Технически университет - Варна

А. Антонов, Вл. Николов, И. Пенев

ДИСКРЕТНИ СТРУКТУРИ

Учебно пособие

Варна

2015

УДК 519.21: 519.87 (076)

Настоящото пособие е предназначено за студентите от специалностите "Компютърни системи и технологии" и „Софтуерни и интернет технологии“ в Технически университет - Варна. Съдържанието на пособието е съобразено с учебната програма по дисциплината "Дискретни структури".

© Анатолий Стефанов Антонов
Владимир Николов Николов
Ивайло Пламенов Пенев

ISBN ……..

ПРЕДГОВОР

Учебното пособие по Дискретни структури е предназначено за студентите от специалностите "Компютърни системи и технологии" и „Софтуерни и Интернет технологии“ в Технически Университет-Варна. Пособието е съобразено с приетата учебна програма по едноименната дисциплина "Дискретни структури", включена като задължителна в държавните изисквания за специалността. Изложеният материал може да бъде използван за подготовка по дисциплината и от студенти по същата специалност в други технически университети или от студенти, обучавани в сродни специалности.

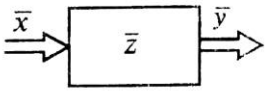
Учебният материал има теоретично-приложен характер и разглежда средства за моделиране на поведението на дискретни системи. Включените в пособието теоретични постановки и практически примери и задачи дават на студентите подходяща основа за представяне и концептуализация на дискретни обекти особено в областта на програмното осигуряване. Следните по-важни теми формират съдържанието на пособието:

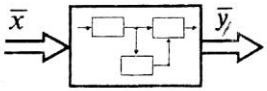
- Множества, отношения и функции върху множества. Регулярни множества и регулярни изрази.
- Формални граматики и езици. Синтактични графи.
- Абстрактни автомати. Машина на Тюринг. Крайни и стекови автомати. Базови и частично рекурсивни функции.
- Паралелни системи и процеси. Мрежи на Петри. Мрежови езици. Координиране и синхронизиране на паралелни процеси.
- Абстрактни информационни структури. Представяне на модели в оперативната памет чрез структури от данни.
- Математическа логика. Съждителна логика. Предикатна логика. Машина за логически извод. Резолюция и инификация.

Основни понятия

Система и структура

Системите могат да се представят чрез 3 вектора от променливи: $\bar{x}, \bar{y}, \bar{z}$ и множество от функции,





където

$\bar{x}$ - входен вектор;

$\bar{y}$ - изходен вектор;

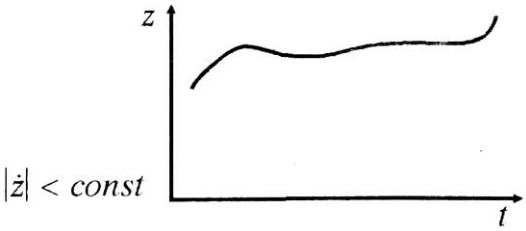
$\bar{z}$ - вътрешен вектор.

Подмножества от променливи на вектора $\bar{z}$ заедно с асоциирани към тях функции могат да бъдат разглеждани съответно на прието ниво на абстракция като елементи (обекти) на системата.

Структурата на системата се състои от множеството на елементите и връзките между тях и определя до голяма степен механизма на нейното действие.

В зависимост от типа на променливите структурите биват:

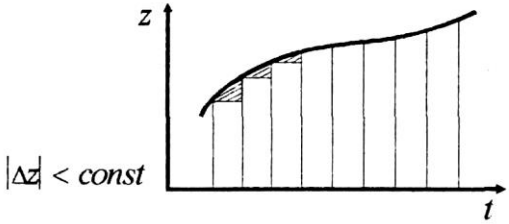
1. Непрекъсната структура. Дефиниционната област на променливите е непрекъсната. Такива структури могат да се опишат чрез система диференциални уравнения и алгебрични уравнения. Тези системи са инертни, като променливите не могат да изменят скокообразно своите стойности.



Фиг. 1. Непрекъсната структура

2. Дискретизирана структура. Дефиниционната област на променливите е непрекъсната, но техните стойности се разглеждат

само в определени моменти по оста на времето (дискретизация по време). Такива структури могат да се опишат чрез система от диференчни уравнения. Такъв вид системи са също инертни.

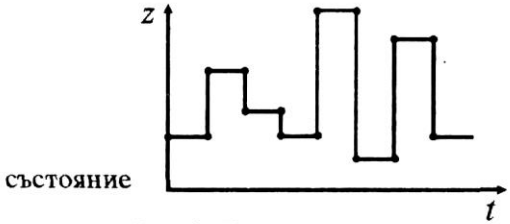


Фиг. 2. Дискретизирана структура

3. Дискретна структура. Дискретната структура е безинертна, от всяка стойност на z може да се премине в следващия момент към всяка друга стойност от дефиниционната област. Системата е дискретизирана по стойност и по време, като линейна (числова) подредба на стойностите на променливите не е необходима.

Стойността на z в даден момент от време се нарича състояние на дискретната система.

Дискретните структури се описват чрез автомати, граматики, предикати, графи и др.



Фиг. 3. Дискретна структура

Формално средство за описание - средство, език, таблица, метод и др., чиито изразни средства са формализирани, т.е. те се подчиняват на строга система от правила. Ако дадена структура се опише чрез формално средство за описание, тогава лесно (механично, без творчески усилия) може да се премине към програмно апаратно или смесено решение.

Следващите глави представят различни формални средства за описание поведението на дискретни структури, използвани предимно при създаването на програмно осигуряване

1. Множества. Отношения и функции в множествата.

1.1. Теория на множествата.

Множество - съвкупност от подредени елементи (обекти) a, b, c, ..., z. Ако елементите се повторят множеството се нарича мултимножество. Всяко множество A се определя от своите елменти и се задава чрез:

1. Изброяване на елементите на множеството:

$A = \{a, b, c, d\}$;

Пример:

плодове = { ябълка, круша, слива };

2. Правила за образуване на множеството, когато елементите се характеризират с някакъв общ белег (признак). Правилата за образуване имат тълкованието - "множеството съдържа всички елементи, удовлетворяващи правилата за образуването му":

A = { общ елемент | правила за образуване);

Примери:

плодове = { x | x = плод }

четни числа = { x | (x mod 2) = 0 };

$A = B = \{ x \mid x \in A, x \in B \}$

Обозначения:

, логически съюз "и"

; логически съюз "или"

$|A|$ мощност на множеството A (брой на елементите му)

Пример:

$|$ плодове $|$ = 3

$A \supset B$ B е подмножество на A

$A \supseteq B$ $B = A$ или B е подмножество на A

$\wedge$ празен елемент

$\{\ \}$ празно множество

$\{\wedge\}$ непразно множество, което съдържа празния елемент

1.2. Операции над множества.

1.2.1. Обединение (дизюнкция)

$$A \cup B = B \cup A = \{x \mid x \in A; x \in B\}$$

$$A \subset B \cup A \qquad B \subset B \cup A$$

1.2.2. Пресичане (конюнкция)

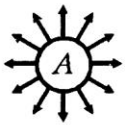
$$A \cap B = B \cap A = \{x \mid x \in A, x \in B\}$$

1.2.3. Разлика

$$A - B = \{x \mid x \in A, x \notin B\}$$

$$A - B \subset A$$

1.2.4. Отрицание (допълване)



$$\overline{A} = \neg A = \{x \mid x \notin A\}$$

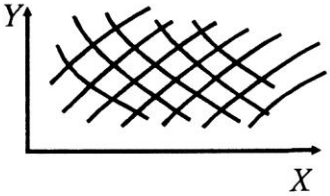
1.2.5. Декартово произведение на множества

$$A_1 \times A_2 \ldots A_n =$$

$$= \{< a_1, a_2, a_3, \ldots, a_n > \mid a_1 \in A_1, a_2 \in A_2, \ldots a_n \in A_n\}$$

Пример за декартово произведение е декартовата координатна система. За множествата X и Y:

$X = \{x \mid x$ натурално число$\}$
$Y = \{y \mid y$ натурално число$\}$



Фиг.1.1. Декартово произведение

Всяка точка от декартовата координатна система представлява наредена двойка от елементи $<x_i, y_i>$, която е елемент от декар-

товото произведение $X \times Y$:

$$X \times Y = \{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle \ldots\} = \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \ldots, \langle n,n\rangle\}$$

1.3. Свойства на операциите над множества.

1.3.1. Комутативност и асоциативност.

Операциите $\bigcup$ и $\bigcap$ са комутативни и асоциативни.

$A \bigcup B = B \bigcup A \quad A \bigcap B = B \bigcap A$ - комутативност

$A \bigcup (B \bigcup C) = (A \bigcup B) \bigcup C \quad A \bigcap (B \bigcap C) = (A \bigcap B) \bigcap C$

1.3.2. Некомутативни операции

Операциите разлика и декартово произведение са некомутативни и неасоциативни.

$A - B \neq B - A \quad A \times B \neq B \times A$

$A - (B - C) \neq (A - B) - C$

$A \times (B \times C) \neq (A \times B) \times C$

Примери:

Нека са дадени множества A, B и C:

A = {жени, мъже},

B = {красиви, грозни},

C = {студенти, работници}

образуват се декартови произведения

A x (B x C) = {<жени, <красиви, студенти>>,....}

(A x B) x C = {<жени, красиви>, студенти>,...}

Елементите на декартовото произведение A x (B x C) се състоят от наредени двойки, съставени от елемент от A и наредена двойка от множеството B x C.

Елементите на декартовото произведение (A x B) x C се състоят от наредени двойки, съставени от наредена двойка от множеството A x B и елемент от C.

Ще покажем, че първите елементи не са равни.

Елементът от първото декартово произведение определя жените като красиви студентки, а елементът от второто - красивите жени като студентки.

жени $\Leftrightarrow$ красиви студенти - има различен смисъл от

красиви жени $\Leftrightarrow$ студенти.

1.4. Отношения в множествата.

1.4.1. Отношение - множество, чиито елементи са подредени двойки:

$$r = \{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \ldots \langle x_n, y_n\rangle\}$$

$$r = \{\langle x, y\rangle \mid x \in D, y \in R\}$$

$D(r) = \{x \mid x \in D\}$ $\quad R(r) = \{y \mid y \in R\}$, където:

$D(r)$ - множество на аргументите

$R(r)$ - множество на стойностите

Аргументите и стойностите могат да се повтарят

$$|D(r)| \le |r| \qquad |R(r)| \le |r|.$$

Отношенията се изразяват с множествата D и R и самите те могат да се разглеждат като множества от елементарни отношения и да се прилагат операциите над множества: $\varphi \bigcup q$, $\varphi \bigcap q$

Декартовото произведение е също отношение, включващо всички отношения за тези множества.

Отношение, чиито елементи са подредени двойки се нарича двуместно отношение. В общия случай отношенията могат да бъдат триместни, четириместни, ...n - местни.

1.4.2. Класификация на отношенията.

Отношението r е рефлексивно, ако $x \in D(r) \rightarrow \langle x, x\rangle \in r$

Отношението r е симетрично, ако $\langle x, y\rangle \in r \rightarrow \langle y, x\rangle \in r$

Отношението е симетрично, ако $r^{-1} = r$

Отношението r е транзитивно, ако

$$\langle x, y\rangle \in r_T, \langle y, z\rangle \in r_T \Rightarrow \langle x, z\rangle \in r_T$$

Примери:

$r = \{\langle x, y\rangle \mid x = y\}$ - рефлексивно и симетрично

$r = \{\langle x, y\rangle \mid y < x\}$ - транзитивно, т.е.

$$r_1 = \{\langle x, y\rangle \mid y < x\}$$
$$r_2 = \{\langle y, z\rangle \mid z < y\}$$
$$r_3 = \{\langle x, z\rangle \mid z < x\}$$

r^{-1} обратно - отношение с разменени елементи

$$r^{-1} = \{\langle y, x\rangle \mid \langle x, y\rangle \in r\}$$

композиция $(r_1.r_2)$ $\langle x, y\rangle \in r_1.r_2 \Rightarrow \exists z \langle x, y\rangle \in r_1$

1.5. Функции в множествата.

Функция - отношение от специален вид, при което за всяко $x \in D(f)$ (дефиниционна област на аргументите), съществува единствен елемент y, така че $\langle x, y\rangle \in f$.

Функцията е еднозначно отношение $y = f(x)$

Функцията се задава чрез:
- изброяване;
- задаване множеството на аргументите;
- правила за изчисляване на стойностите на функциите

Пример:

$$f = \left\{\langle x, y\rangle \middle| \underbrace{x \in \{0,1\}}_{D(f)}, y = x^2 + 3x + 2\right\}$$

$$f = \{\langle 0,2\rangle, \langle 1,6\rangle\}$$

$$f = \{\langle x, y\rangle \mid x(x-1) = 0, y - (x^2 + 3x + 2) = 0\}$$

Функциите са клас отношения и към тях могат да се прилагат операциите над множества, които не нарушават тяхната еднозначност.

Частни случаи:

Тъждествена функция

$$f = \{\langle x, y\rangle | x = y, x \in D(f)\} \qquad f(x) = x$$

Обратна функция

$$f^{-1}(f(x)) = f(f^{-1}(x))$$

Константна функция

$$f = \{\langle x, y\rangle | x \in D(f), y = \{y_0\}\} \quad f(x) = const = y_0$$

Вложена функция

$$f = \{\langle x, y\rangle | y = h(g(x))\}$$

Предикатна функция

$$f = \{\langle x, y\rangle | x \in D(f), y = \{TRUE, FALSE\}\},$$

където TRUE (T) - истина, FALSE (F) - лъжа

Чрез предикатни функции може да се дефинират други функции:

Примери:

$$f = \{\langle x, T\rangle | x \geq 0\} \cup \{\langle x, F\rangle | x < 0\}$$

$$f = \{\langle x, y\rangle | x = \{-1, 2, 3\}, (x \geq 0 \rightarrow y = 3; x < 0 \rightarrow y = 4)\}$$

$$f = \{\langle -1,4\rangle, \langle 2,3,\rangle, \langle 3,3\rangle\}$$

Рекурсивна функция

$$f = \{\langle x, y\rangle | x \in D(f); y = f(x)\}$$

$$f = \{\langle x, y\rangle | x \geq 1, y = f(x-1).x\}$$

Дефиницията на рекурсивните функции включва самите функции, например x! се описва чрез (x-1)!.

2. Подредени множества и низове. Операции с низове. Регулярни множества и изрази.

2.1. Подредени множества.

Множества, чиито елементи са подредени в ред (низ или дума). Съществуват понятия като първи, последен, предходен, следващ елемент и т.н.

Азбука - произволно крайно множество от символи.

Обозначение: $\Sigma = \{ a, б, в, ..., я \}$

Низ - (или дума) над Σ се нарича всяка крайна редица от букви от азбуката.

Обозначения:

$\Lambda(\varepsilon)$ - дума, която няма букви (празна дума)

a, b, c - символи, букви от Σ

t, u, v - символни низове, думи

A, B, C - множества от низове

$|\ |$ - дължина на низ (дума)

Σ^* - множество от всички възможни низове включително и празния низ Λ

Σ^+ - множество от всички възможни низове без празния низ Λ

Примери:

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

$$\Sigma^* = \{\wedge, a, b, ab, ba, aa, bb, \ldots\}$$

$$\Sigma^+ = \{a, b, ab, ba, aa, bb, \ldots\}$$

$\{\ \} = \varnothing$ празно множество от низове

$\{\Lambda\}$ - множество, съдържащо празен низ като елемент. Това множество не е празно множество.

2.2. Операции с низове.

1. Съединяване (конкатенация) на низове и множество от низове:

Нека са дадени низове u и w, които принадлежат на множествата U и W съответно:

$u \in U,\ w \in W$

Множеството UW, получено чрез операция съединяване на множествата U и W съдържа елементи x.

$$UW = \{x | x = uw, u \in U, w \in W\}$$

Всяка дума от UW се получава от съединяване на дума от U с дума от W.

Пример:

Дадени са множествата:

$U = \{a, ab, aab\} \quad W = \{b, bb\}$

Резултатът от конкатенацията на UW е:

$UW = \{ab, abb, aabb, abb, abbb, aabbb\}$

Резултатът от конкатенацията на WU е:

$WU = \{ba, bba, bab, bbab, \ldots\}$

$UW \neq WU$ - операцията конкатенация е некомутативна, но е асоциативна:

$u(vw) = (uv)w$ - където, $u, w, v \in \Sigma^*$,

2. Обединяване на низове, обединяване на множества от низове.

Пример:

$U = \{a, ab, aab\} \quad W = \{b, bb\}$

$U \cup W = \{a, ab, aab, b, bb\}$

3. Затворена обвивка S* (действие звезда).

Резултатът от операцията представлява множество, състоящо се от празната дума и всички думи, образувани чрез съединяване на краен брой думи от S.

$\Sigma^* \leftrightarrow S^*$ - съединяване на букви (низове) от множеството S.

$S^* = S^o \cup S^1 \cup S^2$ като $S^o = \{\Lambda\}$

$S^i = S^{i-1}S$ за всяко $i > 0$

Пример:

$$S = \{ab, bb\} \rightarrow S^* = \{\Lambda, ab, bb, abab, abbb, bab, \ldots\}$$

2.3. Регулярни множества.

Регулярни множества са клас множества от думи на Σ. Те се получават чрез краен брой прилагания на следните правила.

1. Всяко крайно множество на думи над Σ включително и празното е регулярно.

2. Ако u и v са регулярни множества, то тяхното обединение $u + v$ или конкатенация uv са също регулярни множества.

3. Ако S е регулярно множество над Σ, то и S* е също регулярно.

Пример: Множество от думи над Σ, което съдържа думи, само с aa и/или bb е регулярно.

$$T = \{aa, aab, abb, abba, baab, \ldots\}$$

$T = \{a^n b^n \mid n > 0\}$ съдържа n "a" и след това n "b". Това множество не е регулярно, тъй като не е крайно.

2.4. Регулярни изрази (РИ).

Регулярните изрази са средства за представяне или запис на регулярни множества, което дава възможност за алгебрично изчисление с регулярни множества.

Регулярни изрази над азбуката се дефинират рекурсивно по

следния начин:

1. Празната дума и празното множество от думи са регулярни изрази.

2. Всяка буква (σ)от Σ е регулярен израз, $\sigma \in \Sigma$

3. Ако P и Q са регулярни изрази, то $(P+Q),(PQ),(P)*$ са регулярни изрази.

<u>Приоритет на операциите:</u>

(U) - изрази в скоби имат най-висок приоритет

*U** - затворена обвивка

UV - конкатенация

U+V - обединение

<u>Примери:</u> Дадена е азбука

$\Sigma = \{a,b\} \to ((a+ba))^*$ - регулярен израз върху азбуката

*ab** - регулярен израз, описващ множеството от всички думи над Σ, започващи с *a*

*a*ba*ba** - регулярен израз, описващ множеството от всички думи над Σ, включващи 2 *b*

2.5. Изчисления с регулярни изрази.

Формули за еквивалентност на регулярни изрази

Нека *R, S, T* са произволни множества над Σ.

Всеки регулярен израз R описва множество $\tilde{R}$ от думи над Σ, т.е. $\tilde{R} \subseteq \Sigma^*$

$R = \Lambda \to \tilde{R} = \{\Lambda\}$ - множество, състоящо се от празна дума.

$R = \varnothing \to \tilde{R} = \{\varnothing\}$ - празно множество, не съдържащо нито една дума.

$R = \sigma \to \tilde{R} = \{\sigma\}$ - множество, състоящо се от буквата σ.

Нека R_1 и R_2 са регулярни изрази, описващи $\tilde{R}_1$ и $\tilde{R}_2$:

$\tilde{R} = \tilde{R}_1.\tilde{R}_2 = \{r_1 r_2 | r_1 \in \tilde{R}_1, r_2 \in \tilde{R}_2\}$ - конкатенация

$R = R_1 + R_2 \rightarrow \tilde{R} = \{x | x \in \tilde{R}_1, x \in \tilde{R}_2\}$ - обединение

$R = (\tilde{R}_1)^* \rightarrow$ множеството, образувано чрез операция звезда върху елемента на множеството $\tilde{R}_1$.

За произволни регулярни изрази R, S и T са валидни следните формули за еквивалентност:

1. $R + S = S + R; \quad R + \varnothing = \varnothing + R; \quad R + R = R;$

$$(R + S) + T = R + (S + T)$$

2. $R\Lambda = \Lambda R = R; R\varnothing = \varnothing R = \varnothing; (RS)T = R(ST)$

3. $R(S + T) = RS + RT \qquad (S + T)R = SR + TR$

4. $R^* = R^*R^* = (R)^* = (R^*)^* = (\Lambda + R)^*$

5. $R^* = \Lambda + R + R^2 + \ldots R^k + R^{k+1}R^*, \quad \forall (k \geq 0)$

$R^* = \Lambda + RR^*$

6. $(R + S)^* = (R^* + S^*)^* = (R^*S^*)^* = (R^*S)^*R^* = R^*(SR^*)^*$

$$(R + S)^* \neq R^* + S^*$$

7. $R^*R = RR^* \qquad R(SR)^* = (RS)^*R$

8. $(R^*S)^* = \Lambda + (R + S)^*S; \qquad (RS^*)^* = \Lambda + R(R + S)^*$

$\Lambda \neq S$ - правило на Ардън: Ако $\Lambda \notin S$

9. $R = S^*T \leftrightarrow R = SR + T$

$R = TS^* \leftrightarrow R = RS + T$

3. Формални граматики и езици.

3.1. Основни понятия.

<u>Формален език L</u> - формален език е множество от низове с крайна дължина на азбуката $\sum$. Езиците се бележат с L.

L е подмножество на Σ^* $(L \subseteq \Sigma^*)$

Пример: $\Sigma = \{a, b, c\}$

$L_1 = \left\{ \underbrace{\Lambda, a, ab, aab, bc, abc}_{6\,думи} \right\}$ - краен език (броят на елементите е краен)

$L_2 = \{a^n \mid n \geq 0\}$ - безкраен език, тъй като броят на елементите е неограничен.

<u>Понятия:</u> Нека съществуват два езика L_1 и L_2.

Ако $L_1 \subseteq L_2$, то

1) L_1 е подезик на L_2 (езикова съвместимост - нагоре).

2) L_2 е надезик на L_1 (езикова съвместимост - надолу).

3) Ако думите в двата езика съвпадат, т. е. мнжествата от низове са равни: ($L_1 = L_2$), то съществува езикова еквивалентност.

<u>Описание на езика L.</u>

Езиците могат да се опишат по два начина:

1. Чрез изреждане - използва се при малка мощност на множеството $|L|$.

2. Чрез правила на образуване - за голяма мощност $|L|$ на множествата от думи или при безкрайни езици. За всички езици броят на правилата, по които се извършва описанието на елементите е краен.

Езиците се описват с помощта на пораждаща (генерираща, образуваща) граматика. Формалната граматика е формално средство за описание на езиците, а формален е езикът, породен от такава граматика.

3.2. Формална граматика *G*.

Това е множество, състоящо се от следните 4 множества

$G = \{\Sigma, N, P, S\}$, където

Σ - множество от букви или азбука на езика L, описван с граматика, наречени още терминални символи.

N - крайно, непразно множество от символи (променливи) наречени нетерминални символи.

$N \bigcap \Sigma = \varnothing$, т.е. N и Σ нямат общи символи

$U = N \bigcup \Sigma$ - обединеното множество от нетерминални и терминални символи се нарича речник на езика.

S - начална променлива (начален символ). От него започва пораждането на елементите на езика. S също е нетерминален символ т.е. $S \in N$ (S се включва в N).

P - крайно множество от правила от вида $\alpha \to \beta$ (правила за пораждане, продукции). Чрез тях се описва как се получават всички елементи (думи) на езика.

$\alpha \in (\bigcup)^+ = (N \bigcup \Sigma)^+ = (N \bigcup \Sigma^*) \setminus \{\Lambda\}$

$\beta \in (N \bigcup \Sigma^*)$, т.е. липсватпродукции от вида $\Lambda \to \beta$, $\alpha \to \{\Lambda\}$.

α е множество от всички думи над речника, образувани от елементите на речника без празния низ, β може да се състои от множеството от всички низове, образувани чрез азбуката.

Формалната граматика G генерира езика L. Това означава, че G генерира думи на езика L.

$S \to w_1 \to w_2 \to \ldots \to w_n \;/\; n \geq 1, S$ начален символ $w_n \in L$.

Една дума w се поражда (генерира) от граматиката G ако съществува веригата $S \to \ldots \to w_i \to w_{i+1} \to \ldots w$

S е начален символ, а w_{i+1} се получава от w_i като заместим някой подниз α във w_i с β, използувайки някое правило от P.

Пример:

$G = \{\Sigma, N, P, S\}\Sigma = \{a, b\}$

$N = \{A, B, S\}$ - нетерминални символи

P се състои в правилата:

1. $S \to aSBA$
2. $S \to abA$
3. $AB \to BA$
4. $bB \to bb$
5. $bA \to ba$
6. $aA \to aa$

Езикът L, породен от G е безкраен език $\{a^n b^n a^n | n \geq 1\}$

Получаването на произволен низ от езика L се осъществява чрез следната последователност.

1. Прилага се *n-1* пъти правило 1.

$a^{n-1}S(BA)^{n-1}$

2. Прилага се един път правило 2.

$a^n bA(BA)^{n-1}$

3. Прилага се *n-1* пъти правило 3.

$a^n bB^{n-1}A^n$

4. Прилага се *n-1* пъти правило 4.

$a^n b^n A^n$

5. Прилага се 1 път правило 5.

$a^n b^n aA^{n-1}$

6. Прилага се *n-1* пъти правило 6.

$a^n b^n a^n$

Няма друг език, породен от разглежданата граматика G.

