Лабораторная работа № 2

Преобразования грамматик

1. Цель и задачи работы

Цель работы: приобретение практических навыков реализации наиболее важных (но не всех) видов преобразований грамматик, чтобы удовлетворить требованиям алгоритмов синтаксического разбора.

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 преобразования грамматики в соответствии с предложенным вариантом.

2. Материал для изучения и ознакомления

Перед выполнением работы рекомендуется ознакомиться со следующими материалами:

Формальный язык. URL: https://ru.wikipedia.org/wiki/Формальный_язык
Формальная грамматика. URL: https://ru.wikipedia.org/wiki/Формальная_грамматика
Иерархия Хомского. URL: https://ru.wikipedia.org/wiki/Иерархия_Хомского
Контекстно-свободная грамматика. URL: https://ru.wikipedia.org/wiki/Контекстно-свободная_грамматика
Context-free grammar. URL: https://en.wikipedia.org/wiki/Context-free_grammar
Left recursion. URL: https://en.wikipedia.org/wiki/Left_recursion

3. Теоретическая часть

Данную грамматику часто требуется модифицировать так, чтобы порождаемый ею язык приобрел нужную структуру. Общего алгоритмического метода, который придавал бы данному языку произвольную структуру, не существует. Но с помощью ряда преобразований можно видоизменить грамматику, не испортив порождаемого грамматикой языка. В данной лабораторной работе рассматривается несколько преобразований такого рода. Для быстрого погружения в данную предметную область рекомендуется самостоятельно проработать главу 3 (параграфы 3.1, 3.2, 3.3) учебного пособия [3].

4. Практическая часть

Для решения задач лабораторной работы необходимо обратиться к следующим алгоритмам, примерам и упражнениям:

1) Устранение левой рекурсии: Алгоритм 2.13. [1], Пример 2.28. [1], Алгоритм 4.8. [2], Пример 4.9. [2], Алгоритм 4.10. [2], Пример 4.11. [2], работа [4].
2) Устранение недостижимых символов: Алгоритм 2.8. [1]
3) Устранение бесполезных символов: Алгоритм 2.9. [1], Упражнение 2.4.6. [1].
4) Преобразование в грамматику без ε-правил: Алгоритм 2.10. [1], Пример 2.23. [1], Упражнение 2.4.11. [1].
5) Устранение цепных правил: Алгоритм 2.11. [1], Пример 2.24. [1],
6) Преобразование к приведенной грамматике: Упражнение 2.4.13. [1].
7) Преобразование к нормальной форме Хомского: Алгоритм 2.12. [1], Пример 2.26. [1], Упражнение 2.4.16. [1].
8) Преобразование к нормальной форме Грейбах 1: Алгоритм 2.14. [1], Пример 2.29. [1], Упражнение 2.4.19. [1].
9) Преобразование к нормальной форме Грейбах 2: Алгоритм 2.15. [1], Пример 2.30. [1], Упражнение 2.4.19.

[1].

При реализации алгоритма преобразования грамматики возникает вопрос о форме представления исходной и преобразованной грамматики. Входные и выходные данные описывают грамматику в виде четверки $G = (N, \Sigma, P, S)$. В простейшем случае файл с исходными данными будет содержать (в указанном порядке):

1) Число нетерминалов $|N|$,
2) Нетерминалы N,
3) Число терминалов $|\Sigma|$,
4) Терминалы Σ,
5) Число правил вывода $|P|$,
6) Правила вывода P,
7) Начальный символ грамматики (или аксиому) S.

Выходные данные описывают (как правило) преобразованную грамматику $G_1 = (N_1, \Sigma_1, P_1, S_1)$. В простейшем случае файл с выходными данными будет содержать (в указанном порядке):

1) Число нетерминалов $|N_1|$,
2) Нетерминалы N_1,
3) Число терминалов $|\Sigma_1|$,
4) Терминалы Σ_1,
5) Число правил вывода $|P_1|$,
6) Правила вывода P_1,
7) Начальный символ грамматики (или аксиому) S_1.

