ПЛОВДИВСКИ УНИВЕРСИТЕТ „ПАИСИЙ ХИЛЕНДАРСКИ“
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
КАТЕДРА “КОМПЮТЪРНА ИНФОРМАТИКА”

Николай Георгиев Ханджийски

Итеративен парсиращ алгоритъм с приложение в профилирането на парсери

АВТОРЕФЕРАТ

на дисертационен труд
за присъждане на образователна и научна степен „Доктор“

Област на висше образование:
4. Природни науки, математика и информатика
Професионално направление:
4.6 Информатика и компютърни науки
Докторска програма: „Информатика“

научен ръководител
проф. д-р Елена Сомова

Пловдив
2024

Дисертационният труд е обсъден и насочен за защита пред научно жури, на заседание на катедра „Компютърна информатика“ при Факултета по математика и информатика на Пловдивския университет „Паисий Хилендарски“, на 28.02.2024 г.

Дисертационният труд „Итеративен парсиращ алгоритъм с приложение в профилирането на парсери“ съдържа 163 страници. Списъкът на използваната литература включва 191 източника. Списъкът на авторските публикации по темата се състои от 6 заглавия.

Материалите по защитата са на разположение в Деканата на Факултета по математика и информатика, Нова сграда на ПУ „Паисий Хилендарски“, каб. 330 всеки работен ден от 8:30 до 17:00 часа.

Автор: Николай Георгиев Ханджийски
Заглавие: Итеративен парсиращ алгоритъм с приложение в профилирането на парсери

Съдържание

Списък с използваните съкращения

Съкращение	Пълна форма	Страница/Източник
НПБНФ	Напреднала Подсилена Бекус-Наур Форма	13/Дисертационен труд
ПБНФ	Подсилена Бекус-Наур Форма	7/[5]
ТР	Tunnel Parsing	17/Дисертационен труд

Увод

Математическата основа на транслаторите, а следователно и на езиците за програмиране, е теорията на формалните езици и обработващите ги абстрактни машини. Понеже всеки език за програмиране е формален език, то е удачно и общоприето езиците за програмиране или части от тях (лексика, синтаксис, семантика и т.н.) да се описват с пораждащи граматики, а части от самия процес на транслация (лексикален/лексически, синтактичен/синтактически и семантичен анализи) да се описват с различни класове абстрактни разпознаватели (крайни автомати, автомати със стек, машини на Тюринг и др.).

Настоящият дисертационен труд е посветен на два от важните елементи на всеки транслатор, а именно лексикалния и синтактичния анализи. В целия дисертационен труд, ако не е казано друго, под „данни“ ще се разбират данни, обработвани от транслатор.

Проверката дали конкретни данни принадлежат на даден език е обект на лексикалния и синтактичния анализи и се нарича разпознаване. Парсирането (синтактичният анализ) на конкретните данни е тяхното разпознаване и извеждането на конкретна структурна информация за тях.

Развитието на алгоритмите свързани с разпознаването и парсирането на данни се прави от много автори, като приликите и разликите между алгоритмите не винаги са незабавно видни. Известните алгоритми за разпознаване и парсиране на данни имат различни предимства и недостатъци.

Съществуват компютърни програми, които автоматично генерират програмния код на парсер въз основа на зададена граматика, който код е написан на конкретен език за програмиране. Този код може да се използва за парсирането на произволни данни и за извеждането на съответната структура. Изведените структури от данни могат се използват както за по-нататъшно компилиране, така и за анализ.

Дисертационната работа включва проектирането на машина за парсиране на данни, наречена парсираща машина (parsing machine), в която могат да се използват различни алгоритми за парсиране. Ползите от тази парсираща машина са, че обединява различни подходи/алгоритми за анализ и добавя някои възможности, липсващи в известните в литературата конкретни парсери.

Важна част от дисертацията е и проектирането нов алгоритъм за парсиране, наречен Тунелно парсиране (Tunnel Parsing - ТР), който е вграден в парсер модула на парсиращата машина и се възползва от добавените възможности в парсиращата машина.

Част от тази дисертация е насочена и към изследването на клас от многозначни граматики, същински подклас на многозначните безконтекстни граматики, пораждащи данни, за който споменатото по-горе Тунелно парсиране се извършва за линейно време.

Друг съществен елемент на работата е разработката на инструмент за измерване на ресурсите, които се използват по време на разпознаване/парсиране на данни (породени от различни безконтекстни граматики) от автоматично генерирани парсери с различни парсер генератори и компилатори, наречен профилатор на парсер генератори (Parser Generator Profiler), накратко профилатор.

Цел и задачи на дисертационния труд

Основната цел на дисертационното изследване е да се изследват, предложат, проектират, разработят, експериментално приложат и апробират средства (машини, алгоритми, модели, езици и инструменти), които са подходящи за линейно транслиране на данни на базата на някои многозначни безконтекстни граматики.

За постигане на поставената цел на дисертационното изследване бяха планирани следните шест основни задачи:

Задача 1. Проучване на теории, формални средства, подходи, методи, алгоритми, модели, архитектури, машини и системи, които са свързани с транслирането на данни;

Задача 2. Проектиране на обща архитектура на парсираща машина, която да обединява различни подходи/алгоритми за лексически и синтактичен анализи с някои добавени нови възможности;

Задача 3. Дефиниране на нов вид безконтекстни граматики, равномощни на безконтекстните граматики, съобразени с предложената обща архитектура на парсираща машина;

Задача 4. Проектиране на нов алгоритъм за парсиране, вграден в парсиращата машина, който използвайки добавените възможности в нея може да парсира данни на базата на новия вид граматики, когато в тях няма лява рекурсия;

Задача 5. Проектиране и реализиране на прототип на инструмент за измерване и сравняване на използваните ресурси от различни парсери, включващ автоматично създаване на граматики, написани на специално създаден език за метапрограмиране, както и създаване на парсери за тези граматики с различни парсер генератори и компилатори;

Задача 6. Осъществяване на експерименти с помощта на създадения инструмент.

Структура на дисертацията

Дисертацията се състои от списъци с таблици и фигури, увод, четири глави, заключение, списък на авторските публикации по темата, списък със забелязани цитирания, приложения, списък с използваната литература и декларация за оригиналност.

Основния текст на дисертацията се състои от 163 страници и е съпроводен от 2 (две) приложения (2 страници).

В глава 1 Обзор се съдържат теории на формалните езици и обработващите ги абстрактни машини, свързани с транслирането. Дискутирани са елементите на един транслатор, които извършват лексикален и синтактичен анализи, както и пораждащите граматики, които се използват от тях. В главата се прави преглед на крайни автомати, стекови автомати, машини на Тюринг, алгоритми на Марков и др. Направеният обзор включва известни алгоритми за разпознаване и парсиране работещи на базата на безконтекстни граматики.

В глава 2 Парсираща машина се съдържат формални дефиниции за напреднали граматики, фразова машина, която се създава на базата на тези граматики, както и детайлно описание на парсираща машина. Главата съдържа дефинициите на езиците, които са дефинирани от напреднали граматики с напреднали символи. Показаната парсираща машина съдържа различни видове модули като доставчик, скенер, лексер, парсер, оптимизатор, строител и филтър. Различните модули и техните функционалности са описани в детайли. Главата завършва с дефинициите на различни видове синтактични дървета и командите за строеж, на базата на които дърветата могат да се създадат.

В глава 3 Алгоритъм Тунелно парсиране се съдържа детайлно описание на алгоритъма Тунелно парсиране. Различните обекти, които се създават преди началото на парсирането, на базата на дадена напреднала граматика, са описани подробно. Списъкът от тези обекти включва: стек за изпълнение, стек за дълбочина, стек за повторение, архивни стекове,

автомати, достижими дървета, конфликти, тунели, маршрутизатори и контролни обекти. Детайлното описание на контролните обекти съдържа стъпките, които парсерът извършва и ефективно е псевдо-кода на алгоритъма Тунелно парсиране. Главата съдържа пример с дефинираните обекти, които се използват от алгоритъма и завършва с примерно изпълнение на алгоритъма за избран низ от входни данни.

В глава 4 Профилиране на автоматично генерирани парсери се дискутира специално създаден за целта на дисертацията инструмент, наречен профилатор, с помощта на който могат да се генерират голям брой контекстно свободни граматики и входни данни и да се извършват експерименти с парсиращи машини, които са генерирани от различни парсер генератори на базата на тези граматики. Главата съдържа описанието на специално създаден за целта на дисертацията език на шаблонна граматика за императивно метапрограмиране на граматики. Показани са различните граматики, които дефинират валидните според шаблонния език скриптове. Главата завършва с интерпретацията на резултатите от четири различни експеримента, които са проведени с помощта на профилатора.

В заключението са обобщени и систематизирани основните резултати, като са посочени научните, научно-приложните и приложните приноси на дисертационния труд. Формулирани са перспективите за развитие на дисертационната тематика.

Списъкът на използваната литература включва 191 (сто деветдесет и едно) заглавия, като две са на български език, две са на руски език, а останалите са на английски език.

Апробация

Основните резултати на изследването са докладвани на международна конференция и международен научен форум – 14th international conference Education and research in information society (ERIS), Plovdiv и 8th Summer School, CEFP 2019, Budapest, Hungary.

Резултати от дисертационното изследване са представени в 6 (шест) публикации – 4 (четири) в специализирани списания, 1 (една) – в трудовете на международна конференция и 1 (една) – в трудовете на международен форум. Шестте публикации са индексирани в световноизвестните бази от данни: 4 (четири) в Web of Science и 5 (пет) в Scopus. Пет от публикациите са в издания с SJR.

Забелязан е цитат на 1 (една) от публикациите по темата в едно научно изследване, което е индексирано в световните бази от данни.

Благодарности

Изказвам благодарности на моя научен ръководител проф. д-р Елена Сомова за предоставената ми възможност за провеждане на дисертационно изследване в областта и за голямото количество конструктивен критицизъм, който доведе до по-високо ниво на дисертацията. Благодаря на проф. д-р Христо Кискинов, проф. д-р Емил Хаджиколев и гл. ас. д-р инж. Георги Пашев за ценните препоръки за подобрение на дисертацията, както и на цялата катедра „Компютърна информатика“ за подкрепата по време на докторантурата. Благодаря също на моята съпруга Полина Ханджийска за демонстрираната подкрепа, показаното търпение и помощта в откриването на различни несъответствия в работната версия на дисертацията. Благодаря на моята майка Йорданка Кацарова за предложени езикови подобрения на неясни твърдения в работната версия на дисертацията. Не на последно място, благодаря на Димитър Нанев за моралната подкрепа.

Посвещавам този труд на моята дъщеря Мая Ханджийска.

1 Обзор

Теорията на транслацията включва работата с формалните езици и обработващите ги абстрактни машини [1, с.208г], с които транслаторите работят. Общоприето е, че един транслатор извършва лексикален и синтактичен анализи.

1.1 Формални средства

Формалните граматики и съответните им абстрактните машини правят възможно записването на езици в различни форми и проверката за принадлежност на произволни низове в тези езици. Комбинирането на формалните средства позволява работата с програмите, написани на програмни езици.

Комбинационни схеми

Начин за работа с комбинационни схеми съставени от релета се разглежда от Шанън (Claude Elwood Shannon), който прави паралел между пропозиционната логика (съждителната логика) и схемите, които я изпълняват.

Невронни мрежи

Времето, в което една мрежа работи, се приема за дискретно (разделено на последователни и равни интервали от време) и започва от едно. Във всеки интервал от време всички неврони пораждат, или не, импулс. Според това как една мрежа може да записва и прочита информация в средата са възможни различни, по-сложни, видове машини.

Регулярни изрази

В [2, с.104с] $\mathbf{X}^+$ се използва като съкращение на **XX***.

Ефективен начин за трансформиране на краен автомат в регулярен израз е да се използва генерализиран недетерминиран краен автомат — автомат с преходи, съставени от регулярни изрази [3, с.70г]. Всеки недетерминиран автомат е генерализиран, но не и обратното.

В практиката написаните от потребителите регулярни изрази не винаги дефинират точно низовете, които потребителят очаква [4].

Пораждащи граматики, йерархия на Чомски

Подсилена Бекус-Наур Форма (augmented Backus-Naur form; ПБНФ) е описана в [5] и в [6]. Колкото повече изразителна възможност има един метаезик, толкова по-лесно е за разработчика да разработи граматиката, която дефинира обектния език.

Езиците, дефинирани от четири вида граматики (неограничени, контекстни, безконтекстни и регулярни) образуват йерархия на Чомски.

Казва се (вече използвано в някои случаи по-горе), че всички действия (генерация на низовете в езика, който е дефиниран от граматиката, разпознаване на данни и др.) се извършват на база на граматиката.

Безконтекстни граматики

Преименуващите правила, могат да се премахнат от една граматика без да се промени езика, но това ще промени синтактичните дървета, които се генерират на база на граматиката. С други думи, трансформацията е винаги възможна, но не винаги е приемлива.

В една самовградена (self-embedding) граматика за активен и централен нетерминал **A** съществува $\mathbf{A} \Rightarrow^+ \mathbf{xAy}$ за $\mathbf{x,y} \in \mathbf{V}^+$ [7, с.148д].