Съществено условие за правилно функциониране на граматиката е осигурявана на възможност за произволно прилагане на правилата. В какъвто и ред да се прилагат правилата, трябва да се получава низ от езика на граматиката. В примерната граматика прилагането на правило 2 като начално, изисква прилагане на правило 5 и получаване на частния случай (aba) на езика L. Правилно съставена граматика не допуска получаване на низ, който не се съдържа в множеството L.

3.3. Класификация и йерархия на G (по Хомски)

При тази класификация граматиките се групират съгласно наложени ограничения към правилата. Разграничават се четири типа граматики, като първият тип, наречен още "естествен език", е дефиниран според определението за формална граматика. За всеки следващ тип се налагат допълнителни ограничения към дължината на низовете на лявата и дясната част и към символите, които могат да бъдат включени.

Таблица 3.1. Класификация на граматики

Тип на G	Наименование	Ограничения към $P(\alpha \to \beta)$
G_0	без ограничения	Няма (според определение дадено в т. 3.2)
G_1	Контектно зависима	$\lvert\alpha\rvert \le \lvert\beta\rvert$ дължина на лява, дясна част
G_2	Контекстно свободна	$\alpha \in N, \beta \in (N \cup \Sigma)^+$
G_3	Регулярна (линейна)	$A \to \sigma B;$ $A \to \sigma \mid A, B \in N, \sigma \in \Sigma$ σ е елемент от азбуката или $\sigma = \Lambda$

Най-висшият тип граматика включва в себе си низовете на по-ниските типове от йерархията на граматиките:

$$G_0 \supset G_1 \supset G_2 \supset G_3, \ (G_3 \text{ е най-маломощна})$$

Регулярна (линейна) е граматиката, която се състои от правила, в лявата част на които има само един нетерминален символ, а в дясната конкатенация на терминален с нетерминален символ, само терминален символ или празен символ.

Контексно свободна е граматиката, в правилата на която, който и нетерминален символ да вземем и където да го поставим, той означава едно и също $\alpha A \beta \to A = A$.

Контексно зависима - Всеки символ променя значението си в зависимост от това, къде е поставен (т.е. от контекста, в който се

среща). Под контекст на символ се разбира конкатенация на символ с други (терминални и/или нетерминални) символи. G_o съответствува на човешкия език.

3.4. Видове продукции.

В зависимост от структурата на рекурсивната продукция се разграничават следните видове продукции:

Двойно рекурсивна продукция:

$x \to \alpha x \beta$ (x се получава от себе си, като α и β са низове)

Ляво рекурсивна продукция

$x \to x\beta$ (x се получава чрез добавяне на низ отдясно)

Дясно рекурсивна продукция

$x \to \alpha x$ (x се получава чрез добавяне на низ отляво)

В зависимост от предназначението на продукциите те се делят на следните видове:

Лексика: правила от P за пораждане на думи.

Синтаксис: правила от P за пораждане на изрази.

Семантика: допълнителни правила, свързани със смисъла на породените думи или изрази.

3.5. Език на граматиките Бакус-Наур формализъм.

Бакус-Наур-Форма (БНФ) за описание на продукциите Р.

БНФ е вид граматика за описание на формалните граматики.

Означения:

Нека u и $v \in (\Sigma \cup N)$. За описание на продукциите се използват следните правила:

$\alpha \to \beta$ се записва като:	$\alpha ::= \beta$ или $\alpha \Rightarrow \beta$
конкатенация:	UV
алтернатива (изключв. или):	$U \mid V$
незадължителен елемент:	$[u]$
помощни скоби:	(u)
итерация: $u^n \mid n \geq 0$	u^* или $(u)^*$ или $\{u\}$

итерация без празното

множество: $u^n \mid n > 0$ u^+ или $(u)^+$ или uu^* или $u\{u\}$

Нетерминалните символи се изписват обикновено с техните смислови обозначения в съответсвие с граматиката. БНФ на граматиката е формата, в която езикът се представя за използване от потребителя, поради което наименованията на нетерминалните символи трябва да бъдат възможно най-ясни за него. Използват се както малки, така и главни букви. При използване на повече от една дума като нетерминален символ, думите се свързват със символ _ (например: избран_оператор). Терминалните символи се заграждат със символ ' (например: 'case').

<u>Пример:</u>

Число $\Rightarrow$ Цифра$^+$ Име $\Rightarrow$ буква(буква|цифра)*

Нетерминални символи са число и име, а терминални цифра и буква:

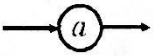
цифра $\Rightarrow$ '0'|'1'|'2'|'3'|...|'9'|

буква $\Rightarrow$ $'a'|'b'|...$

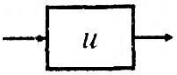
3.6. Синтактични графи - графично средство за представяне на правилата P.

Синтактичните графи служат за онагледяване на правилата на граматиката P. Всяко правило се представя чрез графично изображение, в което се записват символите на граматиката. Елементите се свързват със стрелки. Формата на изображението зависи от това, дали символът е терминален или нетерминален.

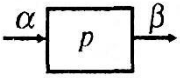
$a \in \Sigma$



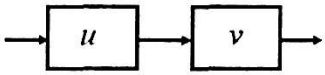
$u \in N$



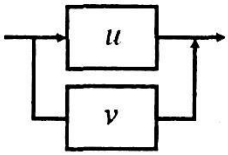
$\alpha \rightarrow \beta$



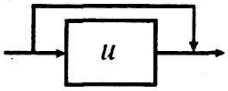
uv (конкатенация)



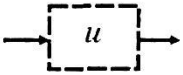
u | *v* (алтернатива)



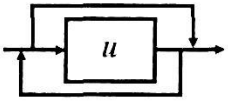
[u]



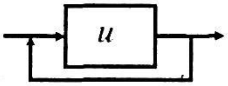
(u)



u^*

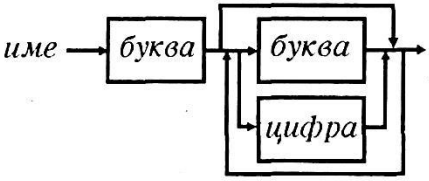


u^+



Всички елементи имат по един вход и по един изход.

Например, при правилото за описание на име на променлива се използва следният синтактичен граф:



Фиг.3.1. Синтактичен граф за име на променлива

Съответната БНФ на правилото е следната:

име $\Rightarrow$ буква (буква|цифра)*

<u>Правилото гласи:</u>

"Името се съставя от задължителна начална буква и след нея може да има и други букви или цифри".

Пример:
Оператор "case" селектор в Паскал

case Израз of
k_1, k_2,... k_e: оператор;
L_1, L_2,... L_m: оператор;
...
M_1, M_2,... M_n: оператор;
end;

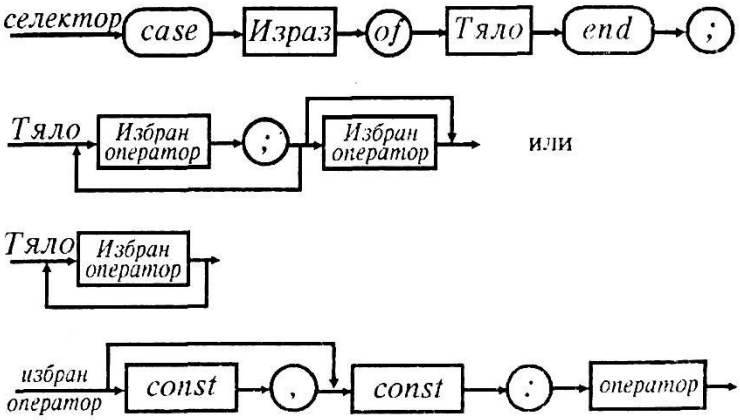
} Тяло

Съответната БНФ форма за представяне е следната:
Селектор $\Rightarrow$ 'case' Израз 'of' Тяло 'end' ';'
Тяло $\Rightarrow$ (избран_оператор)*избран_оператор $\equiv$ (избран_оператор)$^+$
избран_оператор $\Rightarrow$ (константа ',')* константа ':' оператор';'

В примерната граматика са описани правилата за нетерминалите Селектор, Тяло и избран_оператор. Нетерминалните символи израз, константа и оператор се считат за инвестни (или описани).

Ключовите думи на граматиката ('case', 'of', 'end', ';', ':', и ',') се заграждат в апострофи

Синтактичните графи на примерната граматика са следните:



Фиг. 3.2. Синтактични графи за оператор “case”

Анализ на един избран_оператор (ред):

123,	12	:	Delta	:=	17;	Лексически анализ
число	число	име	присвояване		точка и запетая	
запетая	двуеточие			число		
число	число		име	присвояване	число	Синтактичен анализ
			оператор			
		избран оператор				

При първото (лексическо) ниво на анализ се прилагат правилата за отделяне на думи, в резултат на което се разграничават нетерминалите: число, име, оператор за присвояване и т.н. Тези нетерминали са входни символи за второто (синтактично) ниво на анализ, където сработва правилото за избран_оператор. Интервалите в низова поредица се пропускат.

4. Абстрактни автомати.

4.1. Изчислимост и разрешимост.

4.1.1. Изчислимост.

Изчислимостта е свързана с изчисляване на функции и автоматизиране на процеса с помощта на компютър.

Функцията $f(x_1, x_2, \ldots, x_n)$ се смята за изчислима, ако съществува някаква "механична процедура" (предписание), която се следва и с помощта, на която може да се намери стойността на функцията при зададени аргументи.

$$f(x_1, x_2, \ldots, x_n) \xrightarrow[\text{+ аргументи } k_i]{\text{предписания}} f(k_1, k_2, \ldots, k_n)$$

Под "механична процедура" се разбира някакъв процес, при който не се прилага никаква изобретателност или творчество.

Пример:

Евклидов алгоритъм за намиране най-голям общ делител (НГОД) на две цели положителни числа.

Дадени са a и b - натурални числа (естествени, цели, положителни).

Да се намери $d = f(a,b)$, f е НГОД.

1. Сравняват се a и b (възможните случаи са: $a = b, a > b, a < b$)

2. Ако $a = b$, тогава $d = a$, край

3. Ако $a < b$, - разменят се местата на a и b: $a \leftarrow b, a \rightarrow b$

4. Ако $a > b$, тогава $a = b$, $b = a - b$ и следва преход към 1.

Таблично решение:

Таблица 4.1. Алгоритъм на Евклид за НГОД

a	b	Стъпка
9	21	1
21	9	3
9	12	4, 1
12	9	3
9	3	4, 1, 3
3	6	4
6	3	3
3	3	4, 1, 2 $\rightarrow$ край
		$d = 3$

4.2.1. Задачи, свързани с изчислимост на функции.

1. При дадена функция да се определи дали тя е изчислима въобще, (функциите могат да бъдат изчислими и неизчислими). Пример за неизчислима фунция е намирането на корените на уравнението от пета степен.

2. Ако фунцията е изчислима да се определи описание или алгоритъм за изчисление.

3. При даден алгоритъм да се определи с какъв вид изчислително устройство може да се извърши изчислението (краен автомат, стеков автомат).

4. При даден алгоритъм да се определи с какво

формално средство да се опише процесът на изчисление на функцията.

5. При дадено формално средство за описание да се определи класът фунции, които могат да бъдат описани (т.е. каква е описателната мощ на средството за описание).

6. При дадено изчислително устройство да се определи класът функции, които могат да бъдат изчислени (т.е. каква е изчислителната мощ на изчислителното устройство).

4.1.2. Разрешимост

Разрешимостта е свързана с изчислението на отношения. Изчислимостта на отношение се нарича разрешимост. Ако отношенията за разрешими, то след изчислението се получава логическа стойност (да/не).

Пример: $b \in B$? Ако това отношение е разрешимо, може да се определи дали отговорът е да или не.

Разрешимостта всъщност е изчислимост за предикатни функции.

4.2. Понятие за алгоритъм.

Под алгоритъм се разбира предписание (механична процедура) от елементарни операции, чисто последователно изпълнение съгласно предписанието води до изчислението на дадена функция или до разрешение на дадено отношение.

Характеристики на алгоритмите:

1. Оперира се с конкретни подлежащи на въздействие обекти. Обектите могат да се изброят и номерират така, че да се работи само с номерата им.

2. Алгоритъмът се задава чрез следната петорка.

$A = \{P, I, W, O, n\}$,

където

P - крайно множество от предписания;

I - входна азбука;

W - работна азбука;

O - изходна азбука;

n - размерност на обектите, с които се работи (мощност на

номерата).

3. *P* гарантира поетапно изпълнение на операциите и преход от едно междинно състояние на изчислението към друго.

4. *P* гарантира еднозначност.

Операциите се изпълняват еднозначно, без свободни интерпретации, тъй като целта е изчисление с помощна на машина без съдействие на творчеството на човек.

5. *P* гарантира възпроизводимост. Възпроизводимостта означава повторяемост на резултатите при едни и същи условия.

6. *P* е пълно предписание, т.е. цялата нужна за изчислението информация се съдържа в *P* и *I* (*P* + *I* е достатъчна за извършване на изчислението).

7. Икономически условия - поставят се пред алгоритмите. Това означава, че същите боравят с крайни елементи (азбука, думи, P, изчислява памет и др). Същите трябва да завършват действието си за крайно време. Например, безсмислен е алгоритъм, който завършва след дни или години.

Функцията $f(x_1, x_2, \ldots, x_n)$ е изчислима, ако съществува алгоритъм А, при който за всяко x_i от дефинираната област се изчислява стойността на *f*.

Наложила се е необходимост от разработване на универсален изчислител на базата на който да се определя изчислимост или разрешимост и оттам да се решат б-те задачи, свързани с изчислимост на функции (4.2.1.)

4.3. Машина на Тюринг.

Машината на Тюринг (МТ) не служи за изчисления, а да определя принципна изчислимост или разрешимост.

След създаване на МТ Чърч изказва тезиса: Всяка изчислима функция и всяко разрешимо отношение е изчислимо (разрешимо) чрез МТ. (Ако дадена функция може да се изчисли чрез МТ, то тя е изчислима.)

4.3.1. Краен автомат (КА)

Крайният автомат е най-простият вид абстрактен автомат. Той е една от частите на МТ. КА се описва чрез краен ориентиран

граф.

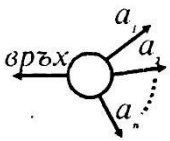
4.3.1.1. Краен ориентиран граф(КОГ) - множество, състоящо се от върхове и дъги $\{V, D\}$.

$V = \{V_i \mid i \geq 1\}$; V_i - крайно множество от върхове (състояния) на автомата

$D = \{<V_i, V_j> \mid j, i \geq 1\}$ - крайно множество подредени двойки, съставени от върхове (състояния).

Определяне на КА чрез КОГ.

КА A над дадена азбука $\sum = \{a_1, a_2, ..., a_n\}$ се нарича КОГ, от всеки връх на който излизат n насочени дъги и всяка е обозначена с етикет.



Един от върховете е начален се бележи с $\ominus$

$\ominus$ начално състояние (начален връх)

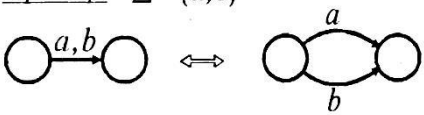
$\oplus$ множество от крайни състояния, което може да бъде и празно

Път от връх i до връх j се нарича дума, образувана чрез конкатенация от етикети $a_\kappa (\kappa = i \div j)$. Конкатенацията от етикети се бележи с w (w е дума, т.е. $w \in \sum^*$).

Думата w се разпознава от автомата А, ако тя се образува чрез път от начално до крайно състояние.

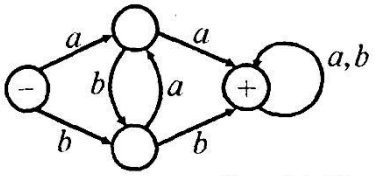
Всички думи, които се разпознават от автомата, образуват множество, което е крайно. Автоматите разпознават регулярни множества.

Пример: $\sum = \{a, b\}$



този автомат разпознава $\varnothing$ (празното множество)

разпознава $\sum^* | \sum^* = \{\Lambda, a, b, \ldots\}$



От всеки връх трябва да излизат 2 дъги.

Фиг. 4.1. Пример за КА

Думите, които разпознава автоматът са:

$$\left.\begin{array}{l} aa\sum^* \\ bb\sum^* \\ abb\sum^* \\ baa\sum^* \end{array}\right\}$$

Този автомат разпознава входни думи, които имат минимум 2 последователни a или b.

4.3.1.2. Диаграма на преходите.

КОГ, на който всяка насочена дъга е отбелязана с дума от азбуката $\sum$, т.е. $w \in \sum^*$ (евентуално $\wedge$). Тази дума се нарича етикет на дъгата. Има поне един връх, отбелязан със знак - , наречен начален връх и (евентуално празно) множество от върхове, отбелязани със знак + (крайни върхове).

w - път в диаграма на преходите от връх V_i до V_j се нарича такъв краен път от V_i до V_j, конкатенацията от етикетите на насочените дъги по който представлява думата w. Ако има празна дума във пътя на w, тя се пренебрегва.

Една дума $w \in \sum^*$ се разпознава от диаграма на преходите, ако съществува w - път от начален до краен връх. В частност празната дума $w = \Lambda$ се разпознава от диаграма на преходите, ако съществува Λ - път от начален до краен връх или съществува връх, който е едновременно начален и краен.

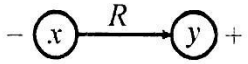
4.3.1.3. Обобщена диаграма на преходите.

Такава диаграма на преходите, насочените дъги на която могат да бъдат обозначени с регулярни изрази.

4.3.1.4. Теорема на Клини: Едно множество е регулярно тогава, когато се разпознава от диаграмата на преходите.

4.3.1.5. Метод на подмножествата за синтез на краен автомат по зададен регулярен израз:

а) Започва се от обобщената диаграма на преходите, съставена от начален връх x и краен връх y, с дъга между тях, обозначена със зададения регулярен израз R.



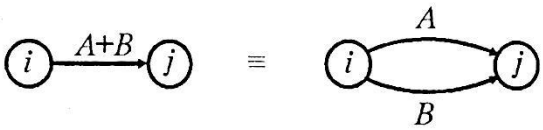
б) Последователно се разклонява регулярният израз R като се прибавят нови върхове и дъги. Процесът продължава, докато всяка дъга бъде обозначена с буква от азбуката Σ или Λ.

Използват се следните правила:

1. Ако между два върха i и j съществува регулярен израз, съставен от конкатенацията на два регулярни израза A и B, се въвежда нов връх k между i и j и насочена дъга от i към k, отбелязана с етикета A и дъга от k към j, отбелязана с етикет B.

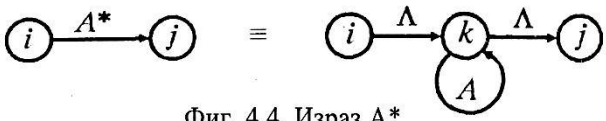
Фиг. 4.2. Преобразуване на израз AB

2. Ако между два върха i и j съществува регулярен израз, описващ обединението на регулярни изрази A и B, между върховете i и j се добавят две дъги от i към j, с обозначения съответно A и B.



Фиг. 4.3. Преобразуване на израз A+B

3.) Ако между два върха има регулярен израз от вида A*. Добавя се нов връх k, между i и j, насочени дъги от i към k и от k към j, обозначени с $\wedge$ и дъга от k към k, обозначена с регулярния израз A.



Фиг. 4.4. Израз A*

В получената диаграма на преходите върхът x е все още единствен начален, а y - единствен краен връх.

в) Нека T е диаграма на преходите, получена след стъпка б). Таблица на преходите за получената диаграма T се построява по следния начин:

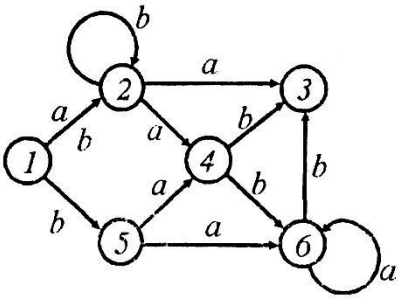
Нека за простота да приемем, че азбуката е $\Sigma=\{a,b\}$.

Нека M е произволно подмножество от върхове на T.

За всяка $w \in \Sigma^*$ се дефинира множеството M_w като подмножество от всички върхове на диаграмата T до които има w - път от някакъв връх на подмножеството M.

<u>Например:</u> M_{ab} се състои от всички върхове T, до които има ab - път от някакъв връх, принадлежащ на множеството M или все едно всички върхове на T, до които има b - път от някакъв връх на M_a.

<u>Пример:</u>



Фиг. 4.5.Краен ориентиран граф

1) $M = \{1\}$ $M_{ab} = \{2\}$ $M_a = \{2\}$
2) $M = \{1, 5\}$ $M_{ab} = \{2, 3,6\}$ $M_b = \{5\}$
3) $M = \{1, 4, 6\}$ $M_{ab} = \{2, 3\}$ $M_a = \{2, 6\}$

Таблицата на преходите се състои от три стълба. Елементите на таблицата са подмножества от върхове на Т (евентуално празно множество). Подмножеството в горния ляв ъгъл на таблицата (1 ред 1 стълб) е $\{x\}_\Lambda$, т.е. подмножество, състоящо се от началния връх x и всички върхове на Т, до които има Λ път от x. За всеки ред от таблицата се извършва следното:

Ако множеството M е в стълб 1, образуваме множеството M_a като подмножество от всички върхове, до които има a - път от някакъв връх на M и го записваме във 2^ри^ стълб. По аналогичен начин образуваме множество M_b като подмножество от всички върхове на Т, до които има b - път от някакъв връх M в стълб 3. Ако M_a не се среща преди това в 1^ви^ стълб, прибавя се на следващия ред и процесът се повтаря.

	M	M_a	M_b
	$\{x\}_\Lambda$	-	-
	-		

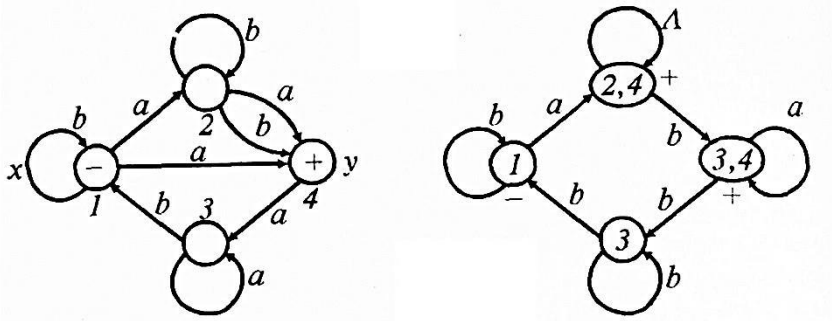
Крайният автомат се построява по получената таблица. Той се състои от толкова върха, колкото са редовете в таблицата. Начален връх е върхът, съответствуващ на ред 1, колона 1 на таблицата. Насочените дъги на КА съответствуват на наредените двойки $<M, M_a>$ за дъга с етикет а и $<M, M_b>$ за дъга с етикет b.

Крайни върхове на КА са тези върхове, чиито множества в таблицата съдържат краен връх на диаграмата, от която е построена таблицата.

<u>Пример:</u> Дадена е таблица Т.

Фиг. 4.2. Таблица на преходите

M	Ma	Mb
{1}	{2,4}	{1}
{2,4}	{2,4}	{3,4}
{3,4}	{3,4}	{3}
{1,3}	{2,3,4}	{1}
{2,3,4}	{2,3,4}	{1,3,4}
{1,3,4}	{2,3,4}	{1,3}



Фиг. 4.6. Крайни автомати

4.3.1.6. Теорема на Клини: За всяка диаграма на преходите T над азбуката Σ съществува регулярен израз R над Σ, за който $\tilde{R} = \tilde{T}$.

Следствие:

За всеки регулярен израз R над Σ, съществува краен автомат A над Σ, за който $\tilde{A} = \tilde{R}$.

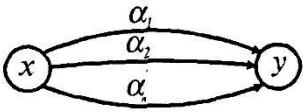
4.3.1.7. Метод за построяване на регулярен израз R по дадена диаграма на преходите T, така че $\tilde{R} = \tilde{T}$.

а) Към зададената T се прибавят нов връх x и дъги с етикет Λ, водещи от върха x към всички начални върхове на диаграмата (т.е. до началния връх и всички свързани с Λ дъги към него върхове на T). Към T се прибавя и нов връх y и дъги с етикети Λ водещи от всички крайни върхове на T към върха y. По този начин се образува модифицирана диаграма на преходите, в която има само един начален връх и един краен връх.

б) Елиминират се стъпка по стъпка всички върхове на T, докато останат само новите върхове x и y. По време на този процес дъгите и техните етикети се променят. Всяка дъга може да бъде обозначена с регулярен израз, т.е. по този начин се построява обобщена диаграма на преходите.

Получената диаграма на преходите след всяка стъпка разпознава същото множество думи, както и първоначалната диаг-

рамаТ. Процесът продължава до получаване на обобщена диаграма на преходите, в която съществуват само върховете x и y, от вида:



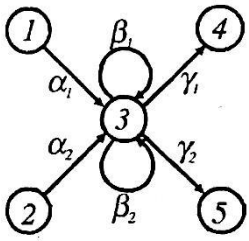
Фиг. 4.7. Обобщена диаграма на преходите

Тогава търсеният регулярен израз е:

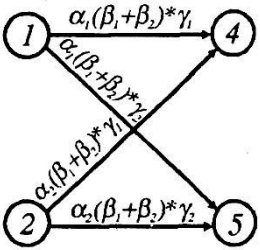
$$R = \alpha_1 + \alpha_2 + \ldots + \alpha_n$$

<u>Правила за елиминиране на върхове:</u>

Нека е необходимо да се елиминира върхът 3 от следната диаграма на преходите:



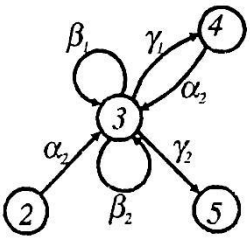
Фиг. 4.8. Примерна диаграма



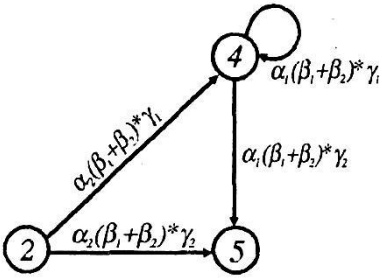
Фиг. 4.9. Елиминиран връх 3

където $\alpha_1, \beta_1, \alpha_2, \beta_2, \gamma_1, \gamma_2$ са регулярни изрази.

