ПРОГРЕСИВНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

ПРОГРЕССИВНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

PROGRESSIVE INFORMATION TECHNOLOGIES

УДК 004.9

Литвин В. В.[1], Бобик І. О.[2], Висоцька В. А.[3]

[1]Д-р техн. наук, професор, завідувач кафедри «Інформаційні системи та мережі» Національного університету «Львівська політехніка», Львів, Україна
[2]Канд. физ.-мат. наук, доцент кафедри «Вищої математики» Національного університету «Львівська політехніка», Львів, Україна
[3]Канд. техн. наук, доцент кафедри «Інформаційні системи та мережі» Національного університету «Львівська політехніка», Львів, Україна

ЗАСТОСУВАННЯ СИСТЕМИ АЛГОРИТМІЧНИХ АЛГЕБР ДЛЯ ГРАМАТИЧНОГО АНАЛІЗУ СИМВОЛЬНИХ ОБЧИСЛЕНЬ ВИРАЗІВ ЛОГІКИ ВИСЛОВЛЮВАНЬ

Розроблено архітектуру та реалізовано програмну систему граматичного аналізу схем системи алгебраїчних алгебр та їх інтерпретації. Програмна система дає змогу автоматизовано генерувати програми за такими створеними схемами та їх відлагоджувати у відповідних схемах. Визначено чіткий розподіл системи алгебраїчних алгебр на окремі модулі, кожен з яких характеризуватиметься своїм функціональним навантаженням. Використано методи синтаксичного аналізу для розроблення та подання граматики таких схем. Реалізовано автоматичне їх перетворення в списковy форму. Розроблено машини системи алгебраїчних алгебр як абстрактний механізм інтерпретації граматики засобами синтаксичного аналізу. Словник V складається з скінченої не порожньої множини лексичних одиниць. Вираз над V є ланцюжком скінченої довжини лексичних одиниць із V. Порожній ланцюжок, який не містить лексичних одиниць, позначимо через Λ. Множина всіх лексичних одиниць над V позначимо V'. Мова над V є підмножиною V'. Мову задають через множину всіх лексичних одиниць мови або через означення критерію, якому повинні задовольняти лексичні одиниці, щоб належати мові. Ще є один важливий спосіб задати мову – через використання породжувальної граматики. Граматика складається з множини лексичних одиниць різного типу та множини правил або продукцій побудови виразу. Граматика має словник V, який є множиною лексичних одиниць для побудови виразів мови. Деякі лексичні одиниці словника (термінальні) не можуть замінятися іншими лексичними одиницями. Текст реалізує структурно подану діяльність, що передбачає суб'єкт і об'єкт, процес, мету, засоби і результат, які відображаються в змістовно-структурних, функціональних, комунікативних показниках. Одиницями внутрішньої організації структури тексту є алфавіт, лексика (парадигматика), граматика (синтагматика), парадигми, парадигматичні відношення, синтагматичні відношення, правила ідентифікації, висловлювання, між фразова єдність та фрагменти-блоки. На композиційному рівні виділяють речення, абзаци, параграфи, розділи, глави, підглави, сторінки тощо, які, крім речення, побічно пов'язані з внутрішньою структурою, тому не розглядаються. За допомогою бази даних (бази термінів/морфем і службових частин мови) та визначених правил аналізу тексту виконують пошук терміну. Синтаксичні аналізатори працюють в два етапи: ідентифікують змістовні лексеми та створюють дерево розбору.

Ключові слова: текст, україномовний, алгоритм, контент-моніторінг, ключові слова, лінгвістичний аналіз, синтаксичний аналіз, породжувальні граматики, структурна схема речення, інформаційна лінгвістична система.

НОМЕНКЛАТУРА

САА – системи алгоритмічних алгебр;

ІС – інформаційна система;

ООП – об'єктно-орієнтоване програмування;

ПО – предметна область;

LL(k) граматика яке підмножина контекстовільних граматик для LL-аналізатора як низкосхідного алгоритму синтаксичного разбору; цифра k – кількість лексем для розбору;

A – команда типу термінал або семантичні процедури перетворення САА-схеми у формулу цієї схеми;

V – умова виконання команди;

B – команда типу нетермінал;

K – команда, яка формує логічний результат перевірки у вигляді глобальної логічної змінної САА-машини;

L – команда пересуває вказівник схеми на кількість символів в тексті команди у випадку збігу його з вхідним потоком;

DOI 10.15588/1607-3274-2016-4-10

I – забезпечує візуалізацію тексту повідомлення на екрані у вікні повідомлень САА-машини;

W – команда обчислення складеної логічної умови в LL(k)-граматиці або САА-схемах;

E – логічна константа, яка тотожно істинній логічній умові;

X – забезпечує послідовне виконання команд автомата;

ДНФ – диз'юнктивна нормальна форма;

БД – база даних;

DFD – діаграма потоків даних (англ. Data Flow Diagram);

A_1 – установку індексу масиву $i = 1$;

A_2 – перестановка місцями елементів масиву з індексами i, $i+1$;

A_3 – збільшення індексу масиву на i на одиницю;

α_1 – умова виконання операторів, де $\alpha_1 = 1$, якщо індекс масиву дорівнює $n-1$, де n – розмірність масиву, $\alpha_1 = 0$ в іншому випадку;

α_2 – умова виконання операторів, де $\alpha_2 = 0$, якщо елемент масиву з індексом i більше елемента масиву з індексом $i+1$, $\alpha_2 = 1$ в протилежному випадку.

ВСТУП

Алгебра алгоритмів досліджує властивості алгоритмів [1–2]. Теорія САА вивчає і будує алгебри алгоритмів або алгоритмічні алгебри [3–4]. Основними поняттями алгебри алгоритмів є операції над множинами, булеві операції, предикати, функції й оператори; бінарні і n-арні відношення, еквівалентність, частково і цілком упорядковані множини; графи-схеми й операції над графовими структурами; операції сигнатури САА, аксіоми і правила визначення властивостей програм на основі стратегії згортання, розгортання і їх комбінацій; методи синтаксичного аналізу структурних програм і символьне опрацювання [5–9]. Операції алгебри задовольняють аксіоматичні закони асоціативності, ідемпотентності, комутативності, виключення третього і суперечності [1–4]. Застосування САА найчастіше провадиться в теорії секвенційних алгоритмів і проектуванні ІС; редактори формул алгоритмів і аналіз синтаксису та семантики алгебри алгоритмів-секвенцій; засобах еквівалентних перетворень алгоритмів; методах підвищення ефективності математичного моделювання алгоритмів ІС; принципах побудови електронної бібліотеки абстрактних алгоритмів [10–14]. Практичним результатом досліджень САА є побудова оригінальних інструментальних систем проектування програм на основі сучасних засобів підтримки ООП (Rational Rose), в тому числі і для граматичного аналізу символьних обчислень виразів логіки висловлювань на основі синтаксичного аналізу текстових масивів даних [10–14].

1 ПОСТАНОВКА ЗАДАЧІ

Функціональність системи синтаксичного аналізу складається з таких складових:

– механізм синтаксичного аналізу та відлагодження опрацьованої САА-схеми у вигляді абстрактної САА-машини;

– опис граматики САА-мови та формул САА-схеми, перетворення граматики у списокову форму подання;

– генерація коду за САА-схемою, результатом чого є формульне подання САА-схеми;

– механізм інтерпретації формул САА-схеми, виклик та реалізація семантичних процедур.

Для цих задач створена абстрактна САА-машина з певним набором команд для задач синтаксичного аналізу. Через аналогію в поданні LL(k)-граматик та формул САА-мови такий механізм можна використати в якості не лише генератора коду, але й для тестового відлагодження САА-програм. На рис. 1 подана схема функціонування САА-машини в процесі синтаксичного аналізу

Рисунок 1 – Схема функціонування САА-машини

САА-схеми та побудови її формули. По вхідній стрічці (на рис. 1 схема сортування бульбашка) пересувається вказівник вхідного потоку, його зсув задається командою L-граматики. LL(k)-граматика для опису синтаксису САА-схем є керуючою програмою для САА-машини. Попередньо, до початку синтаксичного аналізу та трансляції, САА-машина перетворює описану граматику у спискову форму. По LL(k)-граматиці рухається вказівник граматики, його пересування визначається командами послідовного виконання команд САА-машини, або залишається на місці при ідентифікації *B*. Механізм виконання команд типу *B* є рекурсивним, що реалізують через використання стеку команд САА-машини. На кожному кроці САА-машина обирає з граматики чергову її команду і виконує її. У випадку команд типу *A* САА-машина здійснює виклик окремого компонента, що задає реалізацію цих семантичних процедур у вигляді процедур та функцій мови програмування, на якій реалізована відповідна САА-машина. Семантичні процедури генерують формулу САА-схеми у відповідності з деревом виводу схеми і записують цю формулу у вихідну стрічку.

Описаний процес синтезу прорами в деякій мові програмування за САА-схемами підтримується автоматично при наявності:

– абстрактної САА-машини із механізмом синтаксичного аналізу методом рекурсивного спуску;

– перетворювача описаної граматики згідно САА-схем у спискову форму;

– інтерпретатор граматики мови згідно САА схем;

– інтерпретатор граматики формул згідно САА-схем;

– генератор формул згідно САА-схем, який реалізовує набір семантичних процедур, що задають процес побудови формули схем.

В основі проектування ІС лежить моделювання ПО. Для отримання адекватного проекту ІС у вигляді системи правильно працюючих програм, необхідно мати цілісне, системне уявлення моделі, яке відображає всі аспекти її функціонування. При цьому під моделлю ПО є система, яка імітує структуру або функціонування досліджуваної ПО і відповідає основній вимозі – бути адекватною цій області. Інформаційна модель системи символьних обчислень в логіці висловлювань повинна вміти:

– розпізнавати вирази логіки висловлювань;

– здійснювати еквівалентні перетворювання над ними, генеруючи покроковий звіт щодо процесу перетворення;

– у випадку некоректно введених даних генерувати звіти про помилки з поясненнями;

– зберігати статистику виконань програми;

– відображати результати роботи користувачу у зрозумілому для нього вигляді.

