Учебно-методические пособия Учебно-научного центра ОИЯИ Дубна

УНЦ-2011-47

В. Б. Злоказов

ТЕОРИЯ АВТОМАТОВ

2011

УЧЕБНО-НАУЧНЫЙ ЦЕНТР ОИЯИ

В. Б. Злоказов

ТЕОРИЯ АВТОМАТОВ

Учебно-методическое пособие

Дубна 2011

368

Злоказов В. Б.

368 Теория автоматов: Учебно-методическое пособие. — Дубна: ОИЯИ, 2011. — 69 с.

ISBN 978-5-9530-0288-2

Настоящее учебно-методическое пособие предназначено прежде всего для студентов, обучающихся по специальности «Вычислительные машины, комплексы, системы и сети», и в компактной форме содержит следующий необходимый учебный материал, рекомендованный для курса «Теория автоматов» по данной специальности: формальная (булева) алгебра, теория абстрактных автоматов и алгоритмов, конкретные примеры автоматов и систем автоматов, примеры простейших микропроцессоров — дискретных преобразователей информации В. М. Глушкова, теория формальных грамматик.

Пособие снабжено примерами и задачами и содержит список дополнительной литературы по данному курсу.

Zlokazov V. B.

Theory of Automata: Manual. — Dubna: JINR, 2011. — 69 p.

The manual is intended to help, first of all, the students of the speciality «Computers, Computer Complexes, Systems and Nets», and contains in a compact form all the necessary training material, recommended for the course «Theory of Automata» of this speciality: formal (Boolean) algebra, theory of abstract automata and algorithms, concrete instances of automata and automaton systems, instances of simple microprocessors — Glushkov discrete information processors, and the theory of formal grammars.

The manual is supplied by examples and exercises, and contains a list for supplementary reading on this course.

ISBN 978-5-9530-0288-2

Целью изучения дисциплины "Теория автоматов" является получение студентами начальных теоретических знаний о формальных языках и дискретных автоматах, изучение элементов теории конечных автоматов, основных этапов абстрактного и структурного синтеза конечных автоматов, элементов теории формальных грамматик, получение представлений об актуальных задачах теории автоматов и соответствии классов языков и моделей автоматов.

Знания, полученные студентами в ходе освоения дисциплины, требуются для изучения циклов специальных дисциплин: операционные системы, схемотехника, моделирование, технология программирования, конструирование ЭВМ, искусственный интеллект.

Для изучения настоящей дисциплины от студентов требуются общематематические знания и знания по таким дисциплинам, как дискретная математика, информатика и языки программирования, но в первую очередь формальная (или математическая) логика. Именно с нее мы и начнем наш курс.

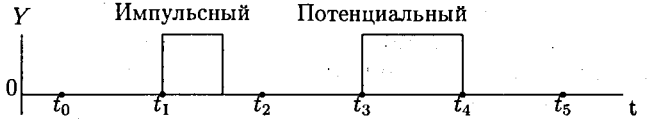
МАТЕМАТИЧЕСКАЯ ЛОГИКА

Математическая (формальная, булева и т.д.) логика - это формализация обычной классической логики, в которой законы правильного мышления перенесены на операции с моделями произвольных объектов и отношений между ними. Соответствие между базисными понятиями классической и математической логики следующее. Предложениям (высказываниям) классической логики соответствуют абстрактные переменные математической, а значениям "истина", "ложь" предложений - значения "1", "0" переменных. Впрочем, термины "истина", "ложь" употребляются и в формальной логике как своеобразные терминологические реликты. В формальной логике существуют и обобщения логических переменных с двоичных на n-значные. Для вводного курса в автоматные системы наибольший интерес представляет двоичная формальная логика.

ИСЧИСЛЕНИЕ ПРЕДЛОЖЕНИЙ - классическая логика начинается именно с него. Простым примером является атомарное (нерасчленимое на другие предложения) высказывание. Например, имеем два высказывания "2 четное число" и "2 x 2 = 5"; первое является истинным, второе ложным. Их истинность и ложность были установлены анализом содержания этих высказываний. Напротив, в формальной логике, где роль таких высказываний играют простые логические переменные p или q, их логические значения постулируются.

В автоматике эти логические переменные используются для обозначения сигналов (электрических, пневматических и т.д.). При этом высокий уровень сигнала в рамках простейшей двоичной классификации обозначается как "1", низкий - как "0". Сигналы циркулируют во времени. Так как основным режимом работы устройств автоматики является дискретный, то соответствующий интервал времени, в котором работает то или иное устройство, делится на такты - временные промежутки, в которые происходят всевозможные воздействия сигналов на устройство. Сигналы при этом могут иметь либо импульсную форму, если длительность воздействия сигнала меньше длительности тактового интервала, либо потенциальную, если

значение сигнала не меняется в течение всего тактового промежутка. Графически можно изобразить это следующим образом:



От логических переменных могут быть построены логические функции, принимающие тоже лишь два значения: 0 или 1. Поскольку число значений переменных невелико, всевозможные функции являются лишь различными комбинациями нулей и единиц на наборах из n значений переменных. Число таких наборов равно 2^n, и, следовательно, число всевозможных комбинаций нулей и единиц будет равно K - максимальному значению двоичного числа, имеющего 2^n разрядов, т.е. 2 в степени 2^n.

Для $n = 1$, т.е. для функции одной переменной, $K = 4$. Мы можем записать их все в форме таблиц, пользуясь тем, что число как значений переменных, так и значений функций невелико (в отличие от функций математического анализа, где число и тех, и других, как правило, бесконечно):

x	f_1	f_2	f_3	f_4
0	0	0	1	1
1	0	1	0	1

Из них функции f_1, f_4 - константы и f_2 (функция типа $y = x$) - неинтересные, а вот функция f_3 - инверсия (или отрицание) играет в формальной логике огромную роль. Мы будем обозначать функцию инверсии переменной x как x' или x'. Из таблицы видно, что значения x и x' будут инверсными.

Для $n = 2$, т.е. для функции двух переменных, $K = 2^4 = 16$; выпишем таблицу для 7 важнейших из них:

xy	f_1	f_2	f_3	f_4	f_5	f_6	f_7
00	0	0	1	1	0	1	1
01	0	1	1	0	1	0	1
10	0	1	0	0	1	0	1
11	1	1	1	1	0	0	0

Функция $f_1(x, y)$ - это конъюнкция переменных x и y. Она обозначается как $x \,\&\, y$, или просто xy, или как функция И. Заметим, что она равна 1 только при $x = 1,\ y = 1$.

Функция $f_2(x, y)$ - это дизъюнкция переменных x и y. Она обозначается как $x \vee y$, или $x\ V\ y$, или как функция ИЛИ. Заметим, что она равна 0 только при $x = 0,\ y = 0$.

Функция $f_3(x, y)$ - это импликация переменных x и y. Она обозначается как $x \to y$ и равна 0 только при $x = 1,\ y = 0$.

Функция $f_4(x, y)$ - эквивалентность переменных x и y; обозначается как $x \equiv y$.

Функция $f_5(x, y)$ - это сумма по модулю 2 переменных x и y. Она обозначается как $x \oplus y$ и является функцией, инверсной эквивалентности.

Функция $f_6(x, y)$ - это стрелка Пирса переменных x и y; обозначается как $x \uparrow y$. Она равна 1 только при $x = 0,\ y = 0$, т.е. является функцией, инверсной дизъюнкции, поэтому часто обозначается как ИЛИ-НЕ, т.е. $f_6 = (x \vee y)'$.

Функция $f_7(x, y)$ - это штрих Шеффера переменных x и y. Она обозначается как x/y, или как функция И-НЕ, так как она равна 0 только при $x = 1,\ y = 1$, т.е. является функцией, инверсной конъюнкции, т.е. $f_7 = (xy)'$.

Остальные 9 функций большого практического интереса не представляют.

Функции конъюнкции (И) и дизъюнкции (ИЛИ) могут быть обобщены на случай n переменных:

конъюнкция (И) n переменных $f = x_1x_2x_3...x_n$

$$f = \begin{cases} 1 & \text{если все } x_i = 1; \\ 0 & \text{иначе} \end{cases}$$

дизъюнкция (ИЛИ) n переменных $f = x_1 \vee x_2 \vee x_3 \vee ... \vee x_n$

$$f = \begin{cases} 1 & \text{если хотя бы одна } x_i = 1; \\ 0 & \text{иначе} \end{cases}$$

Импликация $p \to q$ может быть выражена через операции дизъюнкции и отрицания:

$$p \to q \equiv p' \vee q,$$

так как истинности обеих функций совпадают:

pq	$p \to q$	$p' \vee q$
00	1	1
01	1	1
10	0	0
11	1	1

К логическим функциям применима процедура суперпозиции, т.е. построение из элементарных функций, таких как конъюнкция, дизъюнкция, отрицание, импликация и т.д., более сложных выражений с помощью этих же функций. Для того чтобы порядок операций в таких выражениях понимался однозначно, установлены следующие приоритеты для логических операций:

- Высший приоритет принадлежит операции отрицания.
- Следующий приоритет принадлежит операции конъюнкции.
- Остальные операции имеют одинаковый приоритет.

Если все же приоритеты надо изменить, следует использовать скобки. Примеры:

1. $f_1(x, y, z) = (x'y \vee y'z) \to xz$. В этом выражении сначала будут вычислены x', y', затем все конъюнкции, затем дизъюнкция в скобках и только потом импликация.
2. $f_2(x, y, z) = (xy \to z) \equiv (xz \to y \to x)$. Здесь сначала будут вычислены конъюнкции, затем импликации. Так как импликации в правой части не выделены скобками, то сначала следует вычислить 1-ю импликацию, затем 2-ю. В конце вычисляется эквивалентность.

3. $f_3(x,y,z) = xy'z \vee x'z' \vee yz$. Здесь сначала вычисляются отрицания, затем конъюнкции, затем дизъюнкции.

4. $f_4(x,y,z) = (x \vee y' \vee z)(x' \vee z')(y \vee z)$. Здесь сначала вычисляются отрицания, затем дизъюнкции, затем конъюнкции.

5. $f_5(x,y,z) = x \vee z(xy \vee x)'$.

Функция f_3 - это так называемая дизъюнктивная нормальная форма, т.е. логическая функция, являющаяся дизъюнкцией конъюнкций переменных или их отрицаний. Кратко она обозначается ДНФ. Члены этой формы (конъюнкции переменных) $xy'z$, $x'z'$, yz называются термами ДНФ.
Функция f_4 - это так называемая конъюнктивная нормальная форма, т.е. логическая функция, являющаяся конъюнкцией дизъюнкций переменных или их отрицаний. Кратко она обозначается КНФ. Члены этой формы (дизъюнкции переменных) $(x \vee y' \vee z)$, $(x' \vee z')$, $(y \vee z)$ называются термами КНФ.
Функция f_5 похожа на эти формы, но не является ни одной из них.
Пример вычисления сложной функции с помощью таблицы. Задача: вычислить таблично функцию двух переменных: $f(p,q) = ((pq') \to q) \vee (p \to q)$.

pq	pq'	$\to_1$	$\to_2$	$\vee$, т.е. f
00	0	1	1	1
01	0	1	1	1
10	1	0	0	0
11	0	1	1	1

А теперь возьмем функцию f_1 из примеров выше - функцию трех переменных:

xyz	x'y	y'z	$\vee$	xz	$\to$ т.е. f_1	Комментарий.
000	0	0	0	0	1	$f_1 = 0 \to 0 = 1$
001	0	1	1	0	0	$f_1 = 1 \to 0 = 0$
010	1	0	1	0	0	$f_1 = 1 \to 0 = 0$
011	1	0	1	0	0	$f_1 = 1 \to 0 = 0$
100	0	0	0	0	1	$f_1 = 0 \to 0 = 1$
101	0	1	1	1	1	$f_1 = 1 \to 1 = 1$
110	0	0	0	0	1	$f_1 = 0 \to 0 = 1$
111	0	0	0	1	1	$f_1 = 0 \to 1 = 1$

Далее рассмотрим свойства операций конъюнкции, дизъюнкции и отрицания:

- ассоциативность x(yz)=(xy)z; $x \vee (y \vee z) = (x \vee y) \vee z$;
- коммутативность xz=zx; $x \vee z = z \vee x$;
- дистрибутивность $xy \vee xz = x(y \vee z)$; $(x \vee y)(x \vee z) = x \vee (yz)$;
- $a0 = 0$; $a1 = a$; $aa = a$; $aa' = 0$;
- $a \vee 0 = a$; $a \vee 1 = 1$; $a \vee a = a$; $a \vee a' = 1$;

- $a'' = a$.

Доказательство, как обычно, табличным методом. Докажем, например, что $(x \vee y)(x \vee z) = x \vee (yz)$ (свойство дистрибутивности, оно в этом случае для логических операций не такое же, как для арифметических), т.е. что левая часть всегда равна правой. Имеем

xyz	$x \vee y$	$x \vee z$	левая часть	yz	правая часть
000	0	0	0	0	0
001	0	1	0	0	0
010	1	0	0	0	0
011	1	1	1	1	1
100	1	1	1	0	1
101	1	1	1	0	1
110	1	1	1	0	1
111	1	1	1	1	1

что и требовалось доказать.

Законы де Моргана:

1. $(pq)' \equiv p' \vee q'$.
2. $(p \vee q)' \equiv p'q'$.

Доказательство - проверкой таблиц:

pq	$(pq)'$	$p' \vee q'$	$(p \vee q)'$	$p'q'$
00	1	1	1	1
01	1	1	0	0
10	1	1	0	0
11	0	0	0	0

СОВЕРШЕННЫЕ НОРМАЛЬНЫЕ ФОРМЫ. Мы уже ввели определение нормальных форм ДНФ и КНФ от n переменных, а сейчас мы дадим определение совершенных нормальных форм. Совершенная дизъюнктивная нормальная форма СДНФ - это ДНФ, в каждый терм которой входят все n переменных в прямом или инверсном виде. Пример: $f(x,y,z) = x'yz \vee x'y'z \vee xyz'$.
Аналогично совершенная конъюнктивная нормальная форма СКНФ - это КНФ, в каждый терм которой входят все n переменных (в прямом или инверсном виде).
Пример: $f(x,y,z,u) = (x \vee y \vee z \vee u)(x' \vee y \vee z \vee u)$.
Достоинства обеих форм видны из их двух следующих свойств:

1. Быстрота вычисления: любая из этих форм может быть вычислена за 2 независимых шага - на языке электроники за два логических такта;
2. Универсальность представления логических функций; это следует из ТЕОРЕМЫ:
   Любая логическая функция может быть записана как СДНФ или СКНФ.

Доказательство: его идею можно пояснить на примере функции $f(x,y,z)$:

x	y	z	f	СДНФ	СКНФ
0	0	0	1	$x'y'z'$	
0	0	1	1	$x'y'z$	
0	1	0	0		$x \vee y' \vee z$
0	1	1	1	$x'yz$	

x	y	z	f	СДНФ	СКНФ
1	0	0	0		$x' \vee y \vee z$
1	0	1	1	$xy'z$	
1	1	0	0		$x' \vee y' \vee z$
1	1	1	1	xyz	

Мы строим термы СДНФ так, чтобы на тех наборах значений переменных, где функция имеет значение 1, терм тоже принимал значение 1. Для этого если значение переменной в наборе было 1, то переменная входит в прямом виде; если же 0, то в инверсном. Соответственно, для набора xyz = 000 мы строим терм x'y'z'; для xyz = 001 терм будет x'y'z, и т.д. Окончательно получаем 5 термов по числу единиц функции f. Легко видеть, что на тех наборах, где f равна 0, все термы СДНФ обратятся в нуль - в силу взаимно-однозначного соответствия между двоичными кодами и построенными таким способом термами и того факта, что конъюнкция равна 1 только при равенстве 1 всех ее составляющих.
СКНФ мы строим так, чтобы на тех наборах значений переменных, где функция имеет значение 0, терм тоже принимал значение 0. Для этого если значение переменной в наборе было 0, то переменная входит в прямом виде; если же 1, то в инверсном. Т.е. наоборот, по сравнению с СДНФ.
Соответственно, для набора xyz = 010 терм будет $x \vee y' \vee z$, для xyz = 100 терм будет $x' \vee y \vee z$, и т.д. Здесь тоже можно видеть, что на тех наборах, где f равна 1, все термы СКНФ обратятся в 1 - в силу взаимно-однозначного соответствия между двоичными кодами и построенными таким способом термами и того факта, что дизъюнкция равна 0 лишь при равенстве 0 всех ее составляющих.
Окончательно имеем СДНФ: $f(x,y,z) = x'y'z' \vee x'y'z \vee x'yz \vee xy'z \vee xyz$
и СКНФ: $f(x,y,z) = (x \vee y' \vee z)(x' \vee y \vee z)(x' \vee y' \vee z)$.

Значение доказанной теоремы в том, что через СДНФ и СКНФ мы можем построить аналитическое представление любой логической функции. Другими словами, с помощью операций отрицания, дизъюнкции и конъюнкции любая логическая функция может быть изображена как формула, т.е. функции конъюнкция, дизъюнкция и отрицание образуют в пространстве логических функций базис. Он называется базис И, ИЛИ, НЕ.
Базисов в действительности много. Здесь можно указать на 2 важных базиса, которые играют большую роль в практической реализации логических устройств автоматики на основе интегральных схем. Это

- базис ИЛИ-НЕ - функция "стрелка Пирса",
- базис И-НЕ - функция "штрих Шеффера".

Доказательство: так как любая логическая функция может быть записана как СДНФ или СКНФ, то достаточно доказать, что функции НЕ, ИЛИ, И можно выразить через И-НЕ (/) либо ИЛИ-НЕ (↑). Действительно,

НЕ:	x'	$=$	$(xx)'$	$= x/x;$
	x'	$=$	$(x \vee x)'$	$= x \uparrow x.$
И:	xy	$=$	$(xy)'' = ((xy)')'$	$= (x/y)/(x/y);$
	xy	$=$	$(xy)'' = (x' \vee y')'$	$= (x \uparrow x) \uparrow (y \uparrow y).$
ИЛИ:	$x \vee y$	$=$	$(x \vee y)'' = (x'y')'$	$= (x/x)/(y/y);$
	$x \vee y$	$=$	$(x \vee y)'' = ((x \vee y)')'$	$= (x \uparrow y) \uparrow (x \uparrow y).$

Табл.1. Формулы перевода И,ИЛИ,НЕ в И-НЕ и ИЛИ-НЕ

Мы воспользовались законами де Моргана и тем фактом, что двойное отрицание функции или переменной не меняет их. В результате мы видим, что в окончательных выражениях используются только функции / или ↑, а тем самым и любая логическая функция оказывается выраженной через одну из этих базисных функций.
Запись в этих базисах, конечно, более громоздкая и менее обозримая, чем в базисе И,ИЛИ,НЕ, но с технологической точки зрения при использовании интегральных схем гораздо более легко реализуемыми оказываются логические функции, записанные именно в этих базисах.

ЗАДАЧА. Записать в базисе И-НЕ функцию $f(x,y,z) = xyz$.
Решение. Сразу заметим, что операции / и ↑ неассоциативны $x/(y/z) \neq (x/y)/z$ (проверить таблично!); поэтому порядок выполнения этих операций важен и его следует обозначить скобками. Имеем $f = xyz = (xyz)'' = ((xy)' \vee z')' = ((x/y) \vee z')'$. Обозначим g=x/y. Тогда $(g \vee z')' = g'z$. Воспользуемся формулами перевода конъюнкции и отрицания из табл.1. Имеем f= (g'/z)/(g'/z) и g' = (x/y)/(x/y). Окончательно имеем: f=((((x/y) / (x/y))/z)/ (((x/y)/(x/y))/z)).
Проверим правильность решения с помощью таблицы.

x	y	z	x/y	g'/z	f	xyz
0	0	0	1	1	0	0
0	0	1	1	1	0	0
0	1	0	1	1	0	0
0	1	1	1	1	0	0
1	0	0	1	1	0	0
1	0	1	1	1	0	0
1	1	0	0	1	0	0
1	1	1	0	0	1	1

Решение найдено правильно: f = xyz.

МИНИМИЗАЦИЯ СДНФ и СКНФ. Запись функций в виде СДНФ или СКНФ универсальна, обладает определенными эстетическими достоинствами, но является избыточно насыщенной переменными. Однако и СДНФ, и СКНФ могут быть трансформированы в просто ДНФ или КНФ и при этом оставаться по значениям теми же самыми логическими функциями. При этом может оказаться, что и ДНФ, и КНФ содержат меньше термов, и, естественно, каждый терм может содержать меньше переменных. Такая компактизация СДНФ и СКНФ необходима при использовании их в устройствах автоматики.
Например, пусть дана функция $f = (xyz \vee xyz')$. Воспользовавшись свойством дистрибутивности дизъюнкции и конъюнкции и тем, что $x \vee x' = 1$, мы можем записать $(xyz \vee xyz') = xy(z \vee z') = xy$. Новая ДНФ имеет всего 1 терм и 2 переменные.