В частност, когато върховете 1 и 4 съвпадат се получава:



Фиг. 4.10. Примерна диаграма



Фиг. 4.11. Елиминиран връх 3

4.3.1.8. Теорема на Мур.

Съществува алгоритъм, посредством който може да се установи дали крайните автомати A и A' над азбуката Σ са еквивалентни, т.е. $\tilde{A} = \tilde{A}'$.

4.3.1.9. Алгоритъм за проверка:

а) Построява се сравнителна таблица, състояща се от три стълба. Елементите на таблицата са двойки върхове $< V, V' >$ на двата КА, където V е връх от автомата A, а V' - от A'.

Двойката от 1ред 1 стълб е двойката начални върхове на двата автомата.

Таблица 4.3. Алгоритъм за проверка на еквивалентност

1	2	3
$< V, V' >$	$< V_a, V'_a >$	$< V_b, V'_b >$
$< V_H, V'_H >$		

б) За всеки ред от таблицата се процедира по следния начин: Ако двойка $< V, V' >$ е в стълб 1, прибавя се в стълб 2 наредена двойка, образувана от върхове V_a и V'_a на двата автомата, където V_a е връх на автомата A, до който има a - дъга от връх V на автомата A, V'_a е връх на автомата A', до който има a - дъга от връх V' на автомата A'. Аналогично в стълб 3 се записва двойка $< V_b, V'_b >$, където V_b е връх с b - път от V, V'_b е връх с b - път от V'.

в) Ако $< V_a, V'_a >$ или $< V_b, V'_b >$ не се срещат преди това в стълб 1, прибавят се и процесът се повтаря.

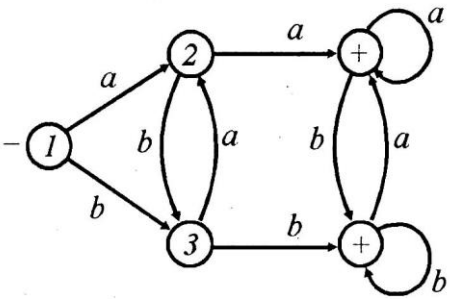
г) Ако в таблицата се получи такава двойка $< V, V' >$, за която V е краен връх на A, а V' е некраен връх на A' или обратно - процесът се прекратява със заключение - A и A' не са еквивалентни.

В противен случай A и A' са еквивалентни.

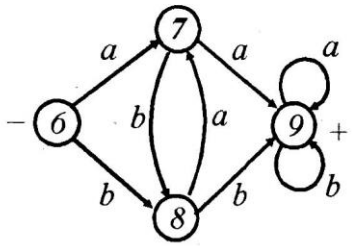
Процесът трябва да приключи, защото всички двойки в стълб 1 са различни, а съществува само краен брой различни двойки върхове от A и A'.

<u>Пример:</u>

Дадени са два крайни автомата A и A'. Автоматите да се проверят за еквивалентност



Фиг. 4.12. Краен автомат А



Фиг. 4.13. Краен автомат А'

Построява се сравнителна таблица за двата автомата

Таблица 4.4. Таблица за еквивалентност

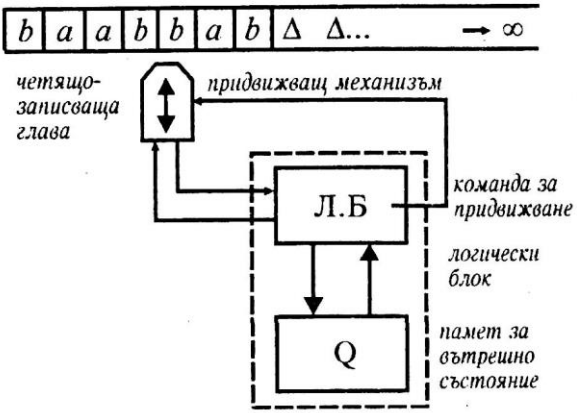
$< V, V' >$	$< V_a, V'_a >$	$< V_b, V'_b >$
< 1, 6 >	< 2, 7 >	< 3, 8 >
< 2, 7 >	< 4, 9 >	< 3, 8 >
< 3, 8 >	< 2, 7 >	< 5, 9 >
< 4, 9 >	< 4, 5 >	< 5, 9 >
< 5, 9 >	< 4, 9 >	< 5, 9 >

От сравнителната таблиця се вижда, че крайните върхове 4 и 5 на автомата A влизат в наредена двойка само с връх 9 на автомата A', който също е краен връх. Следователно двата автомата са еквивалентни.

5. Същност и действие на машината на Тюринг (МТ).

5.1. Същност на МТ.

Машината на Тюринг се състои от лента, разделена на клетки, четящо-записваща глава и краен автомат.



Фиг. 5.1. Структура на МТ
Краен автомат, представен чрез КОГ.

5.2. Съставни части на МТ над азбуката $\Sigma = (a, b)$

1. Лента, разделена на клетки. Съществува най-лява клетка, отдясно лентата е безкрайна. Всяка клетка съдържа точно 1 символ от Σ или от помощната азбука V, или празен символ (Δ).

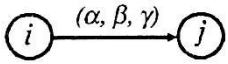
2. Четящо-записваща глава. Може да прочете символа, на който се намира, да се придвижва с 1-ва клетка вляво или вдясно, като евентуално запише върху лентата на мястото на прочетения символ, нов символ.

3. Краен автомат - описан чрез КОГ.

МТ съдържа и краен автомат, който се представя чрез краен ориентиран граф. Крайният автомат има едно състояние, обозначено със знак - или START и наречено начално (стартово) състояние.

КОГ на МТ има и крайно състояние, обозначено със знак + или HALT (стоп) и наречено крайно състояние.

Всяка дъга на КА има вида:



α, β, γ - дъга, маркирана с тройка.

Първият (α) и вторият (β) етикети на дъгата са елементи от множеството Σ или от работната азбука V или е празен символ:

$$\alpha \in \Sigma \cup V \cup \{\Delta\}$$

$$\beta \in \Sigma \cup V \cup \{\Delta\}.$$

Третият етикет (γ) определя посоката на придвижване на главата:

$$\gamma \in \{L, R\}; \quad L \text{ - ляво; } R \text{ - дясно.}$$

5.3. Действие на МТ.

Ако КА в МТ е в състояние i и главата е върху символа α, то КА ще премине в състояние j, главата ще запише символа β на мястото на α и ще се придвижи 1 стъпка вляво или вдясно в зависимост от γ. Всички дъги излизащи от състоянието i трябва да имат различен α (за да е детерминирано следващото състояние на КА).

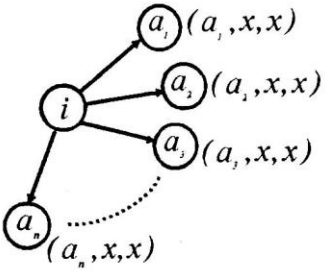
Първоначално главата е върху най-лявата позиция и отдясно на главата има записана някаква дума над Σ, наречена входна дума W.

Всички останали клетки до ∞ съдържат Δ. Изчислението започва от началното състояние на КОГ, като в зависимост от символите върху лентата и маркировките върху дъгите се преминава от състояние в състояние.

<u>I. Машината спира в следните случаи:</u>

1. При достигане на крайно състояние (HALT) - в този случай МТ е разпознала думата W. Множеството от всички думи W, които се разпознават от МТ се означават с accept (M).

2. Когато МТ не разпознава дадена дума, т.е. ако МТ е в състояние i и дъгите са маркирани с $\alpha_1, \alpha_2, \alpha_3, \ldots \alpha_n$, различни от символа върху лентата:



Фиг. 5.2. КОТ на МТ

α е текущият символ върху лентата,

където

$$\alpha \neq \alpha_1, \alpha_2, \ldots, \alpha_n.$$

В този случай върху дъгите има маркировки, различни от допустимите за текущото състояние, машината спира като отхвърля думата W.

3. Главата е най вляво и следва команда за движение наляво.

В този случай МТ също спира и също отхвърля разпознаваната дума.

II. Множеството от всички думи, които се отхвърлят се бележи с reject (M).

III. Във всички останали случаи машината зацикля.

Множеството от думи, при които МТ зацикля, бележим с loop (M)

<u>За произволна азбука Σ е в сила:</u>

1. $accept(M) \cap reject(M) \cap loop(M) = \varnothing$ - трите множества не се пресичат, т.е. няма дума, която да се разпознава и същевременно да се отхвърля и да предизвиква зацикляне.

2. $accept(M) \cup reject(M) \cup loop(M) = \Sigma^*$ - трите множе-

ства дават всички възможни думи, които могат да се образуват от Σ. Няма друго четвърто множество от думи, т.е. трите множества образуват тоталното множество от думи над Σ.

5.4. Изчисляване с МТ.

МТ може да се използва за разпознаване на езици, описани чрез всички видове граматики.

Пример:

Дадено: МТ, наречена M_1 над $\Sigma = \{a, b\}$ и $V = \{A, B\}$, КА описан с КОГ, разпознаващ множеството (езика L)

$$L = \{a^n b^n | n \geq \varnothing\}$$

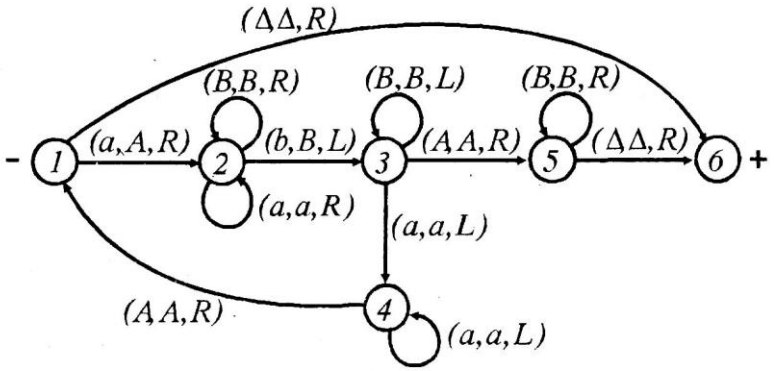
Трите множества от думи на M_1 са:

$$accept(M_1) = \{a^n b^n | n \geq \varnothing\}$$

$loop(M_1) = \varnothing$ - няма думи, при които МТ зацикля

$$reject(M_1) = \Sigma^* \backslash accept(M_1) \backslash loop(M_1) = \Sigma^* \backslash accept(M_1)$$

КОГ, който описва КА е от вида,

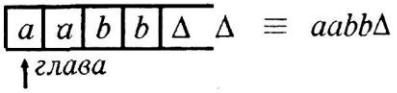


Фиг. 5.3. КА на МТ

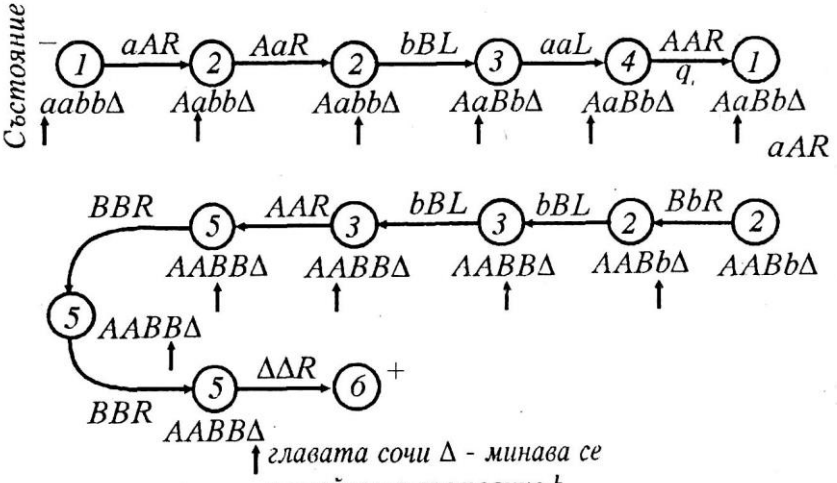
където R - дясно; L - ляво .

Проверката на действието на МТ може да се извърши чрез анализ на КОГ за думата *aabb*. За отбелязване е, че думите $a^n b^n$ не могат да се разпознаят само чрез КА, нужна е и външна памет, предоставена в случая от лентата на МТ).

Лентата е в начално състояние и върху нея е записана входната дума:



При преминаването във всяко ново състояние се обозначава текущото съдържание на лентата и позицията на главата:



Фиг. 5.4. Действие на МТ

МТ достига крайното състояние 6 без да спре, поради което тя разпознава входната дума. Описаната МТ разпознава и всяка друга дума от езика L на базата на циклична замяна на символите a с A,b и B с проверка. Тя ще спре преди крайното състояние, ако броят на a е различен от броя на b в поредиците върху лентата.

Машината заменя последователно по един символ от входната $\sum$ с работната $\sum$ като се движи вляво и вдясно, като при това се гарантира броят на А и броят на В да са еднакви.

Пример:

Да се построи МТ над азбуката $\Sigma = \{a, b\}$, която да намира най-големия общ делител на две положителни числа. Числата са записани върху лентата като поредици от a и b ограничени с Δ в началото и в края. Резултатът да се запише върху лентата като поредица от $a \ldots a$.

Решение:

Евклидов алгоритъм

1. Установява се главата на последното m-то a от редицата.

2. Заменя се последователно първото срещнато наляво a с A и първото срещнато надясно b с B (сравняване на числата), докато се достигне левия ограничител Δ $(m \le n)$ или десния ограничител Δ $(m > n)$.

3. При $m \le n$ се заменят всички A с Δ и всички B с a. Ако върху лентата няма повече b-та $\rightarrow HALT$. Резултатът е върху лентата $(m = n)$.

Ако върху лентата има b-та се преминава към точка 2.

4. При $m > n$ се заменят всички B-та с Δ, всички A, освен последното с b (изваждане) и се преминава към точка 2.

Действие на алгоритъма:

$$\Delta \underbrace{aa\ldots a}_{m}\underbrace{bb\ldots b}_{n}\Delta$$

1. $m < n$

$$\Delta \underbrace{aa\ldots a}_{m}\underbrace{bb\ldots b}_{m}\underbrace{bb\ldots b}_{n-m}\Delta \Rightarrow \Delta\Delta\ldots\Delta\underbrace{aa\ldots a}_{m}\underbrace{bb\ldots b}_{n-m}\Delta$$

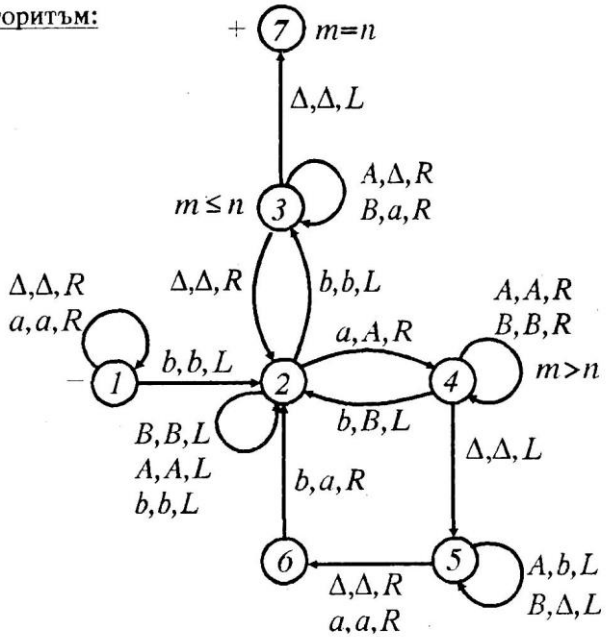
2. $m = n$

$$\Delta \underbrace{aa\ldots a}_{m}\underbrace{bb\ldots b}_{n}\Delta \Rightarrow \underbrace{\Delta\Delta\ldots\Delta}_{m}\underbrace{aa\ldots a}_{m=n}\Delta \qquad (n-m) = \varnothing$$

3. $m > n$

$$\Delta\, \underbrace{aa\ldots a}_{m-n}\, \underbrace{aa\ldots a}_{m}\, \underbrace{bb\ldots b}_{n}\, \Delta \Rightarrow \underbrace{aa\ldots a}_{m-n}\, \underbrace{bb\ldots b}_{m}\, \underbrace{\Delta\,\Delta\ldots\Delta}_{n}\, \Delta$$

<u>Алгоритъм:</u>



Фиг. 5.5. МТ за алгоритъм на Евклид

Илюстрация на действието за $m = 4$ и $n = 6$.

Δ	*a*	*a*	*a*	*a*	*b*	*b*	*b*	*b*	*b*	*b*	Δ	1	
Δ	*A*	*A*	*A*	*A*	*B*	*B*	*B*	*B*	*b*	*b*	Δ	2	
Δ	Δ	Δ	Δ	Δ	*a*	*a*	*a*	*a*	*b*	*b*	Δ	3	
Δ	Δ	Δ	Δ	Δ	*a*	*A*	*A*	*A*	*B*	*B*	Δ	2	
Δ	Δ	Δ	Δ	Δ	*a*	*b*	*b*	*b*	Δ	Δ	Δ	4	
Δ	Δ	Δ	Δ	Δ	*a*	*a*	*b*	*b*	Δ	Δ	Δ	2	
Δ	Δ	Δ	Δ	Δ	Δ	Δ	*a*	*a*	Δ	Δ	Δ	3	→ *HALT*

Илюстрация на действието за $m = 3$ $n = 2$

Δ	a	a	a	b	b	Δ
Δ	A	A	A	B	B	Δ
Δ	b	b	b	Δ	Δ	
Δ	a	b	b	Δ	Δ	
Δ	A	B	b	Δ	Δ	
Δ	Δ	a	b	Δ	Δ	
Δ	Δ	A	B	Δ	Δ	
Δ	Δ	Δ	a	Δ	Δ	

5.5. Класификация и йерархия на абстрактните автомати (АА).

МТ е най-висшият АА, който служи за определяне изчислимостта на функциите и разрешимостта на отношенията. МТ има най-голяма описателна и изчислителна мощност. Всяко средство за изчисление на функции или описание на дискретни системи (структури) се сравнява по мощност с МТ, чрез което се определя принципно класът от задачи, които могат да бъдат решени с него.

Видове АА.

5.5.1. Краен автомат (КА)

Заложени са следните ограничения на МТ, за да се получи КА:

1. Главата се движи само надясно (не може да се връща понеже няма външна памет). Лентата се състои от набора от входните символи, които се разпознават от КОГ на КА. Σ е входна азбука, $\Sigma \cup V$ - изходна азбука.

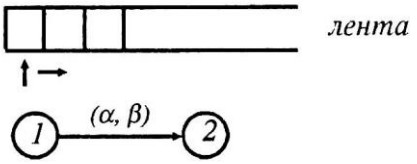
2. Движейки се надясно главата записва нов символ върху лентата вместо прочетения.

3. Всяка една дъга има вида (α, β) - отсъства командата R

или L

α - входен символ от лентата;

β - изходен символ (символът, с който се замества α).

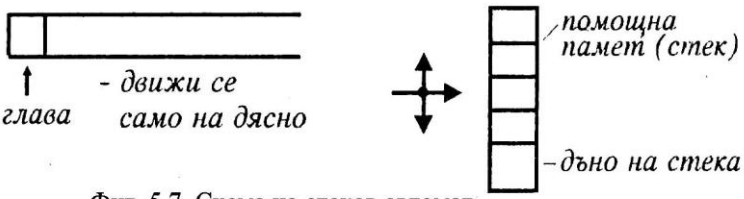


Фиг. 5.6. Схема на краен автомат

4. КА разпознава регулярните множества.

5.5.2. Стеков автомат (СА).

Стековият автомат е автомат с външна памет. Той може да се представи като МТ, която има две ленти - входна и стекова. Входната лента съдържа входния низ. Той се прочита от глава, която се движи само надясно. Стековата лента служи за запис и четене на символи. Върху нея работи четящо-записваща глава, която се движи в двете посоки - нагоре и надолу. Характерно за стековия автомат е връзката между командата за движение, например ***push/pop***, с операцията, извършвана върху лентата - запис/четене. Всяка дъга има три етикета. Първият съдържа символ, който се чете от входната лента. Вторият е символ, който се записва/чете в/от стековата лента. Третият определя операцията върху стековата лента и посоката, в която се придвижва стековата глава.



Фиг. 5.7. Схема на стеков автомат

Принципът на организация на стека е LIFO. Последният записан в стека символ се прочита първи.

Σ - входна азбука $\Sigma \cup V$ - изходна азбука

$\Sigma \cup V$ - стекова азбука.

5.5.3. Броячен автоман (БА).

Броячният автомат е еквивалентен на стеков автомат, при който стековата азбука съдържа само един символ, например a, т.е. в стека може да се записва само един и същ символ и стекът отчита броя на записаните символи.

5.5.4. Недетерминирани автомати (НА).

Недетерминиран се нарича автомат, при който КА, описан с КОГ допуска повече α маркировки от състояние i към останалите състояния. В зависимост от прочетеният символ от лентата, възможните следващи състояния на КА могат да бъдат повече от едно.

Изборът за преход към следващо състояние се извършва случайно.

5.6. Йерархия на описаните АА.

1. Най-слабомощни са КА и недетерминираният краен автомат (НКА) (ако КА не може да реши дадена задача, то и НКА не може да я реши).

2. По-мощни от тях са КА с 1 брояч.

3. КА с 1 стек.

4. НКА с 1 стек са по-мощни от КА с 1 стек.

5. МТ.

Машината на Тюринг е еквивалентна на краен автомат с повече от два брояча, краен автомат с повече от два стека и на недетерминиран краен автомат с повече от два брояча или стека.

КА	=	НКА	1)	$\}G_3$
КА с 1 брояч			2)	
КА с 1 стек			3)	G_2
НКА с 1 стек			4)	
КА с $\geq$ 2 брояча КА с $\geq$ 2 стека НКА с $\geq$ 2 брояча НКА с $\geq$ 2 стека МТ			5)	G_1 и G_0

5.7. Еквивалентност между АА и граматики.

Между граматиките и езиците съществува еквивалентност по изчислителна (описателна) мощност.

Регулярните езици се разпознават от крайни автомати, недетерминирани крайни автомати и крайни автомати с един брояч.

Контекстно свободните езици се разпознават от крайни автомати с един стек или от недетерминирани крайни автомати с един стек.

Контекстно зависимите езици и естествения език се разпознават от МТ и нейните еквивалентни автомати.

Таблица 5.1. Граматики и еквивалентни автомати

Тип на G	Название	Еквивалентен АА
0	Без ограничение	Машина на Тюринг КА с $\geq$ два стека
1	Контексно зависима	КА с $>$ два брояча
2	Контексно свободна	КА с един стек Недетерминиран КА с 1 стек
3	Регулярна	КА Недетерминиран КА

В повечето случаи на програмните езици в практиката се работи с тип 2 и тип 3, по-рядко с тип 1 и изключително рядко с тип 0.

5.8. Фон Нойманов автомат (ФНА).

5.8.1. Същност.

МТ работи строго последователно, като при 1 такт се обработва 1 клетка. По принцип са възможни 2 случая на обработка на лентата.

1) Между обработките на двете клетки съществува причинно следствена връзка - по принуда те се обработват последователно.

2) Не съществува такава причинно-следствена връзка - тези клетки могат да се обработват паралелно.

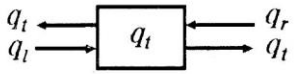
Принципно е възможно върху лентата на МТ да работят много такива екземпляри от КА. За тях лентата е обща. Това важи само

за случаите без причинно-следствена връзка.

Джон Фон Нойман създава своя машина ФНА. Всички съвременни компютри са ФНА. ФНА е еквивалентен на МТ, но изчислява по-бързо.

5.8.2. Принцип на действие на ФНА.

ФНА се състои от много екземпляри автомати, способни да работят паралелно, като единичният автомат има следния вид:



Фиг. 5.8. Архитектура на Нойман

Основен елемент е КА над $\Sigma = \{q_1, q_2, \ldots, q_r\}$

Всеки един елементарен автомат има вътрешно състояние q_t, което се предава наляво и надясно. Отдясно постъпва състояние q_r на десния автомат, отляво q_l - на левия автомат.

Състоянието на автомата в следващия момент е функция от текущото състояние и състоянието на автоматите отляво и отдясно:

$$q_t^{+1} = \psi(q_l, q_t, q_r),$$

q_t - текущо състояние.

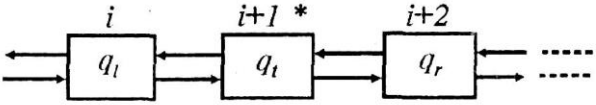
При равенство на състоянията на текущия, левия и десния автомат, текущия автомат се намира в неактивно (спокойно) състояние:

$$q_l = q_t = q_r \text{ - спокойно състояние.}$$

Новото състояние може да се разглежда като получено от текущото, състоянието на левия и десния автомати, получени чрез правило (предписание):

$$q_l, q_t, q_r \rightarrow q_t^{+1}.$$

Основният елемент обединява съхранение и обработка на информацията и като цяло тази машина на Нойман изглежда като верига от клетки.

