|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_Программное обеспечение ЭВМ и информационные технологии\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по лабораторной работе №4**

**по курсу «Проектирование компиляторов»**

**Тема «Синтаксический анализатор операторного предшествования»**

**Вариант 1**

Студент \_\_\_ИУ7-23М\_\_\_\_ \_\_\_\_\_Н.И. Иксарица\_\_\_\_

(группа) (И.О.Фамилия)

Преподаватель \_\_\_\_ А.А. Ступников\_\_\_\_

(И.О.Фамилия)

*2021 г.*

**Цель работы**: приобретение практических навыков реализации таблично управляемых синтаксических анализаторов на примере анализатора операторного предшествования.

**Задачи**

Задачи работы:

1. Ознакомиться с основными понятиями и определениями, лежащими в основе синтаксического анализа операторного предшествования.
2. Изучить алгоритм синтаксического анализа операторного предшествования.
3. Разработать, тестировать и отладить программу синтаксического анализа в соответствии с предложенным вариантом грамматики.
4. Включить в программу синтаксического анализ семантические действия для реализации синтаксически управляемого перевода инфиксного выражения в обратную польскую нотацию.

**Подготовка грамматики**

В листинге 1 приведена грамматика из индивидуального варианта. В листинге 2 представлена приведённая грамматика, правые части правил которой не содержат смежные нетерминалы.

Листинг 1. Грамматика из индивидуального варианта.

|  |
| --- |
| <выражение> ->      <простое выражение> |      <простое выражение> <операция отношения> <простое выражение>  <простое выражение> ->      <терм> |      <знак> <терм> |      <простое выражение> <операция типа сложения> <терм>  <терм> ->      <фактор> |      <терм> <операция типа умножения> <фактор>  <фактор> ->      <идентификатор> |      <константа> |      ( <простое выражение> ) |      not <фактор>  <операция отношения> ->      = | <> | < | <= | > | >=  <знак> ->      + | -  <операция типа сложения> ->      + | - | or  <операция типа умножения> ->      \* | / | div | mod | and |

Листинг 2. Грамматика с удалёнными парными нетерминалами.

|  |
| --- |
| <выражение> ->  <простое выражение> |  <простое выражение> == <простое выражение> |  <простое выражение> <> <простое выражение> |  <простое выражение> < <простое выражение> |  <простое выражение> <= <простое выражение> |  <простое выражение> > <простое выражение> |  <простое выражение> >= <простое выражение>  <простое выражение> ->  <терм> |  + <терм> |  - <терм> |  <простое выражение> + <терм> |  <простое выражение> - <терм> |  <простое выражение> or <терм> |  <терм> ->  <фактор> |  <терм> \* <фактор> |  <терм> / <фактор> |  <терм> div <фактор> |  <терм> mod <фактор> |  <терм> and <фактор>  <фактор> ->  <идентификатор> |  <константа> |  ( <простое выражение> ) |  not <фактор> |

**Матрица отношений**

В таблица 1 представлена матрица операторного предшествования. Помимо терминалов, описанных в грамматике, матрица содержит отношения для маркера-ограничителя строки «$».

Таблица 1 – Матрица операторного предшествования

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | == | <> | < | <= | > | >= | + | - | or | \* | / | div | mod | and | идент | конст | ( | ) | not | $ |
| == |  |  |  |  |  |  | < | < | < | < | < | < | < | < | < | < | < |  | < | > |
| <> |  |  |  |  |  |  | < | < | < | < | < | < | < | < | < | < | < |  | < | > |
| < |  |  |  |  |  |  | < | < | < | < | < | < | < | < | < | < | < |  | < | > |
| <= |  |  |  |  |  |  | < | < | < | < | < | < | < | < | < | < | < |  | < | > |
| > |  |  |  |  |  |  | < | < | < | < | < | < | < | < | < | < | < |  | < | > |
| >= |  |  |  |  |  |  | < | < | < | < | < | < | < | < | < | < | < |  | < | > |
| + | > | > | > | > | > | > | > | > | > | < | < | < | < | < | < | < | < |  | < | > |
| - | > | > | > | > | > | > | > | > | > | < | < | < | < | < | < | < | < |  | < | > |
| or | > | > | > | > | > | > | > | > | > | < | < | < | < | < | < | < | < |  | < | > |
| \* | > | > | > | > | > | > | > | > | > | > | > | > | > | > | < | < | < |  | < | > |
| / | > | > | > | > | > | > | > | > | > | > | > | > | > | > | < | < | < |  | < | > |
| div | > | > | > | > | > | > | > | > | > | > | > | > | > | > | < | < | < |  | < | > |
| mod | > | > | > | > | > | > | > | > | > | > | > | > | > | > | < | < | < |  | < | > |
| and | > | > | > | > | > | > | > | > | > | > | > | > | > | > | < | < | < |  | < | > |
| идент | > | > | > | > | > | > | > | > | > | > | > | > | > | > |  |  |  | > |  | > |
| конст | > | > | > | > | > | > | > | > | > | > | > | > | > | > |  |  |  | > |  | > |
| ( |  |  |  |  |  |  | < | < | < | < | < | < | < | < | < | < | < | = | < |  |
| ) | > | > | > | > | > | > | > | > | > | > | > | > | > | > |  |  |  | > |  | > |
| not | > | > | > | > | > | > | > | > | > | > | > | > | > | > | < | < | < |  | < | > |
| $ | < | < | < | < | < | < | < | < | < | < | < | < | < | < | < | < | < |  | < |  |