Крайни автомати

В [8, с.210г] е отбелязано, че ако функцията **f** извежда повече от един резултат, тогава автоматът става недетерминиран. Всички ленти **T(A)**, които се разпознават от един краен автомат **A**, са един регулярен език.

Размерът на един краен автомат може директно да повлияе на времето за разпознаване на низове от него. Поради тази причина е желателно крайните автомати, които се използват за разпознаване на низове, да имат възможно най-малко на брой състояния и преходи.

Стекови автомати

Функцията на преходите е $\delta: S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to^{\{\}} S \times \Gamma^*$, където с $\to^{\{\}}$ се дефинира, че функцията може да върне повече от един резултат като редица и че всички резултати са елементи на дадено множество.

Един стеков автомат е детерминиран, ако автоматът винаги има не повече от един възможен преход за прилагане. Всички останали стекови автомати са недетерминирани.

Ако един стеков автомат може да разпознае всеки низ без да използва повече от константен брой символи в стека, тогава този стеков автомат може да се трансформира в (детерминиран) краен автомат [9, с.259д].

Дефиниция на алгоритъм

Нормалният алгоритъм на Марков има по-кратко описание от еквивалентната машина на Тюринг. Една от причините е, че резултатът се получава като винаги най-лявата от вградените думи се замества при прилагане на дадена формула. Така не се налага изрично описание за движението на главата в еквивалентната машина на Тюринг.

Функция, която винаги приключва (halts; спира [1, с.208г]), за всички възможни аргументи, след краен брой изпълнени инструкции, се нарича алгоритъм [2, с.27г].

Затвореност на операциите и решимост на проблемите

Според [10, с.159д], за произволни безконтекстни езици $\mathbf{A}$ и $\mathbf{B}$ е нерешимо дали: $L(A)=L(B)$; $L(A)\subset L(B)$; и $L(A)\cap L(B)=\emptyset$.

1.2 Транслатори

Транслирането е основен процес при работата с програмите, написани на програмни езици. Крайните и стековите автомати намират приложение при транслирането.

Друга възможност е езикът да няма лексер граматика (а само парсер граматика) и съответно да няма лексер. Парсирането на този вид езици се нарича еднофазно (single-phase) парсиране в [11, с.3дд] и безскенерно парсиране в [12].

Транслаторите са необходими за програмните езици. Понеже всеки език за програмиране е формален език, описаната по-горе теория се прилага директно при създаването на транслатори.

Възможно е лексерът да бъде контекстно запознат (context-aware) [13] — лексерът изпраща множество от токен типове, вземайки предвид текущото състояние на парсера.

Популярен начин на работа на парсера с токените е, те да се използват за разпознаване от стеков автомат, който е генериран на базата на безконтекстна граматика (видът граматики, които се използват в тази секция).

Разработката на граматики и езици за програмиране е директно свързана с процеса компилиране.

1.3 Изводи

Съществуват много алгоритми за разпознаване и за парсиране. Парсиращите алгоритми често имат проблеми с граматиките, които имат правила генериращи празни думи. Граматиките в ПБНФ съдържат различни обекти, които могат да генерират празна дума. Едни от тези обекти са правилата. На база на изследванията в тази глава се правят следните изводи:

- Популярните алгоритмите за парсиране не поддържат повторения на граматични елементи, които могат да съществуват в една граматика в ПБНФ и извършват много операции по време на анализ, които могат да се преизчислят от граматиката;
- Ключова задача на алгоритмите за парсиране е бързото използване на правила, които генерират празни думи. В [14] се показва, че това може да доведе до много работа за граф, в който няма нито една буква;

- Лексерите в литературата групират поредици от букви в токен с име (уникален идентификатор според използваното правило за групиране). Според ПБНФ стандарта [5], `%x30-32` е буквен диапазон, който може да се представи като множеството от букви `{'0','1','2'}`. Ако парсерът работи с граматика в ПБНФ, тогава съществува следния диапазонен проблем — как да се парсира на базата на диапазона `%x30-32` и с коя част на токена да се сравни той;
- Това създава следния регистров проблем — как в парсер граматиката да се различат имената на токените от имената на правилата;
- Липсват детайлни емпирични измервания за производителността на различните автоматично генерирани парсери, които да дават информация за очакваното количество ресурси, което е необходимо за анализ на данни според конкретни граматични елементи;
- Необходима е парсираща машина, която да съчетае парсирането с и без лексикален анализ, по такъв начин, че ползите от двата начина да се обединят в едно и потребителят да избере дали да има лексикален анализ.

2 Парсираща машина

В тази глава е изложен проектът на машина за парсиране на данни, наречена парсираща машина (Parsing Machine), в която могат да се използват различни алгоритми за парсиране.

Неформално една парсираща машина е абстрактна машина изградена от модули с връзки между тях, която работи като приема неструктурирани данни като вход и изпраща структурирани данни като изход. Всяка парсираща машина винаги сканира (прочита) входните данни и винаги извършва синтактичен анализ.

2.1 Базова концепция на машината

Формализирането на концепцията на парсираща машина е необходимо за да се покажат взаимодействията между модулите в машината и специфичните частни случаи. За да се използват токените от машината е необходим нов вид граматики, които да позволят анализиране на база на всички токени (с изключение на лимит токена) в машината.

Парсиращата машина е проектирана, така че да разпределя анализа на входните данни между различните модули с цел паралелизъм и за да бъде възможно прилагането на различни стратегии за анализ от различните модули. В машината са обособени следните модули: доставчик, скенер, лексер, парсер, оптимизатор, строител и филтър.

Подобно на работата с токени от скенера, лексера и парсера, по-долу се дискутира работата с команди за строеж на синтактични дървета от парсера, оптимизатора и строителя.

2.2 Букви и лексеми в машината

Разумно е да се очаква от една машина, която анализира текстови данни, да може да приема (поне) кодирани входни данни с UTF-8 [15] и да обработва грешките в кодирането автоматично. Дефинира се буква като Уникод позиция — цяло неотрицателно число, с най-голяма възможна стойност, която зависи от използвания Уникод стандарт. Множеството от букви се означава с `U` и се подчертава, че има буква с Уникод позиция нула.

Много автори използват Уникод позиции при имплементацията на транслатори, но на теоретично ниво работят с абстрактни букви. В теорията на предложената парсираща машина буквите са конкретизирани като Уникод позиции и са задължителни за всички модули в машината (някои от които са нови или с нови функционалности), не само за лексера и парсера. Конкретизирането на буквите доближава теорията до практиката и позволява формалното третиране на някои практически проблеми.

2.3 Формална дефиниция на парсираща машина

Една парсираща машина е n-орката **(M,T,I,S,L,P,O,B,F)**, където **M** е крайно непразно множество от модули, **C** е крайно непразно множество от връзки между модулите, **I∈M** е непразно множество от доставчици, **S∈M** е скенер, **L∈M** е множество от лексери, **P∈M** е парсер, **O∈M** е множество от оптимизатори, **B∈M** е непразно множество от строители и **F∈M** е множество от филтри. Един модул е изходен модул, когато изпраща данни извън машината.

2.4 Токени

С цел за решаване на диапазонния и регистровия проблеми, се различават няколко вида токени, които са предназначени за използване като приложни данни по прехода от скенера до лексера, по прехода от лексера до парсера или по прехода от скенера до парсера (ако няма лексер).

Един модул се класифицира като предавател, когато изпраща токени към друг модул, който се класифицира като приемник. За предавател, който използва граматика **G=(Φ,Θ,−,−)** за нетерминали **Φ** и букви **Θ⊆U**, се дефинира следното:

- Атрибут се записва като **l=v**, където **l** е непразен низ от букви (етикет) и **v** е стойност с дефиниционна област, която зависи от етикета;
- Буква токен е n-орката **(t−character,n,α)**, където **n∈Θ** е име и **α** е крайно множество от атрибути. Ако **|α|=0**, тогава токенът се записва като **(t−character,n)**;
- Редица токен е n-орката **(t−sequence,n,e,α)**, където **n∈Φ** е име, **e** е низ над **Θ** (лексема), **|e|>0** и **α** е крайно множество от атрибути. Ако **|α|=0**, тогава токенът се записва като **(t−sequence,n,e)**. Неограничената дължина на **e** прави тези токени елементи в безкрайно множество;
- Лимит токен е n-орката **(t−limit,β,e,α)**, където **β⊆Φ** са нетерминали, **|β|>0**, **e** е лексема, **|e|>0** и **α** е крайно множество от атрибути. Ако **|α|=0**, тогава токена се записва като **(t−limit,β,e)**;
- Край токен е n-орката **(t−eof,α)**, където **α** е крайно множество от атрибути. Ако **|α|=0**, тогава токена се записва като **(t−eof)**;
- Безкрайното множество от токени се означава с **H**.
- Видът на **h∈H** е първия елемент в n-орката на **h**, който се получава като резултат от изпълнението на функция **Π:H→H'** за **H'={t−character,t−sequence,t−limit,t−eof}**.

В горните дефиниции на токени само редица токена (от вида **t−sequence**) има лексема, която може да се използва за анализ, а буква токена (от вида **t−character**) вместо лексема има име. Тази разлика от литературните източници позволява по различни начини работа с тези два вида токени. Лимит токенът (от вид **t−limit**) дава теоретичната основа на имплементациите на машината за да обработят коректно ситуациите (вместо да блокират), в които модулите на машината, работещи с токени, нямат достатъчно памет за да продължат работата си.

2.5 Напреднали граматики на ниво 1

С цел решаване на регистровия проблем и за да стане възможна употребата на лексемата в **t−sequence** токените за анализиране, се добавя нов вид символи в граматиките, които се наричат фраза символи.

Дефинират се множество от сравнители е **M={≈,≡}**, където ≈ означава нечувствителен сравнител, а **≡** чувствителен и регистър функция **Δ:U×U×M→{}U**. Например **Δ('a','a',≈)={'a','A'}**.

2.6 Напреднали граматики на ниво 2

Напредналите граматики на ниво 2 са подобрение на граматиките на ниво 1. Целта на това подобрение е доближаване на напредналите граматиките до тези в ПБНФ чрез добавянето на групи, конкатенации, алтернации, повторения на символи и символ, дефиниращ празна дума.

Една напреднала граматика на ниво 2 е n-орката $\mathbf{A=(C,N,\Sigma,\Omega,R,S)}$ за множество от категории $\mathbf{C}$, множество от нетерминали $\mathbf{N}$, множество от букви $\mathbf{T\subseteq U}$, множество от напреднали символи $\mathbf{\Omega}$, множество от правила $\mathbf{R}$ и множество от начални нетерминали $\mathbf{S\subseteq N}$.

Дефинира се, че:

- Всеки елемент $\omega\in\Omega$ е индексирана n-орка — n-орка, която притежава символен индекс $k\in\mathbb{N}_0$, който се записват като $\alpha^k\in\Omega$ за n-орка α. Индексът не се записва, когато той не се използва;
- $\nexists x,y|x=y\wedge\alpha^x\in\Omega\wedge\beta^y\in\Omega\wedge\alpha\neq\beta$ — всички символни индекси в Ω са различни;
- Видът на $\omega\in\Omega$ е първия елемент в n-орката на ω (с индекс нула) и се получава като резултат от изпълнението на функция $\Pi:\Omega\to\Omega'$ за $\Omega'\in\{s\text{-}reference,s\text{-}character,s\text{-}phrase,s\text{-}eof,s\text{-}concatenation,s\text{-}alternation,s\text{-}group,s\text{-}repeat,s\text{-}epsilon\}$;
- Функция за достъп до елементите в n-орките се дефинира като $GET:\Omega\times\mathbb{N}_0\to\Omega'\cup C\cup N\cup\Omega$, която при изпълнение на $GET(\omega,n)$ за $\omega\in\Omega$ и $n\in[0..|n|)$, връща елемента с индекс n в n-орката на ω и $\Pi(\omega)=GET(\omega,0)$.

Дефинициите от напредналите граматики на ниво 1 се пренасят на ниво 2 като n-орките стават индексирани:

- Фраза е низ от букви над Σ;
- Референция символ е индексирана n-орка $(s\text{-}reference,n)\in\Omega$ за $n\in N$;
- Буква символ е индексирана n-орка $(s\text{-}character,f,t,m)\in\Omega$ за $f,t\in\Sigma$, $f\leq t$, $m\in M$;
- Фраза символ е индексирана n-орка $(s\text{-}phrase,c,p,m)\in\Omega$ за $c\in C$, фраза p и $m\in M$;
- Край символ е индексирана n-орка $(s\text{-}eof)\in\Omega$.