2 ОГЛЯД ЛІТЕРАТУРИ

Синтаксис структурованих та змістовних текстових масивів даних реалізують через сукупність правил для побудови формул та розпізнавання правильних формул серед послідовностей символів [1–4]. Для системи символьних обчислень усі операції логіки висловлювань, окрім заперечення, є бінарними. На цьому і базується синтаксичний аналізатор. Синтаксичний аналіз реалізують через процес аналізу вхідної послідовності символів з метою розбору граматичної структури відповідно до заданої формальної граматики. Синтаксичний аналізатор (або парсер, англ. parser) є програмою або частиною програми для виконання синтаксичного аналізу текстових масивів даних [10–14].

Під поняттям синтаксичного розбору розуміють розбиття тексту на складові частини мови з фіксацією їхніх форм, призначення і синтаксичного зв'язку з іншими частинами. Це визначається на етапі аналізу відмінків і позиціювання частин відповідної мови, які є складними для формалізації у флективних мовах, наприклад таких, як українська.

Термінальний ланцюжок як речення таких мов розібрати програмно нелегко. Наприклад, у структурі людської мови є суттєві неоднозначності (особливо в розмовній), тобто слова і вирази, які передають зміст у великій кількості варіантів, але тільки одне зі значень доречне в конкретному випадку. Успіх вибору правильного варіанту переважно залежить від багатьох факторів контекстного змісту, і передбачити їх всі неможливо. Важко підготувати формальні правила для опису неформальної поведінки. Хоча існують і строгі правила, множина яких утворює базу граматики як основи синтаксичного аналізатора. Під час синтаксичного аналізу текст перетворюють у структуру даних, найкраще для подальшого опрацювання – в дерево згідно синтаксичної структури вхідної послідовності. Зазвичай синтаксичні аналізатори працюють в три етапи: ідентифікують змістовні лексеми або токени (лексичний аналіз або токенізація), створюють дерево розбору та аналізують дерево (парсер).

Токеном є послідовність одного чи більше символів, які виділяють як атомарний об'єкт. Виділяють токени на основі базових правил лексичного аналізатора (лексера), які відрізняються в залежності від області застосування [10–14]. Токени часто класифікують за розташуванням символів у послідовності знаків чи контексту в потоці даних. Це не виділення групи символів, які обмеженні розділовими знаками з обох сторін (пробілами чи знаками пунктуації). Токени визначають правилами лексера і включають граматичні елементи мови (категорії іменників, дієслів, прикметників або знаків пунктуації), що використовують в потоці даних, в подальшому опрацюванні токенів синтаксичним аналізатором або іншими функціями в програмі. До завдань лексичного аналізу належать:

1. Перетворення набору символів тексту у послідовність токенів.

2. Виділення кожного токену як логічної частини тексту (ключове слово, ім'я змінної, знак пунктуації тощо).

3. Встановлення відповідності між токеном і лексемою – конкретний текст токена («for» «variable», «;» тощо).

4. Виділення додаткових атрибутів токена (наприклад, значення змінної).

5. Формування послідовності токенів на виході для парсера в якості вхідних даних.

Лексичний аналізатор зазвичай нічого не робить з комбінацією ідентифікованих токенів. Для прикладу, типовий лексичний аналізатор розпізнає дужки як знаки, але не перевіряє, чи кожному символу відповідає інший символ. Це завдання є для парсера.

3 МАТЕРІАЛИ І МЕТОДИ

Метою розробки системи є створення додатку для спрощення формул в логіці висловлювань (рис. 2). Усі задачі ІС поділені на такі групи: отримання вхідних даних від користувача та приведення їх до вигляду, зрозумілого системі; виділення і категоризація лексем виразів логіки висловлювань; перевірка синтаксичної валідності виразів логіки висловлювань; зведення формул до нормальної форми та їх мінімізація; порівняння перетворених формул на еквівалентність; генерування результатів та відображення їх користувачу. Кожна задача є самостійною програмною одиницю і реалізована модульно. В залежності від кількості виділених рівнів операцій, ІС поділяється на підсистеми (рис. 3):

– підсистема відображення даних, яка забезпечує взаємодію користувача з ІС: введення вхідних даних чи керуючих команд, виведення і форматування результату;

– підсистема ядра системи, яка реалізує основну логіку САА, тобто множину функцій, що аналізують та перетворюють вхідні дані, виконують обчислення і формують результат;

– підсистема управління даними, яка забезпечує зберігання даних в БД та доступ до них.

Ядро ІС складається з таких основних класів (рис. 4).

1. **Клас Main** є основним та має механізм зчитування вхідних даних від користувача. З цього класу створюють і викликають інші класи в порядку застосування перетворень вхідної стрічки. Присутній метод isDNF з тієї причини, що вихідний вираз після синтаксичного опрацювання може бути зразу заданий у вигляді ДНФ, відповідно відпадає потреба створювати об'єкт класу DNFMaker, що значно зекономить ресурси. Оскільки цей клас є головним, його зв'язки є суто *асоціативними*, які служать для взаємозв'язків між класами, є базовим семантичним елементом і структурою для багатьох типів з'єднань між об'єктами. Асоціації є тим механізмом, який надає об'єктам змогу обмінюватися даними між собою. Вони мають призначення і є одно- (лише один з об'єктів знає про існування іншого) чи двосторонніми (у межах зв'язку кожен з об'єктів може надсилати повідомлення іншому). Кожен з кінців асоціації має значення численності, яке визначає кількість об'єктів на відповідному кінці асоціації, які можуть мати зв'язок з одним з об'єктів на іншому кінці асоціації. У даному випадку, зв'язки характеризуються як один-до-багатьох, тобто з одного об'єкту класу Main можна створити і тримати зв'язок з декількома об'єктами класу Parser і класу DNFMaker. Або у зворотному випадку, кожному об'єкту класу Parser чи класу DNFMaker відповідає лише один об'єкт класу Main.

2. **Клас Parser** приймає стрічку виразу і повертає значення true, якщо вираз є синтаксично правильним і false в іншому випадку. Метод analize () є ядром самого парсера, в основі якого покладений алгоритм LL(1) Parser з використанням логіки подання формальних мов контекстно-вільних граматик, що ділять весь алфавіт на термінальні (клас Token) і нетермінальні (клас NonTerminal) символи. Правильність виразів визначають за допомогою стеку аналізатора (параметр stash) і вхідного виразу (expression) шляхом поступового застосування правил нетерміналів у стеку. Окрім зв'язку з об'єктом класу Main, клас Parser має ще один асоціативний зв'язок типу один-до-одного з класом Lexer.

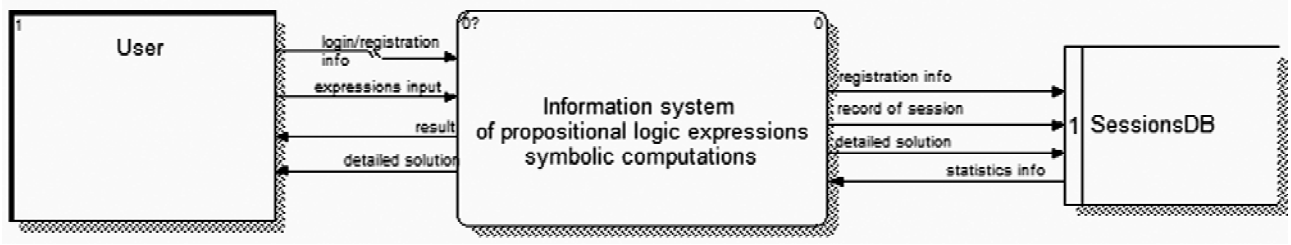

Рисунок 2 – Контекстна DFD ІС символьних обчислень виразів в логіці висловлювань

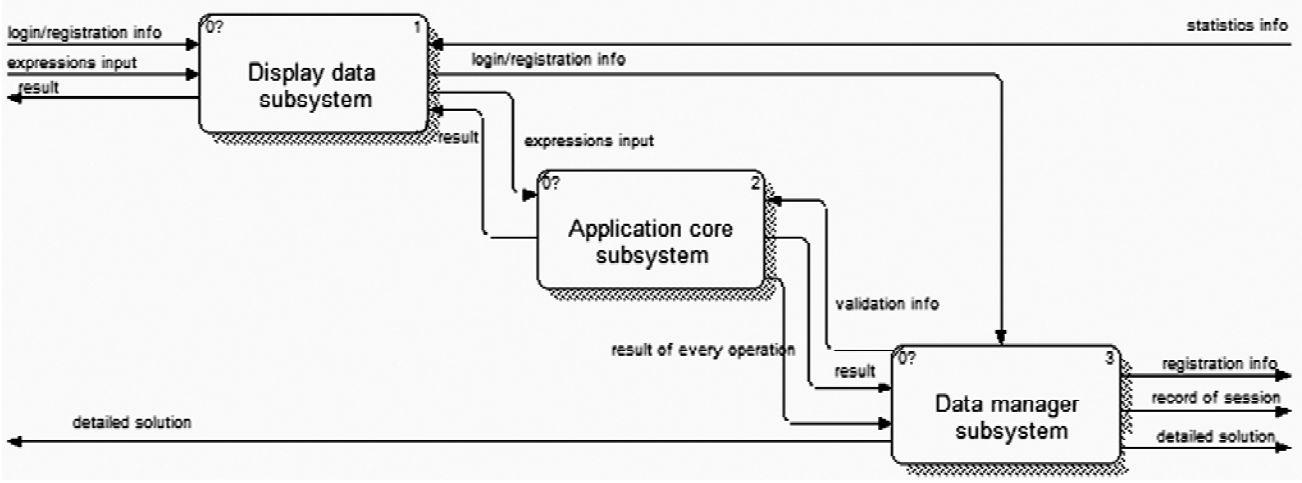

Рисунок 3 – Декомпозиція процесу символьних обчислень виразів в логіці висловлювань