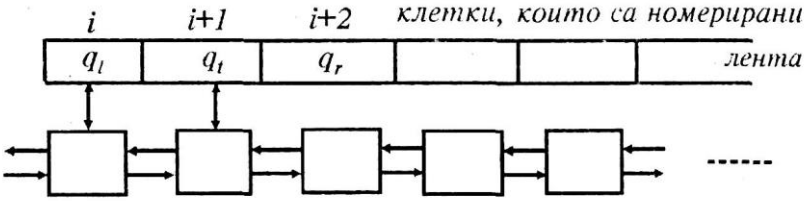


Фиг. 5.9. Машина на Нойман

Ако се постави ограничение от едната страна на веригата функциите на автомата не се влошават.

5.8.3. Представяне на ФНА чрез МТ.

Съществува еквивалентно представяне на ФНА чрез МТ, но тогава машината няма памет. Тя се състои от лента, съставена от клетки, които са номерирани. Клетките съхраняват състоянието на автоматите, а самите автомати са без вътрешна памет:



Фиг. 5.10. Представяне на ФНА чрез МТ

Част от автоматите са активни, а друга неактивни. В даден момент може да се изчислява в две или повече клетки от лентата. Управлението на преместването се осъществява чрез напречните връзки $(\updownarrow)$. Действието на ФНА наподобява действието на процесор, който работи върху оперативна памет.

5.8.4. Съответствие между ФНА и компютър.

ФНА	компютър (ЕИМ)
Работи с безкрайна лента	Работи с крайна памет
Клетки	Думи (байтове, дв. думи)
Достъп чрез четене/запис до клетка	Достъп четене/запис до машина дума
Номер на клетка	Машинен адрес
Основен елемент	Процесор
Верига от еднакви основни елементи	Симулира се с един процесор във времеделение
Връзки между елементите	1) за > 1 процесор - синхронизация между процесорите 2) за 1 процесор, реализира причинно-следствена връзка чрез програма

ФНА обосновава теоретично възможността за паралелни изчисления с повече от един процесор.

ФНА (МТ) могат да емулират себе си. Това се осъществява като на лентата се внесе информация за действието на емулираната машина и обосновава теоретично възможността за създаване на интерпретатори и виртуални процесори.

5.8.5. Приложение на МТ.

МТ се използва главно в три аспекта:

1. Като разпознавател на езици.
2. Като генератор на думи.
3. Като разрешител за предикатни функции.

ФНА има директен достъп до лентата и не губи време за преместване по нея, затова е и по-бърз.

6. Рекурсивни функции (РФ).

Частично рекурсивни функции (ЧРФ).

Една функция от n променливи

$f(x_1, x_2, \ldots, x_n)$, $n \geq 1$ е рекурсивна, ако изобразява n-торки от думи над азбуката в думи от същата азбука $(\sum)$. Частично рекурсивната функция е такава функция, която не е дефинирана за всички n-точки.

Примитивно рекурсивни функции (ПРФ).

Класът на ПРФ над дадена $\sum = \{a, b\}$ се дефинира по следния начин:

6.1. Базови функции:

1. Към ПРФ принадлежат следните функции.

$nil(x) = \Lambda$ за $x \in \sum^*$ стойността е прзната дума (Λ)

$consa(x) = ax$

$consb(x) = bx$

2. Получени чрез суперпозиция:

ако $h, g_1, \ldots, g_m$ са ПРФ, то f е също ПРФ

при $f(x_1, x_2, \ldots, x_m) = h\big(g_1(x_1 \ldots x_n), \ldots, g_m(x_1 \ldots x_n)\big)$

3. Получени чрез примитивна рекурсия:

Ако g, h_1, h_2 са ПРФ, то f е също ПРФ при:

а) $f(\Lambda, x_2, \ldots, x_n) = g(x_2, \ldots, x_n)$

$$f(a, x_1, \ldots, x_n) = h_1(x_1, \ldots, x_n, f(x_1, \ldots, x_n))$$

$$f(b, x_1, \ldots, x_n) = h_2(x_1, \ldots, x_n, f(x_1, \ldots, x_n))$$

някои аргументи могат да липсват

б) за $n = 1$ (едноаргументна функция)

$f(\Lambda) = w$, където $w \in \sum^*$

$$f(ax) = h_1(x, f(x))$$

$$f(bx) = h_2(x, f(x))$$

в) *за* $n = 2$ (двуаргументна функция)

$$f(\Lambda, x_2) = g(x_2)$$

$$f(a, x_1, x_2) = h_1(x_1, x_2, f(x_1, x_2))$$

$$f(b, x_1, x_2) = h_2(x_1, x_2, f(x_1, x_2))$$

<u>Примери за ПРФ</u>

1. Функция константа: $c(x) = c$

$$h = consa \qquad g_1 = nil(x)$$

$$h = consb \qquad g_1 = nil(x)$$

$$a(x) = a = consa(nil(x)) = a\Lambda = a$$

$$b(x) = b = consb(nil(x)) = b\Lambda = b$$

2. Функция идентичност $iden(x) = x$

$$g = nil(x) = \Lambda$$

$$h_1 = consa(x)$$

$$h_2 = consb(x)$$

$$iden(\Lambda) = nil(x) = \Lambda$$

$$iden(a, x) = consa(x) = ax$$

$$iden(b, x) = consb(x) = bx$$

3. Функция конкатенация $append(x_1, x_2) = x_1x_2$

$$g = iden(x)$$

$$h_1 = consa(append(x))$$

$$h_2 = consb(append(x))$$

$$append(\Lambda, x_2) = iden(x_2) = x_2$$

$$append(a, x_1, x_2) = consa\big(append(x_1, x_2)\big)$$

$$append(b, x_1, x_2) = consb\big(append(x_1, x_2)\big)$$

От примитивна рекурсия за $n = 2$

4. Функция обратен ред: $reverse(x)$

$$h = nil(x)$$

$$g_1 = reverse(x)$$

$$g_2 = a(x)$$

$$reverse\ (\Lambda) = nil(\Lambda) = \Lambda$$

$$reverse(ax) = append\big(reverse(x), a(x)\big)$$

$$reverse(bx) = append\big(reverse(x), b(x)\big)$$

Приема се, че *reverse* е известна и се използва за обръщане на думата и след това се конкатенира *a* в края.

6.2. Частично рекурсивни функции.

Частично РФ с $\Sigma = \{a, b\}$

1. Всяка ПРФ е ЧРФ.

2. Ако $h(x_1, x_2, \ldots, x_n, Y)$ е ПРФ, то $f(x_1, x_2, \ldots, x_n)$ е ЧПФ чрез прилагане на оператора за минимизация μ

$f(x_1, x_2, \ldots, x_n) = \mu Y\big(h(x_1, \ldots, x_n, Y = 0\big) = Y$, т.е.

$f(x_1, x_2, \ldots, x_n) = Y$

$h(x_1, \ldots, x_n, 0) \neq 0, \ldots, h(x_1, \ldots, x_n, Y - 1) \neq 0$ и

$h(x_1, \ldots, x_n, Y) \neq 0$

7. Паралелни системи и процеси.

7.1. Паралелизъм - същност.

7.1.1. Физически процеси.

В природата съществуват многобройни физически процеси. Дискретни физически процеси са такива процеси, за които могат да се обособят състояния. Процесите не протичат независимо един от друг. Те се синхронизират, кооперират и комуникират (обменят енергия, материали, информация) помежду си.

Характеристика на процесите:

- състоянията се заемат последователно (един процес не може да заема няколко състояния едновременно);

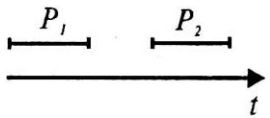
- състоянията траят крайно време, след което процесът преминава в ново състояние. Това важи за всяко състояние с изключение на крайните (ако има такива). Повечето процеси са циклични, те нямат крайни състояния.

Тези процеси могат да се изследват и да се моделират чрез абстрактни автомати.

7.1.2. Отношения между процесите.

Два процеса имат следните отношения:

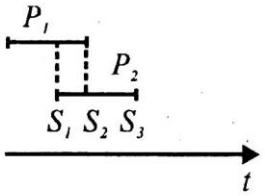
1) Те протичат последователно един след друг.



Фиг. 7.1. Последователни процеси

Процесите не се синхронизират и не комуникират по между си. Два съседни процеса могат да се разглеждат като един.

2) Те протичат паралелно един спрямо друг.



Фиг. 7.2. Паралелни процеси

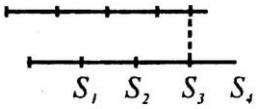
Два процеса P_1, P_2 са успоредни ако се препокриват във времето поне в едно състояние S_i.

Отношения между паралелните процеси (ПП)

1. Независими ПП. - Те са паралелни във времето, но между тях не съществува каузална (причинно-следствена) връзка. Те протичат асинхронно и не обменят информация. Модел на такива процеси са два паралелни автомата.

2. Зависими ПП. - Два паралелни процеса са зависими, ако между тях съществува поне едно каузално отношение. Каузално отношение между P_1 и P_2 е налице, ако преходът към определено състояние на P_1 е обусловено само при достигане на определено друго състояние на P_2. Зависимите процеси се синхронизират и комуникират помежду си в дискретни моменти върху общата ос на времето.

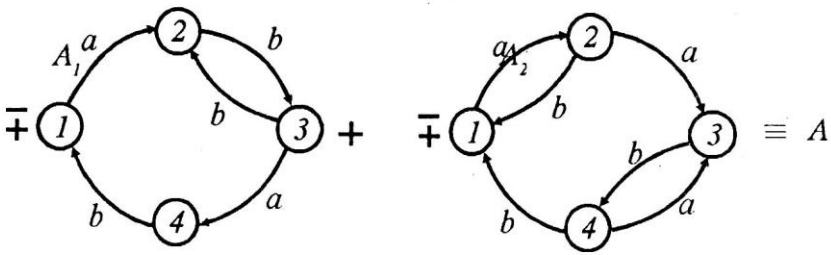
Процесите се синхронизират само в мястото на каузална връзка, а в останалото време протичат асинхронно.



7.1.3. Паралелни автомати (ПА).

Цел - моделиране на паралелни процеси (ПП).

Пример: Дадени са два ПА (A_1, A_2) с по 4 състояния



$$R_1 = \left(ab^*bab\right)^* \qquad R_2 = \left((ab)^*a(ba)^*bb\right)^*$$

Фиг. 7.3. Паралелни автомати

Нека А е общ автомат. Ако A_1 и A_2 са независими, А има общо 16 състояния и голямо количество дъги.

Ако имаме 2 независими ПА и не можем да отделим A_1 от A_2, то трябва да реализираме процесите с общ автомат А, който е много сложен. Трудно може да се определи каква граматика разпознава общия А.

При анализиране на сложни процеси стремежът е те да се декомпозират на зависими или независими отделни ПА, с цел получаване на няколко по-прости ПА.

7.1.4. Паралелизъм в изчислителната техника.

1. Паралелни изчисления. Те се основават на разгледания Фон Нойманов автомат (ФНА).

2. Работа в реално време (Управление на производствени процеси). Процесите протичат синхронно със събитията в управлявания свят.

3. Многопотребителска (многотерминална) работа .

4. Многопрограмна работа

В този режим едновременно се изпълняват няколко програми. Изчислителната мощност се превключва между програмите, симулирайки едновременното им изпълнение.

7.2. Теория на мрежите на Петри (МП).

7.2.1. Структура.

Дефиниране:

Структурата C на МП се състои от четири множества

$$C = (P, T, I, O)$$

P (Places) - множество на местата

T (Transitions) - множество на преходите

I (Input) - множество на входящите функции

O (Output) - множество на изходящите функции

I и *O* описват отношенията между преходи и места.

Входна функция на прехода t_j е $I(t_j)$. Тя присвоява на прехода t_j подмножество от места *P*, наречени входни места на

прехода. Те се определят от $I(t_j)$.

Изходна функция на прехода за t_j и $O(t_j)$. Тя присвоява на прехода t_j подмножество от места P, наречени изходни места за прехода.

7.2.2. Дефиниция за граф на МП (G).

G е насочен двумножествен граф:

$$G = (V, A)$$

$V = \{v_1, v_2, \ldots, v_n\}$ - множество от върховете на G

$A = \{a_1, a_2, \ldots, a_m\}$ - множество от насочени дъги

Всяко a_i може да се представи чрез наредена двойка:

$$a_i = \langle v_j, v_k \rangle \mid v_j, v_k \in V$$

V се представя чрез две подмножества $V = \{P, T\}$, за които са валидни следните операции:

$$V = P \cup T, \quad P \cap T = \varnothing$$

Ако $a_i = \langle v_j, v_k \rangle$, то $v_j \in P, \ v_k \in T$

или $v_j \in T, \ v_k \in P$

7.2.3. Маркиране на МП.

Маркиране означава присвояване на символи (марки) на местата в МП. Местата се разглеждат като памет (броячи), в които се запомня броя на присвоените марки. Броят на марките се променя при изчисление с мрежата (при работа на мрежата).

Маркирането може да се дефинира като n-мерен вектор:

$\mu = \{\mu_1, \mu_2, \ldots, \mu_n\}$, където n е мощност на множеството P

$n = | p | \mu_i \in N$, където N е множество от цели положителни числа

$\mu(p_i) = \mu_i \mid i = 1...n$

Маркирана МП (ММП)

Състои се от структура на мрежата и вектор маркировки:

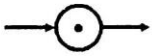
$M = (C, \mu)$ - ММП, където $C = (P, I, T, O)$

$M = (P, T, I, O, \mu)$

Марките в мрежата се представят чрез символи.

Удобно е обозначаване с $\bullet$. Всяко място може да има неограничен брой марки.

Пример:

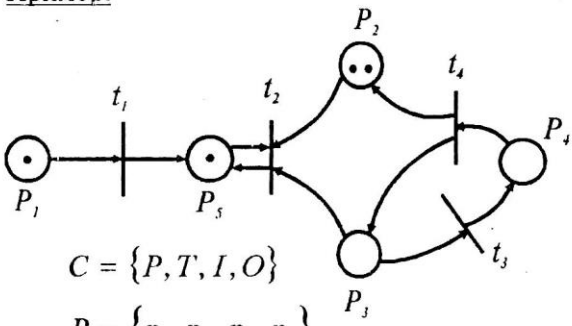


Ако броят марки не може да се разположи в мястото, обозначава се с число:

Множеството от всички възможни маркировки на ММП с n-места е всъщност множество от всички вектори μ.

Някои от тези вектори могат да се разгледат като начални, останалите като текущи.

Пример:



$C = \{P, T, I, O\}$

$P = \{p_1, p_2, p_3, p_4\}$

$T = \{t_1, t_2, t_3, t_4\}$

Фиг. 7.4. Граф на ММП

$I(t_1) = \{p_1\}$ $\qquad$ $O(t_1) = \{p_5\}$

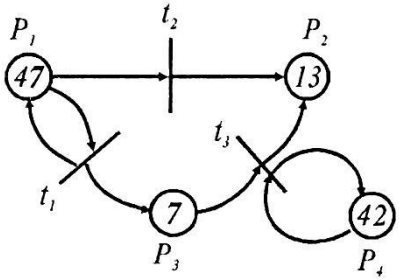
$I(t_2) = \{p_2, p_3, p_5\}$ $\qquad$ $O(t_2) = \{p_5\}$

$I(t_3) = \{p_3\}$ $\qquad$ $O(t_3) = \{p_4\}$

$I(t_4) = \{p_4\}$ $\qquad$ $O(t_4) = \{p_2, p_3\}$

$M = (C, \mu)$ $\qquad$ $\mu = (1,2,0,0,1)$

Пример:



$\mu = (47, 13, 7, 42)$

7.2.4. Работа на ММП.

Работата на мрежата се управлява от текущия брой и положение на марките в мрежата. Марките се преместват като преминават от едно място в съседното, преминавайки през преходите.

Мултидъги се наричат паралелни еднопосочни дъги от преход към място или обратно.

Преходите пропускат марките през себе си, когато са активни ("запалени"). Един преход е активен или преходът сработва, ако във всяко негово входно място има поне толкова марки, колкото дъги има от мястото до прехода, като за мултидъгите са необходими повече от 1 марки.

$t_j \in T$ в ММП

$M = (P, T, I, O, \mu)$ е активен, ако за всички $p_i \in P$

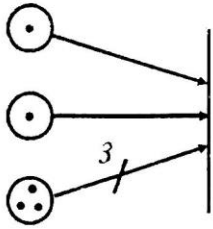
$\mu(P_i) \geq \#(P_i, I(t_j))$

Чрез # се обозначава колко пъти място p_i се явява входно за прехода t_j (колко входни дъги има от място p_i до t_j).

В следващия пример преходът е активен.

Пример:

Един преход сработва, когато е активен. От всяко свое входно място той премахва толкова марки, колкото е броят на входните дъги от мястото към него и във всяко свое изходно място добавя толкова марки, колкото е броят на изходните дъги от него към мястото.



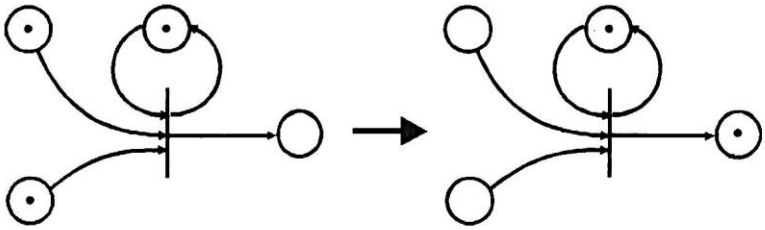
Сработването на преходите променя текущото маркиране μ до ново μ'. Броят на марките в местата не може да бъде отрицателен.

Сработване на преходи.

t_j с брой марки μ сработва, ако е активен и генерира μ'.

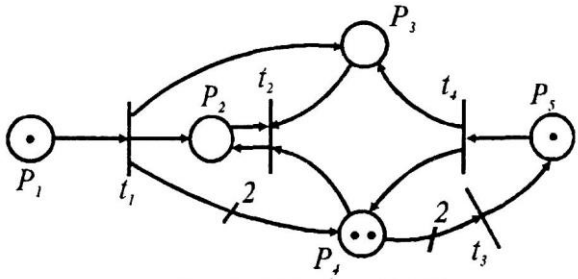
$$\mu'(P_i) = \mu(P_i) - \#(P_i, I(t_j)) + \#(P_i, O(t_j))$$

Пример:



Фиг. 7.5. Сработване на преход

Маркировката на място P_i се намалява с толкова марки $\#(P_i, I(t_j))$, колкото са входните дъги от мястото P_i до активния преход t_j и се увеличава с толкова марки $\#(P_i, O(t_j))$, колкото са изходните дъги между активния преход и мястото P_i.



Фиг. 7.6. Граф на ММП

7.2.5. Свойства на ММП.

7.2.5.1. Надеждност на ММП.

Една ММП е надеждна, ако броят на маркировките в местата никога не е по-голям от 1, т.е. за всички $P_i \mid i \geq 0$,където $P_i \in P$ на мрежата $M(C, \mu)$ важи:

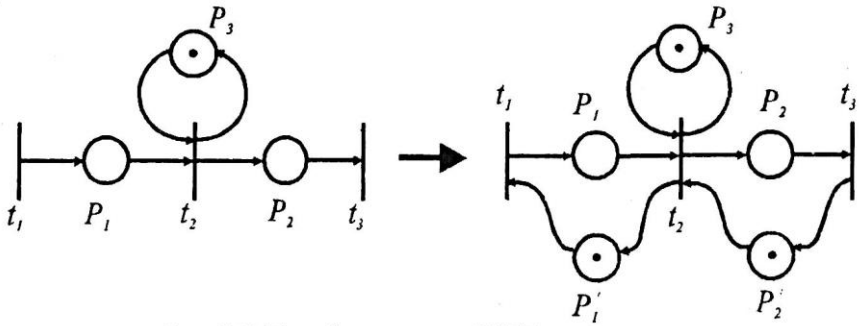
за всички $\mu_j(P_i) \leq 1$.

Това свойство е важно за апаратното (хардуерно) реализиране на мрежата.

Всяка ММП, която не съдържа мултидъги може да се преобразува в надеждна мрежа.

Пример:

Началната мрежа е единична.



Фиг. 7.7. Преобразуване на ММП в надеждна мрежа

Първата мрежа е ненадеждна, тъй като n пъти преходът t_3 ще се пълни с марки, ако t_2 се активизира многократно.

Преобразуваната мрежа е надеждна, тъй като в t_3 не може да се натрупа повече от 1 марка.

7.2.5.2. Ограниченост на ММП.

ММП е ограничена, ако общият брой на маркировките в местата никога не е по-голям от k (k - ограниченост).

За всяко $P_i \in P$ в $M(C, \mu)$

$$\sum_i \mu_j(P_i) \leq k$$

Надеждната мрежа е частен случай на ограничената.

Ограничените ММП са също удобни за моделиране на хардуерни устройства.

7.2.5.3. Консервативност на ММП.

ММП е консервативна, ако общият брой маркировки в мрежата остава постоянен.

$$\sum_i \mu_j(P_i) = k \text{ за всяко } P_i \in P$$

Такива мрежи се използват при разпределение или управление на ресурси. Ресурсите се представят като маркировки и които не изчезват, а се използват от различни консуматори.

7.2.5.4. Живучест на ММП.

Има ситуации, при които ММП спира да работи - изчерпват се възможностите за сработване на всички преходи. Ситуацията се нарича блокада (dead lock).

Възможна е тотална или частична блокада на мрежата.

Частична - само определена част от ММП не работи.

Мрежата има живучест, ако не изпада в състояние на блокада.

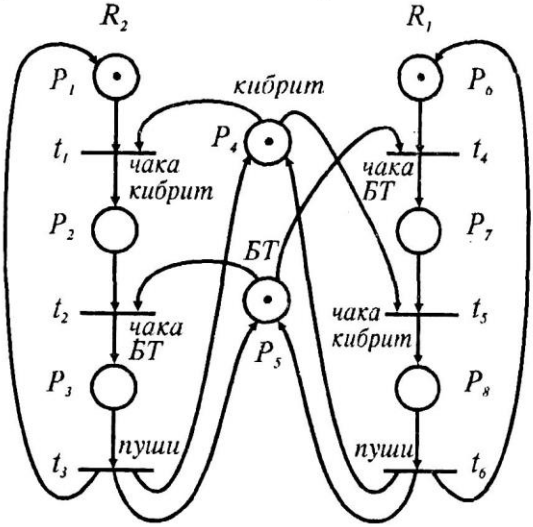
Живучестта зависи от началната маркировка и структурата на ММП.

Да се определи възможността за блокада е една от основните задачи на анализа на ММП.

Преобразуването на ММП в живуча се свежда до формиране на такава маркировка, в която не се получава блокада.

Пример: 2 пушачи.

Маса, на нея кибрит и цигари. Когато всеки един от пушачите вземе един от ресурсите, например първият - кибрита, а вторият - цигарите, двамата взаимно се блокират.



Фиг. 7.8. Моделиране на процеси чрез ММП

7.2.5.5. Достижимост и покриваемост на маркировки.

Свойствата на мрежата могат да се определят посредством анализ на действието на мрежата.

Този анализ се състои в определяне на текущите вектори маркировки, които мрежата получава.

Достижимост: Достижимостта е свързана с множество от достижими текущи маркировки при работа на мрежата. Това множество зависи от началното маркиране.

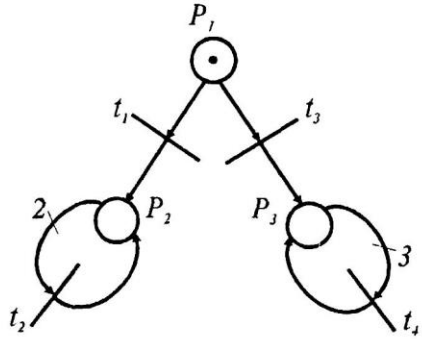
Нека е дадена ММП $M = (C, \mu)$, където μ е начална маркировка.

Определя се съществува ли в множеството от текущите маркировки $\mu' \in R(C, \mu) \mid R$ - множество от достижими маркировки от начална маркировка μ. Маркировката μ' се нарича достижима маркировка.

Покриваемост:

Дадена маркировка може да не е достижима, но да е покриваема. Нека е дадена ММП $M = (C, \mu)$ | μ - начална маркировка.Ако $\mu' \in R(C, \mu)$, покриваемостта определя съществува ли $\mu'' \in R(C, \mu)$, такава че $\mu'' \geq \mu'$

Пример:



P_1 P_2 P_3

$\mu = (1, 0, 0)$

Фиг. 7.9. Граф на ММП

$\mu' = (0, 2, 0)$ е достижима $\rightarrow$ покриваема

(• от P_1 отива в P_2 и след завършване се получават 2 •)

$\mu'' = (0, 0, 2)$ не е достижима, но е покриваема, тъй като съществува вектор $(0, 0, 3)$, за който $\mu(P_3) > \mu''(P_3)$.

Маркировката $\mu''' = (2,0,0)$ не е достижима и не е покриваема.

7.2.6. Техника за анализ на ММП.

Съществуват три техники (способи) за анализ на ММП: чрез дърво на достижимост, чрез матрици и чрез таблица на преходите.

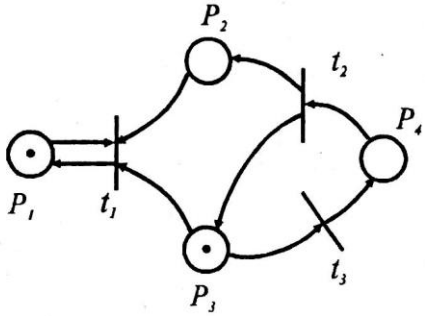
7.2.6.1. Анализ чрез дърво на достижимост.

От началната маркировка се построява дървовиден граф. Във възлите се нанасят достижимите маркировки, а по дъгите - сработилите преходи.