Общие алгоритмы упрощения и минимизации СДНФ и СКНФ основаны на свойствах логических операций (см. выше) и таких формулах, как

- $Za \vee Za' = Z(a \vee a') = Z1 = Z$;
- $(Z \vee a)(Z \vee a') = ZZ \vee Za \vee Za' \vee aa' = Z \vee Z(a \vee a') \vee 0 = Z \vee Z1 = Z$,

где Z - любая логическая функция.
Пример:
$xyz \vee xyz' \vee xy'z \vee xy'z' = xy(z \vee z') \vee xy'(z \vee z') = xy1 \vee xy'1 = x(y \vee y') = x1 = x$.

КАРТЫ КАРНО. Из рассмотренных выше примеров становится ясно, что главным шагом при минимизации СДНФ и СКНФ является вынесение за скобки общих частей термов так, чтобы в скобках оставались дизъюнкции (конъюнкции) переменных и их отрицаний, и "склеивание" этих переменных. Предложено множество методов для облегчения этой процедуры. Их цель - расположить термы нормальных форм так, чтобы в наглядной и очевидной форме было ясно, какие термы, имея общую часть, отличаются друг от друга взаимно инверсными переменными. Наиболее популярный метод - это метод карт Карно. Его работу мы рассмотрим на примере СДНФ (СКНФ) от функций 2, 3 и 4 переменных.
Итак, пусть таблично задана функция 2 переменных $f(x,y)$, СДНФ которой равна $x'y' \vee x'y$, а СКНФ $= (x' \vee y)(x' \vee y')$.

00	1
01	1
10	0
11	0

Карта Карно для нее имеет следующий вид:

00	01	11	10
1+	1+	0	0

В клетках находятся значения функции f, над каждой клеткой ее "координаты" - значения переменых x, y, на которых она принимает значение, находящееся в клетке. Клетки, таким образом, взаимно - однозначно соответствуют термам СДНФ и СКНФ.
Координаты выписаны не так, как при табличном задании функции, а именно: 00, 01, 11, 10, т.е. после 01 идет не 10, как обычно, а 11, и лишь затем 10. Это сделано для того, чтобы любая пара соседних клеток (а тем самым и соответствующие термы СДНФ и СКНФ) имела общую часть и взаимно инверсные переменные, которые могут быть "склеены".
Заметим, что соседними являются также клетки с "координатами" 00 и 10.
Преимущество карт Карно в том, что на них сразу видны примыкающие друг к другу единицы или нули, и выделить соседние пары легко. Обычно такие пары покрывают фигурой из двух клеток, или можно пометить их каким-либо символом (в нашем случае "+").
Минимизация СДНФ осуществляется по единицам, минимизация СКНФ по нулям, поэтому при минимизации СДНФ не наносятся на карту нули, а при минимизации СКНФ - единицы.
В обоих случаях взаимно инверсные переменные склеиваются, общая часть остающихся минимальных термов выписывается по тому же правилу, что и при построении СДНФ (СКНФ) - единицам соответствуют прямые вхождения

переменных, нулям инверсные (в случае СКНФ наоборот).
Видно, что клетки с координатами 00 и 01 являются соседними. Отсюда для СДНФ имеем $f(x,y) = x'$. Переменная y "склеилась".
Минимизация СКНФ дает точно такой же результат.

Далее рассмотрим случай функции 3 переменных $f(x,y,z)$, СДНФ которой равна $x'y'z' \vee x'y'z \vee xyz \vee xy'z \vee xy'z'$.

xy	f
000	1
001	1
010	0
011	0

xy	f
100	1
101	1
110	0
111	1

Карта Карно для нее имеет следующий вид:

xy=	00	01	11	10
z=0	1+			1+
z=1	1+		1-	1+-

Координаты здесь складываются так: сверху xy, сбоку z. Так как мы минимизируем СДНФ, то нули не выписываем.
В случае 3 переменных самые большие покрытия делаются фигурами из 4 клеток (2 x 2) или (4 x 1). В случае нашей функции мы покроем следующие соседние клетки: четверное покрытие (клеток 000,001,100,101) (пометим их знаком +) и двойное (клеток 111,110) (помечены знаком -).
В результате на четверном покрытии взаимно инверсными оказываются переменные x и z, а на двойном переменная y. Окончательно минимальной ДНФ будет $y' \vee xz$.
Единица в правом нижнем углу оказалась покрытой дважды, но это не страшно: мы как бы удвоили соответствующий терм в СДНФ, но дизъюнкция двух одинаковых термов не меняет значения СДНФ. Зато этот прием позволил нам провести более эффективную минимизацию. Если бы мы оставили 1 на клетке 111 непокрытой, терм этой клетки был бы xyz, т.е. содержал бы 3 переменных, а не 2.

И, наконец, рассмотрим случай функции 4 переменных $f(x,y,z,u)$, СДНФ которой равна $x'y'z'u' \vee x'y'z'u \vee xy'z'u' \vee xy'z'u \vee x'y'zu' \vee x'yzu' \vee xyzu' \vee xy'zu'$.

xyzu	f	xyzu	f
0000	1	1000	1
0001	1	1001	1
0010	1	1010	1
0011	0	1011	0
0100	0	1100	0
0101	0	1101	0
0110	1	1110	1
0111	0	1111	0

Карта Карно для нее:

xy=	00	01	11	10
zu=00	1-			1-
zu=01	1-			1-
zu=11				
zu=10	1+	1+	1+	1+

Координаты здесь строятся так: сверху xy, сбоку zu. Мы минимизируем СДНФ, так что нули снова не выписываем.
В случае 4 переменных самые большие покрытия делаются фигурами из 8 клеток (4 x 2); далее идут фигуры из 4 клеток (2 x 2) или, наконец, из 2 клеток (2 x 1). В случае нашей функции мы покроем следующие соседние клетки: четверное покрытие (клеток 0000,0001,1000,1001) символом - и четверное покрытие (клеток

0010,0110,1110,1010) символом +. МДНФ в результате будет такой: $f = y'z' \vee zu'$. По данной карте мы могли построить и МКНФ (покрывая нули функции). Но это не дало бы нам выигрыша. Здесь тоже были бы 2 четверных покрытия нулей и МДНФ была бы такой: $f = (y' \vee z)(z' \vee u')$.
Если бы в клетках 0011 и 1011 стояли единицы, мы смогли бы сделать покрытие фигурой из 8 клеток (0000, 0001,0011,0010,1000,1001,1011,1010) и фигурой из 4 клеток (0010,0110,1110,1010) и МДНФ тогда была бы $f = y' \vee zu'$.
Минимизацию функций более чем 4 переменных делать вручную затруднительно. Здесь уместнее использование программных методов. Их алгоритмы основаны на тех же самых идеях "склеивания" взаимно инверсных переменных при наличии у термов общих частей.

НЕ ПОЛНОСТЬЮ ОПРЕДЕЛЕННЫЕ ФУНКЦИИ. Может так быть, что логическая функция задана не на всех наборах значений своих переменных. Мы можем использовать это обстоятельство для более эффективной минимизации СДНФ (или СКНФ) такой функции, доопределяя ее так, чтобы это позволяло "склеивать" больше переменных. Рассмотрим это на примере следующей функции 4 переменных $f(x, y, z, u)$, СДНФ которой равна $x'y'z'u' \vee xy'z'u' \vee xyzu \vee$. Пусть на наборах $xyzu$ = 0010, 0110,1110,1010 функция не определена. И в таблице, и на карте Карно мы отмечаем это тем, что ставим на соответствующие места символ * вместо 0 или 1:

xyzu	f	xyzu	f
0000	1	1000	1
0001	0	1001	0
0010	*	1010	*
0011	0	1011	0
0100	0	1100	0
0101	0	1101	0
0110	*	1110	*
0111	0	1111	1

Карта Карно для нее:

xy=	00	01	11	10
zu=00	1+			1+
zu=01				
zu=11			1-	
zu=10	*+	*	*-	*+

Мы можем покрыть клетки 0000 и 1000 фигурой из двух клеток, клетку 1111 оставить непокрытой и проигнорировать звездочки. Тогда мы получим МДНФ: $f = y'z'u' \vee xyzu$. Но мы можем поставить вместо звездочек в клетках 0010,1110,1010 единицы, а в клетке 0110 ноль (на функцию это никак не повлияет, поскольку эти наборы не входят в область ее определения), и тогда минимизация будет более эффективной: покрываем клетки 0000,1000,0010,1010 (они тоже соседние) фигурой из 4 клеток (помечены символом +), а клетки 1111,1110 фигурой из двух (помечены символом "-") и получаем МДНФ: $f = y'u' \vee xyz$. Имеем 5 вхождений переменных, тогда как без использования звездочек было 7 вхождений.
В заключение укажем, что во всех случаях покрывать как 1, так и 0 можно только фигурами в 2, 4, 8, 16 и т.д. клеток.
ПРИМЕР 1. Дана функция $f = (a \to b) \vee (a' \to c)$. Имеем таблицу f:

abc	f		abc	f
0 0 0	1		1 0 0	1
0 0 1	1		1 0 1	1
0 1 0	1		1 1 0	1
0 1 1	1		1 1 1	1

Функция f оказалась константой (тавтологией). Посмотрим, можно ли это установить, анализируя формулу f. Имеем

$$f = (a' \vee b) \vee (a \vee c) == a' \vee b \vee c \vee a = (a' \vee a) \vee b \vee c =$$

$$= 1 \vee b \vee c = 1.$$

Следовательно.
ПРИМЕР 2. Рассмотрим функцию $f = (ab \vee bc \vee ac)$ и таблицу для нее:

abc	f		abc	f
0 0 0	0		1 0 0	0
0 0 1	0		1 0 1	1
0 1 0	0		1 1 0	1
0 1 1	1		1 1 1	1

Эта функция уже не константа.
В качестве упражнения можно рассмотреть функции:

$$f_1 = (ab \vee bc \vee ac \vee abc)$$

$$f_2 = (ab \vee bc \vee ac \vee abc \vee a'b'c')$$

$$f_3 = (a \vee b \vee c \vee abc \vee a'b'c$$

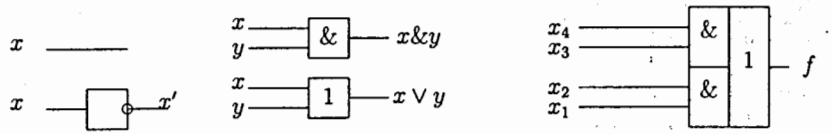
Вопрос: будут ли тавтологиями f_1, f_2, f_3?
Показать тавтологичность функций:

$$f_4 = (a \vee a') \vee (b \vee b') \vee (c \vee c');$$

$$f_5 = (a \to a) \vee (b \to b) \vee (c \to c).$$

ВАЖНЕЙШИЕ КОМБИНАЦИОННЫЕ СХЕМЫ

Далее мы рассмотрим важнейшие приложения математической логики в автоматике и электронике. Начнем с комбинационных схем. КС-схема - это схема, комбинация выходных сигналов которой однозначно определяется комбинацией входных сигналов.
Пример: любой преобразователь без памяти. Мы рассмотрим важнейшие, но сначала введем необходимые
ГРАФИЧЕСКИЕ ОБОЗНАЧЕНИЯ.
При проектировании электронных схем используются следующие графические обозначения: каждому сигналу сопоставляется шина - провод, по которому проходит сигнал. Контакты осуществляются на боксах, которые при этом реализуют ту или иную логическую функцию от сигналов:

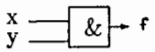


Здесь приведены примеры графического изображения сигнала x, его инверсии x', конъюнкции и дизъюнкции сигналов x и y и функции $f = x_1x_2 \vee x_3x_4$. Заметим, что кружок в начале линии означает инверсию сигнала, который далее будет проходить по этой шине. Например, для графического изображения схемы, вычисляющей инверсию конъюнкции сигналов x и y, используется следующий рисунок:

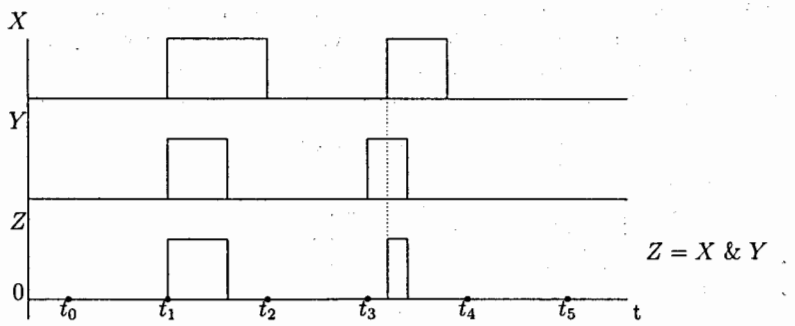
На таких схемах особенно четко видно большое значение минимизации логических функций - чем меньше переменных и функций, тем меньше шин, контактов и элементов, тем меньше расходы, тем выше надежность работы схемы.
ПРОПУСКНОЙ ВЕНТИЛЬ. Даны логические переменные x и y. Требуется: пропустить сигнал x (нулевой или единичный), если y=1, и заблокировать иначе:

$$f = x \,\&\, y \quad \rightarrow \quad f = \begin{cases} x & \text{если y=1;} \\ 0 & \text{иначе (y=0)} \end{cases}$$

Схема:



Пропускной вентиль может использоваться также для формирования сигнала импульсной формы. Пусть сигналы x - потенциал, а y - импульс. Тогда сигнал $z = x\&y$ будет иметь импульсную форму. Это следует из временной диаграммы:



КОМПАРАТОР. Даны 2 n-разрядных кода: $x_n....x_1x_0, \quad y_n....y_1y_0$.
Требуется выработать сигнал $f = 1$ (если полное совпадение кодов) или 0 (несовпадение хотя бы одной i-й пары):

$$f = \&f_i, \quad \text{где } f_i : (x_i = y_i).$$

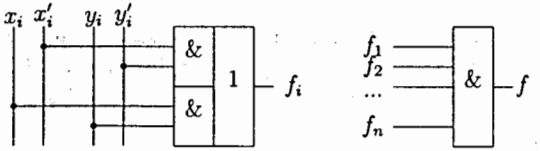
Составим таблицу функции f_i и построим для нее СДНФ:

Таблица f:

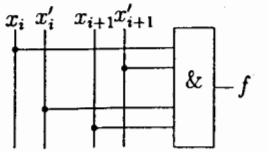
x_i	y_i	f_i
0	0	1
0	1	0
1	0	0
1	1	1

$$f_i = (x_i'\&y_i')V(x_i\&y_i)$$

Схема:

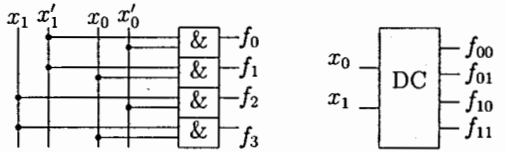


СЛОЖЕНИЕ ПО МОДУЛЮ 2. Дан код $X = x_n....x_1x_0, \quad n+1 = 2^k$.
Требуется: выработать сигнал четности суммы единиц кода:
$f = 0$ (четная сумма), 1 (нечетная).
Разбиваем код на пары, формируем соседние попарные четности всего кода, затем соседние попарные четности для них и т.д.
На первом проходе получаем 2^{k-1} сигналов четности f_i; к каждой паре f_i, f_{i+1} из них применяем ту же самую процедуру и получаем 2^{k-2} новых сигналов четности, и т.д. до тех пор, пока не останутся 2 сигнала; их четность или нечетность совпадет с четностью или нечетностью всего кода. Схема будет следующей:



ДЕШИФРАТОР. Дан двоичный код $X = x_n....x_1x_0$. Из (n+1) разрядов можно построить 2^{n+1} кодовых комбинаций. Допустим, мы хотим сопоставить каждой комбинации единичный выход логической схемы, т.е. всего 2^{n+1} выходов.
Это можно сделать, рассматривая двоичный код как адрес соответствующего выхода.

Пример двухразрядного кода:



Каждый выход будет соответствовать адресу x_1x_0, и с него будет сниматься единичный сигнал.
Например, если $x_1x_0 = 10$, го выход f_2 будет равен 1, остальные выходы будут нулевыми.
Справа находится условное обозначение дешифратора.

ШИФРАТОР. Дан двоичный код $x_n...x_1x_0$, где один $x_i = 1$, остальные равны 0; $n + 1 = 2^k$. Операция шифрации обратна операции дешифрации, т.е. требуется выработать двоичный адрес единицы:

$$x_n..x_1x_0 \quad \Leftrightarrow\Rightarrow \quad a_k...a_1a_0$$

Строим логические функции разрядов адреса по обычной табличной процедуре. Возьмем, например, $n = 3$, т.е. $k = 1$. Имеем

x_3	x_2	x_1	x_0	a_1	a_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Строим СКНФ по 0 обеих функций.
$a_1 = (x_3 V x_2 V x_1 V x_0')\&(x_3 V x_2 V x_1' V x_0)$
$a_0 = (x_3 V x_2 V x_1 V x_0')\&(x_3 V x_2' V x_1 V x_0)$

Минимизируем обе формы с помощью карт Карно.

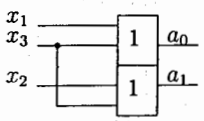
Для a_1

	00	01	11	10
00	*+	0+	*+	0+
01		*	*	*
11	*	*	*	*
10		*	*	*

Для a_0

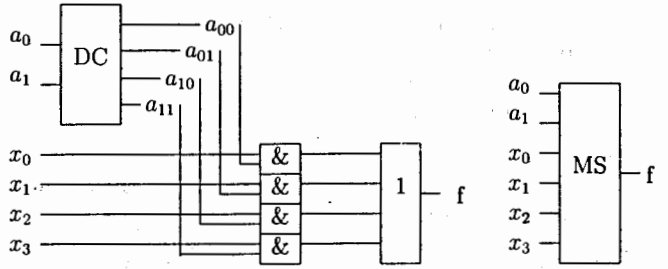
	00	01	11	10
00	*+	0+	*	
01	0+	*+	*	*
11	*	*	*	*
10		*	*	*

Отсюда получаем МКНФ $a_0 = x_1 V x_3$; $a_1 = x_2 V x_3$.
Ниже дана схема шифратора



Минимизация по единицам дала бы следующие МДНФ: $a_1 = x_1'\&x_0'$; $a_0 = x_2'\&x_1'$.

МУЛЬТИПЛЕКСОР. Даны: двоичный адрес $A = a_k...a_1a_0$, двоичные сигналы $x_n..x_1x_0$, $n + 1 = 2^{k+1}$.
Требуется: выбрать сигнал x_j, где j равно адресу A.
Решение: адрес посылается на дешифратор, а выходы с него на пропускные вентили, отпираемые сигналами x_i.
Рассмотрим пример: $k = 1$, $n + 1 = 4$. Имеем следующую схему:



Рассмотренный режим работы мультиплексора используется для чтения данных, но это же устройство можно адаптировать к режиму записи информации - для этого достаточно поменять вход и выход данных местами; адресный вход работает аналогично и осуществляет нужную коммутацию шин записываемых данных и шин приемника этой информации.
Количество коммутаций при большом числе шин чтения / записи и, соответственно, длинном адресе может оказаться чрезмерно большим. В этом случае можно построить иерархическую пирамиду адресации - разбить адрес на подадреса, самый старший подадрес подается на адресный вход верхнего мультиплексора и коммутирует выходы мультиплексоров нижележащего слоя; на адресные входы последних подается подадрес, следующий за старшим, и т.д.

Мультиплексор MS можно использовать для реализации логической функции: если есть $n + 1$ значений функции $(f_0...f_n)$ на $k + 1$-разрядных наборах двоичных переменных $A_0A_1...A_k$, то, взяв мультиплексор $MS(A0...Ak, f_0...f_n)$, мы получим возможность вычислить значение функции f на наборе $A_0A_1...A_k$.
Справа находится условное обозначение мультиплексора.

