Санкт-Петербургский Политехнический университет
Факультет Технической кибернетики

Ю.Г.Карпов

Задачи по курсу

“Автоматы и формальные языки”

Санкт-Петербург, 2013

Содержание

Предисловие

Сборник включает основные типовые задачи по курсу “*Автоматы и формальные языки*”, который читается на факультете технической кибернетики Санкт-Петербургского Политехнического университета. Задачи разбиты по разделам. Рекомендуется решать задачи каждого раздела сразу после прослушивания соответствующей лекции.

Задачи, требующие размышления или некоторого времени для их решения, помечены звездочкой (*). Двумя звездочками помечены задачи, которые представляют собой программный проект.

Освоение методов и приемов решения задач по построению грамматик формальных языков, по использованию методов синтаксического анализа и семантических определений, так же, как и освоение практических умений построения трансляторов являются главными целями курса.

По курсу проводится письменный экзамен, который состоит в решении задач. Экзаменационные варианты составляются из задач данного сборника, возможно, с измененными данными. Поэтому умение самостоятельно решать задачи сборника гарантирует студенту положительную оценку на экзамене.

1. Конечные автоматы - распознаватели формальных языков

1.1. Детерминированные конечные автоматы-распознаватели

1.1.1. Формальное определение того, что состояние s конечного автомата $A=(S, \Sigma, s_0, \delta, F)$ является достижимым, записывается так:

$$\text{Достижимо}(s) \Leftrightarrow (\exists \alpha \in \Sigma^*)\ [\delta^*(s_0, \alpha)=s],$$

что словами можно сказать так:

> *Состояние s достижимо в конечном автомате* $A=(S, \Sigma, s_0, \delta, F)$ *тогда и только тогда, когда существует цепочка символов над входным словарем* Σ *автомата, под воздействием которой автомат из начального состояния переходит в s.*

Запишите формальное определение и словесную формулировку определения недостижимого состояния в конечном автомате.

1.1.2. Формальное определение того, что состояние s конечного автомата $A=(S, \Sigma, s_0, \delta, F)$ является кодостижимым, записывается так:

$$\text{Кодостижимо}(s) \Leftrightarrow (\exists \alpha \in \Sigma^*)\ [\delta^*(s, \alpha) \in F\ ],$$

что словами можно сказать так:

> *Состояние s кодостижимо в конечном автомате* $A=(S, \Sigma, s_0, \delta, F)$ *тогда и только тогда, когда существует цепочка символов над входным словарем* Σ *автомата, под воздействием которой автомат из состояния s переходит в одно из допускающих состояний.*

Запишите формальное определение и словесную формулировку определения не кодостижимого состояния в конечном автомате.

1.1.3. Формальное определение того, что язык L над словарем Σ допускается конечным автоматом $A=(S, \Sigma, s_0, \delta, F)$, записывается так:

$$L \subseteq \Sigma^* \textit{ допускается } A \Leftrightarrow (\forall \alpha \in \Sigma^*)\ [(\alpha \in L) \equiv \delta^*(s_0, \alpha) \in F],$$

что словами можно сказать так:

> *Язык L над словарем* Σ *допускается конечным автоматом A тогда и только тогда, когда любая цепочка символов этого словаря принадлежит языку L если и только если она переводит автомат A в одно из допускающих состояний.*

Запишите формально и словесно определение того, что язык $L \subseteq \Sigma^*$ не допускается конечным автоматом $A=(S, \Sigma, s_0, \delta, F)$.

1.1.4. Два конечных автомата-распознавателя $A=(S_A, \Sigma, s_{0A}, \delta_A, F_A)$ и $B=(S_B, \Sigma, s_{0B}, \delta_B, F_B)$ с одним и тем же входным алфавитом Σ эквивалентны, если и только если они распознают один и тот же язык, т.е. любая входная цепочка приводит автомат A в допускающее состояние тогда и только тогда, когда она приводит в допускающее состояние и автомат B. Формально это записывается так:

$$A \approx B \equiv (\forall \alpha \in \Sigma^*)\ [\delta^*_A(s_{0A}, \alpha) \in F_A \Leftrightarrow \delta^*_B(s_{0B}, \alpha) \in F_B\ ]$$

Определите формально, когда два конечных автомата-распознавателя не эквивалентны, и дайте словесную формулировку этого.

1.1.5. Два состояния p и q конечного автомата-распознавателя $A=(S, \Sigma, s_0, \delta, F)$ эквивалентны, если и только если любая входная цепочка приводит автомат A в допускающее состояние из p тогда и только тогда, когда она приводит его в допускающее состояние из q. Формально это записывается так:

$$q \approx r \equiv (\forall \alpha \in \Sigma^*) [\ \delta^*(p, \alpha) \in F \Leftrightarrow \delta^*(q, \alpha) \in F\ ].$$

Определите формально, когда два состояния конечного автомата-распознавателя не эквивалентны, и дайте словесную формулировку этого.

1.1.6. Запишите формальное определение того, что $L(A) = L(B) \cap L(C)$, т.е. что конечный автомат $A=(S, \Sigma, s_0, \delta, F)$ допускает пересечение двух автоматных языков, которые заданы распознающими их конечными автоматами $B=(S_B, \Sigma, s_{0B}, \delta_B, F_B)$ и $C=(S_C, \Sigma, s_{0C}, \delta_C, F_C)$ над тем же конечным словарем Σ.

1.1.7. Постройте конечный автомат с входным алфавитом $\Sigma=\{a, b\}$, распознающий:

а) пустой язык;

б) язык, содержащий только одну пустую цепочку ε;

в) язык Σ^*, состоящий из всех возможных цепочек в словаре $\Sigma=\{a, b\}$;

г) язык Σ^+, состоящий из всех непустых цепочек в словаре $\Sigma=\{a, b\}$.

1.1.8. Постройте конечный автомат, который распознает все цепочки из символов 0 и 1, которые представляют собой двоичные числа, делящиеся на 5. Примеры таких цепочек: 011001, 1010.

1.1.9. Постройте детерминированный конечный автомат с входным алфавитом {0, 1}, распознающий:

а) все цепочки, кончающиеся кодом 11010.

б) все цепочки, кроме тех цепочек, которые кончаются кодом 11010.

в) все цепочки, в которых не встречается трех единиц подряд.

г) все цепочки, кроме тех, которые включают код 11010 в качестве подцепочки.

1.1.10. Постройте детерминированный конечный автомат с входным алфавитом {a, b, c}, распознающий:

а) все цепочки, кроме тех, которые включают код *abcac* в качестве подцепочки

б) все цепочки, в которые входит четное число символов *a* и нечетное число *b*.

с) все цепочки, в которых перед каждым символом *a* и после каждого *a* стоит символ *b*.

1.1.11. Пусть дан конечный автомат-распознаватель $A=(S, \Sigma, \delta, F)$, где:

$S=\{s0, s1, s2, s3, s4\}$;

$\Sigma=\{a, b, c\}$;

$\delta=\{(s0,a,s1), (s0,b,s3), (s0,c,s3), (s1,a,s1), (s1,b,s2), (s1,c,s3), (s2,a,s1), (s2,b,s3), (s2,c,s4), (s3,a,s1),(s3,b,s3), (s3,c,s3), (s4,a,s1), (s4,b,s3), (s4,c,s3)\}$;

$F=\{s4\}$.

Минимизируйте конечный автомат А с помощью треугольной матрицы. Какой язык определяет этот автомат?

1.1.12. Поведение дискретных событийных систем можно описать цепочками событий, которые случаются в процессе их функционирования. Пусть поезд циркулирует по маршруту, включающему тоннель. Динамику движения поезда можно абстрактно описать автоматом с тремя состояниями: находится вне тоннеля, ожидает разрешения входа в тоннель, находится внутри тоннеля. События, вызывающие переход из состояния в состояние: *a* (*arrive*) - прибытие к тоннелю, *e* - (*enter*) вход в тоннель, *l* (*leave*) – выход из тоннеля. Последовательность событий при движении поезда: *aelaelaelae*... . Возможные цепочки при движении двух поездов, двигающихся в разных направлениях, могут быть корректными и некорректными. Например, корректная цепочка $a_1a_2e_1l_1e_2l_2a_2e_2a_1l_2e_1$ ... - попеременное использование тоннеля. Примеры некорректных поведений:

i) $a_1a_2e_1l_1a_1e_2e_1$. – авария, столкновение в тоннеле,

ii) $a_1a_2e_1l_1a_1e_1l_1a_1e_1l_1a_1e_1l_1$ ... бесконечное ожидание поездом 2 у входа в тоннель, который постоянно занимается встречными поездами.

Постройте примеры цепочек, описывающих правильные поведения:

а) взаимное исключение использования поездами общего ресурса – тоннеля.

б) поочередное использование поездами общего ресурса.

в) “справедливое” использование ресурса (если поезд ожидает перед тоннелем, в который вошел другой поезд, то первый войдет в тоннель до того, как второй займет его в следующий раз).

г) “приоритетное право” одного из направлений пользования ресурсом (поезд на приоритетном направлении не ожидает, если тоннель при его прибытии не занят).

1.1.13. Следующая сформулированная Э.Дейкстрой задача “обедающие философы” является классической задачей параллельного программирования, проясняющая с помощью красивой метафоры суть проблем, возникающих в системе параллельных процессов с общими ресурсами. Задача состоит в следующем. Пять философов обдумывают философские проблемы, сидя за круглым столом, и время от времени едят. Число философов N с номерами 0, ... N-1. Перед каждым философом стоит тарелка со спагетти, между каждой парой философов лежит вилка. Эта вилка является левой для одного философа и правой для другого. Спагетти можно есть только двумя вилками. Проголодавшись, каждый философ берет сначала левую от себя вилку (если она свободна), затем правую вилку, ест, после чего кладет обе вилки. События для i-го философа такие:

iTi - i-й философ взял i-ю вилку (слева от него);

iT(i+1) – i-й философ взял (i+1)-ю вилку (сложение здесь по модулю N);

iR – i-й философ положил обе свои вилки.

а) Постройте два конечных автомата, моделирующих поведение только двух философов за столом с номерами 0 и 1.

б) Постройте два конечных автомата, определяющих состояния двух вилок с номерами 0 и 1.

в) Проверьте, существует ли опасность дедлока в системе из двух философов и двух вилок, построив синхронную композицию процессов.

г) Проанализируйте проблему двух обедающих философов при следующих правилах поведения за столом: проголодавшись, философ берет *любую* свободную вилку, а затем другую вилку, если она свободна.

1.1.14. Проверьте эквивалентность двух заданных конечных автоматов-распознавателей, построив их синхронную композицию.

1.1.15. По заданному конечному автомату-распознавателю постройте эквивалентный минимальный конечный автомат.

1.1.16. Формальный язык L называется префиксно-замкнутым, если с каждой цепочкой языка L содержит и все префиксы этой цепочки (т.е. начальные части цепочки). По любому языку L можно построить префиксно-замкнутый язык, который обозначается Pref(L).

а) Постройте префиксный язык для языка, заданного регулярным выражением ab*(a+b).

б) Является ли префиксно-замкнутым язык, заданный регулярным выражением b*(a+b)*?

в) Постройте детерминированный конечный автомат, который распознает язык Pref(L), если L задан регулярным выражением (a+b)*ab*.

г) Если L – автоматный, то будет ли Pref(L) автоматным?

д) Языки L1 и L2 называются конфликтующими, если

$$\text{Pref}(L1) \cap \text{Pref}(L2) \neq \text{Pref}(L1 \cap L2).$$

Приведите примеры конфликтующих и неконфликтующих языков.

1.1.17. В качестве адреса электронной почты нельзя выбрать произвольную цепочку символов. Такие адреса определяются следующим образом. До символа ‘@’ может идти любая последовательность, состоящая из непустых групп - латинских букв и цифр, разделенных точками. После символа ‘@’ могут идти любые непустые последовательности латинских букв и цифр, разделенных точками. Постройте конечный автомат, распознающий правильную структуру адреса электронной почты.

1.1.18. Процессор обрабатывает поток работ, имея на входе очередь, которая может хранить не более одной работы. Если пришедшая работа нашла очередь непустой, она отбрасывается. Работы во входном потоке могут быть двух приоритетов: 1 и 2. Приоритет 2 работ более высокий. Если работа нашла систему занятой (и очередь, и процессор заняты), эта работа отбрасывается, не попадая в систему. Более приоритетная работа не может ждать менее приоритетную: если пришла приоритетная работа, очередь пуста, а работа более низкого приоритета обрабатывается, то приоритетная работа выталкивает менее приоритетную в специальный буфер. Если процессор завершил работу, и его не ожидает в очереди работа приоритета 2, то он сначала продолжит обрабатывать работу, находящуюся в буфере. Постройте граф переходов для системы очередь-процессор-буфер с событиями: пришла работа 1 приоритета, пришла работа 2 приоритета, процессор закончил обработку очередной работы.

1.1.19. Два процессора обрабатывают поток работ, имея на входе очередь ожидающих работ, которая может хранить не более одной работы. Обработку работы начинает процессор 1, если он свободен. Когда процессор 1 завершает обработку, он может передать работу для

дальнейшей обработки, только если процессор 2 свободен. Если пришедшая работа нашла очередь непустой, она отбрасывается.
Постройте граф переходов для системы очередь-процессор1- процессор2.

1.1.20. Два процессора, P1 и P2 последовательно обрабатывают поток работ, имея на входе буфер B1, который может хранить не более одной работы. Между процессорами P1 и P2 есть буфер B2, который также может хранить не более одной работы. Обработку начинает процессор P1. Когда P1 завершает обработку, он может передать работу для дальнейшей обработки, только если буфер B2 свободен. Если пришедшая работа нашла входной буфер B1 непустым, она отбрасывается. Завершение обработки процессора при наличии во входном буфере работы сразу приводит к загрузке этой работы процессором для обработки. Постройте граф переходов системы B1||P1||B2||P2.

1.1.21. Три процессора, P1, P2 и P3 обрабатывают поток работ, имея на входе буфер B, который может хранить не более одной работы. Обработку работ начинает процессор P1, затем работа обрабатывается либо P2, либо P3. Когда P1 завершает обработку, он может передать ее для дальнейшей обработки либо процессору P2, либо процессору P3, только если эти процессоры свободны. Если пришедшая работа нашла входной буфер B непустым, она отбрасывается. Завершение работы процессора P1 при наличии в буфере B работы сразу приводит к загрузке этой работы процессором для обработки. Постройте граф переходов системы B||P1||P2||P3.

1.1.22. Могут ли два эквивалентных состояния конечного автомата-распознавателя быть:
а) оба допускающими?
б) оба недопускающими?
в) одно допускающим, а другое недопускающим?
Ответ обоснуйте.

1.1.23. Комбинации, открывающие сейф, набираются из алфавита {a, b, c}, каждая состоит из четырех символов и начинается и заканчивающаяся несовпадающими символами. Если набран неверный код, то система поднимает тревогу. Постройте алгоритм работы встроенной системы управления замком сейфа (на паскалеподобном языке).

1.2. Лемма о накачке

1.2.1. Лемма о накачке формально выражается следующим образом.

Лемма 1. Пусть L – язык над алфавитом Σ.

Если L - *автоматный,* то:

$(\exists n\in N)(\forall \alpha\in L: |\alpha|\geq n)(\exists u,v,w\in\Sigma^*)\ [\alpha=uvw \wedge |uv|\leq n \wedge |v|\geq 1 \wedge\ (\forall i\in N\cup\{0\})\ uv^iw\in L\ ]$.

Сформулируйте лемму 1 словесно по этому выражению логики предикатов.

1.2.2. Другая формулировка леммы о накачке, которую удобнее применять, чтобы доказать неавтоматность заданного языка, формально выражается так.

