|  |  |
| --- | --- |
| 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 №2**

**по курсу «Проектирование компиляторов»**

**Тема «Преобразования грамматик»**

**Вариант 3**

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 наиболее важных видов преобразований грамматик, чтобы удовлетворить требованиям алгоритмов синтаксического разбора.

**Задачи**

Задачи работы:

1. Принять к сведению соглашения об обозначениях, принятые в литературе по теории формальных языков и грамматик и кратко описанные в приложении.
2. Познакомиться с основными понятиями и определениями теории формальных языков и грамматик.
3. Детально разобраться в алгоритме устранения левой рекурсии.
4. Разработать, тестировать и отладить программу устранения левой рекурсии.
5. Разработать, тестировать и отладить программу устранения ε-правил.

**Вариант 3**

Постройте программу, которая в качестве входа принимает произвольную КС-грамматику G = (Ν, Σ, P, S) и преобразует ее в эквивалентную КС-грамматику G' = (Ν', Σ', P', S') без ε-правил.

**Листинги кода**

Листинг 1. Удаление левой рекурсии.

|  |
| --- |
| (ns cc.lab02.morph    (:require [cc.lab02.helpers :refer [json->grammar index-dissoc find-occurrence-indexes]]              [clojure.math.combinatorics :as combo]              [clojure.set]))  (defn remove-direct-left-recursion    [nt grammar & {:keys [merge-nonterms? reset-new-nonterms?]}]    (let [grammar (if reset-new-nonterms? (dissoc grammar :new-nonterms) grammar)          old-prods (-> grammar :prods (get nt))          betas (filter #(and (seq %)                              (-> % (first) (not= nt))) old-prods)          betas (map vec betas)          alphas (filter #(and (> (count %) 1)                               (-> % (first) (= nt))) old-prods)          alphas (map (comp vec rest) alphas)]      (cond        (empty? alphas) grammar        (empty? betas) (throw (Exception. "Отсутствует правило для выхода из рекурсии."))        :else (let [new-nt (str nt "'")                    nt-prods (concat betas (map #(conj % new-nt) betas))                    nt-prods (set nt-prods)                    new-nt-prods (concat alphas (map #(conj % new-nt) alphas))                    new-nt-prods (set new-nt-prods)]                (-> grammar                    (assoc-in [:prods nt] nt-prods)                    (assoc-in [:prods new-nt] new-nt-prods)                    (update (if merge-nonterms? :nonterms :new-nonterms)                            clojure.set/union #{new-nt}))))))  #\_(remove-direct-left-recursion "E" (json->grammar "resources/grammar.json"))  (defn insert-rules    "Для всех правил вида `target -> [source gamma]` грамматики grammar осуществялется замена     нетерминала source правыми частями правил `source -> [sigma\_i]`.     gamma и sigma i-ые - цепочки терминалов и нетерминалов.     Итог - правила вида `target -> [simga\_i gamma]`."    [target source grammar]    (let [source-chains (-> grammar :prods (get source))          target-chains (-> grammar :prods (get target))          dc-pairs (for [t-chain target-chains                         :when (= (first t-chain) source)]                     [t-chain (map #(vec (concat % (rest t-chain))) source-chains)])          target-chains (reduce (fn [chains [d c]]                                  (-> chains                                      (disj d)                                      (concat c)                                      (set)))                                target-chains                                dc-pairs)]      (assoc-in grammar [:prods target] target-chains)))  (defn remove-left-recursion [grammar]    (loop [curr-nt (-> grammar :nonterms first)           nts-visited []           nts-to-visit (-> grammar :nonterms rest)           grammar grammar]      (if (nil? curr-nt)        (-> grammar            (update :nonterms clojure.set/union (:new-nonterms grammar))            (dissoc :new-nonterms))        (let [grammar (reduce #(insert-rules curr-nt %2 %) grammar nts-visited)              grammar (remove-direct-left-recursion curr-nt grammar)]          (recur (first nts-to-visit)                 (conj nts-visited curr-nt)                 (rest nts-to-visit)                 grammar)))))  #\_(remove-left-recursion (json->grammar "resources/grammar.json")) |

Листинг 2. Удаление ε-правил.