СУММАТОР. Даны 2 двоичных кода: $X = (x_n....x_1x_0)$ и $Y = (y_n....y_1y_0)$.
Требуется получить двоичную сумму этих кодов.
Сложим 2 двоичных числа. Сложение протекает так:

	01010010	1-е слагаемое
+	00101011	2-е слагаемое
	01111001	1-я поразрядная сумма
+	00000100	плюс переносы
=	01111101	окончательная сумма

Мы видим, что при сложении каждых i-х разрядов складываются три двоичных

числа: x_i, y_i и p_i (перенос из (i-1)-го разряда) и вырабатываются две величины: s_i (i-я поразрядная сумма) и p_{i+1} (перенос в i+1-й разряд).
Функции s_i и p_{i+1} для операции сложение в i-м разряде будут выглядеть так:

x_i	y_i	p_i	s_i	p_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

→

s_i	p_{i+1}
$x_i'y_i'p_i$	
$x_i'y_ip_i'$	
	$x_i'y_ip_i$
$x_iy_i'p_i$	
	$x_iy_i'p_i$
	$x_iy_ip_i'$
$x_iy_ip_i$	$x_iy_ip_i$

Справа указаны термы для СДНФ обеих функций. Для построения их минимальных ДНФ используем карты Карно:

Для s_i

	00	01	11	10
00		1		1
01	1		1	

Для p_{i+1}

	00	01	11	10
00			1+	
01		1+	1+	1+

Отсюда имеем: в 1-й карте покрытий нет, форма не минимизируется; во 2-й покрываем 3 пары единиц фигурами из 2 клеток:

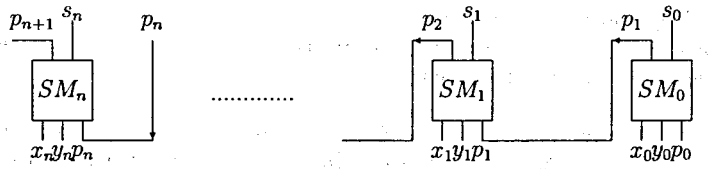
$$s_i = x_i'y_i'p_i V x_i'y_ip_i' V x_iy_ip_i V x_iy_i'p_i',$$

$$p_{i+1} = y_ip_i V x_iy_i V x_ip_i.$$

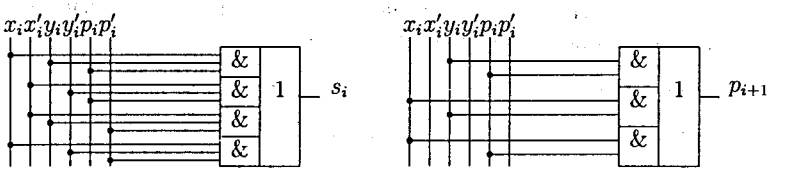
Итак, операция сложения реализуется в 2 этапа:

1. мгновенное поразрядное вычисление разрядов первичной суммы $s(i)$, $i = 0, 1, 2, ..., n$;
2. последовательное вычисление переносов (справа налево) p_1, p_2, ..., p_n ($p_0 = 0$) и последовательное сложение их с разрядами первичной суммы.

Общее время выполнения операции равно $T + ndt$, где
T - время, необходимое для вычисления первичной суммы;
dt - время, необходимое для суммирования с одним переносом.
Правильный результат будет получен только после того, как произойдет сложение с переносом в n-й (последний левый) разряд. Перенос в $n+1$ разряд - признак переполнения разрядной сетки сумматора.
Таким образом, $(n+1)$-разрядный сумматор - это $n+1$ одноразрядных, поставленных рядом, и каждый из них вырабатывает текущую поразрядную сумму и перенос в следующий сумматор. Схема:



Для выработки функций s_i, p_{i+1} в каждом разряде используются следующие схемы:

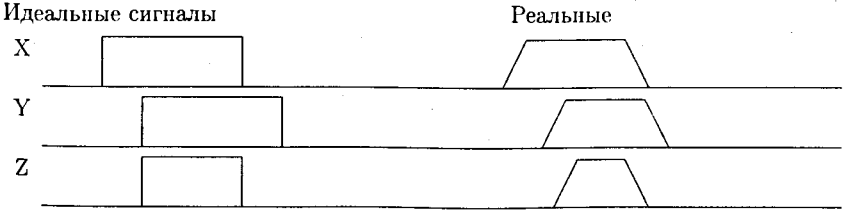


ВЫЧИТАНИЕ. Пусть нам требуется вычислить разность кодов X и Y. Это делается следующим образом.
Вычислим код, инверсный к коду Y, т.е. заменим все x_i на их инверсии: $X \to Y'$. Сумма этого кода и специального кода 000...001 даст нам код, дополнительный к коду Y. Обозначим его Y_d. Тогда разность кодов X и Y есть сумма кодов X и Y_d:

$$X - Y = X + Y_d.$$

Вычитание $X - Y$, таким образом, есть фактически сложение по модулю 2^{n+1} и может быть осуществлено с помощью рассмотренного выше сумматора.
Пример: имеем задачи 6-2=?, или 0110 - 0010 = ?
Строим инверсный код к Y: $Y' = 1101$. Строим дополнительный код к Y: $Y_d = 1110$. Складываем X и Y_d: 0110+1110 = 0100 и 1 в крайнем левом переносе; результат вычитания = 4, т.е. задача решена правильно. Заметим, что в то время как в случае сложения 1 в крайнем левом переносе служила признаком аварийного завершения операции, здесь, в случае вычитания, это признак успешного завершения операции.

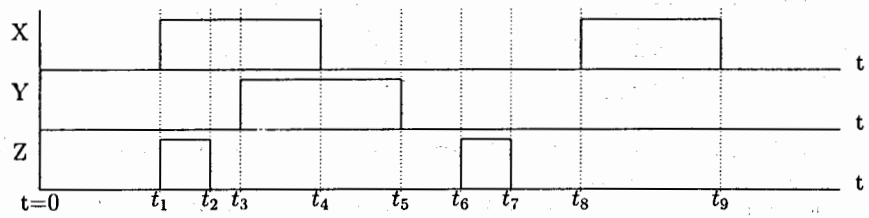
ВРЕМЕННЫЕ ДИАГРАММЫ. Рассмотрим пару временных диаграмм: идеальной и соответствующей ей реальной.
Даны импульсные сигналы X и Y и построенный из них сигнал $Z = X\&Y$. На временных диаграммах их соотношение будет выглядеть так.



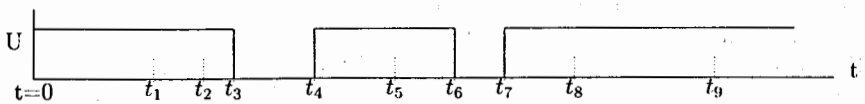
Реальные сигналы имеют наклонные фронты, причем углы наклона флуктуируют и в пространстве, и во времени. Имеют место задержки выполнения операций &,' и ∨. В последовательных схемах возникают накопление сдвигов сигналов по сравнению

с расчетами и т.н. "соревнование сигналов".
Преодоление этого - синхронизация всех операций импульсами с периодом значительно меньшим, чем у рабочих сигналов.

УПРАЖНЕНИЕ. 3 сигнала X, Y и Z поступают на вход устройства, вырабатывающего сигнал $U = XY' \vee X'Z' \vee XYZ$. Начертить временную диаграмму сигнала U.
Решение. Пусть входные сигналы X,Y,Z зависят от времени t так, как это представлено на нижеследующих временных диаграммах:



В момент времени t=0 $U = 01 \vee 11 \vee 000$, т.е. 1. Очевидно, что значения U меняться могут только при изменении одного из сигналов X,Y,Z, т.е. в моменты времени $t_1, t_2, ..., t_9$. Для удобства чтения эти моменты помечены пунктирной линией.
Следовательно, будем иметь:
$U(0, t_1) = 01 \vee 11 \vee 000 = 1.$
$U(t_1, t_2) = 11 \vee 00 \vee 101 = 1.$
$U(t_2, t_3) = 11 \vee 01 \vee 100 = 1.$
$U(t_3, t_4) = 10 \vee 01 \vee 110 = 0.$
$U(t_4, t_5) = 00 \vee 11 \vee 010 = 1.$
$U(t_5, t_6) = 01 \vee 11 \vee 000 = 1.$
$U(t_6, t_7) = 01 \vee 10 \vee 001 = 0.$
$U(t_7, t_8) = 01 \vee 11 \vee 000 = 1.$
$U(t_8, t_9) = 11 \vee 01 \vee 100 = 1.$
$U(t_9, ...) = 01 \vee 11 \vee 000 = 1.$
Другими словами, временная диаграмма сигнала U будет следующей:



ТЕОРИЯ АВТОМАТОВ

Комбинационные схемы не имеют памяти, поэтому класс задач, которые они могут решить, невелик и примитивен. Более сложными устройствами, которые могут выполнять действия, характеризующиеся некоторыми определенными "интеллектуальными" чертами, являются автоматы, играющие сегодня в электронике фундаментальную роль.
Абстрактный автомат преобразует входную информацию в выходную в зависимости от своих внутренних состояний и тем самым является информационным преобразователем более высокого ранга по сравнению с комбинационными схемами, т.е. является системой с памятью.
В автомате различают:

- Входной алфавит X - множество слов $x_1, x_2, x_3, ..., x_n$;
- Выходной алфавит Y - множество слов $y_1, y_2, y_3, ..., y_m$;
- Алфавит состояний Q - множество слов $q_1, q_2, q_3, ..., q_k$.

Дискретный автомат работает в дискретном времени, т.е. в моменты $t = ..., 0, 1, 2, ...$
Частным случаем автомата является конечный автомат - автомат с конечными числами слов в алфавитах. Мы будем иметь дело только с такими автоматами.
АВТОМАТ (Мили) - множество $A = (X, Y, Q, y(X,Q), q(X,Q))$, где
$y(X,Q)$ - функция выходов, вырабатывающая по текущим словам из X и Q на такте i слово из Y;
$q(X,Q)$ - функция переходов, вырабатывающая по текущим словам из X и Q на такте i состояние для такта $i+1$: $Q_{i+1} = q(X_i, Q_i)$.
Мы ограничимся в нашем курсе лишь детерминированными автоматами, в которых обе функции являются неслучайными, т.е. результаты их работы при неизменных условиях всегда одни и те же. В недетерминированных автоматах выработка функциями $y(X,Q)$ и $q(X,Q)$ выходных сигналов и сигналов переключения состояний содержит элемент случайности, описываемый определенной функцией распределения вероятностей.
Для частного случая автоматов (автоматы Мура) функция $q = q(Q_i)$, т.е. не зависит от входной информации.
Автомат может работать в непрерывном режиме (без начала и конца), а может иметь особые (начальные и/или конечные) состояния q_0 и q_f.
Входная последовательность сигналов может быть не упорядочена для любых типов автомата - они могут поступать в любом порядке. Выработка в определенный такт времени i по входному сигналу X_i выходных сигналов Y_i согласно функции $y(X_i, Q_i)$ логически осуществляется автоматом в состоянии Q_i вместе с подготовкой нового состояния Q_{i+1} в зависимости от X_i и текущего состояния Q_i, и, таким образом, пара рекуррентных соотношений

$$Y_i = y(X_i, Q_i); \quad Q_{i+1} = q(X_i, Q_i)$$

описывает возможную эволюцию автомата, во время которой автомат производит некоторую работу по созданию полезной информации.

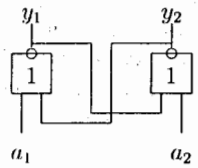
ЗАДАНИЕ АВТОМАТА. Алфавиты задаются перечислением. Функции могут задаваться, если нет формульного описания автомата, двояко:

1. Табличное задание. Для строк с парами x, q строятся колонки значений $Y(x,q)$ и $Q(x,q)$.

2. С помощью графа. Вершины графа обозначают состояния, дуги указывают на переходы из одних состояний в другие; веса дуг указывают на пары x, y, которыми сопровождается данный переход.

Абстрактное описание автомата раскрывает его состав и функцию, но не может быть непосредственно претворено в техническую схему. Для практической реализации автомата самым лучшим методом оказалось использование двоичной кодировки для алфавитов и двоичных логических функций для алгоритмов.
При синтезе автоматов формируют систему из элементарных автоматов, эквивалентную заданному абстрактному автомату, - структурный автомат. Элементарные автоматы (иначе подавтоматы) содержат в себе класс базисных подавтоматов, из которых состоит в общем случае любой крупный автомат. Наше изложение мы начнем с них.

БАЗИСНЫЕ АВТОМАТЫ. Таковыми в первую очередь являются триггера, дающие возможность самым простым и естественным способом представлять собой двоичные коды и хранить их. Класс триггеров очень обширен, и мы будем вводить их, начиная с простейших и постепенно усложняя их.
АСИНХРОННЫЙ RS-ТРИГГЕР.
Это схема следующего вида:



Уравнения триггера:
$y_1 = (a_1 \vee y_2)'$;
$y_2 = (a_2 \vee y_1)'$.

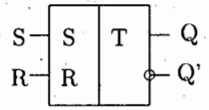
Решение уравнений находим для всевозможных комбинаций входов:

1. $a_1 = 0;\ a_2 = 0;\ \Rightarrow\ y_1 = y_2';\ y_2 = y_1';$
2. $a_1 = 0;\ a_2 = 1;\ \Rightarrow\ y_1 = y_2';\ y_2 = 0;$
3. $a_1 = 1;\ a_2 = 0;\ \Rightarrow\ y_1 = 0;\ y_2 = y_1';$
4. $a_1 = 1;\ a_2 = 1;\ \Rightarrow\ y_1 = 0;\ y_2 = 0.$

Последняя комбинация на входе RS-триггера недопустима, так как 2 нуля на его выходе несовместимы с той ролью, которую играют триггера в схемах устройств автоматики. Можно сказать, что недопустимы и 2 нуля, и 2 единицы на выходе, вследствие того, что триггера оказались самым удобным и универсальным средством

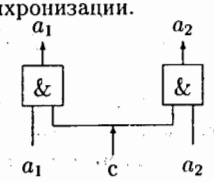
изображения и хранения легко читаемой и легко изменяемой двоичной информации.
С RS-триггером связаны 2 состояния, коды которых следующие: "0" соответствует выходам $y_1 = 0,\ y_2 = 1$; "1" соответствует выходам $y_1 = 1,\ y_2 = 0$.
Оказались очень удобными следующие условные обозначения:
Сигнал R (от английского reset) - это в действительности пара $a_1 = 1,\ a_2 = 0$, которая включает состояние "0", а S (от set)- это в действительности пара $a_1 = 0, a_2 = 1$, включающая состояние "1".
Сигнал 0 - это пара $a_1 = 0, a_2 = 0$, не меняющая текущее состояние триггера ("0" или "1").
Аналогичные условные названия введены и для выходов триггера, используемых для опроса состояния триггера.
Сигнал Q - это в действительности пара сигналов $y_1 = 1, y_2 = 0$; означает, что триггер находится в состоянии "1".
Сигнал Q' - это в действительности пара сигналов $y_1 = 0, y_2 = 1$; означает, что триггер находится в состоянии "0".
В таких обозначениях мы можем описать RS-триггер как автомат следующим образом.
Имеем входной алфавит: X=R,S,0; выходной: Y=Q,Q'; алфавит состояний: Q=0,1 и таблично заданные функции выходов (Q,Q') и переключения состояний (0,1).

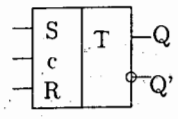
	0			1		
R	Q'	/	0	Q'	/	0
S	Q	/	1	Q	/	1
0	Q'	/	0	Q	/	1



Справа помещено условное графическое обозначение RS-триггера.
Данный триггер реализован в базисе ИЛИ-НЕ, но это можно сделать и в базисе И-НЕ (заменив элементы ИЛИ-НЕ на И-НЕ); логика работы устройства не изменится, и лишь полярность входных и выходных сигналов перейдет в инверсную (роль нулей будут играть единицы, а единиц - нули).

СИНХРОНИЗАЦИЯ RS-ТРИГГЕРА.
Выше рассмотренный асинхронный RS-триггер, реагирующий на входные сигналы, поступающие в случайные моменты времени, может быть синхронизирован, т.е. привязан к сигналам, поступающим в определенные моменты времени, с помощью подачи сигналов a_1, a_2 на пропускные вентили, отпираемые сигналами синхронизации.

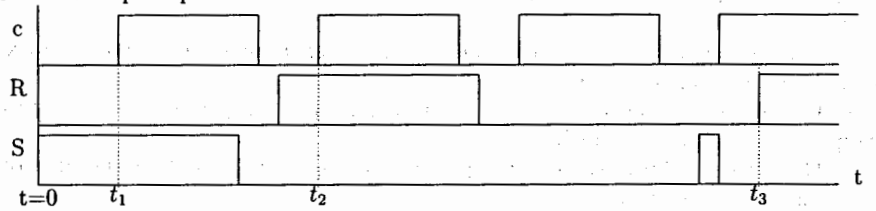




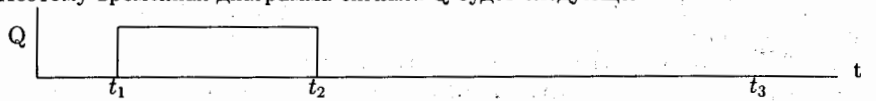
Здесь сигнал c, если он равен 1, отпирает &-элементы и дает возможность сигналам a_1, a_2 воздействовать на RS-триггер. Если c равен 0, &-элементы закрыты и на RS-триггер поступает сигнал 0, на который он не реагирует.

Справа помещено условное графическое обозначение RS-триггера, синхронизируемого сигналом c.

УПРАЖНЕНИЕ. Даны сигналы X и Y. В течение времени [0,t] эти сигналы поступают на S и R входы RS-триггера, синхронизируемого сигналом c. Начертить временную диаграмму Q-выхода этого триггера.
Решение. Начертим диаграмму входных сигналов и будем считать, что начальное состояние триггера есть 0.



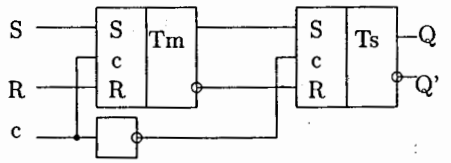
Хотя сигнал S поступает на триггер с 0-го момента времени, его включение в 1-е состояние произойдет только в момент t_1 прихода первого синхроимпульса - выход Q будет равен единице. Поступивший далее R-сигнал сможет переключить триггер в 0-е состояние только в момент t_2, и в этом состоянии триггер так и останется - следующий S-сигнал не сможет переключить триггер, а следующий сигнал R в момент t_3 лишь продолжит нулевое состояние.
Поэтому временная диаграмма сигнала Q будет следующей:

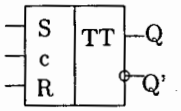


ДВУСТУПЕНЧАТЫЙ RS-ТРИГГЕР.
Обратные связи, на которых основаны замечательные свойства RS-триггера, имеют и негативные стороны - иногда порождают срывы и неустойчивость в его работе. Для придания конструкции триггера большей устойчивости прибегают к двухтактному режиму его работы:

- на первом такте происходит предварительное переключение состояния триггера;
- на втором окончательное.

Первый и второй такты сигнально разделены и работают в непересекающиеся моменты времени.
Двухтактный режим работы обеспечивает конструкция, состоящая из двух последовательно соединенных RS-триггеров, первый из которых (master) управляется прямым синхросигналом, а второй (slave) - инверсией этого синхросигнала. Ниже дана схема такого триггера.



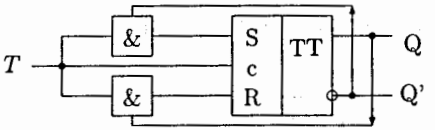


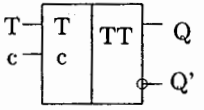
Справа находится графическое изображение 2-тактного триггера.

Т-ТРИГГЕР.
Очень важной и необходимой операцией является инверсия состояния триггера. Эту операцию можно осуществить, слегка изменив конструкцию RS триггера и используя для формирования команды инверсии информацию о текущем состоянии триггера, задаваемую сигналами Q и Q'.
А именно, сигнал инверсии T поступает на пропускные вентили, отпираемые так, что на выходах формируются сигналы:

$$a_1 = T \;\&\; Q'; \quad a_2 = T \;\&\; Q.$$

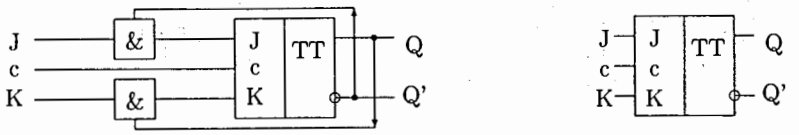
Если триггер был в состоянии "0" (т.е. $Q = 0,\ Q' = 1$), то при $T = 1$ выработается команда $a_1 = 1,\ \ a_2 = 0$, т.е. команда S (включить 1-е состояние).
Если же триггер был в состоянии "1" ($Q = 1,\ Q' = 0$), то при $T = 1$ выработается команда $a_1 = 0,\ \ a_2 = 1$, т.е. команда R.
Нулевой сигнал T состояния триггера не меняет.
Т-триггер как автомат отличается от стандартного RS-триггера только входным алфавитом $(0, T)$. Структурная схема Т-триггера будет следующей:





В схеме сигналы T и c объединены, но конечно они могут быть и раздельными.
Справа находится графическое изображение Т-триггера.
Так как опрокидывание (т.е. появление устойчивого состояния) происходит только после действия сигнала T, то можно сказать, что Т-триггер реагирует на задний фронт импульса T.