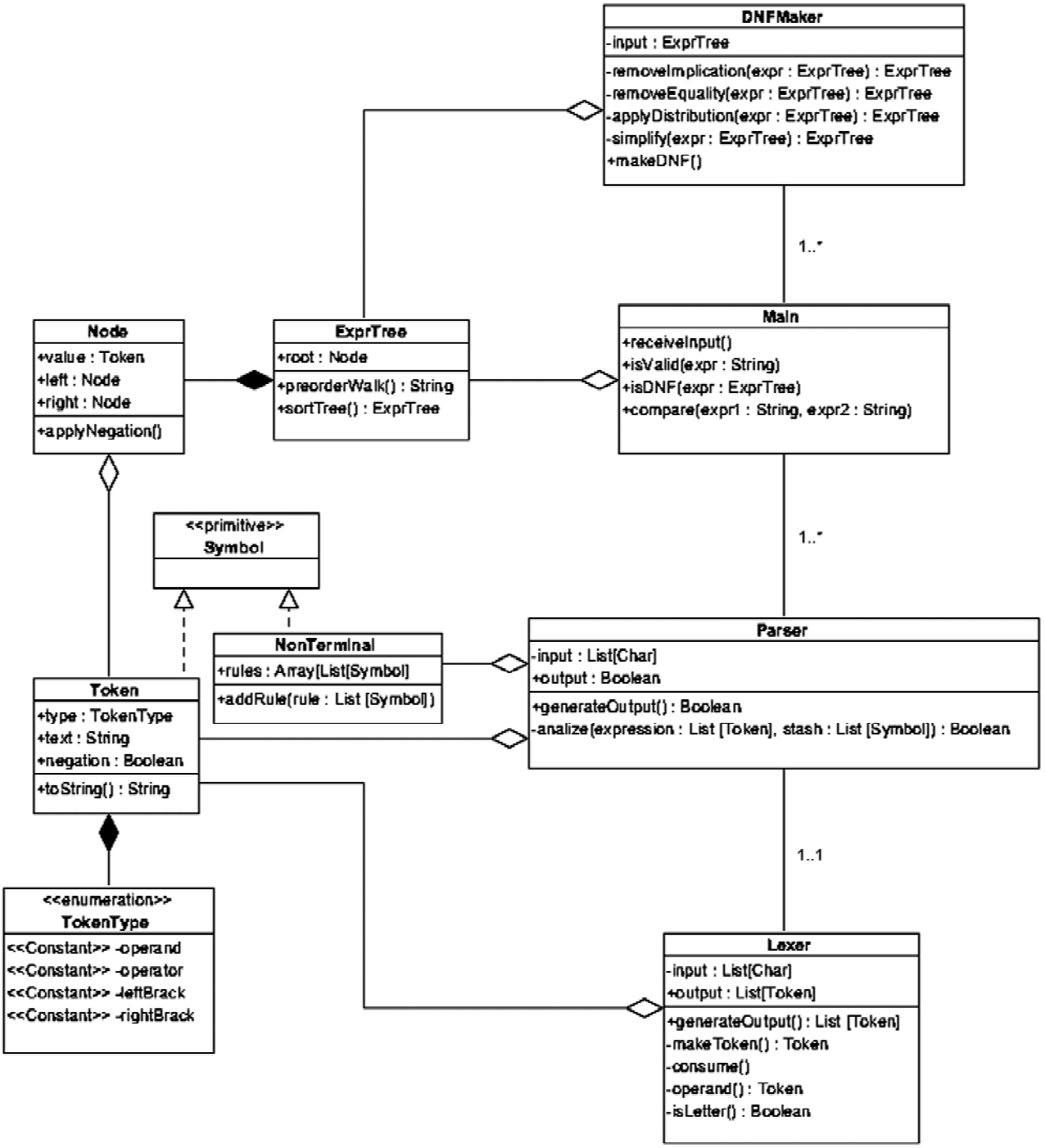

Рисунок 4 – Діаграма класів ядра ІС символьних обчислень в логіці висловлювань

3. **Клас Token** – це клас атомарного висловлювання, яке є одним з переліку (**TokenType** extends Enumeration, клас є сильно залежною сутністю, тому його зв'язок з класом Token характеризується як *композиційний*, або дуже сильно агрегаційний): операнд (operand), оператор (operation), ліва дужка (leftBrack) або права дужка (rightBrack). Поле text:String відображає ім'я операнда (якщо токен є операндом), знак операції (якщо токен є операцією або символом), для відкритої і для закритої дужки відповідно. Додаткове поле negation: Boolean має значення true якщо заперечення над токеном існує і false в іншому випадку. Клас має зв'язок типу *агрегації*, яка є особливим типом асоціацій, де два класи із зв'язком не є рівнозначними, тобто зв'язок типу "ціле-частина". За допомогою агрегації описують, яким чином клас у ролі цілого складається з інших класів у ролі частин. У агрегаціях клас у ролі цілого завжди має численність рівну одиниці.

4. **Клас NonTerminal** для нетерміналів, кожен об'єкт якого міститиме свій набір правил rules, що поданні у вигляді послідовності термінальних і нетермінальних символів. Цей клас разом з класом Token є нащадками абстрактного класу **Symbol** (для визначення об'єднуючого

типу термінальних (токенів) і нетермінальних символів, оскільки стек аналізатора міститиме об'єкти як одного так і іншого класу).

5. **Клас Lexer** приймає стрічку виразу і повертає той же вираз, розділений на токени: операнди, операції та дужки. Метод makeToken () є ключовим, в основі якого є алгоритм LL(1) Lexer. Додаткові методи operand() і isLetter призначенні для коректного виділення змінних виразу, що складаються з більше, ніж одного символу.

6. **Клас DNFMaker** перетворює правильний вираз у ДНФ методом makeDNF.

7. **Клас ExprTree** – клас типу виразу, над яким працює DNFMaker, оскільки для перетворення виразів використовувати форму дерева набагато зручніше. Дерево є бінарним із-за наявності лише бінарних операцій (унарне заперечення є як атрибут самого токена). Метод sortTree() є завершальним методом, і його призначення, використовуючи закони комутативності, відсортувати операнди в визначеному (наприклад, алфавітному порядку) для подальшої зручності посимвольного порівняння.

8. **Клас Node** є складовим класу ExprTree у вигляді однієї вершини. Поля left і right є вказівниками на лівого та правого сина вершини відповідно. Метод applyNegation є атомарним застосуванням закону подвійного заперечення, якщо value.negation == true, або запереченням значення вершини в іншому випадку.

На рис. 5 подана діаграма діяльності ядра ІС символьних обчислень в логіці висловлювань, яка складається з наступних етапів.

1) Отримання вхідних даних (ввід користувачем виразів та перетворення їх до відповідного вигляду для опрацювання ІС).

2) Перевірка виразу на коректність – це робота лексера і парсера системи, по завершенні якої відомо, чи вираз є коректним, і система продовжує роботу. В іншому випадку допущені помилки, через які система не зможе коректно опрацювати ці вирази. Тоді ІС генерує звіт про помилку (який тип та місце помилки) і завершує свою роботу.

3) Створення дерева виразу (перетворення стрічкового виразу у бінарне дерево, при цьому дужки упускаються, адже послідовність виконання виразів диктуватиметься їх положенням у дереві, з яким потім працюватиме DNFMaker).

4) Перевірка чи вираз є у ДНФ. Якщо ні, то виконання кроку 5, інакше – кроку 6.

5) Застосування закону еквівалентних перетворень та перехід до кроку 4. Цикл триватиме доти, поки у дереві не залишиться кон'юктивних вузлів на вищих рівнях.

6) Сортування клауз, використовуючи закони комутативності (посортувати операнди в визначеному порядку для подальшої зручності посимвольного порівняння).

7) Перевірка на рівність (посимвольне порівняння заданих виразів).

8) Генерування результатів (підтвердження або спростування рівності виразів).

Синтаксичний аналізатор реалізований як абстрактний автомат, який здійснює виконання певного набору команд. Його основним завданням є проаналізувати заданий вхідний символьний рядок на належність до мови, яка описується LL(k) граматикою, і являє собою програму роботи автомата. Завдяки аналогічності форм подання формул САА-схем і LL(k)-граматик даний автомат, окрім виконання функції граматичного аналізатора, здат-

