_1) – *eps* ∪ FOLLOW(hostport) ∪ FOLLOW(hostname_1) ∪ ~~FOLLOW(hostname)~~= {$} ∪ {colon} ∪ {slash, qmark, $} = {$, colon, slash, qmark}
20. FOLLOW(**hostname_1**) = FOLLOW(hostname) = {$, colon, slash, qmark}
21. FOLLOW(**xalphas**) = {at} ∪ FIRST(hostname_1) – *eps* ∪ FOLLOW(hostname) ∪ FIRST(search_1) – *eps* ∪ FOLLOW(search) ∪ FOLLOW(user) ∪ FOLLOW(password) ∪ FOLLOW(xalphas_1) = {at} ∪ {dot} ∪ {$, colon, slash, qmark} ∪ {plus} ∪ {$} ∪ {colon, at} ∪ {at} = {at, dot, $, colon, slash, qmark, plus}
22. FOLLOW(**xalphas_1**) = FOLLOW(xalphas) = {at, dot, $, colon, slash, qmark, plus}
23. FOLLOW(**digits**) = FOLLOW(port) ∪ FOLLOW(digits_1) = {slash, qmark, $}
24. FOLLOW(**digits_1**) = FOLLOW(digits) = {slash, qmark, $}
25. FOLLOW(**segment**) = FIRST(path_1) – *eps* ∪ FOLLOW(path) ∪ ~~FOLLOW(segment)~~ = {slash} ∪ {qmark, $} = {slash, qmark, $}
26. FOLLOW(**xalpha**) = FIRST(segment) – *eps* ∪ FOLLOW(segment) ∪ FIRST(xalphas_1) – *eps* ∪ FOLLOW(xalphas) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ∪ {slash, qmark, *eps*, $} ∪ {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ∪ {at, dot, $, colon, slash, qmark, plus} = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, slash, qmark, $, at, dot, colon, plus}
27. FOLLOW(**digit**) = FOLLOW(xalpha) ∪ FIRST(digits_1) – *eps* ∪ FOLLOW(digits) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, slash, qmark, $, at, dot, colon, plus} ∪ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ∪ {slash, qmark, $} = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, slash, qmark, $, at, dot, colon, plus}
28. FOLLOW(**alpha**) = FOLLOW(xalpha) = {A...z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, slash, qmark, $, at, dot, colon, plus}

Pri kontrole potenciálnych FIRST/FOLLOW konfliktov sme nenašli žiaden konflikt.

Finálna množina pravidiel gramatiky typu LL(1):

1. **url** → **httpaddress**
2. **url** → **ftpaddress**
3. **url** → **telnetaddress**
4. **url** → **mailtoaddress**
5. **httpaddress** → http **hostport httpaddr_1**
6. **httpaddr_1** → slash **path httpaddr_2**
7. **httpaddr_1** → qmark **search**
8. **httpaddr_1** → *eps*
9. **httpaddr_2** → qmark **search**
10. **httpaddr_2** → *eps*

11. **ftpaddress** → ftp **login** slash **path**
12. **telnetaddress** → telnet **login**
13. **mailtoaddress** → mailto **xalphas** at **hostname**
14. ***login → user login_1***
15. **login_1** → colon **password** at **hostport**
16. **login_1** → at **hostport**
17. **hostport** → **hostname hostport_1**
18. **hostport_1** → colon **port**
19. **hostport_1** → *eps*
20. **hostname** → **xalphas hostname_1**
21. **hostname_1** → dot **hostname**
22. **hostname_1** → *eps*
23. **port** → **digits**
24. **path** → **segment path_1**
25. **path_1** → slash **path**
26. **path_1** → *eps*
27. **search** → **xalphas search_1**
28. **search_1** → plus **search**
29. **search_1** → *eps*
30. **user** → **xalphas**
31. **password** → **xalphas**
32. **segment** → **xalpha segment**
33. **segment** → *eps*
34. **xalphas** → **xalpha xalphas_1**
35. **xalphas_1** → **xalphas**
36. **xalphas_1** → *eps*
37. **xalpha** → **alpha**
38. **xalpha** → **digit**
39. **digits** → **digit digits_1**
40. **digits_1** → **digits**
41. **digits_1** → *eps*
42. **alpha** → A | .. | Z | a | .. | z
43. **digit** → 0 | .. | 9