JK-ТРИГГЕР.
На основе RS- и Т-триггеров мы можем построить схему универсального триггера, способного работать в любом из этих режимов. Для этого мы можем воспользоваться тем, что комбинация входных сигналов RS-триггера $a_1 = 1,\ a_2 = 1$ является неиспользуемой. Объявив такую комбинацию сигналом T и соответственно реализовав ее, мы построим универсальный триггер.
Ниже приводятся структурная схема такого триггера (JK-триггер) и его графическое обозначение.



Итак, JK-триггер имеет 4 комбинации входных сигналов:

1. J=0, K=0 - состояние триггера не меняется;
2. J=1, K=0 - включается состояние 1 триггера;
3. J=0, K=1 - включается состояние 0 триггера;
4. J=1, K=1 - состояние триггера меняется на противоположное.

Опрокидывание JK-триггера происходит после окончания импульсов J и K.

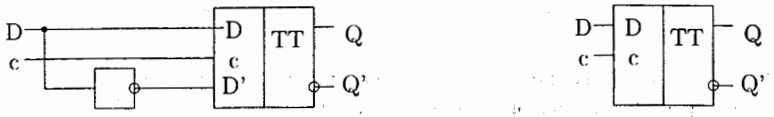
D-ТРИГГЕР.

В заключение упомянем еще одну конструкцию триггера: D-триггер. Это RS-триггер, на входы S и R которого подаются сигналы D и D' соответственно. Значение D=1 (тем самым D'=0) соответствует включению триггера в состояние "1"; D=0 (тем самым D'=1) - включению триггера в состояние "0", т.е. выходы Q,Q' D-триггера всегда повторяют входные сигналы D,D', но с задержкой на 1 временной такт.

Входные сигналы D-триггера обязательно должны иметь потенциальную форму в асинхронном режиме или быть не уже, чем синхроимпульс в синхронизированном.

D-триггер используется как для целей хранения информации, так и для сдвига сигналов по времени.

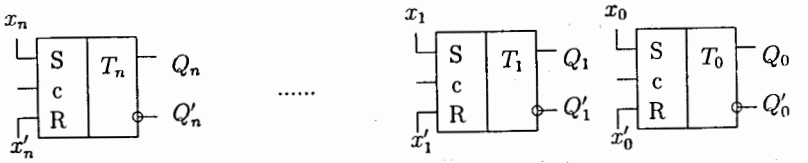
Структурная схема D-триггера и его графическое обозначение следующие.



РЕГИСТР ДЛЯ ХРАНЕНИЯ КОДА.

Если задан двоичный $(n+1)$-разрядный код $x_n....x_1x_0$, то его можно отобразить на $(n+1)$-разрядный регистр: последовательность из $n+1$ триггеров, из которых i-й включается в состояние x_i. Такой регистр хранит данный код, и этот код доступен для дальнейшего использования путем чтения: прямого - с выходов триггеров Q_i и инверсного - с выходов Q'_i.

Схема накопительного регистра может быть следующей.

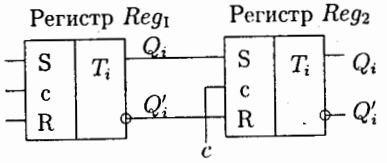


По сигналу c (или в асинхронном режиме) произойдет включение триггеров в нужное состояние. Вместо RS-триггеров для целей хранения информации можно использовать также и D-триггера.

КОПИРОВАНИЕ КОДОВ ПРЯМОЕ И СО СДВИГОМ.

Пусть требуется код, записанный на регистре Reg_1, скопировать на Reg_2. Это можно сделать, например, соединив QQ'-выходы триггеров регистра Reg_1 с SR-входами триггеров регистра Reg_2. Тогда сигнал c, подаваемый на синхровходы триггеров регистра Reg_2, сыграет роль команды требуемого копирования.

Ниже дан элемент схемы копирования i-го разряда регистра Reg_1 на i-й разряд регистра Reg_2. Копирование остальных разрядов осуществляется аналогично.



Если же требуется код, записанный на регистре Reg_3, переписать на регистр Reg_4 со сдвигом на один разряд влево или вправо, то схема реализации такого сдвига будет аналогичной; разница лишь в том, что QQ'-выходы i-го триггера регистра Reg_3 будут соединены с SR-входами не i-го триггера регистра Reg_4, а $(i+1)$-го (сдвиг влево), или $(i-1)$-го (сдвиг вправо).

При этом код, выталкиваемый за пределы регистра Reg_4, будет либо пропадать, и тогда на место освободившегося младшего (или, наоборот, старшего) разряда можно записать, например, нуль; либо он может быть циклически перенесен в младший (соответственно, в старший) разряд.

Роль команды такого сдвига, как и раньше, может играть, например, сигнал синхронизации c, подаваемый на регистр Reg_4.

Сдвиг не обязательно может быть применен к целому коду. Например, код двоичного числа -101.1101, изображающего десятичное число -5.9375, которое имеет двоичный вид $-.1011101 \cdot 2^{+11}$, может быть записан как 10111011101; здесь

цифра кода	1	0	1	1	1	0	1	1	1	0	1
номер разряда	10	9	8	7	6	5	4	3	2	1	0

10-й разряд - это знак числа (минус), 9-й - знак порядка (плюс), 7-8-й разряды - это порядок (двоичное 11, десятичное 3), разряды 0-6 это мантисса числа. При такой кодировке сдвигать целесообразно лишь мантиссу; сдвигать знаки и порядок смысла не имеет. Поэтому различают

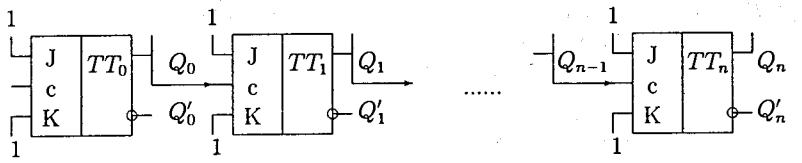
- арифметический сдвиг - сдвиг одной мантиссы;

- логический сдвиг - сдвиг всего кода.

Оба сдвига могут быть осуществлены вышерассмотренным способом.

СЧЕТЧИК.

Это еще одно устройство, играющее большую роль в схемах автоматики. Счетчик представляет собой автомат, имеющий регистр текущего двоичного кода, который увеличивается на 1 при поступлении входного сигнала. Ниже приводится пример счетчика, принцип действия которого основан на распространении положительного перепада сигналов по цепочке JK-триггеров: если i-й триггер меняет состояние "1" на "0", то на Q-выходе такого триггера возбуждается сигнал, опрокидывающий следующий $(i+1)$-й JK-триггер. Если же состояние переходит из "0" в "1", то сигнал не возбуждается.
Такой сигнал может быть произведен, например, дифференцирующим элементом и пороговым ограничителем или другими техническими приемами.



Счетчик работает следующим образом. Входным является синхросигнал, подаваемый на синхровход младшего 0-го триггера. На JK-входы всех триггеров подается импульсный сигнал "1", т.е. достаточно задействовать единичным сигналом синхровход, и произойдет опрокидывание триггера.
Поясним работу счетчика на четырехразрядном примере. Пусть начальным было состояние 0000. При подаче сигнала с на с-вход 0-го триггера он опрокинется, но так как переход был "0 - 1", то с его Q-выхода сигнал не поступит на вход с 1-го триггера. Текущее состояние счетчика будет 0001. Следующий сигнал с установит 0-й триггер в "0", но так как переход был "1-0", то с его Q-выхода будет снят положительный перепад, который поступит на синхровход 1-го триггера и опрокинет его. Поскольку переход 1-го триггера был "0 - 1", то дальнейшего распространения сигналов не будет, и текущим состоянием счетчика будет 0010. Следующий с опрокинет 0-й триггер, но не опрокинет 1-й, и текущим сигналом станет 0011; далее, следующий с опрокинет 0-й, 1-й и 2-й триггера и состояние станет 0100, и т.д.

Счетчик может быть построен и на основе общего подхода к реализации автоматов. Поясним это на примере счетчика более общего типа: с произвольным (не обязательно единичным и положительным) шагом и со счетом по модулю меньшему, чем 2^{n+1}.
ЗАДАЧА 1. Описать трехразрядный счетчик, работающий с шагом 2 по модулю 5 как автомат, т.е. при поступлении входных сигналов состояния счетчика будут такими: 0, 2, 4, 1, 3, затем снова 0; или в двоичном виде 000, 010, 100, 001, 011 и снова 000 и т.д.

Составим абстрактное описание автомата. Имеем:

- Входной алфавит: x - входной сигнал счетчика.
- Выходной алфавит: q - абстрактное значение состояния регистра счетчика и $q_2q_1q_0$ - его двоичное значение.
- Он же алфавит состояний.

Поскольку переключения счетчика происходят только при $x = 1$, функцию переключения состояний можем считать независимой от x. Ниже приведем таблицы абстрактного автомата и соответствующего двоично кодированного, карты Карно для минимизации функций включения следующих состояний Q_2, Q_1, Q_0 и полученные МДНФ.

q	Q
0	2
1	3
2	4
3	0
4	1
5	*
6	*
7	*

q_2	q_1	q_0	Q_2	Q_1	Q_0
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	0	0
1	0	0	0	0	1
1	0	1	*	*	*
1	1	0	*	*	*
1	1	1	*	*	*

	00	01	11	10
				1+
1		*	*	*+

$Q_2 = q_1 \& q'_0$

	00	01	11	10
	1+	1+		
1		*	*	*

$Q_1 = q'_2 \& q'_1$

	00	01	11	10
		1-		
1	1+	*+-	*+	*+

$Q_0 = q_2 \ V \ q'_1 \& q_0$

Здесь значения функций переключения состояний на наборах аргументов 101, 110, 111 мы заменили символом *, поскольку по условиям режима работы счетчика (счет по модулю 5) он не может попасть в эти состояния. Этот символ помог нам осуществить минимизацию СДНФ более эффективно.
Сигналы Q_2, Q_1, Q_0 поступают на входы триггеров регистра счетчика и переключают их нужным образом.

ЗАДАЧА 2. Построить автомат - кольцевой счетчик, т.е. $(n+1)$-разрядный регистр, на одном из разрядов которого записана единица, на остальных нули, и по входному сигналу эта единица циклически перемещается вдоль регистра (влево или вправо).

СИНТЕЗ АВТОМАТОВ ОБЩЕГО ТИПА.

Поясним эту процедуру на паре конкретных примеров.
ПРИМЕР 1. Пусть мы хотим построить автомат, управляющий работой лифта в некотором 3-этажном доме следующим образом: нажатая кнопка на этаже означает вызов лифта, т.е. подачу его на соответствующий этаж; если кнопки отжаты, лифт остается на том этаже, где находится; автомат реагирует только на одну нажатую кнопку.
Начнем с абстрактного автомата. Опишем алфавиты.

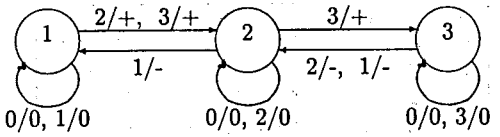
- Входной алфавит X: (1,2,3) - номер этажа с нажатой кнопкой и 0 - отсутствие вызова;

- Выходной алфавит Y: (+,-,0) - команды на мотор: двигаться вверх, вниз, не двигаться;
- Алфавит состояний Q: (1,2,3) - этаж, где находится лифт в данный момент.

Далее идут функции выработки выходной информации и переключения состояний. Начнем с их табличного задания.

x_i	q_i	y_i	q_{i+1}	x_i	q_i	y_i	q_{i+1}
0	1	0	1	2	1	+	2
0	2	0	2	2	2	0	2
0	3	0	3	2	3	-	2
1	1	0	1	3	1	+	2
1	2	-	1	3	2	+	3
1	3	-	2	3	3	0	3

А теперь построим изображение данного автомата в виде графа. Вершинами такого графа будут состояния. Итак, имеем граф:



Внутри вершин графа указаны состояния автомата, рядом с дугами надписи: какие входные сигналы вызывают данный переход и какая при этом вырабатывается выходная информация.

Теперь переходим к процедуре синтеза автомата. Начинаем с двоичной кодировки алфавитов.

Видно, что для двоичной кодировки значений входных и выходных сигналов и сигналов текущего состояния достаточно 2-разрядных кодов x_1x_0, y_1y_0 и q_1q_0. Таблица кодов будет следующей:

x	x_1x_0	y	y_1y_0	q	q_1q_0
0	00	0	00		
1	01	+	01	1	01
2	10	-	10	2	10
3	11			3	11

Номер текущего состояния, т.е. номер этажа, на котором находится лифт, мы будем хранить на 2-разрядном триггерном регистре; переключать триггера для следующего такта мы будем подаваемыми на входы триггеров командами S или R.

Мы видим, что функции выходов будут 2-разрядными функциями 4 двоичных переменных:

$$y_1 = y_1(x_1, x_0, q_1, q_0); \qquad y_0 = y_0(x_1, x_0, q_1, q_0).$$

Функции переключения состояний, т.е. сигналы на входы триггеров, будем обозначать как Q_1Q_0 - чтобы не путать их с сигналами с выходов триггеров q_1q_0.

Эти функции тоже будут 2-разрядными функциями 4 двоичных переменных:

$$Q_1 = Q_1(x_1, x_0, q_1, q_0); \qquad Q_0 = Q_0(x_1, x_0, q_1, q_0).$$

Каждая из этих 4 функций может быть реализована как, например, СДНФ. Для этого построим общую таблицу 4 функций.

$x_1x_0q_1q_0$	y_1	y_0	Q_1	Q_0
0000	*	*	*	*
0001	0	0	0	1
0010	0	0	1	0
0011	0	0	1	1
0100	*	*	*	*
0101	0	0	0	1
0110	1	0	0	1
0111	1	0	1	0

$x_1x_0q_1q_0$	y_1	y_0	Q_1	Q_0
1000	*	*	*	*
1001	0	1	1	0
1010	0	0	1	0
1011	1	0	1	0
1100	*	*	*	*
1101	0	1	1	0
1110	0	1	1	1
1111	0	0	1	1

Так как состояния 0 нет, то все 4 функции на наборах, где q_1q_0 равно нулю, будут не определены. Мы отметим это обстоятельство звездочками.

Мы могли бы по этим таблицам построить СДНФ всех 4 функций. Но такие представления содержали бы много термов и много переменных, а это лишние шины и контакты, а стало быть, и меньшая надежность и большая стоимость. Поэтому мы будем минимизировать эти СДНФ, т.е. строить МДНФ, например, с помощью карт Карно. Построим сначала y_1 и y_0 как МДНФ. Имеем

q_1q_0=	00	01	11	10
x_1x_0=00	*			
x_1x_0=01	*		1+	1+
x_1x_0=11	*			
x_1x_0=10	*		1	

q_1q_0=	00	01	11	10
x_1x_0=00	*			
x_1x_0=01	*			
x_1x_0=11	*+-	1+		1-
x_1x_0=10	*+	1+		

Левая карта для минимизации функции y_1, а правая - для y_0. Покрытия показаны на картах знаком +. В результате имеем

$$y_1 = x_1'x_0q_1 \vee x_1x_0'q_1q_0, \qquad y_0 = x_1q_1' \vee x_1x_0q_0'.$$

А теперь проделаем то же самое для функций Q_1 и Q_0

q_1q_0=	00	01	11	10
x_1x_0=00	*		1-+	1-+
x_1x_0=01	*		1-	
x_1x_0=11	*+	1+	1+-	1+-
x_1x_0=10	*+	1+	1+-+	1+-+

q_1q_0=	00	01	11	10
x_1x_0=00	*+	1+-	1-	
x_1x_0=01	*++	1+		1+
x_1x_0=11	*++		1-	1+-
x_1x_0=10	*			

Левая карта для минимизации функции Q_1, а правая - для Q_0. Карта для Q_1 допускает восьмерное покрытие - самая эффективная минимизация. Для Q_0 используем 2 покрытия из 4 клеток и 2 покрытия из 2 клеток. Имеем

$$Q_1 = x_1 \vee q_1q_0 \vee x_0'q_1,$$

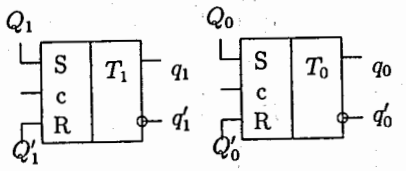
$$Q_0 = x_1'q_1' \vee x_0q_0' \vee x_1'x_0'q_0 \vee x_1x_0q_1.$$

Схема построенного автомата будет состоять из 2 частей:

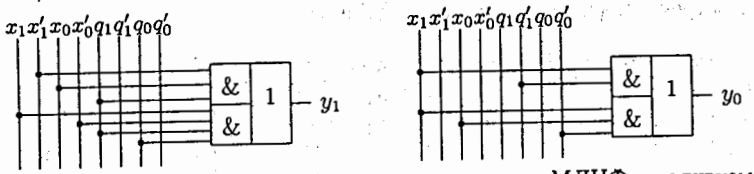
1. 2-разрядный триггерный регистр для хранения текущего состояния - номера этажа, на котором находится лифт;

2. комбинационная схема, реализующая функции y_1,y_0,Q_1,Q_0 и подводящая их на выходное устройство (управление мотором лифта) и на входы триггеров регистра.

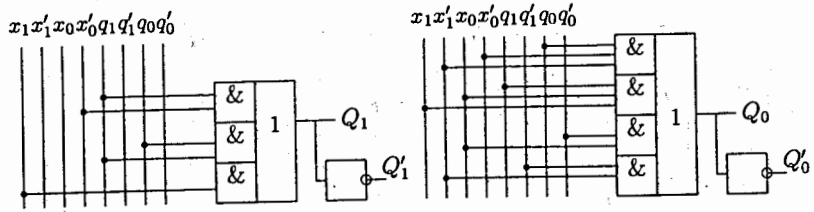
Схема регистра будет следующей.



Сигнал i-го тактового импульса будет подаваться на синхровходы c. С выходов триггеров будут сниматься сигналы q_1 и q_0. Сигналы Q_1,Q_0 будут подаваться на S-входы, а их инверсии - на R-входы триггеров и включать их в нужное состояние. Здесь можно было использовать D-триггера, так как все их состояния включаются принудительно.
Далее, комбинационная схема для реализации функций y_1, y_0:



Сигналы x_1, x_0 - это сигналы от кнопок на этажах. МДНФ, реализуемые данными схемами, это функции $y_1 = x_1'x_0q_1 \vee x_1x_0'q_1q_0$ и $y_0 = x_1q_1' \vee x_1x_0q_0'$.
И, наконец, комбинационная схема для реализации функций Q_1, Q_0:



МДНФ, реализуемые данными схемами, это функции $Q_1 = x_1 \vee q_1q_0 \vee x_0'q_1$ и $Q_0 = x_1'q_1' \vee x_0q_0' \vee x_1'x_0'q_0 \vee x_1x_0q_1$.

ПРИМЕР 2. Попробуем создать автомат, который управляет работой светофора. Этот светофор автоматически переключается сигналами таймера, следующими через определенные промежутки времени, и может принудительно включить зеленый свет для пешеходов (тем самым красный для автомобилей) по сигналу от кнонки на столбе

светофора (такие светофоры для удобства пешеходов в обилии имеются в Западной Европе).
Онишем абстрактный автомат.

- Входной алфавит X: (0-сигнал от таймера; 1-сигнал от кнопки);
- Выходной алфавит Y: (R,G,Y) - включаемый цвет светофора, R - красный, G - зеленый, Y - желтый.
- Алфавит состояний Q: (R,G,Y, J) - текущий цвет светофора, R - красный, G - зеленый, Y - желтый после красного, J - желтый после зеленого.