Добавят се следните символи на ниво 2:

- Конкатенация е индексирана (n+1)-орка $(s\text{-}concatenation,\omega_0,\ldots,\omega_{n-1})\in\Omega$ за $\omega_i\in\Omega$, $n>0$ и $\Pi(\omega_i)\notin\{s\text{-}concatenation,s\text{-}alterantion\}$;
- Алтернация е индексирана (n+1)-орка $(s\text{-}alternantion,\omega_0,\ldots,\omega_{n-1})\in\Omega$ за $\omega_i\in\Omega$, $n>0$ и $\Pi(\omega_i)=s\text{-}concatenation$;
- Група символ е индексирана n-орка $(s\text{-}group,\omega)\in\Omega$ за $\omega\in\Omega\wedge\Pi(\omega)=s\text{-}alternantion$;
- Повторение символ е индексирана n-орка $(s\text{-}repeat,n,m,\omega)\in\Omega$ за $n\in\mathbb{N}_0$, $m\in(\mathbb{N}_0\cup\infty)$, $n\leq m$, $n\neq1\vee m\neq1$, $\omega\in\Omega$ и $\Pi(\omega)\in\{s\text{-}reference,s\text{-}group,s\text{-}character,s\text{-}phrase,s\text{-}eof\}$. Повторение с $n>1\vee(m>1\wedge m\neq\infty)$ се нарича броимо повторение;
- Епсилон символ е индексирана n-орка $(s\text{-}epsilon)\in\Omega$.

Дефинира се бинарната ирефлексивна релация $\mapsto$ като за $x,y\in\Omega$ е вярно, че $x\mapsto y$, ако и само ако $GET(x,n)=y$ за някое $n\in[0..|x|)$. Знакът $\mapsto$ се чете като „има указател към“. Когато x има указател към y ще се казва, че y се намира в x, и че x съдържа или използва y. Транзитивното затваряне (транзитивна обвивка [1, с.26с]) на $\mapsto$ се дефинира като $\mapsto^*$, което се чете като „има транзитивно указател към“.

Правилата в една напреднала граматика на ниво 2 могат да бъдат в различна форма. Дефинира се, че напреднала безконтекстна граматика на ниво 2 е напреднала граматика на ниво 2, която има правила в R от вида $A\to\alpha$ за $A\in N$ и $\Pi(\alpha)=s\text{-}alternation$. От тук надолу, ако не е казано друго, напреднала граматика означава напреднала безконтекстна граматика на ниво 2.

2.6.1 Добре дефинирани напреднали граматики

Една напреднала граматика $\mathbf{A=(-,-,-,\Omega,R,-)}$ е добре дефинирана, когато всеки символ в $\mathbf{\Omega}$ се използва от един и само един обект в $\mathbf{A}$ и всеки символ се използва (транзитивно) в едно правило.

Дефинира се следното:

- С $[\![\ ]\!]$ се означава множество с повтарящи се елементи;
- $\mathbf{POINTER_\Omega(\omega)=[\![x|(x\mapsto\omega|x\in\Omega)]\!]}$ са всички символи, които използват $\boldsymbol{\omega}$;
- $\mathbf{POINTER_R(\omega)=[\![r|(r\mapsto\omega|r\in R)]\!]}$ са всички правила в граматиката, които използват $\boldsymbol{\omega}$;

Напредналата граматика $\mathbf{A}$ е добре дефинирана, ако и само ако следните условия за изпълнени:

1. $\mathbf{|POINTER_\Omega(\omega)|=1}$ за всяко $\boldsymbol{\omega\in\Omega|\Pi(\omega)\neq s\text{-}alternation}$;
2. $\mathbf{|POINTER_\Omega(\alpha)|+|POINTER_R(\alpha)|=1}$ за всяко $\boldsymbol{\alpha\in\Omega|\Pi(\omega)=s\text{-}alternation}$;
3. За всяко $\boldsymbol{\omega\in\Omega}$ е вярно, че $\boldsymbol{r\mapsto^*\omega}$ за някое $\boldsymbol{r\in R}$.

От тук надолу се приема, че всички граматики са добре дефинирани. Това, че една граматика е добре дефинирана, позволява уникално да се определи броя повторения на даден символ, като се дефинира и използва функцията $\mathbf{REPEAT:\Omega\to\mathbb{N}_0\times(\mathbb{N}_0\cup\infty)}$.

2.6.2 Множества от символи в напреднали граматики

От тук надолу се използват следните означения:

- Всички референция символи в $\mathbf{\Omega}$ са $\mathbf{N_\Omega=\{\omega|\omega\in\Omega\wedge\Pi(\omega)=s\text{-}reference\}}$;
- Всички фраза символи в $\mathbf{\Omega}$ са $\mathbf{P_\Omega=\{\omega|\omega\in\Omega\wedge\Pi(\omega)=s\text{-}phrase\}}$;
- Всички фраза символи с празна фраза се наричат универсални фраза символи (или $\mathbf{s\text{-}universal}$) и са $\mathbf{PA_\Omega=\{\omega|\omega\in P_\Omega\wedge|\omega.p|=0\}}$;
- Всички не празни фраза символи с чувствителен сравнител се наричат чувствителни фраза символи (или $\mathbf{s\text{-}sensitive}$) и са $\mathbf{PS_\Omega=\{\omega|\omega\in P_\Omega\wedge|\omega.p|>0\wedge|\omega.m|=\equiv\}}$;
- Всички не празни фраза символи с нечувствителен сравнител се наричат нечувствителни фраза символи (или $\mathbf{s\text{-}insensitive}$) и са $\mathbf{PI_\Omega=\{\omega|\omega\in P_\Omega\wedge|\omega.p|>0\wedge|\omega.m|=\approx\}}$;
- Всички край символи в $\mathbf{\Omega}$ са $\mathbf{F_\Omega=\{\omega|\omega\in\Omega\wedge\Pi(\omega)=s\text{-}eof\}}$;
- Всички група символи в $\mathbf{\Omega}$ са $\mathbf{G_\Omega=\{\omega|\omega\in\Omega\wedge\Pi(\omega)=s\text{-}group\}}$;
- Всички повторение символи в $\mathbf{\Omega}$ са $\mathbf{Y_\Omega=\{\omega|\omega\in\Omega\wedge\Pi(\omega)=s\text{-}repeat\}}$.

2.6.3 Генерация на низове

Разширява се $\mathbf{\Lambda}$ за да работи и с низове, като $\mathbf{\Lambda:U^*\times M\to^{\{\}}U^*}$. Например $\mathbf{\Lambda("ab",\approx)=\{"ab","aB","Ab","AB"\}}$.

Всеки напреднал символ $\boldsymbol{\omega\in\Omega}$ дефинира множество от низове над $\mathbf{N\cup H\cup\Omega}$. Това, че $\boldsymbol{\omega}$ дефинира низ $\boldsymbol{\beta\in(N\cup H\cup\Omega)^*}$ се записва като $\boldsymbol{\omega\rightrightarrows\beta}$. По-долу се представят низовете, които се дефинират от различните напреднали символи:

- За напреднали граматики на всички нива:
  - Ако $\boldsymbol{\omega=(s\text{-}reference,n)\in\Omega}$, тогава $\boldsymbol{\omega\rightrightarrows n}$;
  - Ако $\boldsymbol{\omega=(s\text{-}character,f,t,m)\in\Omega}$, тогава $\boldsymbol{\omega\rightrightarrows(t\text{-}character,x)}$ за всяка буква $\boldsymbol{x\in\Lambda(f,t,m)}$;
  - Ако $\boldsymbol{\omega=(s\text{-}phrase,c,-,-)\in PA_\Omega}$, тогава $\boldsymbol{\omega\rightrightarrows(t\text{-}sequence,c,\beta)}$ за всеки низ $\boldsymbol{\beta\in T^+}$;
  - Ако $\boldsymbol{\omega=(s\text{-}phrase,c,p,m)\in(PS_\Omega\cup PI_\Omega)}$, тогава $\boldsymbol{\omega\rightrightarrows(t\text{-}sequence,c,\beta)}$ за всеки низ $\boldsymbol{\beta\in\Lambda(p,m)}$;
  - Ако $\boldsymbol{\omega=(s\text{-}eof)\in\Omega}$, тогава $\boldsymbol{\omega\rightrightarrows(t\text{-}eof)}$.
- За напреднали граматики на ниво 2:
  - Ако $\boldsymbol{\omega=(s\text{-}concatenation,\omega_0,\ldots,\omega_{n-1})\in\Omega}$, тогава $\boldsymbol{\omega\rightrightarrows\omega_0\ldots\omega_{n-1}}$;

- Ако $\omega=(\mathtt{s\text{-}alternation},\omega_0,\ldots,\omega_{n-1})\in\Omega$, тогава $\omega\rightrightarrows\omega_i$ за всяко $i\in[0..n)$;
- Ако $\omega=(\mathtt{s\text{-}group},\alpha)\in\Omega$, тогава $\omega\rightrightarrows\alpha$;
- Ако $\omega=(\mathtt{s\text{-}repeat},n,m,\mathtt{x})\in\Omega$, тогава $\omega\rightrightarrows\beta$ за всеки низ β над $\{\mathtt{x}\}$, където $n\le|\beta|\le m$;
- Ако $\omega=(\mathtt{s\text{-}epsilon})\in\Omega$, тогава $\omega\rightrightarrows\varepsilon$.

Един напреднал символ $\omega\in\Omega$, който дефинира низ γ, може да се <u>разложи</u> в низа $\alpha\omega\beta$ като резултата от разлагането на символа е $\alpha\gamma\beta$, където $\alpha,\beta\in(N\cup\Omega\cup H)^*$. Редицата $(\mu_0,\ldots,\mu_n)$ за $n\ge0$ е φ разлагане на ψ, ако $\varphi,\psi\in(N\cup H\cup\Omega)^*$, $\varphi=\mu_0$, $\mu_n=\psi$ и μ_{i+1} е резултата от разлагането на някой напреднал символ в μ_i за $i\in[0..n)$. Разлагането е <u>терминално</u>, когато $\mu_n\in(N\cup H)^*$. Казва се, че от φ може да се изведе ψ, когато съществува φ разлагане на ψ.

<u>Нормалното прилагане</u> на правило $D\to\gamma\in R$ на низ $\alpha D\beta$ завършва с резултат $\alpha\gamma\beta$, където $\alpha,\beta\in(N\cup\Omega\cup H)^*$. От тук надолу всички прилагания на правила са нормални. Редицата $(\mu_0,\ldots,\mu_n)$ за $n\ge0$ е <u>нормалното</u> φ извеждане на ψ, ако $\varphi,\psi\in(N\cup H\cup\Omega)^*$, $\varphi=\mu_0$, $\mu_n=\psi$ и μ_{i+1} за $i\in[1..n)$ е резултата от: а) нормалното прилагане на някое правило в R на μ_i; или б) разлагането на напреднал символ $\omega\in\Omega$ в μ_i. От тук надолу всички извеждания са нормални. Броят стъпки в едно нормално φ извеждане на ψ е n. Нормалното извеждане е <u>терминално</u>, когато $\mu_n\in H^*$. Казва се, че от φ може да се изведе ψ, когато съществува нормално φ извеждане на ψ. С $\varphi\Rightarrow\psi$ се записва извеждане с една стъпка, с $\varphi\Rightarrow^*\psi$ — извеждане с нула или повече стъпки и с $\varphi\Rightarrow^+\psi$ — извеждане с една или повече стъпки.

Безкрайното множество от буква и редица токени е $H_{CS}=\{h\mid h\in H\wedge\Pi(h)\in\{\mathtt{t\text{-}character},\mathtt{t\text{-}sequence}\}\}$. Езикът генериран от дадено начално правило $J\in S$ в граматика A е $L_A(J)=J_{CS}\cup J_E$, където $J_{CS}=\{w(\mathtt{t\text{-}eof})\mid J\Rightarrow^+w\wedge w\in H_{CS}^*\}$ и $J_E=\{w\mid J\Rightarrow^+w\wedge w\in H_{CS}^*(\mathtt{t\text{-}eof})\}$. Когато граматиката се подразбира вместо $L_A(J)$ ще се пише само $L(J)$.

2.6.4 Множества от токени

Напредналите символи, които директно дефинират токен, са в множеството $H_\Omega=\{\omega\mid\omega\in\Omega\wedge\Pi(\omega)\in\{\mathtt{s\text{-}character},\mathtt{s\text{-}phrase},\mathtt{s\text{-}eof}\}\}$. Дефинира се функцията $\Psi:H_\Omega\to^{\{\}}H$, такава че изпълнението на $\Psi(\omega)$ с $\omega\in H_\Omega$ връща като резултат множеството от токени дефинирани от ω, според дефинициите по-горе. Ще се казва, че два напреднали символа $x,y\in\Omega$ се <u>припокриват</u>, когато $\Psi(x)\cap\Psi(y)\ne\emptyset$ (т.е. когато множествата, които символите дефинират имат поне един общ елемент).

2.6.5 Метаграматика за напреднали граматики

За да се дефинират напреднали граматики, се добавят няколко правила в ПБНФ стандарта, защото: а) стандартът не позволява Уникод букви; и б) стандартът не може да изрази фраза символи, нито да изрази с един елемент символа $(\mathtt{s\text{-}character},x,x,\approx)$, когато $|\Delta(x,x,\approx)|>1$. Резултатът, който се получава от ПБНФ стандарта с горните добавки се нарича Напреднала ПБНФ (НПБНФ).

2.7 Фразова машина със състояния

По време на парсиране на базата на напреднала граматика е възможно, парсерът да провери дали даден токен $h\in H$ принадлежи едновременно на голям брой множества, всяко от които е резултат от $\Psi(\omega)$ за $\omega\in Z$ и $Z\subseteq H_\Omega$.

