1.2 التحليل اللفظي(Lexical Analysis)

يعتبر التحليل اللفظي المرحلة الأولى في عملية الترجمة (المترجم). تبدأ هذه المرحلة بأخذ كود المصدر المعدل الذي يتم إخراجه من قبل المعالجات المسبقة للغة(Language Preprocessors) ، حيث يكون هذا الكود مكتوبًا على شكل جمل برمجية.

يقوم **المحلل اللفظي (Lexical Analyzer)** بتقسيم هذه الجمل إلى سلسلة من التوكنات(Tokens) ، مع إزالة أي فراغات بيضاء (Whitespace) أو تعليقات (Comments) موجودة في الكود المصدري.

يمكن اعتبار السلاسل النصية(Strings) ، الأعداد(Numbers) ، العوامل(Operators) ، ورموز الترقيم (Punctuation Symbols) جميعها توكنات.

2.2 خطوات تصميم التحليل اللفظي (Lexical Analysis)باستخدام التصميم المتدرج (Stepped Design)

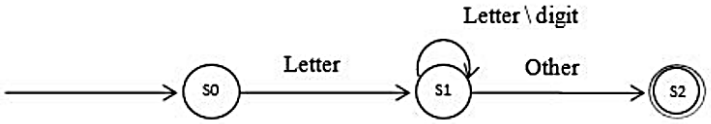
يُعد التحليل اللفظي (Lexical Analysis) المرحلة الأولى في بناء المترجم، ويهدف إلى تحويل سلسلة الأحرف في البرنامج المصدر إلى مجموعة من الرموز (Tokens) التي تُستخدم لاحقًا في مراحل التحليل النحوي والدلالي. يُستخدم في تصميم هذه المرحلة منهج التصميم المتدرج (Stepped Design)، والذي يعتمد على سلسلة من الخطوات المرتبة والمنظمة كما يأتي:

1.2.2كتابة التعبيرات النمطية لكل مفردة (Write Regular Expression for Each Token)

في هذه المرحلة، يتم تحديد كل نوع من المفردات (Tokens) التي يمكن أن تظهر في البرنامج المصدر، ومن ثم كتابة تعبير نمطي (Regular Expression - RE) يمثل الشكل العام لكل مفردة. تشمل هذه المفردات الكلمات المحجوزة(Keywords) ، المعرفات(Identifiers) ، الثوابت العددية (Constants)، الفواصل(Delimiters) ، والعوامل.(Operators)

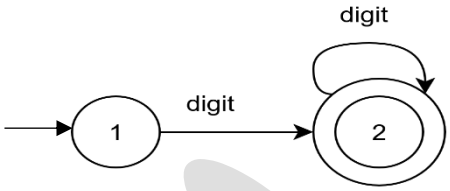
أمثلة على التعبيرات النمطية:

1. المعرفات : identifier → letter (letter | digit)*

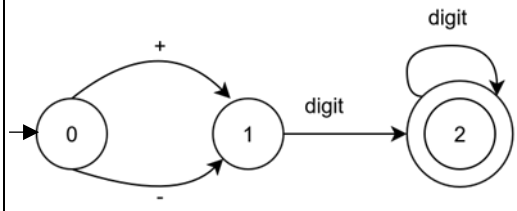


2. الاعداد بكل الأنواع :