Пример. Рассмотрим грамматику $G_0 = (\{E, T, F\}, \{"+", "*", "(", ")", "a"\}, P, E)$, где P состоит из правил:

$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to a \mid (E)$

При сделанных предположениях файл исходных данных для этой грамматики примет вид:

```
3
E T F
5
+ * ( ) a
6
E → E + T
E → T
T → T * F
T → F
F → a
F → (E)
E
```

После устранения левой рекурсии и левой факторизации получим преобразованную грамматику $G_1 = (\{E, E_1, T, T_1, F\}, \{"+", "*", "(", ")", "a"\}, P_1, E)$, где P_1 состоит из правил:

$E \to TE_1$
$E_1 \to +TE_1 \mid \varepsilon$
$T \to FT_1$
$T_1 \to *FT_1 \mid \varepsilon$
$F \to a \mid (E)$

Файл выходных данных для преобразованной грамматики примет вид:

```
5
E E1 T T1 F
5
+ * ( ) a
8
E → T E1
```

```
E1 → + T E1
E1 → ε
T → F T1
T1 → *F T1
T1 → ε
F → a
F → ( E )
E
```

Так как ε это греческая буква, то для нее надо ввести специальное обозначение.

Как входным, так и выходным данным можно придать структуру, которая упростит машинную обработку. Одним из вариантов такой структуры может быть XML-формат. Например, XML-формат используется для представления грамматики в генераторе компиляторов YAPP. Для грамматики *G*0 можно предложить такой формат:

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar name="G0">
  <terminalsymbols>
    <term name="IDENT" spell="a" />
    <term name="ADD" spell="+" />
    <term name="MUL" spell="*" />
    <term name="LPAREN" spell="(" />
    <term name="RPAREN" spell=")" />
  </terminalsymbols>
  <nonterminalsymbols>
    <nonterm name="E" />
    <nonterm name="T" />
    <nonterm name="F" />
  </nonterminalsymbols>
  <productions>
    <production>
      <lhs name="E" />
      <rhs>
        <symbol type="nonterm" name="E" />
        <symbol type="term" name="ADD" />
        <symbol type="nonterm" name="T" />
      </rhs>
    </production>
    <production>
      <lhs name="E" />
      <rhs>
        <symbol type="nonterm" name="T" />
      </rhs>
    </production>
    <production>
      <lhs name="T" />
      <rhs>
        <symbol type="nonterm" name="T" />
        <symbol type="term" name="MUL" />
        <symbol type="nonterm" name="F" />
      </rhs>
    </production>
    <production>
      <lhs name="T" />
      <rhs>
        <symbol type="nonterm" name="F" />
      </rhs>
    </production>
    <production>
      <lhs name="F" />
      <rhs>
        <symbol type="nonterm" name="IDENT" />
      </rhs>
    </production>
    <production>
      <lhs name="F" />
      <rhs>
        <symbol type="term" name="LPAREN" />
```

```
        <symbol type="nonterm" name="E" />
        <symbol type="term" name="RPAREN" />
      </rhs>
    </production>
  </productions>
  <startsymbol name="E" />
</grammar>
```

В порядке личной инициативы можно использовать и другие форматы представления грамматики, например, JSON-формат. Online-сервис, приведенный по адресу http://www.utilities-online.info/xmltojson/#.VrjOwfmLRD8, для ранее рассмотренной грамматики G_0 дает такое представление:

```
{
  "grammar": {
    "-name": "G0",
    "terminalsymbols": {
      "term": [
        {
          "-name": "IDENT",
          "-spell": "a"
        },
        {
          "-name": "ADD",
          "-spell": "+"
        },
        {
          "-name": "MUL",
          "-spell": "*"
        },
        {
          "-name": "LPAREN",
          "-spell": "("
        },
        {
          "-name": "RPAREN",
          "-spell": ")"
        }
      ]
    },
    "nonterminalsymbols": {
      "nonterm": [
        { "-name": "E" },
        { "-name": "T" },
        { "-name": "F" }
      ]
    },
    "productions": {
      "production": [
        {
          "lhs": { "-name": "E" },
          "rhs": {
            "symbol": [
              {
                "-type": "nonterm",
                "-name": "E"
              },
              {
                "-type": "term",
                "-name": "ADD"
              },
              {
                "-type": "nonterm",
                "-name": "T"
              }
            ]
          }
        },
        {
          "lhs": { "-name": "E" },
          "rhs": {
```