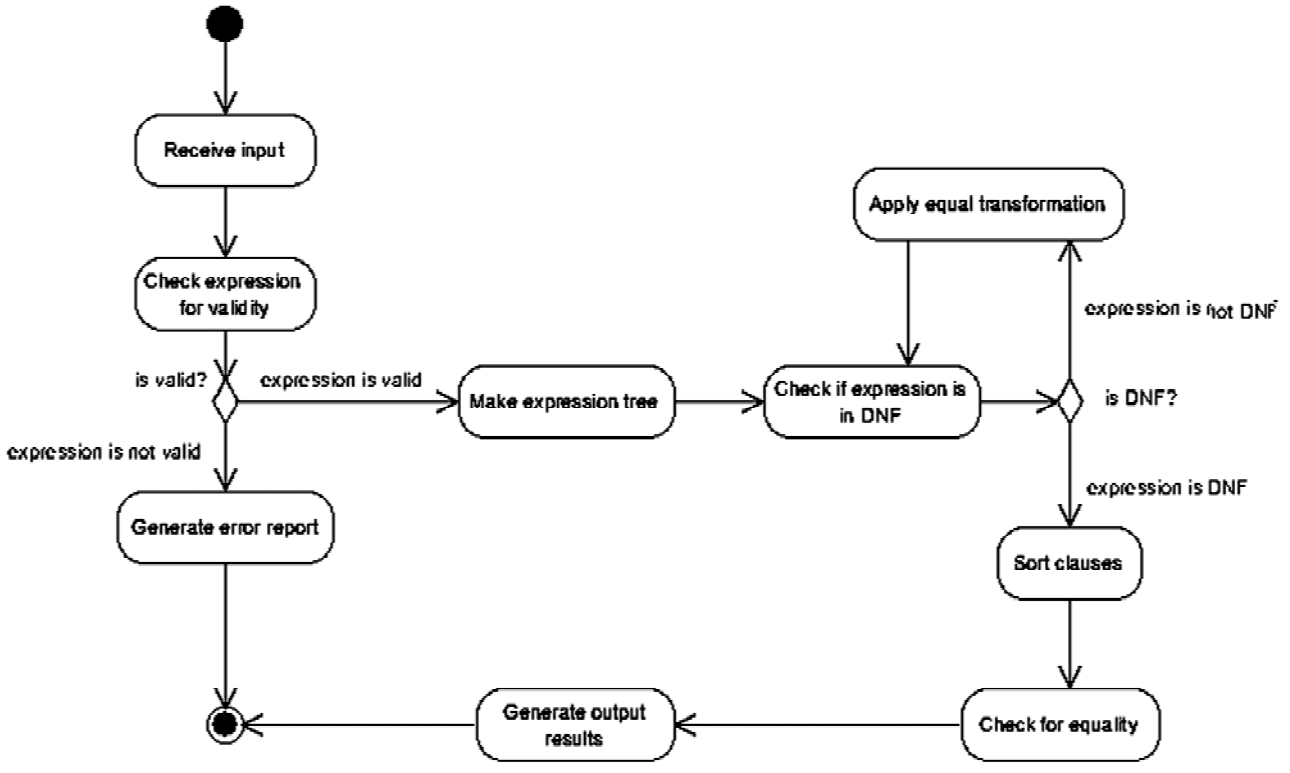

Рисунок 5 – Діаграма діяльності ядра ІС символьних обчислень в логіці висловлювань

ний здійснювати інтерпретацію САА-схем. Тому опис команд автомата розглядають як реалізацію алгоритму граматичного аналізу по LL(k)-граматиці або інтерпретацію САА-схем. САА-машина як автомат виконує наступні команди типу:

– ***B*<код>** є командами опрацювання відповідного нетерміналу з порядковим номером **<код>** в граматиці або складеного оператора в САА-схемі;

– ***A*<код>, *V*<код>** є командами виклику відповідної семантичної процедури, яка забезпечує трансляцію вхідного рядка у вихідний при побудові дерева виведення, або виконання оператора (предиката) САА-схеми, що заданий процедурою/функцією обраної для синтезу мови програмування;

– ***L* 'текст' і *K* 'текст'**, які задають порівняння фрагменту вхідного рядка з текстом, заданим в лапках команди *L*, *K*. При збігу цих фрагментів команда *K* формує логічний результат перевірки у вигляді глобальної логічної змінної САА-машини, а команда *L* додатково ще й пересуває вказівник схеми на кількість символів в тексті команди у випадку збігу його з вхідним потоком. Ці команди спеціалізовані для символьного опрацювання, тому переважно використовуються в режимі інтерпретації LL(k)-граматик;

– ***I* 'текст повідомлення'**, що забезпечують візуалізацію тексту повідомлення на екрані у вікні повідомлень САА-машини.

– ***W* <код>** для обчислення складеної логічної умови в LL (k)-граматиці або САА-схемах;

– **логічна команда & < операнд 1 > ... < операнд N > XL** задає обчислення кон'юнкції зазначених операндів логічного типу, перелічених без будь-яких додаткових розділових символів і зчитуються до послідовності символів *XL*;

– **логічна команда ! < операнд > ... < операнд > XL** задає обчислення диз'юнкції зазначених операндів логічного типу, перелічених без будь-яких додаткових розділових символів і зчитуються до послідовності символів *XL*;

– **логічна команда ¬< операнд >** задає обчислення заперечення вказаного логічного операнда; у всіх логічних командах операндами можуть бути команди *L*, *K*, *V*, або вкладені один в одного зазначені логічні команди !, &, ¬;

– **послідовне виконання команд автомата *X*;** задається їх послідовним записом без розділових знаків (цими командами є тільки команди типу *A*, *B*, *I*); їх послідовність завершує команда X;. Така послідовна гілка (*альтернатива*) задає одне правило підстановки для нетермінала граматики. Весь набір правил підстановки для конкретного нетермінала задається списком альтернатив і з точки зору імплементованої програми є конструкцією SWITCH.

– **альтернативне виконання** визначає виконання послідовних фрагментів програми (так званих альтернатив) в залежності від значення логічної умови, що стоїть на початку кожної альтернативи. У випадку LL(k)-граматик кількість альтернатив є довільна, і виконується та альтернатива, в якої умова входу в альтернативу виявилася істиною першою. При інтерпретації САА-схем альтернативне виконання здійснюється як оператор

ЯКЩО <умова входу 1 > то < гілка 1 > ІНАКШЕ < гілка 2 >;

– **циклічне виконання** послідовного фрагмента програми автомата записується у вигляді **C <логічний вираз> *XL* <послідовна гілка> *X*;** та забезпечує багаторазове виконання послідовної гілки (альтернативи) при істинному значенні логічного виразу. У САА-схемах аналогічна конструкції має вигляд ПОКИ НЕ <логічний вираз> ЦИКЛ <послідовна гілка>;

– **логічна константа *E*** рівна тотожно істинній логічній умові**.**

У табл. 1 наведені приклади відповідності фрагментів граматик і внутрішнього подання формул САА-схеми у вигляді послідовностей команд САА-машини.

Програма автомата перед інтерпретацією перетворюється в списковy внутрішню форму, що складається з:

– *Граматики*, яка визначена як стрічка команд САА-машини без розділових знаків, при цьому тексти повідомлень та лексем в командах K, L винесені у окремі таблиці, а в граматиці зберігається тільки їх порядковий номер у відповідній таблиці.

– *Таблиці лексем*, де в i стрічці таблиці задано індекс лексеми команди K_i/L_i в стрічці символів, де підряд записані всі лексеми граматики без розділових знаків та довжина лексеми (кількість символів в ній).

– *Таблиці інформаційних повідомлень*, яка збудована аналогічно до таблиці лексем: в рядку таблиці задано індекс повідомлення команди в рядку символів, де підряд записані всі повідомлення граматики без розділових знаків, а також довжина повідомлення.

– *Таблиці опису складеної логічної умови* у вигляді логічного предикату з командами диз'юнкції, кон'юнкції та заперечення, яка задає індекс цього опису в стрічці граматики.

– *Таблиці опису альтернативи граматик* (гілки розгалуження в САА-схемі), яка визначає індекс початку опису альтернативи в рядку символів граматики.

– *Таблиці визначення нетерміналів* (складних операторів САА-схеми), кожний рядок якої складається з двох елементів – індексу таблиці опису альтернатив, де вказано початок першої альтернативи нетерміналу, та індексу таблиці опису альтернатив, де вказано початок останньої альтернативи цього нетерміналу в рядку граматики. Різниця між цими двома елементами визначає кількість альтернатив нетерміналу.

Таке внутрішнє подання будується модулем САА-машини inner_gram. Інтерпретація команд типу *B* здійснюється шляхом запам'ятовування в магазині адреси наступної за нею команди та перехід на виконання першої команди альтернативи цього нетерміналу. Поча-

Таблиця 1 – Таблиця перетворень з LL(k) граматики у САА схему

Форма опису LL(k)-граматики	Відповідна формула САА-схеми	Подання формули САА-схеми
B_1=*L*'схема' B_2 A_1 *X*; *K*'обєкт' B_1 A_2 *X*; *E X*;	B_1= ЯКЩО V_1+V_2 ТО A_1 ІНАКШЕ B_2;	B_1=!V_1 V_2 *XL* A_1 *X*; *E* B_2 *X*;
B_2=**C**&*L*'кінець'*K*'='*XL* A_2*X*;	B_2= ПОКИ НЕ V_3 ЦИКЛ A_2;	B_2=*C* V_3 *XL* A_2 *X*;

ток цієї альтернативи обчислюють за номером нетерміналу, який задає індекс *Таблиці визначення нетерміналів альтернатив*, а та визначає початок опису кожної алернативи цього нетерміналу в *Таблиці опису альтернативи граматик*. По завершенню інтерпретації реалізації B, яка наступає при виконанні команди команди X;, відбувається повернення на наступну за цим нетерміналом команду, адреса цієї команди вибирається зі стеку. Механізм інтерпретації команд B ідентичний механізму виклику процедур в мовах програмування.

Інтерпретація команд A та V здійснюється за допомогою виклику їх реалізацій, що задані у вигляді перемикачів SWITCH, де номер команди A задає умову входу в гілку перемикача операторів, а номер команди V – умову входу в гілку перемикача базових логічних умов. При цьому результат виконання логічної функції V зберігається в глобальній логічній змінній САА-машини. Інтерпретація логічних команд автомату здійснюється з використанням традиційної логіки мов програмування та глобальної логічної змінної САА-машини. Генерація програми по САА-схемі та набору реалізацій елементарних операторів та умов реалізують довільною мовою програмування, оскільки таке перетворення задається граматикою, і самі семантичні процедури цієї граматики просто визначають генерацію коду в обраній мові програмування. Заміна семантичних процедур однієї мови програмування на іншу викликає результат генерації коду та подання алгоритму в відповідній мові.