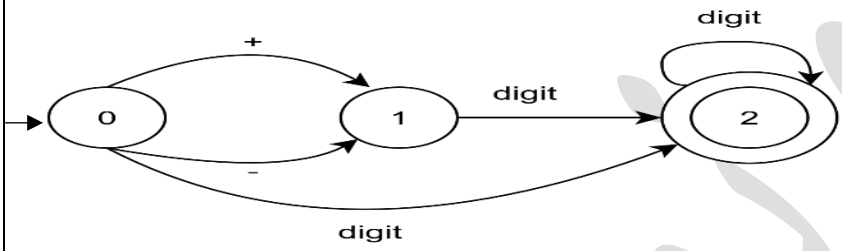
الاعداد الصحيحة



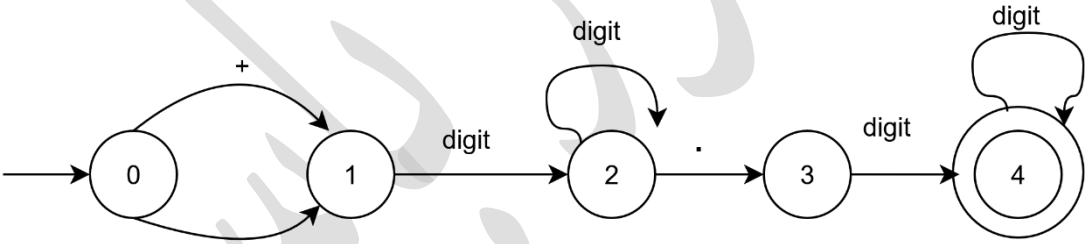
الاعداد الصحيحة الموجبة والسالبة



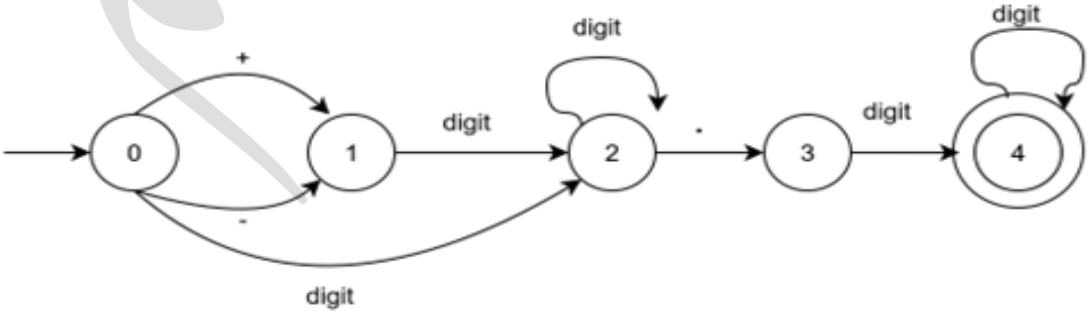
الاعداد الصحيحه بأشاره او بدون إشارة



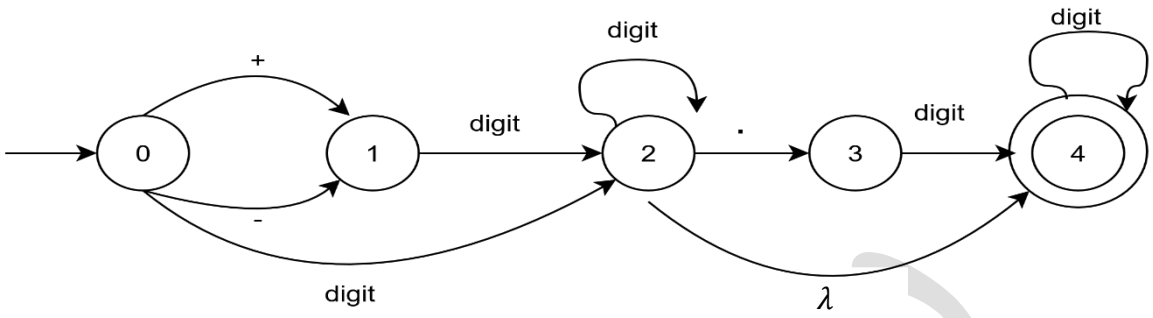
الاعداد الحقيقة



الاعداد الحقيقة التي تحمل إشارة والتي لا تحمل

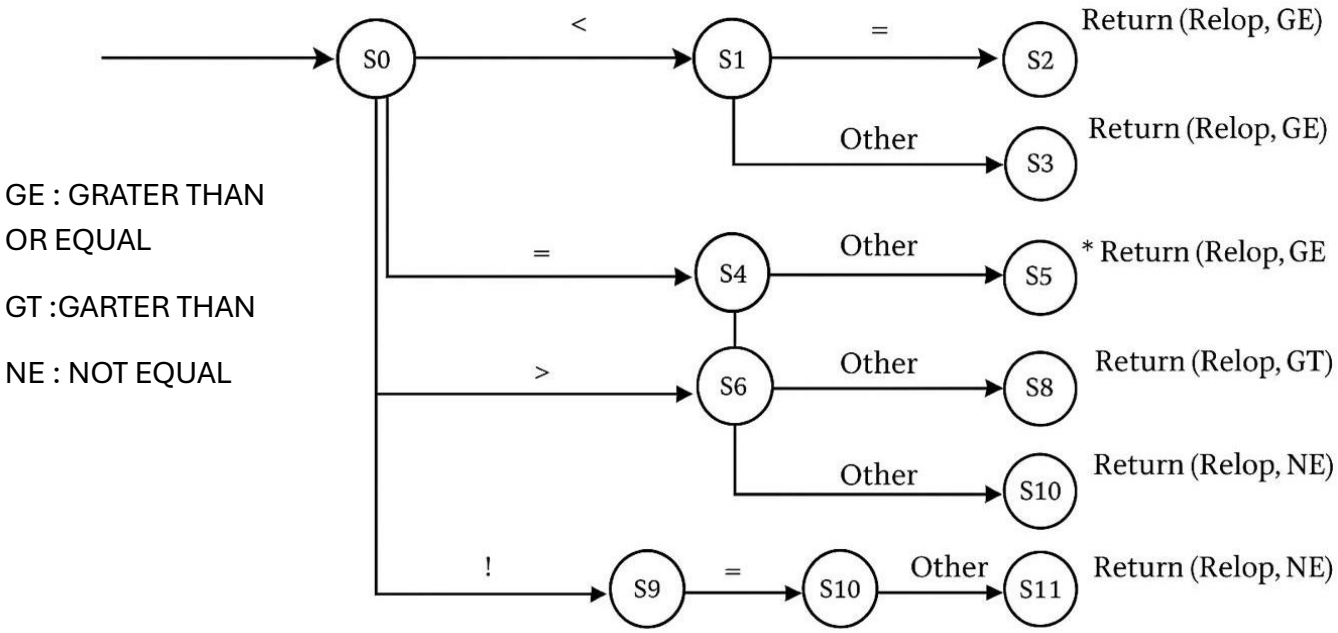


الاعداد الصحيحة و الحقيقة التي تحمل إشارة والتي لا تحمل



3. العمليات العلائقية

R = < = | < | >= | > | = | =.



يُعد هذا التوصيف اللغوي الأساسي خطوة ضرورية لتمكين بناء الآلة المتحكمة بالتعرف على هذه المفردات لاحقًا.

2.2.2 بناء آلة الحالات غير الحتمية (Construct Nondeterministic Finite Automata - NFA)

بعد صياغة التعبيرات النمطية، يتم تحويل كل تعبير نمطي إلى آلة حالات منتهية غير حتمية .(NFA) تتم هذه العملية باستخدام خوارزمية **Thompson's Construction**، والتي تتيح بناء NFA مباشرة من التعبير النمطي عبر سلسلة من القواعد.

تتسم آلة NFA بالقدرة على الانتقال بين الحالات عبر الرموز أو عبر الانتقال الفارغ(ε-move) ، مما يجعلها مناسبة كنموذج مبدئي قبل التحويل إلى آلة حتمية.

3.2.2الآلة محدودة الحالات(Finite State Automata)