Tabuľka FIRST a FOLLOW

Odkaz na tabuľku (prístup cez stuba účet):
https://docs.google.com/spreadsheets/d/1KGzk8QERBEgjrE1475TYZ8OjS_8Kh1xQpVR2NZsoSco/edit#gid=0

Číslo pravidla	Pravidla	FIRST	FOLLOW
1	url → httpaddress	{http}	
2	url → ftpaddress	{ftp}	
3	url → telnetaddress	{telnet}	
4	url → mailtoaddress	{mailto}	
5	httpaddress → http hostport httpaddr_1	{http}	
6	httpaddr_1 → slash path httpaddr_2	{slash}	
7	httpaddr_1 → qmark search	{qmark}	
8	httpaddr_1 → eps	{eps}	{$}
9	httpaddr_2 → qmark search	{qmark}	
10	httpaddr_2 → eps	{eps}	{$}
11	ftpaddress → ftp login slash path	{ftp}	
12	telnetaddress → telnet login	{telnet}	
13	mailtoaddress → mailto xalphas at hostname	{mailto}	
14	login → user login_1	{A..z, 0...9}	
15	login_1 → colon password at hostport	{colon}	
16	login_1 → at hostport	{at}	
17	hostport → hostname hostport_1	{A..z, 0...9}	
18	hostport_1 → colon port	{colon}	
19	hostport_1 → eps	{eps}	{slash, qmark, $}
20	hostname → xalphas hostname_1	{A..z, 0...9}	
21	hostname_1 → dot hostname	{dot}	
22	hostname_1 → eps	{eps}	{$, colon, slash, qmark}
23	port → digits	{0...9}	
24	path → segment path_1	{A..z, 0...9, slash, eps}	{qmark, $}
25	path_1 → slash path	{slash}	
26	path_1 → eps	{eps}	{qmark, $}
27	search → xalphas search_1	{A..z, 0...9}	
28	search_1 → plus search	{plus}	
29	search_1 → eps	{eps}	{$}
30	user → xalphas	{A..z, 0...9}	
31	password → xalphas	{A..z, 0...9}	
32	segment → xalpha segment	{A..z, 0...9}	
33	segment → eps	{eps}	{slash, qmark, $}
34	xalphas → xalpha xalphas_1	{A..z, 0...9}	
35	xalphas_1 → xalphas	{A..z, 0...9}	
36	xalphas_1 → eps	{eps}	{at, dot, $, colon, slash, qmark, plus}
37	xalpha → alpha	{A..z}	
38	xalpha → digit	{0...9}	
39	digits → digit digits_1	{0...9}	
40	digits_1 → digits	{0...9}	
41	digits_1 → eps	{eps}	{slash, qmark, $}
42	alpha → A | .. | Z | a | .. | z	{A...z}	
43	digit → 0 | .. | 9	{0...9}	

Tabuľka prechodov

Odkaz na tabuľku:
https://docs.google.com/spreadsheets/d/1_DMf-oY3jhwFLUFLxhbxmaZwwb_OvSYzIk3pqPvEVvo/edit#gid=0

tokeny -->	http	ftp	telnet	mailto	slash	qmark	at	colon	dot	plus	A ... z	0 .. 9	$
url	1	2	3	4									
httpaddress	5												
httpaddr_1					6	7							8
httpaddr_2						9							10
telnetaddress			12										
ftpaddress		11											
mailtoaddress				13									
login											14	14	
login_1							16	15					
hostport											17	17	
hostport_1					19	19		18					19
hostname											20	20	
hostname_1					22	22		22	21				22
port												23	
path					24	24					24	24	24
path_1					25	26							26
search											27	27	
search_1										28			29
user											30	30	
password											31	31	
segment					33	33					32	32	33
xalphas											34	34	
xalphas_1					36	36	36	36	36	36	35	35	36
xalpha											37	38	
digits												39	
digits_1					41	41						40	41
alpha											42		
digit												43	

Pre zjednodušenie nevypisujeme všetky znaky z A...z a všetky čísla 0...9, pretože na všetky znaky sa aplikuje rovnaké pravidlo.

