МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Методичні вказівки

до виконання контрольних робіт для студентів заочної форми навчання за курсом

«ТЕОРІЯ ПОБУДОВИ ТРАНСЛЯТОРІВ»

Другого рівня вищої освіти

за спеціальністю 123 Комп’ютерна інженерія

галузі знань 12 Інформаційні технології

Кваліфікація: Магістр з комп’ютерної інженерії

Затверджено
редакційно-видавничою
радою НТУ “ХПІ”,
протокол № 4 від 03.07.2020 р.

Харків НТУ “ХПІ” 2020

УДК 004.7

Рецензенти:

професор кафедри «Обчислювальна техніка та програмування»
Філоненко А.М.

Методичні вказівки до виконання контрольних робіт для студентів заочної форми навчання за курсом «теорія побудови трансляторів» для студентів другого рівня вищої освіти за спеціальністю 123 Комп'ютерна інженерія/ Уклад: Гавриленко С.Ю. – Харків: НТУ "ХПІ", 2020. – 17 с.

Укладач: С.Ю. Гавриленко, О.В. Октябрьова
Кафедра обчислювальної техніки та програмування

УДК 004.4'22; 004.4'24; 519.685

ЗМІСТ

ВСТУП

Метою даних вказівок є вивчення інформації за курсом «Теорія побудови трансляторів», викладеної у літературних джерелах, закріплення отриманих знань з курсу шляхом виконання контрольних робіт з пророоблення матеріалу лекцій, оформлення звітів з лабораторних робіт та ін.

Значення роботи полягає в розвитку в студента мотивації вивчення конкретної теми за різними літературними джерелами, виконання порівняльного аналізу матеріалу, навички формулювання питань для наступної дискусії в колективі і з викладачем.

1 МЕТА ТА ЗАДАЧІ КУРСУ

Навчальна дисципліна «Теорія побудови трансляторів» відноситься до циклу дисциплін вільного вибору професійної підготовки.

Предметом дисципліни «Теорія побудови трансляторів» є вивчення основ теорії побудови трансляторів та їх застосування при розробці комп'ютерного software.

Дисципліна має метою:

- отримання студентами теоретичних відомостей про сучасні методи побудови трансляторів;
- отримання знань та навичок практичного застосування прийомів побудови трансляторів для мов програмування при створенні системних програмних продуктів.

В результаті виконання контрольної роботи студенти мають отримати знання в межах розглянутих тем та теоретичних питань, а саме:

- типи трансляторів;
- стадії роботи транслятора;
- принципи побудови лексичного сканера;
- основні принципи побудови трансляторів;
- типи формальних мов та граматик;
- правила побудови правил граматики, які описують основні конструкції мов програмування;
- магазинні автомати та їх зв'язок з граматиками;
- *LL(1), LR*(*0*), *LR*(1) граматики;
- граматики простого та операторного передування;
- побудова таблиці ідентифікаторів;
- перетворення дерева розбору в дерево операцій.
- побудова таблиці ідентифікаторів.
- генерація і оптимізація коду.

Виконання контрольних робіт повинно забезпечити формування у студентів відповідних умінь та навичок, а саме:

- розробляти модель лексичного аналізу компілятора;
- будувати правила граматики для опису конструкцій мов програмування;
- спрощувати побудовані конструкції;
- визначати тип граматик;
- будувати магазинний синтаксичний аналізатор мовних конструкцій;
- розробляти програмне забезпечення синтаксичного аналізу компілятора.

2 ОРГАНІЗАЦІЯ ВИКОНАННЯ КОНТРОЛЬНОЇ РОБОТИ

Виконання контрольних робіт з даного курсу полягає у опрацюванні теоретичного матеріалу за літературними джерелами та складається з:

- конспектування та вивчення матеріалу визначених тем та теоретичних питань курсу «Теорія побудови трансляторів»;
- засвоєння змісту основних понять та термінів за відповідною темою;
- виконання контрольних робіт та оформлення звітів з контрольних робіт;
- оформлення рефератів за тематикою, наведеною у п.2.2. для поглибленого вивчення курсу.

Контрольні завдання видаються студентам під час установочних лекцій та мають бути надісланими до заочного деканату в термін згідно навчального графіку.

