AUTOMATA AND COMPILER

محور الاوتومات و المترجمات

By Eng. Joud Khattab

Content

- Overview.
- Automate.
- Regular Expression.
- Context Free Grammars.
- Compiler.
- MCQ.

AUTOMATA

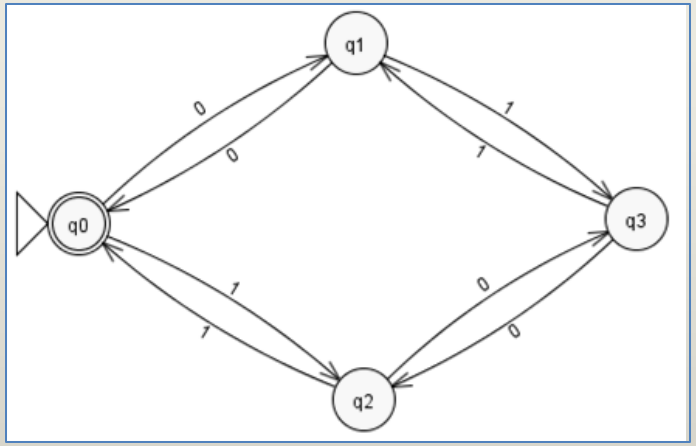
Automata

- أمثلة عن الأوتومات:
  - الحاسب الموجود في آلة بيع القهوة.
  - الحاسب الموجود في الصراف الآلي .ATM
    - في الصراف الآلي نضع البطاقة أولاً ومن ثم نستخدم لوحة المفاتيح للاستعلام عن الرصيد أو لسحب مبلغ معين أو...إلخ.
    - أي يكون لدينا عدد من الأحداث بتسلسل معين ولكن يوجد اختلاف في تسلسلات الأحداث فالتسلسل غير ثابت وتسلسل الأحداث هذا الذي يتم على المعلومات أو على حركة المعلومات هو الحسابات واللغة التي يتعرف عليها الأتومات الخاص بهذا النظام،
    - بالتالي ليس من الضروري السير بنفس التسلسل في كل مرة فمثلاً بعد ادخال البطاقة للصراف ليس من المفروض دائماً اختيار الحساب الجاري بل ممكن اختيار حساب التوفير فكلا العمليتين مقبولتان بالنسبة لنا وبالتالي يمكن الذهاب بفرعين مختلفين في الأتومات.

Automata

- التعريف الرياضي للأوتومات:

- M = (Q , Σ , δ , q0 , F) were
  - Q = { q0 , q1 , q2 , q3 }
  - Σ = { 0 , 1 }
  - q0 = q0 initial state
  - F = { q0 }



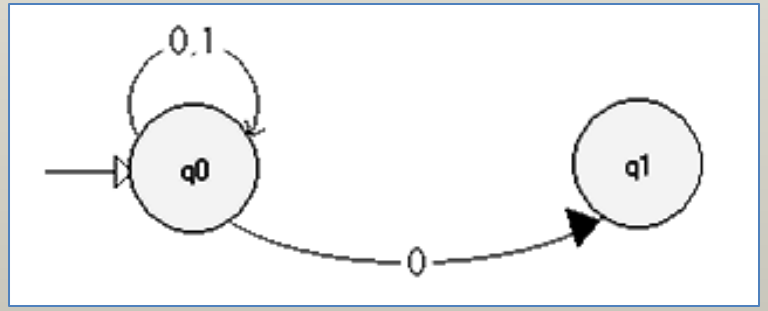
- المثال يقبل جميع السلاسل التي يكون فيها عدد الاصفار والواحدات زوجي.

Different Kinds of Automata (Chomsky) (أنواع اللغات وهرميتها لتوصيف القواعد)

1. Regular Languages:
   - Definition Tool: **Finite Automata** (الأوتومات المنتهي)
   - Memory Type: no temporary memory
   - Ex: ATM, Search.
2. Context-Free Languages:
   - Definition Tool: **Push down Automata (PDA)** (الأوتومات ذات المكدس)
   - Memory Type: stack
   - Ex: Syntax recognition of a specific language, Identify the conditional (IF)
3. Recursive Languages:
   - Definition Tool: **Turing Machines**
   - Memory Type: random access memory (RAM)
   - Ex: Recognize all languages regardless their form

Finite Automata

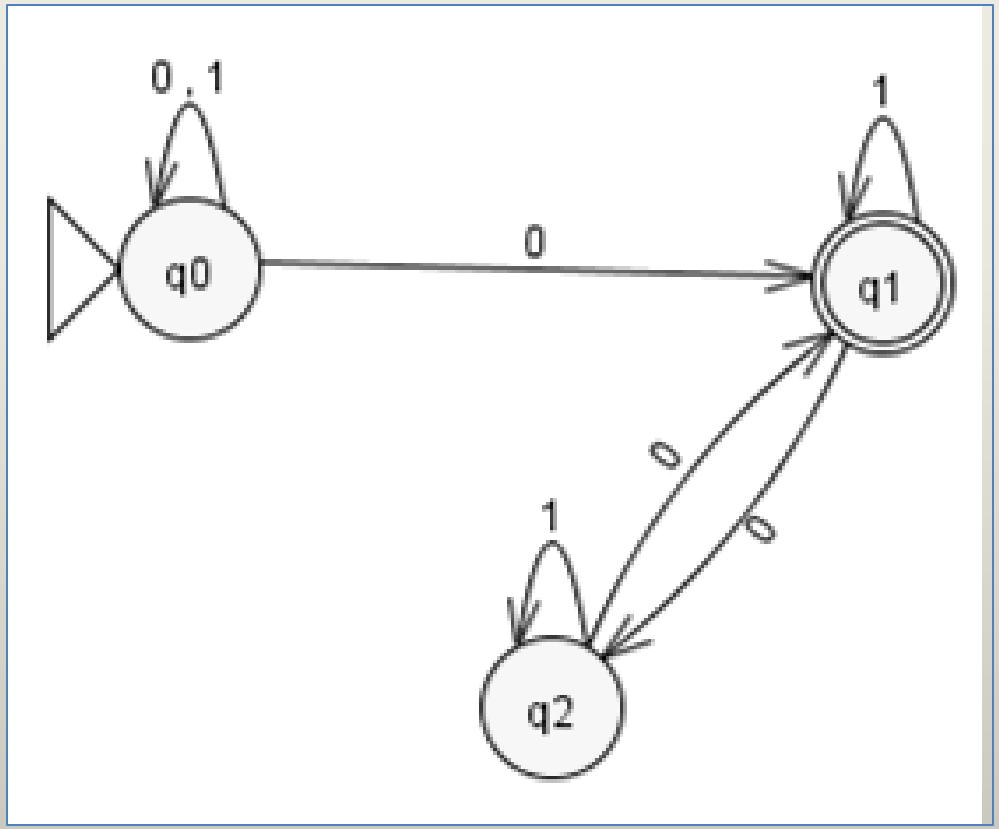
- أنواع الاوتومات المنتهي: Finite Automata
  - الاوتومات المنتهي الحتمي: (Deterministic finite automata DFA)
    - حيث يحقق هذا الاوتومات ان الانتقال من حالة وحيدة ومن اجل رمز معين يكون الى حالة وحيدة محددة.
  - الاوتومات المنتهي اللاحتمي: (Non deterministic finite automata NFA)
    - وهو الذي لا يحقق شرط الاوتومات السابق.
    - أي نستطيع الانتقال الى اكثر من حالة بنفس الرمز.



Finite Automata

- ملاحظات:
  - من اجل اوتومات من نوع DFA يمكن بناء خوارزمية القبول لسلسلة ما.
    - أي الخوارزمية التي نستطيع من خلالها معرفة اذا ما كانت سلسلة ما تنتمي للاوتومات ام لا.
  - اما في حالة NFA فلا يمكن بناء خوارزمية قبول.
    - يمكن بناء خوارزمية قبول تراجعية ولكنها مكلفة جدا.
  - يبرهن على NFA بتحويلها ل DFA مكافئ لها.
  - لكل NFA هنالك DFA مكافئ له.