Implementácia

Riešenie sme implementovali v jazyku Python. Máme 4 moduly:

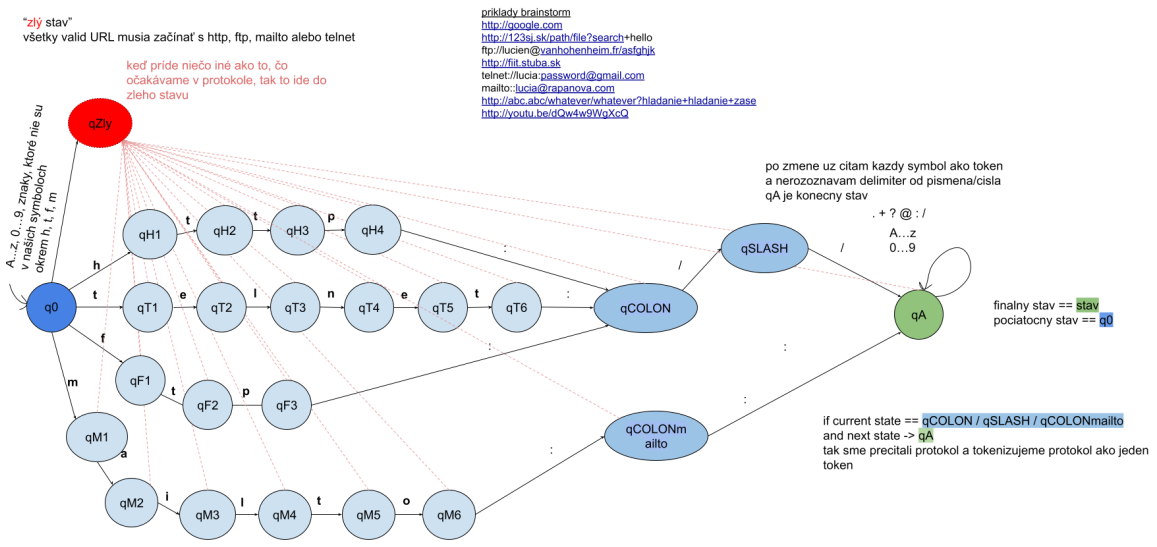
1. `rules.py` obsahujú zoznam terminálych a neterminálych symbolov, tabuľku prechodov a zoznam pravidiel. na reprezentáciu parsovacej tabuľky sme využili dátovú štruktúru slovníka
2. `syntax_analysis.py`
3. `lexical_analysis.py`
4. `main.py` slúži na spustenie hlavného programu

Lexikálna analýza

Spracovanie vstupu a prevod na tokeny robíme pomocou konečného automatu. Vstup čítame od používateľa a tokenizujeme. Očakávame, že každá URL začne nejakým protokolom. Jednotlivé protokoly pokladáme za tokeny:

`http://` sa tokenizuje ako token "`http`",
`ftp://` sa tokenizuje ako token "`ftp`",
`telnet://` sa tokenizuje ako token "`telnet`",
`mailto::` sa tokenizuje ako token "`mailto`"

Znaky zo vstupu prehlásime za token, keď prídeme do stavu qA. Následne čítame ďalšie znaky zo vstupu. Každý symbol zo vstupu pokladáme za jednotlivé tokeny. Pokiaľ príde nepredpokladaný znak alebo znak, ktorý nie je v našich vstupných symboloch, prechádzame do stavu qZly, v ktorom zamietame vstup. Výsledkom tokenizácie zo zoznam tokenov.