```
            "symbol": {
              "-type": "nonterm",
              "-name": "T"
            }
          }
        },
        {
          "lhs": { "-name": "T" },
          "rhs": {
            "symbol": [
              {
                "-type": "nonterm",
                "-name": "T"
              },
              {
                "-type": "term",
                "-name": "MUL"
              },
              {
                "-type": "nonterm",
                "-name": "F"
              }
            ]
          }
        },
        {
          "lhs": { "-name": "T" },
          "rhs": {
            "symbol": {
              "-type": "nonterm",
              "-name": "F"
            }
          }
        },
        {
          "lhs": { "-name": "F" },
          "rhs": {
            "symbol": {
              "-type": "nonterm",
              "-name": "IDENT"
            }
          }
        },
        {
          "lhs": { "-name": "F" },
          "rhs": {
            "symbol": [
              {
                "-type": "term",
                "-name": "LPAREN"
              },
              {
                "-type": "nonterm",
                "-name": "E"
              },
              {
                "-type": "term",
                "-name": "RPAREN"
              }
            ]
          }
        }
      ]
    },
    "startsymbol": { "-name": "E" }
  }
}
```

5. Варианты заданий на лабораторную работу

Общий вариант для всех: Устранение левой рекурсии.

Определение. Нетерминал A КС-грамматики $G = (N, \Sigma, P, S)$ называется рекурсивным, если $A =>+ \alpha A\beta$ для некоторых α и β. Если $\alpha = \varepsilon$, то A называется леворекурсивным. Аналогично, если $\beta = \varepsilon$, то A называется праворекурсивным. Грамматика, имеющая хотя бы один леворекурсивный нетерминал, называется леворекурсивной. Аналогично определяется праворекурсивная грамматика. Грамматика, в которой все нетерминалы, кроме, быть может, начального символа, рекурсивные, называется рекурсивной.

Некоторые из алгоритмов разбора не могут работать с леворекурсивными грамматиками. Можно показать, что каждый КС-язык определяется хотя бы одной не леворекурсивной грамматикой.

Постройте программу, которая в качестве входа принимает приведенную КС-грамматику $G = (N, \Sigma, P, S)$ и преобразует ее в эквивалентную КС-грамматику G' без левой рекурсии.

Указания.

1) Проработать самостоятельно п. 4.3.3. и п. 4.3.4. [2].
2) Воспользоваться алгоритмом 2.13. При тестировании воспользоваться примером 2.27. [1].
3) Воспользоваться алгоритмами 4.8 и 4.10. При тестировании воспользоваться примерами 4.7., 4.9. и 4.11. [2].
4) Устранять надо не только непосредственную (immediate), но и косвенную (indirect) рекурсию. Этот вопрос подробно затронут в [4].
5) После устранения левой рекурсии можно применить левую факторизацию.

Вариант 1. Устранение недостижимых символов.

Определение. Символ $X \in N \cup \Sigma$ назовем недостижимым в КС-грамматике $G = (N, \Sigma, P, S)$, если X не появляется ни в одной выводимой цепочке.

Постройте программу, которая в качестве входа принимает произвольную КС-грамматику $G = (N, \Sigma, P, S)$ и преобразует ее в эквивалентную КС-грамматику $G' = (N', \Sigma', P', S')$, не содержащую недостижимых символов.

Указания. Воспользоваться алгоритмом 2.8. [1].

Вариант 2. Устранение бесполезных символов.

Определение. Назовем символ $X \in N \cup \Sigma$ бесполезным в КС-грамматике $G = (N, \Sigma, P, S)$, если в ней нет вывода вида $S =>^* wXy =>^* wxy$, где w, x, y принадлежат Σ^*.

Чтобы установить, бесполезен ли нетерминал A, надо построить сначала алгоритм, выясняющий, может ли нетерминал порождать какие-нибудь терминальные цепочки, т. е. решающий проблему пустоты множества $\{w \mid A =>^* w, w \in \Sigma^*\}$.

Постройте программу, которая в качестве входа принимает произвольную КС-грамматику $G = (N, \Sigma, P, S)$ и преобразует ее в эквивалентную КС-грамматику $G' = (N', \Sigma', P', S')$, не содержащую бесполезных символов.

Указания. Воспользоваться алгоритмом 2.9. [1]. При тестировании воспользоваться примером 2.22. и упражнением 2.4.6. [1].

Вариант 3. Преобразование в грамматику без ε-правил.

Определение. Назовем КС-грамматику $G = (N, \Sigma, P, S)$ грамматикой без ε-правил (или неукорачивающей), если либо

