### МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

### Національний технічний університет “Харківський політехнічний інститут”

### Кафедра обчислювальної техніки та програмування

### 

Затверджую

Завідуючий кафедрою ОТП

\_\_\_\_\_\_\_\_\_\_(Семенов С.Г) “\_\_\_” \_\_\_\_\_\_\_\_ 2021 р.

ПОБУДОВА СИНТАКСИЧНОГО LR-АНАЛІЗАТОРА

Пояснювальна записка

КІТ.Н121Б.15303 ТЗ

Розробники

Керівник проекту

\_\_\_\_(Гавриленко С.Ю.)

“\_\_” \_\_\_\_\_\_2021 р.

Виконавець

\_\_\_\_\_(Сторожук О.Д.)

“\_\_”\_\_\_\_\_\_\_2021 р.

Харків 2021

ЗМІСТ

ВСТУП

Кінець XX – початок ХХІ ст. характеризується стрімкою комп'ютеризацією, що охопила майже всі сфери людського життя. Дуже важко в даний час знайти галузь, яка не відчула б на собі вплив цього глобального процесу.

Застосування комп'ютерів позбавляє людину виконання трудомістких завдань, дозволяючи сконцентруватися на сутності проблеми. Комп'ютери використовуються при моделюванні процесів, дозволяючи знаходити найоптимальніші рішення. Комп'ютеризація спрощує багато процесів, полегшує взаємодію між людьми, робить наше життя комфортнішим. Комп'ютер вже давно не розкіш, а потреба - це частина нашого повсякденного життя.

Однак значною складовою, що впливає на ефективність спрацювання комп’ютерних є передача команд від користувача. Звичайній людині дуже важко оперувати низькорівневою мовою, яка використовує громіздку послідовність байтів. Звичайна арифметична операція у вигляді складання двох символів буде складатися з не менш як 3 кроків, при якому треба запам’ятовувати машино-специфічні лексеми. Для уникнення цієї перешкоди були розроблені мови високого рівня, які є більш акцентованими на прикладну сферу застосування, аніж на жорстку залежність від архітектури певної комп’ютерної системи.

Для перекладу команд, написаних мовою високого рівня у двійковий формат застосовуються спеціальні транслятори, які в свою чергу вже розподіляються на компілятори та інтерпретатори.

До отримання повноцінного набору машинних інструкцій транслятору необхідно пройти через низку етапів, які перевіряють вхідні інструкції від користувача на їх правильність. Одним із таких етапів слугує синтаксичний аналіз, який перевіряє вхідний набір на відповідність до синтаксису мови – правила побудови допустимих конструкцій мови. Синтаксис мови задається спеціальною граматикою, що зкладається с набору правил, кожним елементом правила є ланцюжок, який містить в собі елементи інших правил, що таким чином дозволяє сформулювати чітку та логічну послідовність вхідних символів.

Мета курсового проектування – отримання навичків зі створення власної контекстно-вільної LR(1) граматики, усвоєння етапів синтаксичного аналізу вхідної послідовності символів.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ ПРОГРАМИ

Розроблювальна програма належить до різновиду синтаксичних аналізаторів, які мають широке розповсюдження у компіляторах та інтерпретаторах багатьох високорівневих мов програмування, які мають дуже широкий діапазон сфер прикладного застосування.

Розроблене програмне забезпечення має зручний інтерфейс користувача, який дозволяє вільно застосувати власну LR(1) граматику та виконувати подальший аналіз вхідної послідовності відносно розроблених правил граматики, спроектований інтерфейс також має додатковий функціонал для покрокового опису роботи розпізнавача із описом вхідних характеристик та усіх необхідних параметрів, на які аналізатор буде опиратися під час аналізу вхідних інструкцій.

2 ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ ПРОГРАМИ

При виконанні курсового проектування буде виконуватися послідовність етапів для виконання дослідження, аналізу, та оцінки роботи.

На початку роботи необхідно створити власну LR(1) граматику, визначити елементи правил, що належать граматики, перевірити усі частини правил на їх досяжність та продуктивність.

Після перевірки правил граматики треба розрахувати множини функцій ВПЕРВ, ВПІСЛЯ та СЛІД, які чітко продемонструють взаємозв'язок символів з різних правил граматики.

Маючи побудовані множини далі будуються таблиці переходів та керування, завдяки яким буде відбуватися синтаксичний аналіз шляхом спрацювання алгоритму роботи магазинного автомата.