На этом примере мы видим, как работает механизм памяти у автомата. Благодаря состояниям автомат знает, какой цвет включать после желтого: красный или зеленый. Комбинационная схема здесь не смогла бы принять правильное решение.
Выпишем таблицу значений функций выработки выходной информации и переключения состояний.

x_i	q_i	y_i	q_{i+1}		x_i	q_i	y_i	q_{i+1}
0	R	Y	Y		1	R	R	R
0	G	Y	J		1	G	R	R
0	Y	G	G		1	Y	R	R
0	J	R	R		1	J	R	R

Переходим к процедуре синтеза автомата. Итак, начинаем с двоичной кодировки алфавитов.
Видно, что для двоичной кодировки значений входного сигнала достаточно 1-разрядного кода x_0, а для выходных сигналов и сигналов текущего состояния 2-разрядных кодов y_1y_0 и q_1q_0. Таблица кодов будет следующей

x	x_0		y	y_1y_0		q	q_1q_0
0	0		R	00		R	00
1	1		Y	01		Y	01
			G	10		G	10
						J	11

Номер текущего состояния, т.е. цвета светофора, будем хранить на 2-разрядном триггерном регистре.
Видно, что функции выходов будут 2-разрядными функциями 3 двоичных переменных:

$$y_1 = y_1(x_0, q_1, q_0); \qquad y_0 = y_0(x_0, q_1, q_0).$$

Функции переключения состояний будем обозначать, как и ранее. Q_1Q_0. Эти функции тоже будут 2-разрядными функциями 3 двоичных переменных:

$$Q_1 = Q_1(x_0, q_1, q_0); \qquad Q_0 = Q_0(x_0, q_1, q_0).$$

Каждая из этих 4 функций будет реализована, как и ранее, как МДНФ. Для этого построим общую таблицу 4 функций.

$x_0q_1q_0$	y_1	y_0	Q_1	Q_0
000	0	1	0	1
001	1	0	1	1
010	1	0	1	0
011	0	0	0	0

$x_0q_1q_0$	y_1	y_0	Q_1	Q_0
100	0	0	0	0
101	0	0	0	0
110	0	0	0	0
111	0	0	0	0

Строим МДНФ для y_1 и y_0 с помощью карт Карно

$q_1q_0=$	00	01	11	10
$x_0=0$		1		1
$x_0=1$				

$q_1q_0=$	00	01	11	10
$x_0=0$	1			
$x_0=1$				

Левая карта для минимизации функции y_1, а правая - для y_0. Имеем

$$y_1 = x_0'q_1'q_0 \vee x_0'q_1q_0',$$

$$y_0 = x_0'q_1'q_0'.$$

Аналогично строим МДНФ для Q_1 и Q_0:

$q_1q_0=$	00	01	11	10
$x_0=0$		1		1
$x_0=1$				

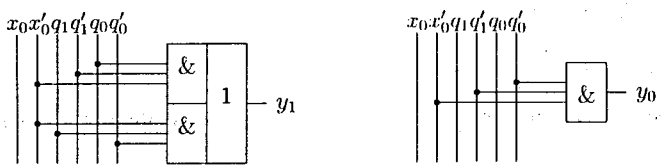
$q_1q_0=$	00	01	11	10
$x_0=0$	1+	1+		
$x_0=1$				

Левая карта для минимизации функции Q_1, а правая - для Q_0. Покрытия показаны на картах знаком +. В результате имеем

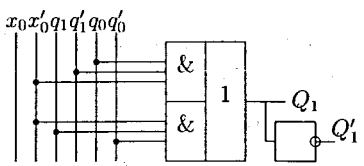
$$Q_1 = x_0'q_1'q_0 \vee x_0'q_1q_0',$$

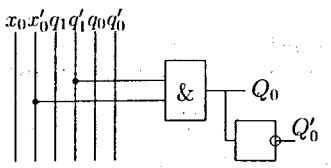
$$Q_0 = x_0'q_1'.$$

2-разрядный триггерный регистр для хранения номера состояния автомата будет тот же самый, что и в вышеприведенном примере, а комбинационная схема для реализации функций y_1, y_0 будет следующей:



И комбинационная схема для реализации функций Q_1, Q_0:





МИНИМИЗАЦИЯ АВТОМАТОВ.

В то время как при проектировании автомата заранее задан тип входной и выходной

информации, требуемой от этого автомата, вопрос о состояниях часто бывает неясен. Неясно, что брать за состояния; сколько их брать. Рассмотрим пример двух автоматов.

У них одинаковые входные a,b и выходные 0,1 алфавиты, но разные алфавиты состояний: у 1-го 1,2,3,4, у 2-го 1,2,4.

1-й автомат					2-й автомат			
x_i	q_i	y_i	q_{i+1}		x_i	q_i	y_i	q_{i+1}
a	1	0	2		a	1	0	2
b	1	0	4		b	1	0	4
a	2	0	3		a	2	0	2
b	2	0	1		b	2	0	1
a	3	0	2		a	4	1	2
b	3	0	1		b	4	1	1
a	4	1	3					
b	4	1	1					

Можно обратить внимание на то, что хотя "жизнь" 1-го автомата более разнообразная, если объединить состояния 2 и 3 1-го автомата, то выходная информация, производимая автоматами, совершенно одинакова и обусловлена одними и теми же входами. С чисто утилитарной точки зрения эти два автомата для нас одинаковы, или, формально, эквивалентны по выходной информации, но 2-й автомат проще по исполнению.

С утилитарной точки зрения реализация 1-го автомата избыточна. Процедура редукции автомата к схеме, в которой подмножества эквивалентных по выходам состояний заменены на единичные состояния, называется минимизацией автомата.

Еще более очевидный элемент избыточности - это т.н. недостижимые состояния, т.е. такие, в которые автомат с данными характеристическими функциями и данными входными алфавитами никогда не попадет. Выявление таких состояний осуществляется путем перебора всевозможных траекторий, ведущих из одних состояний в другие. При этом недостижимыми могут, в принципе, быть группы состояний, т.е. автомат может распасться на несколько подавтоматов, работающих независимо друг от друга. Но исключение таких состояний формальными методами нецелесообразно. Здесь необходим анализ первоначального проекта автомата и его преобразование на основе содержательных выводов.

РАЗВИТИЕ АВТОМАТОВ

МАГАЗИННЫЕ АВТОМАТЫ.

Автоматы явились устройствами, способными реализовать исключительно широкий класс алгоритмов, но все-таки существует большое множество задач, для решения которых автоматы в принципе не пригодны.

Рассмотрим следующую задачу. Имеется информационная цепочка:

$$2+3+2+9+4+6+3+6+....+7+5+8.$$

Требуется: создать устройство, способное в автоматическом режиме вычислить это выражение. Это можно сделать с помощью простейшего автомата, состоящего из регистра текущей суммы и комбинационной схемы, добавляющей к ней прочитанный символ, если это цифра, и игнорирующей его, если это знак +.

Но ситуация меняется радикально, если цепочка имеет, например, вид

$$2*3+3+2+9*8*4*3*5+4+6*7+3*4*9*1+6+....+7*2+5*3*4+8. \quad (1)$$

Здесь требуется (для вычисления произведений) запоминание вперед, причем число запоминаемых символов заранее не ограничено. Другими словами, памяти о прошлом опыте, которую автомат имеет на основе состояний, здесь недостаточно. Память на основе состояний - это статическая внутренняя память, способная удержать лишь весьма ограниченное количество информации. Для решения произвольно широкого класса задач требуется дополнительно внешняя динамическая память, способная расширяться неограниченно, чтобы автомат оставался конечным.

В общем, требуется обобщение понятия конечного автомата. Это обобщение должно не только коснуться памяти, но и вообще пересмотреть формы, структуру и функции организации как входной, так и выходной информации. В частности, мы должны допустить поступление входной информации как в регулярном, упорядоченном режиме, так и в случайном, по сигналам прерываний. Это не просто абстрактный переход от детерминированного автомата к недетерминированному, а моделирование реалистического взаимодействия субъекта и объекта, в котором случайность и предсказуемость перемешаны не формально, а содержательно.

Далее, выходная информация может строиться по очень сложным алгоритмам, т.е. автомат может представлять собой сложную сетевую, в частности, иерархическую структуру, состоящую из семейств подавтоматов; входная и выходная информация может быть как мгновенной, так и предназначенной для длительного хранения.

Характеристические функции автомата могут быть перенастраиваемыми самим же автоматом. И появляется новый тип наименований - ссылка на имя переменной (сигнала), иначе адрес, указатель и т.д.

Мы рассмотрим здесь 3 обобщения автоматов: это магазинный автомат (самое простое) и самые широкие - машина Тьюринга и компьютер; мы ограничимся простейшей формой последнего - это т.н. дискретный преобразователь информации Глушкова.

Магазинный автомат, чаще называемый автоматом с магазинной памятью (МП-автомат), в дополнение к обычным принадлежностям автомата имеет внешнюю

динамическую память, заполнение которой символами напоминает зарядку патронами винтовочного магазина (откуда и термнология), т.е. работающую по принципу: первым вошел, последним вышел. Символы в магазин автомата записываются последовательно, а извлечение их из памяти производится с конца, т.е. с последнего запомненного символа. Впрочем, последний символ в МП-автомате называется его вершиной. Отсутствие информации в магазине обозначается специальным символом: ϵ или пустым символом. Чтение из автомата может быть пассивным (выдача копии символа на вершине) и активным (выемка вершинного символа из магазина).

Итак, продумаем алгоритм, как с помощью МП-автомата мы сможем вычислить выражение (1). Сегодня при проектировании алгоритмов стал привычным стиль, свойственный программированию. Мы тоже воспользуемся техникой составления программ, например, на языке C++. В принципе, языки программирования возникли и развивались в русле идей и методов, принятых при конструировании автоматов, так что в них имеется вполне достаточно средств, непосредственно пригодных для моделирования структуры и функции МП-автомата. Так как в нашем случае открытым является только вопрос о состояниях МП-автомата (входной и выходной алфавиты очевидны), то мы воспользуемся SWITCH - оператором языка C++, case-опции которого и есть то, что соответствует понятию состояния автомата. Использование этого по существу автоматного понятия оказалось очень успешным приемом проектирования программного обеспечения; его можно рассматривать как простейший пример SWITCH (или automata-based programming)-технологии.

Но сначала несколько упростим задачу, так как реализация арифметических действий, не меняя принципиальную сторону дела, серьезно усложнит техническую форму автомата. Перепишем (1) так:

$$00+0+0+11001+0+11+0011+1+10+100+1= \quad (2)$$

И пусть наш алгоритм будет заключаться в следующем: мы читаем символы строки (2), трактуем операцию + как дизъюнкцию, а примыкание символов как их произведение и вычисляем текущую сумму этих произведений до тех пор, пока не будет прочитан символ =.

В ходе вычисления суммы мы должны регулярно опознавать слагаемые суммы - группы сомножителей. Для этого мы записываем в магазин примыкающие цифры, пока не встретится знак +, после чего перемножаем (фактически логически) запомненные цифры и добавляем их к текущей дизъюнкции.

На языке C++ соответствующая программа будет иметь вид:

```
#include <iostream.h>
  int stack[100];                 /* Магазин (по-английски stack) */
  int n=0;                        /* Счетчик магазина */
  int sum=0;                      /* Сумма (\ref{eq12}) */
  int pr;                         /* Текущее произведение */
  int i,j,ic; char ch;
  char* tape="1111+0+11001+1+10+0011+0+11+100=";
```

```
 void push(int c){stack[n++]=c;}       /* function: в магазин */
 int pop(){return stack[--n];}         /* function: из магазина */
 int mult(int a, int b){return a*b;}   /* function: * */
 int add(int a, int b){return a+b;}    /* function: + */
 void make_item()                      /* function: сформировать слагаемое */
  {int ns,k; ns=n;
   for(k=0; k<ns; k++)
    {j=pop(); pr=mult(j,pr);
    }
   sum=add(sum,pr); pr=1;
  }

main()
 {ic=0; pr=1; while(ch != '=')
  {ch=tape[ic]; ic++; j=(int)ch-48;  /* Имитация входной инфо */
   switch (ch)                       /* Состояния: реакция на входы */
   {case '0': {push(j); break;}
    case '1': {push(j); break;}
    case '+': {make_item(); break;}
    case '=': {make_item(); break;}
   }
  }
  cout << "sum= " << sum << '\n';
 }
```

А теперь дадим формальное описание этого автомата на привычном нам языке. Итак, алфавиты

- входной: символы 0,1,+,=;
- выходной: переменные sum,pr;
- магазинный: целые числа 0,1 и вершина магазина;
- состояний: 0,1,2 и 2 дополнительных: стартовое s_0 и финальное s_f; в стартовом состоянии автомат сразу включает состояние 0, в финальном работа автомата заканчивается.

Таблица МП-автомата будет иметь вид:

x_i	m_i	q_i	y_i	q_{i+1}	m_{i+1}
0	ϵ	0	ϵ	0	0
1	ϵ	0	ϵ	0	1
+	ϵ	0	ϵ	1	*stack[n]
=	ϵ	0	ϵ	2	*stack[n]
ϵ	0	1	*pr=0	1	*stack[n]
ϵ	1	1	*pr	1	*stack[n]
ϵ	ϵ	1	*sum=*sum $\vee$ *pr	0	ϵ
ϵ	ϵ	2	*sum=*sum $\vee$ *pr	3	ϵ

Мы используем сокращенную таблицу МП-автомата, так как реакция на входные символы осуществляется только в состоянии 0, а на магазинные только в 1. Обозначения: m_i - текущее значение вершины магазина при чтении, команду записи в магазин будем обозначать M_i; m_{i+1} - значение вершины на следующем такте. Знак ϵ обычно служит для обозначения пустого символа или пустой операции; если он находится на вершине магазина, это означает, что магазин пуст.

Мы имеем 3 элемента внешней памяти: stack, sum и pr.; мы не знаем их конкретного содержания в каждый тактовый момент, но они адресуемы и мы можем использовать ссылку (указатель) на них: *stack, *sum и *pr. Как в языке C++, если в операции использован указатель, то действие осуществляется над переменной, на которую он указывает.

Двоичная кодировка составляющих МП-автомата имеет вид:

x	x_1x_0		y	y_0		q	q_1q_0	m	m_1m_0
0	00		p	0		0	00	0	00
1	01		s	1		1	01	1	01
+	10					2	10	ϵ	10
=	11					3	11		

Выходными переменными будут p - зануление произведения, s - сигнал корректировки суммы.

Для хранения текущего значения суммы и произведения мы будем использовать 2 триггера; магазин будет пока просто частью внешней памяти, в технические детали которой мы пока входить не будем.

Перепишем таблицу МП-автомата, используя уже двоичную кодировку.

x_1x_0	m_1m_0	q_1q_0	y_0	Q_1Q_0	M_0
00		00		00	0
01		00		00	1
10		00		01	
11		00		10	
	00	01	0	01	
	01	01		01	
	10	01	1	00	
	11	01	*	*	
		10	1	11	

Функция y_0 не зависит от входной информации; это либо p - единичный сигнал на элемент pr при значении $m_1m_0 = 00$, $q_1q_0 = 01$, либо s - на sum при $m_1m_0 = 10$ и $q_1q_0 = 01$ или $q_1q_0 = 10$.

Построим карты Карно для обоих сигналов p и s:

q_1q_0=	00	01	11	10
m_1m_0=00		1+		
m_1m_0=01		*+		
m_1m_0=11	*	*	*	*
m_1m_0=10				

q_1q_0=	00	01	11	10
m_1m_0=00				
m_1m_0=01		*		1+
m_1m_0=11	*	*+	*	*+
m_1m_0=10		1+		

Отсюда получаем формулы для этих сигналов

$$p = m_1' q_1' q_0, \qquad s = m_1 q_1' q_0 \ \vee \ m_0' q_1 q_0'.$$

Сигнал записи в магазин равен единице только при условии $x_1x_0 = 00$ или $x_1x_0 = 01$ и $q_1q_0 = 00$. Поэтому

$$M = x_1' x_0' q_1' q_0' \ \vee \ x_1' x_0 q_1' q_0' \ = x_1' q_1' q_0'.$$

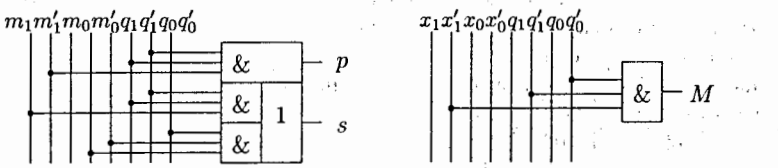
Q_1 равно 1 при условиях $x_1x_0 = 11$, $q_1q_0 = 00$ и $q_1q_0 = 10$. Поэтому

$$Q_1 = x_1 x_0 q_1' q_0' \ \vee \ q_1 q_0'.$$

Аналогично можно показать, что

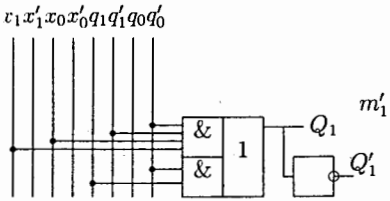
$$Q_0 = x_1 x_0' q_1' q_0' \ \vee \ m_1' q_1' q_0' \ \vee \ q_1 q_0'.$$

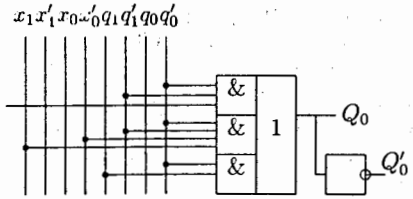
Ниже дана структурная схема данного МП-автомата. 2-разрядный триггерный регистр для хранения текущего состояния будет точно таким же, как и на рис.1 (автомат управления лифтом), так что здесь достаточно привести лишь комбинационную схему, реализующую функции p,s и M,



$$p = m_1' q_1' q_0; \qquad s = m_1 q_1' q_0 \ \vee \ m_0' q_1 q_0'; \qquad M = x_1' q_1' q_0',$$

и комбинационную схему для реализации функций Q_1, Q_0





$$Q_1 \ = \ x_1 x_0 q_1' q_0' \ \vee \ q_1 q_0'; \qquad Q_0 \ = \ m_1' q_1' q_0' \ \vee \ x_1 x_0' q_1' q_0' \ \vee \ q_1 q_0'.$$

УПРАЖНЕНИЕ. В обиходе под автоматом обычно понимается устройство, функционирующее более или менее автономно, т.е. не требующее вмешательства человека в управление им, за исключением старта и остановки. Огромное количество реальных автоматов в таком смысле представляют собой регуляторы, использующие положительную или отрицательную обратную связь. В качестве логической схемы такого устройства может быть взят МП-автомат. Задача: построить структурную

схему такого автомата. Указание: выходная информация запоминается в магазине, и на следующем такте в зависимости от нее вырабатываются команды переключения состояний.

МАШИНА ТЬЮРИНГА.

А теперь перейдем к методу реализации алгоритмов с помощью широкого обобщения понятия автомата - машины Тьюринга, или Т-машины.
Т-машина представляет собой автомат для реализации алгоритмов самого широкого класса, используемых человеком в интеллектуальной работе вообще.
Хотя она называется машиной, но на самом деле речь здесь идет о весьма абстрактных вещах, которые к современному понятию машины, т.е. преобразователя преимущественно вещества и энергии, прямого отношения не имеют. Это, скорее, преобразователь информации, реальные действия которого протекают под управлением человека за счет энергии последнего; но в принципе этот преобразователь может быть и материальным устройством, взаимодействующим с человеком в интерактивном режиме.
А теперь перейдем к ее формальному определению. Итак, Т-машина имеет:

- Внутреннюю память, определяемую ее состояниями, каждому из которых соответствует символ из ее внутреннего алфавита: $q_0, q_1, ..., q_n$. Среди этих состояний мы можем выделить одно, например, q_0 и считать его начальным, и другое, например, q_n и считать его финальным, или заключительным.

- Внешнюю память, представляющую собой некую емкость, например, ленту, разбитую на конечное (но неограниченное) число ячеек; в каждой такой ячейке может находиться символ из внешнего алфавита $a_0, a_1, ..., a_m$; один из них, например, a_0, может быть пустым символом.

- Головку чтения/записи; существует устройство, в каждый тактовый момент времени "обозревающее" определенную ячейку внешней памяти и способное по команде читать из нее записанный в ней символ или, наоборот, писать символ в нее.

- Устройство управления - УУ; в каждый тактовый момент времени УУ может выполнить команду, которая в зависимости от состояния Т-машины и прочтенного внешнего символа может включить новое состояние, записать какой-либо символ в ячейку и, возможно, передвинуть головку. Для сдвига головки имеются команды из алфавита:
  L - сдвиг влево, а R - сдвиг вправо на 1 ячейку; 0 - сдвиг не нужен.

При реализации какого-либо алгоритма Т-машина управляется программой, состоящей из команд типа

$$q_i a_j \ \rightarrow \ q_r a_t s_k, \quad 1 \le i, r \le n; \quad 1 \le j, t \le m; \quad s_k \in (L, R, 0).$$

Смысл команды следующий: если текущее состояние Т-машины q_i и прочтен символ a_j, то следует записать в текущую ячейку символ a_t, включить состояние q_r и выполнить команду сдвига головки s_k.