يُطلق على البرنامج الذي يتعرف على لغة معينة L اسم المُميز(Recognizer) ، وهو برنامج يأخذ سلسلة إدخال xxx كمدخل ويُجيب بـ "نعم" إذا كانت xxx جملة صحيحة ضمن اللغة، وبـ "لا" إذا لم تكن كذلك.

يمكن تحويل أي تعبير نمطي (Regular Expression) إلى مُميز (Recognizer) من خلال بناء مخطط انتقال عام يُعرف باسم الآلة المحدودة(Finite Automaton) ، والتي تنقسم إلى نوعين رئيسيين:

أولاً: الآلة المحدودة غير الحتمية(NFA – Nondeterministic Finite Automata)

الآلة غير الحتمية لا تفرض قيودًا على تسميات الحواف (Edges) بين الحالات. فقد تحتوي الحالة الواحدة على عدة انتقالات بنفس الرمز، كما يمكن أن تتضمن الانتقالات رموزًا فارغة (εإبسيلون)، أي أنها لا تستهلك أي رمز من سلسلة الإدخال. يُستخدم هذا النوع من الانتقالات في بعض النماذج مثل الآلات ذات الحالات المحدودة غير الحتمية (Non-deterministic Finite Automata) NFA للسماح بالتحول من حالة إلى أخرى دون الحاجة لقراءة رمز معين من الإدخال. وتُعرف هذه بالانتقالات الصامتة (Silent Transitions) أو .ε-transitions

4.2.2مكونات NFA الآلة غير الحتمية ذات الحالات المحدودة

تتكوّن NFA من العناصر الخمسة التالية:

1. **مجموعة منتهية من الحالات:(S)**

   وهي جميع الحالات التي يمكن للآلة أن تكون فيها أثناء التنفيذ.