Пример: P_1 P_2 P_3 P_4

Началното маркиране е: (1, 0, 1, 0)

$$(1, 0, 1, 0)$$
$$\downarrow t_2$$
$$(1, 0, 0, 1)$$
$$\downarrow t_2$$
$$(1, w, 1, 0)$$
$$\downarrow t_1 \quad \downarrow t_3$$
$$(1, w, 0, 0) \quad (1, w, 1, 0)$$
$$\downarrow$$
$$(1, w, 1, 0)$$



Фиг. 7.10. Граф на ММП

С w е обозначено безкрайно маркиране

Например в P_2 няма изтичане на марки, затова се обозначава с w. Допуска се, че първо сработва t_3.

Редът, в който се разглеждат сработванията на преходите, няма никакво значение.

7.2.6.2. Анализ чрез матрици.

Входно/изходните функции се представят чрез две матрици (това са I и O). В колони се записват преходите, а по редове местата.

Съдържанието на матриците са текущите маркировки. Във връзка с действията се дефинират съответните програмни операции. Този метод е удобен за машинен анализ.

7.2.6.3. Анализ чрез таблица на преходите.

Таблицата има толкова колони, колкото места има мрежата. На всеки ред се нанася по един вектор маркировки. Началното маркиране е в първия ред. Със сработването на всеки преход се

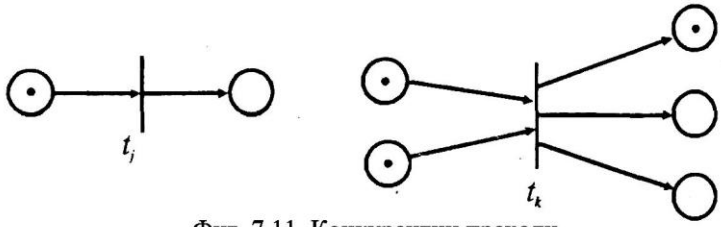
добавя по един ред в таблицата. Методът се използва при ръчно анализиране на действието на ММП.

7.2.7. Конкурентност и конфликт в ММП.

В ММП активните преходи сработват конкурентно (паралелно).

Такива преходи са независими един от друг. Ако има причинно-следствена връзка между преходи, то тя се отразява явно чрез структурата на мрежата.

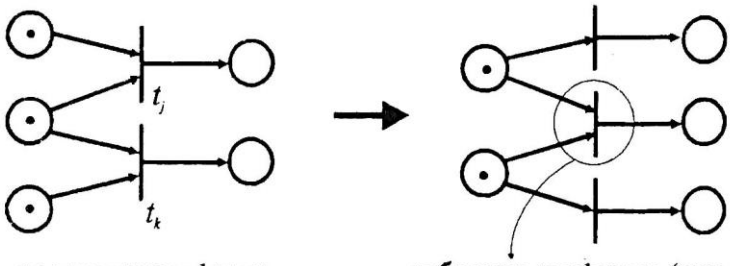
Последователността на разглеждане на конкурентните преходи е без значение. Подхожда се случайно. Времето за сработване се счита идеално за $t = 0$.



Фиг. 7.11. Конкурентни преходи

Конфликт.

В определени случаи се моделират ситуации, при които не е допустимо преходите да сработват конкурентно. Тогава се вземат мерки, за да се гарантира сработване "един след друг" (последователно).



разрешен конфликт

забранен конфликт (има повече от 1 входно място)

Фиг. 7.12. Конфликт в преходи

В първия пример конфликтът е разрешен, тъй като преходите t_j и t_k се изключват взаимно. P_i се явява ресурс за преходите t_j и t_k.

При втория пример има забранен конфликт, тъй като обозначения преход има повече от едно входно място.

7.3. Мрежови езици.

Събитията в паралелните системи, които се моделират чрез ММП, се представят в мрежата чрез сработване на преходите. Преходите могат да бъдат означени (надписани) със символите от дадена азбука. В този случай към мрежата може да се приложи формалната теория на езиците и граматиките. Такова разглеждане води до определяне изчислителната мощност на мрежите и съответно еквивалентността им с абстрактните автомати (АА).

Като се разглеждат мрежовите езици, се поставят две цели: да се определи какъв тип са мрежовите езици и каква е изчислителната мощност на ММП.

7.3.1. Обозначаване на мрежата.

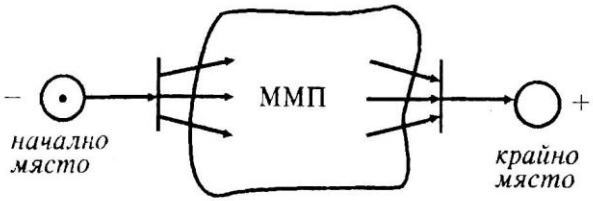
Преходите се обозначават с букви от азбуката $\Sigma=\{a,b\}$. Има 3 случая на обозначаване:

1) Всички обозначения са различни, т.е. всеки преход се обозначава с различна буква.

2) Допускат се еднакви обозначения (може да има преходи, обозначени с еднакви символи).

3) Като 2) + Λ - допускат се еднакви обозначения и преходи, обозначени с празен символ.

Определено множество от маркирания се счита за начално, а определено множество от маркирания се счита за крайно. Едно от местата се обозначава със знак "+" и се нарича крайно място, а друго - със знак "-" е начално място.



Фиг. 7.13. Обозначаване на ММП

Възможно е маркиране на преходите с повече от една буква.

7.3.2. Операции над низове в ММП.

1) Съединяване (конкатенация):

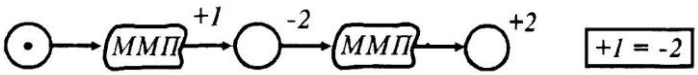
За два езика L_1 и L_2 операцията конкатенация образува множество от думи:

$L_1L_2 = \{x_1x_2 | x_1 \in L_1, x_2 \in L_2\}$, където

x_1 - дума;

x_2 - дума.

Операцията може да се представи чрез мрежите на двата езика, в които крайното място на първата се явява начално на втората.



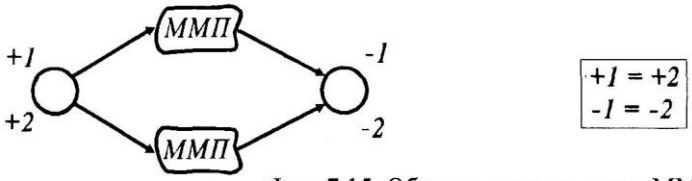
Фиг. 7.14. Конкатенация на места в ММП

2) Обединение:

Операцията представлява обединение на множествата думи на двата езика.

$L_1 \cup L_2 = \{x | x \in L_1 \in L_2\}$

Представена чрез ММП, операцията се реализира с обединяване на началните и крайни места на двете мрежи:



Фиг. 7.15. Обединение на места в ММП

3) Паралелност - || оператор за паралелност

Допуска се паралелност на преходите

"||" се дефинира за $\Sigma\{a, b\}$ и $x_1, x_2 \in \Sigma^*$, където x_1 и x_2 са думи от тази азбука.

$$ax_1 || bx_2 = a(x_1 || bx_2) + b(ax_1 || x_2)$$

Операторът паралелност за низ a и празната дума дава резултат $a || \Lambda = \Lambda || a = a$

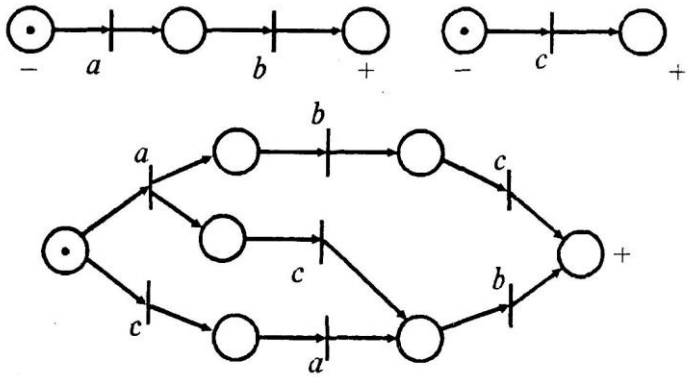
Пример 1:

Нека са дадени езиците:

$L_1 = \{ab\}$, $L_2 = \{c\}$

Резултантният език е следният:

$L_1 || L_2 = \{abc, acb, cab\}$



Фиг. 7.16. Обединяване на мрежи

От графичното представяне на мрежата се вижда, че двете независими последователни мрежи се обединяват в паралелна и като запазват самостоятелност, продължават да функционират. Всеки един генериран символ на езика може да се вмести в низовете на другия(вижда се от дефиницията за паралелност).

Пример 2:

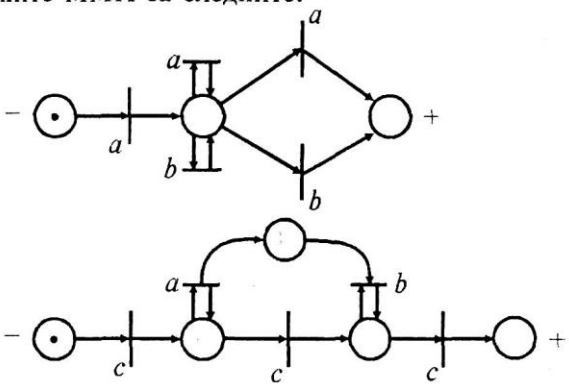
Нека са дадени азбуките:

$$\Sigma_1 = \{a, b\} \qquad \Sigma_2 = \{c\} \qquad x_1 = ab,\ x_2 = c$$

Върху тях са дефинирани езиците:

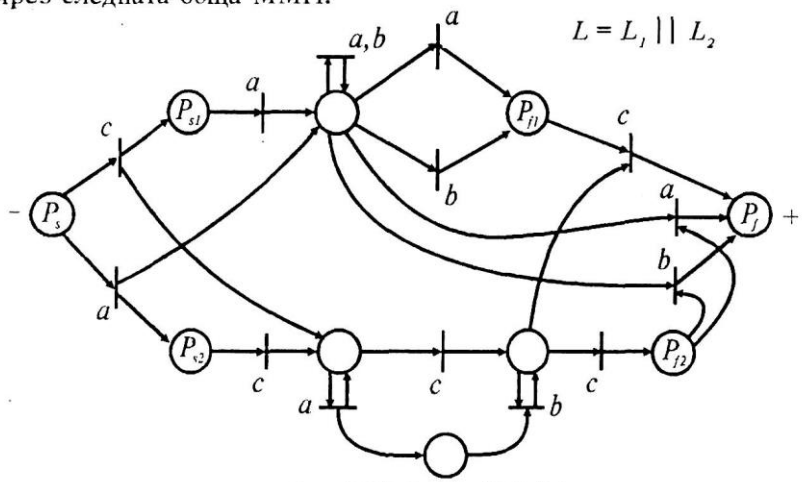
$$L_1 = a(a+b)^+ \qquad L_2 = ca^n cb^n c$$

Съответните ММП са следните:



Фиг. 7.17. Граф на ОММП

Резултатът от оператора паралелност може да се представи чрез следната обща ММП:



Фиг. 7.18. Обща ОММП

Пример 3:

Нека са дадени езиците:

$L_1 = \{ab\}$ $\qquad L_2 = \{c\}$

Резултатът от оператора паралелност е следното множество:

$$L_1 \,||\, L_2 = \{x_1 \,||\, x_2 \mid x_1 \in L_1, x_2 \in L_2\} =$$
$$= \{abc\ ,\ acb\ ,\ cab\}$$

4. Пресичане.

Операторът сечение дава в резултат множеството от общите низове:

$$L_1 \cap L_2 = \{x \mid x \in L_1, x \in L_2\}$$

Пример:

$$L_1 = \{a, b\}, \quad L_2 = \{a\} \Rightarrow L_1 \cap L_2 = \{a\}$$

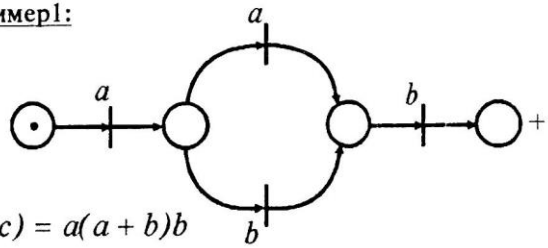
7.3.3. Формални и мрежови езици (МЕП).

1.Регулярни езици (от тип G_3).

Тези езици могат да се разпознават с ММП.

ММП генерират и разпознават всички регулярни езици. Такава мрежа се характеризира с 1 входно и 1 изходно място за всеки преход. Такава мрежа е еквивалентна на краен автомат.

Пример1:



$L(c) = a(a + b)b$

Фиг. 7.19. ОММП на регулярен език

2. Контекстно свободни езици (от тип G_2).

При ММП се използува качеството на мястото да бъде брояч на маркировки (броячен автомат, който не може да бъде стек, т.е. не може да запомня различни думи с различна дължина).

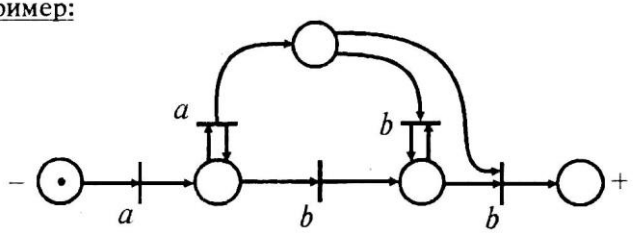
Следователно описателната мощност на ММП не достига за всички видове контекстно-свободни езици. Могат да се обработват ограничено подмножество на контекстно-свободните езици от вида:

$$X^i w Y^i \mid i \geq 0, w \in \Sigma^*,$$

където се използва връзката i,i .

Контекстно свободни езици от вида $w_1 w_2^n \mid n \geq 0, w_1 w_2 \in \Sigma^*$ не могат да се разпознават.

Пример:



Фиг. 7.20. ОММП на контекстно свободен език

Мрежата може да генерира и разпознава следния език:

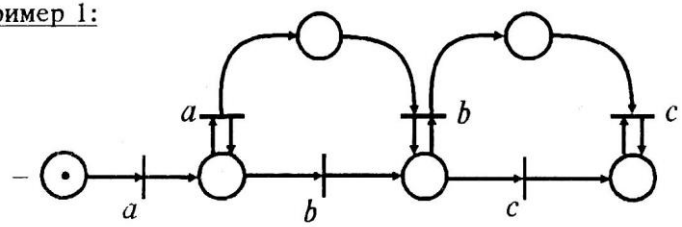
$$L(c) = \{a^n b^n \mid n > 1\}$$

3. Контекстно зависими езици (от тип G_1).

С ММП могат да се обработват и разпознават само някои контекстно зависими езици от вида:

$$X^i w_1 Y^i w_2 Z^i \mid w_1 w_2 \in \Sigma^*$$

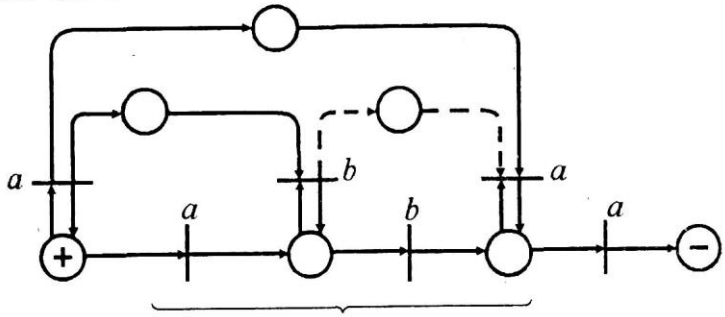
Пример 1:



Фиг. 7.21. ОММП на контекстно зависим език

$$L(c) = \{a^n b^n c^n \mid n > 0\}$$

Пример 2:



Фиг. 7.22. ОММП на контекстно зависим език

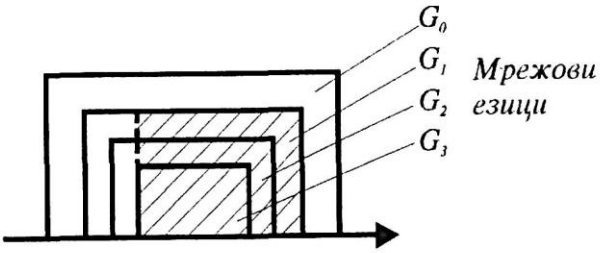
$$L(c) = \left\{a^n b^n c^n \mid n \geq 1\right\}$$

За всички контекстно зависими езици може да се използва машина на Тюринг в качеството на разпознавател.

Мрежовите езици не разпознават езици от тип 0, т.е. тяхната описателна мощност не е достатъчна.

4. Езици без ограничение се разпознават само от МТ.

Схема на типовете формални езици, които се разпознават от ММП.



Фиг. 7.23. Разпознаване на езици чрез ОМП

7.4. Моделиране на паралелни процеси с мрежи на Петри.

7.4.1. Видове отношения между зависимите паралелни процеси.

Зависимите паралелни процеси могат да имат сложни каузални зависимости помежду си, които могат да се декомпозират на следните елементарни каузални отношения:

1. Кооперираци отношения - при тях процесите работят паралелно като се синхронизират помежду си. В общ план те работят

върху обща задача като си поделят паралелните дейности.

2 Взаимно изключващи се (конкуриращи) отношения.

Процесите работят паралелно като взаимно се изключват (конкурират) в определени фази. В общ план те работят по различни (независими) задачи и се конкурират при използуване на общите ресурси.

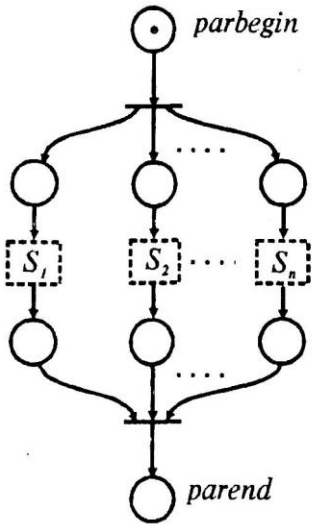
7.4.2. Синхронизиране и комуникация между паралелните процеси.

Съществуват следните две основни постановки:

1 постановка

Дадени са *n* части (фрагмента) от един процес, които започват в един и същ момент от време, но завършват в различни моменти от време. Процесът (задачата) може да се декомпозира на *n* части и да се изпълнява от *n* различни устройства. Устройствата, работещи паралелно във времето, не зависят едно от друго. Следващото действие започва непосредствено след завършване на всички части. Функцията, дефинирана за такива отношения, се нарича parbegin S_1; S_2;... S_n parend, а операцията - do in parallel.

Модел на тази постановка е следната ОММП:



Фиг. 7.24. Моделиране на паралелни процеси

II постановка

1. Синхронизират се две събития А и В от два паралелни процеса;

2. Събитията А и В не настъпват паралелно;

3. Събитието А настъпва относно В безусловно;

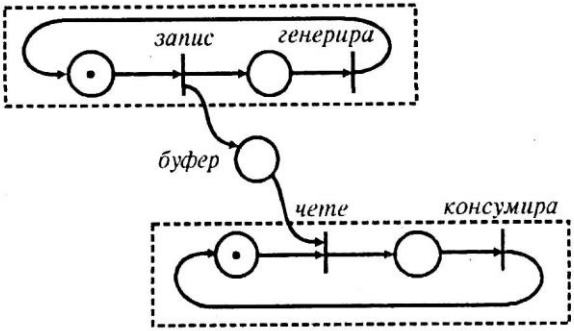
4. Събитието В настъпва само след събитието А.

Пример: Проблем "генератор - консуматор".

Същност на проблема:

Съществуват два процеса "Генератор на обекти", който генерира и записва в буфер и процес "Консуматор на обекти", който очаква появата на обект в буфера, извлича го и го консумира.

Модел на проблема представен чрез ММП.



Фиг. 7.25. Модел на "генератор-консуматор"

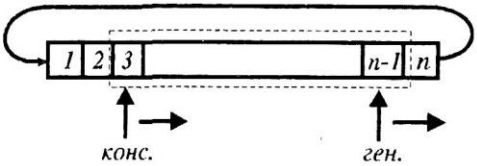
Буферът се представя чрез място в мрежата. Марката в мястото, определя наличието на обект в буфера, генериран от процеса "генератор", но все още не е консумиран от процеса "консуматор".

Вариант на този процес е проблем с много генератори (s) и много консуматори (t). Решението на проблема с ММП е чрез запис на начална маркировка с s марки. Консуматорите t се моделират с t марки в началното място на мрежата консуматор.

Скоростите на работа на двата процеса може да са различни.

Пример за такова взаимоотношение е връзката клавиатура - монитор. Клавиатурата е процес генератор, а мониторът- консуматор. Двата процеса работят асинхронно.

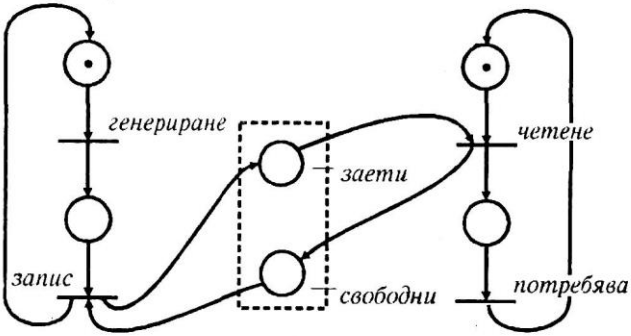
Друг вариант на същото отношение е кръгов буфер с n - позиции (места за обекти). В този случай се налагат ограничения върху броя на генерираните обекти. Процесът генериране спира при запълване на местата в буфера и очаква консуматорът да изконсумира обектите.



Генераторът извършва стъпка надясно + запис

Консуматорът четене + стъпка надясно

Моделът на кръгов буфер с n - места чрез ММП е следният:



Фиг. 7.26. Модел на кръгов Буфер

7.4.3. Координиране и разпределение на ресурси между ПП.

Критичен ресурс: ресурс, който във всеки момент от време се използва от не повече от един процес.

Типичен пример е проблемът "четене/запис" на общи данни от два паралелни процеса.

Пример:

Дадени са два процеса I и II.

I процес чете променливата (ресурс) x.

II процес чете променливата (ресурс) x.

I процес изчислява $x' = f(x)$

II процес изчислява $x'' = g(x)$

I процес записва $x' \rightarrow x$

II процес записва $x'' \rightarrow x'$

В случая възниква проблем с валидността на записаните данни. Резултатът от I процес се презаписва от II процес.

За решаване на този проблем се създава секция от програмата, която изпълнява функции за управление (разпределение) на съвместно използувания от няколко процеса ресурс. При това във всеки момент от време се разрешава само на един процес да променя данните, определящи на кои процеси са присвоени части от дадения ресурс.

В случаите, когато няколко асинхронни процеса трябва да координират своя достъп до критичния ресурс се използува управляем достъп.

Семафорът е едно от средствата за осигуряване на управляем достъп до критичен ресурс.

Семафорът е променлива, в която могат да се записват само цели положителни числа. Върху семафора са дефинирани операциите: *P(S)* - заемане или очакване.

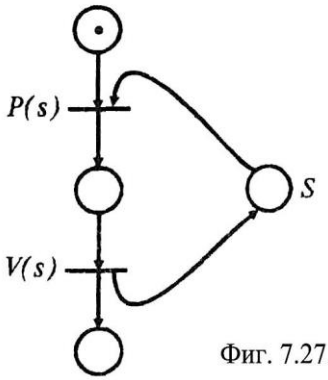
С тази операция се декрементира стойността на семафора. Възможна е само ако стойността е по-голяма от нула. Ако стойността е = 0, операцията *P(S)* чака, докато друг процес изпълни операцията *V(S)*.

V(S) - сигнализация (освобождаване).

С нея се инкрементира стойността на семафора.

Операциите *P* и *V* са дефинирани като примитиви, т.е. никакви други операции не могат да модифицират семафора.

Модел на семафор с ММП:



S е място в ММП.

Стойността на семафора се представя чрез броя марки в S.

Операция *P(S)* използува S като входно място.

Операция *V(S)* използува S като изходно място.

Фиг. 7.27. Модел на семафор

Критична секция

Част от процес, която се изпълнява без прекъсване от други процеси при условията на взаимно изключване. Първоначално се е считало, че критичната секция трябва да се изпълнява без никакви прекъсвания. В настоящия момент се поддържа мнението, че критичната секция не трябва да се прекъсва само от другите критични секции на дадения набор взаимно-изключващи се паралелни процеси.

Постановка на задачата за координиране и разпределение на ресурси.

1. Събитията, които не могат да настъпят едновременно и се изключват взаимно се наричат критични (t_1, t_2, t_3, t_4).

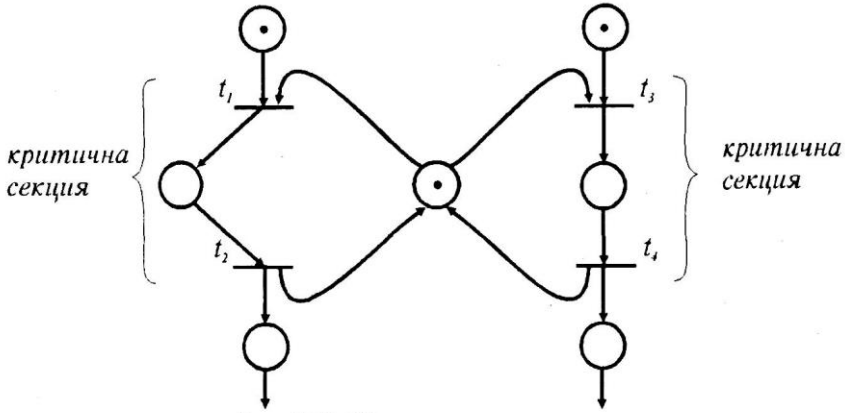
2. Ако $n \geq 2$ процесите съдържат критични едно към друго събития, те се конкурират и изискват координиране.

3. Такива процеси протичат асинхронно един спрямо друг и са времеинвариантни (независими от времето).

4. Координираните паралелни процеси не бива да създават блокада при настъпване на критични събития.