Приведем пример реализации Т-машиной какого-либо алгоритма. Пусть мы хотим сложить 2 числа. Будем считать, что лента до работы заполнена пустым символом a_0. Числа закодируем следующим образом: если x - целое число, то на ленту занесем его как последовательность x единиц, а кодом операции запишем +. Итак, на ленте записаны x единиц первого слагаемого, затем знак +, затем y единиц второго слагаемого; концом записи является пробел a_0.
Т-машина должна так переработать эти символы, чтобы итоговым кодом стали на ленте x+y единиц суммы; остальные символы должны исчезнуть.
В данном случае мы имеем: заданный внешний алфавит (a_0, 1, +), но внутренний алфавит мы строим сами. Пусть им будет следующий алфавит:
q_0 - начальные шаги до достижения первой единицы;
q_1 - движение вправо до достижения знака +;
q_2 - движение вправо до достижения крайне правого пробела a_0;
q_3 - финальное состояние.
Составим такую программу:

1. $q_0\ a_0 \rightarrow q_0\ a_0\ R$;
2. $q_0\ + \rightarrow q_3\ a_0\ 0$;
3. $q_0\ 1 \rightarrow q_1\ a_0\ R$;
4. $q_1\ 1 \rightarrow q_1\ 1\ R$;
5. $q_1\ + \rightarrow q_2\ 1\ R$;
6. $q_1\ a_0 \rightarrow q_3\ a_0\ 0$;
7. $q_2\ 1 \rightarrow q_2\ 1\ R$;
8. $q_2\ a_0 \rightarrow q_3\ a_0\ 0$.

Начальное состояние q_0: по команде 1 УУ Т-машины будет двигать головку вправо, и если читается символ a_0, ни он, ни состояние не меняются.
По команде 2, если в состоянии q_0 встретится знак +, то он будет затерт, а Т-машина остановится, т.е. включится финальное состояние q_3. Это правильно, так как такой случай означает, что решаемая задача была: 0+y.
Команда 3 работает, когда в состоянии q_0 встретилась 1; мы ее затираем, включаем состояние q_1 и движемся вправо.
Команда 4 продолжает движение вправо вдоль единиц, не меняя их.
Но команда 5 заменяет + на 1, включает состояние q_2 и продолжает движение вправо.
Команда 6 , встретив в состоянии q_1 знак a_0, остановит Т-машину, так как в этом состоянии следовало сначала встретить знак +, а отсутствие + в записи означает, что текущий код на ленте является суммой.
Команда 7 продолжает движение вправо вдоль единиц, не меняя их.
И, наконец, команда 8 остановит Т-машину - встретив финальный пробел, включит финальное состояние q_3.

Цель достигнута: затертая в самом начале 1 записана на место знака +, так что на ленте будет код, равный сумме x и y.
Код сумма оказался сдвинутым на ленте, но если бы требовалось хранить его на том же самом месте (начиная с той же самой ячейки), то мы могли бы расширить программу, добавив еще одно состояние и в нем организовав транспортировку крайней правой единицы влево на место перед крайней левой.
Мы рассмотрели очень простой пример, который не дает ответа на вопрос: а где гарантии, что произвольный алгоритм сможет тоже быть реализованным Т-машиной.
В математической теории алгоритмов положительный ответ на него вытекает из общепризнанной сегодня естественно-научной гипотезы, обычно формулируемой как "Тезис Черча" (или "Тезис Тьюринга"): класс алгоритмически (или машинно) вычислимых числовых функций совпадает с классом т.н. общерекурсивных функций.
Было показано, что эти общерекурсивные функции могут быть реализованы Т-машиной, т.е. были созданы соответствующие программы для нее. Однако эти программы слишком громоздки, и их изложение выходит за рамки данного пособия.
Обычно команды программы для Т-машины выполняются последовательно, но вообще-то этот порядок формально не регламентирован. В реальном мире, как мы знаем, объекты и процессы взаимодействуют между собой не только последовательно, но и параллельно. Параллелизм алгоритмов имеет место и в простых автоматах, так что, естественно, он допустим и для Т-машины.
Теория алгоритмов первоначально строится для целочисленных переменных. Результаты переносятся на случай произвольных чисел с помощью тех же самых алгоритмов, что и для целых чисел, после формализации понятия предела последовательности.

Хотя из рассмотренных примеров видно, что Т-машина не самое естественное и легкое средство для реализации повседневных алгоритмов, ее роль в решении различных проблем теоретической информатики - например, проблем алгоритмической разрешимости многих общематематических задач, оказалась огромной.
Что же касается использования идеи Т-машины на практике, то здесь прорывным инструментом автоматизации переработки самого широкого класса информации стал ее не менее универсальный, но неизмеримо более привычный и удобный аналог - компьютер, иначе ЭВМ, (микро)процессор и т.д.

СЕТИ ПЕТРИ.

Очень популярным обобщением простых автоматов были системы, построенные на моделировании работы головного мозга - т.е. представляющие собой сети соединенных между собой элементов, способные перерабатывать в зависимости от текущего состояния и перехода входную информацию в выходную информацию, но так, что реакция элементов подчинена "пороговой" логике (как в настоящем мозгу, где отклик вырабатывается лишь на раздражение определенной силы).
Одной из таких конструкций являются сети Петри (СП). Формализм СП (Petri nets) был создан в 1966 г. (Carl Adam Petri). Формально СП - это

двудольный ориентированный граф, моделирующий динамическую дискретную многокомпонентную систему, работающую в "пороговом" режиме (рис.1, классический пример).

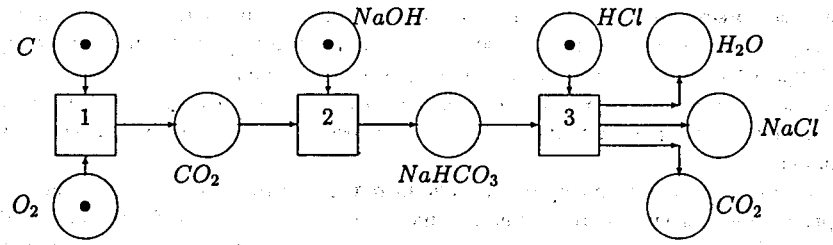


Рис.1. Сеть Петри как модель химического процесса

Главные элементы СП - это состояния P и переходы T. Состояния (называемые также позициями (place), условиями и т.д.) обозначаются на графе кругами, переходы (также события) прямоугольными фигурами. Эти элементы соединяются дугами A, задающими парам "позиция - переход" отношение "вход - выход". Соответственно, в переходах будут определены входящие и выходящие сигналы.
Черные точки в центрах кругов - это марки (token), передвигающиеся при переходах и задающие текущее состояние дел в сети (на рис.1 начальное состояние). Каждая позиция имеет некоторое (0,1,2,..) число марок. Распределение таких марок называется разметкой СП. Переход в СП срабатывает, если его входные дуги содержат достаточное (обязательно больше нуля) число марок. В случае срабатывания марки поглощаются выходной позицией для данного перехода. Срабатывание является атомарным, дискретным, асинхронным и недетерминированным актом.
Совокупность качеств СП делает их пригодным формализмом для моделирования конкурирующих распределенных систем.
Формально работа сети Петри описывается множеством последовательностей срабатываний и множеством реализуемых разметок позиций.
Состояние системы формируется в результате реализации локальных операций, называемых условиями реализации событий. Условие имеет емкость:

- Условие не выполнено - емкость равна нулю.
- Условие выполнено - емкость равна 1.
- Условие выполнено с n-кратным запасом - емкость равна n.

Определенная комбинация условий может стимулировать (fire) определенное событие, которое вызовет в свою очередь изменение условий. Некоторые события могут играть ингибирующую роль для других событий. Если условия ни для одного из переходов не реализованы, сеть переходит в заблокированное состояние.
СП нашли широкое применение при решении следующих задач:

- Software design.

- Управление потоками.
- Анализ данных.
- Параллельное программирование.
- Контроль надежности систем.
- Диагностика систем.
- Управление дискретными процессами.

Подробное изложение материала по СП можно найти, например, в книге: В.Котов. Сети Петри (Petri Nets). М.: Наука, 1984.

КОМПЬЮТЕР

Сегодня нет общепринятого определения последнего и нет общепризнанной истории его появления. Поэтому, даже сегодня нельзя дать уверенный ответ на вопрос: так был электромеханический релейный калькулятор, созданный Конрадом Цузе около 1940 года, компьютером или нет?

Впрочем, многие факты из практики работы с информацией известны, и на их основе можно все же попытаться дать определение компьютера.

Обычно историю создания компьютера излагают в контексте развития идей абстрактной теории алгоритмов (уже упомянутая Т-машина, "алгорифм" Маркова, общерекурсивная функция и т.д.). Но на самом деле первые реальные компьютеры не были столь прямолинейным следствием философско- математической идеологии. Это, кстати, отражено и в их названии - computer означает по-русски вычислитель, что отражало их первоначальное назначение.

В 30-е годы XX века были очень популярны демонстрировавшие высокую эффективность линейные аналоговые интеграторы. Беда, однако, была в том, что самые актуальные задачи, выдвигаемые практикой, имели, как правило, нелинейную формулировку. И, естественно, огромные усилия были приложены в начале 40-х годов прошлого века к созданию нелинейного аналогового интегратора. Однако в ходе проводимых работ по его созданию бесперспективность аналоговых методов для решения поставленной задачи стала очевидной. Возникла идея создать гибрид аналогового и цифрового подходов, и именно цифровые элементы убедительно показали свою уникальную адекватность задаче так, что аналоговая техника исчезла из дизайна сооружаемого устройства вообще.

И вот эти созданные в первой половине 40-х годов цифровые устройства дискретного действия, в обилии снабженные всевозможными известными и вновь изобретенными приемами, превратившими эти устройства в самые универсальные преобразователи информации предельно широкого класса, и были исторически первыми компьютерами.

Так какие же качества выделяют компьютер среди всех преобразователей информации максимальной эффективности? Можно ответить так: компьютер - это преобразователь широкого класса двоично-кодированной информации с помощью алгоритмов, которые реализуются как последовательность операций двоичной же логики; и данные, и сами алгоритмы помещаются в общей адресуемой памяти и формально между собой неразличимы. Последнее делает компьютер единственным преобразователем, способным непосредственно перерабатывать как данные, так и сами алгоритмы переработки этих данных.

Такому определению в общем-то не противоречит и тот факт, что среди реальных компьютеров некоторые имели дело с кодами иной, нежели двоичная, разрядности (троичная, например, или планируемая ныне 2^n-разрядность кодов квантовых компьютеров, и т.д.), так как все такие компьютеры работают на основе многозначной формальной логики, которая есть всего лишь простое расширение двоичной.

Мы также не увязываем понятие компьютера с электрической природой сигналов, на которых основана работа компьютера, так как это, судя по трендам развития последнего, является исторически преходящим моментом.

Как и Т-машина, компьютер способен вычислить любую общерекурсивную (GR) функцию. Но мы здесь в рамках пособия ограничимся лишь простейшей иллюстрацией идей, на которых построен компьютер, - мы рассмотрим ДПИГ:

ДИСКРЕТНЫЙ ПРЕОБРАЗОВАТЕЛЬ ИНФОРМАЦИИ В.М.ГЛУШКОВА.

Рассмотренный пример Т-машины дал нам идею программы - указаний вычислительному устройству, какие шаги предпринять и какие действия выполнить, чтобы реализовать нужный нам алгоритм. Но прежде всего мы должны формализовать эти шаги и эти действия.

Как было сказано, компьютер использует двоичную кодировку данных и операции двоичной логики. Следовательно, если мы покажем, что любые данные могут быть представлены в виде двоичного кода, а любые вычисления - как композиция операций двоичной логики, мы тем самым обоснуем идею универсального двоичного преобразователя информации.

Первое утверждение очевидно: мы записываем положительное число его прямым двоичным кодом, а отрицательное - его двоичным дополнительным.

Второе утверждение требует выяснения, каким образом мы сможем выразить основные арифметические операции через операции двоичной логики. Мы снова ограничимся целочисленными вычислениями, помня, что расширение арифметики принципиальный характер ее алгоритмов не меняет.

Итак, сложение и вычитание. Мы уже видели в главе о сумматоре, что обе операции могут быть реализованы через логические функции: отрицание, конъюнкцию и дизъюнкцию.

Остаются умножение и деление. Для начала возьмем 2 двоичных числа и умножим их, чтобы видеть, как работает процедура умножения двоичных чисел. Пусть требуется выполнить операции: 3 x 5 = ? (в двоичном виде: 0011 x 0101) и 15 : 3 = ? (1111 : 0011)

```
  0011              1111 : 11
x 0101              11     1
------              001    0
 0011               0011     1
0000                  11
0011                Результат = 101 = 5
0000
--------
= 0001111   = 15 (в десятичном коде)
```

Т.е. мы видим, что умножение 2 n-разрядных кодов сводится к последовательному добавлению к текущей сумме величины, которая на i-м шаге равна либо первому сомножителю (если i-й разряд второго равен 1), либо нулю, и последующему сдвигу первого сомножителя на 1 разряд влево. Если сомножитель отрицателен (или оба), то можно перемножить их прямые коды, а затем (по правилам логики) присвоить произведению нужный знак.

Следовательно, для реализации умножения необходимы операции сложения, сдвига, проверки условия и передачи управления (т.е. изменения порядка выполнения

операций), которые вполне могут быть осуществлены с помощью логических функций.
Аналогично, видно, что операция деления тоже может быть реализована через операции вычитания, сдвига, проверки условия и передачи управления. Сдвиг и порядок выполнения операций тоже реализуются логическими же функциями.
Из математики мы знаем, что любая аналитическая функция может быть вычислена с какой угодно точностью через отрезок соответствующего степенного ряда, т.е. через основные арифметические операции, а тем самым и через логические.
Символы и любые элементы графики тоже могут быть закодированы двоичными кодами, а операции с ними - объединение в целое, выделение или разбиение на части и т.д., могут быть реализованы как логические операции над соответствующими двоичными кодами.
Мы тем самым показали общерекурсивность (по терминологии Тьюринга) всех практически вычислимых функций относительно системы основных логических функций и можем переходить к типовому примеру построения практической схемы ДПИГ.

Каноническая структура синхронного ДПИГ состоит из двух частей: операционного и управляющего автоматов.

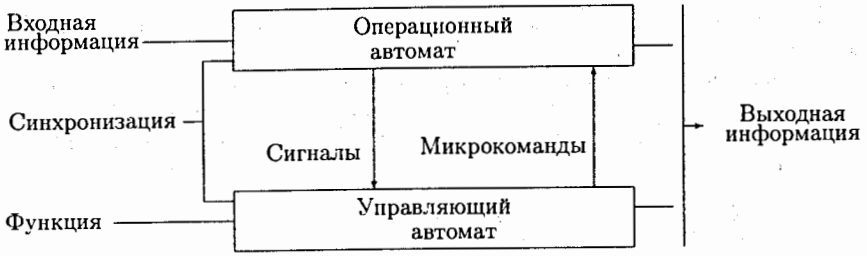


Рис.2. Общая схема ДПИГ

ДПИГ работает следующим образом. Данные поступают в ДПИГ по шинам "Входная информация", команда на выполнение алгоритма - по "Функция". Алгоритм реализуется последовательностью микрокоманд, регулируемой импульсами шины "Синхронизация". ДПИГ опознает код требуемой функции и включает соответствующую программу, микрокоманды которой поступают в "Операционный автомат"(ОА). В последнем находятся все устройства, отрабатывающие эти микрокоманды и тем самым реализующие шаги алгоритма. Расшифровку микрокоманд программы и обеспечение условий для их выполнения осуществляет "Управляющий автомат" (УА), который при этом может использовать информацию о ходе выполнения микрокоманд, поступающую по шине "Сигналы" (в частности, реагировать на запросы ОА). Выходной информацией ДПИГа является результат работы алгоритма и диагностика качества его выполнения (в частности, сообщения об ошибках).

БЛОК СИНХРОНИЗАЦИИ (БС)
Как ранее отмечалось, программа на компьютере - это последовательность команд, выполнение которых упорядочено во времени, и нам необходимо выяснить, как осуществляется привязка шагов программы к временным тактам.
Основой блока синхронизации является генератор тактовых (синхро) импульсов. Он производит цепочку сигналов $c_i = 1,\ i = 1, 2, 3, \ldots$ импульсной формы, задающих начала такта t_i, и если мы хотим, чтобы сигнал запуска отработки k-го шага p_k срабатывал именно в момент времени t_k, мы должны посылать этот сигнал через пропускной вентиль, на разрешающий вход которого подается синхросигнал c_k:

$$p_k = p_k \& c_k.$$

Такая синхронизация может быть осуществлена следующим образом. Для простоты будем считать, что команды исполняемых на данном ДПИГ программ используют 1 временной такт на отработку каждого шага (микрокоманды) данной команды. Разумеется, команды могут иметь различное число шагов. Для точной идентификации синхроимпульсов мы можем определить максимальное число шагов, требуемых командами данного ДПИГ (пусть это будет n), разбить в принципе бесконечную последовательность синхросигналов на группы по n синхросигналов в каждой и занумеровать каждый сигнал в группе номером от 1 до n. Такая организация, конечно, не является оптимальной, но нас пока интересует принципиальная сторона дела. Сказанное можно реализовать, например, такой логической схемой.

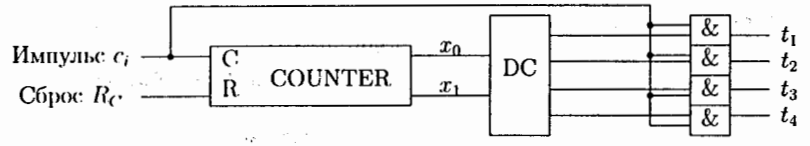


Рис.3. Схема БС

БС работает следующим образом. Синхроимпульсы генератора c_i поступают на C-вход счетчика COUNTER (R-вход служит для установки его в ноль). n-разрядный код счетчика поступает на дешифратор DC, на 2^n выходах которого последовательно появляются сигналы t_i. Счетчик считает по модулю 2^n, так что все сигналы c_i разбиваются на группы по 2^n сигналов в каждой (на схеме n=2). Обычно длительность переходных процессов в счетчике больше, чем у синхроимпульса, поэтому первый сигнал t_1 будет соответствовать начальному состоянию регистра счетчика. Сигналы с выхода DC имеют потенциальную форму; линейка пропускных вентилей справа преобразует их в импульсную.
КОД ОПЕРАЦИИ.
А теперь рассмотрим отработку информации, задаваемой командой "Функция". Общий формат команд на компьютерах 1-го поколения был ориентирован на ручное программирование в машинных кодах и для удобства программистов был поэтому следующим:

$$COP\ \ A_1\ \ A_2\ \ A_3$$

Смысл такой команды: взять числа, находящиеся по адресам A_1 и A_2, применить к ним операцию (например, сложение или вычитание), условно обозначаемую кодом COP, и записать результат по адресу A_3. Например, на созданной в 50-е годы отечественной машине М20 операция сложения могла выглядеть так (в восьмеричных кодах)

001 1067 2003 0533

Однако с развитием алгоритмических языков высокого уровня (АЛГОЛ, ФОРТРАН, и т.д.) формат машинных команд стали определять в соответствии с требованиями технической целесообразности. Например, программа сложения 2 чисел на языке Ассемблера (т.е. тех же машинных команд, но записанных в символической форме) для процессоров самых распространенных в настоящее время компьютеров серии IBM 80486 может выглядеть так:

```
MOV   ECX,A1    ; Послать число по адресу A1 в регистр ECX
ADD   ECX,A2    ; Добавить к ECX число по A2;
MOV   A3,ECX    ; переслать сумму из ECX по A3;
```

Сегодня максимально удобным и в то же время достаточно эффективным является программирование на языках высокого уровня, так что даже системные программисты предпочитают составлять программы не в машинных кодах (прямых или ассемблерных), а например, на языке C++. Посмотрим, как может выглядеть программа на C++, например, для нахождения в группе n чисел максимального, минимального и ближайшего к целому среднего значения чисел в этой группе.

```
#include <iostream.h>
main()
{int ar[100];                   /* массив ar содержит n (n < 100) чисел */
 int i,n,mx,mn,sr;
 n=50;                          /* положим n=50, например */
 mx=ar[0]; mn=ar[0]; sr=0;
 for(i=0; i<n; i++)             /* адресация в массиве на C++ с нуля */
  { if(mx<ar[i]) mx=ar[i];
    if(mn>ar[i]) mn=ar[i];
    sr=sr+ar[i];
  }
 sr=sr/n;
 cout << " max " << " min " << " sr" << '\n';
 cout << mx << ' ' << mn << ' ' << sr << '\n';
}
```

ПРИМЕР ПРОСТЕЙШЕГО ДПИГ.