2. **أبجدية الإدخال:(Σ)**

   وهي مجموعة الرموز التي يمكن للآلة قراءتها من سلسلة الإدخال.

   ملاحظة :لا تشمل هذه المجموعة الرمز ε (الإبسيلون)، والذي يمثل **الانتقال دون قراءة أي رمز**.

3. **دالة الانتقال:(δ)**

   دالة تأخذ حالًة ورمزًا من المجموعة

   $\Sigma \cup \{\varepsilon\}$

   وتُعيد مجموعة من الحالات المحتملة التي يمكن الانتقال إليها.

   أي أن:

   $\delta: S \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2S$

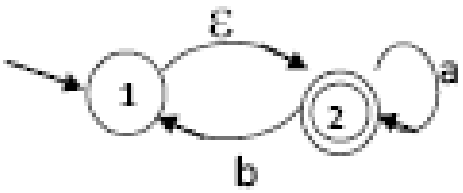
4. **الحالة الابتدائية:(s_0)**

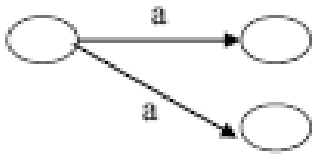
   وهي الحالة التي تبدأ منها الآلة، حيث s0∈Ss_0 \in Ss0∈S.

5. **مجموعة الحالات النهائية:(F)**

وهي مجموعة جزئية من$S: F \subseteq S$

وتُمثل الحالات التي تُعتبر **قبولاً للسلسلة** إذا انتهى التنفيذ فيها





ثانيًا: الآلة المحدودة الحتمية(DFA – Deterministic Finite Automata)

الآلة الحتمية تفرض قيودًا صارمة، حيث أنه:

- لكل حالة، ولكل رمز إدخال من Σ ، يوجد انتقال واحد فقط (وحيد) إلى حالة أخرى.
- لا يُسمح بوجود انتقالات فارغة .ε

5.2.2الفرق بينNFA & DFA

- كل من الآلتين **NFA**و **DFA**قادرتان على **تمييز نفس أنواع اللغات**، وهي ما يُعرف بـ **اللغات المنتظمة**.**(Regular Languages)**
- الفرق الرئيسي يكمن في **آلية الانتقال**:
  - في :NFA يمكن أن يكون هناك أكثر من انتقال لنفس الرمز من نفس الحالة، أو حتى انتقالات بدون رموز.(ε)
  - في :DFA لكل حالة ورمز، يوجد انتقال واحد فقط.

6.2.2التمثيل البياني:(Transition Graph)

يمكن تمثيل كل من NFA و DFAباستخدام **رسم بياني للانتقال(Transition Graph)** ، بحيث:

- كل **عقدة (Node)** تمثل حالة.
- كل **حافة (Edge)** تمثل انتقالًا.
- إذا وُجد انتقال من الحالة sإلى t بالرمز a، فهذا يعني أنه عند قراءة aمن الحالة s، يمكن الانتقال إلى .t

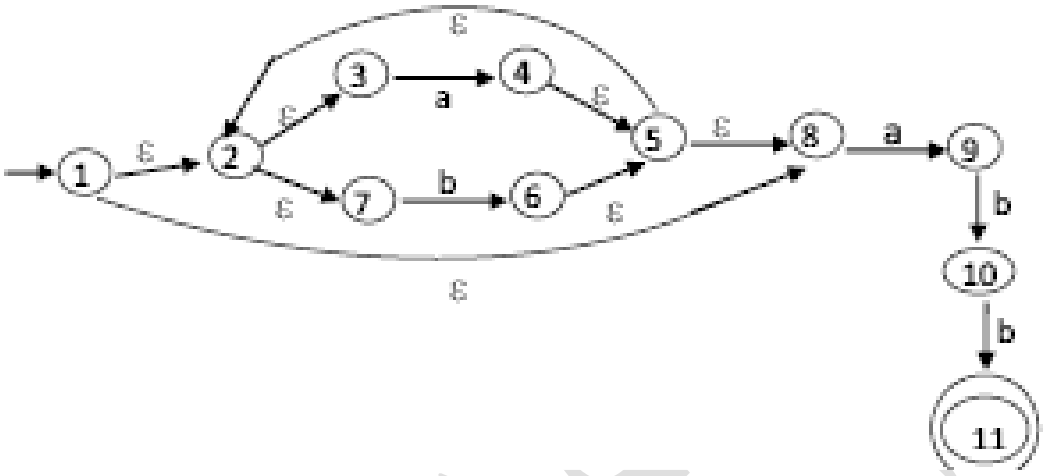
ويُلاحظ في NFA أن:

1. يمكن أن يكون لنفس الرمز انتقالات متعددة من نفس الحالة.
2. يمكن أن تحتوي الحواف على الرمز \εلتمثيل الانتقالات الفارغة.