4 ЕКСПЕРИМЕНТИ

Перераховані команди САА-машини є базовими для опису LL(k)-граматики САА-мови. При створенні цієї граматики враховувалася вкладеність конструкцій, наприклад,

$$B_1 = B_7 B_2 X;\ B_2 = K\text{'СХЕМА'}\ A_5\ B_7\ A_4\ B_7\ B_8\ X;$$

$=EI$'очікується ключове слово СХЕМА' $B_{45}\ X$;

$B_8 = K\text{';'}\ A_1\ B_7\ B_3\ X;\ =EI$'очікується ;' $B_{45}\ X$;

Наведений фрагмент граматики аналізує заголовок схеми. Нетермінал B_7 здійснює прохід через послідовність прогалин будь-якої довжини. Нетермінал B_2 відповідає за аналіз ключового слова СХЕМА та ідентифікатора схеми. За перенесення слова СХЕМА в формулу цієї схеми відповідає семантична процедура A_5, а семантична процедура A_4 визначає ідентифікатор схеми, присвоює схемі унікальний внутрішній ідентифікатор та записує його в формулу. У випадку відсутності ключового слова СХЕМА або помилки при його написанні команда K'СХЕМА' сформує хибне логічне значення, тому вибереться наступна за порядком альтернатива нетермінала B_2 а саме: EI'очікується ключове слово СХЕМА' $B_{45}\ X$;. Ця альтернатива починається тотожно істиною логічною умовою E, тому командою I'очікується ключове слово СХЕМА' друкується повідомлення про помилку та припиняється трансляція через синтаксичну помилку. Нетермінал B_4 здійснює синтаксичний аналіз лівої частини рівняння. Стандартні ідентифікатори складеного оператора починаються символом B, складеної умови – символом W, змістові ідентифікатори операторів подають в подвійних лапках, а змістовні ідентифікатори складних умов подають в одинарних лапках, наприклад,

$B_4 = K\text{'}B\text{'}\ A_5\ B_7\ B_{10}\ B_7\ B_8\ X;\ =K\text{'''}\ A_0\ A_9\ A_0\ B_7\ B_{10}\ B_7\ B_8\ X;$

$=K\text{'}W\text{'}\ A_8\ B_7\ B_{11}\ B_7\ B_8\ X;\ =K\text{'\#'}\ A_0\ A_{10}\ A_0\ B_7\ B_{11}\ B_7\ B_8\ X;$

$=EI$'невірний ідентифікатор в лівій частині рівняння' $B_{45}\ X$;

Наступний фрагмент граматики здійснює поглинання пробілів.

$B_7 = K\text{' '}\ A_1\ B_{44}\ X;\ =EI$'очікується пробіл' $B_{45}\ X;\ B_{44} = K\text{' '}\ A_0\ B_{44}\ X;\ =EX;.$

Нетермінал B_3 здійснює опрацювання тіла схеми з декількох рівнів проектування, заданих складними операторами та складними умовами. Ці рівні задаються окремими рівняннями з ідентифікаторами операторів чи умов в лівій частині рівняння та алгебраїчним виразами в САА в правій частині рівняння, наприклад, $B_3 = K$'КІНЕЦЬ' $B_{45}\ X;\ =EB_4\ B_3\ X;$.

Нетермінал B_{10} аналізує наявність знаку = перед правою частиною рівняння, що визначає складний оператор. Нетермінал B_{11} аналізує наявність знаку = перед правою частиною рівняння, що визначає складну умову. В правій частині рівняння використовують стандартні ідентифікатори операторів та логічних умов (складних і базових) та їх змістовні ідентифікатори. Відповідає за аналіз правої частини рівняння B_5. *I*-ідентифікатори схеми пов'язані між собою операціями алгебри розгалуження та циклу, наприклад,

$B_{10} = K\text{'='}\ A_1\ B_7\ B_5\ B_7\ B_6\ X;\ =EI$'очікується =' $B_{45}\ X$;

$B_5 = K\text{'}A\text{'}\ A_{11}\ X; = K\text{'}B\text{'}\ A_5\ X; = K\text{'''}\ A_7\ X; = K\text{'ПОКИ'}\ A_4\ B_7\ B_9\ X;$

$=K\text{'ЯКЩО'}\ A_4\ B_7\ B_{14}\ B_7\ B_{12}\ X;\ =K\text{'('}\ A_1\ B_7\ B_5\ B_7\ B_{13}\ X;$

$=EI$'невірний оператор в лівій частині рівняння схеми' $B_{45}\ X;$.

Послідовне виконання (*) опрацьовується нетермінали B_6 та B_{13}. B_{13} відповідає за фрагменти операторних виразів, взятих в дужки ().

$B_6 = K\text{'*'}\ A_1\ B_7\ B_5\ B_7\ B_6\ X;\ =K\text{';'}\ A_1\ B_7\ X;$

$=EI$'очікується * або ; в правій частині рівняння' $B_{45}\ X$;

$B_{13} = K\text{')'}\ A_1\ B_7\ B_5\ B_7\ B_6\ X;\ =K\text{'*'}\ A_1\ B_7\ B_5\ B_7\ B_{13}\ X;$

$=EI$'очікується) або * в правій частині рівняння' $B_{45}\ X$;

Нетермінал B_{12} аналізує продовження оператора розгалуження з ключового слова ТО

$$B_{12} = K\text{'ТО'}\ A_2\ B_7\ B_5\ B_7\ B_{15}\ B_7\ X;$$

$=EI$'очікується продовження оператора ЯКЩО – операторна дужка ТО' $B_{45}\ X$;

B_{15} аналізує продовження оператора розгалуження з ключового слова ІНАКШЕ, тобто

$$B_{15} = K\text{'ІНАКШЕ'}\ A_6\ B_7\ B_5\ B_7\ B_6\ X;$$

$=EI$'очікується операторна дужка ІНАКШЕ' $B_{45}\ X$;

Наступний приклад описує аналіз рівняння для подання складної логічної умови.

$$B_{11} = K\text{'='}\ A_1\ B_7\ B_{17}\ B_7\ B_{18}\ X;$$

$=EI$'очікується дорівнює в логічному рівнянні схеми' $B_{45}\ X$;

$B_{17} = K\text{'}V\text{'}\ A_{12}\ X;\ =K\text{'}W\text{'}\ A_{13}\ X;\ =K\text{'\#'}\ A_8\ X;$

$=EI$'невірний логічний операнд в логічній умові' $B_{45}\ X$;

B_{18} описує операції логіки як диз'юнкцію (+), кон'юнкцію (.), заперечення (!), тобто

$$B_{18} = K\text{'+'}\ A_1\ B_7\ B_{17}\ B_{18}\ X;\ = K\text{'.'}\ A_1\ B_7\ B_{17}\ B_{18}\ X;\ = K\text{'!'}\ A_1\ B_7\ B_{17}\ B_{18}\ X;\ = K\text{';'}\ A_1\ B_7\ X;$$

Команда X& відзначає кінець граматики. B_9 аналізує фрагмент оператора циклу:

$$B_9 = K\text{'НЕ'}\ A_2\ B_7\ B_{14}\ B_7\ B_{16}\ X;\ = EI\text{'очікується НЕ в операторі циклу ПОКИ НЕ'}\ B_{45}\ X;$$

Аналогічно B_{16} продовжує аналіз оператора циклу з ключового слова ЦИКЛ, тобто

$$B_{16} = K\text{'ЦИКЛ'}\ A_4\ B_7\ B_5\ B_7\ B_6\ X;\ = EI\text{'очікується операторна дужка ЦИКЛ'}\ B_{45}\ X;.$$

Граматику програмою перетворюють в списокову форму та в масив команд САА-машини. Кожна команда визначається типом, заданим полем типу char та порядковим номером. Цей масив складається з послідовностей альтернатив всіх нетерміналів. Для визначення початку опису конкретного нетерміналу в такому масиві створюються таблиці T_B, де зафіксовані індекси масиву граматики як початок альтернатив. Всі термінальні слова (ключові слова схеми) зібрані в окрему стрічку. Для вибору ключового слова при синтаксичному аналізі номер терміналу при командах K, L задає індекс таблиці терміналів T_TERM. Остання є масивом записів, які складаються з індексу початку терміналу в спільній символьній стрічці та його довжини. Аналогічно побудована таблиця повідомлень T_INF. Загальне вікно програми має 8 робочих областей та одну кнопку Translate, яка ініціює початок роботи системи (рис. 6).

Усі робочі області на рис. 6 пронумеровані для зручності пояснення.

1. Область для відображення введених початкових даних. Завантажують дані з текстового файлу за допомогою Open file. Вхідними даними є САА-схеми або граматика для САА-машини (завдяки подібності структур LL(k)-граматики та формул САА-схем, система володіє можливістю використання САА-машини як інтерпретатора САА-схем, так і створення тестового апарату для відлагодження САА-схем).

2. Відображення спискової форми граматики САА-схем. LL(k)-граматика в списковій формі є керуючою програмою для абстрактної машини з фіксованим набором команд, тобто є в певній мірі вхідними даними, але генеруються автоматично системою.

3. Таблиця граматики T_TERM (лексем або термінальних значень, які властиві заданій граматиці). Лексемами є ключові слова (СХЕМА, КІНЕЦЬ тощо), слова-оператори (ПОКИ, НЕ, ЯКЩО тощо), знаки логічних операторів тощо.

4. Таблиця повідомлень T_INF зберігає повідомлення граматики, які призначені для відображення у вихідній стрічці (робоча область 8), щоб повідомити користувача про синтаксичну помилку у схемі (якщо така виявлена).

5. Таблиця початків альтернатив T_B, які в робочій області 2. Кожна окрема альтернатива розміщується між двома командами типу X_1 – вона відповідає команді граматики X;. У таблиці зафіксовані індекси масиву граматики, що є початками альтернатив.