**Листинги кода**

Листинг 3. Вспомогательные функции.

|  |
| --- |
| (ns cc.lab04.utils  (:require [clojure.set]))  (defmacro when-let\*  ([bindings & body]  (if (seq bindings)  `(when-let [~(first bindings) ~(second bindings)]  (when-let\* ~(drop 2 bindings) ~@body))  `(do ~@body))))  (defn get-all-prods  "Получение множества всех пар правил вида [lhs rhs]."  [{:keys [prods]}]  (-> (for [lhs (keys prods)  rhs (get prods lhs)]  [lhs rhs])  set))  (defn find-prod-by-rhs  "Поиск правила по цепочке терминалов (!!!) из правой части."  [rhs {:keys [terms] :as grammar}]  (let [rhs-terms (filter terms rhs)  prods (get-all-prods grammar)]  (reduce (fn [\_ [l r]]  (when-let\* [r-terms (filter terms r)  \_ (seq r-terms)  \_ (= r-terms rhs-terms)]  (reduced {l r})))  nil  prods))) |

Листинг 4. Функции синтаксического анализатора.

|  |
| --- |
| (ns cc.lab04.analyzer  (:require [cc.lab04.utils :refer [find-prod-by-rhs]]  [com.walmartlabs.cond-let :refer [cond-let]])  (:gen-class))  (defn g-reduce  "Свёртка."  [stack {:keys [precedence-table subroutines border-marker] :as grammar} & {:keys [debug?]}]  (loop [stack stack  chain '()]  (cond-let  :let [\_ (when debug? (println "stack -->" stack))  \_ (when debug? (println "chain -->" chain))]  ; Стек содерджит единственный символ - маркер границы.  (= (vec stack) [border-marker])  (do (when debug? (println :1))  (into stack chain))  :let [top (peek stack)  bottom (peek (pop stack))  relation (get-in precedence-table [bottom top])  chain (conj chain top)  \_ (when debug? (println "new chain -->" chain))]  (contains? #{">" "="} relation)  (do (when debug? (println :2))  (recur (pop stack) chain))  (nil? relation)  (do (println :3)  (throw (Exception. "LOLERROR")))  ; Отношение "<".  :let [prod (find-prod-by-rhs chain grammar)  \_ (when debug? (println "prod -->" prod))]  (nil? prod)  (do (when debug? (println :4))  stack)  :else  (let [lhs (-> prod keys first)  rhs (-> prod vals first)]  (eval (get subroutines rhs))  (do (when debug? (println :5))  (recur (pop stack) []))))))  (defn syntax-analyzer  [tokens {:keys [precedence-table nonterms border-marker] :as grammar} & {:keys [debug?]}]  (loop [stack (list border-marker)  tree :success  tokens tokens]  (cond-let  :let [token (first tokens)  sp (peek stack)]  (= sp token border-marker)  tree    ; Пропуск нетерминала.  (some #{token} nonterms)  (do (if debug? (println "Пропуск нетерминала." {:stack stack :tokens tokens}))  (recur stack tree (rest tokens)))  :let [relation (get-in precedence-table [sp token])]  (nil? relation)  (throw (Exception. (format "Конструкция ... %s %s ... запрещена." sp token)))    ; Перенос.  (contains? #{"<" "="} relation)  (do (if debug? (println "Перенос." {:stack stack :tokens tokens}))  (recur (conj stack token) tree (rest tokens)))    ; Свёртка.  :else  (do (if debug? (println "Свёртка." {:stack stack :tokens tokens}))  (recur (g-reduce stack grammar :debug? debug?) tree tokens))))) |

Листинг 5. Функция -main.

|  |
| --- |
| (ns cc.lab04.core  (:require [clojure.string]  [cc.lab04.analyzer :refer [syntax-analyzer]])  (:gen-class))  (defn -main  [grammar-path text & {:keys [debug?] :or {debug? false}}]  (let [grammar (-> grammar-path slurp read-string)  tokens (-> text  (clojure.string/trim)  (clojure.string/split #"\s+")  (conj "$"))]  (syntax-analyzer tokens grammar :debug? debug?))) |

**Пример работы программы**

Для входной строки “*(not идентификатор) + (- константа) >= (идентификатор \* (идентификатор - константа) mod константа)*” программа строит следующее логическое выражение в обратной польской нотации “*идентификатор not константа - + идентификатор идентификатор константа - \* константа mod >=*”.

**Заключение**

В ходе выполнения работы были изучены методы таблично управляемого синтаксического анализа на примере анализатора операторного предшествования.

Исходная грамматика индивидуального варианта была преобразована к форме без смежных нетерминалов в правых частях правил.

Для полученной грамматики была построена матрица операторного предшествования.

Для сформированных исходных данных была реализована программа восходящего синтаксического анализатора операторного предшествования, пример работы которой был приведён выше.