Після створення власної граматики із детальним синтаксичним розбором буде виконана побудова програми, що містить в собі всі вказані вище етапи, включаючи синтаксичний аналізатор методом магазинного автомата. Завдяки попереднім етапам аналізу створеної граматики, програма буде порівнюватися із розрахованими даними.

Для підвищення рівня користувальницької взаємодії до програми буде розроблений графічний інтерфейс користувача.

Після виконання всіх етапів проектування необхідно підвести підсумки щодо отриманих результатів зі створення граматики та програми-аналізатору.

3 ОПИС ПОБУДОВИ ВИСХІДНОГО LR(1) РОЗПІЗНАВАЧА

**3.1 Побудова правил граматики**

Згідно з технічного завдання необхідно побудувати операцію присвоєння арифметичного виразу.

На початку необхідно виділити операція присвоєння. Вона містить в собі 3 частини: назва змінної, якій виділяється значення, лексичний символ самої операції “=”, та права частини, тобто арифметична частина. Права частина може мати в собі різні форми, однак перші два елемента беруться в якості незмінних. Тому можна скласти початкове правило граматики:

*I → c=V* (1)

Права частина, тобто арифметичний вираз, може мати самі арифметичні операції, їх операнди та дужки будь-якої вкладеності. При цьому вираз повинен закінчуватися або правою дужкою, або операндом. Також варто дотримуватися правил послідовності операндів, знаків операцій та дужок. Для опису таких правил застосуємо наступні граматики:

*V → PE (2)*

*V → (V)E (3)*

*E → SV (4)*

*E → $ (5)*

Нетермінали P та S – операнди та знаки операцій відповідно, їх правила мають наступний вигляд..

*P → a* (6)

*P → b* (7)

*S → +* (8)

*S → –* (9)

*S → \** (10)

*S → /* (11)

В кінці отримуємо наступну граматику:

*I → c=V* (1)

*V → PE (2)*

*V → (V)E (3)*

*E → SV (4)*

*E → $ (5)*

*P → a* (6)

*P → b* (7)

*S → +* (8)

*S → –* (9)

*S → \** (10)

*S → /* (11)

**3.2 Визначення непродуктивних та недосяжних символів**

Символ x ∈ Va називається непродуктивним, якщо з нього не може бути виведений кінцевий термінальний ланцюжок. Розглядаючи правила граматики, можна зробити висновок, що коли всі символи правої частини є продуктивними, то продуктивним є і символ, що стоїть у лівій частині. Останнє твердження дозволяє організувати процедуру виявлення непродуктивних символів у такому вигляді:

– Скласти список нетермінальних символів, для яких знайдеться хоча б одне правило, права частина якого містить термінальні символи або пусто ($).

– Якщо знайдене таке правило і всі нетермінальні символи, які стоять у його правій частині, вже занесені до списку, то слід додати до списку нетермінальний символ, що стоїть у його лівій частині.

– Якщо на кроці 2 список більше не поповнюється, тоді ми отримали список усіх продуктивних нетермінальних символів граматики, а всі нетермінальні символи, які не потрапили в нього, є непродуктивними.

Визначимо продуктивні символи для граматики. На першому кроці заносимо до списку символ *E,* що містить пусто. Далі до списку заносимо *P* і *S*, оскільки їх праві частини правил містять термінали. Далі додаємо символ *V*, що для своїх правил містить дужки-термінали. Також додаємо нетермінал *I*, який має також термінали в своєму ланцюзі. Отже, отримуємо наступні продуктивні нетермінали: *E*, *P*, *S*, *V*, *I*. Непродуктивних символів немає

Далі необхідно перевірити граматику на недосяжні символи.

Символ х ∈ Vт ∪ Va називається недосяжним у КВ-граматиці Г, якщо х не з'являється в жодному виведеному ланцюжку. Розглядаючи правила граматики, можна помітити, що якщо нетермінальний символ у лівій частині правила є досяжним, то і всі символи правої частини є досяжними. Ця властивість правил є основою процедури виявлення недосяжних символів, яку можна описати таким чином:

– Створити одноелементний список, що складається з початкового символу граматики *І*.

– Якщо знайдене правило, ліва частина якого вже є в списку, то включити до списку всі символи, які містяться в його правій частині.