Лемма 2. Пусть L – язык над алфавитом Σ.

Если:

$(\forall n \in N)(\exists \alpha \in L: |\alpha| \geq n)(\forall u,v,w \in \Sigma^*: \alpha = uvw \wedge |uv| \leq n \wedge |v| \geq 1): (\exists i \in N \cup \{0\})\ uv^i w \notin L$,

то L – *НЕавтоматный*.

Сформулируйте лемму 2 словесно по этому выражению логики предикатов.

1.2.3. Эквивалентны ли формулировки лемм 1.3.1 и.1.3.2, или одна из них является следствием другой? Докажите.

1.2.4. Докажите с помощью леммы о накачке, что язык правильных скобочных выражений не является автоматным.

1.2.5. Докажите, что язык над словарем $\{a, b\}$, содержащий все цепочки с одинаковым числом вхождений символов a и b, не автоматный.

1.2.6. Докажите, что язык над словарем $\{a, b\}$, содержащий все цепочки, в которых число вхождений символов a больше, чем число вхождений символов b, не автоматный.

1.2.7. Докажите с помощью леммы о накачке, что следующие языки не автоматные:

а) $\{a^{n^{\uparrow 2}} | \ n \geq 0\}$.

б) $\{ a^{2n} b^n | \ n \geq 1\}$.

в) $\{ww \mid w \in \{a, b\}^*\}$.

г) $\{w \in \{a, b\}^* \mid \ |w|_a = 2|w|_b\}$.

1.3. Недетерминированные и детерминированные конечно-автоматные распознаватели

1.3.1. Постройте детерминированный конечный автомат, эквивалентный заданному недетерминированному конечному автомату

1.3.2. Постройте детерминированный конечный автомат с входным алфавитом $\{a, b, c\}$, распознающий все цепочки:

а) точно с одним вхождением символа b.

б) не более чем с тремя вхождениями символа b.

в) у которых третья буква от конца b.

г) не менее чем с двумя вхождениями символа b.

1.3.3. Постройте детерминированный конечный автомат, распознающий все такие цепочки над словарем $\{a, b, c\}$, которые начинаются и заканчиваются одинаковыми символами.
Указание. Проще сначала построить недетерминированный КА, обладающий этим свойством.

1.3.4. Постройте детерминированный конечный автомат, распознающий все такие цепочки над словарем $\{a, b, c\}$, которые начинаются и заканчиваются различными символами.
Указание. Проще сначала построить недетерминированный КА, обладающий этим свойством.

1.3.5. Постройте детерминированный конечный автомат, распознающий все такие цепочки из цифр, которые представляют число, делящееся на 25. ***Указание.*** Проще сначала построить недетерминированный КА, обладающий этим свойством.

1.3.6. Постройте детерминированный конечный автомат, который допускает над словарем $\Sigma=\{a,b\}$:

а) пустой язык.
б) язык, содержащий только пустую цепочку.
в) язык $\Sigma^*\backslash L$, где L - некоторый заданный автоматный язык.
г) язык L^*, где L - некоторый заданный автоматный язык.
д) все цепочки, заканчивающиеся кодом *aabba*.
е) все цепочки, включающие строку *abbab*.
ж) все цепочки, в которых за каждым символом *a* непосредственно следует *b*.
з) с не менее чем одним вхождением символа *a* и точно с двумя вхождениями *b*.
и) с четным числом вхождений символов *a* и нечетным числом вхождений символов *b* и не менее чем с одним вхождением символа *c*.

1.3.7. Постройте детерминированный конечный автомат, распознающий все цепочки над словарем $\{a, b, c\}$, включающие не менее двух символов *a,* точно один символ *b* и не более двух символов *c*.

1.3.8. Постройте детерминированный конечный автомат, распознающий все такие цепочки над словарем $\{a, b, c\}$, у которых две последние буквы совпадают.

1.3.9. Постройте детерминированный конечный автомат со входным алфавитом $\Sigma=\{a, b\}$, распознающий все цепочки из Σ^* за исключением тех из них, которые содержат подцепочку *aab*.

1.3.10. Пусть L такой язык над словарем $\{a, b, c\}$, каждая цепочка которого заканчивается символом, не совпадающим с предыдущими символами этой цепочки. Будет ли L автоматным? Если да, то постройте детерминированный конечный автомат, распознающий L.

1.3.11. Обозначим RemoveOneSymbol(*w; a*), или сокращенно ROS(*w; a*), операцию по выбрасыванию из цепочки *w* каждого вхождения символа *a*. Например, ROS(*abac; a*) = *bc*, ROS(*bc; a*) = *bc*. Пусть ROS(*L; a*) = {ROS(*w; a*) | $w \in L$} для любого языка *L*. Докажите, утверждение: "*Если язык L –автоматный, то автоматным является и язык* ROS(*L; a*)".

1.3.12. Пусть L - автоматный язык над словарем $\Sigma=\{a, b, c\}$. Будет ли автоматным дополнение этого языка, т.е. язык $\Sigma\backslash L$? Если да, то по заданному конечному автомату, распознающему L, постройте конечный автомат, распознающий $\Sigma\backslash L$.

1.3.13. Пусть L1 и L2 - автоматные языки над словарем $\{a, b, c\}$. Будут ли автоматными:

а) $L1 \cup L2$;
б) $L1 \cap L2$;
в) $L1\backslash L2$.

Если да, то по двум заданным конечным автоматам A(L1) и A(L2) постройте конечные автоматы A(L1∪L2), A(L1∩L2) и A(L1\L2).

1.3.14. Постройте детерминированный конечный автомат, моделирующий алгоритм управления кодовым замком с символами {0, 1} и кнопкой “Открывай”. Автомат должен открывать дверь тогда и только тогда, когда кнопка “Открывай” нажата после любой набранной двоичной последовательности, которая заканчивается кодом 1010010. Заметьте, что кнопки “Сброс” в автомате не нужно: любая цепочка, заканчивающаяся на 1010010, является допустимой (это значит, что набор может происходить с любого состояния автомата).

1.3.15. Арифметические выражения с четырьмя арифметическими операциями {+, -, *, /}записаны без скобок в обычной инфиксной форме, постфиксной форме (польской инверсной записи), и в префиксной форме. Являются ли автоматными соответствующие языки? Если да, то постройте распознающие конечные автоматы.
Примеры цепочек для арифметического выражения *a+b*c/a-b* в постфиксной форме (ПОЛИЗ): “*abc*a/+b-*”, а в префиксной форме: “*-+a/*bcab*”.

1.4. Трансляция автоматных языков

1.4.1. Постройте транслятор простейшего языка присваиваний *Simple* в программу для стековой машины. Правой частью оператора присваивания могут быть либо операнд (целая константа, либо переменная), либо одна бинарная операция над парой операндов. Другими видами оператора являются операторы ввода и вывода значения операнда. Имя переменной задается одной буквой ‘*a*’ с целым индексом: *a*1, *a*2, и т.п. Символ * помечает начало и конец программы, символы пробела и перевода строки не учитывать. Пример программы на языке *Simple*:

```
*
a1:=ввод;
a2:=-54;
a3:=a2*a1;
вывод(a3)
*
```

1.4.2. Язык EXPR содержит операции присваивания переменным значений арифметических выражений без скобок с операциями {+, -, *, ÷}, а также операции вывода. Числа только целые. Выражение в операторе присваивания и операторе вывода имеет не более одной арифметической операции. При таких ограничениях язык – автоматный. Постройте транслятор с этого языка на язык стековой машины.

Пример программы на языке EXPR:

```
begin
    y:=3;
    x:=y÷5;
```

y:=7- x;
*z:=3*y;*
вывод(x+z).

end

1.4.3. Окно для указания номеров печатаемых страниц в программе Word позволяет выдавать эти номера через запятую. В качестве номеров страниц также может использоваться пара A-B, где A и B – натуральные числа, причем A не обязательно больше B (например: 2, 23, 43-32, 3-5). Пробелы являются разделителями наряду с запятыми. В конце цепочки стоит символ *eot* . Постройте транслятор языка, выдающий последовательные номера печатаемых страниц в том порядке, в котором указано.

1.4.4. Структура данных, задающая BDD двоичной функции, состоит из перечисления вершин бинарной диаграммы, например: {<5,x1,4,3>; <4,x2,2,0>; <3,x2,0,2>; <2,x3,1,0>}. Для каждой вершины задан ее уникальный номер, имя двоичной переменной, помечающей вершину, а также два номера вершин, с которыми данная вершина связана при значениях переменной 0 и 1. Постройте конечный автомат, распознающий все такие описания. Постройте транслятор из такого описания в программу вычисления значения функции по значениям переменных.

1.4.5. Постройте синтаксические диаграммы и семантические действия на них для трансляции коэффициентов системы линейных уравнений в списочную структуру матрицы значений коэффициентов и вектора значений правых частей. Пример цепочки языка описания системы уравнений:

Решить систему 4 уравнений:
-3.56 *a*[1] +15.0 *a*[4] = 23.64;
0.26 *a*[2] +25.6 *a*[3] -3.5 *a*[4] = 0.0;
13.36 *a*[2] +28.8 *a*[4] = -5.34;
5.2*a*[1]+23.88 *a*[3] = 234.5
методом Гаусса.

1.4.6*. Постройте транслятор языка представления чисел римской системы счисления. Проверьте работу транслятора на примерах римских чисел: MMIX = 2009, XLIII = 43, XC = 90, CD = 400, CMXCVII = 997.

1.4.7. На основе распознающего конечного автомата постройте программу, которая по последовательности целых, идущих по возрастанию, возвратит строку из набора непрерывных интервалов. Например, после обработки текста “1, 2, 3, 6, 7, 8, 10, 12, 13 eot” должна быть выдана строка “1-3; 6-8; 10; 12-13 eot ”.

1.4.8. Постройте синтаксическую диаграмму и семантические действия преобразования текстовых “смайликов” вида ‘:-)’ , ‘:-(‘, ‘:-))’, ‘:-((‘ в символы  ,  ,  ,  .

1.4.9. Постройте транслятор автоматного языка, задающего шахматную позицию - расстановку фигур на шахматной доске. Пример записи позиции:

Белые: Крg2, Фd4, Лe3, Сe5, пп. b2, d7, f2, g4, h3 (9).
Черные: Крg8, Фd8, Лb7, Kd3, пп. b5, c4, f7, g7, g5 (9).

1.4.10. Постройте транслятор автоматного языка, задающего решение шахматного этюда. Пример записи этюда: 1. Крd2 Кb2 2. Крс2 Крa5 3. Крb3 Крa5 4. Kd6! c4+ 5. K:c4+ Крb5 ... . Выход транслятора – команды по перемещению фигур.

2. Регулярные множества, регулярные выражения и автоматные языки

2.1. Регулярные множества и регулярные выражения

2.1.1. Пусть L_1={ac, bcc}, L_2={a, c, bc } – два множества цепочек. Постройте множества L_1L_2, $L_1^3 \cup L_2$, $L_1 \cap L_2^2$, L_1^*, L_2^+.

2.1.2. Постройте регулярное множество, определяемое регулярным выражением ab*+ba*.

2.1.3. Чему равны множества:

а) $\varnothing^*$; б) $\varnothing^+$;
в) $\{\varepsilon\}^*$; г) $\{\varepsilon\}^+$;
д) $\{a\}^*$; е) $\{a\}^+$.

2.1.4. Докажите, что ((ab)*)*= (ab)*, построив регулярные множества, задаваемые этими регулярными выражениями.

2.1.5. Определите, принадлежит ли слово 1010 языку, определенному регулярным выражением 10*(10+01)00*.

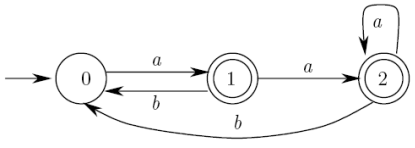
2.1.6. Постройте регулярное выражение, определяющее следующий язык над алфавитом {0, 1}. Язык содержит слова, в котором каждый ноль, если он вообще есть, с обеих сторон окружен единицами. Примеры слов языка: 11010111101, 10111.

2.2. Регулярные выражения и конечные автоматы

2.2.1. Постройте минимальные детерминированные конечные автоматы с входным алфавитом Σ={a, b}, распознающие языки, заданные регулярными выражениями:

а) a*b*; е) a*a*;
б) a*+b*; ж) (a*b*)*;
в) (a*+b*)*; з) (a*b*+b)*b;
г) (ba + bab)* и) ab*b*a*b;
д) (a+b)a*b*; к) (a+b)*(b*+a*).

2.2.2. По заданному конечному автомату–распознавателю постройте регулярное выражение, задающее тот же язык, который распознает автомат.



2.2.3. Язык над алфавитом {*a, b, c*} задан регулярным выражением *a*a* (*bc*)**a*. Принадлежат ли этому языку следующие слова: *aabcbca, aa, abc, a, aaaa*.

2.2.4. Постройте детерминированный конечный автомат, распознающий язык, заданный регулярными выражениями:

а) *a**(*a*+*b*)**a*.
б) (*a***c***a**)**b*(*c***a***c**)*.
в) (*a***a**.
г) *a***b***a**.

2.2.5. Постройте детерминированный конечный автомат с входным алфавитом Σ={*a, b*}, распознающий язык Σ* - L, где L – язык, заданный регулярным выражением *a***b***a*.

2.2.6. Все возможные последовательности событий обращения к ограниченному буферу (запись *w* и выборка *t*) представляют собой регулярный язык. Для буфера емкостью 1 этот язык определяется регулярным выражением (*wt*)*.
Постройте регулярные выражения, определяющие все правильные последовательности обращения к буферу емкостью 2 и 3.

2.2.7. Постройте регулярное выражение, описывающее язык L1-L2, если L1 задается регулярным выражением *ab***c**, а L2 задается регулярным выражением *a***bc*.

2.2.8. Постройте детерминированный конечный автомат, распознающий все цепочки словаря {*a, b, c*}

а) заданные регулярным выражением *b**(*a***ba*)*.
б) за исключением цепочек языка, заданного регулярным выражением *c**(*a*+*b*)**c*.
б) кроме тех, которые задаются регулярным выражением *a**(*ab**+*c**)*.
г) которые НЕ содержат ни одной подцепочки языка, задаваемого регулярным выражением *ab***c*.
д) которые содержат в качестве *подцепочки* какую-либо из цепочек языка, задаваемого регулярным выражением *ab***c*.
е) кроме тех, которые в качестве подцепочек включают слова, заданные регулярным выражением *a*(*b*+*c**)*.

2.2.9. Постройте регулярное выражение и детерминированный конечный автомат, задающие над алфавитом {0, 1}:

а) язык все слова которого начинаются с символа 0 и содержат, по крайней мере, два символа 1.
б) язык, все слова которого не содержат двух единиц подряд.

2.2.10. Постройте детерминированный конечный автомат, который распознает пересечение двух таких регулярных языков над словарем {*a, b, c*}, которые задаются регулярными выражениями R1=*a**(*b*+*c*)* и R2=*a* (*b**+*c**).

2.2.11. Совпадают ли регулярные языки, заданные регулярными выражениями *a**(*a**+*b*)**a* и *aa**(*a*+*b*)**a*?

2.2.12. Задайте все возможные цепочки десятичных цифр, которые представляют числа, делящиеся на 5:

а) недетерминированным конечным автоматом;

б) детерминированным конечным автоматом;

в) регулярным выражением.

2.2.13. Равны ли два регулярных выражения R1=ε+(*ab*+*a*)(*ab*+*a*)* и R2=(*a**ab*)**a**?

2.2.14. Опишите регулярным выражением язык, являющийся пересечением языков, задаваемых регулярными выражениями R1=*a***b** и R2=*b***a**.

