РЕГУЛЯРНИ ИЗРАЗИ И АВТОМАТИ

Нека си представим, че потребител иска да получи повече информация за морски лъв, но при въвеждане в браузъра се появява информация за други морски обитатели или лъв от „Серенгети“, техните местообитания, застрашеност от изчезване, хранителна верига и прочие, което налага конкретизиране на търсенето докато не се появи информация, която потребителят желае.

Вместо да направи това търсене два пъти би предпочел да въведе една команда за търсене, указваща нещо общо. Или може би искал да търси всички цени в някакъв документ, а може би да всички низове, което налага въвеждане на термина регулярен израз, стандартната нотация за характеризиране на текстови последователности.

В информатиката регулярен израз (на английски: regular expression, съкращавано понякога като regex или regexp) е последователност от знаци, която дефинира шаблон за търсене. Обикновено този шаблон се използва от алгоритми за претърсване на низове за операции от типа „търсене“ или „търсене и заместване“ върху низове или за проверка на валидността на въведени данни. Това е техника, разработена в рамките на теоретичната информатика и теорията на формалните езици.

Регулярният израз се използва за указване на текстови низове в ситуации като посочения пример за търсене в мрежата и в други приложения за извличане на формации, но също така играе важна роля в текстообработката, изчисляване на честоти от корпуси и други подобни задачи.

След като са дефинирани регулярни изрази, показва как те могат да бъдат реализирани чрез **автомата с крайни състояния**. Автоматът с крайни състояния е не само математическото устройство, използвано за прилагане на регулярни изрази, но и едно от най-значимите инструменти на компютърната лингвистика.

Вариациите на автомати като преобразуватели с крайни състояния, скрити модели на Марков и N-грам граматики са важни компоненти на приложения, които ще бъдат представени в следващите теми, включително разпознаване на реч и

синтез, машинен превод, проверка на правописа и извличане на информация.

Един от успехите в стандартизацията в компютърните науки е регулярният израз (RE), език за определяне на низове за търсене на текст. Езиците за регулярни изрази, използвани за търсене на текстове в UNIX (vi, Perl, Emacs, grep), Microsoft Word (версия 6 и след това) и WordPerfect са почти идентични и много RE функции съществуват в различните уеб търсачки. Освен тази практическа употреба, регулярният израз е важен теоретичен инструмент в компютърните науки и лингвистиката.

Регулярният израз се представя и като формула на специален език, който се използва за определяне на прости класове от низове . Низът е поредица от символи; за целите на повечето техники за търсене, базирани на текст, низът е всяка последователност от буквено-цифрови знаци (букви, цифри, интервали, раздели и препинателни знаци). За тези цели интервалът е просто символ като всеки друг и се представя с отделен символ.

Формално регулярният израз е алгебрична нотация за характеризиране на набор от низове. Така те могат да се използват за указване на низове за търсене, както и за дефиниране на език синтаксиса на регулярните изрази се простира до всички регулярни изрази на UNIX, Microsoft Word и WordPerfect.

Търсенето на регулярни изрази изисква модел, при който се търси. Функция за търсене на регулярен израз ще търси в корпуса, връщайки всички текстове, които съдържат шаблона. В система за извличане на информация, като например уеб търсачка, текстовете могат да бъдат цели документи или уеб страници.

В текстообработваща програма текстовете могат да бъдат отделни думи или редове от документ. Така, когато се даде шаблон за търсене се приема, че търсачката връща реда на върнатия документ . Това изпълнява командата grep на UNIX и подчертава точната част от шаблона, която съответства на регулярния израз. Търсенето може да бъде проектирано така, че да върне всички съвпадения на регулярен израз или само първото съвпадение.

Основни модели на регулярен израз

Най-простият вид регулярен израз е поредица от прости знаци. Например, за

да се търси *lion*. Така че регулярният израз */sea lion/* съвпада с всеки низ, съдържащ подниза *lion*.

Низът за търсене може да се състои от един знак (като /!/) или поредица от знаци (като /urgl/).