6. Таблиця альтернатив T_ALT складається з двох полів: перше визначає індекс таблиці T_B, що фіксує початок в масиві граматики першої альтернативи нетерміналу, друге поле – індекс останньої альтернативи нетерміналу. Різниця між цими полями задає кількість альтернатив нетерміналу з номером, що є індексом таблиці T_ALT. Для розуміння організації граматики розглянемо для прикладу другу альтернативу з робочої області 2. Вона знаходиться між командами X_1 і має вигляд $K_0\ A_5\ B_7\ A_4\ B_7\ B_8$. Ця альтернатива відповідає правилу підстановки: K'СХЕМА' $A_5\ B_7 A_4\ B_7\ B_8$. Командою K_0 задана лексема СХЕМА в списковій формі. Тут 0 – це індекс лексеми в таблиці T_TERM (таблиця подана на інтерфейсній схемі під номером 3). Перша лексема (значення індексу 0) в цій таблиці і є ключовим словом СХЕМА. Альтернатива (1) починається в списковій формі за індексом 3. Ця альтернатива є першою в списку альтернатив нетерміналу B_2 початкової форми граматики, яка зручна для її розуміння.

$$B_2 = K\text{'СХЕМА'}\ A_5\ B_7\ A_4\ B_7\ B_8\ X;\ = EI\text{'очікується ключове слово СХЕМА'}\ B_{45}\ X;.$$

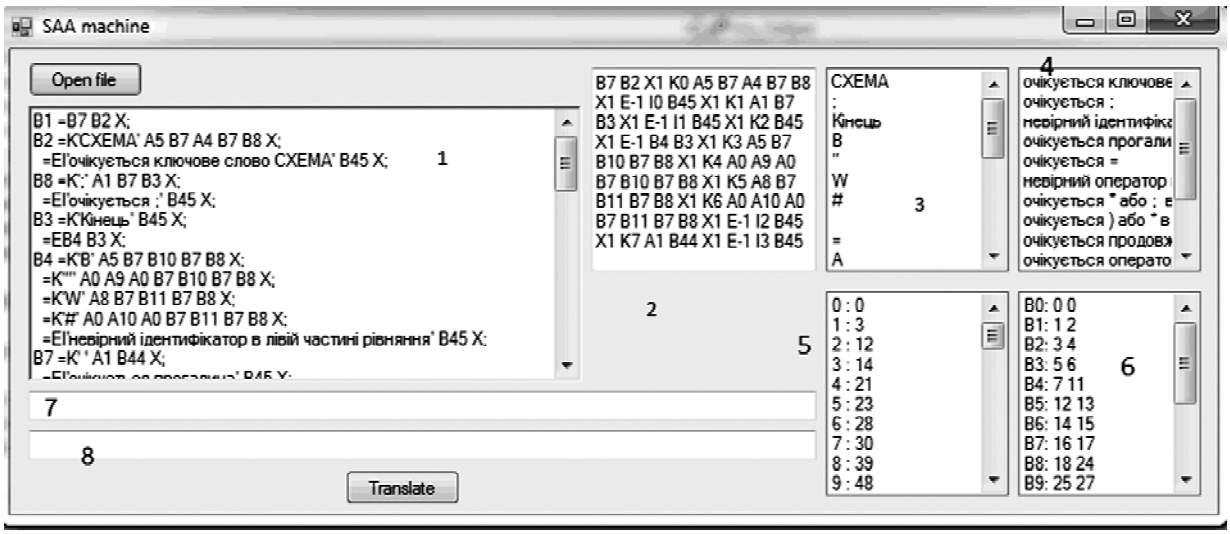

Рисунок 6 – Загальний вигляд робочого вікна системи

Альтернатива (1) розміщена в списковій формі граматики починаючи з індексу 3, вона є другою альтернативою граматики, тому за індексом *i*5=2–1=1 (в таблиці 5 інтерфейсної схеми записано цей індекс). Так як ця альтернатива є першою альтернативою нетерміналу B_2, то по індексу *i*6=2–1=1 таблиці 6 повинен бути записаний індекс *i*5 таблиці 5. Цей індекс записується в перший елемент стрічки таблиці 6 з індексом *i*6, цей елемент в таблиці визначений як поле beg_alt – першої альтернативи. Наступна альтернатива: E–1 I_0 B_{45}. Вона є другою і завершальною альтернативою нетерміналу B_2. Початок її в граматиці задається індексом 12, цей індекс записаний в таблицю 5 в рядок 3 (індекс *i*5=2). Так як це завершальна альтернатива нетерміналу B_2, то в таблицю 6 по індексу *i*6 в другий компонент стрічки таблиці T_ALT заноситься індекс *i*5. Тому 2 рядок таблиці 6 має вигляд 1 2.

7. Вихідний рядок. Ця область призначена для відображення результату успішного завершення роботи програми, а саме – згенерованої формули САА-схеми.

8. Лог системи. У разі виявлення синтаксичних помилок, система припиняє свою роботу з виведеним повідомленням про причину з таблиці T_INF (робоча область 4).

5 РЕЗУЛЬТАТИ

Усі вказані таблиці заповнюються автоматично САА-машиною як генератора граматики із звичайного текстового файлу, який описує граматику за правилами LL(k)-подання. Далі, вони використовуються САА-машиною як синтаксичним аналізатором для прочитання і граматичного розбору заданої САА-схеми.

Роботу системи найкраще подати шляхом покрокового обчислення контрольного прикладу. Для цього використаємо алгоритм бульбашкового сортування. Для початку, розглянемо регулярну схему послідовного бульбашкового сортування по зростанню числового масиву записану в аналітичному вигляді, який зручний для формульного аналізу та перетворень. У наведеній регулярній схемі в явній формі подана алгоритмічна частина сортування, а дані в ній явно не вказуються, тобто $\alpha_1\{A_1 * \alpha_1\{\alpha_2(E \vee A_2) * A_3\}\}$. Умови α_1,α_2; оператори A_1, A_2, A_3, цієї схеми є базовими.

Для початку роботи необхідно ввести початкову граматику САА-схем, яка зможе повно описати мову. Для цього попередньо необхідно підготувати текстовий файл з правилами виведення LL(k)-граматики (рис. 7). Відкривши заданий файл у програмі системи (рис. 8), те саме наповнення появиться у найбільшій з робочих областей програми. Після введення даних запускають програму кнопкою Generate. Назва цієї кнопки міняється в залежності від поточної функції САА-машини. Оскільки на першому етапі основним завданням системи є граматичний розбір і побудова схем граматики, режимом САА-машини буде синтаксичний аналізатор. Однак при побудові САА-формул з САА-схем, САА-машина виконуватиме роль інтерпретатора, а тому і кнопка, що ініціюватиме запуск САА-машини матиме назву Translate. У випадку успішного завершення програми, у результаті буде створена граматика, за якою проводитиметься подальший аналіз САА-схем (рис. 9).

Наступний етап вимагає введення САА-схеми алгоритму бульбашкового сортування.

СХЕМА babl;