Implementovali sme dva módy zotavenia:

1. **Preskočenie nesprávnych znakov** - ak narazíme na znaky, ktoré nie sú predpokladané (napr. pri čítaní protokolu), alebo nie sú v našej abecede, tieto znaky ignorujeme a neukladáme ich do tokenov a pokračujeme ďalej v čítaní vstupu, až kým neprečítame správny znak. (napr. http://goo%gle.com → preskočíme nesprávny znak "%")
2. **Vloženie očakávaného symbolu** - ak pri čítaní protokolu narazíme na symbol, ktorým sa nepresunieme do očakávaného stavu, do tokenov vložíme očakávaný symbol a nastavíme korektný stav. (napr. pri čítaní http:/google.com očakávame ešte jeden "/", ktorý neprišiel. Tento vstup opravíme vložením znaku "/" → http://google.com)

Príklad tokenizovaného vstupu
http://google.com → ['http://', 'g', 'o', 'o', 'g', 'l', 'e', '.', 'c', 'o', 'm']

Syntaktická analýza

Pri syntaktickej analýze sme využili dátovú štruktúru zásobníka. Pri čítaní tokenov sa kontroluje aktuálny token s vrchom zásobníka. Pozerá sa do parsovacej tabuľky, či existuje pravidlo pre daný token a daný vrch zásobníka. Ak áno, vrch zásobníka sa popne a do zásobníka sa pridá dané pravidlo. Ak je navrchu zásobníka rovnaký znak ako na vstupe, posúvame sa pri čítaní vstupu. Pri čítaní písmen a čísel sme pre zjednodušenie definovali tokeny "letter" a "digit", ktoré reprezentujú písmená a čísla. Vstup bude prijatý, ak sme prečítali celý vstup a na konci je na vrchu zásobníka len spodok zásobníka Z0. Program vypisuje prijatý vstup (ako list tokenov) po ukončení analýzy (a to aj v prípade, že *celý* vstup akceptovaný nebol; takto vieme zistiť, na ktorom tokene sme sa zasekli).

Implementovali sme dva módy zotavenia:

3. **Panic Mode Recovery** - ak sa vrch zásobníka nenachádza v parsovacej tabuľke, alebo neexistuje také pravidlo, ktoré by sedelo na token a vrch zásobníka, tak preskakujeme tokeny, až kým nenájdeme token, pre ktorý platí, že existuje pravidlo, na ktoré sedí vrch zásobníka a daný token na vstupe. (napr. – ftp://lucien:@hostport@nieco/ → preskočí prvý neočakávaný @ a výstupom bude ftp://lucien:hostport@nieco/)
4. **Level Phrase Recovery** - ak je na vrchu zásobníka znak delimitera, ktorý očakávame ako token, ale tento token nepríde a nastala by chyba, tak tento token doplníme do vstupu. (napr. pri ftp musí vždy na konci byť "/" ftp://lucien:password@hostport → ftp://lucien:password@hostport/)

Spustenie programu

Program sa spúšťa pomocou `python main.py` a ďalej je možné doplniť rôzne prepínače. Prepínačmi `-lex` a `-syn` sa zapína zotavovanie z chýb pre jednotlivé analýzy a pomocou čísel sa vyberá mód zotavenia. Pomocou `-f` prepínača môžeme zadať názov súboru, do ktorého chceme uložiť výstup zo syntaktického stromu. Prepínačom `-v` nastavujeme "verbózny" režím, počas ktorého sú vypisované podrobné logy o priebehu programu (spracovávané znaky/tokeny, stav zásobníka atď). Tichý režim/nevypisovanie postupu je defaultný stav spúšťania programu (nevyžaduje prepínač).

Pri implementácii tohto projektu sme využili niekoľko knižníc.
Dajú sa nainštalovať pomocou príkazu `pip install -r requirements.txt`

- `from` **`automata`**`.fa.dfa import DFA` – z tejto knižnice sme importovali deterministický konečný automat (DFA) pre lexikálnu analýzu
- **`treelib`** – knižnica na vytvorenie a vykreslenie syntaktického stromu
- **`logging`** – pomocou tejto knižnice sme implementovali tichý režim (možnosť schovania pomocných výpisov)
- **`argparse`** – pomocou tejto knižnice sme nastavovali prepínače na spustenie programu
- `from` **`collections`** `import deque` – dátová štruktúra na reprezentáciu zásobníka