2.2.15. Заданы два регулярных выражения R1=(ε+*b*)*b***a** и R2=(*bb*+*b***)(*ba***+*a***). Совпадают ли регулярные множества L(R1) и L(R2)?

2.2.16. Проверьте, будет ли L(R1) $\subseteq$ L(R2) для R1 = (ε+*b***+*ab*)**a* и R2 = *a*(*ab*+*b***). Каков общий алгоритм определения того, что ли один из регулярных языков является подъязыком другого?

2.2.17. Постройте регулярное выражение, описывающее все цепочки

а) внутри которых не встречается цепочка *abba*.

б) внутри которых не встречается цепочка *bbaa*.

Указание. Сначала лучше построить конечный автомат, допускающий все цепочки, внутри которых есть эта подцепочка *abba*, а потом построить автомат, допускающий дополнение языка, допускаемого первым автоматом, а по этому автомату построить регулярное выражение, описывающее язык, допускаемый автоматом.

2.2.18. Пусть L1 и L2 - два регулярных множества над словарем V. Будут ли регулярны множества: а) V*; б) V*-L1; в) V* - L1*; г) L1-L2; д) L1$\cup$L2; е) L1$\cap$L2? Обоснуйте свой ответ.

2.2.19. Все возможные последовательности событий обращения к ограниченному буферу (запись, p от put и выборка, t от take) представляют собой язык. Анализ статического кода параллельной программы, работающей с таким буфером, может выявить ошибку, если такая последовательность не принадлежит языку. Постройте регулярные выражения, определяющие все правильные и все неправильные последовательности обращения к буферу емкостью 2.

2.2.20. Все возможные правильные последовательности событий в параллельных программах, составляющие так называемый “response pattern” (шаблон отклика), когда два события S (send) и R (receive) идут друг за другом, описываются регулярным выражением (SS*RR*)*.

а) Постройте минимальный конечный автомат, который будет распознавать последовательности событий, соответствующие шаблону отклика.

б) Постройте минимальный конечный автомат, который будет распознавать последовательности событий, не соответствующие шаблону отклика.

в) Постройте минимальный конечный автомат, который будет распознавать последовательности событий, не соответствующие шаблону (DSS*RR*)*, где D – разделитель сессий.

2.2.21. Какие подстроки на основе регулярного выражения, записанного в расширенном синтаксисе php, будут выбраны в тексте?

	Регулярное выражение	Текст
а)	/ [^\s]*о*(*т*|*р*|б)+ *и?*/	“*отис, ботрис, оборотрис*”
б)	/ [^34] [0-9] {3} \w /	“*324b – 235s- 013*”
в)	/ [*щ ц*] (*и*|*е*)+ *п?*/	“*соответствующие цепочки* ”
г)	/ \w [\w\d]* \+? /	“*ab231+ 2 = fotka25*”

2.2.22. Какое регулярное выражение позволит выбрать из произвольного текста все адреса электронной почты (см.задачу 1.1.11)?

2.2.23. Докажите, что ((*ab*)*)*= (*ab*)*, построив минимальные детерминированные конечные автоматы, распознающие соответствующие регулярные зыки.

2.2.24. По регулярному выражению в стандартной записи:

(\+|\-|) [0-9]*. [0-9]+([eE](\+| \-|) [0-9]+)

постройте несколько примеров цепочек соответствующего регулярного языка. Постройте детерминированный конечный автомат, распознающий такие цепочки.

2.2.25. Для конечного словаря Σ и любого языка $L\subseteq\Sigma^*$ обозначим

$tail(L)=\{w\in\Sigma^* \mid \exists u\in\Sigma^*: uw\in L\}$.

Иными словами, *tail*(L) – это множество всех ‘хвостов’ цепочек языка L. Постройте детерминированный конечный автомат, допускающий язык *tail*(L), где L задается регулярным выражением *ab*c*.

2.2.26. Для конечного словаря Σ и любой цепочки $w\in\Sigma^*$ обозначим w^R – цепочку, обратную цепочке *w*, например, для *w=болван* w^R*=навлоб*. Постройте алгоритм, проверяющий, содержит ли регулярный язык $L\subseteq\Sigma^*$ такую непустую цепочку *w*, что $w^R\in L$. Примените этот алгоритм к языку, заданному регулярным выражением *a*bcb**.

2.2.27. Постройте КС-грамматику, порождающую тот же язык, который определяется регулярным выражением $(a+c+ba+bc)^*(b+\varepsilon)$.

2.2.28. Определите языки, которые задаются следующими регулярными выражениями в конкретной нотации:

а) `(\+?[0-9])+|(-?[0-9])+`

б) `(\+?[0-9])+|([0-9])+`

в) `(\+\d)+|(-?\d)+`

2.3. Лексические анализаторы

2.3.1. Постройте регулярное выражение и детерминированный конечный автомат для распознавания комментариев в языке Паскаль.

2.3.2. Целые константы без знака в языке Си определяются так:

- *десятичные*: <последовательность цифр, начинающаяся не с нуля>
- *восьмеричные*: 0 <последовательность цифр>
- *шестнадцатеричные*: 0 {x, X} <последовательность цифр и букв a-f или A-F>

Постройте синтаксическую диаграмму распознавателя и транслятор этого языка.

2.3.3. Комментарий в языке Си определяется как любая символьная цепочка между скобками /* и */, а также любой текст от пары символов // до конца строки. Постройте детерминированный конечный автомат, синтаксическую диаграмму распознавателя и транслятор, выбрасывающий комментарии в языке Си.

2.3.4. Постройте блок лексического анализатора, обрабатывающий цепочки языка вещественных констант: т.е. по входной строке записывающий в таблицу констант значение очередной константы и выдающий тип лексемы (const) и ее номер в таблице констант. Примеры цепочек языка: +92.009, .345, -.002, -23E-07, .1e+3, 2301.2.

2.3.5. Постройте синтаксическую диаграмму и семантические действия лексического анализатора для расширения языка Милан возможностью включения комментариев вида /* < тело комментария> */ и // < тело комментария>. Во втором случае комментарий продолжается до конца строки.

2.3.6. Постройте синтаксическую диаграмму и семантические действия лексического анализатора для языка Милан++, пример программы на котором приведен ниже:

```
begin
    read x, y12;            /* ввод операндов */

    loop x != y12 ->        /* выполнять до равенства аргументов */
        case
             x > y12 -> x   -= y12
        or
             y12 > x -> y12 -= x
        esac
    endloop;
    write (x)              /* полученный НОД */
end
```

3. Иерархия Хомского порождающих грамматик и языков

3.1. Порождающие грамматики Хомского

3.1.1. Постройте различные структуры (сгруппируйте отдельные части) следующих двусмысленных предложений для различных их значений:

Мать любит дочь;
Петерстар поглотил Мегафон;
В Африке бхуту убивают тутси;
Ножницы для стрижки волос 20 см.

3.1.2. Постройте различные структуры (сгруппируйте отдельные части) следующих двусмысленных предложений английского языка для различных их значений:

Fruit flies like banana;
The spy saw a cop with the binocular;
I made her duck;
They are flying planes.

3.1.3. Сообщение СМИ: "*За свои услуги помощник члена Совета федерации Беридзе запросил от 150 до 200 тысяч долларов США*". Сколько смыслов у этого предложения? Постройте различные структуры этого предложения, придающие ему различные смыслы.

3.1.4. Постройте двусмысленную и недвусмысленную грамматики Хомского, порождающие цепочки языка, абстрактно представляющего структуру программ языков высокого уровня. Примеры таких цепочек:

begin int v, v, v; ***real*** v; s; s; s; s ***end***
begin real v, v; ***real*** v, v; s; s; s ***end***

3.1.5. Пусть L1 = {*abc*, *caa*} и L2 = {ε, *c*, *ca*, *caa*} - два формальных языка над словарем Σ={*a*, *b*, *c*}. Постройте следующие языки:

а) L1∪L2;
б) L1∩L2;
в) L1\L2;
г) $L1^2$;
д) L1*.

3.1.6. Пусть грамматика G определяется правилами:

S → aAbB AbB → aAbB bBb → bb A → ε	S → AB AB → CBb CB → ABB A → a aB → a	S → AaB AaB → aAaBb aBb → abb A → ε

Какому классу по Хомскому она принадлежит? Порождается ли L(G) грамматикой более узкого класса?

3.1.7. Постройте несколько цепочек языка, порождаемого следующей грамматикой:

S → P . | P ; S
P → N = E
E → E O E | N | { L } | (E)
O → ∩ | ∪
N → *x* | *y* | *z* | *u* | *v* | *w*
L → A | L , A
A → *a* | *b* | *c* | *d* | *e* | *f*

3.1.8. Пусть грамматика G определяется правилами:

S → AB
AB → *a*DB
DB → ABB
B → *b*
Ab → *b*

Какому классу (по Хомскому) она принадлежит? Порождается ли L(G) грамматикой более узкого класса?

3.1.9. Какому классу по Хомскому принадлежит грамматика с правилами:

S→ASB | AB
AB →BA
A → *a*
B → *b*

Какой язык порождает эта грамматика?

3.1.10. Какому классу по Хомскому принадлежит каждая из перечисленных ниже грамматик:

S → AS | ε A → *a* | *b*	S → AB AB → *a*ABB B → *b* A → *a*	S → ASB | BSA A → *a* B → *b* | ε SB → ε	S → A*c*B*s* A → A*c*A | B B → *a* | *b*	S → *abc* | *a*SQ *b*Q*c* → *bbcc* *c*Q → Q*c*	S → AB*a*C B*a* → *aa*B BC → DC | E *a*D → D*a* AD → AB *a*E → E*a* AE → ε

Какой язык порождает каждая грамматика? Порождается ли язык грамматикой более узкого класса?

3.1.11. Пусть грамматика G определяется правилами:

<Sentence> → <Name> | <List> <End>
<List> → <Name> | <Name>, <List>
<Name> <End> → and <Name>

<Name> → tom | dick | barry

Начальным символом грамматики является нетерминал <Sentence>. Какому классу по Хомскому принадлежит эта грамматика? Какой язык порождает эта грамматика? Какой наиболее простой грамматикой можно породить этот язык?

3.1.12. Какому классу по Хомскому принадлежит грамматика упрощенного английского языка с начальным нетерминалом <Sentence> и с правилами:

<Sentence> → <NP> <VP> | <Sentence><Conjunction><Sentence>
<NP> → <Pronoun> | <Name> | <Noun> | <Article><Noun>
<VP> → <Verb> | <VP><NP>
<Conjunction> → and | or | but | ...
< Pronoun > → me | you | I | it | she | ...
< Verb > → is | see | sees | love | loves | feel | go | carry | went | kill | ...
< Name > → John | Mary | Boston | ...
< Article > → a | an | the
< Noun > → agent | wumpus | gold | nothing | ... .

а) Постройте вывод в этой грамматике предложения: *I love Mary but Mary loves the gold.* б) Приведите примеры предложений, которые порождаются в этой грамматике, но не являются предложениями английского языка.

3.1.13. Одна из возможных контекстно-свободных грамматик упрощенного английского языка с начальным нетерминалом <Sentence> имеет следующие правила:

<Sentence> → <NP> <VP>
<NP> → < Name > | <Article><Noun> | <Article><Noun> <Sentence>
<VP> → <Verb> | <VP> <NP>
< Verb > → *ate* | *was built by* | *saw* | *was seen by* | *lay in* | *crashed* | ...
< Name > → *Jack* | *Mary* | *Peter* | ...
< Article > → *a* | *an* | *the*
< Noun > → *house* | *cheese* | *rat* | *horse* | *cat* | ...

Постройте в этой грамматике деревья вывода следующих предложений английского языка:

а) *the house the cheese lay in was built by Jack.*
б) *the house the cheese the rat saw lay in was seen by Mary.*
в) *the house the cheese the rat the cat Jack chased caught ate lay in crashed.*

3.1.14. В соответствии с грамматикой языка Милан:

Prog → *begin* L *end*
L → S | L;S
S → *i* := E | *if* B *then* L *else* L | *if* B *then* L | *while* B *do* L | *output* (E)
E → E *ots* E | E *otm* E | (E) | *i* | *c*
B → E *q* E

последний оператор в списке операторов программы не может заканчиваться ‘;’. Измените грамматику Милана так, чтобы точка с запятой могла стоять после любого оператора.

3.1.15. Сколько различных выводов цепочки *baaaab* существует в грамматике с правилами:

S → *b*A*b*

$A \to AA \mid a$

3.1.16. Постройте порождающую грамматику Хомского для языка $\{ww \mid w \in \{a, b, c\}^*\}$.

3.1.17. Постройте грамматику Хомского для языка $\{ww^R \mid w \in \{a, b, c\}^*$, w^R – цепочка, обратная к $w\}$.

3.1.18. Постройте порождающую грамматику Хомского для языка $\{a^n b^n c^n \mid n>0\}$, а также последовательность подстановок и структуру вывода в этой грамматике цепочки $a^2b^2c^2$.

3.1.19. Постройте однозначную грамматику Хомского для языка, включающего все такие цепочки из a и b, в которых число вхождений a и число вхождений b совпадают.

3.1.20*. Постройте порождающую грамматику Хомского для языка $\{a^{n\wedge 2} \mid n>0\}$, содержащего цепочки из символов a длиной 1, 4, 9, 16, ... Примеры цепочек языка: *a, aaaa, aaaaaaaaa, ...* . ***Указание***. Воспользуйтесь соотношением $(n+1)^2 = n^2 + 2n + 1$.

3.1.21*. Докажите, что для любого конечного словаря существуют языки, которые не могут быть порождены грамматиками Хомского.
Указание. Проще доказать более общее положение, а именно, что для любого конечного словаря существуют языки, которые не могут быть порождены никаким конечным описанием. Действительно, число всех возможных слов (цепочек) над конечным словарем счетно. Любой язык является некоторым подмножеством всех возможных слов над конечным словарем, поэтому множество всех языков, как множество подмножеств счетного множества, имеет мощность континуум. Для решения задачи необходимо только показать, что множество всех возможных конечных описаний над любым конечным словарем счетно, поэтому оно заведомо меньше всех возможных языков.

3.2. Контекстно-зависимые грамматики

3.2.1. По грамматике типа 0:

$S \to abc \mid aSQ$
$bQc \to bbcc$
$cQ \to Qc$

постройте контекстно-зависимую грамматику.

3.2.2. Постройте контекстно-зависимую грамматику, порождающую язык L:

$L=\{ww \mid w \in \{a, b\}^*\}$.

3.2.3. По следующей контекстно-зависимой грамматике:

$S \to aSA \mid \varepsilon$
$AA \to Ac$

$aA \to ab$

постройте эквивалентную контекстно-свободную грамматику.

3.2. 4. Постройте несколько терминальных цепочек, выводимых в следующей грамматике Хомского с терминальным словарем {0, 1}:

$S \to SS \mid RS$

$R \to RR \mid 0$

$RS \to SR$

$0S0 \to 010$

Какой язык порождает эта грамматика? Порождается ли этот язык грамматикой более простого типа?

3.2.5. Выводима ли цепочка *cabbaac* в грамматике:

$S \to SS \mid a \mid c \mid A$

$B \to bB \mid b$

$aAa \to aBa$

Если да, то постройте вывод этой цепочки.

3.3. Контекстно-свободные грамматики

3.3.1. Постройте КС-грамматику с начальным нетерминалом <Книга>, описывающую структуру книги. ***Указание.*** Любая книга состоит из содержания, оглавления и нескольких глав. Каждая глава состоит из разделов, разделы – из параграфов, параграфы – из предложений.

3.3.2. Понятие списка в математике задается следующим рекурсивным определением:

1. *Односимвольная цепочка p является списком*.
2. *Если v и w – списки, то цепочки [v] и v;w тоже списки.*
3. *Других списков нет*.