2.1 Типові недоліки індивідуальної роботи студента

До типових недоліків контрольної індивідуальної роботи студента можливо віднести наступне:

- порушення графіка виконання завдання,
- аналіз конкретної теми тільки за одним літературним джерелом, що виключає можливість проведення критичного порівняльного аналізу досліджуваного матеріалу,
- низька якість розробленого програмного забезпечення;

2.2 Тематика для самостійної роботи студента

1. Транслятори, компілятори, інтерпретатори – загальна схема роботи. Класичні та кросплатформені компілятори. Поняття проходу. Особливості побудови інтерпретаторів. Транслятори з мови Асемблер.

2. Використання таблиці символів. Інтерфейс таблиці символів. Обробка зарезервованих ключових слів. Реалізація таблиці символів

3. Лексичні аналізатори (сканери). Принципи побудови лексичних аналізаторів

4. Синтаксичні аналізатори. Синтаксично керований переклад. Дерево розбору та дерево операцій..

5. Рекурсивний обхід дерева. Перетворення дерева розбору в дерево операцій

6. Семантичний аналіз. Побудова моделі семантичного аналізатора для мови програмування

7. Розподіл пам’яті. Дисплей пам’яті процедури. Стекова організація дисплея пам’яті.

8. Проміжне подання коду програми. Зворотний польський запис.

9. Тріади. Тетради. Перетворення типових вузлів дерева операцій в список тріад.

10. Перетворення типових вузлів дерева операцій в асемблерний код.

11. Оптимізація об’єктного коду програми методом згортки.

12. Оптимізація об’єктного коду програми методом вилучення зайвих операцій.

13. Оптимізація об’єктного коду програми методом розмотки циклів, видаленням мертвого коду,

14. Пониженням сили операції, заміни одних команд іншими

15. Збірка транслятора. Редактори зв’язків та завантажувачі. Предпроцессори. Асемблери. Загружчики та редактори зв’язку.

16. Загальні відомості про віртуалізацію. Віртуалізація платформ та ресурсів. Віртуальні машини.

3 ПОРЯДОК ВИКОНАННЯ КОНТРОЛЬНОЇ РОБОТИ

Залежно від номера прізвища за списком вибрати тему контрольної роботи (п.3.1) та узгодити з викладачем фрагмент коду програми. Контрольна робота повинна містити наступні пункти:

1. Побудова лексичного аналізатора.
2. Побудова правил граматики.
3. Визначення типу граматики.
4. Розробка команд роботи синтаксичного аналізатора.
5. Перевірка роботи синтаксичного аналізатора на фрагменті коду.

3.1 Індивідуальне завдання для контрольних робіт

1. Опис змінних цілого та дійсного типів.
2. Опис констант символьного типу.
3. Опис масивів та їх ініціалізація.

4. Оператор циклу з параметром.
5. Оператор циклу з передумовою.
6. Оператор циклу з післяумовую.
7. Умовний оператор.
8. Оператор вибору.
9. Оператор присвоювання арифметичного виразу, до складу якого входять: ідентифікатори, операції “+”, “-“ ,”*”,”\”, дужки.
10. Оператор присвоювання арифметичного виразу, до складу якого входять: ідентифікатори, операція “+”, дужки, математичні функції.
11. Оператор присвоювання виразу, до складу якого входять інкрементні та декрементні операції.
12. Оператор присвоювання логічного виразу, до складу якого входять: ідентифікатори, операції “&&”, “ || ” , “>”, “<”, “< >”, дужки.
13. Опис масивів вказівників.
14. Опис змінних та масивів дійсного типу.
15. Опис рядкових змінних. Оператори введення строк.
16. Опис рядкових змінних. Бібліотечні функції роботи зі строками.
17. Опис функцій.
18. Опис констант.
19. Опис структур (записів).
20. Опис типізованих файлів.

4 ПРИКЛАД ВИКОНАННЯ КОНТРОЛЬНОЇ РОБОТИ

4.1 Побудова лексичного аналізатора

Розглянемо приклад виділення лексем у виразі описання масиву чисел, який містить змінну (у вигляді послідовності літер англійського алфавіту та цифр за умови, що першим символом може бути літера, знак присвоювання «=», дужки «[,]», десяткові цифри та кому «,». Допускається також, що вираз може містити пробіл. Прикладом виразу може бути такий рядок: « *ident =[1, 2, 34,]* ».

Для спрощення ситуації будемо вважати, що допустимі літери є тільки рядковими. При виділенні лексем необхідно врахувати такі ситуації. Межею лексеми можуть бути знак присвоювання «=», дужки «[,]», кома «,» та пробіл, при цьому всі перераховані символи, окрім пробілу, також є лексемами.

