Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Фізико-технічний факультет
Кафедра комп’ютерної інженерії та електроніки

# Реферат

на тему:

Генератор парсерів APG

Виконав:
Студент КІ-31
Павлишин А.В

м. Івано-Франківськ
2020р.

# Зміст

# 1. Огляд APG

APG спочатку був розроблений для генерування парсерів рекурсивного спуску безпосередньо із специфікації граматики ABNF для контекстно-вільних мов. Підхід повинен був визнати, що ABNF визначає сім незалежних операцій і що кожна з цих операцій може утворювати вузол дерева розбору рекурсивного спуску. Однак APG швидко розвинувся від парсінгу строго контекстно-вільних мов, описаних ABNF кількома значущими способами.

Спочатку це було через розбірливість. Речення контексно-вільної мови може мати більше одного дерева розбору, яке правильно розбирає речення. (Тобто різні способи фразування одного і того ж речення.) APG роз'єднує, вибираючи з множини лише одне чітко визначене дерево розбору.

По-друге, швидко стало зрозуміло, що цей спосіб визначення дерева розбору операторів вузлів жодним чином не вимагає, щоб оператори відповідали операціям ABNF. Їх можна було б розширити і вдосконалити будь-яким способом, який може бути зручним для існуючої проблеми. Перша з цих модифікацій вузла полягала в додаванні синтаксичних предикатів . Звідси було легко додати визначений користувачем термінал (User Defined Terminal). Тобто операція з вузлом, яку пише вручну користувач для конкретної проблеми співставлення фрази. Пізніше версія JavaScript додає зворотні посилання та інші оператори, щоб допомогти механізму pattern-matching, побудованому на ньому. Ці додаткові оператори розширюють основні оператори ABNF, але не змінюють їх, в результаті чого виникає надмножина ABNF або як часто тут називають SABNF.

Сьогодні APG - це універсальний, добре перевірений генератор парсерів на C / C ++, Java та JavaScript. Оскільки він заснований на ABNF, він особливо добре підходить для розбору мов багатьох технічних специфікацій Інтернету і, власне, є парсером вибраним багатьма великими телекомунікаційними компаніями.

# 2. Реалізації APG

**C/C++ –**6.3 версія APG повністю написана мовою C. Він має невеликий розмір, працює швидко і надійно. Через це і те, що він приймає граматики ABNF, це ідеальний вибір для стеків Інтернет протоколів. Дійсно, ряд великих телекомунікаційних компаній використовують APG у своїх комерційних продуктах для пасингу MEGACO, SIP та інших протоколів зв'язку. Для C ++ користувачів версія 6.3 надає обгортки C ++ для своїх C функцій. Згенерований код включає "початковий набір" функцій для ініціалізації правил та списків зворотних викликів UDT(User defined terminatls) та шаблонів для самих функцій зворотного виклику. Для нового проекту розкоментування та використання цих

функцій шаблону може забезпечити швидкий початок, особливо для граматик з великою кількістю правил.

**Java –** версія APG та парсерів, які він генерує, повністю написана мовою Java. Це дозволяє розробляти проект у повністю Java-середовищі. Крім того, завдяки Ralf Handl та проекту OASIS OData, Java APG має три розширені функції.

- Інкрементальна альтернатива (/ =) реалізована.
- Він приймає кілька вхідних граматичних файлів.
- У ньому є додаткові літерали, що чутливі до регістру які використовують одинарні лапки.

**Javascript** – повна версія APG JavaScript у форматі. Він може працювати в node.js або додатках для веб-сторінок. apg.html - це версія GUI для легкої розробки граматики SABNF. Він замінює "Інтерактивний APG".

## 3. Основи парсінгу

ABNF описує названі фрази, фраза не є не що інше, як масив кодових символів (цілих чисел). Визначені названих фраз називають правилами. Наприклад, розглянемо правило,

```
phrase = "abc"
```

Тобто, парсер ABNF відповідатиме правилу з назвою "phrase" з масивом кодів символів,

```
[97, 98, 99]
```

Повний набір операцій описаний нижче, але тут ми лише занотуємо оператори:

/ - альтернативні фрази, відповідати будь-якій фразі(аналог | з regex)

(пробіл) – конкатенація

name - відповідати фразі, визначеній названим правилом.

У сукупності набір правил визначатиме повний набір фраз та символи алфавіту, що складають мову. Речення мови, самі по собі, є лише фразами, визначеними першим, або початковим правилом - ім'ям правила, яке утворює кореневий вузол дерева розбору.

Ми можемо проілюструвати це простим прикладом.

```
SENTENCE = (RED / GREEN) (PAPER / LEAVES)
RED = "red "
GREEN = "green "
PAPER = "paper"
LEAVES = "leaves"
```

Ця граматика визначає алфавіт:

```
{" ", "a", "d", "e", "g", "l", "n", "p", "r", "s", "v"}
або алфавіт в ASCII кодах:
{32,65,68,69,71,76,78,80,82,83,86,97,100,101,103,108,110,112,114,115,118}
```

Фрази:

```
{"red ", "green ", "paper", "leaves"}
```

Мову(набір речень):

```
{"red paper", "red leaves", "green paper", "green leaves"}
```