Регулярните изрази са чувствителни към главни и малки букви; малка буква /s/ е различна от главна буква /S/

ЗНАК в квадратни скоби елиминира този недостатък [1234567890]

Регулярният израз посочва всяка една цифра. Докато класове на знаци като цифри или букви са важни градивни елементи и изрази, стават неудобни (напр. неудобно е да се уточни /[ABCDEFGHIJKLMNOPQRSTUVWX YZ]/ и означава „всяка главна буква").

В тези случаи скобите могат да се използват с тире (-).

ОБХВАТ за да посочите всеки един знак в диапазон. Моделът/[2-5]/ определя който и да е от символи 2, 3, 4 или 5. Моделът/[bg]/ задава един от символите b, c, d, e, f или g.

RE		Примерни съвпадащи модели
/[AZ]/ /[az]/ / [0-9]/	Свържете главна буква с малка буква и една цифра	"Woodchuck" /[abc]/ 'a', 'b', *or* 'c' "In uomini, in soldati" /[1234567890]/ any digit "plenty of 7 to 5" /[1234567890]/ any digit "plenty of 7 to 5"
Използване на скоби [] плюс тире - за указване на диапазон.		

Квадратните скоби също могат да се използват, за да се укаже какво не може да бъде отделен знак, чрез използване на каретката ". Ако каретката " е първият символ след отворената квадратна скоба ["полученият модел се отрича. Например моделът/["a]/ съвпада с всеки единичен знак (включително специални знаци), с изключение на *a*. Това е вярно само когато каретката е първият символ след отворената квадратна скоба.

RE	Съвпадение (единични	Примерни съвпадащи модели
[A-Z] rss] Г\.] [еЧ а~Ь	знаци) не е главна буква, нито 'S', нито 's', нито точка, нито 'e', нито моделът *'з~Ь'*	“we should call it ‘Drenched Blossoms’” “my beans were impatient to be hoed!” `/[0-9]/` a single digit “Chapter 1: Down the Rabbit Hole”

Използване на каретката “ за отрицание или просто за означаване

Използването на квадратни скоби решава проблема с изписването на главни букви за дървесни разклонения, но все още няма отговор на първоначалнния въпрос; как да уточним кой от двата „лъва“ искаме да получим? Не можем да използваме квадратните скоби, защото докато те позволяват изразяване от типа„s или S", те не позволяват да се каже „s или нищо". За целта се използва въпросителния знак /?/, което означава „предходният знак или нищо"..

Може да се мисли за въпросителния знак като означаващ „нула или едно копие на предишния знак". Тоест, това е начин да се уточни колко от нещо се иска. Досега не е било необходимо да се уточнява, че се иска повече от едно нещо. Но понякога имаме нужда от регулярни изрази, които позволяват повторения на неща. Например, разглеждайки езика на овце, който се състои от низове, които изглеждат като следното:

baa!

baaa!

baaaa!

baaaaa!

baaaaa!

Този език се състои от низове с „b, последвано от поне две „a“ последвано от

удивителен знак. Наборът от оператори, които позволяват да се казват неща като „някое число от а" се основават на звездичката или *, обикновено наричана Kleene * (от англ. се „cleany star"). Звездата Kleene означава „нула или повече повторения на незабавното предишен знак или регулярен израз". Така /a*/ означава „всеки низ от нула или повече ". Това ще съвпада с *a* или *aaaaaa* , Така че регулярният израз за съвпадение на едно или повече *a* е /*aa**/.

По-сложните модели също могат да бъдат повторени. Така /[ab]*/ означава „нула или повече *a* или *b*" (не „нула или повече в квадратни скоби"). Това ще съответства на низове като *aaaa* или *abababab* или *bbbb*.

Знаейки това е възможно да се посочи част от регулярен израз за цени и множество цифри. Регулярният израз за отделна цифра е /[0-9]/. Така че регулярен израз за цяло число (низ от цифри) е /[0-9][0-9]*/. (Защо не е то просто /[0-9]*/?)

Понякога е досадно да трябва да се напише регулярния израз за цифри два пъти, така че има по-кратък начин за указване на „поне един“ от някакъв знак. Това е Kleene +, което означава „един или повече от предишния знак“. Така изразът /[0-9]+/ е нормалният начин за указване на „поредица от цифри". Следователно има два начина за уточняване овчият език: /baaa*!/ или /baa+!/.