– Якщо на кроці 2 нові нетермінальні символи в список більше не додаються, то отримано список усіх досяжних нетермінальних символів, а решта символів, що не потрапили в список, є недосяжними.

На першому кроці заноситься символ *I*. Далі заносимо *V*, оскільки він міститься у початковому ланцюзі. Далі заносяться нетермінали: *E*, *P*, *S*. До списку потрапили усі нетермінали, отже граматика не містить недосяжних символів.

**3.3 Визначення граматичних входжень**

Оскільки кожен граматичний символ може входити в правило граматики один чи декілька разів, а також входити в різні правила, то в залежності від розташування граматичних символів у правилах можуть виконуватися різні дії: перенос символу чи згортка, тому вводиться поняття граматичного входження. Граматичне входження символу граматики задається номером правила і номером позиції символу в цьому правилі. Крайній зліва символ – перший. Якщо символ входить у правило один раз, то він позначається номером правила. Якщо символ зустрічається в граматиці тільки один раз, то його можна не нумерувати.

Переписані правила граматики із застосуванням граматичних входжень наведені нижче:

*I → c=V1* (1)

*V → P1E1 (2)*

*V → (V2)E2 (3)*

*E → S1V3 (4)*

*E → $ (5)*

*P → a* (6)

*P → b* (7)

*S → +* (8)

*S → –* (9)

*S → \** (10)

*S → /* (11)

**3.4 Знаходження функції ВПЕРШ, ВПІСЛЯ та СЛІД**

Функція ВПЕРШ(Y) визначає множину символів, які можуть стояти на першому місці в ланцюжках виведених з Y. У нього входять сам Y і всі символи, виведені з Y без правил, що анулюють. Функція ВПЕРШ(Y) визначається і для початкового символу граматики. ВПЕРШ від терміналу є множина, що містить лише цей нетермінал. Розроблена граматика містить наступні функції ВПЕРШ:

1. ВПЕРШ(V1) = {b, a, P1, V1, (}

2. ВПЕРШ(V2) = {b, a, P1, V2, (}

3. ВПЕРШ(V3) = {b, a, P1, V3, (}

4. ВПЕРШ(S1) = {+, -, \*, /, S1}

5. ВПЕРШ(Р1) = {a, b, P1}

6. ВПЕРШ(E1) = {+, -, \*, /, E1, S1}

7. ВПЕРШ(E2) = {+, -, \*, /, E2, S1}

8. ВПЕРШ(I1) = {I1, c}

Функція ВПІСЛЯ(Y) визначає множину символів, що можуть зустрічатися безпосередньо після Y у ланцюжках, виведених з початкового символу граматики. Граматика має наступні функції ВПІСЛЯ:

1. ВПІСЛЯ(=) = {a, b, P1, (, V1}

2. ВПІСЛЯ(V2) = {)}

3. ВПІСЛЯ(S1) = {a, b, P1, (, V3}

4. ВПІСЛЯ(P1) = {E1, S1, +, -, \*, /}

5. ВПІСЛЯ(‘)’) = {E2, S1, +, -, \*, /}

6. ВПІСЛЯ(‘(‘) = {a, b, P1, (, V2}

7. ВПІСЛЯ(h1) = {I1, c}

8. ВПІСЛЯ(c) = {=}

Граматика також містить наступні значення функцій СЛІД:

1. СЛІД(P) = {+, -, \*, /, )}

2. СЛІД(S) = {a, b, (}

3. СЛІД(E) = {)}

4. СЛІД(V) = {)}

**3.5 Побудова таблиці переходів**

Використовуючи функцію ВПІСЛЯ(Y), будується таблиця переходів, яка визначає зміну станів автомата. Таблиця переходів використовується для визначення граматичних входжень, що записуються у магазин. Таблиця будується в такий спосіб:

1. кожному граматичному входженню відповідає рядок таблиці, кожному граматичному символу відповідає стовпець;

2. клітинки таблиці заповнюються елементами функції ВПІСЛЯ(Y);

3. елемент хк, який належить множині функції ВПІСЛЯ(Yj), заноситься в клітку, що знаходиться на перетині рядка Yj та стовпця х.

Подібні дії виконуємо із усіма рядками таблиці переходів (Табл. 3.1).