Implementácia je dostupná na GitHube:
https://github.com/luciarap/simpleURL-analyzer

Príklad spustenia: `python main.py -lex 1 -syn 3 -v -f syntax_tree`

```
PS C:\Users\user\FIIT-SJ> & C:/Users/user/AppData/Local/Programs/Python/Python312/python.exe c:/Users/user/FIIT-SJ/main.py --help
usage: main.py [-h] [-lex LEXICAL_ANALYSIS_RECOVERY] [-syn SYNTAX_ANALYSIS_RECOVERY] [-f FILE_NAME_TREE] [-v]

options:
  -h, --help            show this help message and exit
  -lex LEXICAL_ANALYSIS_RECOVERY, --lexical_analysis_recovery LEXICAL_ANALYSIS_RECOVERY
                        Choose Lexical Analysis Method 0: No recovery, 1: Skip incorrect symbols, 2: Insert expected symbols
  -syn SYNTAX_ANALYSIS_RECOVERY, --syntax_analysis_recovery SYNTAX_ANALYSIS_RECOVERY
                        Choose Syntax Analysis Method 0: No recovery, 3: Panic Mode Recovery, 4: Phrase Level Recovery
  -f FILE_NAME_TREE, --file_name_tree FILE_NAME_TREE
                        Enter the name of the file where to save the syntax tree graphviz visualization code (a new file will be created).
                        If no name provided, graphviz visualization omitted.
  -v, --verbose         Verbose Mode
```

Výstup v tichom režime:

```
PS C:\Users\user\FIIT-SJ> & C:/Users/user/AppData/Local/Programs/Python/Python312/python.exe c:/Users/user/FIIT-SJ/main.py

°.✧ Lexical and syntax analyzer ✧.°

Type INPUT or PRESS CTRL + C to exit the program: http://google.com

TOKENIZER

Input accepted
Tokenized input: ['http://', 'g', 'o', 'o', 'g', 'l', 'e', '.', 'c', 'o', 'm']

SYNTAX ANALYSIS

Input accepted
Accepted tokens: ['http://', 'g', 'o', 'o', 'g', 'l', 'e', '.', 'c', 'o', 'm']
Type INPUT or PRESS CTRL + C to exit the program:
```

Program s vypnutým tichým režimom vypisuje postup. Premenná `current_token` je súčasne skladaný token (v našom prípade je to token protokolu), `currently_reading` je súčasne čítaný znak zo vstupu.