Визначимо X – множину вхідних станів сканера. Для визначення роботи автомата немає значення яка англійська літера або яка цифра надійшла, тому їх можливо згрупувати в два різних вхідних стани. Появу всіх інших символів, окрім символів присвоювання «=», дужки «[,]» та коми «,», пробілу будемо вважати забороненими.

Закодуємо вхідні стани: x_1 – поява пробілу, x_2 – поява будь-якої англійської літери, x_3 – поява цифри, x_4 – знаку присвоювання «=», x_5 – поява дужки «[», x_6 – поява коми «,», x_7 – поява дужки «]», x_8– поява забороненого символу. Таким чином, множина вхідних станів $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$. Визначимо вихідні стани $Y = \{y_0, y_1, y_2, y_4,\}$, де y_0 – лексема не виділена, y_1 – виділена одна лексема, y_2 – виділено дві лексеми (при появі на вході знаків : «=,], [, , » які одночасно є межею між лексемами і лексемами), y_3 – помилка зчитування; y_4 – виділена одна лексема і помилка. Задаймо внутрішні стани: s_0 – очікування наступної лексеми, s_1 – зчитування ідентифікатора, s_2 – зчитування числа.

Розглянемо роботу автомата (табл. 1). Якщо автомат знаходився в стані $s_{0(t-1)}$ і на його вхід було подано символ (стан x_2), то він виділить його як складову лексеми ідентифікатора, перейде в стан s_1t і видасть на виході стан y_{0t} . Якщо автомат знаходився в стані $s_{0(t-1)}$ і на його вхід була подана цифра (стан x_3),то він перейде в стан s_{2t}. Поява на вході іншої цифри залишить його в стані s_{2t}, оскільки проходить процес визначення числа. Якщо автомат знаходився в стані $s_{2(t-1)}$ і на його вхід був поданий пробіл (стан x_1), то він перейде в початковой стан s_0t, а на виході буде сигнал y_{1t} , тобто буде виділена лексема: число. Якщо автомат знаходився в стані $s_{2(t-1)}$ і на його вхід було подано кому (стан x_6), то він перейде в початковий стан s_0t, а на виході буде сигнал y_{2t} , тобто буде виділено дві лексеми: число та кома.

Таблиця 1 – Стани КА

Внутрішні стани	Вхідні стани							
	x_{1t}	x_{2t}	x_{3t}	x_{4t}	x_{5t}	x_{6t}	x_{7t}	x_{8t}
$S_{0\ t-1}$	S_{0t}/y_{0t}	S_{1t}/y_{0t}	S_{2t}/y_{0t}	S_{0t}/y_{1t}	S_{0t}/y_{1t}	S_{0t}/y_{1t}	S_{0t}/y_{1t}	S_{0t}/y_{3t}
$S_{1\ t-1}$	S_{0t}/y_{1t}	S_{1t}/y_{0t}	S_{1t}/y_{0t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{4t}
$S_{2\ t-1}$	S_{0t}/y_{1t}	S_{0t}/y_{1t}	S_{2t}/y_{0t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{2t}	S_{0t}/y_{4t}

4.2 Побудова синтаксичного аналізатора

4.2.1 Побудова правил граматики. Перевірка граматики на наявність непродуктивних та недосяжних символів.

Задача. Побудувати правила граматики для нижченаведених фрагментів коду програми.

- *{a++; b--; --a; a++;}*
- *{ b--; a++; }*
- *{a--; b--; a++; ++a;}*

Перевірити граматику на наявність непродуктивних та недосяжних символів**.**

У загальному випадку, якщо описана множина ланцюжків, що є фрагментами деякої мови, і потрібно побудувати граматику, яка породжує цю множину ланцюжків, то слід робити таким чином:

- виписати кілька прикладів із заданої множини ланцюжків;
- проаналізувати структуру ланцюжків, виділяючи початок, кінець, що повторюються, або символи з групи символів;
- ввести позначення для складних структур, що складаються з груп символів; такі позначення є нетермінальними символами граматики, яка будується;
- побудувати правила для кожної з виділених структур, використовуючи для задання повторюваних структур рекурсивні правила;
- об’єднати всі правила;
- перевірити отримані правила за допомогою процедури виведення на прикладі отримання ланцюжків з різною структурою.

Проаналізувавши приклади (фрагменти) кодів програми, можливо помітити наступне. Приклади містять список змінних *a* та *b* з операцією інкремент «++» та декремент «--», причому операція може бути розташована до або після змінних (прекремент або посткремент). У якості роздільника елементу списку виступає символ «;».