Таблиця 3.1 – Таблиця переходів

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Граматичні входження | Граматичні символи | | | | | | | | | | | | | | |
| P | S | E | V | I | a | b | c | ( | ) | + | - | \* | / | = |
| a |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| b |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| = | P1 |  |  | V1 |  | a | b |  | ( |  |  |  |  |  |  |
| V1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| V2 |  |  |  |  |  |  |  |  |  | ) |  |  |  |  |  |
| V3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S1 | P1 |  |  | V3 |  | a | b |  | ( |  |  |  |  |  |  |
| P1 |  | S1 | E1 |  |  |  |  |  |  |  | + | - | \* | / |  |
| / |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| - |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| + |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ) |  | S1 | E2 |  |  |  |  |  |  |  | + | - | \* | / |  |
| I1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ( | P1 |  |  | V2 |  | a | b |  | ( |  |  |  |  |  |  |
| h1 |  |  |  |  | I1 |  |  | C |  |  |  |  |  |  |  |
| E1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| c |  |  |  |  |  |  |  |  |  |  |  |  |  |  | = |

**3.6 Побудова керуючої таблиці**

Для опису порядку дій розпізнавача будується керуюча таблиця, у якій:

– буквою П позначається операція переносу;

– буквою З(к) позначається операція згортки, де к – номер правила;

– буквою Д позначається операція “Допустити”, тобто весь ланцюжок розпізнано;

– буквою В (або пустою клітиною) позначається операція “Відкинути”, тобто відбулася помилка і далі розпізнавання неможливо.

Таблиця містить рядки, які є граматичним входженням, і стовпці, які є термінальними символами (символами вхідного ланцюжка). У таблиці також присутній стовпець ⊥ або $, що позначає кінець вхідного рядка.

Заповнюється таблиця таким чином:

– На перетині рядка, позначеного символом I1, та стовпця, позначеного маркером h1, заносимо операцію Д;

– Якщо рядок позначений граматичним входженням Rij, яке не є крайнім правим входженням ніякого правила і якщо елемент таблиці переходів на перетині рядка Rij та стовпця S не є порожнім, то в керуючу таблицю на перетині рядка Rij і стовпця S заноситься операція перенос (П);

– Якщо рядок позначений крайнім правим граматичним входженням рij, і є правило А→α рij з номером κ, то для кожного вхідного символу х, що належить множині СЛІД(А), в керуючу таблицю (на перетині рядка рij і стовпця х) заноситься операція згортки З(к);

– Якщо граматика містить правило що анулює А→ $, з номером к, то необхідно позначити рядки, для яких функція ВПІСЛЯ містить Аij. На перетинанні даних рядків і стовпців відповідних до функції СЛІД(А) ставимо операцію згортки З(к).

Керуюча таблиця, побудована для заданої граматики, наведена в таблиці 3.2.

Табл. 3.2 – Керуюча таблиця для граматики

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Граматичні входження | Термінальні символи | | | | | | | | | | |
| a | b | c | ( | ) | + | – | \* | / | = | ⊥ |
| b |  |  |  |  | З(7) | З(7) | З(7) | З(7) | З(7) |  | З(7) |
| a |  |  |  |  | З(6) | З(6) | З(6) | З(6) | З(6) |  | З(6) |
| = | П | П |  | П |  |  |  |  |  |  |  |
| V1 |  |  |  |  |  |  |  |  |  |  | З(1) |
| V2 |  |  |  |  | П |  |  |  |  |  |  |
| V3 |  |  |  |  | З(4) |  |  |  |  |  | З(4) |
| S1 | П | П |  | П |  |  |  |  |  |  |  |
| P1 |  |  |  |  | З(5) | П | П | П | П |  | З(5) |
| + | З(8) | З(8) |  | З(8) |  |  |  |  |  |  | З(8) |
| – | З(9) | З(9) |  | З(9) |  |  |  |  |  |  | З(9) |
| \* | З(10) | З(10) |  | З(10) |  |  |  |  |  |  | З(10) |
| / | З(11) | З(11) |  | З(11) |  |  |  |  |  |  | З(11) |
| ) |  |  |  |  | З(5) | П | П | П | П |  | З(5) |
| I1 |  |  |  |  |  |  |  |  |  |  | Д |
| ( | П | П |  | П |  |  |  |  |  |  |  |
| h1 |  |  | П |  |  |  |  |  |  |  |  |
| E1 |  |  |  |  | З(2) |  |  |  |  |  | З(2) |
| E2 |  |  |  |  | З(3) |  |  |  |  |  | З(3) |
| c |  |  |  |  |  |  |  |  |  | П |  |