1. P не содержит ε-правил, либо
2. есть точно одно ε-правило $S -> \varepsilon$ и S не встречается в правых частях остальных правил из P.

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

Указания. Воспользоваться алгоритмом 2.10. [1]. При тестировании воспользоваться примером 2.23. и упражнением 2.4.11. [1].

Вариант 4. Устранение цепных правил.

Определение. Правила вида A –> B, где $A \in$ N и $B \in$ N, будем называть цепными.

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

Указания. Воспользоваться алгоритмом 2.11. [1]. При тестировании воспользоваться примером 2.24. [1].

Вариант 5. Преобразование к приведенной грамматике.

Определение. КС-грамматика G = (N, Σ, P, S) называется грамматикой *без циклов*, если в ней нет выводов A –>+ A для $A \in$ N. Грамматика G называется *приведенной*, если она без циклов, без ε-правил и без бесполезных символов.

Грамматики с ε-правилами или циклами иногда труднее анализировать, чем грамматики без ε-правил и циклов. Кроме того, в любой практической ситуации бесполезные символы без необходимости увеличивают объем анализатора. Поэтому для некоторых алгоритмов синтаксического анализа, обсуждаемых в курсе, мы будем требовать, чтобы грамматики, фигурирующие в них, были приведенными.

Постройте программу, которая в качестве входа принимает произвольную КС-грамматику и преобразует ее в эквивалентную приведенную КС-грамматику.

Указания. Воспользоваться определением на стр. 175, алгоритмом 2.9. и алгоритмом 2.10. [1]. При тестировании воспользоваться упражнением 2.4.13. [1].

Вариант 6. Преобразование к нормальной форме Хомского.

Определение. КС-грамматика G = (N, Σ, P, S) называется грамматикой в *нормальной форме Хомского* (или в *бинарной нормальной форме*), если каждое правило из P имеет один из следующих видов:

1. A –> BC, где A, B и C принадлежат N,
2. A –> a, где $a \in$ Σ,
3. S – > ε, если ε $\in L(G)$, причем S не встречается в правых частях правил.

Можно показать, что каждый КС-язык порождается грамматикой в нормальной форме Хомского. Этот результат полезен в случаях, когда требуется простая форма представления КС-языка.

Постройте программу, которая в качестве входа принимает приведенную КС-грамматику G = (N, Σ, P, S) и преобразует ее в эквивалентную КС-грамматику G' в нормальной форме Хомского.

Указания. Воспользоваться алгоритмом 2.12. [1]. При тестировании воспользоваться примером 2.26. и упражнением 2.4.16. [1].

Вариант 7. Преобразование к нормальной форме Грейбах 1.

Определение. КС-грамматика G = (N, Σ, P, S) называется грамматикой в нормальной форме Грейбах, если в ней нет ε-правил и каждое правило из P, отличное от S –> ε, имеет вид A –> $a\alpha$, где $a \in$ Σ и $\alpha \in$ N* .

Постройте программу, которая в качестве входа принимает не леворекурсивную приведенную КС-грамматику G = (N, Σ, P, S) и преобразует ее в эквивалентную КС-грамматику G' в нормальной форме Грейбах.

Указания. Воспользоваться алгоритмом 2.14. При тестировании воспользоваться примером 2.29. [1]. и упражнением 2.4.19.[1].

Вариант 8. Преобразование к нормальной форме Грейбах 2.

Определение. КС-грамматика $G = (N, \Sigma, P, S)$ называется грамматикой в нормальной форме Грейбах, если в ней нет ε-правил и каждое правило из P, отличное от $S \to \varepsilon$, имеет вид $A \to a\alpha$, где $a \in \Sigma$ и $\alpha \in N^*$.

Постройте программу, которая в качестве входа принимает приведенную КС-грамматику $G = (N, \Sigma, P, S)$ без правил вида $S \to \varepsilon$ и преобразует ее в эквивалентную КС-грамматику $G' = (N', \Sigma, P', S)$ в нормальной форме Грейбах.

Указания. Воспользоваться алгоритмом 2.15. [1]. При тестировании воспользоваться примером 2.30. и упражнением 2.4.19. [1].

6. Порядок выполнения работы

1) Ознакомиться с основными понятиями и определениями по рекомендуемым в п. 2 материалам.
2) Проработать главу 3 учебного пособия [3].
3) Изучить алгоритм устранения левой рекурсии по работе [2] и [4] и на его основе разработать, тестировать и отладить программу устранения левой рекурсии. Примеры устранения левой рекурсии взять из работы [4].
4) Изучить алгоритмы, рекомендованные в п. 4 для предложенного варианта задания на лабораторную работу.
5) Разработать, тестировать и отладить программу преобразования грамматики для предложенного варианта задания на лабораторную работу.