а) Постройте словарь языка списков и КС-грамматику Хомского, порождающую все списки с двумя символами a, b.

б) Постройте недвусмысленную КС-грамматику Хомского, порождающую все списки.

в) Приведите примеры списков.

3.3.3. Правильно построенные формулы логики высказываний определяются так:

а) любое атомарное высказывание p есть ППФ;

б) если φ и φ1 – ППФ, то $\neg\varphi$ и $\varphi \vee \varphi 1$ – тоже ППФ;

в) других правильно построенных формул логики высказываний нет.

Постройте КС-грамматику Хомского, порождающую все правильно построенные формулы логики высказываний.

3.3.4. Регулярные выражения над конечным словарем Σ рекурсивно определяются следующим образом:

а) ∅, ε - и любое a∈Σ - регулярные выражения;

б) Если R_1 и R_2- регулярные выражения, то регулярными выражениями будут
их сумма $R_1 + R_2$; (знак '+', иногда записывают как '|')
их произведение R_1R_2;
итерация R_1* и урезанная итерация R_1^+;

в) Скобки ( ) используются для группировки (для задания порядка выполнения операций).

г) Других регулярных выражений нет.

Постройте КС-грамматику Хомского, порождающую все регулярные выражения в алфавите Σ={*a, b*}.

3.3.5. Формулы линейной темпоральной логики (LTL) рекурсивно определяются так:

а) любой атомарный предикат p является формулой LTL;

б) если φ и ψ – формулы LTL, то ¬φ и φ∨ψ – тоже формулы LTL;

в) если φ и ψ – формулы LTL, то **X**φ и φ**U**ψ – тоже формулы LTL;

г) Скобки '(' и ')' используются для группировки (для задания порядка выполнения операций);

д) других формул линейной темпоральной логики нет.

Постройте КС-грамматику Хомского, порождающую все правильно построенные формулы линейной темпоральной логики.

3.3.6. Формулы темпоральной логики ветвящегося времени (CTL) рекурсивно определяются так:

а) любой атомарный предикат p является формулой CTL;

б) если φ и ψ – формулы CTL, то ¬φ и φ∨φ1 – тоже формулы CTL;

в) если φ и ψ – формулы CTL, то **AX**φ, **EX**φ, **AF**φ, **EF**φ, **AG**φ, **EG**φ, **A**(φ**U**ψ) и **E**(φ**U**ψ) – тоже формулы CTL;

г) других формул темпоральной логики ветвящегося времени нет.

Постройте КС-грамматику Хомского, порождающую все правильно построенные формулы темпоральной логики CTL.

3.3.7. Правильно построенные формулы логики предикатов определяются на предметной области M так:

Вводятся символы, которые подразделяются на классы:

константы M	c1, c2, …
переменные M	x, y, z, …
функциональные символы M	f, g, ...

Вводятся термы как константы, переменные
или n-местные функции от термов: c, x, f (t, t, …, t)

Вводятся атомарные m-местные предикатные символы: P

Правильно построенные формулы логики предикатов формально определяются так:

любой m-местный атомарный предикатный символ есть ППФ;

любые ППФ, связанные логическими операторами – ППФ;

ППФ, перед которой стоит квантор ∃x или ∀x, тоже является ППФ.

Пример правильно построенной формулы логики предикатов: $\forall x\ \exists y\ P(x, c, g(y))$.
Постройте КС-грамматику Хомского, порождающую все правильно построенные формулы логики предикатов.

3.3.8. В КС-грамматике:

$S \to ASB \mid \varepsilon$
$A \to aAS \mid a \mid aEb$
$B \to SbS \mid A \mid bb \mid \varepsilon$
$C \to cC \mid EdA$
$E \to cE \mid BdC$

а) освободитесь от бесполезных и циклических продукций;
б) освободитесь от ε-правил.

3.3.9. Освободитесь от бесполезных, пустых и циклических продукций в грамматике:

$S \to 0A0 \mid 1B1 \mid BB$	$S \to SbAc \mid adA$	$S \to aA \mid aBB$
$A \to C \mid EaA$	$A \to BcS \mid abc$	$A \to aaA \mid \varepsilon$
$B \to S \mid A$	$B \to Ec \mid dB$	$B \to bB \mid bbC$
$C \to S \mid \varepsilon \mid CE$	$D \to ccC \mid DdA \mid d$	$C \to B \mid DdA$
$E \to CE \mid aE$	$E \to CE$	$E \to CE$

Какой язык порождает каждая грамматика?

3.3.10. Освободитесь от ε-продукций в грамматике:

$S \to aSbS$
$S \to BaB$
$B \to \varepsilon$
$B \to bS$

какой язык порождает эта грамматика?

3.3.11. Существуют ли грамматики LL(0)? Если да, то постройте пример такой грамматики.

3.3.12. Какой язык порождает КС-грамматика:

$\{S \to \#B\#,\ B \to E=E,\ B \to i,\ E \to E+i,\ E \to i\}$

3.3.13. Нетерминал A называется рекурсивным, если существует вывод: $A \Rightarrow^* \alpha A \beta$ с непустой цепочкой $\alpha\beta$. Можно ли язык $\{a^n b c^m \mid n,m>0\}$ задать грамматикой без рекурсивных правил?

3.3.14. Принадлежит ли цепочка *abaababb* языку, порождаемому КС-грамматикой с правилами:

$S \to SaSb \mid \varepsilon$

3.3.15. Постройте однозначную КС-грамматику Хомского для языка правильных скобочных выражений.

3.3.16. Следующая КС-грамматика Хомского с начальным символом Lexp-seq, представленная в БНФ-нотации, порождает цепочки упрощенного языка Lisp:

Lexp-seq ::= Lexp-seq Lexp | Lexp
Lexp ::= Atom | List
Atom ::= ‘*number*’ | ‘*identifier*’
List ::= ‘(’ Lexp-seq ‘)’

Постройте дерево вывода цепочки *b* (*a* 23 *b* (*m a y*) z) в этой грамматике.

3.4.17. Следующая КС-грамматика Хомского с начальным символом Stmt, представленная в БНФ-нотации, порождает цепочки операторов языка nanoPromela (Process meta language):

Stmt ::= ‘*skip*’ | ‘*x* := *expr*’ | ‘*c*?*x*’ | ‘*c*!*expr*’ |
Stmt { ‘;’ Stmt } | ‘*atomic’* ‘{’ Assignments ‘}’ |
‘*if* :: *g* ⇒’ Stmt { ‘:: *g* ⇒’ Stmt} ‘*fi*’ |
‘*do* :: *g* ⇒’ Stmt { ‘:: *g* ⇒’ Stmt} ‘*od*’
Assignments ::= ‘*x* := *expr*’ { ‘ ; *x* := *expr*’}

Постройте несколько цепочек операторов языка nanoPromela.

3.3.18. Постройте все возможные деревья вывода слова *cabbca* в следующей грамматике Хомского:

S → SS | *a* | *c* | *a*B*c*
B → B*b* | *b*

3.3.19. Грамматика, порождающая арифметические выражения:

E → E+E | E-E | E*E | E/E | (E) | *i*

является двусмысленной. Постройте для следующей цепочки все возможные деревья вывода в этой грамматике:

i * (*i* + *i* – *i*) + *i*

3.3.20. Постройте левый и правый выводы цепочки *abbbb* в грамматике:

G={ S → *a*AB, A → *b*B*b*, B → A | ε }

3.3.21. Принадлежит ли цепочка (( )( )) ( ) языку, порождаемому КС-грамматикой с правилами:

S → SA | A
A → (S) | ( )

Постройте канонический левый и канонический правый выводы этой цепочки в грамматике.

3.3.22. Принадлежит ли цепочка 00011011 языку, порождаемому КС-грамматикой с правилами:

S → SS | A
A → 0A1 | S | 01

Постройте вывод этой цепочки в грамматике.

3.3.23. Принадлежит ли цепочка 0111000 языку, порождаемому КС-грамматикой с правилами:

S → A0B | B1A
A → BB | 0
B → AA | 1
Постройте вывод этой цепочки в грамматике.

3.3.24. Постройте КС-грамматику, порождающую языки:
а) $\{w \in \{a, b\}^* \mid |w|_a = 2|w|_b\}$, т.е. язык, в каждом слове которого число символов *a* вдвое меньше числа символов *b*.
б) $\{ a^n b^m \mid m \geq n+2\}$.
в) $\{wuw^R \mid u,w \in \{a,b\}^* \}$.
г) $\{ a^{2\uparrow n} \mid n \geq 0\}$.

3.3.25. По грамматике Хомского:
S → LDL
L → L*a* | L*b* | *a*
D → D0 | D1 | 0 | 1
постройте деревья вывода следующих слов: *ab*10*aa*, *abba*0*a*, *a*1*a*. Покажите, что слова *ab*10, *b*1*a*, *abba* не выводимы в этой грамматике. Опишите язык, порождаемый этой грамматикой. Существует ли регулярное выражение, задающее тот же язык?

3.3.26. Постройте дерево вывода цепочки α = *cbcaadcb* в следующей грамматике:
S → A S B| *a*
A → *c* | A*b*
B → *a*DA | *b*
D → *d*D |*d*
По построенному дереву вывода восстановите вывод α от начального символа S и вывод α от терминальной цепочки до S.

3.3.27. Постройте дерево вывода цепочки α = *aaabcbbbdbc* в следующей КС-грамматике:
S → *a*SbA| *a*
A → *c* | AB
B → *b*BD | *b*
D → *d*D |*d*
По построенному дереву вывода цепочки восстановите ее левый и правый вывод.

3.3.28.Каждое правило КС грамматики Хомского представляет собой цепочку некоторого языка. КС грамматика Хомского – это конечное множество КС-правил, которое . Пример КС грамматики Хомского: { E –> E+T | T, T –> T*P | P, P –> i | c | (E) }.
а) Постройте КС грамматику языка, задающего все возможные правила КС грамматик Хомского.
б) Постройте КС грамматику языка, задающего все возможные КС грамматики Хомского.

3.3.29. Описание языка грамматикой тоже должно быть записано цепочкой символов, т.е. представляет собой язык (его часто называют ‘метаязыком’. Представление грамматики

Хомского в БНФ-нотации представляет собой цепочку некоторого метаязыка. Пример правил в БНФ:

<variable-declaration-part> ::= var <var-declaration> {; <var-declaration>}

<integer-constant> ::= [+ | -] <digit> {<digit>}

Постройте грамматику этого метаязыка.

3.3.30. Для естественных языков грамматики Хомского неадекватны: они не могут выразить всех синтаксических связей, существующих в естественных языках. Однако для небольших фрагментов естественных языков, например, шаблонов, конечных списков команд и т.п., грамматики Хомского использовать очень удобно. Пример грамматики фрагмента русского языка:

$S \rightarrow Q\ V \mid V\ Q$

$Q \rightarrow \underline{N}$им

$\underline{V} \rightarrow \underline{V}\ O \mid O\ \underline{V} \mid V$

$\underline{N_z} \rightarrow A_z\ \underline{N_z} \mid N_z$ // z – один из шести падежей русского языка

$V \rightarrow$ *шла* | *бежала* | *летела* | *скакала* | ...

$N_z \rightarrow$ *деревня*$_z$ | *дорога*$_z$ | *студентка*$_z$ | *ведьма*$_z$ | *метла*$_z$ ...

$A_z \rightarrow$ *красивая*$_z$ | *молодая*$_z$ | *широкая*$_z$ | *пыльная*$_z$ | *ночная*$_z$ ...

$O \rightarrow$ *пешком* | *верхом* | *галопом* | *на* $\underline{N_{предл}}$ | *над* $\underline{N_{твор}}$ | *в* $\underline{N_{предл}}$ | *по* $\underline{N_{дат}}$...

В этой грамматике постройте деревья вывода предложений:

а) *над темной деревней верхом на метле летела старая ужасная ведьма*.

б) *молодая студентка мчалась по пыльной дороге на новой дорогой машине*.

в) *ехала деревня мимо мужика*.

3.3.31. Пусть в представлении булевых функций в виде BDD обозначение подграфа с вершиной v, помеченной переменной x, записывается формулой так: $[x \rightarrow p_0, p_1]$, где x-переменная, помечающая вершину, а p_0 и p_1 – подграфы, в которые из вершины v идут ребра, помеченные, соответственно, 0 и 1. Терминальные вершины, помеченные 0 и 1, записываются как [0] и [1]. Постройте КС-грамматику, порождающую все такие формулы. Постройте дерево вывода формулы:

[x1→[0], [x2 →[1], [x3 →[0], [x4 →[0],[1]]]]]

и определите, какую булеву функцию представляет эта BDD.

3.3.32. Пусть в представлении булевых функций в виде BDD обозначение подграфа с вершиной vk, помеченной переменной xi, записывается формулой так: vk: $[xi \rightarrow p_0, p_1]$, где p_0 и p_1 – подграфы, в которые из вершины vk идут ребра, помеченные, соответственно, значениями 0 и 1 переменной xi. Терминальные вершины, помеченные 0 и 1, записываются как 0: [0] и 1: [1]. Постройте КС-грамматику, порождающую все такие формулы. Порождает ли эта грамматика редуцированную BDD? Порождает ли эта грамматика упорядоченную BDD? Как с помощью семантических проверок гарантировать упорядоченность BDD, построенных по сгенерированным формулам?

3.3.33*. КС-грамматика в нормальной форме Грейбах содержит только правила вида $A \rightarrow aW$, где a-терминал, а W - последовательность (может быть пустая) нетерминалов, а также правило $S \rightarrow \varepsilon$, если порождаемый грамматикой язык содержит пустую цепочку. Покажите,

что любой КС-язык можно описать грамматикой в нормальной форме Грейбах. Приведите к этой форме грамматику:

$S \to A\ S\ B \mid D\mathit{ab}$

$A \to \mathit{c} \mid A\mathit{b}$

$B \to \mathit{a}DA \mid \varepsilon$

$D \to \mathit{d}D \mid \mathit{d}B$

3.4. Автоматные грамматики

3.4.1. а) Справедливо ли утверждение, что любое подмножество регулярного языка регулярно? б) Справедливо ли утверждение, что у любого контекстно-свободного языка существует подмножество, которое является регулярным языком? Приведите примеры, подтверждающие вашу точку зрения.

3.4.2. Следующая КС-грамматика:

$S \to ABC$

$A \to \mathit{a}A \mid \mathit{a}$

$B \to B\mathit{b} \mid \varepsilon$

$C \to C\mathit{c} \mid c$

порождает язык $\{\mathit{a}^n\mathit{b}^m\mathit{c}^k \mid n, k > 0, m \geq 0\}$. Постройте автоматную грамматику, порождающую тот же язык.

3.4.3. Постройте автоматную грамматику, порождающую язык, задаваемый регулярным выражением 1*10(0+1)*. По этой грамматике постройте недетерминированный конечный автомат, по нему постройте детерминированный конечный автомат, распознающие тот же самый язык.

3.4.4. Грамматика Хомского называется праволинейной, если ее правила имеют вид:

$A \to \alpha A$

$A \to \alpha$

где α - произвольная (в том числе и пустая) цепочка терминальных символов. Докажите, что по любой праволинейной грамматике может быть построена эквивалентная автоматная грамматика Хомского. Покажите, как преобразовать праволинейную грамматику:

$S \to \mathit{abc}A \mid \mathit{c}$

$A \to \mathit{ac}B \mid \mathit{a}$

$B \to \mathit{bcba}B \mid \varepsilon$

к автоматной грамматике.

3.4.5. Постройте детерминированный конечный автомат, распознающий тот же язык, который порождается следующей грамматикой Хомского:

$S \to \mathit{a} \mid \mathit{aT}$

$T \to \mathit{a} \mid \mathit{b} \mid \mathit{aT} \mid \mathit{bT}$

Порождается ли слово *abab* в этой грамматике?