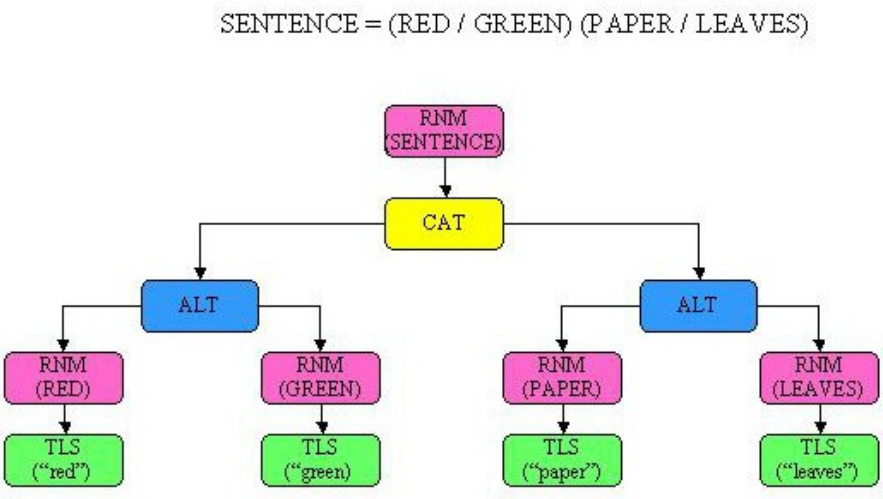


**Рис. 1** Синтаксичне дерево.

Рекурсивно-спусковий аналізатор робить перший обхід вузлів. Тобто, починаючи з верху, кореневого вузла, обхід рухається вниз до самого лівого, раніше непройденого вузла. Від кінцевих вузлів напрямок повертається назад до гілки.

Операція на кожному вузлі полягає в тому, щоб дати інструкції парсеру а) що робити і б) куди йти далі. Наприклад, при зверненні зверху кожен вузол ALT або CAT просто дає вказівку аналізатору перейти до самого лівого, непройденого вузла внизу. При зверненні знизу, вузли ALT вибирають із альтернативних фраз, а вузли CAT з'єднують фрази. Вузли RNM, коли звертаються зверху, розгортають іменоване правило, щоб сформувати під-дерево в кореневому вузлі дерева розбору. При зверненні знизу, вони передають відповідні фрази вгору дерева.

Але термінальні вузли мають набагато простіші операції. Вони просто намагаються зіставити заздалегідь задану фразу і повернути її до батьківського вузла. Тут варто зазначити, що APG з самого початку розроблявся як синтаксичний парсер. Кінцеві вузли не є односимвольними розпізнавачами. Це розпізнавачі, написані від руки, які працюють, або не впізнають всю фразу, яка їх визначає.

Взагалі можна сказати, що термінальні вузли - це розпізнавачі фраз, а нетермінальні - це маніпулятори фразою.

# 4. ABNF оператори

Це буде короткий опис операторів BNF.

Оригінальні оператори ABNF:

- ALT (/): альтернатива. Будь-яка з альтернативних фраз є прийнятною. Якщо знайдено відповідність двом або більше заступникам, фразування відповідного речення неоднозначне. APG вирішує двозначність із правилом "виграє перше зіставлення" (також згадується як "пріоритетний вибір"). Тобто, альтернативи перевіряються зліва направо і перше знайдене зіставлення зберігається, і всі інші альтернативи ігноруються.
- CAT (пробіл): конкатенація. Всі фрази сполучені. Якщо збіг з будь-якою фразою конкатенації не вдається, CAT повертає негативний результат.
- REP (n * m): повторення. REP досягає успіху, якщо визначена фраза відповідає мінімум "n" разів або до максимуму "m" разів. (примітка: *, n * або * m за замовчуванням встановлює "n = 0" і "m = infinity".) Пов'язане з правилами розбірливості для ALT, оператор повторення "жадібний", це означає, що він завжди зіставлятиме найдовшу можливу стрічку. Він ніколи не відступає, щоб побачити, чи можливе коротше зіставлення. З цієї причини правило, A = * "a" "a" не збігається з рядком "aaa", навіть якщо суворо контекстно-вільний парсер мав би успіх.
- RNM (name): назва правила. Зіставлення фрази, визначеної назвою правила.
- TLS ("string"): нечутливий до регістру термінал. Зіставляє визначений рядок.
- TBS (% d98.99): термінал, залежний від регістру. Зіставляє визначений рядок.
- TRG (% d48-57): діапазон символів. Зіставляє якщо будь-який символ у діапазоні.

Додаткові оператори суперсети:

- I (&): позитивний попередній перегляд. Розглядає фразу яка стоїть після &. Позитивний результат при матчі, негативний в іншому випадку. Зауважте, що цей оператор ніколи не споживає символів.

- HE (!): Негативний попередній перегляд. Розглядає фразу яка стоїть після !. Негативний результат при матчі, позитивний в іншому випадку. Зауважте, що цей оператор ніколи не споживає символів.
- UDT u_name: користувацько-визначені термінали. Цей оператор викликає написану користувачем функцію, щоб виконати зіставлення фрази.

Тільки JavaScript:

- BKA (&&): позитивний ретроспективний погляд. Розглядає фразу яка стоїть перед &&. Позитивний результат при матчі, негативний в іншому випадку. Зауважте, що цей оператор ніколи не споживає символів.
- BKN (!!): негативний ретроспективний погляд. Розглядає фразу яка стоїть перед !!. Негативний результат при матчі, позитивний в іншому випадку. Зауважте, що цей оператор ніколи не споживає символів.
- BKR (\name): зворотні посилання. Відповідає фразі, яка раніше відповідала названому правилу.
- ABG (% ^): початок якоря рядка. Відповідає початковому положенню рядка. Відповідає лише позиції. Символи не споживаються.
- ABE (% $): кінець рядкового якоря. Відповідає кінцевому положенню рядка. Відповідає лише позиції. Символи не споживаються.