7.2.2تحويل NFA إلى(Deterministic Finite Automata) DFA بصوره قانونية

نظرًا لصعوبة تنفيذ NFA بشكل مباشر في البرامج الحاسوبية، يتم تحويلها إلى آلة حالات منتهية حتمية (DFA) باستخدام خوارزمية تحويل NFA إلى .DFA تُعد DFA أكثر كفاءة في المعالجة الزمنية، حيث لا تسمح بوجود أكثر من انتقال واحد لكل رمز إدخال.

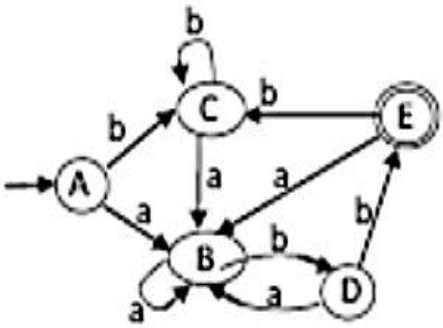
مثال: حول الأتمتة غير المحددة (NFA)أدناه الى أتمتة محدده (DFA)بشكل قانوني



(A تمثل حالة البداية لـ DFA)

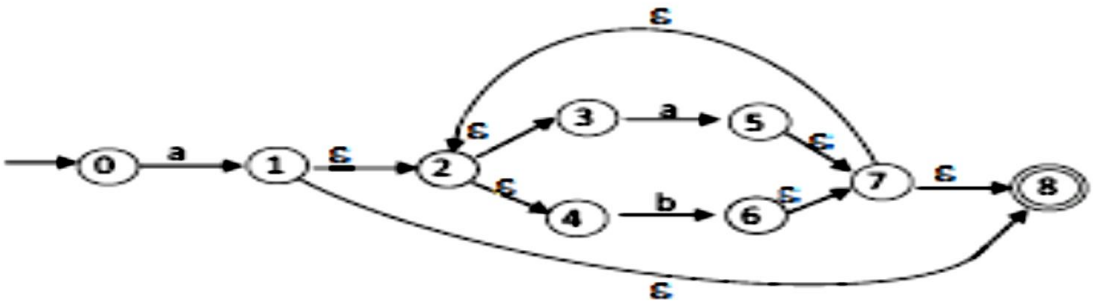
ε-closure(1) = {1, 2, 3, 7, 8} = A

Move (A,a)= {4, 9}
ε-closure(4,9) = {4, 5, 2, 3, 8, 7, 9} = B
Move (A,b)={6}
ε-closure(6) = {6, 5, 8, 2, 3, 7} = C
Move (B,a)= {4, 9} =B
Move (B,b)={10,6}
ε-closure(6,10) = {6, 10, 5, 8, 2, 3, 7} = D
MOVE (C,a) = {9, 4} = B
Move (C,b) = {6}= C
Move (D,a)= {9, 4}= B
Move (D,b)= {11, 6}
ε-closure(11,6) = {11, 6, 5, 8, 2, 3, 7} = E
move (E,a) = {9, 4} = B
MOVE (E,b) = {6} = C



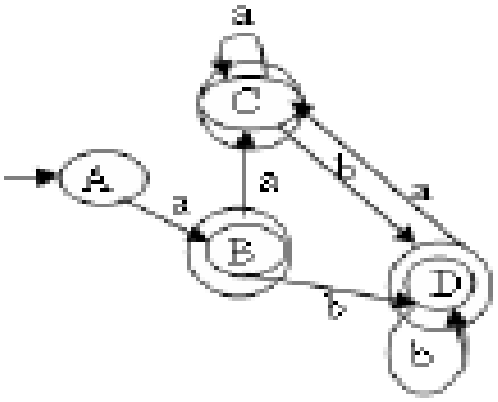
لان حالة NFAالنهائية هي (11) فان المجاميع التي تحتوي الحالة (11) هي Eفقط لذا تصبح الحالة النهائية للأتمتة المحددة الناتجة DFA