Един много важен специален знак е точката (/./) израз със заместващ знак, който съответства на който и да е единичен символ (с изключение на връщане на каретка):

Котви

Котвите са специални знаци, които закотвят регулярни изрази към определени места в низ. Най-често срещаните котви са каретката ^ и знакът за долар $ съответства на края на ред.

По този начин има три употреби на каретката ^ за съпоставяне на началото на ред, като отрицание вътре в квадратни скоби и просто означава карета.

Например /\b99\b/ ще съответства на низа 99 в „Има 99 бутилки бира на стената“ (защото 99 следва интервал), но не и 99 в „Има 299 бутилки бира на стената (тъй като 99 следва число). Но ще съответства на 99 в 99 (тъй$ато 99 следва долар знак ($), който не е цифра, долна черта или буква).

Дизюнкция, групиране и приоритет

Предполага се, че трябва да се търсят текстове за домашни любимци; например сме особено заинтересовани при котки и кучета. В такъв случай може да се въведе в търсачка котка/куче. Тъй като не може да се използват квадратните скоби, за да се търси „котка или куче" има нужда от нов оператор, операторът за дизюнкция , обозначаван също със символ |. Образецът /котка |куче/ съвпада или със стринг котката, или с стринг куче.

Понякога трябва да се използва този оператор за дизюнкция в средата на по-голяма последователност. Например търси се информация за домашни рибки за моя братовчед Дани. Как могат да се посочат и гупа/и (guppy)? Не можем просто да се каже /guppy |ies /, защото това би съвпаднало само с низовете guppy и ies. Това е предимство тъй като последователности като guppy имат предимство пред оператора за дизюнкция |.

За да се отнася оператора за дизюнкция и прилага само към конкретен модел, трябва да се използват операторите за скоби (и/ies). Ограждането на шаблон в скоби го кара да действа като един знак за целите на съседни оператори като „гупа“ | и Клийн*.

Така моделът /gupp(y |ies)/ уточнява, че се има предвид дизюнкцията да се прилага само към наставките y и ies.

Идеята, че един оператор може да има предимство пред друг, изисква понякога да се използват скоби, за да се уточни какво се има предвид, е формализирана от йерархията на приоритета на операторите за регулярни изрази. Следната подредба дава реда на приоритета на RE оператора, от най-високия до най-ниския приоритет:

Скоби ()

*Броячи * + ? {}*

Последователности и котви the ˆmy end$

Приоритет |

Операторът за скоби (също е полезен, когато се използва броячи като Клийн.*

За разлика от | оператор, операторът Kleene* се прилага по подразбиране

само на един знак, а не цяла поредица и съвпада с theeeee но не и с thethe.
Нека се съпоставят повтарящи се случаи.

Моделите могат да бъдат двусмислени и по друг начин. Разглеждайки израза /[а-я*/ при съвпадение с текста „привет земляни“
Тъй като /[а-я]*/ съответства на нула или повече букви, този израз не може да съвпадне с нищо или само с първата буква *п* , или *пр*, или *при* и т.н

В тези случаи регулярните изрази винаги съответстват на най-големия низ, който могат да открият; Това се описва с термина алчност и се казва, че моделите се разширяват, за да покрият колкото се може повече от низа.

Пример

Нека се иска написване на регулярен израз, за да се намерят случаи на английския определителен член the. Прост (но неправилен) модел може да бъде:

/the/

Един проблем е, че този модел ще пропусне думата, когато започва изречение и следователно се пише с главна буква (т.е. The). Това може да доведе до следния модел:

/tT] he/

Но пак ще връща неправилно текстове с вградени други думи (напр. other или theology). Така че трябва да се уточни, че се искат онези с граница на думата от двете страни:

/\b[tT]he\b/

Ако е необходимо или се иска да се направи това без използването на /\b /? е възможно, тъй като /\b / няма да третира долните черти и числата като граници на думата; но може да се иска да се намери в някакъв контекст, където също може да има - подчертавания или числа наблизо (the или the25). Трябва да се уточнят, случаите, в които няма азбучни букви от двете страни на the:

/ ˆ a-zA-Z][tT]he / ˆ a-zA-Z]