5. Интерпретация на координирането е класическата задача за управляемо разпределение на ресурси.

Модел на критична секция с ММП.



Фиг. 7.28. Модел на критична секция

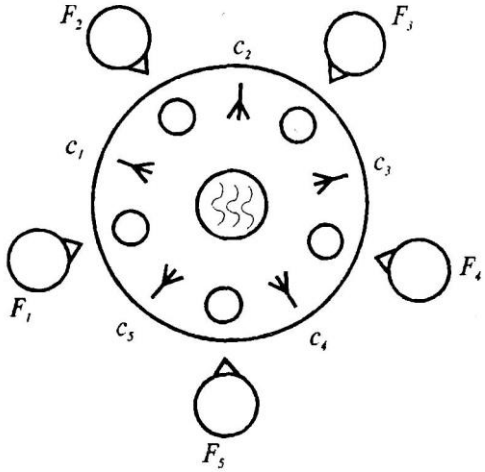
Критичната секция се представя чрез фрагмент от ММП, който участва в отношение на взаимно изключване с другите

фрагменти. Взаимно изключване се осигурява от общото за всички критични секции място, представящо чрез маркировка наличието или отсъствието на ресурс.

Пример: Обядващите философи

Постановка на задачата:

Дадена е маса и купа със спагети, 5 вилици ($c_1 ... c_5$) и 5 философа ($F_1 ... F_5$). За консумация на спагетите са необходими по 2 вилици. Приема се, че всеки философ може да взема вилици отляво и отдясно на мястото, където седи. Приема се, че всеки философ извършва две алтернативни дейности - той или яде, или мисли.

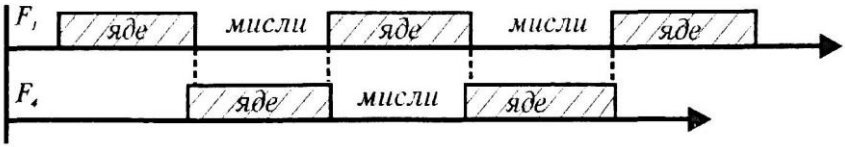


Фиг. 7.29. Модел на обядващите философи

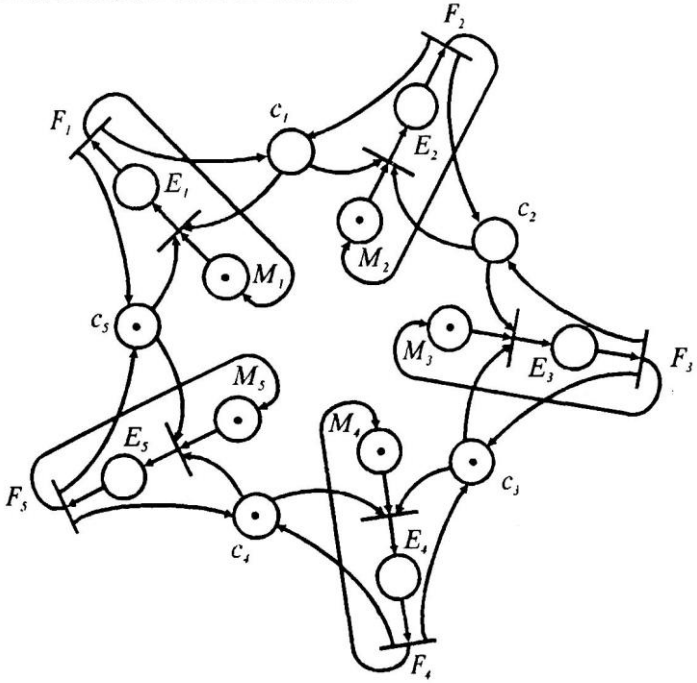
Блокада се създава при:

1. В случай, когато всички сядат едновременно и всеки взема лява (дясна) вилица случай на пълна блокада.

2. F_1 и F_4 се наговарят да ядат последователно и да не разрешават на F_5 достъп до две вилици едновременно. В този случай F_1 и F_4 циклично и със застъпване във времето използват вилиците и предизвикват частична блокада:



Модел на проблема с ММП.



Фиг. 7.30. Модел на задачата чрез ОММП

8. Абстрактни информационни структури.

В този раздел се разглежда как отделните средства за моделиране, които са разгледани в предходните раздели, се представят в паметта на компютрите чрез структури от данни.

За създаване на надеждно апаратно и програмно осигуряване може да се използва следната последователност:

1. Чрез анализ на заданието се избира подходящо средство за

моделиране (описание поведението на системата) с необходимата описателна (изчислителна) мощност.

2. Съставя се адекватен математически модел и се проверява съответствието му с реалната моделирана система.

3. Създаването на такъв модел е предпоставка за създаването на апаратно или програмно решение. Тъй като моделът е създаден чрез формално средство, извършва се формално преминаване към реализация чрез конкретно техническо средство.

Реализация.

1. Апаратна - за модели с малка изчислителна мощност: крайни автомати или прости мрежи.

2. Програмна - всички разгледани модели.

3. Смесена - бързите - апаратно, а бавните - програмно.

Програмната реализация може да се декомпозира на две части.

1. Представяне на моделите в паметта чрез абстрактни информационни структури.

2. Алгоритмична реализация на операциите и действията с тях.

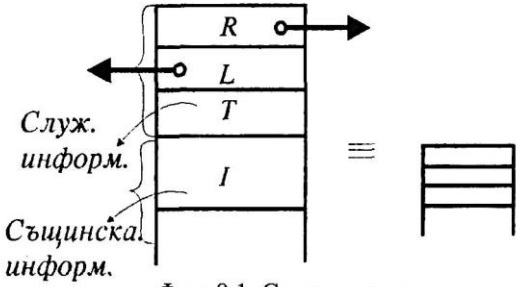
8.1. Йерархия на информационните структури (ИС).

В зависимост от степента на абстракция ИС в ЕИМ се класифицират на:

1. Физически формати;
2. Прости данни;
3. Структурни данни;

Всеки клас се включва като елемент на по-горен клас.

За представяне на информационни възли се използват структури от тип запис:



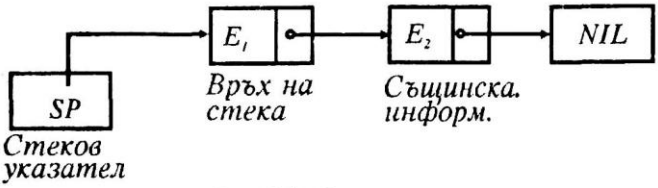
Фиг. 8.1. Структура запис

$A.L$
$A.R$ } - указатели към други възли

$A.T$ - тип на възела и друга служебна информация

$A.I$ - същинска информация

Пример: Стек



Фиг. 8.2. Структура стек

В стека могат да се въвеждат елементи, да се извеждат елементи от началото към края по някаква информация.

Ако се премахне елемент от върха от стека, то със стековия указател трябва да се прочете адресът на следващият възел:

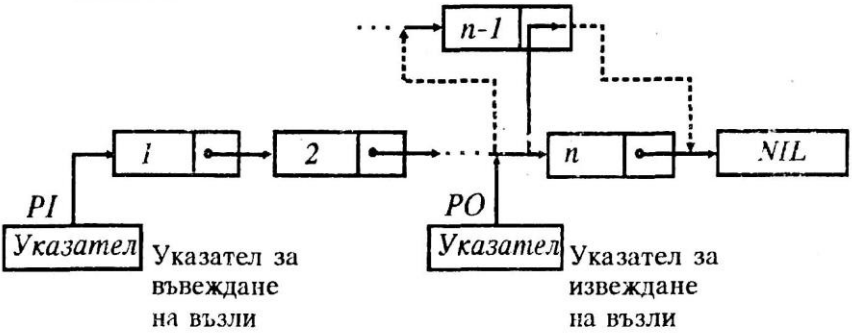
Ако $SP = \Lambda$ - неуспех, в противен случай:

$P := SP,\ SP := L(P),\ X := I(P),\ DELETE\ (P)$

Въвеждане на нов елемент в стека включва следните действия:

$CREATE(P),\ I(P) := X,\ L(P) := SP,\ SP := P$

Пример: Опашка (*FIFO*)

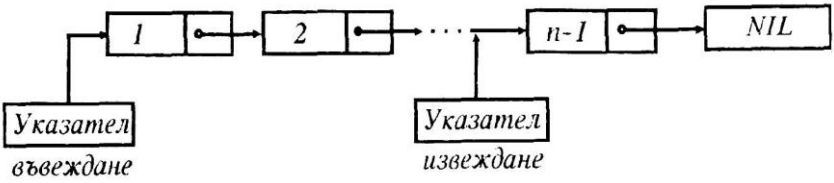


Фиг. 8.3. Структура опашка

Операции:

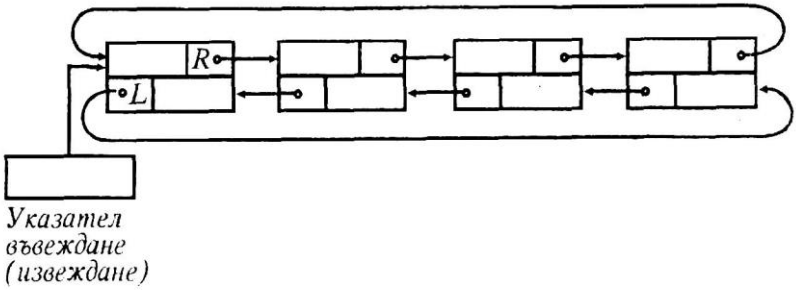
$CREATE(P),\ I(P):=X,\ L(P):=PI,\ PI:=P$

Управлението се извършва по два указателя. С тяхна помощ се извършва разкъсване на веригата.



Фиг. 8.4. Разкъсване на верига

Единично (двойно свързан буфер) представлява друг вид опашка с по-сложно управление:



Фиг. 8.5. Структура буфер

$R(X) \dashrightarrow R(L(X))$

$L(X) \dashrightarrow L(R(X))$

Конкурентни структури.

Абстрактни структури, при които достъпът чрез функции е гарантиран относно взаимно изключване на паралелни процеси при работа с тях.

Такава е концепцията на т.н. "Монитор" на езика CONCURRENT PASCAL.

Има елементи, които осигуряват паралелни процеси.

Достъпът до тях се осъществява чрез вграден механизъм за въвеждане и извеждане.

Функциите работят в режим на "критична секция" за паралелни процеси.

8.2. Представяне на моделите в паметта.

8.2.1. Представяне на множества.

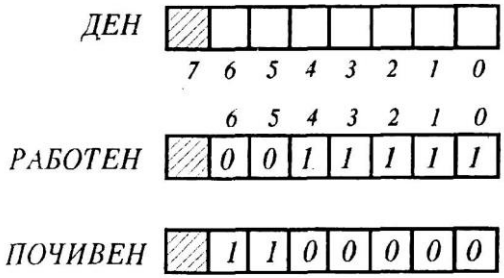
Нека е дадено множеството: $A=\{a,b,c...\}$.

Представянето е възможно чрез битове и низове, като на всеки потенциален член от множеството съответства 1 бит. Ако битът е 0, елементът не е член на това множество, ако е 1 - елементът е член на множеството.

Неформален пример на езика Паскал.

```
type
        седмица = (ПН, ВТ, СР, ЧТ, ПТ, СБ, НД)
        ден  Set of  седмица
var
        работен, почивен: ден
Присвояявания:
        РАБОТЕН: = {ПН, ВТ, СР, ЧТ, ПТ}
        ПОЧИВЕН: = {СБ, НД}
```

В паметта представянето е чрез 7 бита от байт:



Фиг. 8.6. Представяне на множества

Операциите между множествата могат да се реализират чрез битови логически операции, като се използват два байта.

Например:

РАБОТЕН $\cup$ ПОЧИВЕН = ДЕН

РАБОТЕН $\cap$ ПОЧИВЕН = 0

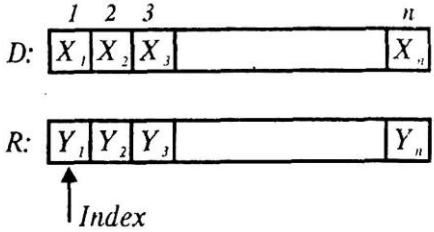
8.2.2.Отношения и функции.

По дефиниция те представляват множество от наредени двойки

$$r = \{\langle X, Y\rangle | X \in D, Y \in R\}$$

Такива отношения и функции могат да се представят чрез два едномерни масива D и R, които се обработват с общ индекс.

Табличен начин на представяне:



Фиг. 8.7. Представяне на отношения

8.2.3. Представяне на граматики.

Граматиките се представят чрез представяне на техните разпознаващи (генериращи) автомати или синтактични графи като списъчни структури.

8.2.4. Представяне на автомати.

8.2.4.1. Представяне на крайни автомати:

Може да се използва представяне на крайните ориентирани графи на КА като списъчни структури.

Пример:

Даден е регулярният израз $R = \left(a(a+b)^{+}c\right)^{*}$ на езикз L.

Да се извърши представяне чрез граматика, КОГ и списъчна структура.

Решение:

1. $G = \{\Sigma, N, P, S\}$

$\sum = \{a, b, c\}$

$N = \{A, B, C\}$

$S =\geq \{A\}$

а) контекстно-свободна граматика

$P_1 : L \to aKc$

$P_2 : K \to M$

$P_3 : K \to KM$

$P_4 : M \to a$

$P_5 : M \to b$

б) регулярна граматика

$P_1 : A \to aB$

$P_2 : B \to aC$

$P_3 : B \to bc$

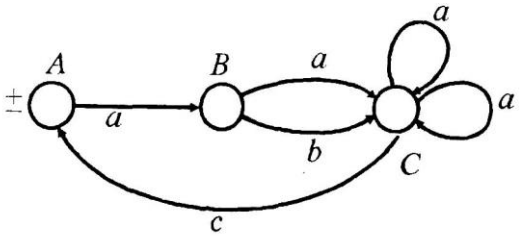
$P_4 : C \to aC$

$P_5 : C \to bC$

$P_6 : C \to cA$

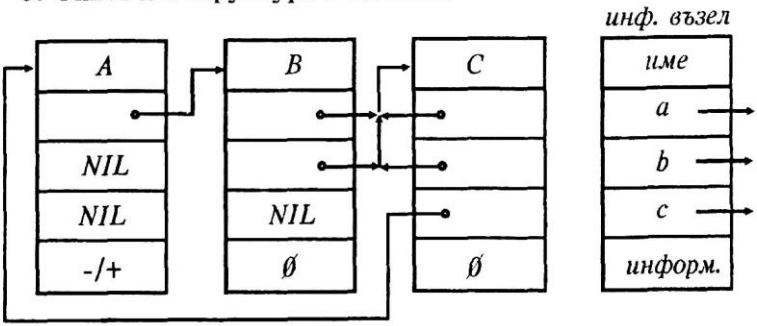
$P_7 : A \to \Lambda$

2. КОГ



Фиг. 8.8. КОГ за езика

3. Списъчна структура в паметта:



Фиг. 8.9. Списъчна структура

Всеки информационен възел се състои от име и толкова указатели, колкото са буквите от азбуката на езика и информация за типа на възела.

8.2.4.2. Представяне на стеков автомат.

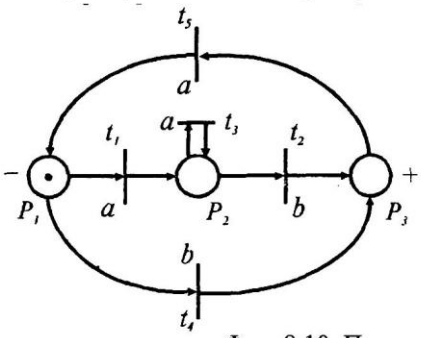
Стековият автомат се представя чрез представяне на краен автомат и списъчната структура стек:

КОГ + СТЕК = стеков автомат

8.2.4.3. Представяне на мрежи на Петри

Мрежите се представят както КОГ като допълнително се означава, кои информационни възли са места, и кои преходи. Входните и изходни места се указват във възела на прехода и съдържат информация за текущия брой маркировки.

При мрежата на Петри чрез списъчни структури се представят два вида възли (преходи и места).



Пример:

$C = (P, T, I, O)$

$M = (C, \mu) \qquad \mu = \langle 1,0,0 \rangle$

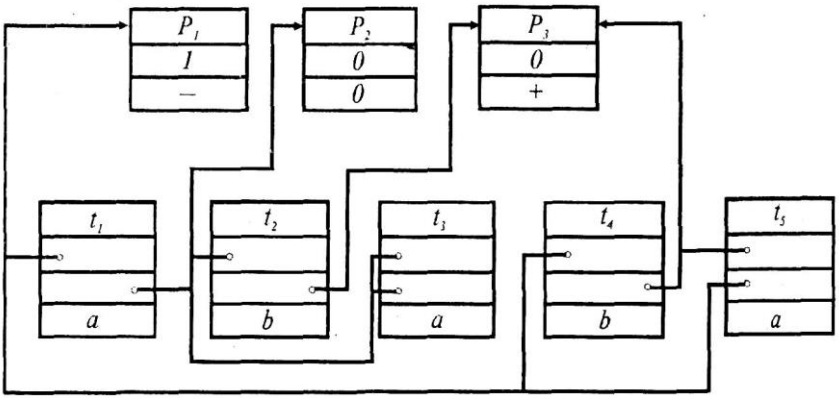
$I(t_1) = \{P_1\} \quad I(t_3) = \{P_2\}$

$O(t_1) = \{P_2\} \quad O(t_3) = \{P_2\}$

Фиг. 8.10. Представяне чрез ОММП

$$I(t_2) = \{P_2\} \quad I(t_4) = \{P_1\}$$
$$O(t_4) = \{P_3\}$$
$$O(t_2) = \{P_3\} \quad I(t_5) = \{P_3\}$$
$$O(t_5) = \{P_1\}$$

Списъчна структура в паметта:



Фиг. 8.11. Списъчна структура

Всяко място се представя чрез запис, съдържащ име на мястото, брой марки и тип на мястото.

Преходите се представят чрез запис, съдържащ име на прехода, указатели към входни места, чиито брой се определя по $max|I(t_i)|$ за всички t_i, указатели към изходни места - $max|O(t_i)|$ и разпознаван символ.

9. Математическа логика.

9.1. Предмет на математическата логика.

Предмет на математическата логика е анализът на методите за разсъждения. Математическата логика не се занимава с доводите на конкретните съждения, а с тяхната логическа схема.

Примери:

1. Всички хора са смъртни
Аз съм човек } ⇒ Аз съм смъртен.

2. Всяко куче гони коя да е котка
Шаро е куче, Маца - котка } ⇒ Шаро гони Маца

Обща схема на примерите:

Всички А се отнасят към В
С е един от А } ⇒ С се отнася към В

Всички А имат свойство В
С е един от А } ⇒ С притежава свойство В

Конкретните твърдения нямат отношение към формата на тяхното описание.

Определянето на истинността на отделните твърдения не е задача на математическата логика. Тя определя дали дадено разсъждение (логическа верига, доказателство) е правилно, независимо дали е истина.

Съществуват парадокси например, когато логическата верига е правилна, но води към противоречие.

Пример:

"Аз лъжа": Аз лъжа по принцип или аз лъжа в момента - смесване на момента на изказване със смисъла на изказването.

9.2. Препозиционна логика (съждително смятане).

9.2.1. Съждение - елементарно твърдение с известна истинност, която се определя от човека и не подлежи на по-нататъшен анализ.

Смисълът на съжденията се игнорира и те се бележат с А, В, С.

Предмет на съждителната логика

Съждителната логика се занимава с методите за установяване истинността на сложни съждения (лог. изрази) независимо от истинността на елементарните съждения.

9.2.2. Език на съждителната алгебра

Азбуката на съждителната логика включва три множества:

$$\Sigma = G_1 \cup G_2 \cup G_3$$

$G_1 = \{A_0, A_1, \ldots, A_n\}$ - множество от символи

$G_2 = \{\ , \wedge, \vee, \rightarrow, \leftrightarrow\}$ - множество на логически съюзи

$\bar{}$	, $\neg$	логическо отрицание
$\wedge$	, &	логическо "и"
$\vee$	, |	логическо "или"
$\leftrightarrow$	, $\equiv$	логическа еквивалентност
$\rightarrow$	, $\supset$	логическа имиликация ("следва")

$G_3 = \{(\,,\,)\,,\,,\}$ - множество на спомагателните символи

9.2.3. Формули в съждителната алгебра (правилно построени формули ППФ).

Това са синтактически правилно (построени) записани логически изрази, които се образуват по следните правила:

1. Съждителните променливи са ППФ.
2. Ако P и Q са ППФ, то

$\overline{P}, (P \vee Q), (P \wedge Q), (P \rightarrow Q), (P \leftrightarrow Q)$ са ППФ

Пример:

$$((P \rightarrow Q) \vee (P \rightarrow (Q \wedge P)))$$

Таблица на истинност

Таблица 9.1. Таблица за истинност

					$\overline{P} \vee Q$	$(P \wedge Q) \vee (\overline{P} \wedge \overline{Q})$
P	Q	$\overline{P}$	$P \wedge Q$	$P \vee Q$	$P \to Q$	$P \leftrightarrow Q$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Формули за преобразуване на ППФ.

Използват се следните обозначения

T (*true*, "1") F (*false*, "0") за $(\to)$:

$$T \to B \equiv B \qquad B \to T \equiv T$$

$$F \to B \equiv T \qquad B \to F \equiv \overline{B}$$

$$B \to B \equiv T \qquad \overline{B} \to B \equiv B$$

$$\overline{\overline{B}} \equiv B$$

за $(\vee)$: $\quad T \vee B \equiv T, \quad B \vee B \equiv B$

за $(\wedge)$: $\quad T \wedge B \equiv B, \quad B \wedge B \equiv B$

за $(\leftrightarrow)$: както и за $\oplus$ (сума по модул 2)

$$T \leftrightarrow B \equiv B \quad B \leftrightarrow T \equiv B \quad F \leftrightarrow B \equiv \overline{B} \quad B \leftrightarrow F \equiv \overline{B}$$

$$B \leftrightarrow B \equiv T \quad B \leftrightarrow \overline{B} \equiv F \quad \overline{B} \leftrightarrow B \equiv F$$

Помощни формули

$$A \to B \equiv \overline{A} \vee B \qquad A \to B \equiv \overline{A \wedge \overline{B}}$$

$$A \leftrightarrow B \equiv (A \to B) \wedge (B \to A)$$

$$\overline{A \to B} \equiv A \wedge \overline{B}$$

$$\overline{A \leftrightarrow B} \equiv \overline{(A \to B) \wedge (B \to A)} = \overline{(A \to B)} \vee \overline{(B \to A)} =$$

$$= (A \wedge \overline{B}) \vee (B \wedge \overline{A})$$

$$A \to (B \to C) \equiv (A \wedge B) \to C$$
$$A \to (B \to C) \equiv B \to (A \to C)$$

$$A \leftrightarrow B \equiv (\overline{A} \vee B) \wedge (A \vee \overline{B})$$
$$A \leftrightarrow B \equiv (A \wedge \overline{B}) \vee (\overline{A} \wedge \overline{B})$$

$$A \leftrightarrow B \equiv \overline{A} \leftrightarrow \overline{B} \qquad A \to B \equiv \overline{B} \to \overline{A}$$

Закони на Де Морган

$$\overline{A \vee B} = \overline{A} \wedge \overline{B}$$
$$\overline{A \wedge B} = \overline{A} \vee \overline{B}$$

Закон на Шепон

$$\overline{\overline{A} \vee B \wedge \overline{C}} = \overline{A} \vee B \vee C$$

Дистрибутивни закони

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$$
$$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$$

Комутативни закони

$$A \leftrightarrow B \equiv B \leftrightarrow A$$

Закони за съкращаване (поглъщане)

$$A \wedge (\overline{A} \vee B) = A \wedge B$$
$$A \vee (\overline{A} \wedge B) = A \vee B$$

$$A \wedge (A \vee B) = A$$
$$A \vee (A \wedge B) = A$$

Асоциативни закони

$$(A \leftrightarrow B) \leftrightarrow C \equiv A \leftrightarrow (B \leftrightarrow C)$$

Пример:

Да се докаже, че правилно построената формула е общовалида (еквивалентна е на *true*)

$$\big(((P \wedge Q) \rightarrow R) \wedge (P \rightarrow Q)\big) \rightarrow (P \rightarrow R) \equiv T$$

Решение:

Премахва се импликацията и еквивалентността, след което се разчленява инверсията, прилагат се законите на съждителната логика до доказване на общовалидност:

$$\overline{\big(((\overline{P} \vee \overline{Q} \vee R) \wedge (\overline{P} \vee Q)\big)} \vee \overline{P} \vee R \equiv T$$

$$\overline{\overline{P} \vee \overline{Q} \vee R} \vee \overline{\overline{P} \vee Q} \vee \overline{P} \vee R \equiv T$$

$$\overline{\overline{P} \vee \overline{Q}} \wedge \overline{R} \vee P \wedge \overline{Q} \vee \overline{P} \vee R \equiv T$$

$$P \wedge Q \wedge \overline{R} \vee P \wedge \overline{Q} \vee \overline{P} \vee R \equiv T$$

$$P \wedge Q \wedge \overline{R} \vee \overline{P} \vee \overline{Q} \vee R \equiv T$$

$$P \wedge Q \wedge \overline{R} \vee \overline{P} \vee \overline{Q} \vee R \equiv T$$

$$Q \wedge \overline{R} \vee \overline{P} \vee \overline{Q} \vee R \equiv T$$

$$\overline{R} \vee \overline{P} \vee \overline{Q} \vee R \equiv T$$

9.2.4. Изчисления със съждения.

Естествена формална система (ЕФС).

Крайно множество от общовалидни ППФ (аксиоми) и крайно множество от съответствия между общовалидни ППФ (правила за извод) се наричат естествена формална система.