مثال: حول الأتمتة غير المحددة (NFA)أدناه الى أتمتة محدده (DFA)بشكل قانوني



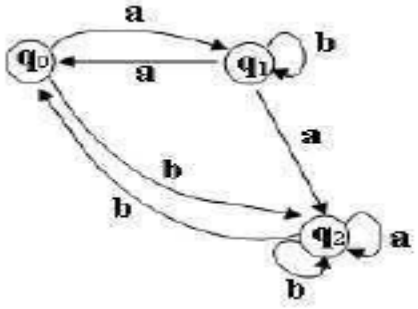
ε-closure (0) = {0} =A
Move(A,a)= {1}
Move(A,b)= {∅}
ε-closure (1) = {2, 3, 4, 8, 1} = B
Move(B,a)={5}
ε-closure (5)={5,7,8,2,3,4}=C
Move (B,b)={6}
ε-closure (6)={6,7,8,2,3,4}=D
Move (C,a) ={5}=C
Move(C,b)= {6} = D
Move(D,a)={5}=C
Move (D,b)={6}=D

لان حالة NFA النهائية هي (8) فأن المجاميع التي تحتوي الحالة (8) هي (B,C,D) لذا تصبح الحالة النهائية للأتمتة المحددة الناتجة (DFA)



8.2.2تحويل NFA إلى(Deterministic Finite Automata) DFA باستخدام الطريقة الغير القانونية (التجريبية)

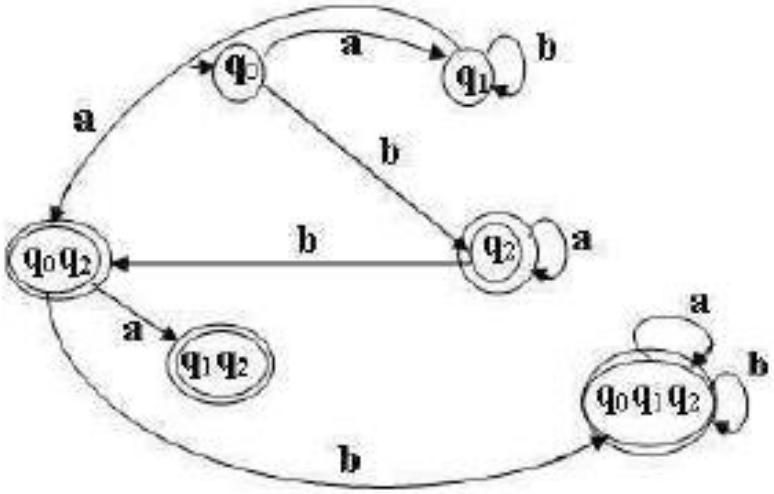
مثال : حول الأتمتة غير المحددة الى أتمتة محدده بالطريقة التجريبية



والمصفوفة التالية تعبر عن الأتمتة غير المحددة المرسومة في اعلاه

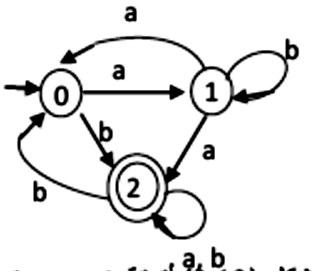
الحالات	a	b
q0	q1	q2
q1	q0q2	q1
(q2)*	q2	q0q2
(q0q2)*	q1q2	q0q2
(q1q2)*	q0q2	q0q1q2
(q0q1q2)*	q0q1q2	q0q1q2

ما نلاحظ من الجدول أعلاه، إذا كانت إحدى الحالات تذهب إلى حالتين على أثر نفس المدخل، فإنه تُستحدث حالة جديدة تمثلنا، فمثلًا فإن الحالة (q1) مع المدخل (a) تذهب إلى حالتين هما (q0) و (q2)، لذا تم استحداث حالة جديدة بالاسم(q0q2) ، وهكذا بالنسبة للبقية. علمًا أن حالة البداية للـ NFAهي نفسها تعتبر حالة البداية للـDFA ، وأن الحالة (q2) تعتبر نهائية بالنسبة للآلة غير المحددة (NFA)، فإن كل الحالات الموجودة في الآلة المحددة والتي تحتوي الحالة (q2) تعتبر نهائية.