3.4.6. Идентификаторы в языке Фортран представляются от одного до шести символов, первый из которых – буква, а остальные – буквы или цифры. Постройте автоматную грамматику и конечный автомат, задающие язык идентификаторов Фортрана.

4. Иерархия распознающих автоматов для порождающих грамматик Хомского

4.1. Автоматные грамматики и конечные автоматы

4.1.1. По заданной грамматике G типа 3 постройте конечный автомат, распознающий тот же язык, который порождает грамматика G. Постройте регулярное выражение, описывающее все цепочки этого языка.

4.1.2. По регулярному выражению R=10*1*(0+1)* постройте конечный автомат-распознаватель A(R) соответствующего регулярного языка, по автомату A(R) постройте грамматику типа 3, которая порождает тот же язык.

4.1.3. По регулярному выражению R=0*(0+1)*0*11* постройте детерминированный конечный автомат-распознаватель соответствующего регулярного языка, по этому автомату постройте грамматику типа 3, которая порождает тот же язык. Постройте регулярное выражение, описывающее все цепочки этого языка. Совпадает ли оно с R?

4.1.4. Постройте грамматику типа 3, порождающую все такие цепочки из 0 и 1, которые содержат хотя бы один символ.

4.1.5. Дано регулярное выражение в стандартной нотации, описывающее все действительные числа:

(\+|-|)digit*. digit+([eE](\+|-|) digit+), *где* digit = [0-9]

По этому регулярному выражению постройте грамматику типа 3, порождающую такие числа. Постройте также детерминированный конечный автомат, распознающий все такие цепочки.

4.1.6. Постройте регулярное выражение, определяющее тот же автоматный язык, который порождается следующей грамматикой типа 3:

$G = \{ S \to aS, S \to A, S \to \varepsilon, A \to bA, A \to S\}$

4.1.7. Постройте грамматику Хомского типа 3, которая порождает тот же автоматный язык, который задается следующим регулярным выражением:

$R = (a + c + ba + bc)^* (b+\varepsilon)$

4.1.8. Над алфавитом $\{a, b, c\}$ задана следующая автоматная порождающая грамматика:

$S \to aS \mid aB \mid cS \mid c$

$B \to bB \mid aB \mid a$

Постройте детерминированный конечный автомат, распознающий язык, порождаемый этой грамматикой.

4.2. Контекстно-свободные грамматики и МП-автоматы

4.2.1. Постройте МП-автомат, распознающий цепочки языка $\{a^n b^n | \ n \geq 0\}$.

4.2.2. Постройте МП-автомат, распознающий цепочки языка $\{a^n b^{2n} | \ n \geq 0\}$.

4.2.3. Постройте МП-автомат, распознающий такие цепочки над словарем $\{a, b\}$, в которых количество вхождений символов *a* вдвое больше числа вхождений символа *b*.

4.2.4. Постройте КС-грамматику и МП-автомат, распознающий такие цепочки над словарем $\{a, b\}$, в которых число вхождений символа *a* не превышает числа вхождений символа *b*.

4.2.5. Постройте МП-автомат, распознающий язык правильных скобочных выражений. Например, цепочка "(( ) (( )) ( )" принадлежит этому языку, а цепочка ") ( )))" –нет.

4.2.6. Язык Дика (Dyck language) порядка n – это язык правильных скобочных выражений с n типами различных скобок. Например, при n=2 цепочкой такого языка может быть '[( ) ([( ) ( )])]. Постройте порождающую грамматику Хомского для языка Дика порядка 3 и МП-автомат, его распознающий.

4.2.7. Постройте порождающую грамматику Хомского для языка правильных скобочных выражений глубины не более 3, в которых в квадратных скобках стоит выражение только из круглых скобок, а в фигурных скобках обязательно стоят квадратные скобки.

4.2.8. Постройте МП-автомат, переводящий арифметическое выражение, заданное в инфиксной форме, в эквивалентное выражение в префиксной форме.

4.3. Машины Тьюринга как распознаватели языков

4.3.1. Постройте машину Тьюринга, распознающую язык $\{a^{n\wedge 2} | \ n>0\}$, содержащий цепочки из символов *a* длиной 1, 4, 9, 16, ... Примеры цепочек языка: *a, aaaa, aaaaaaaaa, ...* .
Указание. Воспользуйтесь соотношением $(n+1)^2 = n^2 + 2n + 1$.

4.3.2.Постройте машину Тьюринга, распознающую цепочки языка $\{a^n b^{2n} c^n | \ n \geq 0\}$.

4.3.3. Постройте машину Тьюринга, распознающую все такие цепочки над словарем $\{a, b\}$, в которых количества вхождений символов *a* и *b* равны.

4.3.4. Постройте машину Тьюринга, распознающую такие цепочки над словарем $\{a, b\}$, в которых количества вхождений символов *a* и *b* не равны.

4.3.5. Постройте машину Тьюринга, распознающую язык правильных скобочных выражений. Например, цепочка ‘(( ) (( )) ( )’ принадлежит этому языку, а цепочка ‘) ( )))’ –нет.

4.3.6. Постройте машину Тьюринга, распознающую все цепочки языка $\{ww \mid w \in \{a,b\}^*\}$, т.е. цепочки, состоящие из двух произвольных повторяющихся слов.

4.3.7. Постройте машину Тьюринга, распознающую все цепочки языка $\{ww^R \mid w \in \{a,b\}^*\}$.

4.3.8. Постройте машину Тьюринга, упорядочивающую любую цепочку из нулей и единиц так, что сначала идут все нули, а затем все единицы исходной цепочки. Пример: по цепочке 011010110 должна быть выдана в качестве результата цепочка 000011111.

5. Другие модели задания формальных языков

5.1. БНФ-нотация и грамматики Хомского

5.1.1. В БНФ нотации определен язык формул:

<формула>::=<цифра>[<цифра>] | (<формула><знак> <формула>)
<знак>::=+ | -
<цифра>::= 0|1|2|3|4|5|6|7|8|9

Постройте грамматику Хомского этого языка и приведите примеры его цепочек.

5.1.2. Иногда при записи грамматики в БНФ нетерминальные символы записываются словами с заглавной буквы, а терминальные выделяются апострофами ‘ и ’. Запишите следующую грамматику, представленную в БНФ нотации, в стандартной записи Хомского:

Program ::= Statement { ‘;’ Statement } ‘.’
Statement ::= Name ‘:=’ Expression
Expression ::= Expression Operator Expression | Name | ‘{’ Elements ‘}’ | ‘(’ Expression ‘)’
Operator ::= ‘∩’ | ‘∪’
Name ::= ‘A’ | ‘B’ | …| ‘X’ | ‘Y’ | ‘Z’
Elements ::= Element { ‘,’ Element }
Element ::= ‘a’ | ‘b’ | … | ‘x’ | ‘y’ | ‘z’

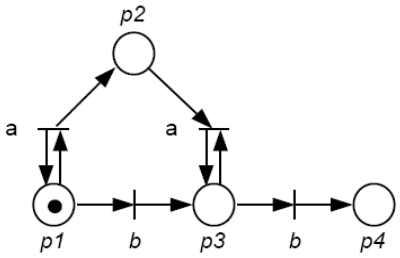
Постройте вывод трех цепочек, порождаемых этой грамматикой.

5.1.3. Иногда при записи грамматики в БНФ нетерминальные символы записываются словами с заглавной буквы, а терминальные выделяются апострофами ‘ и ’. Запишите следующую грамматику, представленную в БНФ нотации, в стандартной записи Хомского:

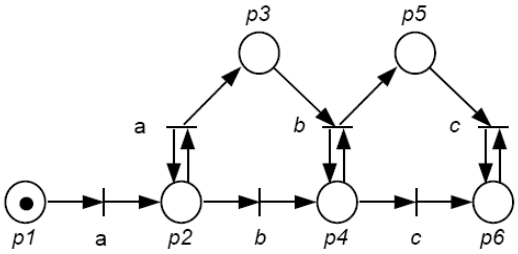
Prog ::= ‘begin’ Stmnt { ‘;’ Stmnt } ‘end’
Stmnt ::= ‘if’ (Expr Rel Expr) ‘{’ Stmnt { ‘;’ Stmnt } ‘}’ | ‘s’
Rel ::= ‘==’ | ‘!=’ | ‘>’ | ‘<’ | ‘>=’ | ‘<=’
Exp ::= Term {AddOp Term}
Term ::= Primary {MulOp Primary}
Primary ::= ‘i’ | ‘c’ | ‘(‘ Exp ‘)’
AddOp ::= ‘+’ | ‘-’
MulOp ::= ‘*’ | ‘/’

5.2. Сеть Петри как модель абстрактного языка

5.2.1. Какой язык порождает следующая сеть Петри?



5.2.2. Какой язык порождает следующая сеть Петри?



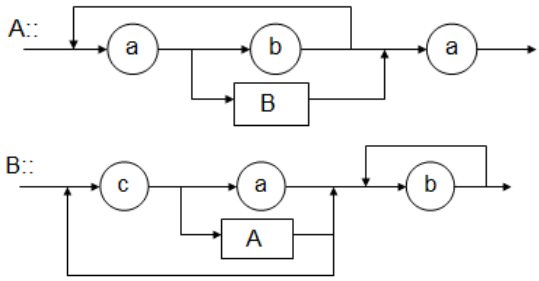
5.2.3. Постройте сеть Петри, порождающую язык, состоящий из цепочек с одинаковым числом вхождений символов *a* и *b*.

5.2.4. Постройте сеть Петри, порождающую язык, в цепочках которого число вхождений символа *a* больше числа вхождений символа *b*.

5.2.5. Постройте сеть Петри, порождающую язык, в цепочках которого число вхождений символа *a* вдвое больше числа вхождений символа *b*.

5.3. Синтаксические диаграммы и грамматики Хомского

5.3.1. Постройте КС-грамматику, порождающую тот же язык, что и следующие синтаксические диаграммы с начальным символом A:



5.4. Грамматики с рассеянным контекстом

5.4.1. Постройте грамматику с рассеянным контекстом, порождающую язык $\{ww \mid w \in \{a,b\}^*\}$. Приведите вывод цепочки aabaabaa этого языка в построенной грамматике.

5.4.2. Постройте грамматику с рассеянным контекстом, порождающую язык $\{w^R w w^R \mid w \in \{a,b\}^*\}$. Приведите вывод цепочки *aababbabaaaabab* этого языка в построенной грамматике.

6. Язык Милан и стековая машина

6.1. Стековая машина

6.1.1. Постройте программу стековой машины для вычисления значения арифметического выражения: a+(b-2)*(x+3) при заданном распределении памяти для констант и переменных.

6.1.2. Постройте программу стековой машины для вычисления условного оператора присваивания:

a := x>y+2 ? 32 –y15 : 3+x – y1*z +v;

6.1.2. Постройте программу стековой машины, которая выполняет следующую программу языка Милан:

```
begin
        m:=read;
        n:=read;
        while m≠n do
                if m>n then m:=m-n else n:=n-m fi
        od;
        write(m)
end
```

6.1.3. Условное выражение состоит из отношений, связанных знаками конъюнкции и дизъюнкции. Отношение строится из двух переменных и знака отношения {=, < >, >, <, =>,<=}. Примеры цепочек языка условных выражений:

x>y ∨ x<=z,

x = = a ∧ x >b ∨ x<c.

Постройте атрибутную грамматику для трансляции условных выражений на язык стековой машины. На языке команд стековой машины используйте команды OR, AND и COMPARE k со значениями k, соответствующими различным знакам отношения, а также LOAD a, загружающую содержимое регистра памяти с адресом a в стек. Программа должна вычислять значения True или False в верхушке стека в зависимости от истинностных значений переменных, входящих в выражение.

6.1.4. Оператор *for* языка Java имеет следующую структуру:

for (<инициализация>; <условие> ; <итерация>) <оператор>;

Постройте программу стековой машины для вычисления следующей программыг:

```
begin
   N := read; y:= 2; z2:= 5;
  for ( i := 0; i<N; i++) a += 2*y +i*z2;
end
```

6.2. Язык Милан

6.2.1. На языке Милан с грамматикой:

```
Prog → begin L end
L → S | S ; L
S →  i:=E | if B then L | if B then L else L | while B do L | output(E)
B → E q E
E → (E) | E ots E | E otm E | i | c | read
```

построена программа вычисления факториала:

```
begin
   z:=1; i:=1;
   while (i≠y) do
        z:=z*x; i=i+1;
    output(z);
end
```

Постройте все возможные деревья вывода этой программы в указанной грамматике.

6.2.2. На языке Милан с грамматикой:

```
Prog → begin L end
L → S | S ; L
S →  i:=E | if B then L | if B then L else L | while B do L | output(E)
B → E q E
E → (E) | E ots E | E otm E | i | c | read
```

построена программа:

```
begin
      m:=read;
      n:=read;
      while m>0 do
            if m≥n then if m=n then m:=m-n else n:=n-m;
      output(m)
end
```

Постройте все возможные деревья вывода этой программы в указанной грамматике.

6.2.3. Постройте все возможные деревья вывода для цепочки *a*+(*b***c*-*d*) в грамматике:

E → E+E | E-E | E*E | (E) | i

6.2.4. В 70-е годы прошлого века в России была построена мини-ЭВМ «Наири», на которой использовался язык АП (*автоматическое программирование*), использующий русские ключевые слова.

Пример программы на языке АП:

```
1. допустим a=1 b=-3 c=2
2. введем x y
3. вычислим d= (b^2-4ac)
4. если d<0 идти к 7
5. вычислим x=(-b-d)/(2a)
6. вычислим y=(-b+d)/(2a)
7. печатаем x y
```

```
8. кончаем
```

Постройте грамматику языка АП и разработайте транслятор этого языка в язык стековой машины методом рекурсивного спуска.

6.2.5. Постройте грамматику языка Милан++, пример программы на котором приведен ниже:

```
begin
    read x, y;              /* ввод операндов */

    loop x != y ->          /* выполнять до равенства аргументов */
        case
             x > y -> x := x - y
        or
             y > x -> y := y - x
        esac
    endloop;
    write (x)               /* полученный НОД */
end
```

6.2.6. На языке Милан++ построена программа:

```
begin
    i,s := 0,0;
    n := read;
    loop
        i < n -> s,i := s+read,i+1
    endloop;
    write (s)
end
```

постройте для этой программы соответствующую программу на языке стековой машины. Постройте синтаксические диаграммы и семантические операции для трансляции языка Милан++ методом рекурсивного спуска.

6.2.7. Постройте грамматику языка Э.Дейкстры из его книги “Дисциплина программирования”. Примеры программ на этом языке:

```
begin
q1,q2,q3,q4:=Q1,Q2,Q3,Q4;
do q1>q2 → q1,q2:=q2,q1;
 [] q2>q3 → q2,q3:=q3,q2;
 [] q3>q4 → q3,q4:=q4,q3
od
end
```

```
begin
k,j = 0,1;
do j≠n →
    if f ≥ g→ j := j+1;
       [] f ≤ g → k,j := j,j+1
     fi;
   [] j = n → j := j-1
od
end
```

7. Неоднозначные КС-грамматики и атрибутные трансляции

7.1. Двусмысленные КС-грамматики

7.1.1. В грамматике:

G1:: S→if *b* then S | if *b* then S else S | *s*

постройте несколько деревьев вывода для цепочки:

if *b* then if *b* then if *b* then *s* else if *b* then if *b* then *s* else *s*

Постройте несколько деревьев вывода для этой цепочки также в грамматике:

G2:: S →if *b* then S | if *b* then S1 else S | *s*

S1 →if *b* then S1 else S | *s*

7.1.2. Является ли следующая грамматика условных операторов однозначной:

G:: S→if *b* then S B | *s*

B→ else S | ε.