Една ППФ А е изводима (доказуема, удовлетворима или общовалидна) в рамките на ЕФС, ако може да бъде образувана крайна редица, наречена доказателство на А, от общовалидни ППФ, в която А е последна, а всички предходни са или аксиоми, или са изведени чрез правилата за извод.

Под изчисления със съждения се разбира извършване на доказателство за общовалидност.

Секвенция от ППФ (означена с Г)

Секвенцията е поредица от ППФ, свързани с логически съюз "и":

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \to B \equiv \underbrace{A_1, A_2, A_3, \ldots, A_n}_{\Gamma} \to B \equiv \Gamma \to B,$$

която се счита за предпоставена и от която следва ППФ В. Редът на предпоставките не е от значение. Съществуват следните три основни форми:

1. $\underbrace{A_1, A_2, A_3, \ldots, A_n}_{\Gamma} \to B$ (секвенция Γ е предпоставка за извода на B)

2. $\to B$ (B е аксиома, винаги е вярно, без предпоставки)

3. $\underbrace{A_1, A_2, A_3, \ldots, A_n}_{\Gamma} \to$ (секвенция е противоречива, от нея не следва нищо).

Правило за извод на ППФ

$$\left.\frac{\Gamma_1, \Gamma_2, \ldots, \Gamma_n}{\Gamma}\right\} \qquad \Gamma_i, \Gamma \text{ - произволни секвенции}$$

Γ може да бъде изведено от Γ_1 до Γ_n.

Доказателство на ППФ A_n.

ППФ се доказва чрез съставяне на такава последователност от преобразувания, при които се извежда ППФ:

$$\Gamma_1 \Rightarrow A_1, \Gamma_2 \Rightarrow A_2, \ldots, \Gamma_n \Rightarrow A_n.$$

Всяко от горните правила:

$$\Gamma_i \Rightarrow A_i,$$

може да бъде или е аксиома, или е изводимо чрез правила за извод.

9.2.5. Аксиоми и правила в съждителната алгебра.

Аксиома за предпоставките

$$\rightarrow \Gamma, A \Rightarrow A$$

от секвенция Γ и предпоставка A следва, че предпоставката A е винаги истина.

въвеждане на предпоставка:

$$\frac{\Gamma \Rightarrow B}{\Gamma, A \Rightarrow B}$$

Ако е дадена секвенция Γ, която представлява предпоставка за B, то от добавянето на някаква друга предпоставка A, следствието не се променя, $\Gamma \Rightarrow B$.

елиминиране на предпоставка:

$$\frac{(\Gamma, A \Rightarrow B) \wedge (\Gamma, \overline{A} \Rightarrow B)}{\Gamma \Rightarrow B}$$

$\Gamma \Rightarrow B$ ако е вярно, че от секвенция Γ и предпоставка A следва B, и че от Γ и $\overline{A} \Rightarrow B$, то от Γ следва B, A е излишна предпоствака и може да се премахне: $\Gamma \Rightarrow B$.

Аксиома за истинност

$$\Gamma \Rightarrow T \qquad \Gamma \Rightarrow \overline{F} \qquad A \Rightarrow A$$

секвенция от предпоставки е истина.

Други аксиоми

$$((A \wedge B) \rightarrow A) \qquad ((A \wedge B) \rightarrow B)$$

$$(A \rightarrow (A \vee B)) \qquad (B \rightarrow (A \vee B))$$

$$(A \rightarrow \overline{B}) \rightarrow (B \rightarrow \overline{A})$$

$$(\overline{A} \rightarrow A)$$

Правила за извод

Правила за $\vee$

$$\frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A \vee B} \qquad \frac{\Gamma \Rightarrow B}{\Gamma \Rightarrow A \vee B}$$

1. въвеждане на $\vee$

$$\frac{(\Gamma, A \Rightarrow C) \wedge (\Gamma, B \Rightarrow C) \wedge (\Gamma \Rightarrow A \vee B)}{\Gamma \Rightarrow C}$$

2. елиминиране на $\vee$

Правила за $\wedge$

$$\frac{(\Gamma \Rightarrow A) \wedge (\Gamma \Rightarrow B)}{\Gamma \Rightarrow A \wedge B}$$

1. въвеждане на $\wedge$

$$\frac{\Gamma \Rightarrow A \wedge B}{\Gamma \Rightarrow A} \qquad \frac{\Gamma \Rightarrow A \wedge B}{\Gamma \Rightarrow B}$$

2. елиминиране на $\wedge$

Правила за $\rightarrow$ (импликация)

$$\frac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow A \rightarrow}$$

1. въвеждане $\rightarrow$

$$\frac{(\Gamma \Rightarrow A) \wedge (\Gamma \Rightarrow A \rightarrow B)}{\Gamma \Rightarrow B}$$

2. елиминиране $\rightarrow$

Правила за инверсия

$$\frac{(\Gamma, A \Rightarrow B) \wedge (\Gamma, A \Rightarrow \overline{B})}{\Gamma \Rightarrow \overline{A}}$$

1. въвеждане на - (правило на абсурда)

$$\frac{(\Gamma \Rightarrow A) \wedge (\Gamma \Rightarrow \overline{A})}{\Gamma \Rightarrow B}$$

2. елиминиране на - (правило на противоречието, от Γ не следва нищо $\neq A$)

Правила на двойното отрицание $=$

$$\frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow \overline{\overline{A}}} \qquad \frac{\Gamma \Rightarrow \overline{\overline{A}}}{\Gamma \Rightarrow A}$$

въвеждане, елиминиране

Правила за еквивалентност $\leftrightarrow$

$$\frac{(\Gamma \Rightarrow A \to B) \wedge (\Gamma \Rightarrow B \to A)}{\Gamma \Rightarrow A \leftrightarrow B}$$

1. въвеждане

$$\frac{\Gamma \Rightarrow A \leftrightarrow B}{\Gamma \Rightarrow A \to B} \qquad \frac{\Gamma \Rightarrow A \leftrightarrow B}{\Gamma \Rightarrow B \to A}$$

2. елиминиране

Пример 1. Да се изведе:

$$\frac{\Gamma \Rightarrow A \to (B \to C)}{\Gamma \Rightarrow (A \wedge B) \to C}$$

1. $\Gamma \Rightarrow A \to (B \to C)$
2. $\Gamma, A \wedge B \Rightarrow A \to (B \to C)$
3. $\Gamma, A \wedge B \Rightarrow A \wedge B$
4. $\Gamma, A \wedge B \Rightarrow A$
5. $\Gamma, A \wedge B \Rightarrow B \to C$
6. $\Gamma, A \wedge B \Rightarrow B$
7. $\Gamma, A \wedge B \Rightarrow C$
8. $\Gamma \Rightarrow (A \wedge B) \to C$

Пример2. Да се докаже:

$$\Rightarrow \big((P \wedge Q \to R) \wedge (P \to Q)\big) \to P \to R$$

1. $P \wedge Q \to R, \quad R \to Q, \quad P \Rightarrow P$
2. $P \wedge Q \to R, \quad R \to Q, \quad P \Rightarrow P \to Q$
3. $P \wedge Q \to R, \quad P \to Q, \quad P \Rightarrow Q$

4. $P \wedge Q \to R, \quad P \to Q, \quad \Rightarrow P \wedge Q \to R$

5. $P \wedge Q \to R, \quad P \to Q, \quad P \Rightarrow P \wedge Q$

6. $P \wedge Q \to R, \quad P \to Q, \quad P \Rightarrow R$

7. $P \wedge Q \to R, \quad P \to Q \Rightarrow P \to R$

8. $\Rightarrow \big((P \wedge Q \to R) \wedge (P \to Q)\big) \to P \to R$

9.3. Предикатна логика.

9.3.1. Същност на предикатната логика (ПЛ)

Примери:

1. Всички хора са млекопитаещи

Следователно, главата на човека е глава на млекопитаещо.

2. Всеки приятел на Иван е приятел на Петър.

Никола е приятел на Иван, следователно Никола е приятел и на Петър.

Верността на разсъжденията не зависи само от верността на отделните твърдения, а и от приложената логическа схема.

Предикат. Същност.

Предикатът е твърдение (изказване), което:

1. Съдържа поне една логическа променлива.
2. Логическата променлив е с неизвестна истинност.

Предикатът изразява отношението между обекти или техните свойства.

Истинността на предиката зависи от текущите стойности на променливите.

$(x < MIN) \wedge (x$ е четно$) \to$ твърдение

Ако се приложи стойност на $x = 3$ и $MIN = 6$ се получава съждението:

$(3 < 6) \wedge (3$ е четно$) \to$ съждение

То може да бъде изчислено.

$A \wedge B \to F$

Числото $x = 3$ е по-малко от MIN, но е нечетно.

Предикатът P от n променливи представлява n аргументна предикатна функция:

$$P(x_1, x_2, \ldots, x_n) \}$$

Пример:

по-голямо(X, Y) - тълкува се като $X > Y$

Операторният и предикатният запис са еквивалентни : $X > Y \leftrightarrow >(X, Y)$

$x_i \in D$ дефиниционна област

За определяне D се използват квантори:

$$\begin{cases} \exists x - Existenz & \text{(за съществуване)} \\ \forall x - All & \text{(за общност)} \end{cases}$$

Квантор за съществуване, бележи се с $\exists x$ и се разбира като:

Съществува поне едно X, за което предикатът $p(x)$ е истинен.

Пример:

$$\exists x p(x) \leftrightarrow \exists x(p(x)) \leftrightarrow (\exists x)(p(x)) \leftrightarrow T$$

Квантор на общност, бележи се с $\forall x$ и се разбира като:

За всички възможни X от дефинираната област, предикатът p е истинен.

Пример:

$$\forall x(p(x)) \leftrightarrow \forall x\, p(x) \leftrightarrow (\forall x)(p(x)) \leftrightarrow T$$

Квантифицирани (свързани) променливи. Такива са променливите, които се намират в областта на действие на кванторите.

$\exists X p(X, Y) \rightarrow T$ $\quad$ Y е свободна, X - свързана

Пример:

$\forall X$(приятел (X, Иван)) $\rightarrow$ приятел (X, Петър)

1. всеки, който е приятел на Иван, е приятел и на Петър

2. от приятел (Никола, Иван) следва приятел (Никола, Петър)

9.3.2. Език на предикатната логика.

$A_i^n(i \geq 1, n \geq 0)$

n - брой на аргументите, които се отнасят към A

i - индекси, по които се отличават променливите A

$A,\ B,\ A_1,\ A_2,\ A(x),\ A_2(X,Y)$

$\downarrow \qquad \downarrow \qquad \downarrow$

на нулева на първа на втора (A_2^2 - по брой аргуметни)

Използва се азбуката $\Sigma = G_1 \cup G_2 \cup G_3 \cup G_4 \cup G_5 \cup_6$

За описване на предикатната логика се използват G_i - 6-те множества.

$$\boxed{G_1 = \{f_i^n\}}$$

n - аргументни константни функции;

$f_i^0 = a_i$ - индивидна константа.

Пример:

$G_1 = \{a_1, a_2, f_2(\ldots), sin(x), \ldots\}$

Константна функция означава, че е постоянно името на функцията, а не стойностите на аргументите.

$$\boxed{G_2 = \{p_i^n\}}$$

Множество от n - местни кностантни предикати

$p_i^0 = p_i$ - съждителна константа.

Пример:

$G_2 = \{p,\ T,\ F,\ p(1,2),\ зелено\ (x)\ldots\}$

Константен предикат се отнася за името на предиката, а не за стойностите на аргументите.

n - местните функции дават като резултат аритметична стойност, а n-аргументните константни предикати - логическа стойност.

$$\boxed{G_3 = \{F_i^n\}}$$

Множеството от n - аргументни функции (името на функцията може да бъде променлива)

$F_i^0 = x_i$ - индивидна променлива.

Пример:

$G_3 = \{X_1, X_2, Y, F_2(\ldots), \ldots\}$,

Променливите се пишат с главна буква.

$\boxed{G_4 = \{P_i^n\}}$

Множеството от n - местни предикати

$P_i^0 = p_i$ - съждителна променлива

Пример:

$G_4 = \{P_1, P_2, Q, P_2(\ldots), \ldots\}$

За множествата G_3, G_4 обектите се делят на функции и предикати, при което аргументите и имената могат да са променливи.

Функциите дават винаги аритметична стойност, а предикатите дават логическа стойност.

$\boxed{G_5 = \{^-, \wedge, \vee, \rightarrow, \leftrightarrow, \exists, \forall\}}$ - множество на логическите съюзи

$\boxed{G_6 = \{'(', ')', ',', '\}}$ - помощни символи

9.3.3. Правила за построяване на правилно построени формули (ППФ).

ППФ се изгражда от три класа изрази:

1. Терми *(t)* - изрази, даващи постоянна или променлива аритметична стойност.

а) всяка индивидна константа и всяка индивидна променлива са терми. a_i и X_i са терми.

б) ако $t_1, t_2, \ldots, t_n$ са терми, то и $f_i^n(t_1, t_2, \ldots, t_n)$, и $F_i^n(t_1, t_2, \ldots, t_n)$ са терми.

Пример:

$3-16;\ X_1;\ Y;\ t(3,4),\ 3+4,\ F(X,2),\ X*(4-2)$ - както и всеки аритметичен израз.

2. Атоми (атомни форми) (аф) - изрази, даващи постоянна или променлива логическа стойност.

а) T и F са атомирани форми

б) всички съждителни константи и всички съждителни променливи са атомни форми.

Пример:

$T, F, A, B > (4,3), 4 > 3, X \leq (Y+6)$, зелено (трева), $P(x)$

Термите участват при изграждането на атомите.

3. ППФ (логически изрази, изградени от АФ, при което резултатът е винаги логическа стойност.

а) всяка аф е ППФ.

б) ако A и B са ППФ, то

$\overline{A}, (A \cup B), (A \cap B), (A \to B), (A \leftrightarrow B)$ са ППФ

в) ако X е променлива (съждителна или индивидна) и A е ППФ, то $(\forall x)A(x)$ и $\exists x\, A(x)$ са ППФ.

Съждителна логика

В съждителната логика са допустими само съждителни променливи, т.е. само съжденията P_i^0.

Предикатна логика от I ред

В предикатната логика от I ред се допускат:

1) n - местни константни функции $f_i^n, f_i^0 = a_i$

2) n - местни константни предикати $p_i^n, p_i^0 = p_i$

3) индивидни променливи X_i

Пример:

Да се анализира структурата на ППФ:

$$\forall \underline{X}\Big(\big(f(\underline{a}) > (\underline{X} + g(\underline{4}, \underline{X}))\big) \wedge \exists \underline{Y}\big(p(\underline{Y}, \underline{X})\big) \to q \vee r(\underline{Y})\Big)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{t}{t}}{t}}{t}}{t}}{t}}{\text{аф}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{t}{t}}{t}}{\text{аф}}}{\text{ппф}} \qquad \dfrac{\dfrac{\dfrac{t}{t}}{\text{аф}} \quad \dfrac{\dfrac{t}{t}}{\text{аф}}}{\text{ппф}}}{\text{ппф}}}{\text{ппф}}}{\text{ппф}}$$

9.3.4. Изчисления с предикати.

Естествена формална система

Използува се същата естествена формална система, която е дефинирана в съждителната логика. Под изчисления с предикати се разбира изчисления на ППФ.

Добавят се допълнително правила за кванторите.

Правила за $\forall$

$$\frac{\Gamma \Rightarrow A(x)}{\Gamma \Rightarrow \forall x\, A(x)}$$ - въвеждане на $\forall$

$$\frac{\Gamma \Rightarrow \forall x\, A(x)}{\Gamma \Rightarrow A(t)}$$ - елиминираме $\forall$ (t е свободно за X в $A(x)$, с премахване на квантора се замества X с t, където t не трябва да бъде свързана.

Правила за $\exists$

$$\frac{\Gamma \Rightarrow A(t)}{\Gamma \Rightarrow \exists x\, A(x)}$$ - въвеждане $\exists$, при което t е свързана за X в $A(x)$;

$$\frac{\big(\Gamma \Rightarrow \exists x\, A(x)\big) \wedge \big(\Gamma, A(b) \Rightarrow c\big)}{\Gamma \Rightarrow c}$$ - Елиминиране на $\exists$ - в

резултат всяка свързана променлива се заменя с индивидна константа, неучастваща в Γ, A и c.

9.4. Метод на резолюцията.

9.4.1. Базов метод на резолюцията. Преобразуване в клаузна форма.

Същност.

Методът на резолюцията е начин (алгоритъм) за определяне дали една ППФ на предикатната логика от I ред е неудовлетворима.

Нека е дадена ППФ A. Прилагайки метода на резолюцията върху нея тогава:

а) Ако A е неудовлетворима, то методът ще установи това;

б) Ако A е удовлетворима, се получава безкраен цикъл.

За установяване удовлетворимост на ППФ A, методът се прилага върху инвертираната ППФ $\overline{A}$. Ако методът установи, че $\overline{A}$ е неудовлетворима, тогава следва, че A е удовлетворима.

Логическата обосновка на това се базира върху еквивалентните преобразувания:

Нека $\Gamma \equiv A_1 \wedge A_2 \wedge \ldots \wedge A_n$ са предпоставки за B,

следователно $\Gamma \rightarrow B \equiv \overline{\Gamma \wedge \overline{B}}$.

Ако се докаже, че $\Gamma \wedge \overline{B}$ е неудовлетворима (false),

следва че $\Gamma \rightarrow B$ е удовлетворима (true).

Методът на резолюцията се прилага само към ППФ от специален вид, наречен клаузна форма.

Задачата по доказателство за неудовлетворимост се разделя на два етапа:

1. Преобразуване на ППФ на предикатната логика в клаузна форма.

2. Прилагане на метода на резолюцията, който се базира на алгоритъма за унификация.

Определение: Клаузна форма - една ППФ е в клаузна форма, ако се състои от конюнкции $(\wedge)$ на клаузи и всички квантори за

общност са изместени вляво на формулата:

$$\underbrace{\forall x_1 . \forall x_2, \ldots, \forall x_n}_{\text{префикс}} \underbrace{\left(c_1 \wedge c_2 \wedge \ldots \wedge c_k\right)}_{\text{матрица}},$$

където $c_i \mid i = 1 \ldots k$ са клаузи.

Определение: Клауза - дизюнкция от литерали.

Пример:

$$\underbrace{p(f(x), a)}_{\text{аф}} \vee \underbrace{\overline{g(x)}}_{\text{аф}} \vee \underbrace{\overline{p(a, y)}}_{\text{аф}} \text{ - клауза.}$$

Определение: Литерал - атомна формула или нейното отрицание.

Определение: Базова клауза - конюнкция от всички базови екземпляри на клаузата. Всеки базов екземпляр на една клауза се получава, като се заменят всички индивидни променливи с константни терми от съответната им дефиниционна област.

Пример: Дадена е клаузата

$$A : p(X) \vee q(c, X)$$

Базова клауза:

$$A: \left(p(a) \vee q(c, a)\right) \wedge \left(p(b) \vee q(c, b)\right)$$

Същност на базовия метод на резолюцията.

Ако G_1 и G_2 са базови клаузи и ако G_1 и G_2 съдържат общ литерал с инверсия и без инверсия, тогава:

1. Общият литерал се премахва от G_1 и G_2 $\Rightarrow$ G_1' и G_2'.

2. Съставя се нова клауза G_3 (резолвента), която се състои от дизюнкцията на G_1' и G_2', т.е.

$$G_3 = G_1' \vee G_2'$$

Процесът се нарича резолиране, а стъпки 1 и 2 - резолюция.

Резолирането продължава, докато се получи празна клауза.

Пример:

$G_1 \wedge G_2$, където $G_1 = l$, а $G_2 = \bar{l}$, дава в резултат $G_3 = \square$ (празна клауза).

Ако в резултат от резолюцията се получи празна клауза, ППФ от предикатната логика е неудовлетворима, в противен случай, тя се счита за удовлетворима.

Базовият метод на резолюцията има следните недостатъци:

1. Преди началото на процеса резолюция, трябва да бъдат образувани всички базови екземпляри на всички клаузи. Това е свързано с пълно комбиниране на всички променливи, като се включат всичките им стойности от дефиниционните им области.

2. В резултат на 1. при практическата реализация на метода би следвало да се изразходва машинното време за формирането на множеството базови екземпляри на всички клаузи.

3. За съхраняване на базовите екземпляри на всички клаузи би следвало да се изразходва памет, в която да бъдат съхранени същите.

4. В процеса на резолюцията е възможно извеждането (получаването) на празната клауза да се извърши на базата на значително по-малка част от базовите екземпляри.

Поради горните недостатъци базовият метод на резолюцията се счита за нискоефективен (използва необосновано количество памет и машинно време), поради което той не е реализиран в този си вид в логически програмен език.

Съществен аргумент е и фактът, че понякога дефиниционните области на променливите са неограничени (пример за това са естествените числа), което прави метода да е практически неосъществим.

9.4.2. Преобразуване на ППФ в клаузна форма.

Нека е дадена ППФ

$$\exists y \, \forall z \left(p(z,y) \leftrightarrow \overline{\exists x \left(p(z,x) \wedge p(x,z) \right)} \right)$$

Алгоритъмът за преобразуване се състои в следните няколко етапа:

I. Премахване на $\rightarrow$ и $\leftrightarrow$ от ППФ. За целта се използват следните формулни преобразувания:

$$A \rightarrow B \equiv \overline{A} \vee B$$

$$A \leftrightarrow B \equiv (\overline{A} \vee B) \wedge (A \vee \overline{B})$$

$$\exists y \forall z \left\{ [\overline{p(z,y)} \vee \overline{\exists x(p(z,x) \wedge p(x,z))}] \wedge \right.$$

$$\left. \wedge [p(z,y) \vee \exists x(p(z,x) \wedge p(x,z))] \right\}$$

II. Разчленяване на инверсията.

Използват се законите на Де Морган, на Шенон и правилата за кванторите.

Правила:

$$\overline{A \vee B} \equiv \overline{A} \wedge \overline{B}$$

$$\overline{A \wedge B} \equiv \overline{A} \vee \overline{B}$$

$$\overline{\forall x(p(x))} \equiv \exists x(\overline{p(x)})$$

$$\overline{\exists x(p(x))} \equiv \forall x(\overline{p(x)})$$

$$\exists y \forall z \left\{ [\overline{p(z,y)} \vee \forall x(\overline{p(z,x)} \vee \overline{p(x,z)})] \wedge \right.$$

$$\left. \wedge [p(z,y) \vee \exists x(p(z,x) \wedge p(x,z))] \right\}$$

III. Премахване на кванторите за съществуване (скулемизация).

Правила:

$\forall y \ldots \{\exists x(p(x))\} \equiv \forall y\{(p(f(y)))\}$ - чрез произволна функция $f(y)$

$\exists x(p(x)) \equiv p(a)$ - чрез константа

По този начин се определя свързаната променлива спрямо

дефиниционната област на променливата, в обхвата на която се намира квантифицираната променлива

$$\forall z \left\{ [\overline{p(z,a)} \vee \forall x \left(\overline{p(z,x)} \vee \overline{p(x,z)} \right)] \wedge \right.$$
$$\left. \wedge [p(z,a) \vee p(z,f(z)) \wedge p(f(z),z)] \right\}$$

IV. Изместване кванторите за общност наляво във формулата:

$$\forall z \forall x \left\{ [\overline{p(z,a)} \vee \overline{p(z,x)} \vee \overline{p(x,z)}] \wedge \right.$$
$$\left. \wedge [p(z,a) \vee p(z,f(z)) \wedge p(f(z),z)] \right\}$$

V. Прилагане на дистрибутивните закони.

<u>Правила:</u>

$$\boxed{\begin{array}{l} A \wedge (B \vee C) \equiv A \wedge B \vee A \wedge C \\ A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C) \end{array}}$$

$$\forall z \forall x \left\{ [\overline{p(z,a)} \vee \overline{p(z,x)} \vee \overline{p(x,z)}] \wedge \right.$$
$$\left. \wedge [p(z,a) \vee p(z,f(z))] \wedge [p(z,a) \vee p(f(z),z)] \right\}$$

$\forall z \forall x$ - префикс; $\left\{ \overline{p(z,a)} \ldots \right\}$ - матрица.

<u>Прилагане на базовия метод на резолюцията</u>

Получени са клаузите:

$$c_1: \overline{p(z,a)} \vee \overline{p(z,x)} \vee \overline{p(x,z)}$$

$$c_2: p(z,a) \vee p(z,f(z))$$

$$c_3: p(z,a) \vee p(f(z),z)$$

Прилагане на базовия метод на резолюцията:

Нека е дадена ППФ S в клаузна форма:

$$S: \overline{p(a,a)} \wedge \left(\overline{p(a,a)} \vee \overline{p(a,b)} \vee \overline{p(b,a)} \right) \wedge$$
$$\wedge \left(p(a,a) \vee p(a,b) \right) \wedge \left(p(a,a) \vee p(b,a) \right)$$

Да се провери S за неудовлетворимост по базовия метод на резолюция.

Базовите екземпляри на клаузите са:

c_1: $\overline{p(a,a)}$

c_2: $\overline{p(a,a)} \vee \overline{p(a,b)} \vee \overline{p(b,a)}$

c_3: $p(a,a) \vee p(a,b)$

c_4: $p(a,a) \vee p(b,a)$

c_5: $\overline{p(b,a)}$ резолюция на c_2 и c_3

c_6: $p(a,a)$ резолюция на c_4 и c_5

c_7: $\square$ резолюция на c_6 и c_1

Резултатът е празна клауза, поради което ППФ S е неудовлетворима.

9.5. Общ метод на резолюцията.

9.5.1. Унифициращ алгоритъм. Същност. Особености при реализацията на алгоритъма в езика Пролог.