″babl″ = ″старт″ * ″встановити вказівник B(1) перед початком масиву (MASS)″

* ″виконати FL=1″

* ПОКИ НЕ ‘FL = 0 ‘

ЦИКЛ (″виконати FL = 0″*

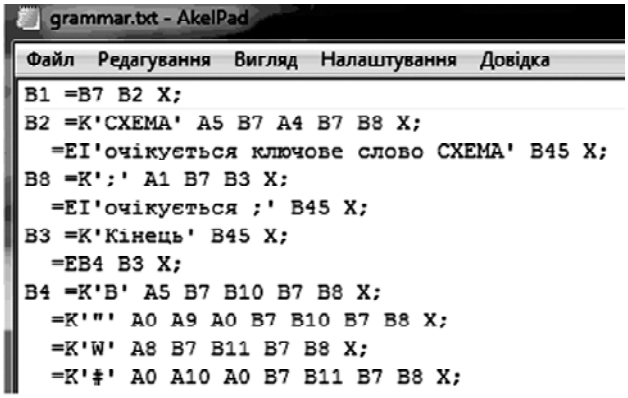

Рисунок 7 – Текстовий файл граматики мови САА схем

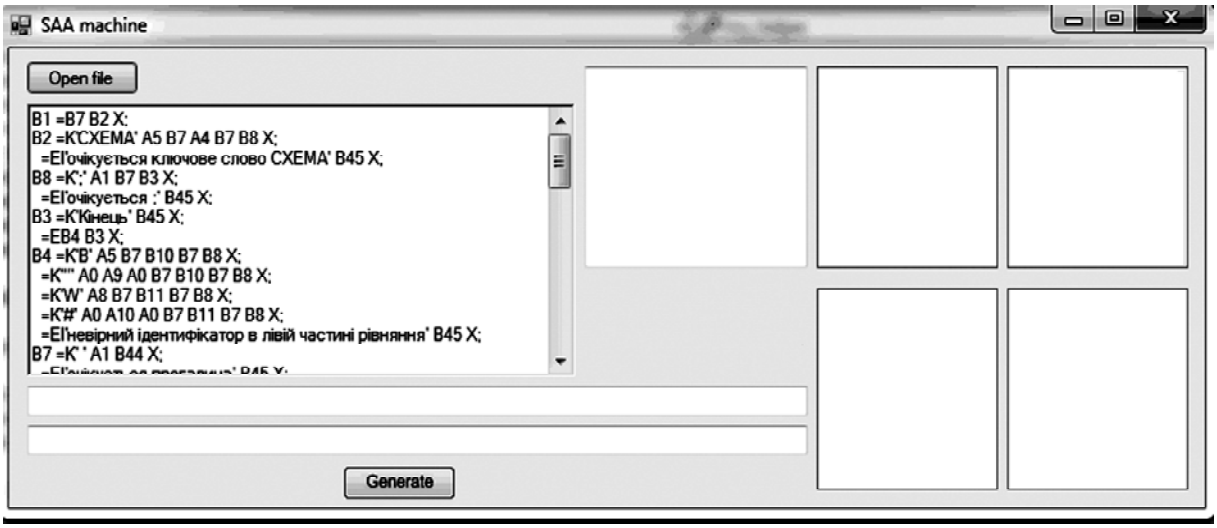

Рисунок 8 – Введення граматики у систему

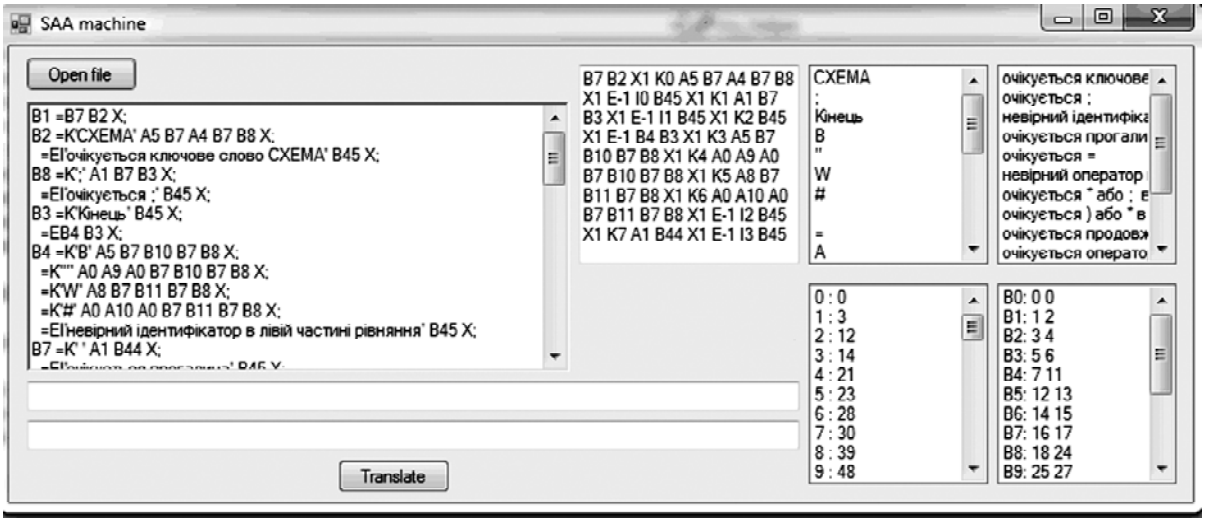

Рисунок 9 – Результат роботи синтаксичного аналізу

″встановити вказівник В(1) перед початком масиву (MASS)″ *

ПОКИ НЕ ‘вказівник В(1) в кінці (MASS) ‘

ЦИКЛ (ЯКЩО ‘по В(1) елемент зліва L>R елемента справа в (MASS)’

ТО ″переставити елементи L,R по В(1) місцями в MASS″

* ″виконати FL=1″

* ″зсунути У(1) на (1) по (MASS) вправо″))

*

* ФІН;

КІНЕЦЬ

Основними об’єктами мови САА опису регулярних схем алгоритмів є абстракції операторів і умов. При цьому в мові розрізняються елементарні оператори та умови і складені оператори та умови. Складені оператори та умови фіксують певні рівні проектування конкретної регулярної схеми. Аналогічно, основними абстракціями мови САА є складені і елементарні оператори та умови. Для подання складених і елементарних операторів, умов введені стандартні і смислові ідентифікатори. Ту ж саму САА-схему, змодифікувавши її під один з варіантів вдосконаленого бульбашкового сортування, подано у табличному вигляді (табл. 2) з вказанням номеру стрічки початків кожної з операцій. Задана схема подана одним рівнянням. Базисними елементарними операторами в ній є оператори, змістовні ідентифікатори, які починаються в рядках 1, 2, 5, 8, 11, 12. Базисні умови подані змістовними ідентифікаторами в рядках 4, 7, 10.

Таблиця 2 – Таблична схема вдосконаленого алгоритму бульбашкового сортування

№ стрічки	САА-схема
	СХЕМА bable_удосконалена;
1	"Бульбашка / ВД" = "старт" *******
2	"Встановити вказівники B_1 та B_2 в кінець (mass)"
3	******
4	ПОКИ НЕ ‘В (2) перед початком (mass)’ ЦИКЛ
5	("Встановити В (1) перед початком (mass)"
6	**********
7	ПОКИ НЕ ‘Відстань між В (2) і В (1) в (mass) дорівнює 0
8	ЦИКЛ ("Пересунути В (1) на (1) по (mass) вправо"
9	**********
10	ЯКЩО ‘L по У (1)> R по В (1) в (mass)’
11	ТО "Поміняти місцями R по В (1)"
12	ІНАКШЕ "Зрушити В (1) на (1) по (mass) вліво"))
13	КІНЕЦЬ

6 ОБГОВОРЕННЯ

Вдосконалення звичайного бульбашкового сортування здійснюється за рахунок введення вказівника В (2), який фіксує відсортовану частину масиву з найбільшими його елементами. Вказівник В (1), який початково встановлений на початок масиву, за один прохід внутрішнього циклу зсувається від початку масиву до вказівника В (2), пересуваючи при цьому найбільший на цій ділянці елемент масиву на місце, що фіксується вказівником В (2). Після злиття вказівників В (1) і В (2), В (2) зсувається на один символ вліво, а В (1) встановлюється на початок масиву. У схемі на рис. 10 в змістовних ідентифікаторах використовують об’єкти в круглих дужках, які є параметрами відповідних цим ідентифікаторам операторів або умов. Наприклад, в оператор зі змістовним ідентифікатором «Пересунути В (1) на (1) по (mass) вправо» залежить від трьох параметрів:– вказівника B_1; – величини зсуву по масиву (1); – імені масиву, по якому відбувається переміщення вказівника. При цьому реалізація такого оператора нагаду’ опис процедури із зазначенням цих трьох параметрів як формальних параметрів, тоді як змістовний ідентифікатор в схемі є викликом цієї реалізації із зазначенням фактичних параметрів. Такий виклик при синтезі програми може замінюватися як справжнім викликом реалізації операторів (у вигляді процедури), так і як вставлений фрагмент програми, що реалізує цей оператор, із заміною в цьому фрагменті формальних параметрів на фактичні (макрогенерація).

ВИСНОВКИ

У роботі досліджено застосування САА для граматичного аналізу символьних обчислень виразів логіки висловлювань. САА-схема розглядається як вхідна стрічка синтаксичного аналізу. Цю стрічка під час трансляції перетворюється у макет програми у вибраній мові програмування. Перетворення здійснено в процесі побудови дерева виведення і задано атрибутною граматикою мови САА-схем. Розроблено механізм синтезу програм за їх САА-схемою. Це було здійснено у вигляді абстрактної САА-машин. Ця машина через подібність механізмів порівневого проектування САА-схем та LL(k)-граматик може служити не тільки для задач синтезу, але й для

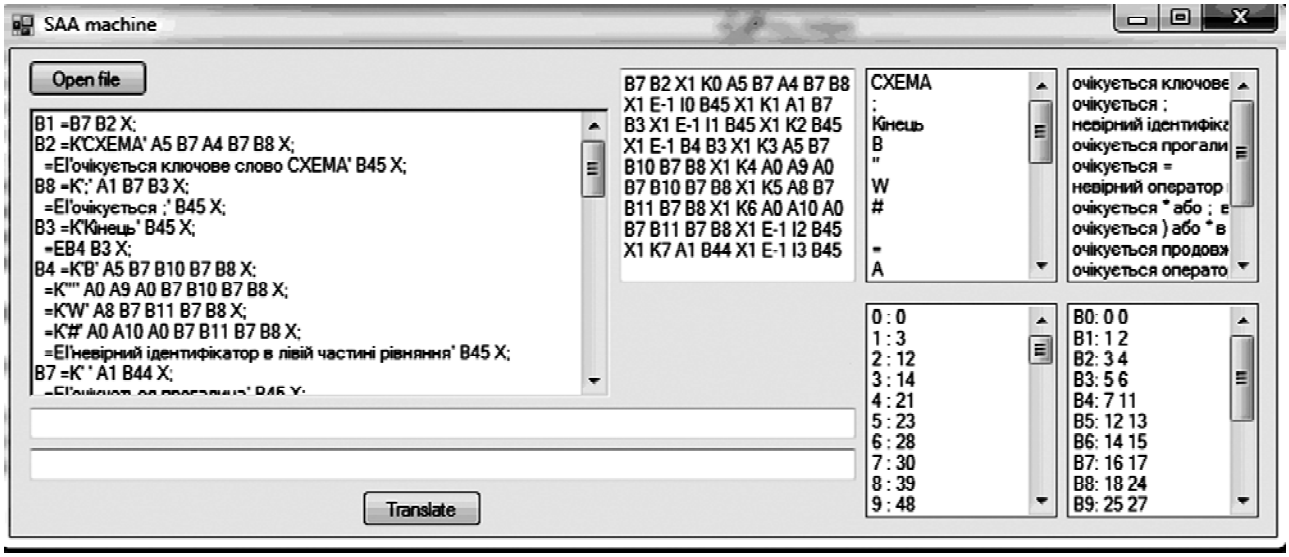

Рисунок 10 – Результат опрацювання САА схеми

інтепретації (відлагодження) САА-схем програм. Розроблено інтелектуальну систему генерації програм за допомогою засобів синтаксичного аналізу в системах алгоритмічних алгебр Глушкова. Система здатна проводити синтаксичний аналіз та забезпечувати користувача повною інформацією про помилки, якщо такі будуть виявлені. Автоматично генерує подання граматики в списковому вигляді, що дає змогу реалізувати метод рекурсивного спуску без використання рекурсивних процедур. Синтаксичний аналізатор системи представлений у вигляді абстрактної машини з фіксованим набором команд, для якої LL(k)-граматика в списковій формі є керуючою програмою. Для імплементації ядра системи була використана об'єктно-функціональна мова програмування Scala. Функціональні частини були декомпоновані на окремі модулі, кожний з яких реалізовується окремим об'єктом у Scala-проекті. Результатом системи є формульне подання САА-схеми алгоритму. Така абстракція дозволяє на подальших етапах трансформаціїї генерувати будь який алгоритм у задану мову програмування, за умови опису цією мовою базових компонент імплементації алгоритмів: операторів присвоєння, циклів, умови та логічних порівнянь. У майбутньому планується удосконалити систему шляхом її розширення для реалізації ширшого кола опрацювання мов програмування й задач, які пов'язанні з САА.