Постройте в этой грамматике возможные деревья вывода для цепочек:

if *b* then if *b* then *s* else if *b* then *s* else *s*,

if *b* then if *b* then if *b* then *s* else if *b* then *s* else if *b* then *s* else *s*

7.1.3*. Следующая грамматика порождает язык условных операторов языка Паскаль:

S → AX | *s*

A → if *b* then *s* A | ε

X → if *b* then *s* X else S |ε

Однозначная ли эта грамматика? Постройте дерево вывода в этой грамматике цепочки:

if *b* then if *b* then if *b* then if *b* then *s* else if *b* then *s* else *s*

7.1.4. В порождающей КС-грамматике с начальным нетерминалом L:

L→ S | L; S

S→ while *b* do L | s

постройте все возможные деревья вывода цепочки:

while *b* do *s*; while *b* do while *b* do *s*; *s*

7.1.5. Постройте несколько деревьев вывода в этой грамматике для цепочки:

s; *s*; *s*; *s*

в грамматике

G:: S→ *s* | S;S

Эта грамматика, порождающая последовательности операторов, разделенных точками с запятой, является двусмысленной. Неоднозначность дерева вывода в этой грамматике объясняется возможностью по-разному структурировать группу рядом стоящих операторов (правило S→ S;S). Постройте две недвусмысленные грамматики, порождающие последовательности операторов, которые выделяют по одному оператору из такой группы либо слева, либо справа. Постройте деревья вывода этой цепочки в построенных грамматиках.

7.1.6. В двусмысленной грамматике правильных скобочных выражений:

$S \rightarrow (S) \mid S\ S \mid \varepsilon$

неоднозначность дерева вывода объясняется возможностью по-разному структурировать группу рядом стоящих скобок (правило $S \rightarrow S\ S$). Постройте несколько деревьев вывода в этой грамматике для цепочки:

(( )) ( ) (( ) ( ) ( ))

Постройте две однозначные грамматики правильных скобочных выражений, которые выделяют по одной вложенной скобке из такой группы либо слева, либо справа. Постройте деревья вывода этой цепочки в построенных грамматиках.

7.1.7. Пусть задана грамматика упрощенного английского языка с начальным нетерминалом <Sentence> и с правилами:

<Sentence> → <NP> <VP> | <Sentence><Conjunction><Sentence>
<NP> → <Pronoun> | <Name>
<VP> → <Verb> | <VP><NP>
<Conjunction> → *and* | *or* | *but* | ...
< Pronoun > → *me* | *you* | *I* | *it* | *she* | ...
< Verb > → *is* | *see* | *sees* | *love* | *loves* | *feel* | *go* | *carry* | *went* | *kill* | ...
< Name > → *John* | *Mary* | *James* | ...

а) Постройте вывод цепочки "*I like Mary and she loves me*" в этой грамматике.

б) Является ли эта грамматика двусмысленной?

в) Если да, то приведите пример предложения с двумя различными деревьями вывода в этой грамматике.

г) Постройте недвусмысленную грамматику, порождающую тот же язык.

7.1.8. Все формулы логики высказываний могут быть выражены в базисе $\{ \Rightarrow, 0\}$. Постройте недвусмысленную КС-грамматику, в которой бинарная операция импликации вычисляется справа налево. Формула

$A1 \Rightarrow A2 \Rightarrow \ldots \Rightarrow An \Rightarrow B$

понимается как $(A1 \Rightarrow (A2 \Rightarrow (\ldots \Rightarrow (An \Rightarrow B))\ldots))$.

7.1.8. КС-грамматику, задающую все формулы линейной темпоральной логики LTL часто записывают так:

$\varphi \rightarrow p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi$

а) Покажите, что эта грамматика двусмысленна.

б) Постройте для этой грамматики все возможные деревья вывода формулы $\mathbf{GF}p \vee \mathbf{X}q$.

в) Постройте недвусмысленную грамматику языка формул LTL.

7.1.9. Является ли следующая грамматика недвусмысленной?

$S \rightarrow P\ . \mid P\ ;\ S$
$P \rightarrow N = E$
$E \rightarrow E\ O\ E \mid N \mid \{\ L\ \} \mid (\ E\ )$
$O \rightarrow \cap \mid \cup$
$N \rightarrow x \mid y \mid z \mid u \mid v \mid w$
$L \rightarrow A \mid L\ ,\ A$

A → a | b | c | d | e | f

Постройте такую цепочку, порождаемую этой грамматикой, которая имеет два дерева вывода.

7.1.10. Для следующей КС-грамматики с начальным нетерминалом Program, записанной в БНФ:

Program ::= Statement '.' | Statement ';' Program
Statement ::= Name '=' Expression
Expression ::= Expression Operator Expression | Name | '{' Elements '}' | '(' Expression ')'
Operator ::= '∩' | '∪'
Name ::= 'A' | 'B' | …| 'X' | 'Y' | 'Z'
Elements ::= Element | Element ',' Elements
Element ::= 'a' | 'b' | … | 'x' | 'y' | 'z'

постройте эквивалентную недвусмысленную грамматику, в которой соблюдается приоритет операций '∩' и '∪'.

7.1.11. Множество формул логики предикатов представляет собой язык. Пусть *x* обозначает любую переменную предметной области, *a* обозначает любую константу предметной области, p – обозначает отношение на множестве объектов предметной области.
Грамматика, порождающая все формулы логики предикатов иногда задается так:

argument ::= *x* | *a*
argument_list ::= argument | argument, argument_list
atomic_formula ::= p | p(argument_list)
formula ::= atomic_formula | ¬formula | formula ∨ formula
formula ::= ∀x formula | ∃x formula

Является ли эта грамматика двусмысленной? Определите все источники двусмысленности. Приведите примеры двусмысленных формул.

7.2. Теория атрибутной семантики Кнута

7.2.1. Постройте грамматику, порождающую описания переменных вида:

integer a, b, c;
real d, e;

и постройте семантические атрибуты, позволяющие присвоить каждой переменной в таблице имен свой тип. Покажите семантические вычисления на дереве синтаксического анализа цепочки ***integer*** a, b, c;.

7.2.2. Контекстно-свободную грамматику, порождающую описания переменных языка Паскаль, можно задать следующим образом (D – начальный нетерминал):

D → ***var*** V : T;
V→ i | V, i
T→ ***real*** | ***integer***

Постройте семантические атрибуты, позволяющие присвоить после трансляции описания каждой переменной в таблице имен свой тип. Покажите семантические вычисления на дереве синтаксического анализа цепочки ***var*** a, b, c : ***integer***;.

Указание: Для нетерминала V следует определить два атрибута: наследуемый атрибут V.*тип* и синтезированный атрибут V.*множ* - множество имен тех переменных, которые выводятся из V. Значение атрибутаV.*тип* должно быть присвоено всем переменным множества V.*множ*

7.2.3. Для следующей грамматики, порождающей рациональные двоичные числа, постройте атрибутную семантику для трансляции символьных цепочек в их значения:

Num ::= Int '.' Frac | '.' Frac | Int '.'
Int ::= '*ц*' | '*ц*' Int
Frac ::= '*ц*' | Frac '*ц*'

7.2.4. Определите семантические атрибуты и их зависимости для вычисления значения рациональных двоичных чисел, определяемых следующими грамматиками:

G1:	S→L**.**L L→BL | B B→0 | 1	G2:	S→L**.**R L→BL | B R→RB | B B→0 | 1	G3:	S→R**.**R | R**.** R→B | BR | RB B→0 | 1

G4:	S→L**.**L | L **.** | **.**L L→B | B L B→0 | 1	G5:	S→L**.**L | **.**L L→B | BL | LB B→0 | 1	G6:	N→L**.**L | L**.** L→B | L B B→0 | 1

G7:	N→L**.**R L→B | L B R→ B R | ε B→0 | 1	G8:	N→L**.**R L→B | B N R→B | R B | ε B→0 | 1	G9:	S→L**.**R | L**.** | **.**R L→B | B L R→B | R B| B→0 | 1

Для цепочки 110.011 постройте аннотированные деревья вывода в каждой грамматике с вычислением значения этой цепочки.

7.2.5. Для грамматики:

$S \rightarrow 0S11 \mid 1S00 \mid \varepsilon$

постройте атрибутную семантику, которая для каждой порожденной цепочки позволяет подсчитать: число единиц в цепочке и максимальную длину непрерывной цепочки единиц.

7.3.6. Язык формул логики высказываний для базиса $\{\neg, \Rightarrow\}$ описывается следующей грамматикой в БНФ нотации:

$\mu ::= p \mid (\neg\mu) \mid (\mu\Rightarrow\mu)$

где μ - начальный символ, а p – атомарное утверждение.

а) К какому классу относится эта грамматика?

б) Постройте атрибутную семантику этой грамматики для вычисления истинностного значения логических формул.

в) Вычислите истинностное значение формулы $((p_1\Rightarrow(\neg(p_2\Rightarrow p_3)))\Rightarrow(\neg p_1))$ по синтаксическому дереву при следующих значениях семантических атрибутов терминалов: (p_1.*val* = И; p_2.*val* = Л; p_3.*val* = И.

7.3. Примеры атрибутных трансляций

7.3.1. По заданной ниже атрибутной грамматике, порождающей вещественные десятичные числа, постройте дерево вывода входной цепочки 252.028 и покажите последовательность вычисления значений семантических атрибутов для этой цепочки:

N	Синтаксическое правило	Семантическое правило
1	$S \to A\ .\ B$	S.*val*:=A.*val* + B.*val*; B.p := -1;
2	$A \to \varepsilon$	A.*val* := 0; A.*p* := 0;
3	$A \to ц\ A_1$	A.*val* := *ц.val**10 ↑ A_1.*p* + A_1.*val*; A.*p* := A_1.*p* +1;
4	$B \to \varepsilon$	B.*val* :=0;
5	$B \to ц\ B_1$	B.*val* := *ц.val* *10 ↑ B.*p* + B_1.*val*; B_1.*p* := B.*p* - 1

7.3.2. Постройте атрибутную грамматику интерпретатора логических формул, представленных в базисе Буля. Примеры логических формул: ***not*** (A ***or not*** (B ***and*** C)), A ***and not*** B ***and*** C.

7.3.3. Грамматика формул логики высказываний обычно задается следующим образом:

$$\varphi \Rightarrow p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \Rightarrow \varphi \mid \varphi \Leftrightarrow \varphi$$

Покажите, что эта грамматика двусмысленная. Постройте недвусмысленную грамматику формул логики высказываний с учетом приоритетов. Приоритеты логических операций обычны: в порядке убывания приоритетов идут операция отрицания, затем операция конъюнкции, затем операция дизъюнкции. Приоритеты всех остальных бинарных логических операций одинаковы и меньшие, чем приоритет дизъюнкции. Скобки ‘(‘ и ‘)’ имеют обычное значение: они связывают подвыражения.
Постройте атрибутную семантику, позволяющую вычислять логическое значение любой правильно построенной формулы логики высказываний при известных истинностных значениях ее аргументов.

7.3.4. Схема мультиплексора (MUX), т.е. схема с тремя входами и одним выходом, реализует функцию ***ite***(*x, y, z*) = *if x then y else z*. Эта функция может быть записана и в конъюнктивном базисе: ***ite***(*x, y, z*) = *xy*∨¬*xz*. Схемно мультиплексор в некоторых технологиях очень просто реализуется. Известно, что множество функций {0, 1, ***ite***} составляет базис, т.е. любая логическая функция может быть выражена через константы 0,1 и функцию ***ite***. Например, ¬x=***ite***(x, 0, 1); x∧y=***ite***(x, y, 0); x∨y=***ite***(x, 1, y).
Постройте транслятор, который по булевой формуле, выраженной с помощью функций И, ИЛИ, НЕ, строит формулу в базисе {0, 1, ***ite***}. Такая формула будет основой для синтеза схемы, реализующей из мультиплексоров заданную логическую функцию.

7.3.5. Постройте атрибутную грамматику для трансляции заданной логической формулы представленной в базисе Буля с тесными отрицаниями (отрицания только у переменных) в соответствующую переключательную релейно-контактную схему

7.3.6. Постройте атрибутную грамматику для трансляции заданной логической формулы в базисе Буля в логическую формулу в базисе {***ite***, 0, 1}. Примеры логических формул на входе транслятора: ***not*** (A ***or not***(B ***and*** C)), A ***and not*** B ***and*** C.

7.3.7. Постройте атрибутную грамматику для трансляции заданной логической формулы в базисе Буля в логическую формулу в базисе Шеффера. Примеры логических формул на входе транслятора: ***not*** (A ***or not***(B ***and*** C)), A ***and not*** B ***and*** C.

7.3.8. Для грамматики, порождающей двоичные формулы с операциями {И, ИЛИ, НЕТ, $\Rightarrow$, $\oplus$}, постройте атрибутную семантику, позволяющую транслировать логическую формулу в формулу в дизъюнктивном базисе {ИЛИ, НЕТ}. Покажите, как будет выполняться трансляция на дереве вывода формулы $(x \oplus y) \Rightarrow z$.

7.3.9. Для грамматики, порождающей двоичные формулы с операциями {И, ИЛИ, НЕТ}, постройте атрибутную семантику, позволяющую транслировать логическую формулу в базис Жегалкина (т.е. в базис {$\oplus$, И, 1}). Покажите, как будет выполняться трансляция на дереве вывода формулы $\neg(x \vee y) \wedge z$.

7.3.10. Для грамматики, порождающей двоичные формулы в базисе Жегалкина (т.е. в базисе {$\oplus$, И, 1}), постройте атрибутную семантику, позволяющую транслировать любую логическую формулу в базис Буля {И, НЕТ}. Покажите, как будет выполняться трансляция на дереве вывода формулы $1 \oplus x \oplus y \wedge z$.

7.3.11. Постройте атрибутную грамматику для трансляции заданной логической формулы в базисе Буля в программу стековой машины, в которой определены команды AND, OR, NOT. Примеры логических формул: ***not*** (A ***or not***(B ***and*** C)), A ***and not*** B ***and*** C.

7.3.12. Для темпоральных формул логики линейного времени LTL постройте атрибутную семантику, которая преобразует введенную темпоральную формулу к темпоральной формуле только с темпоральными операторами **X** и **U**, позволяя замены темпоральных операторов **F** и **G** в соответствии с правилами вывода $\mathbf{F}\varphi = \mathit{True}\mathbf{U}\varphi$, $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$.

7.3.13. Пара скобок в арифметическом выражении называется избыточной, если после ее удаления выражение остается эквивалентным исходному. Постройте атрибутную грамматику, которая позволяет удалять все избыточные пары скобок арифметических выражений.

7.3.14. Постройте атрибутную грамматику, которая по формуле двоичной функции в инфиксной нотации строит эквивалентную формулу:

а) в префиксной нотации;

б) в постфиксной нотации (ПОЛИЗ).

7.3.15. Постройте грамматику, порождающую описания переменных вида:

integer a, b, c;
real d, e;

и постройте семантические атрибуты и атрибутную семантику, позволяющие присвоить каждой переменной в таблице имен свой тип. Покажите семантические вычисления на дереве синтаксического анализа цепочек ***real*** d, e; ***integer*** a, b, c;.

7.3.16*. Для грамматики, порождающей дробные десятичные числа без знака, ниже приведено представление атрибутной грамматики.

ALPHABET
Num :: float V.
Int :: float V; int P.
Frac :: float V; int P.
digit :: int Val.

RULE Num ::= Int '.' Frac
SEMANTICS V<0> = V<1>+V<3>; P<3>=1.

RULE Int ::= e
SEMANTICS V<0> = 0; P<0>=0.