Но има още един проблем с този модел: той няма да намери думата , когато

започне ред. Това е така, защото регулярният израз `[^a-zA-Z]`, който бе използван, за да се избегнат вградените the, предполага, че трябва да има някакъв единствен (макар и неазбучен) знак преди the. Може да се избегне и това, като се уточни началото на реда или неазбучен знак и същото в края на реда:

/ |ra-zA-Z])[tT]he(ra-zA-Z] |$)/

Процесът се основава на коригиране на два вида грешки: фалшиви положителни резултати, низове, които неправилно съпоставки като other или there, и фалшиви отрицания, низове, които неправилно са пропуснати, като The. Отстраняването на тези два вида грешки се появява отново и отново при изграждането и подобряването на системите за обработка на реч и език. По този начин намаляването на процента грешки за дадено приложение включва:

— Повишаване на точността (минимизиране на фалшивите положителни резултати)

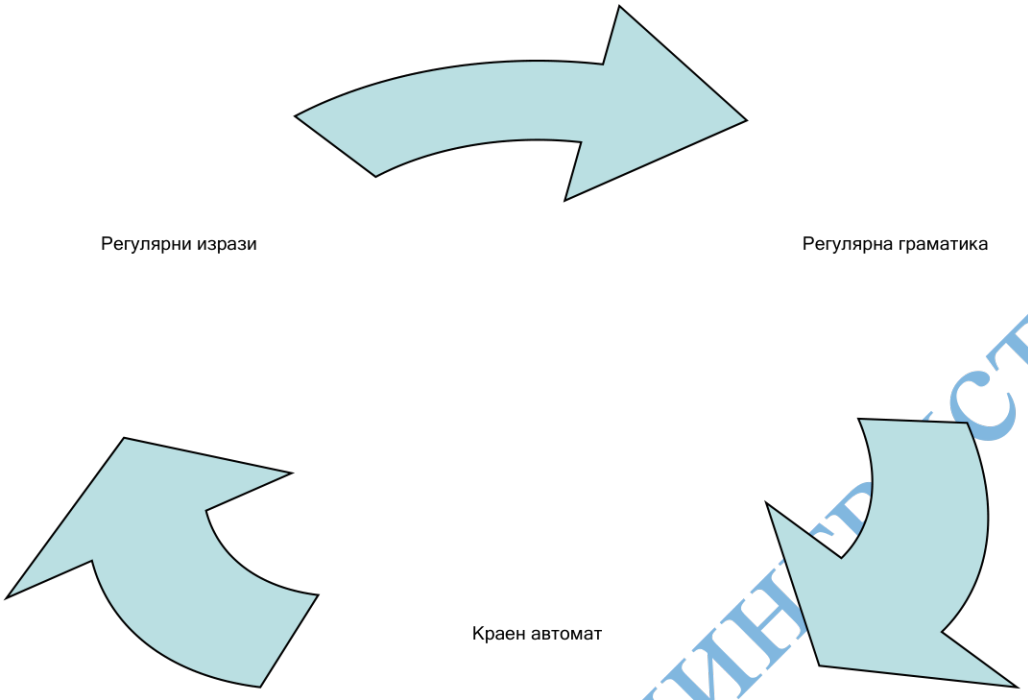
— Увеличаване на покритието (минимизиране на фалшивите отрицателни резултати).

КРАЙНИ АВТОМАТИ

Регулярният израз е нещо повече от удобен метаезик за търсене на текст.

Първо, регулярният израз е един от начините за описване на автомат с крайно състояние (finite state automat FSA). Всеки регулярен израз може да бъде имплементиран като автомат с крайни състояния (с изключение на регулярни изрази, които използват функцията памет;). Симетрично, всеки автомат с крайни състояния може да бъде описан с регулярен израз. Второ, регулярният израз е един от начините за характеризиране на определен вид на формален език, наречен нормален език. Както регулярните изрази, така и автоматите с крайни състояния могат да се използват за описание на регулярни езици. Трети еквивалентен метод за характеризиране на регулярните езици, регулярната граматика .