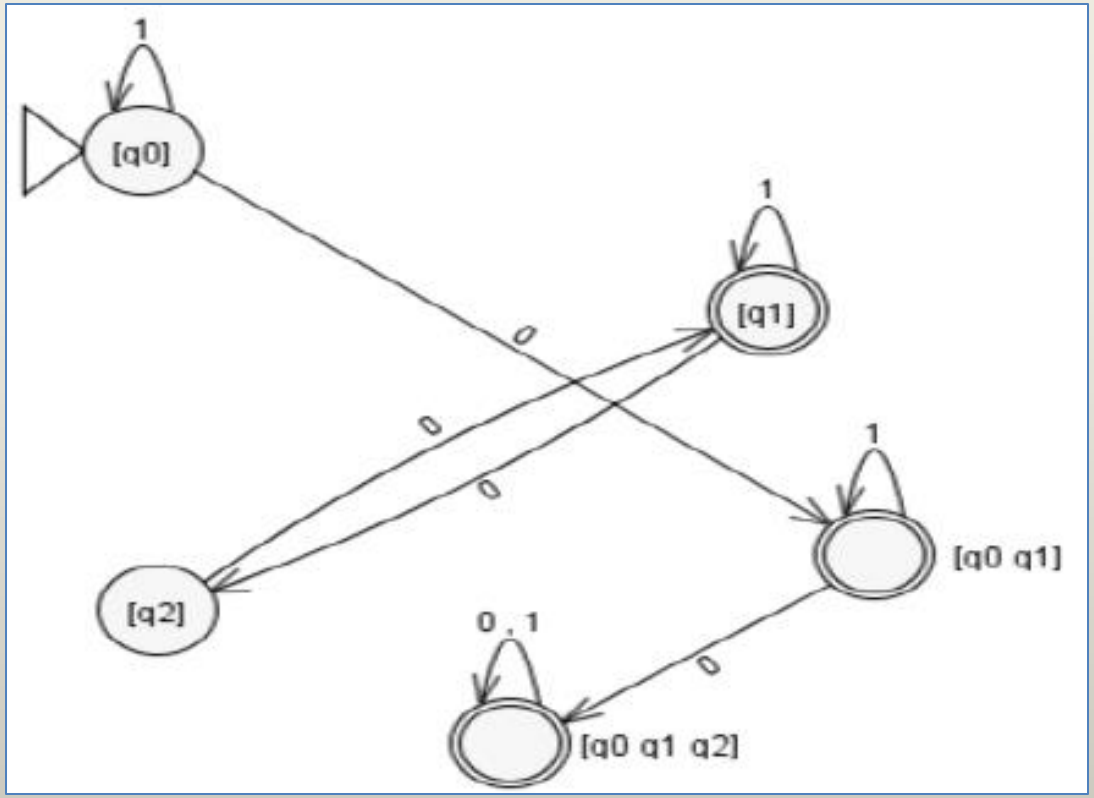
Finite Automata
From NFA To DFA



Finite Automata
From NFA To DFA

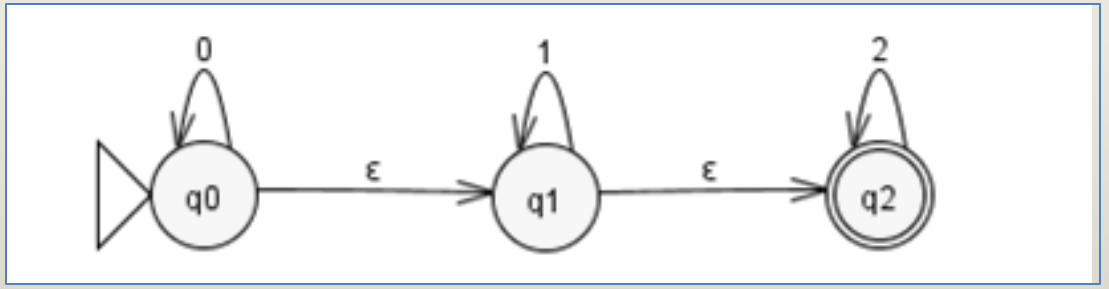
δ	0	1
$[q_0]$	$[q_0, q_1]$	$[q_0]$
$[q_1]$	$[q_2]$	$[q_1]$
$[q_2]$	$[q_1]$	$[q_2]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_1]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$

Finite Automata
From NFA To DFA



Finite Automata
ε-NFA

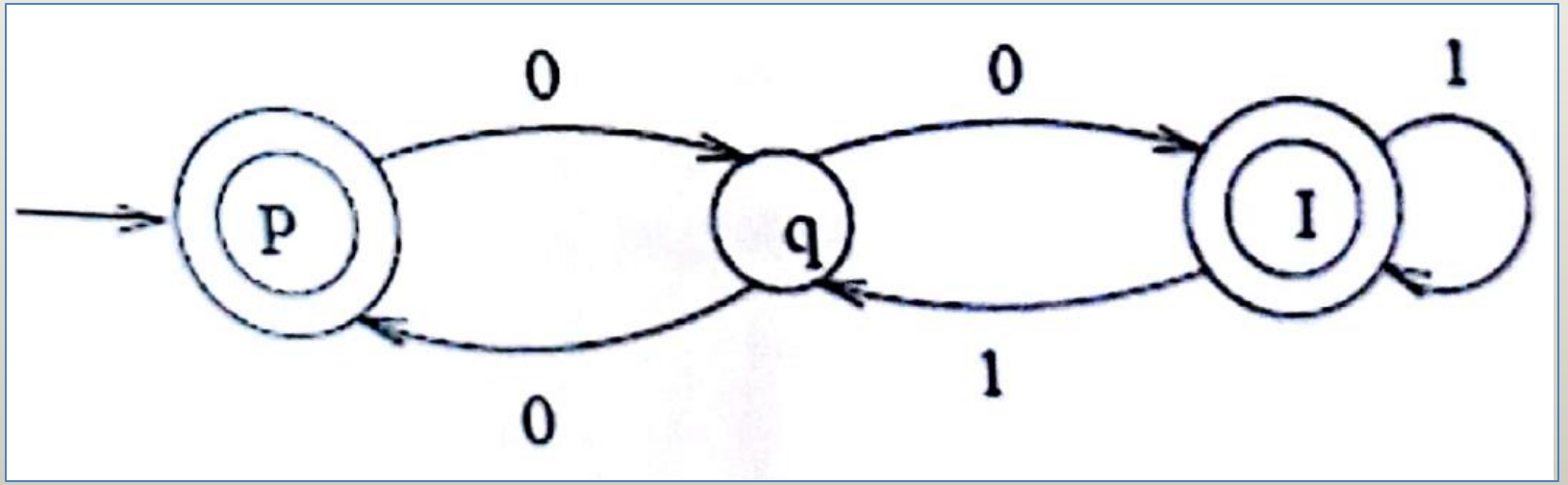
- الاوتومات اللاحتمي مع الانتقال ابسيلون
  - مثال:



  - نعتبر ان الابجدية هي:
    - 0 1 2 ε

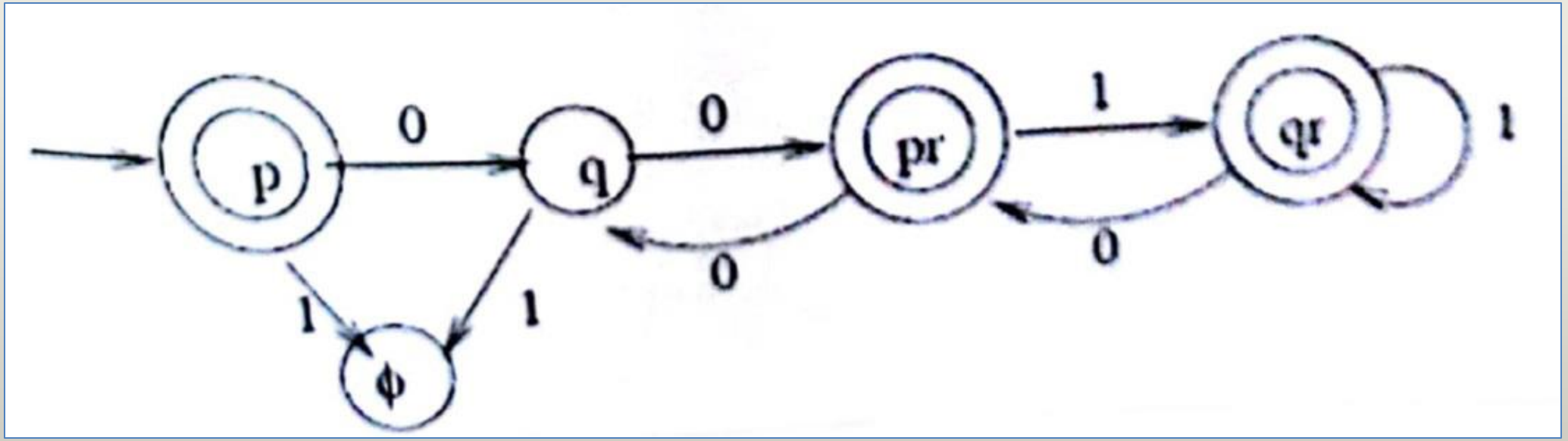
Question 1

- أعط الأوتومات المنتهي الحتمي المكافئ للأوتومات المنتهي اللاحتمي التالي:



Question 1

- الجواب:



REGULAR LANGUAGE

اللغات المنتظمة

Regular Language

- التعبير النظامي:
  - هو عبارة عن طريقة مبسطة مختصرة لتمثيل الاوتومات الذي يوصف لغة معينة ويعتمد على مجموعة من الترامیز.
- اللغة المنتظمة:
  - هي كل لغة يمكن تمثيلها ب RE أي عمليا تكون ممثلة ب DFA.

Regular Language

- امثلة:
  - اذا كانت لدينا الابجدية $\sum = \{a\}$ نستطيع بناء العديد من التعابير النظامية:
    - a: يكافئ السلسلة المؤلفة من حرف واحد.
    - ε: يكافئ السلسلة الخالية.
    - a^*: يكافئ السلسلة الخالية او السلسلة المؤلفة من تكرار a عدد من المرات أكبر أو يساوي الواحد.
    - a^+: يكافئ السلسلة المؤلفة من تكرار a عدد من المرات أكبر أو يساوي الواحد.
    - a^n: يكافئ السلسلة المؤلفة من تكرار a (n) مرة فقط.
  - اذا كانت لدينا الأبجدية $\sum = \{a,b\}$ نستطيع بناء العديد من التعابير المنتظمة:
    - a^*b^*: يكافئ السلسلة الناتجة عن عملية concatenation أي (.) بين تعبيرين نظاميين.

Regular Language

- L1 = { a^*b^* : a,b $\in \sum$ }
  - L1 is regular language.

- L2 = { a^nb^n : a,b $\in \sum$, n>=0}
  - L2 is not regular language.

- نلاحظ ان اللغة L2 تحتاج الى ذاكرة، وذلك لأنه حتى تكون السلسة مقبولة يجب أن يرى b بنفس عدد مرات ورود a.

Regular Language

- L1, L2: Regular Languages then:
  - L = L1 . L2 = { X.W were X ∈ L1 , W ∈ L2 } is Regular Language
  - L1 ∪ L2 is Regular Language
  - $\overline{L1}$, $\overline{L2}$ is Regular Language
  - L1 ∩ L2 is Regular Language

Regular Expressions Exercise 1

- Suppose the only characters are 0 and 1.
- A regular expression for strings containing 00 as a substring:

(0 | 1)* 00 (0 | 1)*

11011100101

0000

11111011110011111

Regular Expressions Exercise 2

- Suppose the only characters are 0 and 1.
- A regular expression for strings of length exactly four:

(0|1)(0|1)(0|1)(0|1)

(0|1){4}

0000

1010

1111

1000

Regular Expressions Exercise 3

- Suppose the only characters are 0 and 1.
- A regular expression for strings that contain at most one zero:

1*(0 | ε)1*

1*0?1*

11110111

111111

0111

0

Regular Expressions Exercise 4

- Suppose that our alphabet is all ASCII characters.
- A regular expression for even numbers:

(+|-)?(0|1|2|3|4|5|6|7|8|9)*(0|2|4|6|8)

(+|-)?[0123456789]*[02468]

(+|-)?[0-9]*[02468]

42

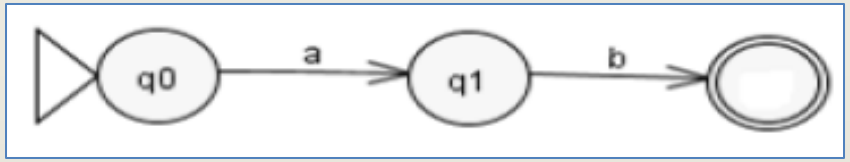
+1370

-3248

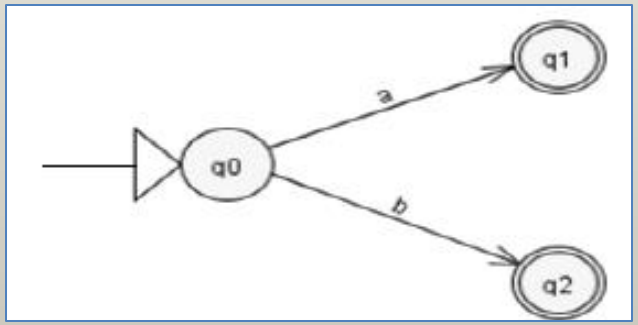
-9999912

Regular Language & Automata

- التعبير (ab) يمكن مقابلته بالاوتومات:

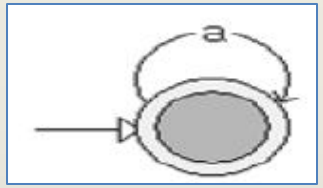


- التعبير (a + b) يمكن مقابلته بالاوتومات:

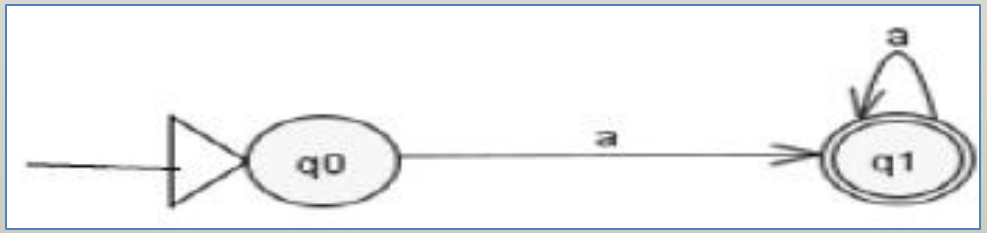


Regular Language & Automata

- التعبير (a^*) يمكن مقابلته بالاوتومات:



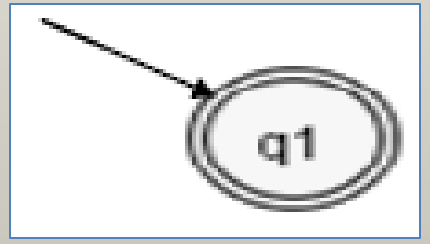
- التعبير (a^+) يمكن مقابلته بالاوتومات:



Regular Language & Automata

- التعبير (θ) تعبر عن عدم وجود طريق (أي مجموعة الحلول هي مجموعة فارغة):

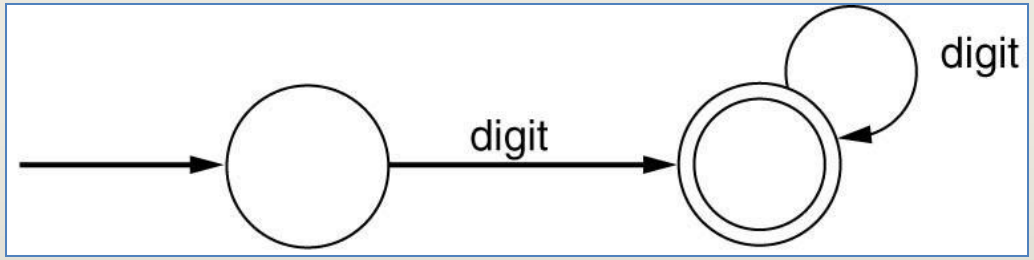
- التعبير (ε) تعبر عن وجود طريق مفتوح ننتقل عبره دون الحاجة لوجود أي رمز من رموز الابجدية:



Regular Language & Automata Example numbers

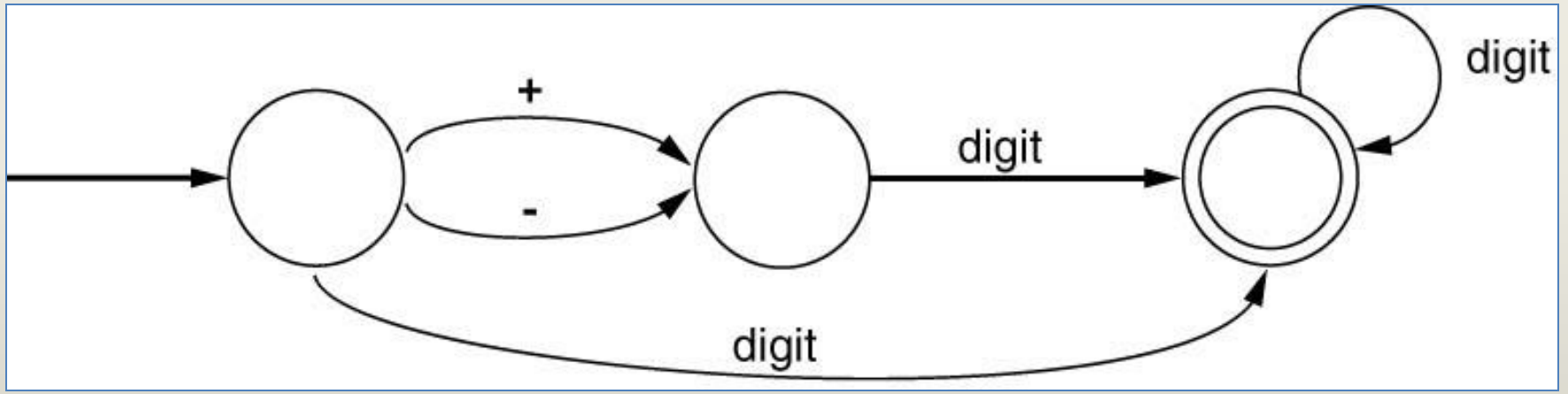
- Draw a DFA equivalent to the following regular expression:
  1) digit = [0-9]
  2) nat = digit+
  3) signedNat = (+|-)? nat
  4) number = signedNat ("." nat)? (E signedNat)?

Regular Language & Automata Example numbers



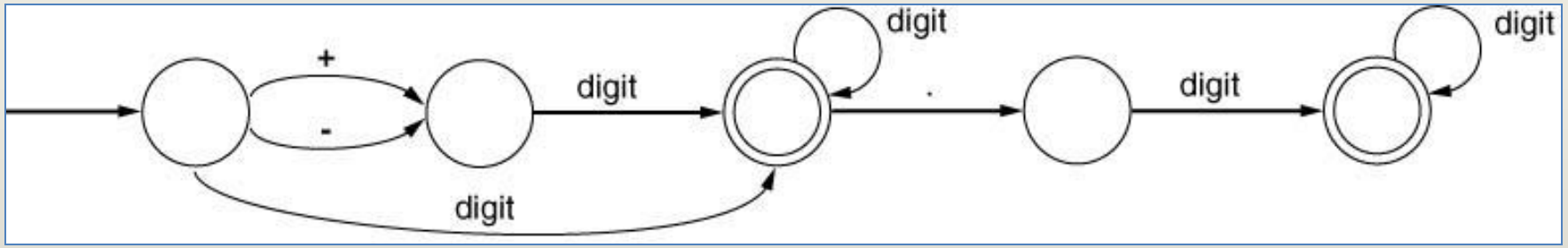
nat = digit+

Regular Language & Automata Example numbers



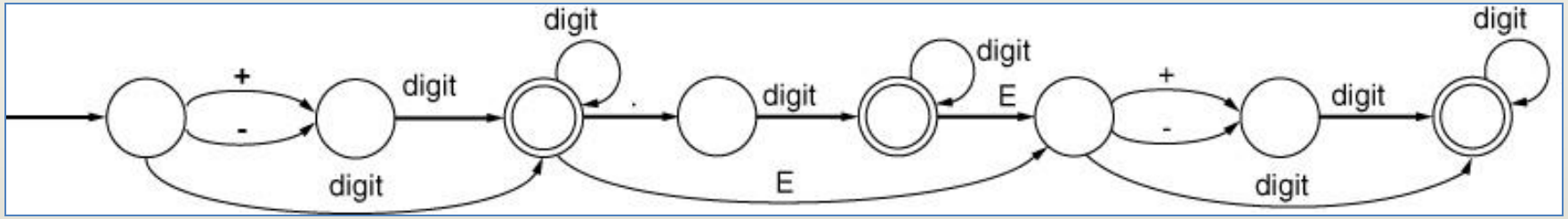
signedNat = (+|-)? nat

Regular Language & Automata Example numbers



signedNat ("." nat)?

Regular Language & Automata Example numbers



number = signedNat ("." nat)? (E signedNat)?

Question 1

- ما هي اللغات التي توصفها التعابير المنتظمة التالية المعرفة على الأبجدية {a,b}:
  - $a\ (a + b)^*\ b$
  - $aab\ (aa|bb)^*$
  - $(aa)^*\ a$

Question 1

- ما هي اللغات التي توصفها التعابير المنتظمة التالية المعرفة على الأبجدية {a,b}:
  - $a\ (a + b)^*\ b$
  - $aab\ (aa|bb)^*$
  - $(aa)^*\ a$
- الجواب:
  - اللغة التي تبدأ جميع كلماتها بالحرف a وتنتهي بالحرف b
  - اللغة التي تبدأ جميع كلماتها بالسلسلة aab ومن ثم يليها تكرار واحد على الأقل لثنائية aa أو من bb
  - اللغة التي لا تحتوي كلماتها إلا على حرف a ويكون طول كلماتها مفردا

Question 2

- ما هي التعابير المنتظمة التي يمكن أن تعرف اللغات التالية:
  - اللغة على الأبجدية {a,b,c} والتي تبدأ جميع كلماتها بالحرف a
  - الأعداد الصحيحة من مضاعفات 5

Question 2

- ما هي التعابير المنتظمة التي يمكن أن تعرف اللغات التالية:
  - اللغة على الأبجدية {a,b,c} والتي تبدأ جميع كلماتها بالحرف a
  - الأعداد الصحيحة من مضاعفات 5
- الجواب:
  - a (a + b + c)*
  - (0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9)* 5

CONTEXT FREE GRAMMARS

اللغات عديمة السياق (خارج السياق)

Context Free Grammars
(اللغات عديمة السياق)

- مفهوم النموذج القواعدي:
  - هو مفهوم أكثر تطورا من مفهوم ال DFA الذي يعبر عن نمط بسيط من اللغات هو اللغات المنتظمة ولا يستطيع تعدي هذا النمط.
  - وبالتالي يعد ال DFA قاصرا عن تمثيل اللغات التي تحمل ذاكرة وبالتالي نلجا الى النموذج القواعدي لتمثيل هذا النوع من اللغات.
- اذا اللغات المنتظمة هو نوع جديد من الاوتومات، وهو مزود بذاكرة يستطيع تمثيل لغات تحتاج الى ذاكرة مثل السلاسل من النمط (a^nb^n) حيث ان هذا الاوتومات يتذكر اعداد الرموز (a,b) التي مرت عليه.

Context Free Grammars (اللغات عديمة السياق)

- Non Terminal Symbols (V):
  - تمثل مجموعة من الحروف البسيطة التي تساعد على عملية توليد السلاسل (أي مجموعة من التوابع المساعدة).
- Terminal Symbols (T):
  - تؤلف الابجدية التي تتكون من الحروف التي تتألف منها سلاسل اللغة.
- Production (P):
  - هو قواعد توليد اللغة.
- Starting Symbol (S):
  - الرمز البدائي الذي تبدأ منه في عملية توليد اللغة.