# 5. Функції зворотного виклику назв правил(Rule name callbacks)

Функції зворотного виклику назв правил дозволяють користувачеві взаємодіяти з парсером при кожній визначеній фразі. Вузли RNM(Rule name callbacks) викличуть функцію, написану користувачем двічі для кожного вузла, спочатку на низхідному обході вузла до того, як буде зроблено будь-який розбір фрази, а потім знову на висхідному обхіді після того, як буде виконана робота з розпізнавання фраз парсера для вузла . Існує кілька типів дій, за допомогою яких правила зворотного виклику можуть здійснюватись.

- Низхідний обхід (попередній парсинг гілки під вузлом): Адміністративна робота, означає будь-який тип таблиці чи буфера даних або аналогічні обробки даних, які не впливають на розбір рядка. Обчислення, які робляться є паралельними процесу розбору.
- Низхідний обхід: повністю перевизначте Парсер. Тобто, зробіть власне зіставлення фрази і повністю пропустіть розбір(парсинг). У цьому випадку правило зворотного виклику функціонує як UDT. Однак реалізація UDT таким чином не очевидна і трохи незграбна. UDT були розроблені для подолання цього недоліку.

- Висхідний обхід (пост-розбір гілки внизу): знову ж таки, будь-яка адміністративна робота, яка може бути зручною щоб виконатись паралельно процесу розбору.
- Висхідний обхід: Розгляд результат зіставлення фрази парсера і вирішення, приймати чи відхиляти його. Наприклад, можуть бути вимоги, що не належать до контекстно-вільної граматики, які потрібно виконувати. Це може бути зроблено шляхом відхилення, повернення не зіставлення замість зіставлення. Або у вас є ще один шанс, щоб повністю перевизначити розпізнавання фрази та повернути якусь фразу, крім тієї, яку знайшов парсер.

# 6. Визначені користувачем термінали(User-Defined Terminals)

Давайте трохи перепишемо приклад граматики вище.

```
SENTENCE = (RED / GREEN) (PAPER / u_LEAVES)
RED = "red "
GREEN = "green "
PAPER = "paper"
```

Тепер ABNF термінал LEAVES був замінений на визначений користувачем термінал u_LEAVES. Оскільки підкреслення не є дійсним символом в ABNF, генератор парсера не має проблем з розпізнаванням його як UDT(User-Defined Terminal). А оскільки він написаний користувачем, для його визначення не потрібно ніякого правила ABNF.

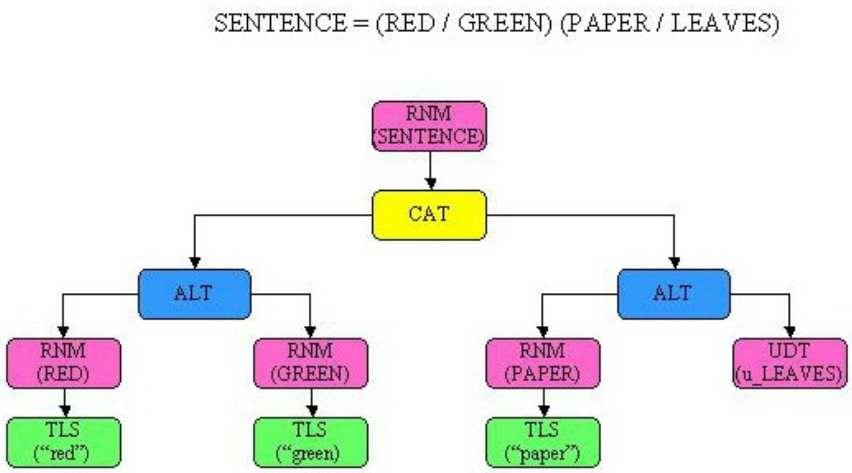


**Рис. 2** Синтаксичне дерево з користувацько-визначеним терміналом.

Як випливає з назви, це термінальний вузол. Єдина його операція - викликати написану користувачем функцію розпізнавання фрази та повертати

результат на батьківський вузол. Посібники користувача мають детальну інформацію щодо прототипу цих функцій та способів їх ідентифікації до Парсера.

u_LEAVES має деякі "безпечні" використання та деякі "небезпечні". Обидва є дуже потужними вдосконаленнями парсера.

"Безпечним" використанням було б записати його так, щоб воно просто розпізнавало фразу "листя" лише оптимізовано робити це набагато швидше, ніж це робить термінал ABNF. Це використання не змінює мови, яку граматика жодним чином визначає. Це просто робиться набагато швидше. Наприклад, у прикладі SIP, випущеному з кодом APG, вдалося збільшити швидкість розбору в 10 разів, не змінюючи жодним чином мову, визначену оригінальною граматикою ABNF.

Наприклад, "небезпечним" вживанням було б, наприклад, те, щоб воно робило ретросективний перегляд рядка речення, а якщо воно бачить "red", то воно розпізнає лише "faced", а якщо бачить "green", воно розпізнає лише "cheese". Тобто це повністю змінює мову, яку розпізнає парсер у контекстно-залежному вигляді.

Пошук дискусій рукописних аналізаторів викличе багато-багато критики до них. Основна скарга полягає в тому, що взагалі немає способу дізнатися, яку саме, мову розбирають. Але рукописні аналізатори також мають дві важливі переваги. Перше, що рукописні аналізатори, як правило, набагато швидші. Друга полягає в тому, що вони здатні розбирати контекстно-вільні граматики. Напевно, через обидві ці причини компілятор gcc GNU насправді є рукописним, рекурсивно-спусковим парсером. І саме з обох цих причин UDT були внесені до APG.

Для резюме, існує три типи використання, передбачених для UDT.

• "безпечний" - просто прискорити процес розбору.

• "небезпечно" - включити кілька вимог, що не належать до контекстно-вільних граматик, до граматики, яка є переважно контекстно-вільною.