**3.7 Алгоритм роботи висхідного розпізнавача**

Алгоритм роботи використовує таблицю станів і управляючу таблицю і працює таким чином:

1. Прочитати черговий символ вхідного ланцюжка х;

2. Прочитати символ стану, що знаходиться на вершині магазина уij;

3. Прочитати значення елемента управляючої таблиці, що знаходиться в рядку уij і стовпці х;

4. Якщо прочитане значення є В (Відкинути) чи Д (Допустити), то роботу автомата закінчити;

5. Якщо отримане значення визначене операцією П (Перенос), то прочитати в таблиці переходів елемент, що знаходиться в рядку уij і стовпці х. Записати отриманий символ у магазин, перейти до п.1;

6. Якщо отримане значення в керувальній таблиці визначено операцією З(k) (Згортки) у нетермінал Z (згідно з к-тим правилом), то необхідно в таблиці переходів прочитати елемент Zij, що знаходиться в стовпці Z і рядку, що відповідає верхньому символу магазина, який не брав участі в згортці. Записати Zij у магазин і перейти до п.1.

**3.8 Приклад роботи розпізнавача**

Робота розпізнавача перевіримо на прикладі розпізнавання ланцюжка *с=(a+b)/a.* Приклад наведено в табл. 3.3.

Табл. 3.3 – Приклад роботи розпізнавача

|  |  |  |
| --- | --- | --- |
| Вхід автомата | Магазин | Операція |
| c=(a+b)/a⊥ | h1 | П |
| =(a+b)/a⊥ | h1c | П |
| (a+b)/a⊥ | h1c= | П |
| a+b)/a⊥ | h1c=( | П |
| +b)/a⊥ | h1c=(a | З(6) |
| +b)/a⊥ | h1c=(P1 | П |
| b)/a⊥ | h1c=(P1+ | З(8) |
| b)/a⊥ | h1c=(P1S1 | П |
| )/a⊥ | h1c=(P1S1b | З(7) |
| )/a⊥ | h1c=(P1S1P1 | З(5) |
| )/a⊥ | h1c=(P1S1P1E1 | З(2) |
| )/a⊥ | h1c=(P1S1V3 | З(4) |
| )/a⊥ | h1c=(P1E1 | З(2) |
| )/a⊥ | h1c=(V2 | П |
| /a⊥ | h1c=(V2) | П |
| a⊥ | h1c=(V2)/ | З(11) |
| a⊥ | h1c=(V2)S1 | П |
| ⊥ | h1c=(V2)S1a | З(6) |
| ⊥ | h1c=(V2)S1P1 | З(5) |
| ⊥ | h1c=(V2)S1P1E1 | З(2) |
| ⊥ | h1c=(V2)S1V3 | З(4) |
| ⊥ | h1c=(V2)E2 | З(3) |
| ⊥ | h1c=V1 | З(1) |
| ⊥ | h1I1 | Д |

**СПИСОК ЛІТЕРАТУРИ**

1. Гавриленко С.Ю. Основи побудови компіляторів: навч. посіб./ Гавриленко С.Ю., Главчев М.І., Філоненко А.М. – Харків: Курсор, 2005. – 107 с

2. Системне програмування. Системні сервісні компоненти /Дерев’янко С.Ю., Межерицький С.Г., Гавриленко С. Ю., Клименко А.Н./ – Харків: НТУ "ХПІ", 2009. – 160 с.

3. А.В. Гордеев. Системное программное обеспечение: учебник/Гордеев А.В, Молчанов А.Ю. – СПБ., 2002 г.– 734 с.

4. И.А. Волкова Формальные языки и грамматики. Элементы теории трансляции/ Волкова И.А., Руденко Т.В. – М.:МГУ, 1999. – 62 с.

5. Бржезовский А.В. Лексический и синтаксический анализ. Формальные языки и грамматики/ Бржезовский А.В., Корсакова Н.В., Фильчаков В.В. – Л.: ЛИАП, 1990. – 331 с.

6. Альфред Ахо. Компиляторы. Принципы, технологии, инструменты/ Альфред Ахо, Равви Сети, Джеффри Ульман.– М., СПБ., К., 2001. – 768 с.