Връзката между тези теоретични конструкции е показана на фигурата.



jj

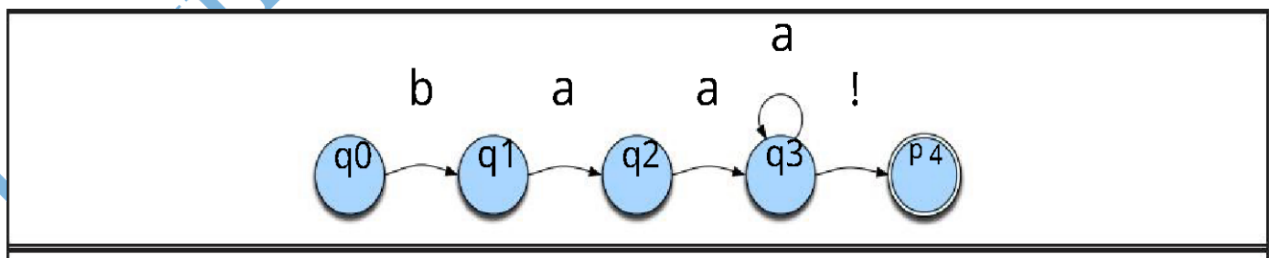
Примера с „овчия език", който бе показан по-горе. Припомняйки си, че дефинирането на овчия език е като всеки низ от следния (безкраен) набор:

баа!

бааа!

баааа!

бааааа!

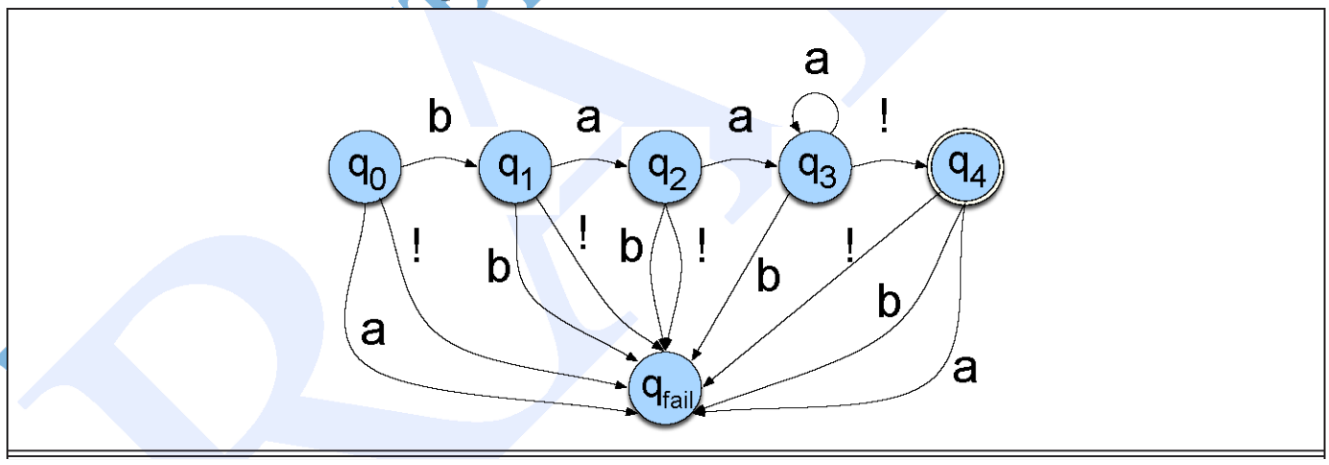


Фигура 2.10 Автомат с крайни състояния за говорещи овце.

Машината стартира в начално състояние (q^O) и повтаря процеса: Проверка за следващата буква от входа. Ако съвпада със символа на дъга, напуска това състояние, след това преминава към следващото състояние и също напредва с един символ.