Пусть даны 2 n-разрядных кода чисел X и Y; попробуем построить устройство для реализации следующих команд:

1. Сложить X и Y; COP этой операции пусть будет 01;
2. Вычесть из X Y; COP пусть будет 10;

3. Сдвинуть X влево на 1 разряд; COP пусть будет 11.

Свободный COP = 00 будет означать отсутствие операции: NOP (no operation).
Для реализации таких операций нам будут нужны: сумматор, счетчик и сдвиговый регистр.
Определим нужные микрокоманды. Мы можем ограничиться такими операциями, как:

- посылка кодов на входные регистры сумматора, счетчика, сдвигового и выходного регистров, т.е. разрешающих сигналов на соответствующие пропускные вентили;
- подача сигналов на счетный вход и вход сброса счетчика.

Тогда можно реализовать команды:

- сложения 3 тактами: посылка X и Y на сумматор, посылка результата на выходной регистр, завершение операции;
- вычитания 5 тактами: посылка инверсии Y на счетчик, добавление 1, посылка результата и X на сумматор, посылка результата на выходной регистр, завершение операции;
- сдвига 3 тактами: посылка X на сдвиговый регистр, посылка результата на выходной регистр, завершение операции.

Максимальное число тактов 5, т.е. блок синхронизации должен вырезать из последовательности c_i группы по 5 синхроимпульсов.
Входной информацией будут коды чисел X и Y.
Общей схемой этого устройства будет та, что изображена на рис.2, а здесь мы построим структурные схемы его отдельных узлов. Начнем с УА.

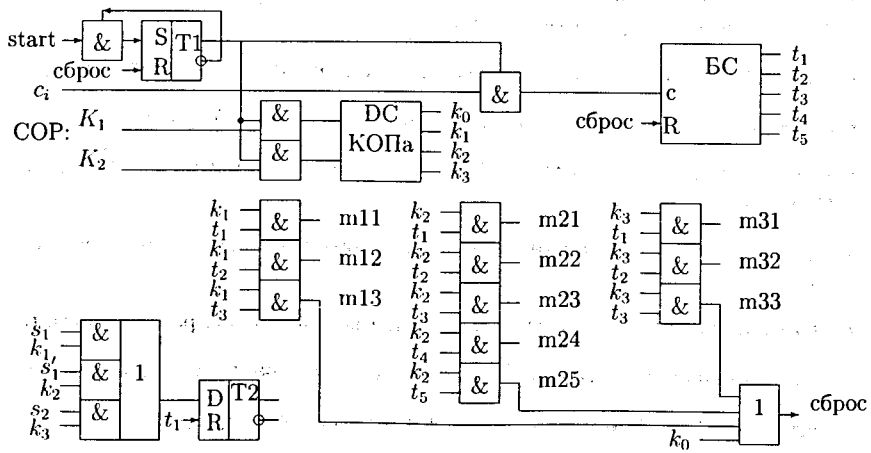


Рис.4. Структурная схема УА

Пояснения к схеме.
В состав УА входят:

- шина синхроимпульсов c_i;
- блок "Функция", на который подается 2-разрядный код операции k_1k_0;
- триггер "занято / свободно"; этот триггер блокирует доступ к ДПИГ во время исполнения текущей операции;
- БС (рис.2); нам нужны 5 тактов; для этого счетчик БС должен иметь разрядность как минимум n=3 либо считать по модулю 5 с шагом 1. Мы будем использовать только выходы t_1, t_2, t_3, t_4, t_5;
- шина сигнала запуска устройства Start;
- схема анализа сигналов от ОА и выработки микрокоманд на ОА.

Триггер Т1 задает режим "свободно - 0" / "занято - 1". В режиме "свободно" его Q-выход блокирует поступление синхроимпульсов на блок синхронизации БС и 2-разрядного КОПа K_1K_0 на дешифратор DC. В режиме "занято" Q'-выход Т1 препятствует поступлению нового сигнала start до конца выполнения операции.
Т1 устанавливается подачей сигнала start на S-вход; сбрасывается сигналом "сброс".
Счетчик БС опрокидывается сигналом c_i; устанавливается в нуль сигналом "сброс".
БС вырабатывает тактовые импульсы $t_1, ..., t_5$.
DC преобразует K_1K_0 в потенциалы k_1, k_2, k_3 - сигналы операций сложения, вычитания и сдвига.
3 линейки конъюнкторов по сигналу операции k_j на каждом такте t_i вырабатывают микрокоманды на операционный блок ОА - сигналы импульсной формы m_{ji}:

- m_{11} - для подачи кодов X и Y на входные регистры сумматора (открывая соответствующие пропускные вентили);
- m_{12} - для подачи суммы X+Y с регистра сумматора на выходной регистр;
- m_{13} - сигнал завершения операции - выработка сигнала "сброс";
- m_{21} - для подачи X на 1-й входной регистр сумматора, а инверсного Y на регистр счетчика;
- m_{22} - для увеличения счетчика на 1 - формирования дополнительного кода к Y;
- m_{23} - для подачи кода с регистра счетчика на 2-й входной регистр сумматора;
- m_{24} - для подачи разности X-Y с регистра сумматора на выходной регистр;
- m_{25} - сигнал завершения операции - выработка сигнала "сброс";
- m_{31} - для подачи кода X на сдвиговый регистр;

- m_{32} - для подачи сдвинутого кода на выходной регистр;
- m_{33} - сигнал завершения операции - выработка сигнала "сброс".

Команда k_0 однотактная - выработка сигнала "сброс".
С блока ОА УА получает сигналы s_1 и s_2; сигнал s_1 - значение (n−1)-го разряда регистра сумматора; если он равен единице при сложении или нулю при вычитании, то произошло переполнение (overflow). Сигнал s_2 - значение (n−1)-го разряда сдвигового регистра; если он равен единице, произошло выталкивание кода за разрядную сетку. Все такие случаи являются аварийным завершением операции.
Триггер Т2 служит индикатором нормального завершения (состояние 0) или аварийного. Аварийное состояние устанавливается сигналом $k_1\&s_1 \vee k_2\&s_1' \vee k_3\&s_2$.
В данной схеме все сигналы считаются идеальными: кроме того, мы пренебрегаем переходными процессами, так что анализ временных диаграмм в данном случае опущен - мы опираемся лишь на логику последовательности событий. Но в случае реальных устройств, разумеется, был бы необходим анализ точных временных соотношений между сигналами и реакциями на них.
А теперь перейдем к ОА. Он будет содержать следующее оборудование:

- сумматор с 2 n-разрядными входными регистрами RX и RY и один (n+1)-разрядный регистр для формирования суммы RS;
- счетчик Counter с n-разрядным регистром; регистр для формирования суммы;
- сдвиговый регистр RegSHL разрядности n+1.

Структурная схема ОА данного устройства будет следующей.

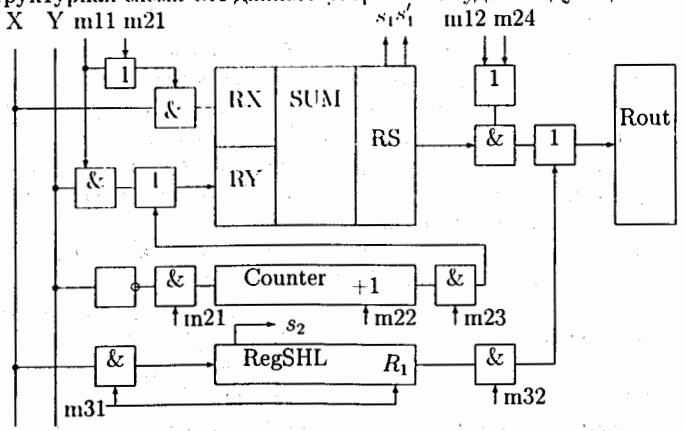


Рис.5. Структурная схема ОА

Все сигналы на схеме были объяснены выше. Поступающие микрокоманды mij управляют работой схемы; результаты операций записываются на выходной регистр Rout; информация об успехе/неуспехе операций (сигналы s_1, s_1', s_2) подается на УА.

Можно лишь добавить, что изображения шин данных X и Y и результатов операций над ними условные (для большей наглядности схемы). В действительности это n-мерные векторы шин, равно как и соответствующие пропускные вентили, на которые подаются отпирающие сигналы для прохождения данных к объектам назначения; n-мерны также и линейки конъюнкторов (дизъюнкторов).

УСЛОЖНЕНИЕ ПРИМЕРА. Построенное устройство было слишком примитивным. Чтобы приблизить его к модели сколько-нибудь зрелого микропроцессора, мы должны снабдить его чертами, которые бы позволяли ему автоматически отрабатывать последовательности команд, т.е. программы, и для этого оно должно брать данные и сами команды из адресуемой памяти.

Адресуемая память, как внешняя, так и внутренняя оперативная, с точки зрения логики - это последовательность n-разрядных регистров, каждому из которых присвоен номер, т.е. адрес.

Чтение информации из такой памяти логически осуществляется через мультиплексор: адрес читаемого кода подается на адресный вход мультиплексора, а на его выходе появится сам читаемый код. В нашем примере это был бы код X или Y, а микрокоманды отпирания пропускных вентилей (m11, m21, m31) посылали бы предварительно запросы на мультиплексор выдать требуемый код из ячейки памяти с заданным адресом на шины X или Y. Логически работа по посылке, например, X и Y на регистры сумматора сложнее - потребовались бы 4 отдельные микрооперации по пересылке X и Y. Но в остальном работа УА и ОА не изменилась бы.

Запись информации в адресуемую память можно осуществлять и с помощью мультиплексора, у которого обращены вход и выход данных; адресная часть будет работать так же, как в случае чтения.

Однако наиболее существенным изменением в конструкции устройства было бы создание аппарата отработки программы.

В нашем примере по команде start должно было бы включаться специальное устройство управления (УУ), которое бы вводило команды программы в оперативную память и отрабатывало их автоматически, без участия оператора. Так работало УУ уже упомянутой советской ЭВМ М-20: после нажатия кнопки "Ввод" на пульте ЭВМ программа вводилась с перфокарт в оперативную память, начиная с заданного на перфокартах же адреса, а затем начиналось последовательное чтение команд из оперативной памяти и их исполнение.

Последовательность чтения команд может изменяться самой же программой: специальные команды могут дать указание УУ изменить порядок чтения (как говорят, передать управление) в зависимости от условия (например, от значения результата предыдущей операции) или без всяких условий: например, если выполняемая команда была прочитана из ячейки A1, то следующей должна быть выполнена не команда из ячейки A1+1, а, скажем, из ячейки A2.

В качестве примера можно привести фрагмент ассемблерной программы для процессоров IBM 80486

```
        add     count,1     ; увеличить число по адресу count на 1
        mov     al,count    ; переслать из count в быстрый регистр AL
        cmp     al,20h      ; сравнить с 20H (32 десятичное)
        jl      ready       ; если < 32, передать управление ready
```



```
        jmp     get_out     ; иначе передать управление get_out
        ...     ..,
ready:  ...     ..,
        ...     ..,
get_out: ...
```

Здесь число из ячейки count сравнивается с 32 (16-ричное 20) и, если оно меньше, чем 32, происходит условный переход по адресу ready; если нет, то безусловный переход по адресу get out.

Все приведенные команды - обыкновенные n-разрядные (длиной кратные байту) адресуемые коды. Формат команд на этих процессорах: "КОП, быстрый регистр, адрес памяти", либо "КОП, адрес памяти, быстрый регистр". JL и JMP и есть команды условного и безусловного переходов.

Останов УУ происходит тоже по специальной команде.

Чтобы получить представление о том, как работают программы, рассмотрим конкретный пример C++ программы, вычисляющей x максимального значения заданной таблично функции f(x), x=1,2,3,...,50.

```
#include <iostream.h>
main()
{int ar[50];
 int i,xm,fmax;
 fmax=ar[0]; xm=0;
 for(i=0; i<50; i++)
  {if(fmax<ar[i])
    {fmax=ar[i]; xm=i;
    }
  }
}
```

Разумеется, так написанная программа на компьютере исполняться не может. Она предварительно должна быть преобразована в "машинную" форму, т.е. в последовательность двоичных команд. Такие преобразования осуществляются специальными программами - трансляторами, для которых C++ текст играет роль данных.

Видно, что для выполнения C++ программы нужны следующие команды:

- присвоение значения величине - запись его по адресу этой величины;
- сложение чисел;
- вычитание чисел (для сравнения двух чисел)
- сложение и вычитание самих адресов (изменение i, ar+i);
- запись самого адреса в ячейку по заданному адресу;
- команда условного перехода для реализации процедуры цикла (for);

- команда останова.

На этом примере видно, каков должен быть минимальный набор команд, необходимый для исполнения вышеприведенной программы.
А сейчас посмотрим, какой должна быть блок-схема простейшего УУ, достаточная для исполнения программ для реализации арифметических алгоритмов.
Итак, мы имеем ДПИГ с адресацией как данных, так и программ и работающий по таким программам. Для простоты опустим этап первичной загрузки программы с внешнего носителя (с перфокарт, магнитного носителя или интерактивно с пульта) и будем считать, что программа уже находится в памяти, начиная с адреса Ар. Будем считать, что это произведено сигналом start.
Таким образом, первое, что мы должны иметь, - это регистр для хранения адреса исполняемой команды; пусть это будет Radr.
Наши команды будем считать трехадресными.
В примере ДПИГ на рис.4,5 мы уже имели данные на шинах X,Y и код команды на шине COP. Программно управляемый ДПИГ должен позаботиться сам о том, чтобы извлечь команду из памяти по адресу, хранимому на Radr (в начале там будет Ар), выделить из нее КОП, выделить из нее адрес операндов, прочесть коды по этим адресам и подать КОП на шину COP, а коды операндов на шины X,Y. Дальнейшая отработка команд может проходить по той же схеме, что и выше, но предварительно УУ должно модифицировать содержимое регистра Radr - увеличить его на 1 (в устройствах с байтовой адресацией на количество байтов в слове команды). Если команда окажется командой перехода (условного или безусловного), она сама запишет адрес, на который нужно передать управление, на регистр Radr.
Предположим, что мы хотим на нашем дополненном адресацией и программным управлением ДПИГе (из примера на рис.4,5) выполнить следующую программу. Пусть в памяти по адресам A1,A2,A3 находятся числа. Мы хотим сложить их, сумму умножить на 4 и результат записать по адресу A4. Тогда программа следующего вида, расположенная в памяти, начиная с ячейки B1, выглядела бы так:

Адрес программы	КОП	Адрес1	Адрес2	Адрес3	Radr
B1	ADD	A1	A2	A4	B1
B1+1	ADD	A4	A3	A4	B1+1
B1+2	SHL	A4		A4	B1+2
B1+3	SHL	A4		A4	B1+3

Здесь содержимое регистра наращивалось бы последовательно, передач управления нет. 2 команды SHL (shift left) - сдвиг влево - выбраны потому, что на нашем ДПИГ нет команды умножения, но сдвиг целого числа влево на 1 разряд эквивалентен умножению на 2.

Распишем по тактам выполнение одной команды. Сначала следуют подготовительные такты работы УУ.

1. 1-й такт. Выборка команды из памяти, т.е. посылка адреса из Radr на адресный вход мультиплексора чтения и, тем самым, посылка выбранной команды на регистр команды Rc.

2. 2-й такт. Выделение кода операции и посылка его на регистр КОПа; выходы последнего - это шины COP; одновременно выделение адресов операндов и посылка их на адресные входы специальных мультиплексоров чтения операндов; пользоваться ранее упомянутыми нельзя - пришлось бы выполнить 3 последовательных такта чтения.

3. 3-й такт. Прием операндов с выходов данных мультиплексоров и запись их на регистры операндов; выходы последних - шины X и Y.

Далее начинается исполнение тактов арифметических операций блоком ОА нашего ДПИГа точно так же, как и в ранее рассмотренном случае, за исключением такта завершения операции. Запись результата в память и завершение выполнения команды осуществляются снова тактами УУ.
Таким образом, для команд сложения нам понадобятся 2 такта (сигналы t_4, t_5), а для вычитания 4 (сигналы t_4, t_5, t_6, t_7). Они будут управлять работой ОА.
Далее следуют такты записи результата операции в память (t_8) и подготовки к чтению следующей команды (t_9).

1. 8-й такт. Посылка адреса результата на адресный вход, а самого результата на вход данных мультиплексора записи.

2. 9-й такт. Завершение операции - увеличение адреса на Radr на 1.

Если бы наш ДПИГ имел команды перехода, то на Radr был бы послан адрес команды в памяти, которой следует передать управление (через вход счетчика A).
Итак, блок-схема нашего УУ будет следующей:

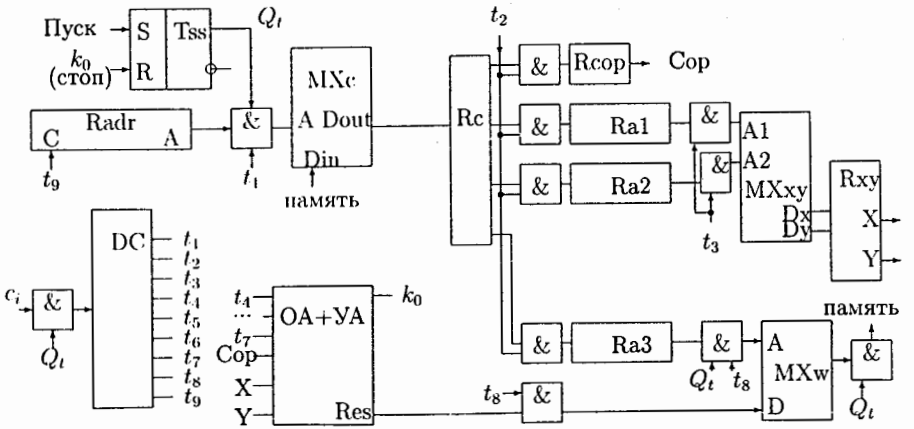


Рис.6. Блок-схема УУ

Пояснения к схеме. Регистр Radr будем считать регистром счетчика, на который можно записать новый код (по входу A) или увеличить старый на 1.
Триггер Tss при подаче сигнала "пуск" открывает пропускной вентиль для подачи

адреса текущей исполняемой команды с Radr на мультиплексор MXc, который читает из памяти код команды и подает его на регистр Rc; отсюда части этого кода (код операции и адреса операндов) подаются на регистры Rcop, Ra1, Ra2, Ra3; далее на мультиплексор MXxy, который читает из памяти коды X и Y и подает их на соответствующие шины. Сигнал Q_t разрешает работу УУ до тех пор, пока он не будет сброшен командой k_0 (стоп).

Затем для выполнения самих команд программ подключается блок "ОА+УА" - это описанный ранее ДПИГ (рис.4,5), в котором убран триггер "занято / свободно", а дешифрация синхроимпульсов и выработка тактовых импульсов перенесена в данное УУ (дешифратор DC).

Выполнение команд управляется тактовыми сигналами t_4, t_5, t_6, t_7, а свободный КОП = 00 УУ использует как команду останова. Результат выполнения команд с выходного регистра Res в блоке "ОА+УА" при разрешающем тактовом сигнале t_8 подается через мультиплексор записи MXw в память.

Тактовый сигнал t_9 завершает выполнение команды - повышает регистр счетчика Radr на 1; сигнал t_1 следующего цикла пошлет его на MXc, и т.д. Команда останова k_0 сбросит триггер "Tss" в "0" и заблокирует подачу адреса команды на MXc и тем самым прекратит работу (полезную) устройства. "0" на выходе Q_t заблокирует также всякую запись в память - точнее, записываться будет нулевой код по адресу 0.

ФОРМАЛЬНЫЕ ГРАММАТИКИ

Исходным понятием является буква (символ): a, x, N, c, 1, 5, +, - и т.д. Есть также пустая буква (пустой символ) - условное понятие, обозначающее букву, существующую лишь "виртуально", но не реально - как бы "незанятое место" для буквы.

Последовательность букв, рассматриваемая как целое, называется словом. Количество букв в слове - это длина слова. Буква иначе может быть названа словом длины 1. Слово, состоящее из пустых букв, будет пустым словом, т.е. словом длины 0.

Часть слова, т.е. часть последовательности букв слова, называется подсловом данного слова. Слово a есть подслово слова b, если b = cad, где c,d - слова (возможно, пустые) данного алфавита.