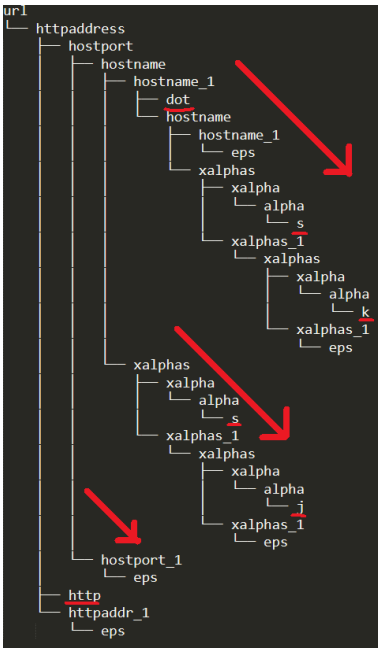
```
Type INPUT or PRESS CTRL + C to exit the program: http://google.com

TOKENIZER

INFO: Error recovery is turned on: ignore
INFO:

INFO: Currently reading: h, current_token:
INFO: Current state: q0
INFO: Next state: qH1
INFO: Appending protocol symbols
INFO: 'next state': q0 / next state: qH1
INFO: Appending: h
INFO: Current state: q0
INFO:

INFO: Currently reading: t, current_token: h
INFO: Current state: qH1
INFO: Next state: qH2
INFO: Appending protocol symbols
INFO: 'next state': qH1 / next state: qH2
INFO: Appending: t
INFO: Current state: qH1
INFO:
```

```
INFO: Currently reading a new symbol
stack - deque(['Z0', 'httpaddr_1', 'hostport_1', 'hostname_1', 'xalphas_1']) / top - letter / current_token - g / token - letter
INFO: Accepted tokens: ['http://']
 Remaining tokens: ['o', 'o', 'g', 'l', 'e', '.', 'c', 'o', 'm', '$']
INFO: Match (top == token)
 Top: xalphas_1 --- Stack: deque(['Z0', 'httpaddr_1', 'hostport_1', 'hostname_1']) --- Token: letter
INFO: Reading next token from input
INFO: Currently reading token: letter
INFO: Parsing the RULES:
Top - xalphas_1 / Token - letter / Rule number - 35 / Rule - xalphas
INFO: Current stack: deque(['Z0', 'httpaddr_1', 'hostport_1', 'hostname_1', 'xalphas'])
INFO:

INFO: Currently reading a new symbol
stack - deque(['Z0', 'httpaddr_1', 'hostport_1', 'hostname_1']) / top - xalphas / current_token - o / token - letter
INFO: Accepted tokens: ['http://', 'g']
 Remaining tokens: ['o', 'g', 'l', 'e', '.', 'c', 'o', 'm', '$']
INFO: Parsing the RULES:
Top - xalphas / Token - letter / Rule number - 34 / Rule - xalpha xalphas_1
INFO: Current stack: deque(['Z0', 'httpaddr_1', 'hostport_1', 'hostname_1', 'xalphas_1', 'xalpha'])
INFO:

INFO: Currently reading a new symbol
stack - deque(['Z0', 'httpaddr_1', 'hostport_1', 'hostname_1', 'xalphas_1']) / top - xalpha / current_token - o / token - letter
INFO: Accepted tokens: ['http://', 'g']
 Remaining tokens: ['o', 'g', 'l', 'e', '.', 'c', 'o', 'm', '$']
INFO: Parsing the RULES:
Top - xalpha / Token - letter / Rule number - 37 / Rule - alpha
INFO: Current stack: deque(['Z0', 'httpaddr_1', 'hostport_1', 'hostname_1', 'xalphas_1', 'alpha'])
INFO:
```

Syntaktický strom

Syntaktický strom sme implementovali pomocou knižnice **`treelib`**.
Ak zvolíme "verbózny" režim (čiže nie tichý režim), program vypíše syntaktický strom do konzoly a tiež uloží vytvorený strom v podobe **graphviz** kódu do súboru, ktorý je možné zobraziť na tejto stránke:
https://dreampuf.github.io/GraphvizOnline

Príklady syntaktických stromov pre vstup `http://sj.sk` je možné vidieť na obrázkoch nižšie. Strom sa číta "zľava-zhora" (názorná ukážka na obrázku nižšie). Najprv sa prečíta vetva, ktorá ide vertikálne – na obrázku z `httpaddress` čítame najprv `httpaddr_1`, potom `http`, potom `hostport`. Keď sme takto vošli do vetvy, tak čítame už zhora.