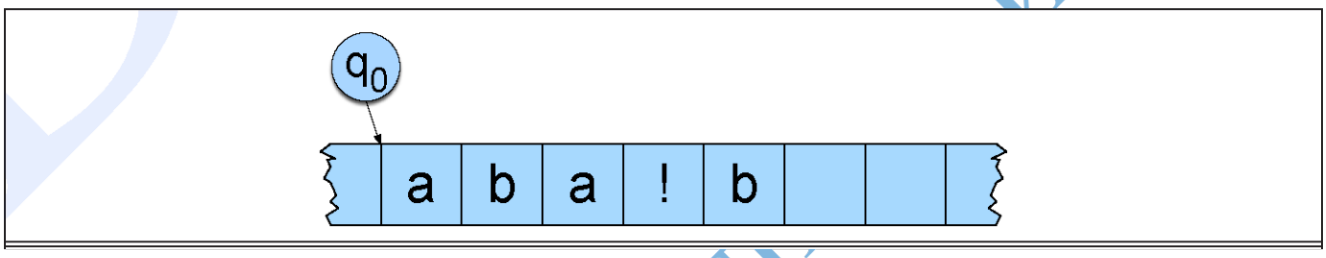
Ако е в състояние на приемане (*q4*), когато входът свърши, машината има успешно разпознат образец. Ако машината никога не стигне до финално състояние, или защото изчерпва входа, или получава някакъв вход, който не съответства на дъга или ако просто се случи да остане в някакво некрайно състояние, се казва, че отхвърля или не успява да приеме вход.

Алгоритъмът ще се провали, когато няма легален преход за дадена комбинация от състояние и вход. Входът *abc* няма да може да бъде разпознат, тъй като няма легален преход от състояние q^O на входа *a*. Дори ако автоматът позволи първоначално *a*, със сигурност щеше да се провали на *c*, тъй като *c* дори не е в азбуката на овчия език. Може да се мисли за тези „празни" елементи в таблицата, сякаш всички те сочат към едно „празно" състояние, което би могло да се нарече състояние на отказ или състояние на потъване. В известен смисъл тогава би могло да се разглежда всяка машина с празни преходи, сякаш е разширена със състояние на отказ и са начертани всички допълнителни дъги, така че винаги е имало къде да се отиде от всяко състояние при всеки възможен вход.



Официални езици - Модел, който може едновременно да генерира и разпознава всички и само низовете на формален език, действа като дефиниция на формалния език.

Може да се използва същата графика на фиг. 2.10 като автомат за генериране на овчи разговор. Ако се направи би могло да се каже, че автоматът започва от състояние q^O и пресича дъги до нови състояния, отпечатвайки символите, които обозначават всяка дъга, която следва. Когато автоматът стигне до крайното състояние, той спира. Забележка - в състояние 3, автоматът трябва да избира между отпечатването на ! и преминаване към състояние 4, или отпечатване на *a* и връщане към състояние 3.



Нека да се използват следните означения.

Q = q0,q1,q2,q3,…, qn-1 -краен набор от N състояния

Σ - краен брой символи от входяща азбука

q0 – начално състояние

F – набор от крайни състояния F ⊂Q

	Input		
State	b	a	!
0	1	Ø	Ø
1	Ø	2	Ø
2	Ø	3	Ø
3	Ø	3	4
4:	Ø	Ø	Ø

Формалният език е набор от низове, като всеки низ е съставен от символи от краен набор от символи, наречен азбука (същата азбука, използвана по-горе за дефиниране на автомат). Азбуката за овчия език е множеството I = {a,b,!}. При даден модел m (като конкретен FSA), може да се използва L(m), за означаване „формалният език, характеризиран с съответното състояние". Така че формалният език, дефиниран от овчия език за автомат m на фиг. 2.10 е безкрайното множество:

L(m) = {баа!,бааа!,баааа!,бааааа!,баааааа!,...}

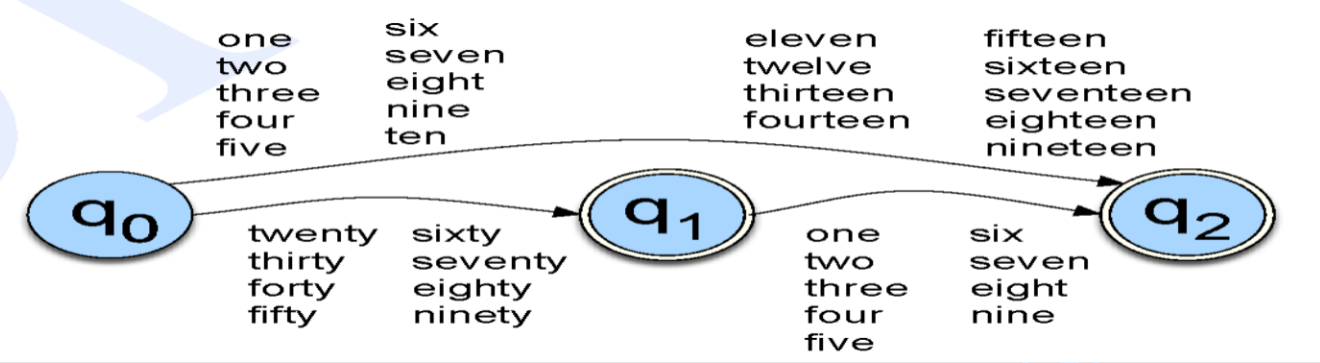
Полезността на автомата за дефиниране на език е, че той може да изрази безкраен набор (като този по-горе) в затворена форма. Официалните езици не са същите като естествените езици, които са видовете езици, които говорят хората.