Об’єднаємо змінні *a* та *b* в нетермінальний символом *A*, а операції інкремент «++» та декремент «--» – в нетермінальний символом *B*. Введемо

нетермінальний символ *S* – елемент списку з роздільником. Символ *R* будемо використовувати для позначення рекурсії. Приклади формавання нетермінальних символів наведено на рис. 1, 2.

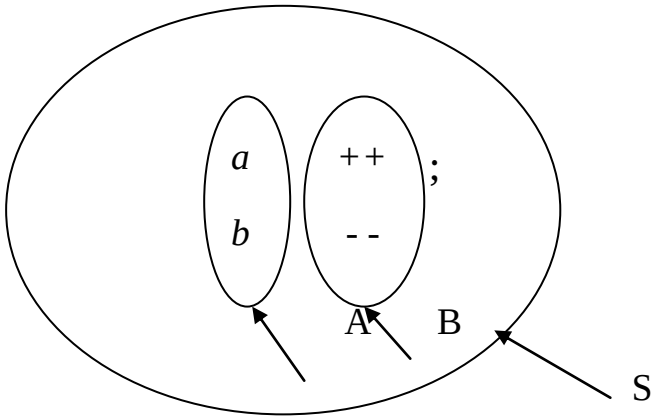

Рисунок 1 – Формування одного із варіантів нетермінального символу *S*

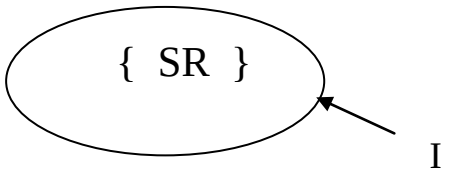

Рисунок 2 – Формування нетермінального символу *I*

Отримаємо такі правила граматики:

Г: R = {1. $I \rightarrow \{SR\}$

2. $R \rightarrow SR|\$$

3. $S \rightarrow AB;|BA;$

4. $A \rightarrow a|b$

5. $B \rightarrow ++|-- \}$.

Перевіримо правила граматики для фрагменту коду програми *{a++; --b:}* за допомогою виведення:

$I \rightarrow \{SR\} \Rightarrow \{AB;R\} \Rightarrow \{aB;R\} \Rightarrow \{a++;R\} \Rightarrow \{a++;SR\} \Rightarrow \{a++;$ $BA;R\} \Rightarrow \{a++; --A;R\} \Rightarrow \{a++; --b;R\} \Rightarrow \{a++; --b; \}$.

Синтаксичний розбір для даного прикладу має такий вигляд:

[1, 3.1, 4.1, 5.1, 2.2, 3.2, 5.2, 4.2, 2.2].

Перевірка граматики на наявність непродуктивних символів виконується наступним чином:

Символ $x \in V_a$ 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, який стоїть у лівій частині. Останнє твердження дозволяє організувати процедуру виявлення непродуктивних символів у такому вигляді:

1. Скласти список нетермінальних символів, для яких знайдеться хоча б одне правило, права частина якого містить термінальні символи або пусто (*$*).
2. 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.
3. 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.

Перевіримо вищенаведену граматику на наявність непродуктивних символів:

1) S,A,B

2) S,A,B,R

3) S,A,B,R,I

Непродуктивних символів немає.

Перевірка граматики на наявність недосяжних символів виконується наступним чином:

Символ $x \in V_т \cup V_a$ називається **недосяжним** у КВ-граматиці Г, якщо x 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:

1. Створити одноелементний список, що складається з початкового символу граматики *I*.
2. Якщо знайдене правило, ліва частина якого вже є в списку, то

включити до списку всі символи, які містяться в його правій частині.

3. 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.

Перевіримо вищенаведену граматику на наявність недосяжних символів.

Пошук недосяжних символів:

1) I

2) I,S,R

3) I,S,R,A,B

Недосяжних символів немає.

4.2.2 Визначення типу граматики

Для вищенаведеної граматики маємо такі правила:

1. $I \rightarrow \{SR\}$
2. $R \rightarrow SR|\$$
3. $S \rightarrow AB;|BA;$
4. $A \rightarrow a|b$
5. $B \rightarrow ++|--$

Дана граматика містить анулюючі правила, права частина правил починається з термінальних та нетермінальних символів, отже граматика не може бути розділеною та слаборозділеною граматикою. Перевіримо належність граматики до класу *LL(1)* граматики. Знайдемо функції ПЕРШ(), СЛІД() і множину ВИБІР.