Context Free Grammars Example 1

- اوجد النموذج القواعدي للغة:
- الحل:

  - $L = \{ a^n b^n : a,b \in \sum , n>=0\}$
  - L = {V , T , S , P} where
    - S → a S b (1)
    - S → a b (2)
  - V = {S}
  - S = {S}
  - T = {a,b}

Context Free Grammars
Example 2

- ليكن لدينا اللغة L الممثلة للتعابير الحسابية، نستطيع تعريف هذه اللغة بالشكل القواعدي التالي:

  - Exp → Exp Op Exp | - (Exp) | id
  - Op → + | - | * | /

- الحل:

  - V = {Exp, Op}
  - T = { + , - , / , id , (,) }
  - S = {Exp}

Context Free Grammars
Left Most Derivation

- Exp → Exp Op Exp | - (Exp) | **id**
- Op → + | - | * | /

- كيف نشتق السلسلة التالية: id + id * id
  - نعوض ال (None Terminal Symbol) الموجود في اقصى اليسار أولا وهكذا..
  - يتم ذلك عبر المراحل التالية:

- Exp → Exp Op Exp
  → **id** Op Exp
  → **id +** Exp
  → **id +** Exp Op Exp
  → **id + id** Op Exp
  → **id + id *** Exp
  → **id + id * id**

Context Free Grammars
Right Most Derivation

- Exp → Exp Op Exp | - (Exp) | **id**
- Op → **+** | **-** | ***** | **/**
- هنالك خوارزمية أخرى في الاشتقاق ينطوي مبدؤها على تعويض ال (None Terminal Symbol) الموجود في اقصى اليمين أولا وهكذا...
- Exp → Exp Op Exp
  → Exp Op **id**
  → Exp ***** **id**
  → Exp Op Exp ***** **id**
  → Exp Op **id** ***** **id**
  → Exp **+** **id** ***** **id**
  → **id** **+** **id** ***** **id**

Context Free Grammars Ambiguity

- نلاحظ على ان اللغة في المثال السابق هي لغة غامضة
  - وذلك لأنه من أجل نفس خوارزمية الاشتقاق هنالك طريقتين لاشتقاق نفس السلسلة وهذا الغموض لا يتناسب مع قابلية الاوتومات للإسقاط البرمجي.

COMPILER

المترجم

Compiler Definition

- تعريف المترجم:
  - يستخدم أي مبرمج أداة ضرورية جدا في عملية البرمجة، ندعوها المترجم.
  - برنامج حاسوبي يترجم النص البرمجي الذي نكتبه بلغة برمجية عالية المستوى (C, Pascal, C++, C#, Java, ...) الى مجموعة تعليمات قابلة للتنفيذ من قبل الحاسوب.
  - تكون هذه التعليمات التنفيذية مكتوبة بلغة منخفضة المستوى سواء كانت لغة ثنائية مؤلفة من اصفار و واحدات او لغة تجميع.
  - يمكن أيضا بناء مترجمات من لغة عالية المستوى الى لغة أخرى عالية المستوى. (تغيير البيئة البرمجية)
- تعريف البرنامج المصدري (Source Program):
  - مجموعة النصوص البرمجية التي تؤلف برنامج حاسوبي.

Compiler Definition

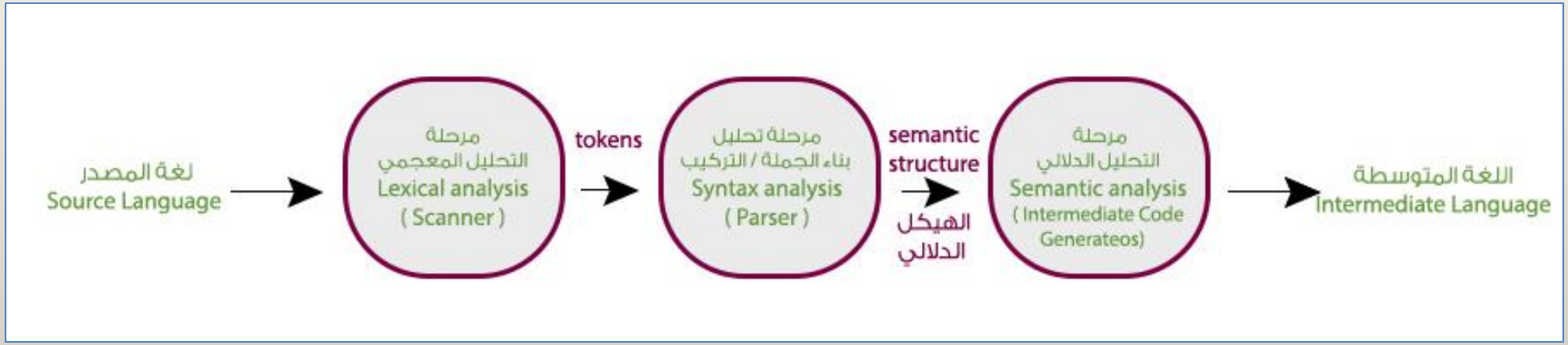
- إن عملية بناء المترجم تتعلق بعنصرين اثنين بأن واحد:
  - لغة البرمجة المصدرية عالية المستوى الذي يستخدمها المبرمج.
  - نظام التشغيل الذي سيجري تشغيل البرنامج عليه.
    - مثال:
      - يختلف مترجم لغة C++ الذي يعمل على نظام windows عن مترجم لغة C++ الذي يعمل على نظام Linux، نظرا لضرورة توليد تعليمات تشغيل تنفيذية مختلفة في هاتين الحالتين، بالرغم من اننا نتكلم عن نفس اللغة.
      - يختلف مترجم لغة C++ عن مترجم لغة Pascal حتى ولو كان المترجمان يعملان على نظام Windows، نظرا اننا نترجم لغتين برمجيتين مختلفتين.

Compiler Structure

Compiler Structure

- تتألف عملية الترجمة من مرحلتين أساسيتين:
  1. مرحلة التحليل (Analysis Phase):
     - التي يجري فيها تقسيم النص البرمجي الى كلمات وجمل والتأكد من صحتها ودلالاتها.
  2. مرحلة التركيب (Synthesis Phase):
     - التي يجري فيها تركيب نص برمجي جديد بنفس دلالة النص المصدري ولكن بلغة أخرى يفهمها نظام التشغيل.

Compiler Structure
Analysis Phase (مرحلة التحليل)



Compiler Structure
Analysis Phase (مرحلة التحليل)

- مرحلة التحليل تحوي ثلاث خطوات:

1. مرحلة التحليل المعجمي (Lexical Analysis - Scanner):
   - مهمته أن يقرأ الدخل input ويحللها الى tokens وكل من هذه الكلمات تمثل جزء محدد من اللغة سواء أكان متغير ام من الكلمات الثابتة في اللغة (reserved word) ويقوم ايضاً بمسح المسافات وحفظ الـ tokens في جدول الرموز (table symbol).
2. مرحلة بناء الجملة (Syntax Analysis - Parser):
   - مهمته ان يأخذ الـ tokens الناتجة عن مرحلة (Lexical Analysis) ويكونها في جمل برمجية ويختبر صحتها على أساس قواعد اللغة.
   - هذه المرحلة ينتج عنها هيكل دلالي يدعى (semantic structure) وتكون الجمل البرمجية على شكل شجري يسمى ب (parser tree).
3. مرحلة التحليل الدلالي (Semantic Analysis – Intermediate Code Generator):
   - في هذه المرحلة يتم التحقق من الأخطاء المتعلقة بالمنطق، مثل صحة اسناد القيمة لنوع المتغير وغيرها.

Compiler Structure
Analysis Phase (مرحلة التحليل)

- مراحل عمل ال compiler:
  - التحليل اللفظي:
    - أي معرفة اذا ما كانت الكلمة موجودة في لغة البرمجة ام لا.
  - التحليل القواعدي:
    - أي التعرف على ال syntax، أي قواعد كتابة اللغة كما في اللغات المحكية.
    - مثال:
      - في اللغة العربية يجب ان يأتي (فعل – فاعل – مفعول به).
      - في لغات البرمجة: if (statement)
  - التحليل الدلالي:
    - اي ان الجملة من الممكن ان تكون صحيحة قواعديا و لكن ليس دلاليا.
    - مثال:
      - اكل الولد التفاحة.
      - اكل الولد المقعد.