Всъщност един формален език може изобщо да не прилича на истински език (напр. формален език може да се използва за моделиране на различните състояния на машина за газирани напитки). Но често използваме формален език, за да се моделира част от естествен език, като части от фонологията, морфологията или синтаксиса. Терминът **генеративна граматика** понякога се използва в лингвистиката за означаване на граматика на формален език; произходът на термина е това използване на автомат за дефиниране на език чрез генериране на всички възможни низове.

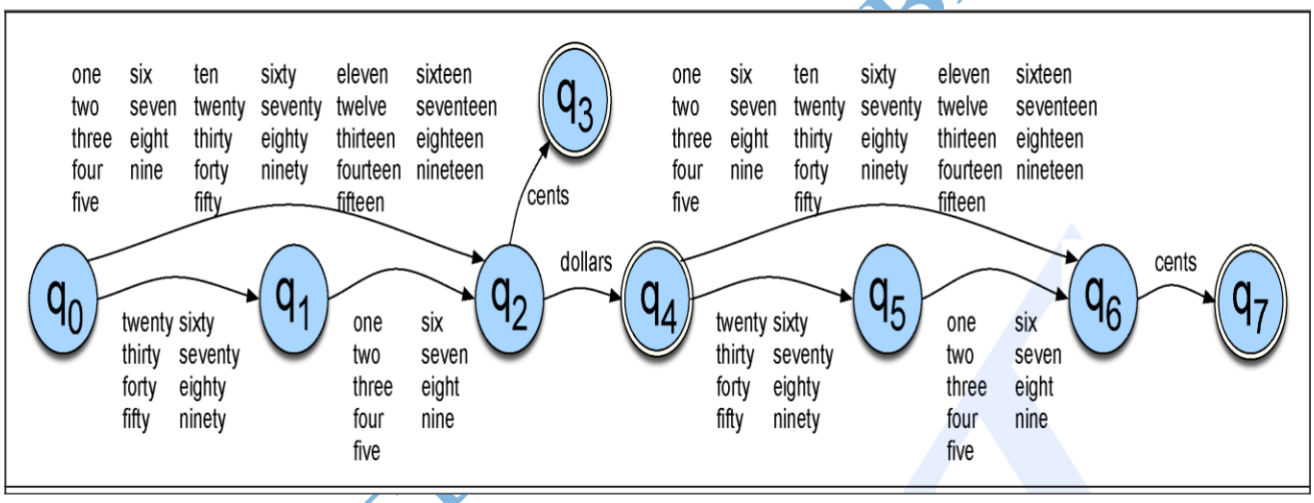
Пример 2

В предишните примери официална азбука се състоеше от букви; но може да има и азбука от по-високо ниво, състояща се от думи. По този начин може да се напишат автомати с крайни състояния, които моделират факти за комбинации от думи. Например иска се автомат, който моделира подчастта на английски, занимаваща се с парични суми. Такъв формален език би моделирал подгрупата на английския, състояща се от фрази като десет цента, три долара, един долар тридесет и пет цента и т.н.

Можем да се разбие, като първо се изгради само автомата, който да отчита числата от 1 до 99, тъй като ще трябват и при работа с центове.



Вече може да се добавят центове и долари към автомата и той би изглеждал както е показан на следващата фигура.



Сега при добавяне на граматика за различни суми в долари; включително по-големи числа като сто хиляди. Трябва също така да се отчете, че съществителните като цент и долараса в единствено число, когато е подходящо (един цент, един долар), и в множествено число, когато е подходящо (десет цента, два долара).