Для спрощення запису, у якості аргументу будемо вказувати номер правила.

Знайдемо функцію **ПЕРШ( )**:

ПЕРШ(1)={{}

ПЕРШ(2.1)= ПЕРШ(*S*)= ПЕРШ(*A*) U ПЕРШ(*B)={+,-,a,b}*

ПЕРШ(2.2)={$}

ПЕРШ(3.1)= ПЕРШ(*A)={a,b}*

ПЕРШ(3.2)= ПЕРШ*(B)={+,-}*

ПЕРШ(4.1)={*a*}

ПЕРШ(4.2)={*b*}

ПЕРШ(5.1)={+}

ПЕРШ(5.2)={-}

Знайдемо функцію **СЛІД ( )** :
СЛІД(*S*)={+,-,*a*,*b*,*}*}
СЛІД(*R*)={*}*}
СЛІД(*A*)={+,-,;}
СЛІД(*B*)={*;, a, b*}

Знайдемо множину **ВИБІР( )**:
ВИБІР(1)={{}
ВИБІР (2.1)= Перш(*S*)={+,-,*a*,*b*}
ВИБІР (2.2)= СЛІД(*R*)={*}*}
ВИБІР (3.1)= Перш(*A*)={*a*,*b*}
ВИБІР (3.2)= Перш(*B*)={+,- }
ВИБІР (4.1)={*a*}
ВИБІР (4.2)={*b*}
ВИБІР (5.1)={+}
ВИБІР (5.2)={-}

Елементи множини Вибір для правил з однаковим лівим нетермінальним символом не перетинаються (правила 2.1 та 2.2, 3.1 та 3.2, 4.1 та 4.2, 5.1 та 5.2). Отже дана граматика є *LL*(1) граматикою.

4.2.3. Побудова команди синтаксичного аналізатора

Розглянемо правило – $I \rightarrow \{SR\}$. Правило починається з термінального символу «{» . Множина ВИБІР(I$\rightarrow$ {*SR*})={{}. Для даного правила будуємо команду:

f (s, { , I) = (s, }RS)

Аналогічно будуємо команди для правил A$\rightarrow${a|b}, B$\rightarrow${++|--}, адже вони теж починаються з термінальних символів:

f(s, a , A) = (s, $)
f(s, b , A) = (S, $)
f(s, + , B) = (s, +)
f(s, - , B) = (s, -)

Розглянемо правило: $R \rightarrow SR$. Правило починається з нетермінального символу. Для даного правила будуємо команду без зрушення вхідної голівки. Таких команд для правила буде чотири, тому що множина ВИБІР ($R \rightarrow SR$)= {+, - , a, b} містить чотири елементи.

f(s, + , R) = (s, RS)*

*f *(s, - , R) = (s, RS)*
f(s, a , R) = (s, RS)*
f(s, b , R) = (s, RS)*

Аналогічно будуємо команди для правила $S \rightarrow \{AB;|BA;\}$. Воно також починається з нетермніальних символів.

f(s, b , S) = (s, ;BA)*
f(s, a , S) = (s, ;BA)*
*f *(s, -, S) = (s, ;AB)*
*f *(s, + , S) = (s, ;AB)*

Друга частина правила $R \rightarrow SR|\$$ є анулюючою, отже будуємо для неї команду наступного типу.

f(s, } , R) = (s, $)*

Наступні команди будуємо для всіх термінальних символів, розташованих на кінці або в середини правил граматики:

f(s, } , }) = (s, $)
f(s, ; , ;) = (s, $)
f(s, + , +) = (s, $)
f(s, - , -) = (s, $)

Для переходу в заключний стан будуємо наступну команду:

f(s,$, h_0) = (s, $).*

Отримуємо наступний список команд (функцій переходів):

1. *f (s, { , I) = (s, }RS)*
2. *f(s, } , }) = (s, $)*
3. *f*(s, + , R) = (s, RS)*
4. *f *(s, - , R) = (s, RS)*
5. *f*(s, a , R) = (s, RS)*
6. *f*(s, b , R) = (s, RS)*
7. *f*(s, } , R) = (s, $)*
8. *f*(s, b , S) = (s, ;BA)*
9. *f*(s, a , S) = (s, ;BA)*
10. *f(s, ; , ;) = (s, $)*
11. *f *(s, -, S) = (s, ;AB)*
12. *f *(s, + , S) = (s, ;AB)*
13. *f(s, a , A) = (s, $)*
14. *f(s, b , A) = (S, $)*
15. *f(s, + , B) = (s, +)*
16. *f(s, + , +) = (s, $)*
17. *f(s, - , B) = (s, -)*

18. $f(s, -, -) = (s, \$)$
19. $f^*(s, \$, h_0) = (s, \$)$.