• "абсолютно небезпечно" - просто надати рекурсивно-спускову основу повністю рукописному парсеру. UDT зазвичай є кінцевими вузлами. Однак в APG є можливість, яка дозволяє їм рухатись по синтаксичному дереві. Дивіться посібники користувача щодо функцій парсерів vExecuteRule () та vExecuteUdt (). Ви дійсно повинні знати, що ви робите, перш ніж використовувати їх, але ви можете створити аналізатор для абсолютно довільної мови без граматики ABNF, яка б керувала нею.

Імена UDT можуть починатися з префікса "u_" або "e_". Різниця між ними важлива. Для того, щоб генератор парсерів міг визначити атрибути граматики,

дуже важливо знати, які правила приймають порожні рядки, а які ні. Оскільки UDT написані користувачем, генератор парсерів не може знати, чи буде UDT чи не прийме порожні рядки. Тому було прийнято конвенцію, що лише UDT, імена яких починаються з "e_", можуть приймати порожні рядки. Тим, хто починається з "u_", заборонено приймати порожні рядки. Ця конвенція виконується парсером. Тобто, якщо функція, написана користувачем, наприклад, u_myfunc (), повертає нульову фразу довжиною, Парсер видасть помилку терміналу і завершиться викликом до встановленого наразі обробника.

## 7. Атрибути

Одне з перших речей, які ви дізнаєтесь про аналізатори рекурсивного спуску - це те, що ви не можете розібрати ліво-рекурсивну граматику.

```
Start = Start "s" / "y"
```

Парсер буде повторюватися нескінченно (або більш практично, поки не буде видано stack overflow), перш ніж він коли-небудь розпізнає один символ однієї фрази. Ліва рекурсивність тут вважається атрибутом граматики. Це важливий атрибут, про який слід знати, оскільки це фатальна вада в граматиці. Але є й інші, не менш важливі атрибути.

У наступних двох розділах нижче будуть визначені типи атрибутів та типи рекурсії, які вам знадобляться для розуміння деякої інформації, що відображається генератором парсерів. Однак, якщо ви хочете дізнатись, як це все робиться, вам доведеться прочитати код.

Визначення атрибутів:

- Ліво рекурсивні(фатальні): `Start = Start "s" / "y"`
  Фатальні як було написано вище.
- Право рекурсивні: `Start = "s" Start / "y"`
  Не фатальні , але добре знати.
- Вкладена рекурсивність: `Start = "s" Start "s" / "y"`
  Це не фатально, але важливо знати, оскільки саме це відрізняє контекстно-вільні граматики від звичайних граматик. Якщо цей атрибут хибний, граматика є регулярною, якщо правда, граматика є контекстно-вільна.
- Циклічні: `Start = Start`
  Правильний синтаксис граматики, але не визначає фраз.
- Нескінченні(фатальні): `Start = "s" Start`
  Це визначає лише нескінченні строкові фрази. Таким чином, аналізатор не може працювати для кожного кінцевого рядка або пропозиції.
- Пусті: `Start = "s" Start / ""`

Мова, описана цією граматикою, включає порожні рядки. Це не фатально, але знання того, чи можуть фрази правила бути порожніми чи ні, є критичним для визначення рекурсивних атрибутів.

Зауважте, що правило може мати більше одного справжнього атрибуту одночасно. Наприклад, наведені вище нескінченні та порожні правила також є право рекурсивними. Інша важлива річ, яку потрібно знати про сукупні атрибути. Вони означають лише те, що атрибут може трапитися, а не те, що це завжди стається. Наприклад, порожня граматика прикладу може приймати порожній рядок "", але також може приймати не порожній рядок "s".

## 8. Типи рекурсій

У відображених результатах генератора парсерів ви побачите, крім атрибутів, імена правил, перелічені під декількома різними типами. Це:

- N_RECURSIVE - нерекурсивний. Правило ніколи не посилається на себе.
- R_RECURSIVE - рекурсивний. Правило посилається на себе.
- MR_RECURSIVE - взаємно рекурсивний. Набори правил. У межах кожного набору кожне правило посилається на кожне інше правило в наборі, включаючи себе.
- NMR_RECURSIVE - нерекурсивний, але стосується одного або декількох взаємно-рекурсивних правил.
- RMR_RECURSIVE - рекурсивний, але також відноситься до одного або декількох взаємно-рекурсивних наборів правил, членом яких він не є.

Причини цих відмінностей у рекурсивних типах правил полягають у тому, що вони відіграють важливу роль у визначенні атрибутів. Зокрема, взаємно-рекурсивні множини створюють ситуацію, catch-22, оскільки вони представляють неможливе завдання потребувати атрибутів одного члена набору для визначення атрибутів інших та навпаки.

## 9. apg-exp

Багато програмістів часто уникає використання регулярних виразів, коли це необхідно, тим чи іншим способом. Для багатьох синтакс регулярних виразів є незрозумілим і відштовхуючим. Проте існує інший рушій зіставлення паттернів – apg-exp це  альтернатива регулярним виразам з синтаксом ABNF паттернів який є більш читабельний.

Для порівняння регулярний вираз для підтвердження поштової адреси виглядає наступним чином:

```
^[\w!#$%&'*+/=?^_`{|}~-]+(?:\.[\w!#$%&'*+/=?^_`{|}~-]+)*@(?:[A-Z0-9-]+\.)+[A-Z]
{2,6}$
```

Він добре написаний та чудово працює, проте він:

- важко читабельний
- важкий для написання
- важкий для підтримки

Регулярні вирази мають довгу історію і глибоко інтегровані в інструменти і мови які використовують програмісти, щоденно.

Проте альтернативний синтаксис існує майже так само довго, а саме ABNF(Augmented Backus-Naur Form) визначений організацією IETF в стандартах RFC 5234 і RFC 7405.