Natural Language Translation

Teh mungry teamher are eak a mook and seading an banban

Check for and correct
LEXICAL (spelling) errors

The hungry teacher are eat a book and reading an banana

Check for and correct
SYNTACTIC (grammar) errors

The hungry teacher is eating a book and reading a banana

Check for and correct
SEMANTIC (meaning) errors

The hungry teacher is eating a banana and reading a book

Translate

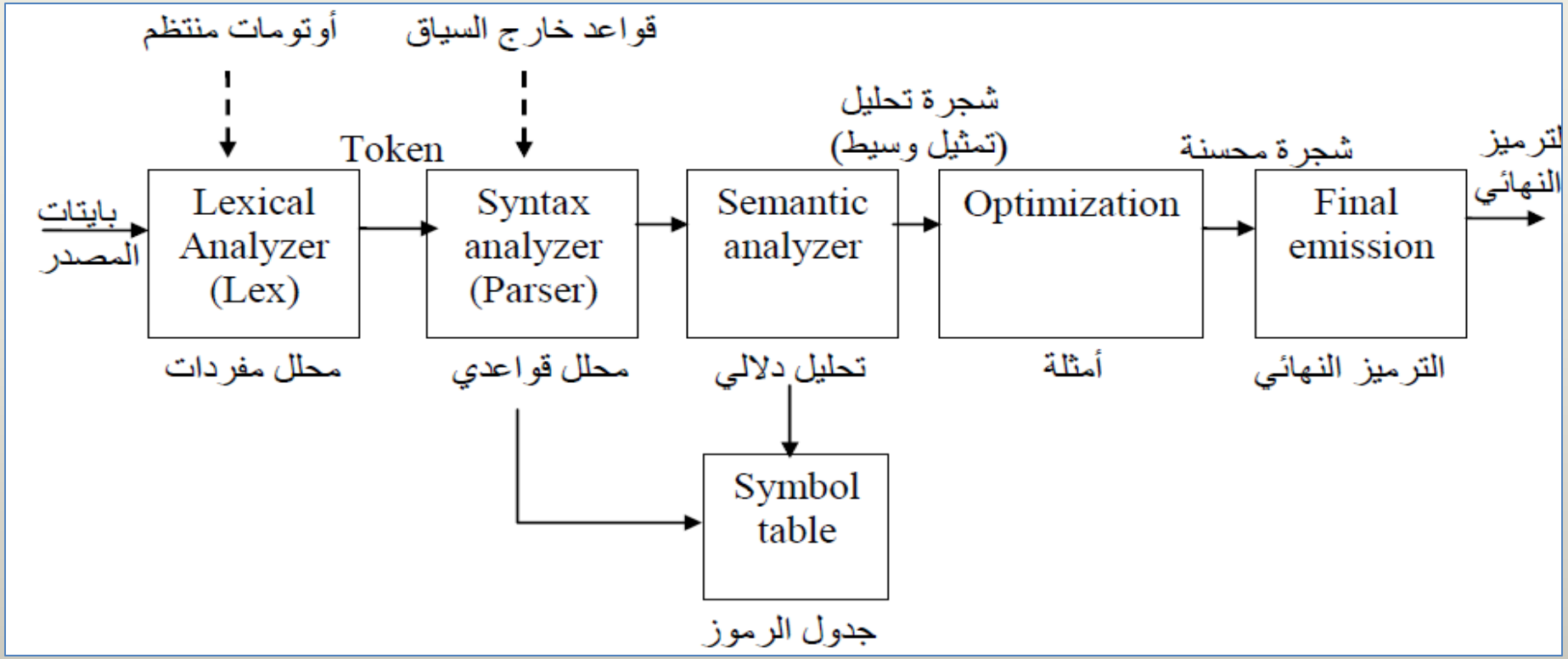
الأستاذ الجائع يأكل موزة ويقرأ كتاب

Compiler Structure
Synthesis Phase (مرحلة التركيب)

- في هذه المرحلة يتم تحويل اللغة المتوسطة (Intermediate Language) الى لغة تفهمها الآلة (Machine Language) ويتم ذلك على النحو التالي:

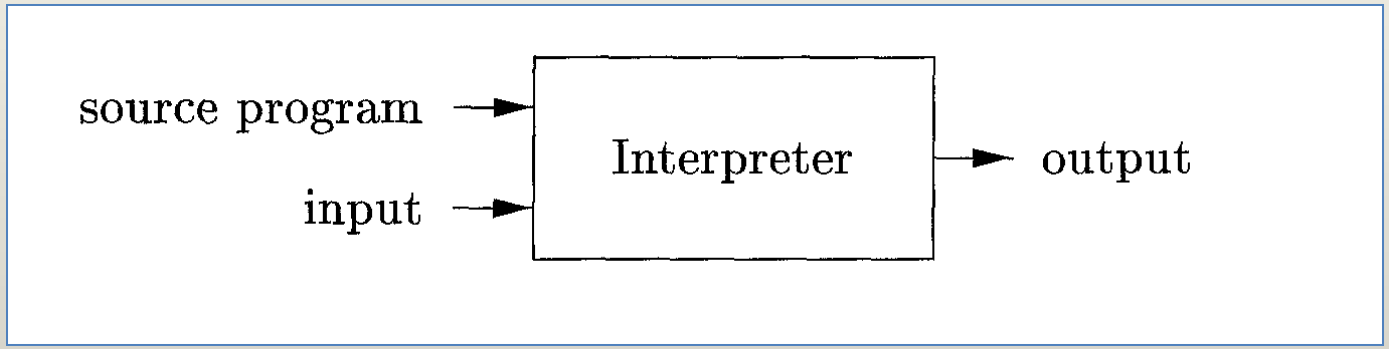
1. مرحلة تحسين الأكواد (Code Optimization):
   - هذه الخطوة تتولى مسألة تحسين الكود وابعاد التكرار وتطوير البرنامج والتأكد بأن يكون البرنامج في أحسن حالاته و هذه الخطوة هي التي تميز مترجم عن مترجم آخر.
2. مرحلة مولّد الأكواد (Code Generation):
   - هنا يتم تحويل الكود بشكل نهائي الى شكل تفهمه الآلة.

Compiler Structure



Interpreter (المفسر)

- لا يحول البرنامج ولكن ينفذه خطوة خطوة.



Compiler Kinds

- ما معنى multi-pass compiler و one pass compiler؟
  - One pass compiler:
    - يتم عمل التحليل المفرداتي في نفس اللحظة التي يتم فيها التجميع والتحليل القواعدي ثم الانتقال مباشرة الى التحليل الدلالي ثم عملية المسح هذه من الأعلى للاسفل.
    - المرور الوحيد يعني المرور على التحليل اللفظي والقواعدي والدلالي يعني بقراءة وحدة.
  - Multi pass compiler:
    - يتم مسح البرنامج اكتر من مرة وتتطلب وقت اكتر.