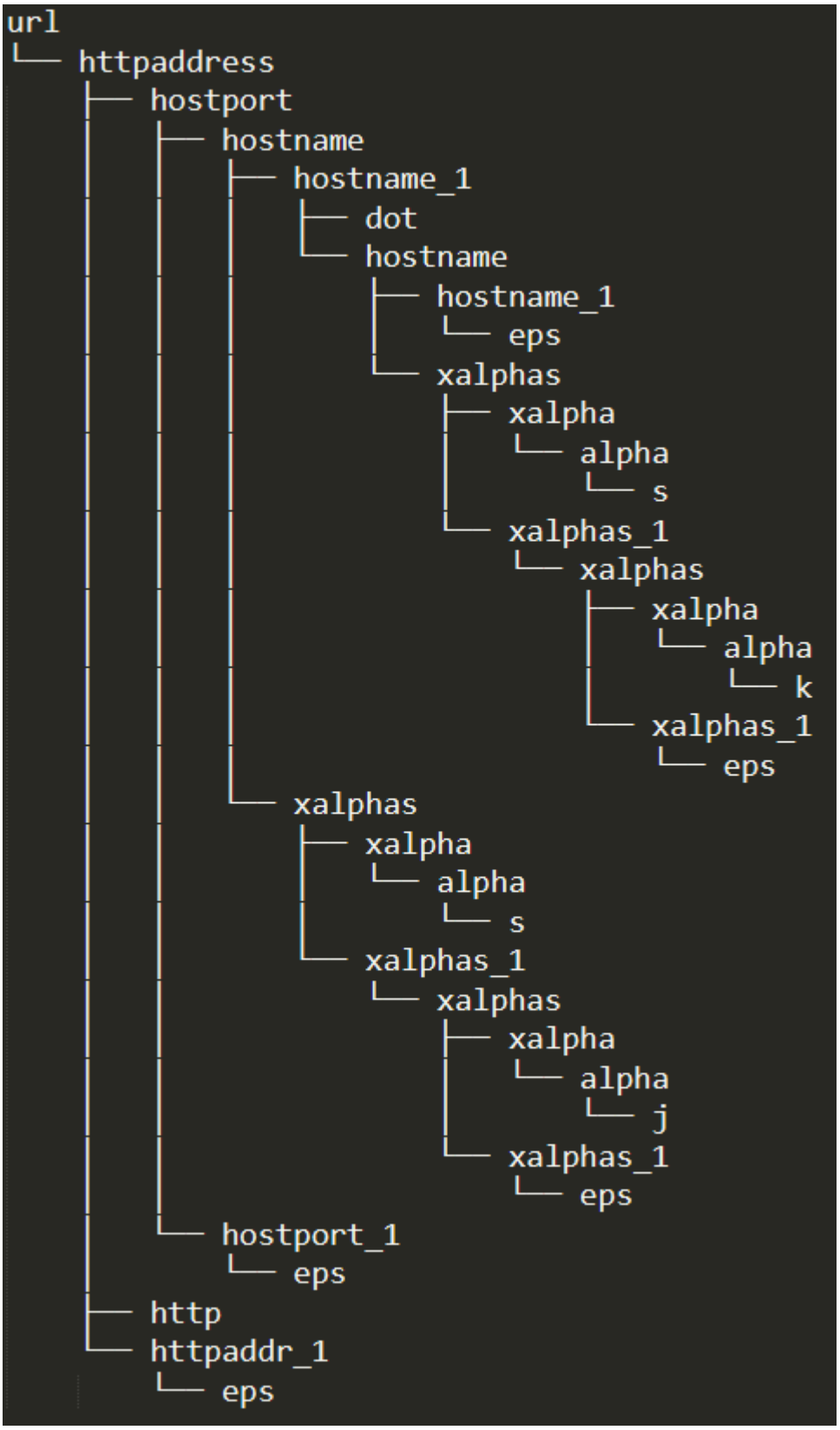
Prednastavené zobrazenie stromu nemá sortovanie (čiže jednotlivé vetvy stromu môžu byť v ľúbovoľnom poradí). Ak chceme mať strom odsortovaný podľa poradia, v ktorom sú pravidla jednoducho čítateľné, musíme v graphviz zápise špecifikovať **ordering=out** pre každý node grafu. Takto bude graf čítaný sprava doľava.