От множествата, които се дефинират от фраза символи, следва директно, че всички фраза символи $P\subseteq Z$ могат да се разделят на множества, които нямат общи елементи P_c според категорията им — $P_c=\{\omega\mid\omega\in P\wedge\omega.c=c\}$.

За да може сравнението на лексема с фраза да е бързо, се създава <u>фразова машина</u>, която класифицира в класове различните лексеми в токените и фразите във фраза символите на

граматиката, така че вместо сравнения между низове да се извършат сравнения между класовете на низовете.

Всяка уникална фраза в чувствителен фраза символ получава уникален чувствителен индекс. Всяка фраза в нечувствителен фраза символ получава нечувствителен индекс.

След като всички фрази във фраза символи са класифицирани, за да се провери принадлежност на даден редица токен в множествата от токени, дефинирани от различните фраза символи в дадено множество $\mathbf{P_c}$, трябва да се изпълнят редица от действия.

Фразовата машина със състояния (накратко фразова машина) е n-орката $\mathbf{(\Sigma,Q,\delta,F,q_0)}$ за азбука $\mathbf{\Sigma\subseteq U}$, непразно множество от състояния $\mathbf{Q}$, функция на преходите $\mathbf{\delta:Q\times\Sigma\to Q\times M}$, множество от финални състояния $\mathbf{F}$ и начално състояние $\mathbf{q_0\in Q}$.

Функцията на преходите $\mathbf{\delta}$ се представя като множество от преходи във вида $\mathbf{\alpha\to\beta}$ за n-орката от аргументи $\mathbf{\alpha=(s,\sigma)}$, n-орката от резултати $\mathbf{\beta=(d,m)}$, вход $\mathbf{s\in Q}$, преходна буква $\mathbf{\sigma\in\Sigma}$, изход $\mathbf{d\in Q}$ и сравнител $\mathbf{m\in M}$. Всяко финално състояние е във вида $\mathbf{q\to(n,m)}$ за $\mathbf{q\in Q}$, $\mathbf{n\in\mathbb{N}_1}$ и $\mathbf{m\in M}$. Няма две финални състояния с еднакво $\mathbf{q}$ — $\mathbf{\nexists x,y|x,y\in F\wedge x\neq y\wedge x.q=y.q}$. За $\mathbf{x\in\delta}$ се казва, че $\mathbf{x}$ е чувствителен преход, когато $\mathbf{x.m=\equiv}$ и нечувствителен преход, когато $\mathbf{x.m=\approx}$.

За да може машината да работи по необходимия начин тя трябва да бъде добре дефинирана, за което са необходими допълнителни ограничения към формалната дефиниция. Първият начин за анализ на низ $\mathbf{w}$ от една добре дефинирана фразова машина се нарича чувствителен анализ. При този вид анализ, фразовата машина работи подобно на краен автомат. Вторият начин за анализиране на низ $\mathbf{w}$ от една добре дефинирана фразова машина се нарича нечувствителен анализ.

Функция **INSENSITIVE:Σ*→$\mathbb{N}_0$×M** с параметър **w**

```
 1. begin
 2.   c←q₀, i←0                          ▷ подготовка
 3.   while i<|w| do                     ▷ стъпки
 4.     l←LOWER(wᵢ), u←UPPER(wᵢ)         ▷ варианти на wᵢ
 5.     if l=u then                      ▷ един вариант
 6.       t|t∈δ∧t.s=c∧t.σ=l              ▷ търсене
 7.       if ∄t then return (0,≡)        ▷ неуспех ако ∄t
 8.       c←t.d                          ▷ следващо състояние
 9.     else
10.       L|L∈δ∧L.s=c∧L.σ=l              ▷ търсене
11.       if ∄L then return (0,≡)        ▷ неуспех ако ∄L
12.       U|U∈δ∧U.s=c∧U.σ=u              ▷ търсене
13.       if ∄U then return (0,≡)        ▷ неуспех ако ∄U
14.       if L.m=≈∧U.m=≈ then c←L.d
15.       else if L.m=≈ then c←L.d       ▷ следващо състояние
16.       else if U.m=≈ then c←U.d       ▷ следващо състояние
17.       else c←L.d                     ▷ предпочитание на L
18.     i←i+1                            ▷ следващата буква
19.   f|f∈F∧f.q=c                        ▷ търсене
20.   if ∃f return (f.n,f.m)             ▷ успех ако ∃f
21.   return (0,=)                       ▷ неуспех
22. end
```

Фигура 1: Псевдо код за нечувствителен анализ от фразова машина

Формалното описание на нечувствителния анализ е показано с псевдокод на Фигура 1.

Една напреднала граматика може да се компилира до фразова машина, като се използват фраза символите в граматиката.

Резултатът от класификацията на всеки низ е n-орката **(n,m)** за **n∈**$\mathbb{N}_0$ и **m∈M**.

2.8 Модули на парсираща машина

След като са дефинирани различните обекти, които се използват в машината, в тази секция ще се представят модулите, които работят с тези обекти.

Първият модул в една машина се нарича доставчик. Този модул изпраща поредици от битове на следващия модул. В една машина има поне един доставчик.

Скенер се нарича модула, който приема редица от битове от доставчика и ги декодира в букви, които изпраща към следващия модул в машината. Декодирането може да бъде на базата на различни видове стандарти, но се приема, че декодирането е според използвания Уникод стандарт от машината.

За всяка декодирана буква **ψ** от входните данни, скенерът изпраща токен **(t-character,ψ,α)**. Този модул служи като „граница“ в машината, която разделя операциите с битове и операциите с токени.

Лексер се нарича модула, който приема токени от предишния модул и изпраща същите или други токени към следващия модул, като токените, които се изпращат зависят от конкретния лексер. Може да има нула (което означава, безлексерна машина) или повече лексери, всички подредени един след друг и всички след скенера.

Не се поставя стриктно ограничение за вида граматика в спецификацията на лексера, но за улеснение ще се използва нормална граматика **G=(N,-,-,-)**. Правилото в граматиката на лексера, на базата на което лексерът приема най-дългата възможна редица от токени, се отбелязва с **q**. Променя се традиционния начин на работа на лексера (при който, ако няма правило в граматиката, което да приеме текущите данни, това е грешка) като се дефинира нов. Във всеки един момента от работата на лексера:

- Ако правило **q** е уникално установено (след поне един приет токен), тогава:
  1. Лексерът изпраща **t-sequence(n,e,α)** токен, където **n∈N** е името на **q**, а **e** е низ от имената на приетите **t-character** токени на базата на правило **q**. Ако машината работи с определени атрибути, тогава лексерът ги добавя в **α**;
  2. Лексерът премахва използваните **t-character** токени;
  3. Анализът започва от начало с оставащите токени, които са приети от скенера.
- Ако лексерът установи, че правило **q** няма да бъде установено, тогава:
  1. Лексерът изпраща към парсера първия от токените, който е приел от скенера;
  2. Анализът започва от начало с оставащите токени.
- В момента, в който лексерът достигне своя лимит (ако има такъв), преди правило **q** да бъде еднозначно установено, тогава:
  1. Лексерът изпраща токен **t-limit(Q)** към парсера, където **Q** е множеството от имената на правилата, за които лексерът не е установил, че не приемат редицата от токени, която е получена от скенера;
  2. Анализът спира.
- Ако лексерът трябва да анализира само един оставащ токен от вида **t-eof**, тогава:
  1. Лексерът изпраща край токен към следващия модул;
  2. Анализът спира.

Лексер, който работи по описания начин се нарича продължителен лексер (continuous lexer).

Парсер се нарича модула, който приема входни токени, анализира ги на базата на дадена спецификация и изпраща команди за строеж на синтактична структура (syntax structure construction commands; съкратено само команди).

Оптимизатор се нарича модула, който приема команди и изпраща същите или други команди към следващия модул, като изпратените команди зависят от конкретния оптимизатор.

Строител се нарича модула, който приема команди от предишния модул и извършва дейности, които са свързани със синтактичната структура на дървото. Строител модулът е „граница“ между работата с команди и работата със синтактични структури.

Филтър се нарича модула, който приема синтактични структури и изпраща същите или други синтактични структури, като структурите, които се изпращат зависят от конкретния филтър.

2.9 Синтактични дървета

Популярно разбиране е, че абстрактните синтактични дървета може да не съдържат всички възможни възли с етикети от нетерминали и може да не съдържат определени възли, когато те се подразбират от контекста. Тези неформални дефиниции, въпреки че изразяват разликата между дърветата, оставят много неяснота в това, колко информация е „достатъчна“ за да се различат различните част на едно дърво и по какъв критерии едно конкретно дърво става абстрактно.

За да се изведе едно синтактично дърво, строителят използва факти. Дефинира се множеството от всички факти с Φ. В зависимост от взаимоотношенията между обектите в парсер граматиката се различават групи от факти, извлечени от напредналата граматика $\mathbf{A=(-,-,-,-,R,-)}$, която се използва за парсиране от парсера в машината:

- Коя алтернация е в кое правило или група;
- Коя конкатенация се намира в коя алтернация;
- Кой токен е анализиран от парсера като част от множеството, което е дефинирано от определен символ, преди токенът да бъде изпратен към строителя;
- Кой нетерминал се референцира от кой референция символ.

В зависимост от повторенията на даден символ в $\mathbf{A}$ се различават няколко видове факти. Тези факти са взаимно изключващи се за всеки отделен символ и са както следва:

- пропускаем — съществува за символ $\omega^k \in \Omega$, който може да бъде разпознат във входните данни нула или един на брой пъти. Формално има по един факт $\mathbf{(f-skippable,k)}$ за всеки от символите $\omega^k \mid \omega^k \in \Omega \wedge \mathbf{REPEAT}(\omega^k)=(0,1)$;
- единичен — съществува за символ $\omega^k \in \Omega$, който трябва да бъде във входните данни точно един път. Формално има по един $\mathbf{(f-single,k)} \in \Phi$ за всеки от символите $\omega^k \mid \omega^k \in \Omega \wedge \mathbf{REPEAT}(\omega^k)=(1,1)$;
- масив — съществува за символ $\omega^k \in \Omega$, който трябва да бъде във входните данни, когато минималният и максималният брой повторения на символа са равни, са повече от едно и са краен брой. Формално има по един $\mathbf{(f-array,k,n)} \in \Phi$ за всеки от символите $\omega^k \mid \omega^k \in \Omega \wedge \mathbf{REPEAT}(\omega^k)=(n,n) \wedge n>1 \wedge n \neq \infty$;
- списък — съществува за символ $\omega^k \in \Omega$, който трябва да бъде във входните данни, когато минималният брой повторения на символа са по-малко от максималния брой и максималният брой е по-голям от едно. Формално има по един $\mathbf{(f-list,k,n,m)} \in \Phi$ за всеки от символите $\omega^k \mid \omega^k \in \Omega \wedge \mathbf{REPEAT}(\omega^k)=(n,m) \wedge n<m \wedge m>1$;

Дефинират се фактите, които строителят притежава, с $\mathbf{K} \subseteq \Phi$. Приема се, че парсерът притежава всички знания Φ и че строителят използва цялото знание, което притежава.

Синтактичните дървета се класифицират според структурата на напредналите граматики в НПБНФ. Едно от най-важните свойства на напредналите граматики е, че те имат минимален и максимален брой повторения за отделните елементи.

Според информацията, която се съдържа в дървото, синтактичните дървета са, както следва:

- Конкретно — синтактично дърво, което е построено на базата на всички налични факти, когато **K=F**;
- Абстрактно — дърво, което е построено без използването на поне един необходим факт, когато **K<F**. Това значи, че максималният брой абстрактни дървета са $2^{|F|-1}$ (за да е сигурно, че дървото не е конкретно, едно се изважда от броя факти).

Различават се команди за подготвяне (вид **d-prepare**), разваляне (вид **d-unprepare**), вход в правило (вид **d-rule-enter**), успех на правило (вид **d-rule-success**), назад в правило (вид **d-rule-back**), неуспех на правило (вид **d-rule-fail**), вход/успех/назад/неуспех на група (вид **d-group-enter/success/back/fail**), вход/успех/назад/неуспех на конкатенация (вид **d-con-enter/success/back/fail**), токен напред (вид **d-token-front**), токен назад (вид **d-token-back**), следващ елемент (вид **d-next**), предишен елемент (вид **d-previous**), създаване/унищожаване на списък (вид **d-list-create/destroy**), създаване/унищожаване на масив (вид **d-array-create/destroy**), открита грешка (вид **d-error**), успех на анализа (вид **d-success**) и край на анализа (вид **d-done**).

2.10 Изводи

Във втора глава е представен концептуален модел на парсираща машина, която е подходяща за реализиране на различни стратегии, методи, подходи и алгоритми, които са популярни в литературата. Дефиниран е също нов вид функциониране на лексер модула, което позволява на парсер модула да анализира според регистъра на буквите в лексемите и да приема букви според диапазони от букви.

Предложено е разширение на ПБНФ, с което напредналите граматики могат да се дефинират. Предложен е модел на фразова машина, която предварително категоризира различните фрази в парсер граматиката с цел ускоряване на анализа. Предложена е промяна на известния в литературата начин на работа на скенер и лексер модулите. Дефинирани са нови модули, като доставчик и оптимизатор.