4.2.4 Перевірка роботи команд синтаксичного аналізатора

Виконаємо перевірку роботи команд синтаксичного аналізатора на прикладі ланцюжка *{a++;--b;}*. Перевірка виконується таким чином. Беремо самий крайній символ вхідної стрічки – «*{*». Беремо символ з вершини магазину – «*I*». Відповідно до першої команди синтаксичного аналізатора – *f (s, { , I) = (s, }RS)*, яка виконується зі зрушенням вхідної голівки, видаляємо символ «{» з вхідної стрічки, видаляємо символ «*I*» з вершини магазину і замість нього записуємо – «*}RS*».

Порядок використаних команд для розпізнавання ланцюжка *{a++;--b;}* такий: 1, 9, 13, 15, 16, 10, 4, 12, 17, 18, 14, 10, 7, 2, 19.

Отримаємо наступну зміну конфігурацій:

(s, {a++;--b;}, h_0I) ├ 1
(s, a++;--b;}, h_0}RS) ├ 9
(s, a++;--b;}, h_0}R;BA) ├ 13
(s, ++;--b;}, h_0}R;B) ├ 15
(s, +;--b;}, h0}R;+) ├ 16
(s, ;--b;}, h_0}R;) ├ 10
(s, --b;}, h_0}R) ├ 4
(s, --b;}, h_0}RS) ├ 11
(s, --b;}, h_0}R;AB) ├ 17
(s, -b;}, h_0}R;A-) ├ 18
(s, b;}, h0}R;A) ├ 14
(s, ;}, h_0}R;) ├ 10
(s, }, h_0}R) ├ 7
(s, }, h_0}) ├ 2
(s, $, h_0) ├ 19
(s, $, $)

Має місце послідовність змін конфігурацій від початкової до заключної конфігурації. Отже ланцюжок розпізнано.

5 МЕТОДИЧНЕ ЗАБЕЗПЕЧЕННЯ ДИСЦИПЛІНИ

1 Дерев’яноко О.С., Гавриленко С.Ю., Межерицький С.Г., Клименко А.М. Системне програмування. Системні сервісні компоненти. – Харків, НТУ «ХПІ», 2009. – 160 с.

2 Гавриленко С.Ю., Філоненко А.М., Главчев М.І. Основи побудови трансляторів. – Харків: Курсор, 2005.–107

3 Гавриленко С.Ю. Методичні вказівки до виконання курсового проекту з курсу «Теорія побудови трансляторів». Харків, НТУ «ХПІ», 2009. – 16 с.

4 Гордеев А.В. Молчанов А.Ю. Системное программное обеспечение. Учебник – Санк-Петербург, 2002. –734 с.

5 Альфред Ахо, Равви Сети, Джеффри Ульман. «Компиляторы. Принципы, технологии, инструменты». - Москва •Санк-Петербург•Киев,2001.– 768 с.

6 Гавриленко С.Ю., Клименко А.М., Любченко Н.Ю., Смоляр В.Г., Тишко С.О. Теорія цифрових автоматів та формальних мов. Навчальний посібник. **Рекомендовано МОН** України – Харків: НТУ "ХПІ", 2011. – 176 с.

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

8 А. Ахо, Дж.Ульман «Теория синтаксического анализа перевода и компиляции», т.1, – М.: Мир,1978.– 612 с

9 Хантер Р. Проектирование и конструирование компиляторов». – М.: Финансы и статистика, 1984.–345 с.

10 Бек Л. Введение в системное программирование. М: Мир,1988.–263.

11 Гавриленко С.Ю. Методичні вказівки до виконання лабораторних робіт з курсу «Теорія побудови трансляторів»., Харків: НТУ "ХПІ", –2013. –76 с.

Навчальне видання

ГАВРИЛЕНКО Світлана Юріївна,

ОКТЯБРЬОВА Олена Володимирівна

Методичні вказівки

до виконання контрольних робіт за курсом “ТЕОРІЯ ПОБУДОВИ ТРАНСЛЯТОРІВ”для студентів заочної форми навчання спеціальності “Комп’ютерна інженерія

Роботу до видання рекомендувала Філоненко А.М

В авторській редакції

Самостійне електронне видання