Для порівняння розглянемо те саме підтвердження поштової скриньки в синтаксисі ABNF.

```
email-address   = local "@" domain
local           = local-word *("." local-word)
domain          = 1*(sub-domain ".") top-domain
local-word      = 1*local-char
sub-domain      = 1*sub-domain-char
top-domain      = 2*6top-domain-char
local-char      = alpha / num / special
sub-domain-char = alpha / num / "-"
top-domain-char = alpha
alpha           = %d65-90 / %d97-122
num             = %d48-57
special         = %d33 / %d35 / %d36-39 / %d42-43 / %d45 / %d47
                / %d61 / %d63 / %d94-96 / %d123-126
```

Цей запис не такий компактний як в попередньому прикладі проте, як і HTML, і XML він створений, щоб бути читабельним для людей і машин.

Розглянемо детальніше цей запис:

- електронна адреса визначена як локальна частина та домен, розділені символом @
- локальна частина - це одне слово, за яким слідують необов'язкові слова, розділені крапкою
- домен - це один або кілька розділених точкою і піддомени, за якими слідує один верхній домен

- Єдині речі, яких ви можете тут не знати, але, напевно, можете здогадатися, це:
    - подібно до того, як символ підказки * означає «нуль або більше», 1 * означає «один або більше», а 2 * 6 означає мінімум 2 і максимум 6 повторень
    - / розділяє альтернативні варіанти
    - % d визначає коди десяткових символів та діапазони коду символів
    - наприклад,% d35 являє собою #, ASCII десятковий код 35
    - % d65-90 представляє будь-який символ у діапазоні A-Z, десяткові коди ASCII 65-90

Інші приклади використання agp-exp.

Телефонні номери:

```
phone-number = ["("] area-code sep office-code sep subscriber
area-code    = 3digit                        ; 3 digits
office-code  = 3digit                        ; 3 digits
subscriber   = 4digit                        ; 4 digits
sep          = *3(%d32-47 / %d58-126 / %d9) ; 0-3 ASCII non-digits
digit        = %d48-57                       ; 0-9
```

Дати(формату місяць/день/рік):

```
date     = %^ (mm-first / dd-first) %$
mm-first = mm "/" dd "/" yyyy   ; month before day
dd-first = dd "/" mm "/" yyyy   ; day before month
dd       = "0" digit1           ;    01-09
         / ("1"/"2") digit      ; or 10-29
         / "3" %d48-49          ; or 30-31
         / digit1               ; or 1-9
mm       = "0" digit1           ;    01-09
         / "1" %d48-50          ; or 10-12
         / digit1               ; or 1-9
yyyy     = ("19" / "20") 2digit ; 1900-1999 or 2000-2099
         / 2digit               ; or 00-99
digit    = %d48-57              ; 0-9
digit1   = %d49-57              ; 1-9
```

Для порівняння цей самий підхід в регялярниxx виразах виглядає наступним чином:

```
^(?:((?:0[1-9])|(?:1[0-2])|(?:[1-9]))/((?:0[1-9])|(?:(?:1|2)[0-9])|
(?:3[0-1])|(?:[1-9]))|((?:0[1-9])|(?:(?:1|2)[0-9])|(?:3[0-1])|(?:[1-
9]))/((?:0[1-9])|(?:1[0-2])|(?:[1-9])))/((?:19|20)?[0-9][0-9])$
```

# 10. APG Java

Поточна версія APG Java є 1.0 вона доступна на Github за наступним посиланням: https://github.com/ldthomas/apg-java

Короткий опис:

- Java APG - APG - генератор парсерів ABNF, повністю написаний мовою Java.
- генерує парсери на Java
- генерує мовні парсери та транслятори із набору синтаксису граматики Augmented Backus-Naur Form (ABNF RFC5234)
- приймає дійсні граматики ABNF
- приймає синтаксичні оператори предикатів AND & NOT для умовного синтаксичного аналізу на основі визначених фраз попереднього перегляду.
- приймає визначені користувачем термінали (UDT), які дозволяють, користувацькі оператори розпізнавання контекстно-вільних фраз.
- призначені користувачем функції зворотного виклику забезпечують повний моніторинг та контроль потоку парсера
- нещодавно розширено, щоб прийняти чутливі до регістру рядки в одних лапках
- нещодавно розширений для прийому декількох вхідних файлів граматики ABNF
- необов'язкове генерування абстрактного синтаксичного дерева (AST)
- переклад AST з написаними користувачем функціями зворотного виклику
- обширні засоби відстеження
- збір статистики для повного зображення покриття дерева розбору
- обширна генерація атрибутів для огляду характеристики граматики

Файлова структура проекту:

| **Директорії/файли** | **Опис** |
|---|---|
| src/apg | Java APG, генератор і бібліотека необхідна всім згенерованим парсерам. |
| src/examples | Приклади, які демонструють, як налаштувати та використовувати аналізатор Java APG. Основна функція тут - драйвер для запуску будь-якого з усіх наведених нижче прикладів. |
| src/examples/anbn | Порівняння CFG і UDT парсерів для граматик $a^n b^n$, n>0. |
| src/examples/anbncn | Порівняння CFG і UDT парсерів для $a^n b^n c^n$, n>0. |
| src/examples/demo | Демонстрація багатьох головних можливостей Java APG включаючи UDTs(User-defined terminals) |
| src/examples/ | Порівняння CFG і UDT парсерів граматики виразів |

| | |
|---|---|
| expressions | |
| src/examples/inifile | Порівняння CFG і UDT парсерів для граматики ini файлів. |
| src/examples/mailbox | Порівняння CFG і UDT парсерів для граматики поштової скриньки. |
| src/examples/testudtlib | Порівняння CFG і UDT парсерів для набору Udtlib. |
| Build/ | Скрипти для компіляції і документування Java APG. |
| LICENSE | Версія 2 публічної ліцензії GNU. |
| README.md | Файл з описом проекту. |

Встановлення:

build/ директорія має скрипти та файли для компіляції та документації Java APG. Скрипти написані на bash і тому призначені для Unix-подібних ОС тому наступні інструкції потрібно виконувати в Linux/Mac OS.