RULE Int ::= digit Int
SEMANTICS V<0> = Val<1>*10**P<2>+V<2>; P<0> = P<2>+1.

RULE Frac ::= e
SEMANTICS V<0> = -1.

RULE Frac ::= digit Frac
SEMANTICS V<0> = Val<1>*10**P<0>+V<2>; P<2> = P<0>-1.

Постройте грамматику метаязыка, на котором описана эта атрибутная грамматика.

7.4. Трансляция арифметических выражений

7.4.1. Переведите в ПОЛИЗ арифметическое выражение a+b*(c+d)/e +(f - d).

7.4.2. С помощью стека сгенерируйте последовательность команд стековой машины для вычисления значения арифметического выражения c*(c+d)/e +(f - d), предварительно представленного в ПОЛИЗ.

7.4.3. Представьте в ПОЛИЗ арифметическое выражение
(a+b)*c-(c+a)/(e+f).
Сгенерируйте последовательность команд стековой машины для вычисления значения этого выражения по его представлению в ПОЛИЗ.

7.4.4. Синтаксическим графом арифметического выражения называется направленный граф, полученный из синтаксического дерева для арифметического выражения объединением всех вершин, соответствующих совпадающим подвыражениям. Такой граф позволяет эффективно вычислять значения арифметических выражений и строить оптимальные компиляторы.

Постройте атрибутную грамматику, которая позволяет представить арифметическое выражение синтаксическим графом.

7.4.5. В функциональных языках, например, в языке LISP, арифметические выражения записываются в префиксной форме, а именно <операция> (<операнд1> <операнд2>) для бинарных операций. Например, выражение a – b*c в префиксной форме будет записано так: - (a (* (b c))). Постройте грамматику, порождающую записи арифметических выражений в префиксной форме языка LISP, и определите семантические атрибуты, генерирующие программу для стековой машины, вычисляющую значение таких арифметических выражений.

7.4.6. Постройте атрибутную грамматику арифметических выражений, содержащих как целые, так и вещественные операнды. Семантические атрибуты должны позволить вычислить значение выражения правильного типа. Тип операнда и его значения известны (они являются семантическими атрибутами). Если типы операндов арифметической операции не совпадают, то целый операнд должен быть приведен к вещественному типу. Результатом арифметической операции, в которой тип хотя бы одного из операндов вещественный, должен быть вещественного типа. Команды для выполнения операций над целыми и вещественными различаются.

7.4.7. Грамматика, описывающая все формулы линейной темпоральной логики LTL обычно строится следующим образом:

$$\varphi \Rightarrow p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \Rightarrow \varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

Покажите, что эта грамматика двусмысленная. Постройте недвусмысленную грамматику формул темпоральной логики LTL с учетом приоритетов. Темпоральные операторы имеют наивысший приоритет, а приоритеты логических операций обычные. Скобки ‘(‘ и ‘)’ имеют обычное значение: они связывают подвыражения.

7.4.8. Темпоральным операторам **X, U, F** и **G** в формулах темпоральной логики ветвящегося времени CTL непосредственно предшествуют кванторы существования и всеобщности. Грамматика, порождающая формулы этой логики:

$$\varphi \Rightarrow p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \Rightarrow \varphi$$
$$\mid \mathbf{AX}\varphi \mid \mathbf{EX}\varphi \mid \mathbf{A}(\varphi\mathbf{U}\varphi) \mid \mathbf{E}(\varphi\mathbf{U}\varphi) \mid \mathbf{AG}\varphi \mid \mathbf{EG}\varphi \mid \mathbf{AF}\varphi \mid \mathbf{EG}\varphi$$

Покажите, что эта грамматика двусмысленная. Постройте недвусмысленную грамматику формул логики CTL с учетом приоритетов. Темпоральные операторы имеют наивысший приоритет, а приоритеты логических операций обычные. Скобки ‘(‘ и ‘)’ имеют обычное значение: они связывают подвыражения.

8. Распознаватели подклассов КС-языков

8.1. s-грамматики

8.1.1. Пусть задана в s-грамматика:

S→ aSRb | bRcA
R→ bA | aR
A→aAb | c

в которую семантические действия добавлены так:

S→‘(’ aSRb ‘)’ | ‘(’ bRcA ‘)’
R→‘(’ bA ‘)’ | ‘(’ aR ‘)’
A→‘(’ aAb ‘)’ | ‘(’ c ‘)’

здесь ‘(‘ и ‘)’ являются ‘семантическими’ символами, не влияющими на ход синтаксического анализа. При синтаксическом анализе пусть каждый вытолкнутый из стека символ (нетерминалы, терминалы и семантические символы) также выводится. Покажите, какое дерево вывода в скобочной записи будет построено в результате обработки терминальной строки abccbcbb.

8.1.2. Известно, что проблема однозначности КС-грамматик алгоритмически неразрешима. Можно ли утверждать, что КС-грамматика:

S→aSRbcA | bRA
R→bA | cRAR | aRA
A→aAb | c

является однозначной? Почему?
Провести синтаксический анализ цепочки bbcabb, порожденной в этой грамматике (возможно, с синтаксическими ошибками)

8.1.3. Можно ли привести следующую грамматику к эквивалентной s-грамматике?

S → aS | aA
A → bA | aB | aC
B → bA | b
C → ε | bA

Если можно, то проведите в этой s-грамматике распознавание цепочки aababba.
Указание. Эта грамматика – автоматная.

8.1.4. Можно ли привести следующую грамматику к эквивалентной s-грамматике?

S → abSba | acAb
A → bbAa | bcB | a
B → bBA | c

Если можно, то проведите в этой s-грамматике распознавание цепочки acbbbccab.

8.1.5. Для следующей s-грамматики:

S'→ begin S end

S → if E then S else S | S → begin S L | S → print num

L → end | ; S L

E → num = num

выполните синтаксический анализ цепочки:

begin if num=num then print num ; else begin print num end end

Указание. Здесь все терминалы – лексемы входного языка.

8.2. LL(k)-грамматики

8.2.1. Существуют ли грамматики класса LL(0)? Если да, то постройте пример такой грамматики.

8.2.2. Постройте множества FIRST и FOLLOW для каждого нетерминала следующих КС-грамматик:

S→aAB| BA A→BB | a B→AS | b	S→ABC A→BB | ε B→CC | a B→CC | b	S→S+T| T T→S(S) | a	S→aAB| B A→aA | a B→BS | A | b	S→begin D; R end | s D→D;d | d R→R; S | S

8.2.3. Постройте КС-грамматику, порождающую язык L={abbabb, abbb}. При каком k эта грамматика является LL(k)-грамматикой?

8.2.4. Известно, что проблема однозначности КС-грамматик алгоритмически неразрешима. Можно ли утверждать, что КС-грамматика:

S→aSRd | RA

R→bA | cRA | ε

A→fAb | d

является однозначной? Почему?

8.2.5. При каком значении k следующая грамматика является LL(k)-грамматикой?

S →□aAaa | □bAba

A →□b | □ε

8.2.6. Является ли нижеследующая грамматика s-грамматикой? Является ли она LL(1)-грамматикой? Постройте управляющую таблицу LL-анализатора и проведите синтаксический анализ цепочки abbab в этой грамматике.

S →□aBS | b

A →□a

B →□A | □bSB

8.2.7. Постройте LL(1) грамматику и синтаксический анализатор для языка, включающего строки (программы) такого вида:

for x:= m *to* n *do*
  begin
    x := 3 + x;
    y :=5
  end;

8.2.8. Для грамматики:

S’ →□S $
S →□A*a*S | *b*
A →□CA*b* | B
B →□*c*S*a* |□ε
C →□*c* |□*ab*

постройте множества выбора для альтернатив каждого нетерминала.

8.2.9. Проверьте, является ли следующая грамматика LL(1)-грамматикой:

G = { S’ → S$, S → B A, A → +B A | ε, B → DC, C → *DC | ε, D → (S) | *a* }

построив множества выбора для альтернатив каждого нетерминала.

8.2.10. Грамматику:

G = { S →*a*T | TbS, T → *b*T | *ba* }

приведите к виду LL(1), проведя факторизацию. Проведите анализ цепочки *bbababba* в построенной грамматике.

8.2.11. Проверьте, является ли следующая грамматика LL(1)-грамматикой:

S’ → S$
S → A*b*B | *d*
A → C*ab* | B
B → *c*S*d* | ε
C → *a* | *ed*

построив множества выбора для альтернатив каждого нетерминала.

8.2.12. Можно ли привести грамматику G к эквивалентной LL(1)-грамматике?

G = {S → BA*b*, A → *a*ABC | *b*B | *a*, B → *b*, C → *c*A }.

Проведите в этой LL(1)-грамматике распознавание цепочки *babccaab*.

8.2.13. Покажите, что следующая грамматика с начальным символом <программа> является LL(1)-грамматикой:

<программа> → <оператор>
<оператор> → begin <описание> : <список операторов> end
<описание> → d | d ; <описание>
<список операторов> → s ; <список операторов> | ε

Постройте таблицу принятия решений для этой грамматики. Выполните синтаксический анализ цепочки:

begin d; d : s ; s; s; end

8.2.14. Покажите, что следующая грамматика является LL(1)-грамматикой:

{S→BA, A→BS | *d*, B→*a*A | *b*S | *c*}.

Постройте таблицу для принятия решений при синтаксическом анализе цепочек, порождаемых этой грамматикой. Выполните синтаксический анализ цепочки *bcdcadd*.

8.3. Грамматики рекурсивного спуска

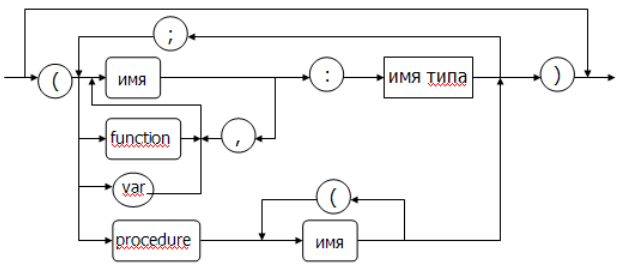
8.3.1. Постройте программу, которая вводит поток телеграмм, и для каждой телеграммы подсчитывает и печатает число слов, стоимость телеграммы (при известной стоимости слова), число телеграмм в потоке и общую стоимость телеграмм. Слово определяется, как последовательность букв, заканчивающаяся пробелом, телеграмма – как последовательность слов, заканчивающаяся маркером конца '#', поток телеграмм – как последовательность телеграмм, заканчивающаяся символом конца текста *eot*.

8.3.2. Постройте синтаксическую диаграмму и семантические действия для генерации команд стековой машины для расширения языка Милан условным оператором присваивания вида *a:= **if** b>c **then** e*f **else** a+b **fi***.

8.3.3. Постройте синтаксическую диаграмму и семантические действия для генерации команд стековой машины для расширения языка Милан оператором цикла языка Java вида ***do*** { <список операторов> } ***while*** (<условие>). При необходимости введите дополнительные команды стековой машины.

8.3.4. Постройте синтаксическую диаграмму и семантические действия для генерации команд стековой машины для расширения языка Милан оператором цикла языка Java вида ***for*** (<инициализация>; <условие>; <итерация>) { < список операторов > }. При необходимости введите дополнительные команды стековой машины.

8.3.5. По синтаксической диаграмме для нетерминала <Список_параметров> языка Паскаль постройте процедуру на С, распознающую все цепочки подъязыка, порождаемого этим нетерминалом.



8.3.6. По КС-грамматике:

S→*a*SR*d* | RA
R→*b*A | *c*RA | ε
A→*a*A*b* | *d*
постройте синтаксические диаграммы. Является ли эта грамматика грамматикой рекурсивного спуска? Почему? Постройте по синтаксическим диаграммам распознающие процедуры входного языка.

8.3.7. Преобразуйте следующую КС-грамматику:
S'→ #S#
S → *ab*AR*d* | *abb*
R → R ; A | *b*
A → *a*A*b* | *d*
к грамматике рекурсивного спуска. Постройте синтаксические диаграммы этой новой грамматики и по ним соответствующие распознающие процедуры. Выполните по построенным синтаксическим диаграммам разбор цепочки #a*bddb;d*#.

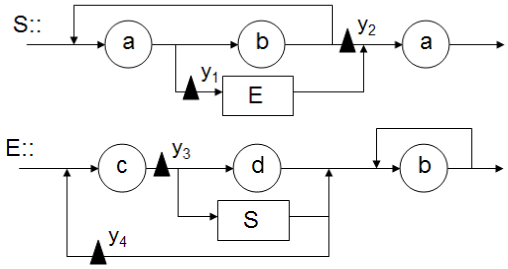
8.3.8. Покажите, что следующая КС-грамматика с начальным нетерминалом Program, записанная в БНФ:
Program ::= Statement '.' | Statement ';' Program
Statement ::= Name '=' Expression
Expression ::= Expression Operator Expression | Name | '{' Elements '}' | '(' Expression ')'
Operator ::= '∩' | '∪'
Name ::= 'A' | 'B' | …| 'X' | 'Y' | 'Z'
Elements ::= Element | Element ',' Elements
Element ::= 'a' | 'b' | … | 'x' | 'y' | 'z'
является двусмысленной. Постройте эквивалентную недвусмысленную грамматику, в которой выполнение операций '∩' и '∪' определено слева направо без учета приоритетов.

8.3.9. Преобразуйте следующую грамматику с начальным нетерминалом Program, записанную в БНФ:
Program ::= Statement '.' | Statement ';' Program
Statement ::= Name '=' Expression
Expression ::= Expression Operator Expression | Name | '{' Elements '}' | '(' Expression ')'
Operator ::= '∩' | '∪'
Name ::= 'A' | 'B' | …| 'X' | 'Y' | 'Z'
Elements ::= Element | Element ',' Elements
Element ::= 'a' | 'b' | … | 'x' | 'y' | 'z'
к грамматике рекурсивного спуска, в которой операции '∩' и '∪' выполняются слева направо без учета приоритета. Постройте синтаксические диаграммы этой новой грамматики и по ним соответствующие распознающие процедуры. Выполните по построенным синтаксическим диаграммам разбор цепочки:
M={a , b}; N=M ∪ {d , a, b}∩ M.

8.3.10. Для грамматики
G={S'→*begin* S *end*, S→*if* E *then* S *else* S | *begin* SL | *print* E,L → *end* | ;SL, E→i= i}
постройте распознаватель методом рекурсивного спуска.

8.3.11. Постройте распознаватель методом рекурсивного спуска для грамматики
G = { S'→S$, S → (B)B, B → (B)B | ε }.

8.3.12. Для грамматики, представленной синтаксическими диаграммами с включенными семантическими действиями, постройте компилятор: процедуры распознавания, дополненные семантическими операциями:



8.3.13. На языке Милан++ построена программа суммирования чисел из входного потока:

```
begin
    i,s := 0,0;
    n := read;
    loop
        i < n -> s,i := s+read,i+1
    endloop;
    write (s)
end
```

а) Постройте программу стековой машины для выполнения этой программы.
б) Постройте синтаксические диаграммы и семантические операции языка Милан ++ для транслятора рекурсивного спуска.

8.3.14. На языке Милан++ построена программа подсчета факториала:

```
begin
    read x;
    if x>0 then
          fact := 1;
          repeat
          fact *= x;
          x –
    until x = 0;
             write (fact)
end
```

а) Постройте программу стековой машины для выполнения этой программы.
б) Постройте синтаксические диаграммы и семантические операции языка Милан ++ для транслятора рекурсивного спуска.