|  |
| --- |
| (ns cc.lab02.morph    (:require [cc.lab02.helpers :refer [json->grammar index-dissoc find-occurrence-indexes]]              [clojure.math.combinatorics :as combo]              [clojure.set]))    (defn find-generating-nonterms    "Функция для поиска нетерминалов, пораждающих терминал/нетерминал sym."    ([grammar sym]     (let [eps-nts (->> (:prods grammar)                        (filter #(get (second %) [sym]))                        (map first)                        (set))]       (find-generating-nonterms grammar sym eps-nts)))    ([grammar sym eps-nts]     (let [new-nts (->> (:prods grammar)                        (map (fn [[k v]]                               (when (reduce #(or % (every? eps-nts %2)) false v)                                 k)))                        (filter some?)                        (set)                        (clojure.set/union eps-nts))]       (if (= new-nts eps-nts)         new-nts         (find-generating-nonterms grammar sym new-nts)))))  #\_(find-generating-nonterms (json->grammar "resources/grammar.json") "F")  #\_(combo/selections [0 1] 3)  #\_(combo/subsets [0 1 2])  (defn remove-epsilon-rules [grammar]    (let [eps-nts (find-generating-nonterms grammar (:epsilon grammar))          find-replacement-for-chain          (fn [chain]            (->> (find-occurrence-indexes chain eps-nts)                 (combo/subsets)                 (map (comp reverse sort))                 (reduce (fn [acc to-dissoc]                           (->> (reduce #(index-dissoc % %2) chain to-dissoc)                                (vec)                                (conj acc)))                         [])))          eliminate-eps-chains          (fn [chains]            (->> chains                 (mapcat find-replacement-for-chain)                 (filter #(and (not= % [(:epsilon grammar)])                               (seq %)))                 (set)))          prods (reduce (fn [prods [nt chains]]                          (assoc prods nt (eliminate-eps-chains chains)))                        {}                        (:prods grammar))            grammar (assoc grammar :prods prods)          grammar (if (get eps-nts (:start-symbol grammar))                    (let [new-start (str (:start-symbol grammar) "'")]                      (-> grammar                          (assoc :start-symbol new-start)                          (update :nonterms conj new-start)                          (assoc-in [:prods new-start] #{[(:start-symbol grammar)]                                                         [(:epsilon grammar)]})))                    grammar)]      grammar)) |

**Пример работы программы**

1. **Удаление левой рекурсии**

Исходная грамматика:

|  |
| --- |
| {      "name": "G\_0",      "terms": ["a", "+", "\*", "(", ")"],      "nonterms": ["E", "T", "F"],      "prods": {          "E": [              ["E", "+", "T"],              ["T"]          ],          "T": [              ["T", "\*", "F"],              ["F"]          ],          "F": [              ["(", "E", ")"],              ["a"]          ]      },      "startSymbol": "E",      "epsilon": "eps"  } |

Полученная грамматика без левой рекурсии:

|  |
| --- |
| {      "terms": [")", "\*", "(", "a", "+"],      "startSymbol": "E",      "name": "G\_0",      "nonterms": ["T", "E'", "E", "F", "T'"],      "prods": {          "T": [              ["F", "T'"],              ["F"]          ],          "E'": [              ["+", "T"],              ["+", "T", "E'"]          ],          "E": [              ["F", "E'"],              ["F", "T'"],              ["F", "T'", "E'"],              ["F"]          ],          "F": [              ["(", "E", ")"],              ["a"]          ],          "T'": [              ["\*", "F"],              ["\*", "F", "T'"]          ]      },      "epsilon": "eps"  } |

1. **Удаление ε-правил**

Исходная грамматика:

|  |
| --- |
| {      "name": "G\_eps",      "terms": ["a", "d", "c"],      "nonterms": ["S", "A", "B", "C"],      "prods": {          "S": [              ["A", "B", "C", "d"]          ],          "A": [              ["a"],              ["eps"]          ],          "B": [              ["A", "C"]          ],          "C": [              ["c"],              ["eps"]          ]      },      "startSymbol": "S",      "epsilon": "eps"  } |

Полученная грамматика без ε-правил:

|  |
| --- |
| {      "terms": ["d", "a", "c"],      "startSymbol": "S",      "name": "G\_eps",      "nonterms": ["S", "C", "B", "A"],      "prods": {          "S": [              ["B", "C", "d"],              ["A", "C", "d"],              ["C", "d"],              ["B", "d"],              ["A", "d"],              ["d"],              ["A", "B", "d"],              ["A", "B", "C", "d"]          ],          "C": [              ["c"]          ],          "B": [              ["A", "C"],              ["A"],              ["C"]          ],          "A": [              ["a"]          ]      },      "epsilon": "eps"  } |

**Заключение**

В результате выполнения лабораторной работы были изучены алгоритмы удаления левой рекурсии и ε-правил, а также создана программа, позволяющая выполнять следующие преобразования над грамматикой, загружаемой из файла в формате JSON:

* построение новой грамматики, эквивалентной исходной, без левой рекурсии;
* построение новой грамматики, эквивалентной исходной, без ε-правил.