Щоб скомпілювати весь сирцевий код і створити .jar пакет потрібно:

1. Завантажити проект з Github за посиланням – https://github.com/ldthomas/apg-java як zip файл
   Або командою: git clone https://github.com/ldthomas/apg-java.git
2. Далі запустити процес компіляції виконавши команди:
   ```
   cd <директорія проекту>
   cd build
   ./make-jars
   ```

Вивід компіляції:

```
Note: ../src/apg/Trace.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
make-jars: apg source compiled ok
make-jars: created apg.jar
Note: ../src/apg/Trace.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
make-jars: examples source compiled ok
make-jars: created examples.jar
```

Це створить файли apg.jar і examples.jar.

Щоб протестувати agp.jar виконайте команду:

```
java -jar apg.jar /version
```

Вивід команди:

```
Java APG Version 1.0, released 10/23/2011
Copyright (c) 2011 Lowell D. Thomas, all rights reserved
```

Licence Notification

……..

……..

Щоб протестувати agp.jar виконайте команду:

```
java -jar examples.jar /help
```

Щоб побачити документацію виконайте команду:

```
./make-javadoc
```

І відкрийте `../javadoc/index.html` в будь-якому браузері.

Запуск прикладу порівняння CFG і UDT парсерів для граматики поштової скриньки:

```
java -jar examples.jar /test=mailbox
```

Вивід команди дуже громіздкий він буде поділений на окремі частини.

Заголовок:

```
TEST NAME: mailbox
ABSTRACT: CFG/UDT time & statistics comparison for the email address
grammar
```

CFG граматика:

```
CFG GRAMMAR: (mailbox)
;
; examples.mailbox.Mailbox
;
;
; From RFC 5321
; Rules not referenced by Mailbox removed
; "atext" taken from RFC 5322
; Core rules ALPHA, DIGIT, etc. from RFC 5234
;
;
Mailbox        = Local-part "@" ( Domain / address-literal )
Domain         = sub-domain *("." sub-domain)
;sub-domain     = Let-dig [Ldh-str]
sub-domain     = 1*Let-dig *Ldh-str
Let-dig        = ALPHA / DIGIT
;Ldh-str        = *( ALPHA / DIGIT / "-" ) Let-dig
Ldh-str        = *"-" 1*Let-dig
address-literal  = "[" ( IPv4-address-literal /
                    IPv6-address-literal /
                    General-address-literal ) "]"
```

```
                    ; See Section 4.1.3
Local-part     = Dot-string / Quoted-string
                  ; MAY be case-sensitive
Dot-string     = Atom *("."  Atom)
Atom           = 1*atext
Quoted-string  = DQUOTE *QcontentSMTP DQUOTE
QcontentSMTP   = qtextSMTP / quoted-pairSMTP
quoted-pairSMTP  = %d92 %d32-126
                    ; i.e., backslash followed by any ASCII
                    ; graphic (including itself) or SPace
qtextSMTP      = %d32-33 / %d35-91 / %d93-126
                  ; i.e., within a quoted string, any
                  ; ASCII graphic or space is permitted
                  ; without blackslash-quoting except
                  ; double-quote and the backslash itself.
IPv4-address-literal  = Snum 3("."  Snum)
IPv6-address-literal  = "IPv6:" IPv6-addr
General-address-literal  = Standardized-tag ":" 1*dcontent
Standardized-tag  = 1*Ldh-str
                     ; Standardized-tag MUST be specified in a
                     ; Standards-Track RFC and registered with IANA
dcontent       = %d33-90 / ; Printable US-ASCII
                  %d94-126 ; excl. "[", "\", "]"
Snum           = 1*3DIGIT
                  ; representing a decimal integer
                  ; value in the range 0 through 255
IPv6-addr      = IPv6-full / IPv6-comp / IPv6v4-full / IPv6v4-comp
IPv6-hex       = 1*4HEXDIG
IPv6-full      = IPv6-hex 7(":" IPv6-hex)
IPv6-comp      = [IPv6-hex *5(":" IPv6-hex)] "::"
                  [IPv6-hex *5(":" IPv6-hex)]
                  ; The "::" represents at least 2 16-bit groups of
                  ; zeros.  No more than 6 groups in addition to the
                  ; "::" may be present.
IPv6v4-full    = IPv6-hex 5(":" IPv6-hex) ":" IPv4-address-literal
IPv6v4-comp    = [IPv6-hex *3(":" IPv6-hex)] "::"
                  [IPv6-hex *3(":" IPv6-hex) ":"]
                  IPv4-address-literal
                  ; The "::" represents at least 2 16-bit groups of
                  ; zeros.  No more than 4 groups in addition to the
                  ; "::" and IPv4-address-literal may be present.
;
;RFC 5322
atext           =   ALPHA / DIGIT /    ; Printable US-ASCII
                       "!" / "#" /     ; characters not including
                       "$" / "%" /     ; specials.  Used for atoms.
                       "&" / "'" /
                       "*" / "+" /
                       "-" / "/" /
                       "=" / "?" /
                       "^" / "_" /
```

```
                      "`" / "{" /
                      "|" / "}" /
                      "~"
;
;RFC 5234                          ABNF                      January
2008
;B.1.  Core Rules
;
;   Certain basic rules are in uppercase, such as SP, HTAB, CRLF,
DIGIT,
;   ALPHA, etc.
;
ALPHA          =  %x41-5A / %x61-7A   ; A-Z / a-z
DIGIT          =  %x30-39
                      ; 0-9
DQUOTE         =  %x22
                      ; " (Double Quote)
HEXDIG         =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
```