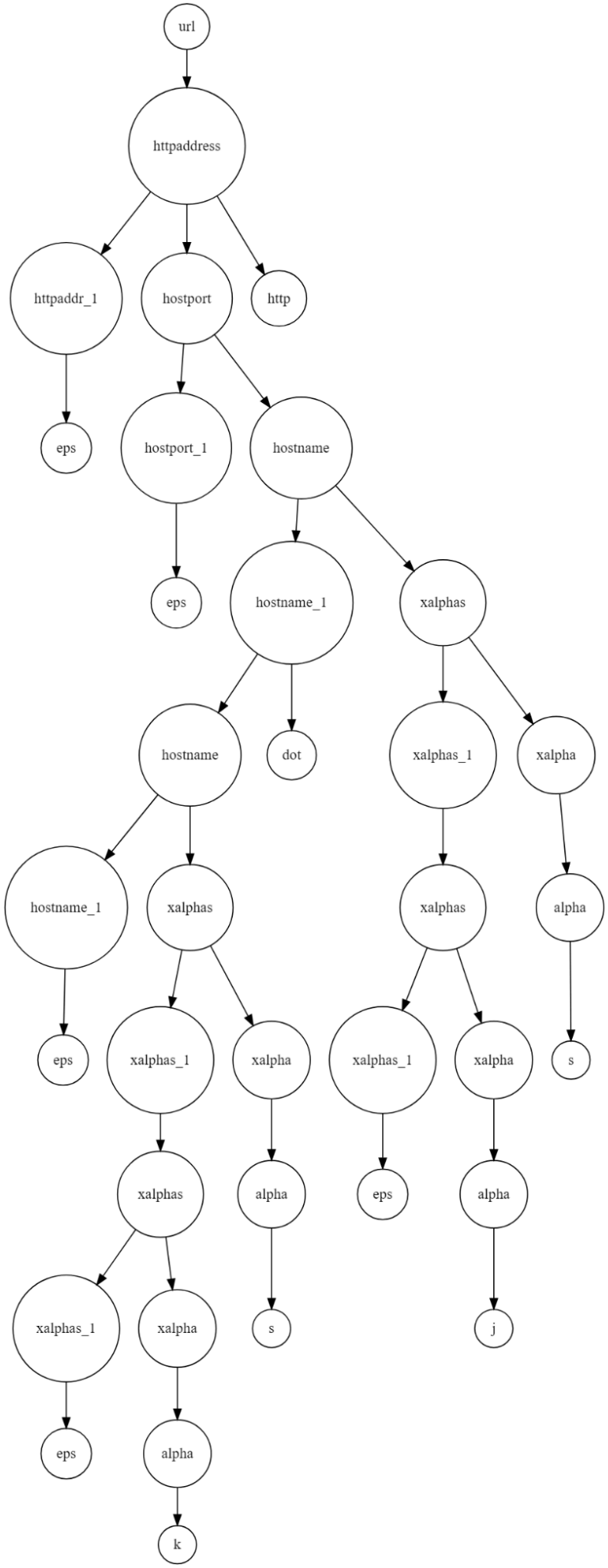
Namiesto toho, aby sme to robili manuálne, môžeme upraviť knižnicu **`treelib`**, ktorá nám vytvára syntaktický strom.

1. Prejdeme do súboru **`tree.py`** v priečinku v ktorom je nainštalovaná knižnica treelib; v prípade využívania package manageru `conda` je cesta `C:\Users\`*`username`*`\anaconda3\Lib\site-packages\treelib\`**`tree.py`**
2. Vo funkcii `to_graphviz()` zmeníme riadok `1100` na nasledovný kód (v podstate iba doplníme `ordering=out` do špecifikácie jedného node):

```
state = '"{0}" [label="{1}", shape={2},
ordering=out]'.format(nid, self[n].tag, shape)
```

Výpis syntaktického stromu v termináli a následne pomocou graphviz zobrazovača:




url
httpaddress
httpaddr_1
hostport
http
eps
hostport_1
hostname
eps
hostname_1
xalphas
hostname
dot
xalphas_1
xalpha
hostname_1
xalphas
xalphas
alpha
eps
xalphas_1
xalpha
xalphas_1
xalpha
s
xalphas
alpha
eps
alpha
xalphas_1
xalpha
s
j
eps
alpha
k

Záver

V tomto projekte sme navrhovali lexikálny a syntaktický analyzátor, ktorý rozpoznáva rôzne typy URL podľa pravidiel definovaných v zadaní. Pravidlá sme upravili tak, aby gramatika bola typu LL(1). Pomocou množín FIRST a FOLLOW sme vytvorili parsovaciu tabuľku, pomocou ktorej sme v Pythone implementovali syntaktickú analýzu. Lexikálnu analýzu sme implementovali pomocou konečného automatu. Pre každú analýzu máme dva druhy zotavenia z chýb. Na konci program vykreslí aj syntaktický strom. Riešenie sme overovali na niekoľkých príkladoch správnych a chybných URL adries.