6) Подготовить отчет о проделанной работе.
7) Подготовить ответы на контрольные вопросы.

7. Требования к отчету

Отчет по лабораторной работе выполняется в электронном виде и должен включать:

1) Идентификатор группы, имя и фамилию студента, дату выполнения работы.
2) Название лабораторной работы.
3) Описание задания – постановку задач, подлежащих выполнению в процессе лабораторной работы.
4) Текст программы, в которой решаются поставленные задачи.
5) Набор тестов и ожидаемые результаты для проверки правильности программы.
6) Результаты выполнения программы.
7) Анализ результатов и краткие выводы по работе.
8) Список дополнительной использованной литературы или дополнительных использованных электронных ресурсов.

8. Контрольные вопросы

1) Как может быть определён формальный язык?
2) Какими характеристиками определяется грамматика?
3) Дайте описания грамматик по иерархии Хомского.
4) Какие абстрактные устройства используются для разбора грамматик?
5) Оцените временную и емкостную сложность предложенного вам алгоритма.

9. Рекомендуемая литература

1. АХО А., УЛЬМАН Дж. Теория синтаксического анализа, перевода и компиляции: В 2-х томах. Т.1.: Синтаксический анализ. - М.: Мир, 1978.
2. АХО А.В, ЛАМ М.С., СЕТИ Р., УЛЬМАН Дж.Д. Компиляторы: принципы, технологии и инструменты. – М.: Вильямс, 2008.
3. БУНИНА Е.И., ГОЛУБКОВ А.Ю. Формальные языки и грамматики. Учебное пособие. – М.: Изд-во МГТУ им. Н.Э.Баумана, Москва, 2006. URL: http://iu9.bmstu.ru/data/book/fl.pdf

4. Eliminating left-recursion: three steps. URL: http://www.d.umn.edu/~hudson/5641/l11m.pdf

Приложение

Соглашения об обозначениях

1. Терминалы обозначаются
   - Строчными буквами английского алфавита (с начала), выделенными курсивом *a, b, c, d,* …
   - Символами операций +, -, *, ...
   - Символами пунктуации (,), ,, ...
   - Арабскими цифрами 0, 1, ..., 9
   - Именами, выделенными жирным шрифтом **id, num,** ...
2. Нетерминалы обозначаются
   - Заглавными буквами английского алфавита (с начала), выделенными курсивом *A, B, C*, ...
   - *S* – начальный символ грамматики
   - Именами, выделенными курсивом и/или заключенными в угловые скобки *expr*, *stmt*, <выражение>, <утверждение>, ...
3. Грамматические символы (как терминалы, так и нетерминалы) обозначаются заглавными буквами английского алфавита (с конца), выделенными курсивом *U, V, ..., Z*
4. Цепочки терминалов обозначаются строчными буквами английского алфавита (с конца), выделенными курсивом *t, u, v, w, x, y, z*
5. Цепочки грамматических символов обозначаются строчными буквами греческого алфавита (с начала) α, β, γ, ...
6. Обозначения для правила вывода
   - Правило вывода (правило подстановки, продукция или просто правило) обозначается так $A \rightarrow \alpha$
   - Если

     $A \rightarrow \alpha_1$

     $A \rightarrow \alpha_2$

     …

     $A \rightarrow \alpha_k$

     правила для *A* в левой части (*A*-правила), то удобно пользоваться сокращенной записью

     $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_k$

     где $\alpha_1, \alpha_2, \dots, \alpha_k$ – альтернативы для *A*.
   - Первое правило содержит начальный символ грамматики в левой части
7. Обозначения для отношения вывода
   - Отношение вывода обозначается так $\varphi \Rightarrow \psi$ и читается: ψ непосредственно выводима из φ
   - Транзитивное замыкание отношения $\Rightarrow$ обозначается так $\varphi \Rightarrow^+ \psi$ и читается: ψ выводима из φ нетривиальным образом
   - Транзитивно-рефлексивное замыкание отношения $\Rightarrow$ обозначается так $\varphi \Rightarrow^* \psi$ и читается: ψ выводима из φ
   - Через $\Rightarrow^k$ обозначается *k*-я степень отношения $\Rightarrow$
8. Обозначения для такта автомата
   - Такт автомата представляется бинарным отношением $\vdash$
   - Отношения $\vdash^+$ и $\vdash^*$ являются соответственно транзитивным и рефлексивно-транзитивным замыканием отношения $\vdash$
   - Через $\vdash^k$ обозначается *k*-я степень отношения $\vdash$
9. Обозначения для алфавитов

   Алфавиты обычно будут обозначаться прописными греческими буквами, например, так N, Σ, Δ.

Перечисленные соглашения распространяются также и на буквы и имена с нижними и верхними индексами. Мы не будем напоминать об этих соглашениях, когда рассматриваемые символы им удовлетворяют.