Проектираната обща архитектура на парсираща машина е ориентирана повече към типовете данните, които се приемат и изпращат от модулите в машината, и по-малко към начина им на обработка.

3 Алгоритъм Тунелно парсиране

В тази глава се предлага и дискутира алгоритъма Тунелно парсиране (Tunnel Parsing — TP) като алгоритъм, който се изпълнява от парсер модула в парсиращата машина, въведена в предната глава.

3.1 Базова концепция на алгоритъма

Целта на алгоритъма е ефективно да изпълнява стеков автомат, имащ преходи от напреднали символи и броими повторения, който е представен като свързани диаграми на преходи. За да бъде изпълнението ефективно, алгоритъмът групира определени преходи и състояния на стековия автомат в „части“.

Ситуацията се усложнява от факта, че в граматиките в ПБНФ може да има повторение на референция (например **5*8R**), като референцираното правило генерира празна дума (например **R=0*1"x"**). В тази ситуация TP анализира възможно най-много повторения като използва токени (например **"xx"**), а броят оставащи повторения (**3**) до минималния необходим (**5**) се извършват без токени. Алгоритъмът не навлиза в безкраен цикъл, когато повторението е до безкрай (например **5*R**).

3.2 Стекове

Алгоритъмът TP използва стек за изпълнение, който се състои от елементи, които се представят като n-орката (c,n) за контролно състояние c и брой архивирани елементи от стека за дълбочина $n \in \mathbb{N}_0$.

По време на изпълнение алгоритъмът TP използва стек за дълбочина, който се състои от сегменти. Един сегмент съдържа информация за операциите, които парсерът може да извърши в зависимост от текущия входен токен. Всеки сегмент се представя като n-орката (p,n,m,rm,rd,rn) за родител $p \in (N_\Omega \cup G_\Omega \cup N)$, минимален брой повторения n, максимален брой повторения m, минимален маршрутизатор rm от вида $r\text{-}minimum$, вътрешен маршрутизатор rd от вида $r\text{-}inner$ и следващ маршрутизатор rn от вида $r\text{-}next$. Ако $p \in (N_\Omega \cup G_\Omega)$, тогава $(n,m)=REPEAT(p)$ и ако $p \in N$, тогава $n=1$ и $m=1$.

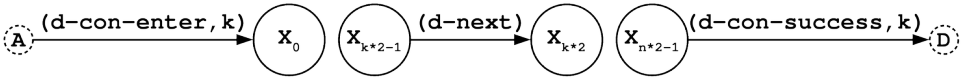
За различните анализи, които следват по-долу, се използва стек за повторения с елементи от $\mathbf{N}_0$, които се наричат броячи. Този стек съдържа броя повторения, които са вече намерени (или са в процес на намиране) за даден напреднал символ. Повторения се броят за символ $\omega \in \Omega$, когато за $(n,m)=REPEAT(\omega)$ е вярно, че $n>1 \vee (m>1 \wedge m \neq \infty)$ (повторението е броимо).

По време на изпълнение алгоритъмът TP може да прогресира назад в автоматите. За да бъде това възможно, елементите в стека за дълбочина не се изтриват, а се преместват в архивния стек за дълбочина с операция, която се нарича архивиране. Аналогично преместването на един елемент от архивния стек за дълбочина в стека за дълбочина се нарича възстановяване.

По подобен начин работи и архивния стек за повторения — когато алгоритъмът премине от разпознаването на даден напреднал символ към следващия, броят повторения, които са разпознати до момента за дадения елемент (ако има такива), се архивират в архивния стек за повторения.

3.3 Автомати

За целта на анализирането на низ от токени на базата на напреднала граматика, се създават редица от автомати. Всеки автомат се създава рекурсивно чрез прилагането на различни шаблони, от които се създават състояния и преходи с етикети.



Фигура 2: Шаблон със състояния и преходи създадени на базата на конкатенация

За всеки символ $(s\text{-}concatenation,\omega_0,\ldots,\omega_{n-1})$ се създават състояния и преходи, като се прилага шаблона на Фигура 2. Състояния A и D не се създават, а са контекста, в който шаблонът се прилага. На Фигура 2, k е поредния номер на конкатенацията в алтернативата, която използва конкатенацията. Двойките състояния X_{k*2-1} и X_{k*2}, както и преходите между тях, за $k \in [1..n-1]$, се създават само ако $n>1$. Всяка двойка X_{i*2} и X_{i*2+1}, за $i \in [0..n-1]$ е контекста, в който се прилага шаблона за ω_i.

Родителят на състоянието, което е създадено при прилагането на даден шаблон за символ $\omega \mid (H_\Omega \cup N_\Omega \cup G_\Omega \cup Y_\Omega)$ е $p \mid p \in (R \cup G_\Omega) \wedge p \mapsto \alpha \mapsto \beta \wedge (\beta \mapsto \omega \vee \beta \mapsto \gamma \mapsto \omega)$ за $\Pi(\alpha)=s\text{-}alternation$, $\Pi(\beta)=s\text{-}concatenation$ и $\Pi(\gamma)=s\text{-}repeat$.

От тук надолу се работи само с редуцирани и добре дефинирани напреднали граматики, които нямат лява рекурсия. Това премахва определени видове ε-цикли (но много други видове остават), които могат да съществуват в автомата, който е създаден на база на граматиката.

3.3.1 Намиране на най-кратките пътища

Избира се, че ако има два ε-пътя с еднакви по брой ε-преходи, които преминават през състоянията, които са създадени за две различни конкатенации в една и съща алтернация, тогава по-краткият път е този, който преминава през състоянията за конкатенацията с по-малък пореден номер в алтернацията.

По време на генерацията на парсера, първо се намира най-краткия ε-път, ако има такъв, от началното до финалното състояние, които са създадени за всяко отделно правило. След това се намира най-краткия ε-път, ако има такъв, от началното до финалното състояние, които са създадени за всяка отделна група.

На базата на намерените ε-пътища за правила и групи се намират най-кратките ε-пътища за прескачането на всеки отделен символ ω, който се намира в дадена конкатенация. Това се прави като се търси най-краткия ε-път между състояния $\mathbf{X}_{i*2}$ и $\mathbf{X}_{i*2+1}$, които са създадени при прилагането на шаблона на Фигура 2 за символ ω_i в конкатенация **k=(s-concatenation,**$\omega_0,\ldots,\omega_{n-1}$**)**, където **i∈[0..n)**. На базата на всички намерени ε-пътища до момента се намират ε-пътищата за всеки символ ω_i от състоянието след ω_i до финалното състояние за правилото (или групата), която използва алтернацията, която използва **k**.

3.3.2 Достижими напреднали символи

Всеки символ $\omega \in \mathbf{H}_\Omega$, който е етикета на даден преход **t**, се нарича достижим от състояние **q**, когато съществува поне една редица от нула или повече ε-преходи от **q** до началото на **t**. Ключово състояние се нарича всяко състояние, което е начално (на правило или група) и всяко състояние след даден символ, което е създадено при прилагането на шаблона на Фигура 2.

По време на генерацията на парсера се извършва търсене за достижимите състояния от всяко ключово състояние. Търсенето е подобно на популярното търсенето в дълбочина, но с няколко разлики:

1. Единственият начин търсенето да прескочи даден символ (да премине през състоянията и преходите, които са създадени за символа) е като се премине по ε-пътя на символа, ако има такъв;
2. Преходите **(d-con-enter,k)** с по-малко **k**, които не са част от избрания ε-път (ако има такъв) за родителя на **q**, се използват за търсене първи;
3. При търсене от дадено състояние **q**, преходът в избрания ε-път (ако има такъв) за родителя на **q** се използва за търсене последен.

Резултатът от търсенето на достижимите символи от състояние **q** е достижимото дърво на **q**, в което наследниците на всеки възел са подредени по реда им на намиране. Етикетът на дадено листо в достижимото дърво е равен на етикета на прехода, който е използван за намирането на листото — символа $\omega \in \mathbf{H}_\Omega$.

От начина на създаване на автоматите за напреднала граматика и дефинициите за достижими напреднали символи рекурсивно следва, че всеки достижим напреднал символ от дадено начално състояние не е в конфликт със себе си в следната ситуация:

1. Състоянието е създадено за група (или за правило, което се референцира от референция);
2. Групата (или референцията) се повтаря поне два пъти;
3. За групата (или за правилото) има избран ε-път.

3.3.3 Конфликти

В напредналите граматики има буква, фраза и край символи, както и броими повторения за тях. Като следствие от това, по време на парсиране на базата на тези граматики стават

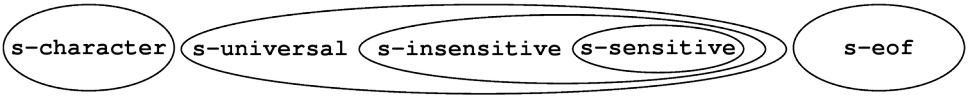
възможни различни нови видове конфликти между символи, които не се получават по време на работата на други алгоритми за парсиране, използващи други граматики.

Ако в достижимото дърво на състояние $\mathbf{q}$ за символ $\omega_a \in H_\Omega$ в етикета на листо l_a и символ $\omega_b \in H_\Omega$ в етикета на листо $l_b | l_a \neq l_b$ е вярно, че $\Psi(\omega_a) \cap \Psi(\omega_b) \neq \emptyset$, тогава се казва, че ω_a и ω_b са в конфликт от $\mathbf{q}$.

Разглежда се множеството $\mathbf{E}$ от листата на едно достижимо дърво на $\mathbf{q}$. По време на генерацията на парсера от това множество $\mathbf{E}$ се извеждат всички различни конфликти $\mathbf{E_i}$. Всеки конфликт $\mathbf{E_i}$ съдържа подредено множество от листа в $\mathbf{E}$, по реда им на намиране, които имат етикети от напреднали символи, както и информация (показана по-долу) за конкретния конфликт.

Различават се няколко вида конфликти, които са представени като n-орки, в които първият елемент е вида на конфликта $\mathbf{l \in L}$, за $\mathbf{L=\{l\text{-}character, l\text{-}sensitive, l\text{-}insensitive, l\text{-}universal, l\text{-}eof\}}$, а последният елемент $\mathbf{s}$ е подредено множество от листата в $\mathbf{E}$ с етикети от напредналите символи, които са в конфликт:

- $\mathbf{(l\text{-}character, f, t, s)}$ — буква конфликт с $\mathbf{f, t \in U}$ и $\mathbf{f \leq t}$, който съдържа подреденото множеството $\mathbf{s}$ от листа в $\mathbf{E}$ с етикети $\mathbf{(s\text{-}character, x, y, m)}$, такива че $\mathbf{f \leq r \leq t}$ за поне едно $\mathbf{r | r \in \Delta(x, y, m)}$;
- $\mathbf{(l\text{-}sensitive, c, n, s)}$ — чувствителен конфликт за категория $\mathbf{c \in C}$, фразов индекс $\mathbf{n = SENSITIVE(p)}$ с фраза $\mathbf{p}$ и подреденото множеството $\mathbf{s}$ от листа в $\mathbf{E}$ с етикети от:
  - Поне един символ $\mathbf{(s\text{-}phrase, c, p_n, \equiv) \in PS_\Omega}$;
  - Нула или повече символи $\mathbf{(s\text{-}phrase, c, q, \approx) \in PI_\Omega}$, такива че $\mathbf{n = SENSITIVE(j)}$ за поне едно $\mathbf{j \in \Delta(q, \approx)}$;
  - Нула или повече символи $\mathbf{(s\text{-}phrase, c, -, -) \in PA_\Omega}$;
- $\mathbf{(l\text{-}insensitive, c, n, s)}$ — нечувствителен конфликт за категория $\mathbf{c \in C}$, фразов индекс $\mathbf{n = INSENSITIVE(p)}$ за фраза с малки букви $\mathbf{p}$ и подреденото множеството $\mathbf{s}$ от листа в $\mathbf{E}$ с етикети от:
  - Поне един символ $\mathbf{(s\text{-}phrase, c, q_n, \approx) \in PI_\Omega}$;
  - Нула или повече символи $\mathbf{(s\text{-}phrase, c, -, -) \in PA_\Omega}$;
- $\mathbf{(l\text{-}universal, c, s)}$ — универсален конфликт за категория $\mathbf{c \in C}$ и подреденото множеството $\mathbf{s}$ от листа в $\mathbf{E}$ с етикети от поне един символ $\mathbf{(s\text{-}phrase, c, -, -) \in PA_\Omega}$;
- $\mathbf{(l\text{-}eof, s)}$ — край конфликт за подреденото множеството $\mathbf{s}$ от листа в $\mathbf{E}$ с етикети от



Фигура 3: Възможни конфликти между напреднали символи

поне един символ $\mathbf{(s\text{-}eof) \in F_\Omega}$.