Слова могут быть терминалами (окончательными) и нетерминалами (предварительными). Терминальные слова состоят только из терминалов, нетерминальные содержат слова обоих типов.

Для слов определены следующие операции:

- конкатенация, т.е. соединение 2 или более слов в одно слово; например, пусть символ конкатенации будет $\oplus$; тогда выражение $abc \oplus 22 \oplus -p$ будет словом abc22-p; любое слово есть конкатенация букв;
- разбиение слова на подслова - обратная операция к конкатенации;
- замещение одного подслова слова на другое - важная операция в формализме работы со словами; замещение производится подстановками, которые описываются правилами данного языка типа $Z \rightarrow W$, где Z,W - слова данного языка; например, замена abc $\rightarrow$ DH преобразует слово abc22-p в DH22-p.

Слово не меняется от добавления (удаления) пустого символа (слова).

Буквы образуют алфавит, а слова, составленные из этих букв, словарь языка, будь то язык человеческого общения или любой формальный язык (программирования, инструктажа, графики, формальных описаний, музыки и т.д.).

Словам можно сопоставить их значения - в обычных языках это смыслы слов, в формальных языках это некоторые формальные коды. Смыслы (как формальные, так и неформальные) образуют иерархию, т.е. древовидную структуру. В обычных языках это родовые, видовые и т.д. отношения, в формальных - отношения между нетерминалами и терминалами.

Например, слово (2+8):2 имеет следующие смыслы: это арифметический терм и (уровнем пониже) это 5 - арифметическое значение этого терма.

ТЕРМИНАЛЫ И НЕТЕРМИНАЛЫ. Слова каждого конкретного языка порождаются с помощью грамматики - свода правил, по которым из букв алфавита этого языка строятся исходные слова, а из исходных все последующие. Эти слова делятся на две важные категории - терминалы и нетерминалы. Отношения между ними - это сложные отношения между общим и частным, между целым и частью и т.д.

С проблемой терминалов и нетерминалов, их роли в процедурах вывода вплотную

столкнулась кибернетика. Вопреки "естественному" мнению, что восприятие органами наших чувств дает нам полную информацию о нашем окружении, работы кибернетиков по созданию искусственного интеллекта сразу же выявили тот факт, что содержание этого восприятия есть лишь грубый суррогат этой информации, а точнее, ее сырой полуфабрикат.

Действительно, выглянув, например, из окна, мы увидим дома, улицы, прохожих, автомобили и т.д. Но самая простая проверка выявляет, что на самом деле мы видели не конкретные дома, а лишь "дома вообще", не автомобили, а "автомобили вообще"; на простейшие вопросы о конкретных чертах этих объектов (сколько окон у дома, какова марка автомобиля и т.д.) самый типичный ответ: "Не обратил(а) внимания". Оказывается, для формирования полноценного образа объекта одного восприятия мало - необходима еще интенсивная работа активного агента нашей психики - внимания, результатом работы которого является "терминализация" воспринятого нами образа (а точнее, весьма запутанной смеси образов различной степени "терминальности"). В приведенных примерах начальный "дом", "автомобиль", и т.д. были нетерминалами, а после внимательного их рассмотрения превратились в терминалы (англ. terminal означает оконечный) "двухэтажный дом", "автомобиль марки Лада" и т.д. Вообще, можно предположить, что наше восприятие внешнего мира работает следующим образом: ряд сигналов (в первую очередь самых интенсивных) по ассоциативной цепи порождает в нас первичную структуру нетерминалов, а далее с помощью логического и ассоциативного вывода начинается их терминализация. Другими словами, анализ подобной практики показал, что содержание нашего сознания - это не фотоснимок, не зеркальное отражение внешнего мира, а, скорее, картина, нарисованная художником.

Творческий характер "терминализации" особенно становится ясен, когда мы пытаемся выяснить сущность и свойства непривычного объекта в нестандартном окружении и нерутинной обстановке. С большой вероятностью мы можем пойти по неверному пути и усмотреть в данных "информацию", которой там нет и быть не может. В качестве примера можно сослаться, например, на данные аэро- и космической съемки земной поверхности. Эти данные - такая фантастическая мешанина терминалов и нетерминалов различных уровней, что для того, чтобы при их автоматическом анализе искусственному интеллекту разобраться в ней, требуется сложнейший и изощреннейший аппарат нового направления в математике - теории распознавания образов.

В этом новом научном направлении методология формальных языков проявила себя как весьма мощный и эффективный инструмент.

КЛАССИФИКАЦИЯ ГРАММАТИК ПО ХОМСКОМУ.

Грамматики идентифицируются по типу используемых и допустимых подстановок. Итак, пусть задан словарь языка, состоящий из множества нетерминалов V_n и терминалов V_t. В дальнейшем мы будем обозначать нетерминалы заглавными буквами, а терминалы строчными. Для иллюстрации того, как строится формальная грамматика, обратимся к неформальному образцу - естественному языку общения.

Возьмем предложение: "Эта прекрасная картина изображает очень живописный пейзаж". Она может быть построена с помощью следующих замещений:

ФРАЗА -> ПОДЛЕЖАЩЕЕ СКАЗУЕМОЕ ДОПОЛНЕНИЕ;
ПОДЛЕЖАЩЕЕ -> ИМЯ
ИМЯ -> АТРИБУТ ИМЯ
ИМЯ -> УКАЗАТЕЛЬ ИМЯ
ИМЯ -> картина
ИМЯ -> пейзаж
АТРИБУТ -> НАРЕЧИЕ АТРИБУТ
АТРИБУТ -> прекрасная
АТРИБУТ -> живописный
УКАЗАТЕЛЬ -> эта
СКАЗУЕМОЕ -> изображает
ДОПОЛНЕНИЕ -> ИМЯ
НАРЕЧИЕ -> очень

В ранее введенной терминологии здесь нетерминалами является следующее множество слов: (ФРАЗА, ПОДЛЕЖАЩЕЕ, СКАЗУЕМОЕ, ДОПОЛНЕНИЕ, ИМЯ, АТРИБУТ, УКАЗАТЕЛЬ, НАРЕЧИЕ).

Терминалы: (эта, прекрасная, картина, изображает, очень, живописный, пейзаж).

Посмотрим, как с помощью подстановок мы можем строить фразы данного языка. Возьмем подстановки:

ФРАЗА -> ПОДЛЕЖАЩЕЕ СКАЗУЕМОЕ ДОПОЛНЕНИЕ.
ПОДЛЕЖАЩЕЕ -> ИМЯ | СКАЗУЕМОЕ -> изображает
ДОПОЛНЕНИЕ -> ИМЯ | ИМЯ -> картина | ИМЯ -> пейзаж

Первый нетерминал всегда вершина иерархии нетерминалов. Таким будет ФРАЗА. Имеем: ФРАЗА -> ПОДЛЕЖАЩЕЕ СКАЗУЕМОЕ ДОПОЛНЕНИЕ.

Продолжаем подстановки:

ПОДЛЕЖАЩЕЕ СКАЗУЕМОЕ ДОПОЛНЕНИЕ -> ИМЯ изображает ИМЯ.

Следующий шаг подстановок дает:

ИМЯ изображает ИМЯ -> картина изображает пейзаж.

Если бы мы привлекли еще подстановку ИМЯ -> АТРИБУТ ИМЯ, то смогли бы построить фразу: прекрасная картина изображает пейзаж, или фразу: прекрасная картина изображает живописный пейзаж.

Но вот слово "эта очень изображает" не может быть получено из корневого нетерминала ФРАЗА с помощью перечисленных подстановок, значит, такое слово не принадлежит нашему языку, его словарю.

Далее, добавив к нашему языку пустое слово ϵ и подстановку СКАЗУЕМОЕ -> ϵ, мы смогли бы построить фразу: эта прекрасная картина очень живописный пейзаж.

С помощью грамматик можно решить две задачи:

1. образования слов данного языка;
2. распознавания - определения принадлежности или, наоборот, непринадлежности слов данному языку.

Для этого в множестве V_n должен быть выделен корневой нетерминал (будем обозначать его впредь как S), с которого начинаются все подстановки, а далее подставлять на место нетерминалов разрешенные правилами терминалы до тех пор, пока слово не окажется состоящим из одних терминалов.
Вторая задача решается "обратными" подстановками, т.е. заменой терминалов на разрешенные нетерминалы, пока не получится S.
Если система правил, т.е. грамматика, задана корректно, подстановки, прямые и обратные, должны рано или поздно привести к искомому результату.
Н.Хомский ввел следующую формализацию грамматик: формальная грамматика - это четверка понятий: (V_n, V_t, P, S), где V_n и V_t - множества нетерминалов и терминалов соответственно; P - множество подстановок, а S - корневой нетерминал. В зависимости от типа подстановок P различают 4 типа грамматик (символы t,N,G обозначают соответственно слова типа терминалов, нетерминалов и обоих типов).

1. Неограниченные грамматики - возможны любые варианты подстановок

$$G_1 \rightarrow G_2,$$

где G_1 - любое непустое слово, а G_2 - любое слово.
Пример. V_n = S,A,B; V_t = a,b
$S \rightarrow aAb \mid Ab \rightarrow aA \mid aA \rightarrow aB \mid B \rightarrow b$
Результат таких подстановок: слова aab и abb. Конечность словаря данного языка - следствие отсутствия рекурсии в подстановках.

2. Контекстные (иначе контекстно-зависимые) грамматики

$$t_1 N t_2 \rightarrow t_1 G t_2,$$

где t_1, t_2 могут быть пустыми, G - любое непустое слово.
Пример. V_n = S,A; V_t = a,b
$S \rightarrow aAb \mid Ab \rightarrow bAb \mid Ab \rightarrow bb \mid aA \rightarrow aAa \mid aA \rightarrow aa \mid$
Применяя данные правила, получим такие слова: $a^n b, ab^n$, n>0.

3. Бесконтекстные (иначе контекстно-независимые) грамматики

$$N \rightarrow G,$$

где G - любое непустое слово.
Пример. V_n = S V_t a,b $S \rightarrow ab \mid S \rightarrow aSb$.
Результат подстановок: $a^n b^n$, n 0.

4. Автоматные (иначе регулярные) грамматики:

$$N_1 \rightarrow tN_2, \text{ или } N \rightarrow t,$$

где t - любой (также и пустой) терминал.
Пример. V_n = S,A V_t a,b $S \rightarrow A \mid S \rightarrow bA \mid A \rightarrow aA \mid A \rightarrow b$
Результат: слова b, bb, aaa..ab и baaa...b.

Посмотрим, как решаются задачи образования слов и их распознавания в случае формальных языков при использовании абстрактных обозначений. Возьмем пример контекстной грамматики (пункт 2). Будем иметь:
S -> aAb -> abAb -> abbAb -> abbbAb -> ... -> ab^n.
Мы использовали подстановки $Ab \rightarrow bAb$ и $Ab \rightarrow bb$. Используя подстановки $aA \rightarrow aAa$ и $aA \rightarrow aa$, мы будем иметь
S -> aAb -> aAab -> aAaab -> aAaaab -> ... -> $a^n b$.
А теперь посмотрим, как можно решить задачу построения грамматики по заданным образам.
Пример. Построить грамматики разных типов на основе 3 образов: caaab, caab, baaab.

- бесконтекстная: $S \rightarrow cAb \mid S \rightarrow bAb \mid A \rightarrow aA \mid A \rightarrow a$
- автоматная: $S \rightarrow cA \mid S \rightarrow bA \mid A \rightarrow aA \mid A \rightarrow b$

А теперь посмотрим, как работают "обратные" подстановки для решения задачи о том, может ли данное слово быть словом языка с данными грамматиками. Возьмем слово baaab и случай бесконтекстной грамматики:
baaab -> baaAb -> baAb -> bAb -> S. Ответ положительный.
Возьмем далее слово caab и случай автоматной грамматики:
caab -> caaA -> caA -> cA -> S. Ответ тоже положительный.

РАСПОЗНАВАНИЕ ОБРАЗОВ С ПОМОЩЬЮ ФОРМАЛЬНЫХ ГРАММАТИК.

Алгоритм распознавания в случаях произвольных грамматик носит частично эвристический характер - при обилии выбора вариантов нет правила, по какому пути следует идти в первую очередь. Этого недостатка лишены автоматные грамматики, а именно, слова языка с автоматной грамматикой могут распознаваться автоматом $A = (X, Y, Q, y(x, q_i), q_{i+1}(x, q_i))$, у которого на место входного алфавита X, выходного Y, алфавита состояний Q и характеристических функций $y(x, q_i)$, $q_{i+1}(x, q_i)$ подставлены величины, определенные нижеследующей теоремой.

ТЕОРЕМА. Если $G(V_n, V_t, P, S)$ - грамматика автоматного языка, то автомат $A = (V_t, [], V_n, [], P(X, S))$ с начальным состоянием S и множеством конечных состояний способен распознать слова данного языка, начиная в состоянии S. Здесь [] - пустой символ, т.е. выходной информации данный автомат не производит.

При этом подстановка $A \rightarrow aB$ работает так: получив в состоянии A на входе читаемый символ a, автомат переходит в состояние B; подстановка $A \rightarrow a$ означает переход в одно из успешных конечных (в частности, начальное S) состояний: если a - последний символ, то слово распознано и конечное состояние - состояние успеха; иначе - признак того, что слово не принадлежит данному языку.

Пример. Автоматная грамматика: $S \rightarrow cA \mid S \rightarrow bA \mid A \rightarrow aA \mid A \rightarrow b$.
Автомат: (a,b, [], S,A, [], P(x,s))
Построим его табличное описание. Мы можем взять только такие комбинации

"вход" - "текущее состояние", как (b,S),(c,S),(a,A),(b,A). Прочие данным автоматом не опознаются, следовательно, слова в них языку автомата не принадлежат.

вход	текущее состояние	следующее состояние
b	S	A
c	S	A
a	A	A
b	A	конец "успех"

Слова языка с бесконтекстной грамматикой могут быть распознаны МП-автоматом; но в простых случаях можно использовать, как сказано выше, и эвристические приемы.

Пример. Язык целых арифметических формул.

V_t = 0..9+-*/(); V_n = S,Z,N,D.

Смысл нетерминалов очевиден: это "формула, знак, число, цифра".

$S \to SZS \mid S \to N \mid S \to (S)$

$Z \to +, -, *, /$

$N \to D \mid N \to ND$

$D \to$ 0..9

Это типичная бесконтекстная грамматика. Предположим, нам требуется проверить, является ли заданное слово словом этого языка - например, при трансляции вводимых в компьютер арифметических выражений необходимо проверять их на корректность написания.

Пусть, например, дано выражение (12+6)/3. Чтобы проверить формально, т.е. алгоритмическим путем, корректно ли оно, мы должны установить принадлежность этого слова нашему языку. Имеем

$$(12+6)/3 \to (DD+D)/D \to$$

$$\to (ND+N)/N \to (N+N)/S \to (S+S)/S \to (S)/S \to S/S \to S$$

Вывод: выражение написано правильно.

ПРИМЕР синтаксического описания и распознавания образов.

Рассмотрим две геометрических фигуры (С и М) на рис.7.

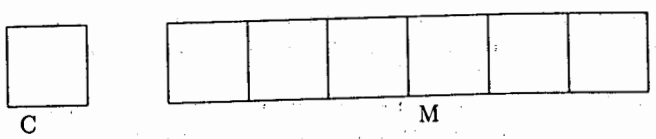


Рис. 7

Попробуем описать эти фигуры на формальном языке. В качестве терминалов целесообразно взять элементарные единицы, из которых построены фигуры на этом рисунке. Такие единицы должны быть как можно проще, чтобы из них можно было построить (хотя бы приближенно) линию любой формы, поскольку в реальных задачах формализации изображений объекты интереса могут иметь очень сложную форму (например, формализация отпечатков пальцев; в картотеках их хранятся

миллионы, и линии каждого образуют причудливый узор; задача автоматизации опознавания их является исключительно актуальной).

Наша задача проста, тем не менее мы тоже в качестве терминала возьмем самое простое - отрезок прямой линии в разных ракурсах и введем следующие обозначения:

Тогда формально описать фигуру С можно так: С = cadb (с точностью до циклических перестановок терминалов). Грамматика ее будет примитивной (автоматной):

$(V_n = (S), V_t = (a,b,c,d), P = (S \to cadb | S \to dacb), S)$.

Грамматика для фигуры М будет сложнее, если допустить, что число квадратов в ней может быть не ограничено. Терминалы будут те же, а число нетерминалов можно увеличить и взять, например, такую грамматику:

V_n=(S,A), V_t=(a,b,c,d), P, S, где P:

$S \to c; | S \to SA; | A \to adb; | A \to acb$

Это уже бесконтекстная грамматика. Смысл нетерминала А очевиден - это "скобка" adb, добавлением которой к "стержню" с и получается фигура М.

Слова этого языка: cadb, cadbadb, cadbadbadb, ...

Благодаря рекурсивной подстановке S → SA словарь языка будет неограничен.

УПРАЖНЕНИЕ. Описать словарь языка с грамматикой

$$S \to 2S \quad S \to *S \quad S \to 1A$$

$$A \to 1B \quad A \to = B$$

$$B \to aC$$

$$C \to aB \quad C \to * \quad C \to 1$$

К какому типу (по Хомскому) относится данная грамматика?

Будет ли слово a=2*1 принадлежать данному языку? Какие слова порождает данная грамматика?

Это грамматика автоматного типа, фразы, начинающиеся с а, не могут быть построены ей. Показать это с помощью графа автомата.

ЗАДАЧА. Надо с помощью компьютера заменить в русскоязычном тексте слова "автомашина" на "автомобиль", а "автомат" на "пулемет". Алгоритм решения задачи должен использовать формальную грамматику для подстановок. Описать эту грамматику. Какого она типа?

Решение. Алфавит - все типографские символы, включая пробел @. Грамматика имеет вид:

$S \to AB$; $S \to C$; $A \to$ авто ; $B \to$ машина ; $C \to x$автоматy; авто$B \to$ автомобиль; @автомат@ → @пулемет@.

x,y - множество префиксов и суффиксов к слову "автомат".

Грамматика контекстная. Корневой нетерминал S обозначает часть слов, подлежащих анализу с целью замен слов. Контекстом в 6-м правиле будет терминал "авто" слева, в 7-м - пробел @ с обеих сторон. Контекст в последнем случае нужен, т.к. иначе программа будет подставлять "пулемет" в такие слова, как автоматический, полуавтомат и т.д.

ЗАДАЧА. Описать формально подстановки, использованные в следующих цепочках слов:

(Друг -> Враг): друг круг круп крап краб граб грач врач враг

(Вода -> Вино): вода кода кора кара карт кант бант бинт винт вино

(Муха -> Слон): муха маха мама лама лапа папа пара кара каре кафе кафр каюр каюк крюк урюк урок срок сток стон слон

СПИСОК ЛИТЕРАТУРЫ

1. Бильгаева Н.Ц. Теория алгоритмов, формальных языков, грамматик и автоматов. Учебное пособие. Улан-Удэ: Изд-во ВСГТУ, 2000.
2. Лупал А.М. Теория автоматов. Учебное пособие. СПб.:СПбГУАП, 2000.
3. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.:Наука. 1998.
4. Гронда О.В., Антик М.И., Иваненко Н.С. Прикладная теория цифровых автоматов. Методические указания по выполнению лабораторных работ. М.: МИРЭА, 1991.
5. Потемкин И.С. Функциональные узлы цифровой автоматики. М.: Энергоатомиздат, 1988.
6. Е.П.Угрюмов. Цифровая схемотехника. 2004.
7. Дж.Ту, Р.Гонсалес. Принципы распознавания образов. М.:Мир, 1978.
8. Хопкрофт Дж. Э., Мотвани Р., Ульман Дж. Д. Введение в теорию автоматов, языков и вычислений, 2-е изд. М.: Вильямс; 2002.

Огромное количество материала по теории автоматов и смежным дисциплинам можно найти в Интернете. Для поиска следует выйти на сайт

http://www.google.ru

набрать слова для поиска либо "Теория автоматов" (для поиска русскоязычных источников), либо "Automata theory" (для англоязычных), и запустить процедуру поиска: нажать на клавишу "Enter" либо сделать click мышью на клавише "Поиск в Google".

СОДЕРЖАНИЕ

Учебное издание

Злоказов Виктор Борисович

Теория автоматов

УНЦ-2011-47

Редактор *Е. В. Калинникова*

Получено 14.03.2011. Подписано в печать 10.08.2011.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 4,31. Уч.-изд. л. 6,14. Тираж 130 экз.
Заказ № 57343.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.
E-mail: publish@jinr.ru
www.jinr.ru/publish/