Parse Tree Example
(number – number) * number

exp ⇒ exp op exp

⇒ exp op **number**

⇒ exp *** number**

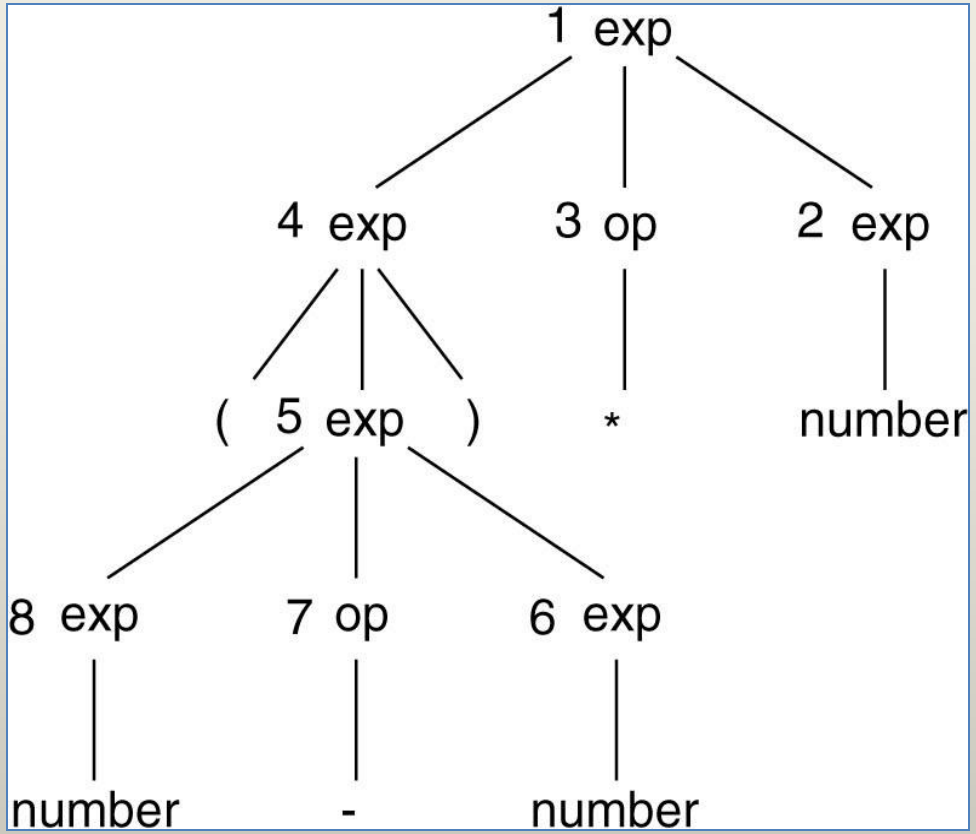
⇒**(** exp **) * number**

⇒ **(** exp op exp **) * number**

⇒ **(** exp op **number) * number**

⇒ **(** exp **– number) * number**

⇒ **(number – number) * number**



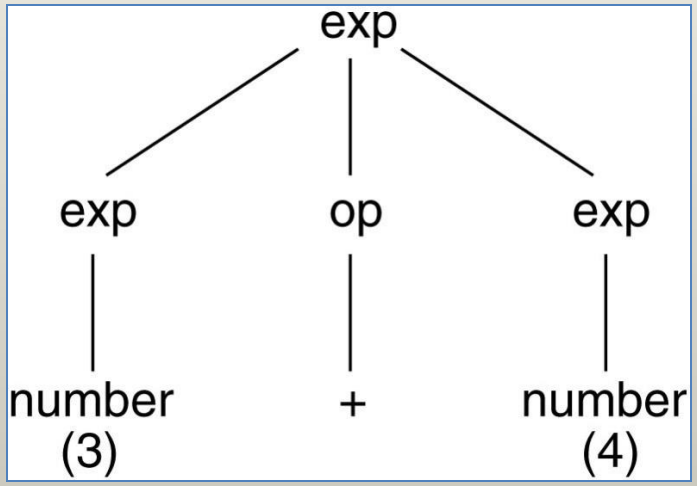
Abstract Syntax Trees

- A parse tree contains much more information than is necessary for a compiler to produce executable code.
- Syntax trees contain just the information needed for translation.

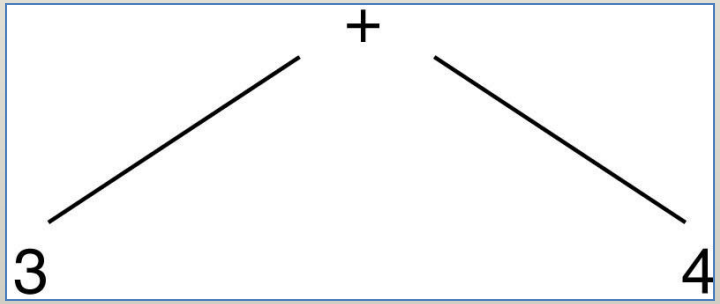
Abstract Syntax Trees Example

- Show the Parse and Syntax Trees of the derivation of the string **3 + 4** using the grammar.

exp → exp op exp | (exp) | **number**
op → **+** | **-** | ***



Parse Tree



Syntax Tree

Left-Recursion Removal

$A \rightarrow A\ \alpha \mid \beta$

β does not begin with A

$A \rightarrow \beta$ is the base case
$A \rightarrow A\ \alpha$ is the recursive case

To remove the recursion we re-write the rule :

$A \rightarrow \beta\ A'$
$A' \rightarrow \alpha\ A' \mid \varepsilon$

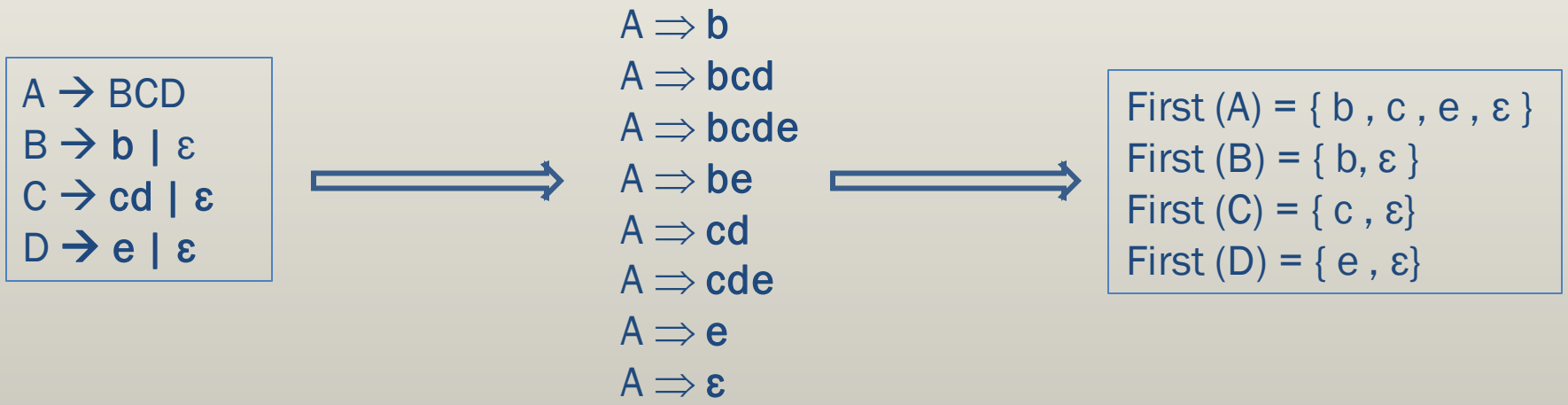
Example

exp → exp addop term | term

exp → term exp'
exp' → addop term exp' | ε

First Sets
(مجموعة الرموز الأولى)

- The First Set of a nonterminal A is the set of terminals that may appear as the first symbols in a string derived from A.



First Sets
(مجموعة الرموز الأولى)

- The First Set of a nonterminal A is the set of terminals that may appear as the first symbols in a string derived from A.

Let X be a grammar symbol (a terminal or nonterminal) or ε. Then the set **First (X)** consisting of terminals, and possibly ε, is defined as follows:

1. If X is a terminal or ε, then First (X) = {X}
2. If X is a nonterminal, then
    1. for each production choice X → X_1 X_2 … X_n , First (X) contains First (X_1) – {ε}.
    2. and if for some i < n, all the sets First (X_1), …, First (X_i) contain ε, then First (X) contains First (X_{i+1}) – {ε}.
    3. and if all the sets First (X_1), …, First (X_n) contain ε, then First (X) also contains ε.

First Sets Example

exp → exp addop term | term
addop → + | -
term → term mulop factor | factor
mulop → *
factor → (exp) | **number**

Grammar Rule	Pass 1	Pass 2	Pass 3
exp → exp addop term			
exp → term			First (exp) = { (, number}
addop → +	First (addop) = {+}		
addop → -	First (adop) = {+,-}		
term → term mulop factor			
term → factor		First (trem) = { (, number}	
mulop → *	First (mulop) = {*}		
factor → (exp)	First (factor) = { (}		
factor → number	First (factor) = { (, number}		

Follow Sets
(مجموعة الرموز اللاحقة)

- The Follow Set of a nonterminal A is the set of terminals that may appear after A in a string derived from the start symbol.

A → BCD
B → **b** | ε
C → **cd** | **ε**
D → **e** | **ε**

⟹

Follow (A) = $
Follow (B) = First (CD) – {ε} + Follow (A) = {c, e, ε} – {ε} + {$} = {c, e, $}
Follow (C) = First (D) – {ε} + Follow (A) = {$, e}
Follow (D) = Follow (A) = {$}

Follow Sets
(مجموعة الرموز اللاحقة)

- The Follow Set of a nonterminal A is the set of terminals that may appear after A in a string derived from the start symbol.