Това, че даден токен $\mathbf{h \in H}$ принадлежи на даден конфликт $\mathbf{E_i}$ се означава с $\mathbf{h \in E_i}$. Случаите на принадлежност на токен в конфликт са, както следва:

- $\mathbf{(t\text{-}character, u) \in E_i}$, ако и само ако $\mathbf{E_i = (l\text{-}character, f, t, -)}$ и $\mathbf{f \leq u \leq t}$;
- $\mathbf{(t\text{-}sequence, c, e) \in E_i}$, ако и само ако:
  - $\mathbf{E_i = (l\text{-}sensitive, c, n, -)}$ с $\mathbf{n = SENSITIVE(e)}$; или
  - $\mathbf{E_i = (l\text{-}insensitive, c, n, -)}$ с $\mathbf{n = INSENSITIVE(e)}$; или
  - $\mathbf{E_i = (l\text{-}universal, c, -)}$;
- $\mathbf{(t\text{-}eof) \in E_i}$, ако и само ако $\mathbf{E_i = (l\text{-}eof, -)}$.

3.3.4 Дървета с празни възли

Едно оригинално синтактично дърво е ε-кондензирано, когато всеки ε-възел в синтактичното дървото е оптимален според следните критерии за оптималност:

1. Всеки ε-възел в дървото има възможно най-малко подвъзли според граматиката, от която е построен;
2. Ако повече от една конкатенация в дадена алтернация може да бъде база за създаването на същия брой ε-възли, тогава се създават възлите на базата на конкатенацията с по-малък индекс в алтернацията;
3. Възлите, които не са ε-възли, са подредени преди ε-възлите, когато всички тези възли са създадени на базата на същия символ (като следствие от това се предотвратява вероятна комбинаторна експлозия).

Ако две различни синтактични дървета са построени на базата на една и съща граматика и за едни и същи входни токени, те станат напълно еднакви, когато се трансформират в ε-кондензирана форма, тогава те са ε-еквивалентни. Трансформацията е замяна на всички ε-възли с техния оптимален вариант.

3.4 Тунели

Информацията, която се съдържа във всеки тунел е за промяна на различните стекове, които алгоритъмът използва и за командите, които могат да се изпратят към следващия модул. Един тунел се представя като n-орката **(t,e,o,d,a)** за вид на тунела **t∈{τ–first,τ–inner,τ–last,τ–ε–inner,τ–ε–last,τ–ε–direct–front,τ–ε–direct–back,τ–ε–main–front,τ–ε–main–back}**, редица от команди **e**, редица от операции **o** над **O**, брой елементи за премахване **d∈**$\mathbb{N}_0$, които трябва да се премахнат от стека за повторение и брой елементи за добавяне **a∈**$\mathbb{N}_0$ (всеки със стойност едно), които трябва да се добавят в стека за повторение (след като се премахнат **d** на брой елементи).

По време на генерацията на парсера се извличат различни видове тунели на базата на подреденото множеството **s** в даден конфликт $\mathbf{E_i}$, който е елемент в непразно множеството от конфликти **E**, които са изведени от листата на достижимото дърво **Z** за ключово състояние **q** с движение по **Z**. За целта на извличането се създава стек за извличане, който е съставен от сегменти и:

- Ако **q** е състояние за правило **B→α**, тогава сегментът на **B** се добавя в стека за извличане; или
- Ако **q** е състояние за група, тогава сегментът на групата се добавя в стека за извличане.

По време на извличането на тунелите, в стека за извличане новите добавени сегменти са само за референции или групи. Тунелите за конфликт $\mathbf{E_i}$ се извличат по следния начин:

- Извлича се тунел от вида **τ–first** от **q** до първия елемент в **s**;
- Извлича се тунел от вида **τ–inner** между всеки два съседни елемента в **s** (от даден елемент до следващия в подреденото множество);
- Ако от **q** няма избран ε-път до финалното състояние за родителя на **q**, тогава се извлича тунел от вида **τ–last** от последния елемент в **s** до **q**;
- Ако от **q** има избран ε-път до финалното състояние за родителя на **q**, тогава се извлича тунел от вида **τ–ε–inner** от последния елемент в **s** до края на ε-пътя. За всеки тунел от вида **τ–ε–inner** се извлича тунел от вида **τ–ε–last** от финалното състояние за родителя на **q** до **q**.

Ако за ключово и не начално състояние **q** има избран ε-път до финалното състояние за родителя на **q** и множеството от конфликти за **q** е празно, тогава се извлича тунел от вида **τ–ε–direct–front** от **q** до това финално състояние. В обратната посока се извлича тунел от вида **τ–ε–direct–back**.

За всяко правило (или група), за което има избран ε-път, се извлича тунел от вида **τ–ε–main–front** от началното до финалното състояние. В обратната посока се извлича тунел от вида **τ–ε–main–back**.

3.5 Маршрутизатори

За всяко ключово състояние **q** се създава обект, който се нарича маршрутизатор. Маршрутизаторите имат за цел да съдържат предварително изчислена информация за това как по време на парсиране, парсерът може да продължи анализа от дадено автоматно състояние. Един маршрутизатор се представя като n-орката (t,p,c_ε) за вид на маршрутизатора $t \in \{r\text{-}origin, r\text{-}minimum, r\text{-}inner, r\text{-}next\}$, подредено множество от пътища **p** и контролно състояние за продължение c_ε. Дефинира се път в маршрутизатор като $E_i \to c$ за конфликт E_i, който е изведен от **q** и контролно състояние (накратко к-състояние) **c**.

Различните видове маршрутизатори се създават в следните ситуации:

- **r-origin** — за всяко начално правило в **S**;
- **r-minimum** — за всеки символ $\omega \mid \omega \in (N_\Omega \cup G_\Omega) \wedge REPEAT(\omega) = (n,m) \wedge n > 1$ (за всяка референция или група, която се повтаря поне два пъти);
- **r-inner** — за всеки символ $\omega \mid \omega \in (N_\Omega \cup G_\Omega) \wedge REPEAT(\omega) = (n,m) \wedge n < m \wedge m > 1$ (за всяка референция или група, за която минималният брой повторения е по-малък от максималния брой повторения и максималният брой повторения е повече от едно);
- **r-next** — за всеки символ $\omega \mid \omega \in (N_\Omega \cup G_\Omega \cup H_\Omega)$ (за всички референции, групи и всички напреднали символи, които директно дефинират токени).

Пътищата в различните видове маршрутизатори се създават, както следва:

- В **r-origin**, **r-minimum** и **r-inner** се създава по един път за всяко първо листо в подреденото множеството от листа **s** във всеки конфликт E_i, който може да се изведе от листата **E** на достижимото дърво на началното състояние на правилото (или групата), за което е създаден маршрутизатора;
- В **r-next** се създава по един път за всяко първо листо в подреденото множеството от листа **s** във всеки конфликт E_i, който може да се изведе от листата **E** на достижимото дърво на състоянието след символа, за който е създаден маршрутизатора.

3.6 Контролни обекти

Алгоритъмът TP използва множество от контролни обекти (накратко к-обекти), които използват различни тунели и маршрутизатори. Всеки контролен обект се състои от поне едно контролно състояние.

Броят на контролните състояния в един контролен обект зависи от контролния обект. Всеки контролен обект показва „къде“ в автоматите се намира парсера, а всяко контролно състояние дефинира „какви“ операции трябва да се изпълнят. Всеки тунел се изпълнява в определен контекст, който представлява всички стекове в даден момент.

Всички операции, които са дефинирани от различните к-състояния, са такива, че парсерът следва точно автоматите, които са създадени за напредналата граматика, като винаги се движи по най-краткия път от едно състояние до друго.

Произход контролен обект

Създава се по един произход к-обект за всяко начално правило $J \in S$. Преди парсерът да започне да анализира входните токени, в зависимост от избраното начално правило **J**, парсерът добавя създадения произход к-обект (създаден за **J**) като първи елемент в стека за изпълнение. Произход к-обектът е n-орката $(c\text{-}origin, \{use\}, r)$ за едно к-състояние **use** и маршрутизатор **r** от вида **r-origin**.

Терминал контролен обект

Създава се по един терминал к-обект за всяко терминално състояние в автоматите. Целта на този к-обект е да увеличи стека за изпълнение с един елемент преди парсерът да

започне работа със следващия токен. Терминал к-обектът е n-орката **(c–terminal, {use},r)** за едно к-състояние **use** и маршрутизатор **r** от вида **r–next**.

Токен контролен обект

Създава се по един токен к-обект за всяко листо в подреденото множество от листа **s** във всеки конфликт $\mathbf{E}_i$, който може да се изведе от листата **E** на дадено достижимо дърво, когато символът в етикета на листото има максимален брой повторения равен на едно. Токен к-обектът е n-орката **(c–token,{use,used},n,t,τ)** за две к-състояния (**use** и **used**), следващ к-обект **n**, к-обект **t** от вида **c–terminal** и тунел **τ** от вида **τ–first** или **τ–inner**.

Партида контролен обект

Създават се партида к-обекти подобно на токен к-обектите, но когато етикетът на листото има максимален брой повторения повече от едно. Този к-обект има сложна функционалност, защото извършва повторение на **ω** и напред, и назад, без помощта на други к-обекти. Начинът на работа на к-състоянията в този обект е такъв, че те приемат възможно най-много **h∈Ψ(ω)**. Партида к-обектът е n-орката **(c–batch, {use,repeat,back,used},n,t,τ)** за четирите вида к-състояния, следващ к-обект **n**, к-обект **t** от вида **c–terminal** и тунел **τ** от вида **τ–first** или **τ–inner**.

Епсилон-произход контролен обект

Създават се епсилон-произход к-обекти за всяко $\mathbf{c}_\varepsilon$ в маршрутизатор **r–origin**, който е създаден за дадено начално правило **J∈S**, за което има избран ε-път, защото **J⇒*ε**. Целта на този к-обект е да бъде използван, когато парсерът започва парсирането за начално състояние **J**, но няма път в споменатия маршрутизатор за първия токен от входните данни. Епсилон-произход к-обектът е n-орката **(c–epsilon–origin,{use,used},τf,τb)** за двата вида к-състояния, тунел за напред **τf** от вида **τ–ε–main–front** и тунел за назад **τb** от вида **τ–ε–main–back**.

Епсилон-следващ контролен обект

Създават се епсилон-следващ к-обекти за всяко ключово състояние **q** (различно от началното на правило или група), от което има избран ε-път до финалното състояние за родителя на **q**. Единственото к-състояние на този вид к-обект се използва като $\mathbf{c}_\varepsilon$ в маршрутизатора **r** от вида **r–next**, който е създаден за **q**. След като парсерът се намира в състояние **q**, тогава парсерът ще търси път в **r** за текущия токен. Ако път не е намерен, тогава единственото к-състояние на този обект ще замени върха на стека за изпълнение. Епсилон-следващ к-обектът е n-орката **(c–epsilon–next,{use},τf,τb)** за едно к-състояние, тунел за напред **τf** от вида **τ–ε–inner** и тунел за назад **τb** от вида **τ–ε–last**.

Епсилон-запълване контролен обект

Създава се по един епсилон-запълване к-обект за всеки маршрутизатор от вида **r–minimum**, когато правилото или групата, за което се създава този маршрутизатор има ε-път. Единственото к-състояние се използва като $\mathbf{c}_\varepsilon$ в споменатия маршрутизатор и никога не се добавя в стека за изпълнение, защото се използва от други к-обекти. Епсилон-запълване к-обектът е n-орката **(c–epsilon–fill,{use},τf,τb)** за едно к-състояние, тунел за напред **τf** от вида **τ–ε–main–front** и тунел за назад **τb** от вида **τ–ε–main–back**.

Проход-произход контролен обект

Създава се един проход-произход к-обект в края на списък от **c–token** или **c–batch** к-обекти в маршрутизатор от вида **r–origin**, който е създаден за дадено начално правило **J∈S**, за което има избран ε-път, защото **J⇒*ε**. За да се стигне до употребата на този к-обект

парсерът е изпълнил всички възможни начини за анализиране на входните токени и е прогресирал назад, докато е достигнал началното състояние за началното правило, което има избран ε-път. Проход-произход к-обектът е n-орката **(c-passage-origin,{use,used},τf,τb)** за две к-състояния, тунел за напред **τf** от вида **τ-ε-inner** и тунел за назад **τb** от вида **τ-ε-last**.

Проход-минимум контролен обект

Създава се един проход-минимум к-обект в края на списъка от **c-token** или **c-batch** к-обекти в маршрутизатор **r** (от вида **r-minimum**), който е създаден за дадено начално правило **J∈S**, за което има избран ε-път, защото **J⇒*ε**. За да се стигне до употребата на този к-обект парсерът е използвал всички **c-token** и **c-batch** к-обекти, които са подредени в списък от к-обекти в минималния маршрутизатор **r**. Проход-минимум к-обектът е n-орката **(c-passage-minimum,{use},τ)** за едно к-състояние и тунел **τ** от вида **τ-ε-inner**.

Проход-следващ контролен обект

Създава се един проход-следващ к-обект в края на списъка от **c-token** или **c-batch** к-обекти в маршрутизатор от вида **r-next**, който е създаден за дадено състояние **q**, от което има избран ε-път до финалното състояние за родителя на **q**. За да се стигне до употребата на този к-обект, парсерът е изпълнил всички възможни начини за анализиране на входните токени след даденото състояние и прогресира назад. Проход-следващ к-обектът е n-орката **(c-passage-next,{use},τ)** за едно к-състояния и тунел **τ** от вида **τ-ε-inner**.