UDT граматика:

```
UDT GRAMMAR: (mailbox)
;
; examples.mailbox.UMailbox
;
;
; From RFC 5321
; Rules not referenced by Mailbox removed
; "atext" taken from RFC 5322
; Core rules ALPHA, DIGIT, etc. from RFC 5234
;
;
Mailbox        = Local-part "@" Domain-part
Domain-part    = Domain / address-literal
Domain         = sub-domain *("." sub-domain)
;sub-domain     = Let-dig [Ldh-str]
;sub-domain     = 1*Let-dig *Ldh-str
sub-domain     = u_Let-dig e_Ldh-str
;Let-dig        = ALPHA / DIGIT
;Ldh-str        = *( ALPHA / DIGIT / "-" ) Let-dig
;Ldh-str        = *"-" 1*Let-dig
address-literal  = "[" ( IPv4-address-literal /
                   IPv6-address-literal /
                   General-address-literal ) "]"
                   ; See Section 4.1.3
Local-part     = Dot-string / Quoted-string
                 ; MAY be case-sensitive
Dot-string     = u_Atom *("."  u_Atom)
;Atom           = 1*atext
Quoted-string  = DQUOTE *QcontentSMTP DQUOTE
```

```
QcontentSMTP   = qtextSMTP / quoted-pairSMTP
quoted-pairSMTP  = %d92 %d32-126
                   ; i.e., backslash followed by any ASCII
                   ; graphic (including itself) or SPace
qtextSMTP      = %d32-33 / %d35-91 / %d93-126
                 ; i.e., within a quoted string, any
                 ; ASCII graphic or space is permitted
                 ; without blackslash-quoting except
                 ; double-quote and the backslash itself.
IPv4-address-literal  = u_Snum 3("."  u_Snum)
IPv6-address-literal  = "IPv6:" IPv6-addr
;General-address-literal  = Standardized-tag ":" 1*dcontent
General-address-literal  = u_Standardized-tag ":" u_dcontent
;Standardized-tag  = 1*Ldh-str
;Standardized-tag  = u_Ldh-str
                     ; Standardized-tag MUST be specified in a
                     ; Standards-Track RFC and registered with IANA
;dcontent       = %d33-90 / ; Printable US-ASCII
;                  %d94-126 ; excl. "[", "\", "]"
;Snum           = 1*3DIGIT
                 ; representing a decimal integer
                 ; value in the range 0 through 255
IPv6-addr      = IPv6-full / IPv6-comp / IPv6v4-full / IPv6v4-comp
;IPv6-hex       = 1*4HEXDIG
IPv6-full      = u_IPv6-hex 7(":" u_IPv6-hex)
IPv6-comp      = [u_IPv6-hex *5(":" u_IPv6-hex)] compression
                 [u_IPv6-hex *5(":" u_IPv6-hex)]
                 ; The "::" represents at least 2 16-bit groups of
                 ; zeros.  No more than 6 groups in addition to the
                 ; "::" may be present.
compression    = "::";
IPv6v4-full    = u_IPv6-hex 5(":" u_IPv6-hex) ":" IPv4-address-
literal
IPv6v4-comp    = [u_IPv6-hex *3(":" u_IPv6-hex)] "::"
                 [u_IPv6-hex *3(":" u_IPv6-hex) ":"]
                 IPv4-address-literal
                 ; The "::" represents at least 2 16-bit groups of
                 ; zeros.  No more than 4 groups in addition to the
                 ; "::" and IPv4-address-literal may be present.
;
;RFC 5322
;atext           =   ALPHA / DIGIT /    ; Printable US-ASCII
;                     "!" / "#" /     ; characters not including
;                     "$" / "%" /     ; specials.  Used for atoms.
;                     "&" / "'" /
;                     "*" / "+" /
;                     "-" / "/" /
;                     "=" / "?" /
;                     "^" / "_" /
;                     "`" / "{" /
;                     "|" / "}" /
```

```
;                       "~"
;
;RFC 5234                          ABNF                      January
2008
;B.1.  Core Rules
;
;   Certain basic rules are in uppercase, such as SP, HTAB, CRLF,
DIGIT,
;   ALPHA, etc.
;
;ALPHA          =  %x41-5A / %x61-7A   ; A-Z / a-z
;DIGIT          =  %x30-39
                       ; 0-9
DQUOTE         =  %x22
                       ; " (Double Quote)
;HEXDIG         =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
```

Вхідні тестові стрічки:

```
INPUT STRINGS: (mailbox)
[0] test@domain.com
[1] test.middle.last@domain.com
[2] test@dom-ain.com
[3] test@dom--ain.com
[4] test@n.com
[5] test_two@domain.com
[6] &quot;test_two&quot;@domain.com
[7] test@[1111:2222:3333:4444:5555:6666:7777:8888]
[8] test@[IPv6:1111:2222:3333:4444:5555:6666:7777:8888]
[9] test@[IPv6:1111:2222:3333:4444:5555:6666::8888]
[10] test@[IPv6:1111:2222:3333:4444:5555:255.255.255.255]
[11] test@[1.2.3.4]
[12] test@[01.002.0.000]
[13] test@[301.2.3.4]
```

Тестування CFG парсера:

```
CFG PARSER: TEST ALL STRINGS: (mailbox)
[0] success
```

```
[1] success
[2] success
[3] success
[4] success
[5] success
[6] success
[7] success
[8] success
[9] success
[10] success
[11] success
[12] success
[13] success
```