Given a nonterminal A, the set **Follow(A)**, consisting of terminals, and possibly $, is defined as follows:

1. If A is the start symbol, then $ is in Follow(A)
2. If there is a production B → α A γ, then First (γ) – {ε} is in Follow (A).
3. If there is a production B → α A γ such that ε is in First (γ), then Follow (A) contains Follow (B).

Follow Sets Example

exp → exp addop term | term
addop → + | -
term → term mulop factor | factor
mulop → *
factor → (exp) | **number**

First (exp) = { (, **number**}
First (term) = { (, **number**}
First (factor) = { (, **number**}
First (adop) = {+,-}
First (mulop) = {*}

Grammar Rule	Pass 1	Pass 2
exp → exp addop term	Follow (exp) = {$, + , -} Follow (addop) = { (, number} Follow (term) = {$, + , -}	Follow (term) = { $, + , - , *,) }
exp → term		
term → term mulop factor	Follow (term) = {$,+,-, *} Follow (mulop) = { (, number} Follow (factor) = {$, +, -, *}	Follow (factor) = { $, + , - , *,) }
term → factor		
factor → (exp)	Follow (exp) = { $, + , - ,) }	

MCQ

Question 1

- حدد العبارة الصحيحة:

A. المفسر (interpreter) هو اسم اخر للمترجم (compiler) والاختلاف فقط في المصطلح المستخدم.

B. المفسر هو نمط من المترجمات التي تتم فيها عملية التحقق من الكود البرمجي وتنفيذ للعمليات سطرا سطرا.

C. المفسر هو مترجم بدون مرحلة الأمثلة (Optimization Phase) للكود البرمجي.

D. لا علاقة للمفسرات بالمترجمات نهائيا.

Question 1

- حدد العبارة الصحيحة:

A. المفسر (interpreter) هو اسم اخر للمترجم (compiler) والاختلاف فقط في المصطلح المستخدم.

B. المفسر هو نمط من المترجمات التي تتم فيها عملية التحقق من الكود البرمجي وتنفيذ للعمليات سطرا سطرا.

C. المفسر هو مترجم بدون مرحلة الأمثلة (Optimization Phase) للكود البرمجي.

D. لا علاقة للمفسرات بالمترجمات نهائيا.

Question 2

- اختر الخيار الصحيح فيما يتعلق ب top down parsing عند بناء المترجم:
  A. لازم دائما حساب first دون حساب follow.
  B. لازم حساب follow دون first.
  C. لازم حساب follow واحيانا first.
  D. كل ما سبق خطأ.

Question 2

- اختر الخيار الصحيح فيما يتعلق ب top down parsing عند بناء المترجم:
  A. لازم دائما حساب first دون حساب follow.
  B. لازم حساب follow دون first.
  C. لازم حساب follow واحيانا first.
  D. كل ما سبق خطأ.

Question 3

- في حال استخدمنا القواعد الصرفية (grammer) التالية التي توصف بنية أي عدد صحيح او حقيقي R ضمن لغة برمجة:
- اي صيغة من الصيغ التالية تعتبر صيغة مقبولة للتعبير عن عدد ما وفق القواعد السابقة.

R	→	sign digits frac exp
sign	→	minus | ε
digits	→	digit*
digit	→	{'0'-'9'}
frac	→	(dot digits) | ε
exp	→	(('e'+'E') sign digits) | ε
dot	→	{'.'}
minus	→	{'-'}

A. 2.45E02

B. E1-1.1-

C. 1.1.1

D. E02+1

Question 3

- في حال استخدمنا القواعد الصرفية (grammer) التالية التي توصف بنية أي عدد صحيح او حقيقي R ضمن لغة برمجة:
- اي صيغة من الصيغ التالية تعتبر صيغة مقبولة للتعبير عن عدد ما وفق القواعد السابقة.

R	→	sign digits frac exp
sign	→	minus | ε
digits	→	digit*
digit	→	{'0'-'9'}
frac	→	(dot digits) | ε
exp	→	(('e'+'E') sign digits) | ε
dot	→	{'.'}
minus	→	{'-'}

A. 2.45E02

B. E1-1.1-

C. 1.1.1

D. E02+1

Question 4

- A grammar that produces more than one parse tree for some sentence is called:
  A. Ambiguous.
  B. Unambiguous.
  C. Regular.
  D. None of the mentioned.

Question 4

- A grammar that produces more than one parse tree for some sentence is called:
  A. Ambiguous.
  B. Unambiguous.
  C. Regular.
  D. None of the mentioned.

Question 5

- An intermediate code form is:
  A. Postfix notation.
  B. Syntax trees.
  C. Three address codes.
  D. All of these.

Question 5

- An intermediate code form is:
  A. Postfix notation.
  B. Syntax trees.
  C. Three address codes.
  D. All of these.

Question 6

- Compiler translates the source code to:
  A. Executable code.
  B. Machine code.
  C. Binary code.
  D. Both B and C.

Question 6

- Compiler translates the source code to:
  A. Executable code.
  B. Machine code.
  C. Binary code.
  D. Both B and C.

Question 7

- Which of the following groups is/are token together into semantic structures?
  A. Syntax analyzer.
  B. Intermediate code generation.
  C. Lexical analyzer.
  D. Semantic analyzer.

Question 7

- Which of the following groups is/are token together into semantic structures?
  A. Syntax analyzer.
  B. Intermediate code generation.
  C. Lexical analyzer.
  D. Semantic analyzer.

Question 8

- _____ is a process of finding a parse tree for a string of tokens:
  A. Parsing.
  B. Analyzing.
  C. Recognizing.
  D. Tokenizing.

Question 8

- _____ is a process of finding a parse tree for a string of tokens:
  A. Parsing.
  B. Analyzing.
  C. Recognizing.
  D. Tokenizing.

Question 9

- What is the action of parsing the source program into proper syntactic classes?
  A. Lexical analysis.
  B. Syntax analysis.
  C. General syntax analysis.
  D. Interpretation analysis.

Question 9

- What is the action of parsing the source program into proper syntactic classes?
  A. Lexical analysis.
  B. Syntax analysis.
  C. General syntax analysis.
  D. Interpretation analysis.

Question 10

- Which of the following languages is generated by the given grammar?
- S → a S | b S | ε
  A. $a^n b^m$
  B. $\{a,b\}^*$
  C. $\{a,b\}^n$
  D. Other.

Question 10

- Which of the following languages is generated by the given grammar?
- S → a S | b S | ε
  A. $a^n b^m$
  B. $\{a,b\}^*$
  C. $\{a,b\}^n$
  D. Other.

Question 11

- Which of the following strings is not generated by the following grammar?
- $S \rightarrow SaSbS|\varepsilon$
  A. aabb
  B. abab
  C. aababb
  D. aabbb

Question 11

- Which of the following strings is not generated by the following grammar?
- $S \rightarrow SaSbS|\varepsilon$
  A. aabb
  B. abab
  C. aababb
  D. aabbb

Question 12

- Which of the following is NOT the set of regular expression R = (ab + abb)* bbab
  A. ababbbbab
  B. abbbab
  C. ababbabbbab
  D. abababab

Question 12

- Which of the following is NOT the set of regular expression R = (ab + abb)* bbab
  A. ababbbbab
  B. abbbab
  C. ababbabbbab
  D. abababab

Question 13

- In a one pass compiler, the syntax analysis is performed:
  A. After the lexical analysis.
  B. After the semantic analysis.
  C. With the lexical analysis.

Question 13

- In a one pass compiler, the syntax analysis is performed:
  A. After the lexical analysis.
  B. After the semantic analysis.
  C. With the lexical analysis.

Question 14

- When a syntax error appears, the compiler:
  A. Stops Immediately.
  B. Stops after collecting few Errors.
  C. Depends on the organization of the syntax rules.

Question 14

- When a syntax error appears, the compiler:
  A. Stops Immediately.
  B. Stops after collecting few Errors.
  C. Depends on the organization of the syntax rules.

Question 15

- Each deterministic finite automata (DFA) has an equivalent regular expression
  A. True.
  B. False.

Question 15

- Each deterministic finite automata (DFA) has an equivalent regular expression
  A. True.
  B. False.

Question 16

- Each Non deterministic finite automata (NFA) has an equivalent regular expression
  A. True.
  B. False.

Question 16

- Each Non deterministic finite automata (NFA) has an equivalent regular expression
  A. True.
  B. False.

CONTACT INFO