Назад-произход контролен обект

Създава се един назад-произход к-обект в края на списъка от **c-token** и **c-batch** контролни обекти в маршрутизатор от вида **r-origin**, който е създаден за дадено начално правило, за което няма избран ε-път. Този к-обект е подобен на проход-произход к-обекта с разликата, че няма избран ε-път за началното правило и поради тази причина, парсерът преминава от състоянието за последния **c-token** или **c-batch** к-обект директно в началното състояние на началното правило. Назад-произход к-обектът е n-орката **(c-back-origin,{use},τ)** за едно к-състояние и тунел **τ** от вида **τ-last**.

Назад-универсален контролен обект

Създава се един назад-универсален к-обект в края на списъка от **c-token** или **c-batch** к-обекти в маршрутизатор от вида **r-next**, който е създаден за състояние **q** след символ **ω∈(N$_ω$∪G$_ω$∪H$_ω$)**, като от **q** няма избран ε-път до финалното състояние за родителя на **q** и за **ω∈(N$_ω$∪G$_ω$)** не се броят повторения или **ω∈H$_ω$**. Когато парсерът прогресира назад и се връща в състояние **q**, тогава този к-обект изпълнява тунела от финалното състояние за родителя на **q** до **q**. Назад-универсален к-обектът е n-орката **(c-back-universal,{use},τ)** за едно к-състояние и тунел **τ** от вида **τ-last**.

Назад-броим контролен обект

Създава се един назад-броим к-обект в края на списъка от **c-token** или **c-batch** к-обекти в маршрутизатор **r** от вида **r-next**, който е създаден за състояние **q**, след символ **ω∈(N$_ω$∪G$_ω$)** със сегмент **g**, от което няма ε-път до финалното състояние за родителя на **q** и за **ω** се броят повторения. Функционалността на този к-обект е подобна на назад-универсален к-обект с разликата, че парсерът възстановява един елемент в стека за повторение и прави ε-запълване назад, ако е необходимо. Назад-броим к-обектът е n-орката **(c-back-countable,{use},g,τ)** за едно к-състояние, споменатия сегмент **g** и тунел **τ** от вида **τ-last**.

Назад-минимум контролен обект

Създава се един назад-минимум к-обект в края на списъка от **`c-token`** или **`c-batch`** к-обекти в маршрутизатор от вида **`r-minimum`**, който е създаден за дадено правило (или група), за което няма избран ε-път. За да се стигне до употребата на този к-обект, парсерът е изпълнил всички възможни начини за анализиране на входните токени с повторения (за анализиране на минималния брой повторения) на даденото правило или група и вече прогресира назад. Назад-минимум к-обектът е n-орката **`(c-back-minimum,{use},t)`** за едно к-състояние и тунел **`t`** от вида **`t-last`**.

Назад-вътрешен контролен обект

Създава се един назад-вътрешен к-обект в края на списъка от **`c-token`** или **`c-batch`** к-обекти в маршрутизатор от вида **`r-inner`**, който е създаден за дадена референция (или група), за която няма избран ε-път. За да се стигне до употребата на този к-обект парсерът е изпълнил всички възможни начини за анализиране на входните токени чрез повторение (за анализиране на повече от минималния брой повторения) на даденото правило (или група) и вече прогресира назад. Назад-вътрешен к-обектът е n-орката **`(c-back-inner,{use},t)`** за едно к-състояние и тунел **`t`** от вида **`t-last`**.

Развиване контролен обект

Създава се само един глобален развиване к-обект в контролния слой. По време на изпълнението на операциите, които са дефинирани от този к-обект, парсерът архивира стека за дълбочина (подобно на излизането от функция). Ако за родителя на архивирания сегмент не са анализирани минималния брой повторения, тогава парсерът опитва да направи повторение на родителя. Ако повторение не може да се направи, тогава ако е възможно, парсерът изпълнява ε-запълване и след него опитва да продължи парсирането след родителя на сегмента. Ако минималният брой повторения за родителя на архивирания сегмент са вече анализирани, но максималният брой не са, тогава парсерът опитва да направи повторение на родителя, но при неуспех не изпълнява ε-запълване. Ако повторение не е възможно, тогава парсерът опитва да продължи след родителя на сегмента. Ако продължаването след родителя на сегмента не е възможно, парсерът ще започне движение назад. Развиване к-обектът е n-орката **`(c-unwind,{use})`** с едно к-състояние.

Възстановяване контролен обект

По време на генерацията на парсера се създава само един глобален възстановяване к-обект в контролния слой. Този к-обект опитва да възстанови един сегмент на всяка стъпка, като едновременно с това изпълни, ако е необходимо, тунел за назад от финалното състояние на правилото (или групата), в която се намира родителя на сегмента до състоянието след родителя на сегмента и заедно с това, ако е необходимо, да изпълни ε-запълване назад в зависимост от броя анализирани до момента повторения на родителя на сегмента. Възстановяване к-обектът е n-орката **`(c-restore,{incomplete,complete})`** с две к-състояния.

3.7 Парсиране

След като всички стекове са подготвени, всички тунели са извлечени, всички маршрутизатори и целия контролен слой са създадени, тогава парсерът, който работи с алгоритъма TP, може да започне да анализира входни токени. Парсирането започва с поставянето на произход к-обект, който е създаден за избраното начално правило **`J∈S`** и продължава, докато парсерът изпрати команда **`(d-done)`**. Ако парсерът има за цел да намери само едно синтактично дърво, тогава парсирането приключва при първото изпращане на **`(d-success,true)`**. Ако парсерът има за цел да парсира, докато намери първата грешка

във входните токени, тогава парсирането приключва след изпращането на първата команда **`(d-error)`**.

3.8 Свойства на Тунелното парсиране

Парсиращата машина, която работи с алгоритъма TP, използва контролните обекти и техните състояния, тунелите и маршрутизаторите, за да преминава от едно вътрешно състояния в друго. Според всички дефиниции по-горе:

- Алгоритъмът парсира на базата всяка не ляво рекурсивна добре дефинирана и редуцирана напреднала граматика;
- Паметта използвана от алгоритъма е линейна на броя токени в най-лошия случай;
- Алгоритъмът парсира за линейно време на базата на всяка граматика, която може да се изведе от детерминиран стеков автомат и ако в граматиката няма рекурсия, парсирането може да се изпълни с константен обем памет;
- Времето за парсиране е експоненциално в най-лошия случай за някои входни данни, които се парсират на базата на определени граматики. За справяне с експоненциалното време за парсиране може да се използва мемоизация;
- Някои **`LL(k>1)`** граматики могат да се парсират с **`k=1`** токена поглед напред чрез тривиална оптимизация;
- Алгоритъмът може да парсира детерминирано и за линейно време на базата на някои многозначни граматики и в този случаи извежда командите за създаването на ε-кондензирано дърво.

В алгоритъма TP броят на операциите, които се изпълняват от парсера на всяка итеративна стъпка не е зависим от броя входни символи.

Възлите, които не са ε-възли, са подредени преди ε-възлите, когато всички възли са създадени за даден символ, като това е следствие от начина на работа на развиване к-обекта, начина за извличане на тунели и дефинициите за достижими символи.

3.9 Изводи

Алгоритъмът TP предлага ефективно изпълнение на стеков автомат с фраза символи и броими повторения, който е представен като свързани диаграми на преходи.

В главата е показано използването на шаблони за създаване на автомати от правилата в напреднала граматика. Анализът може да бъде чувствителен или нечувствителен към регистъра на буквите в лексемите. Поддържат се диапазони от букви директно в парсер граматиката. Определен е нов подклас на безконтекстните граматики на базата, на който могат да се генерират низовете от токени, които алгоритъмът TP може да парсира за линейно време и памет.

4 Профилиране на автоматично генерирани парсери

Интерес представляват следните практически въпроси: колко памет се използва от синтактичните дървета, генерирани от конкретен парсер, каква информация съдържат и колко време отнема създаването им и работата с тях.

За да се избегне разработването на голям брой граматики на ръка (което освен това може да увеличи риска от механични грешки), се използва специално създаден за целта на дисертацията инструмент, който се нарича парсер генератор профилатор (накратко само профилатор). С помощта на профилатора става възможно извършването на голям брой тестове на различни парсиращи машини, които са генерирани от различни парсер генератори.

Грубо казано, профилаторът позволява или експлицитното въвеждане на една или няколко граматики или програмното извеждане на съвкупност от граматики, различаващи се по избрани елементи. След това за всяка от граматиките профилаторът създава изпълними кодове на един или повече съответстващи парсери, като използва външни програми.

Пускайки различни данни през различните парсери може да се получи сравнителна информация за използваните ресурси. По-долу този процес е описан в детайли.

За целта на профилирането се създават шаблонни граматики, които описват една или повече граматики в кратка форма. Всяка шаблонна граматика е представена като скрипт (низ от букви), който е валиден според специално създаден за целта на дисертацията скриптов език, който се нарича шаблонен граматичен език (накратко шаблонен език). Шаблонните граматики се програмират императивно на шаблонния език.

Граматиките в ПБНФ са директно дефинирани — не са необходими допълнителни изчисления за да се използва граматиката. За разлика от тях, шаблонните граматики дефинират една или повече граматики в кратка форма и поради тази причина са необходими допълнителни изчисления (оценки) за да се изведат всички обектни граматики, които са дефинирани от дадената шаблонна граматика. Обектните граматики са граматики, за които не е необходима оценка.

4.1 Токени

Лексер модулът в парсиращата машина работи като продължителен лексер, дефиниран по-горе. Този продължителен лексер изпраща лимит токен (вид **`t-limit`**), когато броят букви, за които лексерът не успява уникално да установи на кое правило принадлежат е повече от $2^{24}-1$, което е повече от достатъчно. На Фигура 4 е показана граматиката в спецификацията на лексера.

```
identifier   = ('a'-'z' / 'A'-'Z' / '_')
               *('a'-'z' / 'A'-'Z' / '_' / '0'-'9')
number       = 1*('0'-'9')
comment      = "//"
```

Фигура 4: Лексер граматиката в лексер спецификацията на шаблонния език

4.2 Език на шаблонна граматика

В парсер граматиката се използват фраза символи, за да разграничат идентификаторите, които са ключови думи, от останалите. Ключови думи в език са: **`rule`**, **`token`**, **`template`**, **`input`**, **`out`**, **`define`**, **`in`**, **`if`**, **`then`** и **`else`**.

Шаблонният език позволява извикването на различни шаблони (подобни на функции) на различни места в скрипта. Извикването на шаблона ще се изпълни по време на оценяване и резултатът от извикването ще се запише директно в скрипта на граматиката на мястото на извикването.

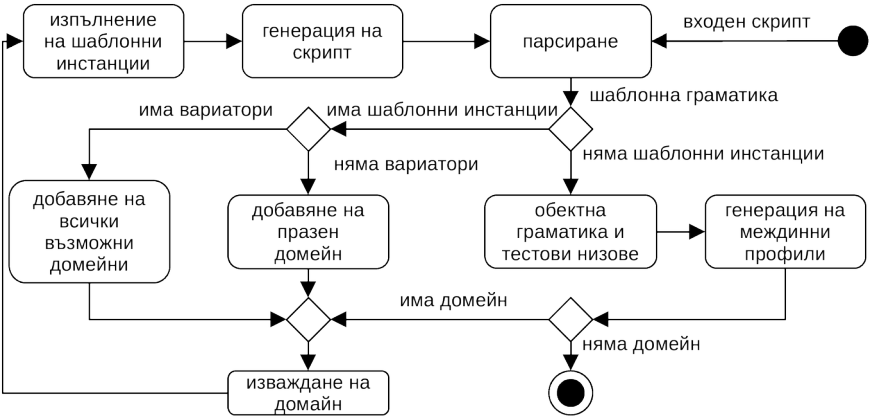
4.3 Профилатор на парсер генератори

Профилаторът на парсер генератори (Parser Generator Profiler) е компютърна програма, написана на C++, която според даден скрипт на шаблонна граматика, първо генерира редицата от обектни граматики и редица от входни данни. Парсирането на скрипта се извършва с алгоритъма TP. След това всяка обектна граматика се транслира автоматично от профилатора до вида, който се използва от различни парсер генератори. Тези транслирани граматики се наричат междинни профили. Всеки междинен профил става входни данни за различните парсер генератори, които генерират файлове, които се наричат изходни профили. Всеки изходен профил е написан на даден програмен език и впоследствие се компилира с компилатора на съответния език с различни опции за компилиране (32/64 бита, ниво на оптимизация и др.). Компилираните файлове се наричат целеви профили. Всяко изпълнение на всеки целеви профил се нарича тест. След успешното завършване на всички тестове, визуализацията на използваните ресурси (време и памет) се извършва от потребителя.

```
ECODE PGP_Machine::Run(void) {
  while(true) {
    ECODE ecode = Progress();           // try to do a step
    if (ecode == E_DONE) return E_DONE; // return when done
    if (ecode != E_OK) return ecode;    // return on an error
  }
}
ECODE PGP_Machine::Progress(void) {
  switch(FState) {
    case JS_ERROR: return E_ERROR;  // had an error
    case JS_DONE : return E_DONE;   // already done
    default: if (!Step()){ FState = JS_ERROR; return E_ERROR;}
  }
  return E_OK; // made a progress
}
bool PGP_Machine::Done(void){ FState = JS_DONE; return true;}
```

Фигура 5: Функции за контрол на итерацията

Имплементацията на езика използва итерация [1, с.43г] за всичките си основни функционалности, защото това позволява на шаблонния език да се разработват шаблонни граматики с голям брой граматични елементи. Различни итеративно работещи машини в профилатора, наследяват класа PGP_Machine, който контролира итерацията, като част от този код е показан на Фигура 5.