СПИСОК ЛІТЕРАТУРИ

1. Глушков В. Методы символьной мультиобработки / В. Глушков, Г. Цейтлин, Е. Ющенко. – К. : Наук. думка, 1980. – 252 с.
2. Глушков В. Алгебра. Языки. Программирование / В. Глушков, Г. Цейтлин, Е. Ющенко. – К. : Наук. думка, 1989. – 376 с.
3. Цейтлин Г. Введение в алгоритмику / Г. Цейтлин. – К. : Фара, 1999. – 310 с.
4. Цейтлин Г. Алгебры Глушкова и теория клонов / Г. Цейтлин // Кибернетика и системный анализ. – 2003. – № 4. – С. 48–58.
5. Цейтлин Г. Структурное программирование задач символьной мультиобработки / Г. Цейтлин // Кибернетика. – 1983. – № 5. – С. 22–30.
6. Цейтлин Г. Распараллеливание алгоритмов сортировки / Г. Цейтлин // Кибернетика. – 1989. – № 6. – С. 67–74.
7. Цейтлин Г. Проектирование последовательных алгоритмов сортировки / Г. Цейтлин // Программирование. – 1989. – № 3. – С. 3–21.
8. Многоуровневое структурное проектирование программ: Теоретические основы, инструментарий / [Е. Ющенко, Г. Цейтлин, В. Грицай, Т. Терзян]. – М. : Финансы и статистика, 1989. – 208 с.
9. Калужнин Л. Об алгоритмизации математических задач / Л. Калужнин // Пробл. кибернетики. – 1959. – Вып. 2. – С. 51–69.
10. Мартинюк Т. Методи та засоби паралельних перетворень векторних масивів даних / Т. Мартинюк, В. Хом'юк. – В. : УНІВЕРСУМ-Вінниця, 2005. – 203 с.
11. Дорошенко А. О методе проектирования абстрактного типа данных в алгебре алгоритмики / А. Дорошенко, О. Иовчев // Проблеми програмування. – 2012. – № 1. – С. 3–16.
12. Акуловский В. Некоторые аспекты формализации архитектурного этапа разработки алгоритмов / В. Акуловский // Проблеми програмування. – 2009. – № 2. – С. 3–11.
13. Алгеброалгоритмические модели и методы параллельного программирования / [Ф. Андон, А. Дорошенко, Г. Цейтлин, Е. Яценко]. – К. : Академпериодика, 2007. – 634 с.
14. Иовчев В. Инструментальные средства алгебры алгоритмики на платформе WEB 2.0 / В. А. Иовчев, А. С. Мохница // Проблеми програмування. – 2010. – № 2–3. – С. 547–556.

Стаття надійшла до редакції 23.02.2016.
Після доробки 04.03.2016.

Лытвын В. В.[1], Бобык И. О.[2], Высоцкая В. А.[3]

[1]Д-р техн. наук, профессор, заведующий кафедры «Информационные системы и сети» Национального университета «Львовская политехника», Львов, Украина

[2]Канд. физ.-мат. наук, доцент кафедры «Высшей математики» Национального университета «Львовская политехника», Львов, Украина

[3]Канд. техн. наук, доцент кафедры «Информационные системы и сети» Национального университета «Львовская политехника», Львов, Украина

ПРИМЕНЕНИЕ СИСТЕМЫ АЛГОРИТМИЧЕСКИХ АЛГЕБР ДЛЯ ГРАММАТИЧЕСКОГО АНАЛИЗА СИМВОЛЬНЫХ ВЫЧИСЛЕНИЙ ВЫРАЖЕНИЙ ЛОГИКИ ВЫСКАЗЫВАНИЙ

Разработана архитектура и реализовано программную систему грамматического анализа схем системы алгебраических алгебры и их интерпретации. Программная система позволяет автоматизировано генерировать программы по таким созданными схемами и их

отлаживать в соответствующих схемах. Определены четкое разделение системы алгебраических алгебры на отдельные модули, каждый из которых будет характеризоваться своим функциональным нагрузкам. Использованы методы синтаксического анализа для разработки и представления грамматики таких схем. Реализовано автоматическое ее преобразования в списочную форму. Разработан машины системы алгебраических алгебры как абстрактный механизм интерпретации грамматики средствами синтаксического анализа. Словарь V состоит из конечного не пустые множества лексических единиц. Выражение над V является цепочкой конечной длины лексических единиц с V. Прежний цепочку, не содержит лексических единиц, обозначим через. Множество всех лексических единиц над V обозначим. Язык над V является опилками. Язык задают через множество всех лексических единиц языка или через определение критерия, которому должны удовлетворять лексические единицы, чтобы принадлежать языке. Еще один важный способ задать язык – через использование порождающих грамматики. Грамматика состоит из множества лексических единиц разного типа и множества правил или продукций построения выражения. Грамматика имеет словарь V, есть множество лексических единиц для построения выражений языка. Некоторые лексические единицы словаря (терминальные) не могут заменяться другими лексическими единицами. Текст реализует структурно представленную деятельность, предполагает субъект и объект, процесс, цель, средства и результат, которые отражаются в содержательно-структурных, функциональных, коммуникативных показателях. Единицами внутренней организации структуры текста является алфавит, лексика (парадигматика), грамматика (синтагматика), парадигмы, парадигматические отношения, синтагматические отношения, правила идентификации, высказывания, между фразовой единство и фрагменты-блоки. На композиционном уровне выделяют предложения, абзацы, параграфы, разделы, главы, во главу, страницы и т.д., которые, кроме предложения, косвенно связанные с внутренней структурой, поэтому не рассматриваются. С помощью базы данных (базы терминов / морфем и служебных частей речи) и определенных правил анализа текста выполняют поиск срока. Синтаксические анализаторы работают в два этапа: идентифицируют содержательные лексемы и создают дерево разбора.

Ключевые слова: текст, украиноязычный, алгоритм, контент-мониторинг, ключевые слова, лингвистический анализ, синтаксический анализ, порождающих грамматики, структурная схема предложения, информационная лингвистическая система.

Lytvyn V. V.[1], Bobyk I. O.[2], Vysotska V. A.[3]
[1]Dr. Sc., Professor, Head of Information Systems and Networks Department, Lviv Polytechnic National University, Lviv, Ukraine
[2]PhD, Associate Professor of Higher Mathematics Department, Lviv Polytechnic National University, Lviv, Ukraine
[3]PhD, Associate Professor of Information Systems and Networks Department, Lviv Polytechnic National University, Lviv, Ukraine

APPLICATION OF ALGORITHMIC ALGEBRA SYSTEM FOR GRAMMATICAL ANALYSIS OF SYMBOLIC COMPUTATION EXPRESSIONS OF PROPOSITIONAL LOGIC

The architecture and implemented a software system parsing schemes of algebraic algebra and their interpretation. The software system allows to generate automated in such schemes create a program and debug the relevant schemes. A clear division of algebraic algebra into separate modules is defined, each of which will be characterized by its functional load. The methods of parsing for the development and presentation of grammar such schemes are used. Automatic transforming it in a list form is implemented. A machine system of algebraic algebra as an abstract interpretation of the mechanism of grammar parser is defined. The vocabulary V consists of finite not empty set of lexical units. The expression on V is a finite-length string of lexical units with V. An empty string does not contain lexical items and is denoted by Λ. The set of all lexical units over V is denoted as V'. The language over V is a subset V'. The language displayed through the set of all lexical units of language or through definition criteria, which should satisfy lexical items that belong to the language. Another is one important method to set the language through the use of generative grammar. The grammar consists of a lexical units set of various types and the rules or productions set of expression constructing. Grammar has a vocabulary V, which is the set of lexical units for language expressions building. Some of lexical units of vocabulary (terminal) can not be replaced by other lexical units. The text realizes structural submitted activities through provides subject, object, process, purpose, means and results that appear in content, structural, functional and communicative criteria and parameters. The units of internal organization of the text structure are alphabet, vocabulary (paradigmatics), grammar (syntagmatic) paradigm, paradigmatic relations, syntagmatic relation, identification rules, expressions, unity between phrasal, fragments and blocks. On the compositional level are isolated sentences, paragraphs, sections, chapters, under the chapter, page etc. that (except the sentence) indirectly related to the internal structure because are not considered. With the help of a database (database for terms/morphemes and structural parts of speech) and defined rules of text analysis searching terms. Parsers operate in two stages: lexemes content identifying and a parsing tree creates.

Keywords: text, a Ukrainian, algorithm, content monitoring, keywords, linguistic analysis, parsing, generative grammar, structured scheme sentences, information linguistic system.

REFERENCES

1. Glushkov V., Tseitlin G., Yushchenko E. Methods symbolic multiprocessing. Kiev, Science, Dumka, 1980, 252 p.
2. Glushkov V., Tseitlin G., Yushchenko E. Algebra. Languages. Programming. Kiev, Science, Dumka, 1989, 376 p.
3. Tseitlin G. Introduction to Algorithms. Kiev, Spotlight, 1999, 310 p.
4. Tseitlin G. Glushkov Algebra and theory of clones, *Cybernetics and Systems Analysis*, 2003, No. 4, pp. 48–58.
5. Tseitlin G. Structured programming tasks symbolic multiprocessing, *Cybernetics,* 1983, No. 5. pp. 22–30.
6. Tseitlin G. Parallelization algorithms for sorting, *Cybernetics,* 1989, No. 6, pp. 67–74.
7. Tseitlin G. Design sequential algorithms sorting, *Programming*, 1989, No. 3, pp. 3–21.
8. Yushchenko E., Tseitlin G., Gritsay V., Terzian T. Multilevel structural design program: Theoretical Foundations, tools. Moscow, Finances and Statistics, 1989, 208 p.
9. Kaluzhnin L. On algorithmization mathematical problems, *Probl. Cybernetics*, 1959, Issue 2, pp. 51–69.
10. Martyniuk T., Homyuk B. Metodi that zasobi parallel peretvoren vector masiviv danih. Vinnitsa UNIVERSUM-Vinnitsa, 2005, 203 p.
11. Doroshenko A., Iovcev O. A method of designing an abstract data type in the algebra of algorithmics, *Problems programuvannya*, 2012, No. 1, pp. 3–16.
12. Akulovsky V. Some aspects of formalizing the architectural design phase algorithms, *Problems programuvannya*, 2009, No. 2, pp. 3–11.
13. Andon F., Doroshenko A., Tseitlin G., Yatsenko E. Algebroalgoritmicheskie models and methods of parallel programming. Kiev, Academperiodika, 2007, 634 p.
14. Iovcev V., Mohnitsa A. S. Tools algorithmic algebra platform WEB 2.0, *Problems programuvannya*, 2010, No. 2–3, pp. 547–556.