Тестування UDT парсера:

```
UDT PARSER: TEST ALL STRINGS: (mailbox)
[0] success
[1] success
[2] success
[3] success
[4] success
[5] success
[6] success
[7] success
[8] success
[9] success
[10] success
[11] success
[12] success
[13] success
```

Часове порівняння CFG і UDT парсерів:

```
TIME COMPARISON: (mailbox, reps=1000000)
string:       cfg       udt   cfg/udt
     0:      1479       376     3.934
     1:      2509       421     5.960
     2:      1536       315     4.876
     3:      1555       353     4.405
     4:      1193       301     3.963
     5:      1747       338     5.169
     6:      1792       849     2.111
     7:      2097       525     3.994
     8:      1875       861     2.178
     9:      2967       887     3.345
    10:      5640       899     6.274
    11:       973       446     2.182
    12:      1022       431     2.371
```

```
    13:       958       427     2.244
                   average:     3.786
```

Статистика вузлів CFG:

```
CFG: NODE STATISTICS: (mailbox,reps=1000000)
  operator      MATCH      EMPTY    NOMATCH       hits
       ALT        554          0        129        683
       REP        118         43         31        192
       AND          0          0          0          0
       NOT          0          0          0          0
       CAT        126          0         67        193
       TRG        404          0        347        751
       TBS          2          0          1          3
       TLS         92          0        388        480
       UDT          0          0          0          0
       RNM        873          0        251       1124
     -----      -----      -----      -----      -----
     TOTAL       2169         43       1214       3426

RNM node statistics: sorted by hit count, omitted if no hits
     MATCH      EMPTY    NOMATCH       hits  rule name
       196          0         74        270  DIGIT
       126          0         63        189  ALPHA
       169          0          4        173  HEXDIG
        66          0         41        107  Let-dig
        66          0         16         82  atext
        74          0          2         76  dcontent
        43          0          1         44  IPv6-hex
        14          0          7         21  sub-domain
         4          0         16         20  Ldh-str
        15          0          1         16  Atom
        13          0          3         16  Snum
        14          0          0         14  Mailbox
         7          0          7         14  Domain
        14          0          0         14  Local-part
        13          0          1         14  Dot-string
         8          0          1          9  QcontentSMTP
         8          0          1          9  qtextSMTP
         7          0          0          7  address-literal
         3          0          4          7  IPv4-address-literal
         2          0          2          4  IPv6-address-literal
         2          0          1          3  IPv6-addr
         1          0          2          3  IPv6-full
         2          0          0          2  General-address-literal
         2          0          0          2  Standardized-tag
         1          0          1          2  IPv6-comp
         2          0          0          2  DQUOTE
         1          0          0          1  Quoted-string
         0          0          1          1  quoted-pairSMTP
         0          0          1          1  IPv6v4-full
         0          0          1          1  IPv6v4-comp
```

Статистика вузлів UDT:

```
UDT: NODE STATISTICS: (mailbox, reps=1000000)
  operator      MATCH      EMPTY    NOMATCH       hits
       ALT         51          0          5         56
       REP         18         18          7         43
       AND          0          0          0          0
       NOT          0          0          0          0
       CAT         87          0         69        156
       TRG          8          0         19         27
       TBS          2          0          1          3
       TLS         53          0         40         93
       UDT         64         12         11         87
       RNM        109          0         44        153
     -----      -----      -----      -----      -----
     TOTAL        392         30        196        618

RNM node statistics: sorted by hit count, omitted if no hits
     MATCH      EMPTY    NOMATCH       hits  rule name
        14          0          7         21  sub-domain
        14          0          0         14  Mailbox
        14          0          0         14  Domain-part
         7          0          7         14  Domain
        14          0          0         14  Local-part
        13          0          1         14  Dot-string
         8          0          1          9  QcontentSMTP
         8          0          1          9  qtextSMTP
         7          0          0          7  address-literal
         3          0          4          7  IPv4-address-literal
         0          0          4          4  IPv6-address-literal
         4          0          0          4  General-address-literal
         0          0          3          3  IPv6-addr
         0          0          3          3  IPv6-full
         0          0          3          3  IPv6-comp
         0          0          3          3  compression
         0          0          3          3  IPv6v4-full
         0          0          3          3  IPv6v4-comp
         2          0          0          2  DQUOTE
         1          0          0          1  Quoted-string
         0          0          1          1  quoted-pairSMTP

UDT node statistics: sorted by hit count, omitted if no hits
     MATCH      EMPTY    NOMATCH       hits  rule name
        14          0          7         21  u_let-dig
        15          0          1         16  u_atom
        13          0          3         16  u_snum
         2         12          0         14  e_ldh-str
        12          0          0         12  u_ipv6-hex
         4          0          0          4  u_standardized-tag
         4          0          0          4  u_dcontent
```

Завершення тесту:

test complete: mailbox

# Список використаних джерел


1. APG – an ABNF parser generator [Електронний ресурс]. – Режим доступу: https://www.coasttocoastresearch.com/overview
2. APG documentation [Електронний ресурс]. – Режим доступу: https://www.coasttocoastresearch.com/documentation
3. APG-exp [Електронний ресурс]. – Режим доступу: https://www.coasttocoastresearch.com/apgexp
4. Сторінка проекту на Github [Електронний ресурс]. – Режим доступу: https://github.com/ldthomas/apg-java
5. Alfred V. Aho, Ravi Sethi, Monica S. Lam and Jeffrey D. Ullman, "Compilers: Principles, Techniques, and Tools", 2nd Edition, Addison-Wesley, 2007
6. John E. Hopcroft and Jeffrey D. Ullman, "Introduction to Automata Theory, Languages and Computation", Addison-Wesley, 1979