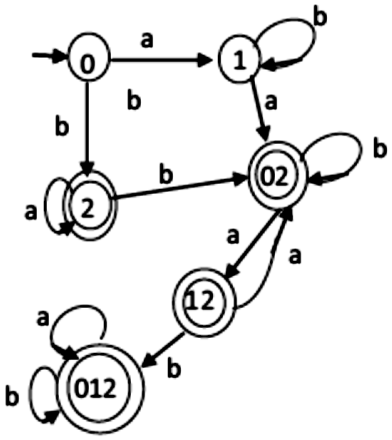
الأتمتة المحددة

مثال : حول الأتمتة الغير محدده الى أتمتة محددة



NFA	a	b
0	1	2
1	0,2	1
2	2	0,2

DFA	a	b
0	1	2
1	02	1
2	2	02
02	12	02
12	02	012
012	012	012



9.2.2 تنفيذ DFAبرمجياً(DFA Implementation)

في هذه المرحلة، يتم تنفيذ DFA باستخدام لغة برمجة مناسبة (مثل C أو Java أو Python) يتضمن التنفيذ تتبع حالات الآلة، وتمييز الحالات النهائية (Final States) المرتبطة بكل نوع من أنواع

المفردات. يجب أن يتضمن التنفيذ آلية لمعالجة الأخطاء في حال ظهور مدخلات لا تطابق أي نمط معروف.
أي تحويل الآلة التي تم تصميمها (مخطط DFA) إلى كود برمجي فعلي يعمل على الكمبيوتر. والهدف من هذا الكود هو:

- أن يستقبل سلسلة من الأحرف.(input string)
- ثم **يتنقّل بين الحالات** وفقًا للرموز التي يقرأها.
- حتى يصل إلى نهاية السلسلة، وعندها:
  - إذا كان في حالة نهائية ← يكون قد تعرّف على **Token**صالح.
  - إذا لم يكن في حالة نهائية أو وجد رمزًا غير متوقع ← يُظهر **رسالة خطأ**.

10.2.2 توليد الرموز(Token Generation)

يتم تحليل السلسلة المصدرية حرفًا باستخدام DFA ، ومع كل تطابق لنمط معين يتم توليد رمز (Token)يحتوي على:

- نوع token (Type)
- القيمة النصية(Lexeme)
- معلومات إضافية مثل رقم السطر والموقع (اختياريًا)

مكونات token (Token Structure)

كل token يحتوي عادةً على:

المكوّن	الوصف
النوع(Type)	IDENTIFIER, NUMBER, KEYWORD, OPERATOR, SEPARATOR
القيمة(Lexeme)	السلسلة الفعلية من الأحرف التي تطابقت مع النمط مثل: sum, 123, +
الموقع (اختياري)	رقم السطر والموقع داخل السطر (يساعد في رسائل الأخطاء والتحليل الدلالي)

11.2.2 التكامل مع مراحل المترجم الأخرى(Integration)

بعد أن يُكمل المحلل المعجمي (Lexical Analyzer) عمله ويقوم بتوليد سلسلة من (Tokens)، يجب أن يتم نقل هذه (tokens) بشكل منظّم إلى المرحلة التالية في المترجم، وهي مرحلة التحليل النحوي(Syntax Analysis) ، ثم التحليل الدلالي(Semantic Analysis) ، وأخيرًا توليد الشيفرة Code Generation).)

حيث يتم التكامل عن طريق تمرير قائمة tokensالمولدة)مثل: <KEYWORD, int> ,<IDENTIFIER, x>)إلى محلل الجُمل.(Parser)

- يعمل الـ Parser على بناء شجرة تحليل نحوي (Parse Tree) أو شجرة تجريدية – AST) (Abstract Syntax Treeباستخدام هذه tokens.
- يتطلب هذا أن يكون تنسيق tokens واضحًا ومتناسقًا حتى يتمكن محلل الجمل من معالجتها بشكل صحيح.

وتعتبر هذه الخطوة مهمة لان المترجم ليس وحدة واحدة، بل هو سلسلة من المراحل تعتمد كل منها على مخرجات المرحلة السابق، وبدون التكامل الصحيح، لا يمكن للمراحل اللاحقة (مثل التحليل النحوي) تفسير tokens أو العمل عليها، كما أن أي خلل في تنسيق tokensأو في ترتيبها قد يؤدي إلى فشل في بناء الشجرة النحوية وبالتالي فشل عملية الترجمة كلها.