8.3.15. Существует мнение, что формулы линейной темпоральной логики LTL, используемые в верификации ПО и аппаратуры, не очень понятны разработчикам программ. В некоторых системах верификации спецификация требований к поведению программ задается на ограниченном естественном английском языке PSL (Property Specification Language). Примеры таких спецификаций на упрощенном языке PSL:

Цепочка языка PSL	Соответствующая формула LTL
assert always ! (a⇒ ***next*** b)	**G**¬(a⇒**X**b)
assert never (a & b)	**G**¬(a&b)
assert always (req ⇒ ***next*** (busy ***until*** done))	**G**(req ⇒ **X**(busy **U** done))
assert always (a ⇒ ***next*** b)	**G**(a⇒**X**b)
assert eventually (b ⇒ ***next*** (***always*** ! b))	**F**(b ⇒ **X** (**G** ¬b))

Постройте грамматику упрощенного языка PSL и транслятор с него в язык формул линейной темпоральной логики.

8.3.16. Для грамматики простого языка спецификации свойств программ:

G: S → ***assert*** Φ

Φ → P | (Φ) | ***not*** Φ | (Φ ***or*** Φ) | (Φ ***and*** Φ) | (***if*** Φ ***then*** Φ) |
always Φ | ***next*** Φ | ***eventually*** Φ | Φ ***until*** Φ

P → a | b | c | ... | x | y | z

постройте транслятор рекурсивного спуска (синтаксические диаграммы и семантические операции) для перевода фраз ограниченного естественного языка в формулы темпоральной логики LTL. Например, по тексту:

assert always (***if not*** a ***then eventually*** (b ***until*** (c ***or next*** d))

должна быть выдана следующая формула темпоральной логики LTL:

assert **G**(¬a ⇒ **F** (b **U** (c∨ **X**d))),

8.4. Грамматики простого предшествования

8.4.1. Постройте матрицу отношений предшествования для грамматики:

0. S'→#S#
1. S →aST
2. S →a
3. S →[S]
4. T→b
5. T→cT

Восстановите правый вывод цепочки #[a a [a] cbb]# в этой грамматике.

8.4.2. Выполните синтаксический анализ цепочки #aababdcdccc# в грамматике:

S'→#S#
S →aSc | bBc
B →baSd|d

методом простого предшествования.

8.4.3. Выполните синтаксический анализ цепочки #abadaccac# в грамматике:

S'→#S#
S →aBSc | a
B →bSc | d

методом простого предшествования.

8.4.4. Выполните синтаксический анализ цепочки # [(((a a) a) a)] # в грамматике:

S’→#S#
S →[M]
M →(L | a
L → Ma)

методом простого предшествования.

8.4.5. Выполните синтаксический анализ цепочки #abababcb# в грамматике:

S’→#S#
S →aASc|ba
A →baSc

методом простого предшествования.

8.4.6. Выполните синтаксический анализ цепочки #((cc)c)# в грамматике:

S’→#S#
S →(SS) | c

методом простого предшествования.

8.4.7. Выполните синтаксический анализ цепочки #(((aa)a)a)# в грамматике:

S’→#S#
S→(R | a
R→S a)

методом простого предшествования.

8.4.8. Постройте матрицы отношений предшествования для грамматик:

G1 = { S’→#S#, S → aSbb | abb }
G2 = { S’→#S#, S → aSAb | aSb, A→b }
G3 = { S’→#S#, S → if B then S else S | s, B → B or b | b }
G4 = { S’→#S#, S → SA | A, A → (S) | ( ) }
G5 = { S’→#S#, S → AS | A, A → (S) | ( ) }
G6 = { S’→#S#, S → aRSd | c, R → Rb | b }
G7 = { S’→#S#, S → SaSb | c }
G8 = { S’→#S#, S → aSSb | c }
G9 = { S’→#A#, A → (A) | a }

8.4.9**. Разработайте программу-конструктор, автоматизирующую процесс синтаксического анализа цепочек языков, порождаемых грамматиками простого предшествования. Программа должна выполнять следующие функции:

1) ввод произвольной грамматики;
2) формирование для нее матрицы простого предшествования;
3) проверка того, что введенная грамматика удовлетворяет требованиям грамматики простого предшествования;
4) выполнение распознавания входных цепочек.

8.5. LR-грамматики

8.5.1. Для грамматики:

{ S'→#S#, S→ aBc, B → Bb | b }

постройте LR(0) анализатор и восстановите вывод цепочки #abbbc#.

8.5.2. Для грамматики { S→#A#, A→ (A) | a } постройте самый простой возможный LR анализатор и восстановите вывод цепочки #((a))#. Для той же грамматики постройте нисходящий анализатор и восстановите вывод этой цепочки с помощью этого анализатора.

8.5.3. Правильно построенные списки рекурсивно определяются так:

i) любой элемент p есть список;

ii) если A и B – списки, то [A] и A;B – тоже списки;

iii) других списков нет.

а) Постройте недвусмысленную КС-грамматику Хомского, порождающую все списки.

б) Постройте LR(0)-анализатор для этой грамматики и выполните с его помощью синтаксический анализ следующих цепочек:

1) [a;[b]; c] 2) a; [[a;b]; c];[d] 3) [[a; c];[d]

8.5.4. Для грамматики, порождающей скобочные скелеты:

S'→#S#

S → S (S) | ε

постройте самый простой LR анализатор и восстановите правый вывод цепочки
(( ) ( )) ( ) #.

8.5.5. Постройте LR(0) анализатор для грамматики:

S'→#S#

S →aST | [Sb] | a

T → cT | b

Восстановите правый вывод цепочки #[a b [abc] a]# в этой грамматике.

8.5.6. Для грамматики:

{ S'→#S#, S →dAb | Bb, B →cd, A →cBA|d}

постройте самый простой LR анализатор и восстановите правый вывод цепочки #dccdccddb#

8.5.7. Для грамматики:

S'→#E#

E →(L) | *i*

L → L , E | E

постройте LR(1)-распознаватель и SLR(1)-распознаватель. Выполните синтаксический анализ цепочки #((i), i, (i, i))# с помощью каждого из двух этих распознавателей.

8.5.8. Для грамматики:

{ S’→#S#, S →BSb | ε, B →bBca | c }

постройте самый простой LR анализатор и восстановите правый вывод цепочки #accaccbb#

8.5.9. Для грамматики:

S’→#S#

S →aSBa | [S] | a

B →bB| b

постройте LALR(1) анализатор и восстановите правый вывод цепочки #[abb[a]]#

8.5.10. Для грамматики:

S →ac | bAc | Aa

A → a

постройте LALR(1) анализатор и восстановите правый вывод цепочки bac. Какой язык порождает эта грамматика?

8.5.11. Для грамматики:

S’→#S#

S →SAb | b | ε

A →Ab| b

постройте SLR(1) анализатор и восстановите правый вывод цепочки #abbabbb#

8.5.12. Для грамматики:

S→#E#

E →E - T | T

T →i | (E)

постройте самый простой LR анализатор и восстановите правый вывод цепочки #*i*-(*i*-*i*-*i*)-*i*#

8.5.13. Для грамматики:

S’→#S#

S →A B c

A →a

B →b | ε

постройте LR(1) анализатор и восстановите правый вывод цепочки #*abc*#. Какой язык порождает эта грамматика?

8.5.14. Для грамматики:

S’→#S#

S →aAa | aBb| bAb | bBa

A →c

B →c

постройте LALR(1) анализатор и восстановите вывод цепочки #*acb*#. Какой язык порождает эта грамматика?

8.5.15. Для грамматики:

{S’ → #S#, S → ac | aBbc, B → Bb | ε}

постройте LALR(1) анализатор и восстановите вывод цепочки #*abbbc*#.

8.5.16. Для грамматики:

{S→#B#, B→E=E, B→*i*, E→E+*i*, E→E-*i*, E→*i*}

постройте простейший LR- анализатор и восстановите вывод цепочки # *i*+*i* = *i* - *i*+*i* #.

8.5.17. Для грамматики:

{ S’→# E #, E →E -T | T, T →i | (E) }

постройте LALR(1) анализатор и восстановите вывод цепочки # *i* - (*i* – *i*) #.

8.5.18. Для грамматики:

{ S→#E#, E →(L) | a, L →E L | E }

постройте LALR(1) анализатор и восстановите правый вывод цепочки #((a) a (a a))#. Какой язык порождает эта грамматика?

8.5.19. Покажите что грамматика:

{ S’→#S#, S → aAd | bBd | aBe | bAe, A → c, B → c }

является LR(1) грамматикой, но не LALR(1) грамматикой. Для этого постройте LALR(1)-анализатор и LR(1)-анализатор, и покажите, что первый имеет неразрешимые конфликты, а во втором конфликтов нет.

8.5.20. Покажите, что грамматика:

{ S→#A#, A → aAa | ε }

не является LR(k) грамматикой при любом k (покажите это построением LR(0), LR(1) и LR(2)-анализаторов). Является ли она двусмысленной?

8.5.21*. Для грамматики:

S’→#S#

S →***if*** b ***then*** S ***fi*** | ***if*** b ***then*** S ***else*** S ***fi*** | *on*

постройте LR(0)-распознаватель. Существуют ли коллизии в LR(0)-автомате? Какое дерево вывода в этой грамматике будет у цепочки:

if b ***then if*** b ***then if*** b ***then*** *on* ***fi else if*** b ***then if*** b ***then*** *on* ***fi else*** *on* ***fi***

(***Указание***. При построении распознавателя всю терминальную цепочку ‘***if*** b ***then***’ удобно обозначить одним терминальным символом ***if***).

8.5.22. Для двусмысленной грамматики:

S’→#E#

E →E-E | *i*

постройте LALR(1)-распознаватель. Существуют ли в нем коллизии?

а) Если при конфликте типа “перенос/свертка” всегда выполнять свертку, к чему это приведет при синтаксическом анализе? Какое дерево вывода при таком разборе будет у цепочки:

 # i – i – i – i #

б) Провести такой же анализ при условии, что при конфликте типа “перенос/свертка” всегда выполняется перенос.

8.5.23. Дана двусмысленная грамматика:

S →#E#

E →E+E | E*E | *i*

а) Постройте для нее LALR(1)-распознаватель.

б) Можно ли определить правила разрешения коллизий в этом распознавателе, чтобы операция ‘*’ была более приоритетна, чем операция ‘+’, а операции одного приоритета выполнялись бы слева неправо?

в) Постройте с помощью этого анализатора анализ цепочки

i + *i* * *i* * *i* + *i* + *i* #.

8.5.24*. Для грамматики:

S’→#S#

S →***if*** b ***then*** S | ***if*** b ***then*** S ***else*** S | *on*

постройте SLR(1)-распознаватель. Существуют ли коллизии в SLR(1)-автомате?

а) Если при конфликте типа “перенос/свертка” всегда выполнять свертку, к чему это приведет при синтаксическом анализе? Какое дерево вывода при таком разборе будет у цепочки:

if b ***then if*** b ***then if*** b ***then*** *on* ***else if*** b ***then if*** b ***then*** *on* ***else*** *on*

б) Провести такой же анализ при условии, что при конфликте типа “перенос/свертка” всегда выполняется перенос.

(***Указание***. При построении распознавателя всю терминальную цепочку ‘***if*** b ***then***’ удобно обозначить одним терминальным символом ***if***).

8.5.25*. Для грамматики:

S’→#S#

S →***if*** b ***then*** S | ***if*** b ***then*** S1 ***else*** S | *on*

S1 →***if*** b ***then*** S1 ***else*** S | *on*

постройте SLR(1)-распознаватель.

а) Существуют ли коллизии в SLR(1)-автомате?

б) Какое дерево вывода в этой грамматике будет у цепочки:

if b ***then if*** b ***then if*** b ***then*** *on* ***else if*** b ***then if*** b ***then*** *on* ***else*** *on*

(***Указание***. При построении распознавателя всю терминальную цепочку ‘***if*** b ***then***’ удобно обозначить одним терминальным символом ***if***).

8.5.26. Двусмысленная грамматика:

S’→# stmt #

stmt → s | stmt ; stmt

порождает цепочки из операторов s, разделенных точками с запятой (здесь stmt – нетерминал, от слова statement, оператор). Постройте для этой грамматики SLR(1)-распознаватель. Существуют ли коллизии? Как будет выполняться анализ цепочки # s; s; s; s; s #, если коллизии в SLR(1)- распознавателе решать в пользу свертки? Как будет выполняться анализ этой цепочки, если коллизии решать в пользу переноса?

8.5.27. Двусмысленная грамматика:

$S' \to \# S \#$

$S \to SS \mid (S) \mid \varepsilon$

порождает скобочные формы. Постройте для этой грамматики LR(1)-распознаватель. Существуют ли коллизии? Как будет выполняться анализ цепочки # (( )) ( ) (( )) #, если коллизии в LR(1)- распознавателе решать в пользу свертки? Как будет выполняться анализ этой цепочки, если коллизии решать в пользу переноса?

8.5.28. Является ли следующая грамматика логических формул недвусмысленной?

$G=\{ S' \to S\$, S \to p \mid (S \vee S) \mid (S \wedge S) \mid (S \Rightarrow S) \mid \neg S \}$

К какому классу она относится? Постройте для этой грамматики распознаватель.

8.6. Универсальные алгоритмы распознавания КС языков

8.6.1. Для двусмысленной грамматики:

$S \to SS \mid (S) \mid ( )$

постройте два дерева вывода цепочки ( ) (( )) ( ) с помощью алгоритма Кока-Янгера-Касами.

8.6.2. Для двусмысленной грамматики:

$S \to S ; S \mid on$

порождающей последовательности операторов, разделенные точками с запятой, постройте два дерева вывода цепочки

on ; *on* ; *on*

с помощью алгоритма Эрли.

9. Трансляция конструкций языков программирования

9.1. Контекстно-зависимые элементы в языках программирования

9.1.1. Чему станет равна переменная b в результате выполнения следующего фрагмента кода на языке C?

```
int b = 2
/*ptrx ;
b = 3;        /***  переменная b изменяет свое значение    ***/
/****
      ptrx объявлен ранее как указатель
      на переменную x, которая равна 2
****/;
```

9.1.2. Постройте грамматику, порождающую описания переменных вида:

integer a, b, c;

real d, e;

и постройте семантические атрибуты, позволяющие присвоить каждой переменной в таблице имен свой тип. Покажите семантические вычисления на дереве синтаксического анализа цепочки ***integer*** a, b, c;.

9.1.3. Целые константы без знака в языке C определяются так:

десятичные: <последовательность цифр, начинающаяся не с нуля>

восьмеричные: 0<последовательность цифр от 0 до 7>

шестнадцатеричные: 0 {X, x} <последовательность цифр 0-9 и букв a-f или A-F>

Постройте грамматику, синтаксическую диаграмму распознавателя и транслятор языка целых констант C.

9.1.4. Десятичные константы с плавающей точкой в языке C имеют следующий формат: целая часть, десятичная точка, дробная часть, экспонента, начинающаяся с e либо E, например: 2.34E-23, 17.344, 2E-2, .34e3. Либо целая, либо дробная часть константы может быть опущена, но не обе сразу. Либо десятичная точка с дробной частью, либо экспонента могут быть опущены, но не обе сразу. Знаки ‘+’ и ‘-‘ после знаков e и E могут присутствовать или отсутствовать.

а) Постройте описание формата такой константы с помощью грамматики Хомского.

б) Постройте описание этого формата в БНФ нотации.

в) Постройте описание этого формата в виде синтаксической диаграммы.

г) Постройте описание этого формата в виде конечного распознающего автомата.

д) Постройте транслятор, переводящий любую константу без знака во внутреннее представление.

9.1.5. В некоторых языках программирования (например, в языке Modula-2) описание процедуры имеет следующий вид:

PROCEDURE Proc;

BEGIN

...

END Proc;

Завершается процедура ключевым словом **END**, после которого идет имя этой процедуры. Постройте КС-грамматику, порождающую описания процедур языка Modula-2.