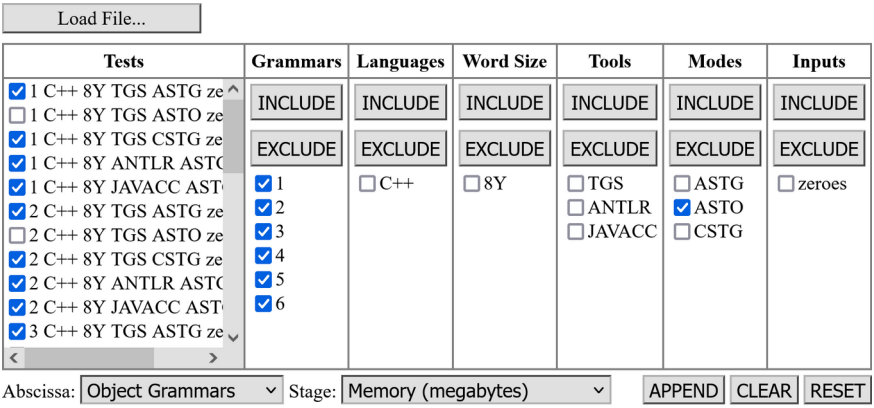


Фигура 6: Първа основна функционалност на профилатора

Диаграма с различните стъпки, които са свързани с обработката на шаблонните граматики, които могат да се изпълнят от профилатора, е показана на Фигура 6, където домейн е множеството от стойности на вариаторите в граматиката заедно с текущата шаблонната граматика.

4.4 Визуализатор

За визуализация на тестовите резултати се използва специално създадена за целта помощна програма (визуализатор), която е написана на JavaScript, HTML и CSS.



Фигура 7: Графичен интерфейс на визуализатора със заредени тестови резултати

4.5 Експерименти

Измерване на ресурсите, които са използвани от парсер машини по време на изпълнение е необходимо, защото има значение по практически причини, дали парсирането ще приключи за секунда или две [16].

Настройките на различните парсери, парсер генератори и компилатори са обикновено голям брой, което затруднява определянето на влиянието на различните комбинации от настройки върху произволен парсер, който парсира на базата на произволна граматика и анализира произволни данни в произволна софтуерната среда на произволен хардуер и е генериран и компилиран от конкретен парсер генератор и компилатор с произволни настройки. Поради тази причина се правят експерименти с възможно най-кратки граматики, за да се елиминира „шума“ в резултатите, когато в граматиката има голям брой различни елементи.

Проведени са четири експеримента. Дизайнът на всеки експеримент е с цел да се измерват използваните ресурси (време и памет) от различните парсери, генерирани от различни парсер генератори за различни обектни граматики, които са изведени от различни шаблонни граматики.

Експеримент 1. Ресурси по време на транслация

Целта на експеримента е даде емпирична представа за ефективността на различните парсери, без да се взима в предвид сложността на лексера и парсера в парсер машината, както и не според тяхното описание в литературата или документацията на парсер генератора, а на база на техните практически имплементации.

Експеримент 2. Ресурси при парсиране в дълбочина

Целта на експеримента е да покаже, колко ресурси се използват за “дълбоко” парсиране и строеж на различни видове синтактични дървета с такава структура.

Експеримент 3. Ресурси при конкатенация

Целта на експеримента е да покаже, до колко броят на елементите в една конкатенация влияе на използваните ресурси.

Експеримент 4. Ресурси при пропускаеми елементи

Целта на експеримента е да покаже използваните ресурси, когато граматиката има пропускаеми елементи, които никога не се откриват във входните данни.

4.6 Изводи

В четвърта глава е представено проектирането и разработването на език за метапрограмиране на граматики, на който се програмират шаблонните граматики, необходими за извършване на експериментите в дисертацията.

Профилаторът събира измерванията на използваните ресурсите, които се използват по време на разпознаване и парсиране на базата на различни безконтекстни граматики. Визуализирането на събраните измервания се извършва със специална помощна програма, също създадена за целта на дисертацията.

Заключение

В дисертацията е показан лексикален и синтактичен анализ на данни, който се изпълнява от парсираща машина, в която има парсер модул, който работи с алгоритъма Тунелно парсиране. Предложена е обща архитектура на парсираща машина. Функционалностите на модулите в парсиращата машина и обектите, които се изпращат и приемат от модулите, са описани в детайли.

Показан е шаблонен език, на който могат да се дефинират и/или програмират шаблонни граматики и инструмент (профилатор на парсер генератори), който извежда една или повече обектни граматики от дадена шаблона граматика. Профилаторът може да извършва експерименти с различни парсер машини, които са генерирани от различни парсер генератори на базата на изведените обектни граматики.

Добро свойство на итеративното парсиране е че след всяка итеративна стъпка може да се прави пауза. Друго свойство на итеративното парсиране е че данните, с които алгоритъмът работи са налични в инстанции на структури от данни (не в стека на нишката) и могат по-лесно да се запазят в (възстановят от) носител на памет при нужда.

Приноси

Дисертацията съдържа следните научни, научно-приложни и приложни приноси:

Научни приноси:

Н1. Предложен е концептуален модел на парсираща машина, която е подходяща за реализиране на различни стратегии, методи, подходи и алгоритми, предоставяща някои допълнителни възможности спрямо съществуващите;

Н2. Дефинирани са напреднали граматики с фраза символи, като съставени от правила и напреднали символи, с близка структура до граматиките в подсилена Бекус-Наур форма;

Н3. Предложен е модел на фразова машина, която предварително категоризира различните фрази в парсер граматиката с цел ускоряване на анализа.

Научно-приложни приноси:

НП1. Описана е функционалността на парсиращата машина;

НП2. Описана е функционалността на алгоритъм Тунелно парсиране, парсиращ на базата на всяка не ляво рекурсивна напреднала граматика, като времето за парсиране е линейно за някои многозначни граматики и за всяка граматика, която може да се изведе от детерминиран стеков автомат (ако в изведената граматиката няма рекурсия парсирането може да се изпълни с константен обем памет);

НП3. Проектиран и създаден е език за метапрограмиране на граматики;

НП4. Проектиран е профилатор на парсер генератори, позволяващ извършването на тестове (измерване на използваните ресурси по време на разпознаване и парсиране) с парсиращи машини, създадени от различни парсер генератори и компилатори.

Приложни приноси:

П1. Разработен е прототип на софтуерен инструмент – профилатор на парсер генератори (вкл. с модул за визуализиране на резултатите), позволяващ експериментиране с

директно въведена граматика или с програмно изведена от инструмента съвкупност от граматики;

П2. Извършени са експерименти с помощта на профилатора.

Перспективи

Дисертацията засяга много теми, които са свързани с парсиране на данни и поради тази причина има много възможни начини за развитие, като по забележимите са, както следва:

- Списъкът от команди, които се изпращат от парсер модула и се приемат от следващите модули може да се допълни с нови, така че да могат да се строят не само синтактични дървета, но и други синтактични структури;
- Във връзка с напредналите граматики може да се изследва възможността, че след използването на лексемата в даден редица токен (от вида **`t-sequence`**), токенът да се разложи (във входните данни на конкретния модул) на отделни буква токени (от вида **`t-character`**), по един за всяка буква в лексемата и тези токени да се използват за последващо парсиране;
- Естествено продължение на текущата дисертация е създаването на добавка към алгоритъма Тунелно парсиране, която да позволи парсиране на базата на граматики с лява рекурсия;
- На базата на дисертацията може да се създаде алгоритъм, който да проверява дали парсирането на произволни данни с алгоритъма Тунелно парсиране на базата на конкретна граматика ще е винаги за линейно време;
- Възможното бъдещото подобрение на профилатора е в добавянето на поддръжка на други парсер генератори и компилатори. Поддържаните версии на различните парсер генератори и компилатори до момента, не следва да се премахват, за да може тестовете да показват развитието на технологиите, свързани с автоматичната генерация на парсери във времето. Това прави профилатора програма, ориентирана към бъдещето;
- Друго възможно развитие на профилатора е да се измерва времето за компилация на изходните и целевите профили, времето за стартиране на целевите профили и размера на всеки файл.

Списък с публикации по темата на дисертационния труд

1. N. Handzhiyski and E. Somova, "Tunnel Parsing with counted repetitions", Journal Computer Science, vol. 21, n. 4, p. 441-462, 2020;
2. N. Handzhiyski and E. Somova, "A parsing machine architecture encapsulating different parsing approaches", International Journal on Information Technologies and Security (IJITS), vol. 13, n. 3, p. 27-38, 2021;
3. N. Handzhiyski and E. Somova, "The Expressive Power of the Statically Typed Concrete Syntax Trees", CEUR Workshop Proceedings, vol. 3061, p. 136-150, 2021;
4. N. Handzhiyski and E. Somova, "Tunnel Parsing with the Token's Lexeme", Journal Cybernetics and Information Technologies, vol. 22, n. 2, p. 125-144, 2022;
5. N. Handzhiyski and E. Somova, "Tunnel Parsing with Ambiguous Grammars", Journal Cybernetics and Information Technologies, vol. 23, n. 2, p. 34-53, 2023;
6. N. Handzhiyski and E. Somova, "Tunnel Parsing", in book "Composability, Comprehensibility and Correctness of Working Software", CEFP 2019, Lecture Notes in Computer Science, vol. 11950, p. 325-343, 2023.

Забелязани цитирания

1. N. Handzhiyski and E. Somova, "Tunnel Parsing with the Token's Lexeme", Journal Cybernetics and Information Technologies, vol. 22, n. 2, p. 125-144, 2022:
    - S. A. Qassir, M. T. Gaata, A. T. Sadiq, "SCLang: Graphical Domain-Specific Modeling Language for Stream Cipher", Cybernetics and Information Technologies, vol. 23, n. 2, p. 54-71, 2023.

Литература

[1] Х. Кискинов, "Въведение в дискретната математика", Пловдивски университет "Паисий Хилендарски" Факултет по математика и информатика, Пловдивско университетско издателство 2022

[2] A. V. Aho and J. D. Ullman, "The Theory of Parsing, Translation, and Compiling", Prentice-Hall Inc. 1972.

[3] M. Sipser, "Introduction to the Theory of Computation", 2nd edition, Course Technology 2006.

[4] L. Zheng, S. Ma, Z. Chen and X. Luo, "Ensuring the Correctness of Regular Expressions: A Review", International Journal of Automation and Computing, vol. 18, n. 4, p. 521-535, 2021

[5] D. Crocker and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, 2008.

[6] P. Kyzivat, "Case-Sensitive String Support in ABNF", RFC 7405, 2014, visited at https://www.rfc-editor.org/rfc/rfc7405.html.

[7] N. Chomsky, "On Certain Formal Properties of Grammars", Information and Control, vol. 2, n. 2, p. 137-167, 1959.

[8] A. W. Burks and H. Wang, "The Logic of Automata", Journal of the ACM, vol. 4, n. 2/3, Association for Computing Machinery, p. 193-218/279-297, 1957.

[9] Z. Bednárová and V. Geffert and C. Mereghetti and B. Palano, "Removing nondeterminism in constant height pushdown automata", Information and Computation, vol. 237, p. 257-267, 2014.

[10] Y. Bar-Hillel, M. Perles and E. Shamir, "On Formal Properties of Simple Phrase Structure Grammars", Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung, vol. 14, p. 143-172, 1961.

[11] A. Afroozeh and A. Izmaylova, "One Parser to Rule Them All", in book "2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)", p. 151-170, Association for Computing Machinery 2015.

[12] M. G. J. van den Brand and J. Scheerder and J. J. Vinju and E. Visser, "Disambiguation Filters for Scannerless Generalized LR Parsers", in book "Compiler Construction", p. 143-158, Springer Berlin Heidelberg 2002.

[13] E. R. Van Wyk and A. C. Schwerdfeger, "Context-Aware Scanning for Parsing Extensible Languages", in book "Proceedings of the 6th International Conference on Generative Programming and Component Engineering", p. 63-72, Association for Computing Machinery 2007.

[14] E. Scott and A. Johnstone, "GLL syntax analysers for EBNF grammars", Science of Computer Programming, vol. 166, p. 120-145, 2018.

[15] Unicode® 15.0.0, Unicode Standard, 2022, visited at https://www.unicode.org/versions/Unicode15.0.0/.

[16] B. Dean, "We Analyzed 5.2 Million Desktop and Mobile Pages - Here's What We Learned About Page Speed", Backlinko, 2019, visited at https://backlinko.com/page-speed-stats.