Същност на унифициращия алгоритъм.

Общият метод на резолюцията се прилага върху клаузна форма на правилно построена формула на предикатната логика без да се търсят базови екземпляри на клаузите, т.е. при наличие в клаузите на индивидни променливи.

Определение: субституция - крайно множество от подредени двойки от вида:

$$\alpha = \{\langle v_1, t_1\rangle, \langle v_2, t_2\rangle \ldots \langle v_n, t_n\rangle\},$$

където v_i са различни индивидни променливи

t_i са терми, различни от v_i

Определение: унификатор на дизюнкция от атомни формули от вида:

$$A_1 \vee A_2 \vee \ldots \vee A_n$$

е такава субституция, при която е възможно заместването на всички променливи с терми до получаване на равенство на отделните атомарни форми в дизюнкцията

$$A_1' \leftrightarrow A_2' \leftrightarrow \ldots \leftrightarrow A_n',$$

където A_i' се получава от A_i след заместване на двойките $\langle v_i, t_i \rangle$.

Определение: <u>унифицируема дизюнкция</u> - такава дизюнкция, за която съществува унификатор.

Определение: <u>най-общ унификатор</u> - такъв унификатор β, за който всеки друг унификатор α за същата дизюнкция я превръща в базов екземпляр на унифицируемата с β дизюнкция.

<u>Пример:</u>

Дадена е ППФ в клаузна форма:

$$\forall x \forall y \forall z \forall y_1 \left(p(x, f(x)) \wedge q(y, g(y)) \wedge \left(\overline{p(a, y_1)} \vee \overline{q(y_1, z)}\right)\right)$$

Клаузите са следните:

c_1: $p(x, f(x))$ $\qquad$ x, y, y_1, z - индивидни променливи

c_2: $q(y, g(y))$ $\qquad$ f, g - едноаргументни функции

c_3: $\overline{p(a, y_1)} \vee \overline{q(y_1, z)}$ $\qquad$ p, q - двуместни предикати

В множеството клаузи се търсят предикати с еднакво име и брой аргументи. След това същите се съпоставят:

$p(x, f(x))$ $\qquad$ от c_1;

$\overline{p(a, y_1)}$ $\qquad$ от c_3.

Ако съществуват литерали, участвуващи с права и инверсна форма се прави опит за субституция , която да ги изравни:

Субституцията = $\{\langle x, a \rangle, \langle y_1, f(a) \rangle\}$ изравнява предикатите

Резолвентата е:

$$c_4: \overline{q(f(a), z)}$$

Съпоставят се литералите на c_2 и c_4. Те са с еднакво име и местност, поради което се прави опит за изравняване

$$c_2: q(y, g(y))$$

$$c_4: \overline{q(f(a), z)}$$

Субституцията = $\{\langle y, f(a)\rangle, \langle z, g(f(a))\rangle\}$ - изравнява двата литерала до:

$$q(f(a), g(f(a))),$$

при което се извежда празна клауза.

Вследствие на приложението на общия метод на резолюцията се получава празна клауза. По-съществен резултат са условията, при които е изведена неудовлетворимостта на ППФ, а те са следните:

$$x = a; \quad y = y_1 = f(a); \quad z = g(f(a))$$

Най-общият унификатор е:

$$\beta = \{\langle y, a\rangle, \langle z, f(x)\rangle\}$$

9.5.2. Действие на унифициращия алгоритъм

Алгоритъмът започва от празната субституция $\alpha = \{\ \}$ и построява постъпково най-общия унификатор β, ако такъв съществува.

На всяка стъпка $k \geq 1$ алгоритъмът работи по следния начин:

1. Прилага текущата субституция.

2. Ако всички атомни формули в дизюнкцията са идентични, алгоритъмът спира, обявява унифицируемост, а текущата субституция α_k за най-общ унификатор.

3. Ако атомните формули не са идентични, определя множеството на несъвпадение, разглеждайки всяка атомарна форма като низ от символи, които не съвпадат.

4. Проверява има ли в множеството на несъвпадение поне една променлива. Ако няма - спира като обявява неунифицируемост.

5. Разширява α_k с нова двойка, която да направи поне два члена от множеството на несъвпадение единични.

6. Увеличава k и преминава към 1.

Особености при реализацията на унифициращия алгоритъм в езика за логическо програмиране Пролог.

В Пролог унифициращият алгоритъм се изпълнява на всяка стъпка от действието на машината за логически извод. Той се осъществява върху два обекта от базата знания посредством процедура, написана на езика, реализиращ интерпретатора на Пролог.

Същността на действието на алгоритъма се реализира чрез подаване параметрите (двата обекта, подлежащи на унификация), като вследствие от действието му се връща логическа стойност *true* при унифицируемост или *false* при невъзможност за унифициране.

Съпоставянето се извършва на базата на следната таблица:

Таблица 9.2. Таблица на съответствие

	atom b	*variable* X_b	*term* t_b
atom a	*if* $a = b$ *true* *else false* $u = \{\ \}$	*true* $u = \{\langle X_b, a\rangle\}$	*false* $u = \{\ \}$
variable X_a	*true* $u = \{\langle X_a, b\rangle\}$	*true* $u = \{\langle X_a, X_b\rangle\}$	*true* $u = \{\langle X_a, t_b\rangle\}$
term t_a	*false* $u = \{\ \}$	*true* $u = \{\langle X_b, t_a\rangle\}$	*Забележка*

Забележка: При съпоставянето на термите t_a и t_b се проверява:

1) Еднаквост на имената. Прави се синтактично съответствие на името на t_a с името на t_b. При несъвпадение унификацията е неуспешна.

2) Проверка за еднаква местност (брой аргументи на термите). Ако не съвпадат унификацията е неуспешна.

3) Прави се цикъл по всички аргументи на термите отляво надясно като върху всеки аргумент се прилага същото съпоставяне по дадената таблица. Ако има възможност за изравняване на всички аргументни двойки, тогава унификацията е успешна.

Обикновено обектите в Пролог интерпретатора се представят посредством указател към обекта. Ефективен достъп и бързо изпълнение на алгоритъма може да се постигне като се установи възможно най-бързо типа на обекта.

9.6. PROLOG - логически програмен език
(Programming in Logic)

9.6.1 Същност.

База на PROLOG е изчисляване на ППФ от предикатната логика от I ред чрез общ метод на резолюцията и унифициращ алгоритъм.

Изхожда се от клаузната форма:

$$c\colon A_1 \vee A_2 \vee \ldots \vee A_n \vee \overline{B}_1 \vee \overline{B}_2 \vee \ldots \vee \overline{B}_m$$

Преди да се добави в базата от правила на Пролог, новата клауза c се обработва:

$$c\colon A_1 \vee A_2 \vee \ldots \vee A_n \vee \overline{\overline{\overline{B}_1 \vee \overline{B}_2 \vee \ldots \vee \overline{B}_m}}$$

$$c\colon A_1 \vee A_2 \vee \ldots \vee A_n \vee \overline{B_1 \wedge B_2 \wedge \ldots \wedge B_m}$$ - въвежда се импликация

$$c\colon A_1 \vee A_2 \vee \ldots \vee A_n \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_m$$

След заместване с:

$$\wedge \equiv ,$$
$$\vee \equiv ;$$
$$\leftarrow \equiv :-$$

се получава клауза на PROLOG, която накрая завършва с точка.

$$A_1; A_2; \ldots; A_n :- B_1, B_2, \ldots, B_m.$$

В PROLOG са допустими само Хорн-клаузи, т.е. клаузи, съдържащи само един неинвертиран литерал. В резултат се получава:

$$\underbrace{A}_{\text{глава}} :- \underbrace{B_1, B_2, \ldots, B_m}_{\text{тяло}}$$

клауза на PROLOG, дефинираща правило, записано в базата знания:

Всяко правило се тълкува както следва:

A е истинно, ако B_1 и B_2 и ... и B_m са удовлетворими.

Базата знания на PROLOG се състои от множество правила:

$$\left.\begin{array}{l} B_1 :- \ldots \quad . \\ B_2 :- \ldots \quad . \\ \ldots\ldots \\ B_m :- \ldots \quad . \end{array}\right\} \text{База знания}$$

<u>Частни случаи</u>

1. В клаузата липсват инвертирани литерали, т.е. тя се състои само от 1 инвертиран литерал.

$A :- .$ Тялото е празно. Тогава B е безусловно истина.

$A.$ Клауза от този вид се нарича факт.

Множеството от тези правила образуват база от данни на PROLOG.

2. Липсва неинвертирания литерал (описателя, главата) на клаузата:

$:- B_1, B_2, \ldots, B_m.$

$?- B_1, B_2, \ldots, B_n.$

Описателят е празен. Такава клауза се нарича въпрос. Чрез него може да се изведе празната клауза.

Пример:

(база знания)

сестри (X,Y): - жена (X), жена (Y), родител (X,Z),

родител (X,Y): - майка (X,Y).

родител (X,Y): - баща (X,Y).

(база факти)

жена (ана). жена (мария). жена (ваня). жена (ева). ...

баща (ема, георги). баща (ваня, георги). ...

майка (ана, ева). майка (мария, ева). ...

Във всяка една клауза променливите представят един и същи обект. В различните клаузи еднаквите променливи не означават един и същи обект. В случай, че имената трябва да се запишат с главна буква се използва апостроф ('Ана'...) за да се отличават от променливите, които се пишат с главна буква.

Въпросите се въвеждат обикновено от потребителския вход:

? - сестри (ана, мария).

Отговора се извежда в потребителския изход:

да

? - сестри (ева, X).

Унификаторите на входните променливи се извеждат също в потребителския изход:

X = ваня.

? - сестри (X,Y).

X = ана. Y = мария.

X = ева. Y = ваня.

9.6.2. Основни принципи при обработката на структурите в езика Пролог. Същност на машината за логически извод.

1. Описание на унификацията

Основните моменти, свързани с реализацията на операцията унификация са следните:

- Намиране на клаузите, съдържащи целта (*pg*), която трябва да се изпълни и глава на някоя клауза, която може да бъде унифицирана с нея (*g*). Клаузите в езика от потребителска гледна точка са предикати, въведени чрез файл, или от клавиатурата. Целевата клауза постъпва обикновено от клавиатурата.

- Осъществяване на унификацията. Извършва се последователно, като първоначално се започва от 1 аргумент и се извършва за всички аргументи отляво надясно. Възможно е предаване на значения в двете посоки - от целевата клауза към клауза в базата данни (правила) и обратно. Алгоритъмът за унификация се основава на следната последователност.

1. $a \Leftrightarrow b$, където a, b са константи

   1.1. Ако $a = b$ връща true и унификатора $u = \{\ \}$

   2.2. Ако $a \neq b$ връща false и унификатор $u = \{\ \}$

2. $Var_a \Leftrightarrow b$, Var_a е променлива b - const;

   връща *true* и свързва в унификатор $u = \{\langle Var_a, b\rangle\}$

3. $term_a \Leftrightarrow b$, където $term_a$ е терм $f(...)$

   връща *false*

4. $Var_a \Leftrightarrow Var_b$

   връща *true* и унификатор съвместени променливи

   $u = \{\langle Var_a, Var_b\rangle\}$

5. $Var_a \Leftrightarrow term_b$

   връща *true* и унификатор $u = \{\langle Var_a, term_b\rangle\}$

6. $term_a \Leftrightarrow term_b$

6.1. Проверяват се за еднаквост имената на двата терма. При съвпадение - 6.2. В противен случай - *false*.

6.2. Проверка за еднаква местност (аргументност) на термите. При съвпадение 6.3. В противен случай - *false*.

6.3. Прави се цикъл по всички променливи (аргументи) на термите, при което се реализира за всяка двойка аргументи съответно на t_a и t_b рекурсивно същият алгоритъм за унификация.

2. Описание на алгоритъма за възраст

Когато има няколко клаузи с една и съща глава е безразлично по принцип коя клауза ще се избере за унификацията. По такъв начин се внася недетерминираност в езика. При реализацията на Пролог обикновено се избира ред за изпълнение, основан на последователността, получена при въвеждане на клаузите в базата. Започвайки от глава на клаузата, изпълнението се извършва последователно в тялото, като при неуспешно изпълнение се преминава към следващата клауза. Тази процедура се изпълнява до изчерпване на всички възможни клаузи (изчерпване на алтернативите).

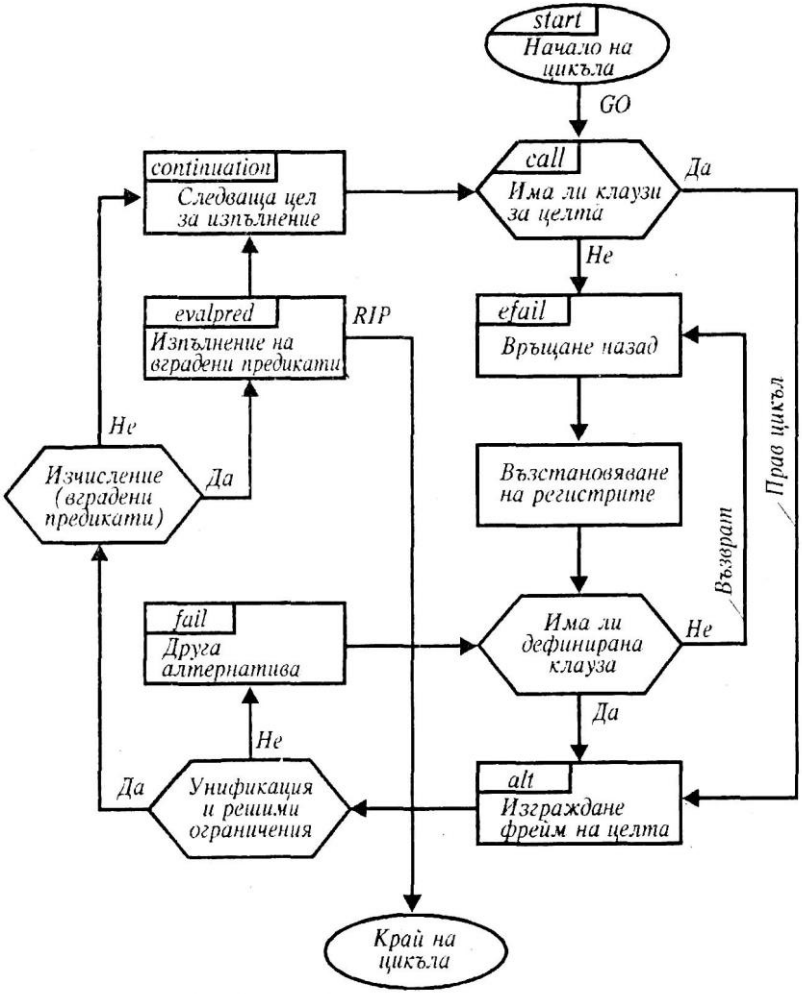
Механизмът се нарича *backtracing* (възврат).

Той се извършва в следните случаи:

2.1. В случай, че при търсенето на клаузата (g) за унификация не може да се намери такава. Тук се включва и случая, когато изпълнението на всички клаузи с използуването на *backtracing* е завършено или в случай, че клаузата (g) е намерена, но операцията унификация е завършила неуспешно.

2.2. В случай, че унификацията между клаузата цел и главата на унифицираната клауза е завършила успешно, но изпълнението на g се окаже неуспешно.

Машината за логически извод може да се илюстрира чрез следния граф:



Фиг. 9.1. Граф на машина за логически извод

Формирането на извода се получава на базата на циклично изпълнение (доказване) на цели. Извеждането на една цел и свързаната с това работа по установяване на регистрите на машината и изграждане на структури за представяне на целта се нарича цикъл на работа на Пролог - машината. Различават се два цикъла

на работа на машината - прав цикъл и възврат.

Прав цикъл - Първоначално влизане в цикъла се осъществява от точка START. В точка *icall* са установени регистрите (глобални указатели) на Пролог машината, като регистър *pg* съдържа адреса на целта, подлежаща на доказателство. След уточняване на функтора се проверява има ли дефинирани клаузи за него. В случай, че такива няма се преминава в режим на възврат. При наличие на клаузи за целта (*pg*) се избира първата (дефинирана) от тях в регистър (*g*) и започва нейното изпълнение. Изгражда се структурата (фрейм) на целта и се преминава във възел *alt*. Във възела *alt* се извършва подготовка на стековото пространство за променливи на изпълняваната клауза (след структурата на целта) като се нулира напред. Извиква се процедурата за унификация. При неуспешна унификация се влиза в цикъл по възел *fail* за изчерпване на всички клаузи, кандидати за тази цел. Ако не се открие подходяща клауза, се предизвиква възврат. При успешна унификация цикълът на работа продължава като се проверява тялото на клаузата. Ако то е число, тогава се изпълнява съответствуващия програмен текст. Клаузи с тяло число са т.н. вградени предикати, които определят еднозначно програмен текст (модул), който се изпълнява и се преминава към възел *call*. В тази точка се определя следващата цел за изпълнение. В случай, че клаузата има тяло, то следващата цел е първата подцел от тялото. След намиране на новата цел за изпълнение цикълът се затваря.

Възврат - той се осъществява като при всеки цикъл се извършва премахване на фрейма на проваленатa цел и възстановяване на всички регистри. Прави се опит за успешно преминаване в правия цикъл и т.н.

3. Представяне на цели

Цел в Пролог се явява структура, инвертирана цел или последователност от цели. Най-общата структура е последователността от цели, която се изгражда с помощта на стеков механизъм от елемента FRAME.

FRAME *Inf* за управление
Inf за променлива

Inf за управление включва:

- адрес на целта, която трябва да се изпълни в случай, че изпълнението на всички цели от тялото се окаже успешно;

- адрес на целта за избор, в случай, че изпълнението на текущата цел се окаже неуспешно;

- адрес на клаузата, която трябва да се изпълни при неуспешно изпълнение на целта, но съществуват алтернативи за успешно продължение;

- адрес на клаузата, която трябва да се изпълни при неуспешно изпълнение на целта и при липса на алтернативи за успешно продължаване.

Inf за променливите

Формата е указател към стек за променливите (*trail stack*), в който са записани всички унификатори на всички променливи, които влизат в целта posredstwom указатели.

4. Механизъм за работа със стека

Всеки път, когато някаква целева клауза *pg* извиква за унификация глава на клауза *g*, в стека се внасят условията за изпълнение на унификацията. Структурата, която се използва за целта е FRAME. Информацията, която се включва във фрейма, се разделя на две части:

- за управление на изпълнението;

- за променливите, влизащи в състава на цел *g* .

Информацията за управление на изпълнението включва:

- адрес на целта, която трябва да бъде поставена след успешното изпълнение на всички цели от тялото на целта *pg*;

- адрес на целта, която се избира в случай, че изпълнението

на текущата клауза се окаже неуспешно;

- адрес на клаузата, която трябва да се изпълни при "плитък *backtracing*" (съществуват все още алтернативи за решаване на целта);

- адрес на клаузата, която трябва да се изпълни при липса на алтернативи "дълбок *backtracing*".

Информацията за променливите, влизащи в състава на целта *g* включва текущите унификатори на всички променливи (т.е. условията за осъществяване на резолюцията).

Пример:

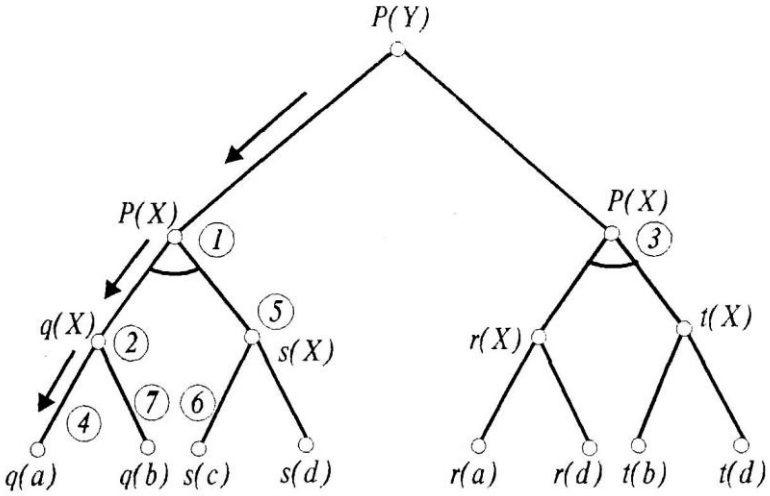
Дадена е програмата Пролог.

$p(x):-q(x), s(x).$

$p(x):-r(x), t(x).$

$q(a). q(b). r(a). r(d). s(d). t(b). t(d).$

В съответствие с тази програма може да се състави AND-OR граф, описващ еквивалентно клаузите на Пролог програмата.



Фиг. 9.2. Граф на Пролог програма

Характерното при удовлетворяване на целта *P(Y)* е, че дървото - хипотеза се генерира динамично, като се съблюдават

както текущите стойности на променливите (унификатора), така и възможностите за обхождане на всички алтернативи (пътища в графа), които са възможни решения на този граф. По своята същност алгоритъмът се явява търсене в дълбочина с хронологичен възврат и намиране на алтернативи.

Нека предположим, че текуща цел се явява $q(a)$. Тогава параметрите за текущия фрейм на целта са:

$continue \equiv s(x)$	към	$CLAUSEREC\ s(x)$
$alt \equiv q(b)$	към	$CLAUSEREC\ q(b)$
$fail \equiv p(x)$	към	$FRAME$ в точка 1
$efail \equiv p(x)$	към	$CLAUSEREC\ p(x){:-}v(x), t(x)$

Текущият унификатор е $\langle Y, X \rangle$, записан в стека за променливите (*trail* stack). Последователността на обхождане (изграждане на фрейми) е 1, 2, 4, 5, 6 при *fail* на 4 с alt = 7 и т.н.

ЛИТЕРАТУРА

1. *Агафонов, В. Логическое програмирование, Мир, Москва, 1988.*
2. *Денев, Й. Математическа лингвистика, Наука и изкуство, София, 1980.*
3. *Кросли, Д. Що е математическа логика, Наука и изкуство, София, 1980.*
4. *Лавров, И. Задачи по теории множеств, математической логике и теории алгоритмов, Наука, Москва, 1984.*
5. *Линднер, Р. Теория и практика структурного програмирования, Мир, Москва, 1982.*
6. *Манна, Зоар Математическа теория на информацията, Наука и изкуство, София, 1985.*
7. *Никольская, И. Математическая логика, Высшая школа, Москва, 1981.*
8. *Олюнгрен, А. Определение языков програмирования интерпретирующими автоматами, Мир, Москва, 1982.*
9. *Радославов, И. Математическа лингвистика, Народна просвета, София, 1982.*
10. *Рейуорд, В. Теория формальных языков, Радио и связь, Москва, 1988.*
11. *Стойчев, Л., А. Антонов. Програмни езици за изкуствен интелект, Техника, София, 1989.*
12. *Яблонский, С. Введение в дискретную математику, Наука, Москва, 1986.*
13. *Якоба, К. Машина Тьюринга и рекурсивные функции, Мир, Москва, 1980.*
14. *Янков Б. Транслатори и операционни системи, ВМЕИ, София, 1984.*
15. *Peterson, J. Petri net theory and the modeling of systems, Prentice Hall, 1981.*

СЪДЪРЖАНИЕ

Основни понятия 4

1.Множества. Отношения и функции в множествата..... 6

1.1. Теория на множествата..... 6

1.2. Операции над множества..... 7

1.3. Свойства на операциите над множества..... 8

1.4. Отношения в множествата..... 9

1.5. Функции в множествата..... 10

2. Подредени множества и низове. Операции с низове.

Регулярни множества и изрази..... 12

2.1. Подредени множества..... 12

2.2. Операции с низове..... 13

2.3. Регулярни множества..... 14

2.4. Регулярни изрази (РИ)..... 14

2.5. Изчисления с регулярни изрази..... 15

3. Формални граматики и езици..... 16

3.1. Основни понятия..... 16

3.2. Формална граматика G..... 18

3.3. Класификация и йерархия на G (по Хомски)..... 20

3.4. Видове продукции..... 21

3.5. Език на граматиките Бакус-Наур формализъм..... 21

3.6. Синтактични графи..... 22

4. Абстрактни автомати 25

4.1. Изчислимост и разрешимост 25

4.2. Понятие за алгоритъм 27

4.3. Машина на Тюринг 28

5. Същност и действие на машината на Тюринг (МТ) 38

5.1. Същност на МТ 38

5.2. Съставни части на МТ над азбуката $\Sigma = (a, b)$ 38

5.3. Действие на МТ 39

5.4. Изчисляване с МТ 41

5.5. Класификация и йерархия на абстрактните автомати (АА) 45

5.6. Йерархия на описаните АА 47

5.7. Еквивалентност между АА и граматики 48

5.8. Фон Нойманов автомат (ФНА) 48

6. Рекурсивни функции (РФ) 52

6.1. Базови функции: 52

6.2. Частично рекурсивни функции 54

7. Паралелни системи и процеси 55

7.1. Паралелизъм - същност 55

7.2. Теория на мрежите на Петри (МП) 57

7.3. Мрежови езици 68

7.4. Моделиране на паралелни процеси с мрежи на Петри 74

8. Абстрактни информационни структури 81

8.1. Йерархия на информационните структури (ИС) 82

8.2. Представяне на моделите в паметта 85

9. Математическа логика 89

9.1. Предмет на математическата логика 89

9.2. Препозиционна логика (съждително смятане) 90

9.3. Предикатна логика 99

9.4. Метод на резолюцията 105

9.5. Общ метод на резолюцията 110

9.6. PROLOG - логически програмен език 114

Литература 124

ДИСКРЕТНИ СТРУКТУРИ

Учебно пособие

Автори:
доц. д-р Анатолий Стефанов Антонов
доц. д-р Владимир Николов Николов
ас. д-р Ивайло Пламенов Пенев

Рецензент:
доц. д-р Трифон Русков