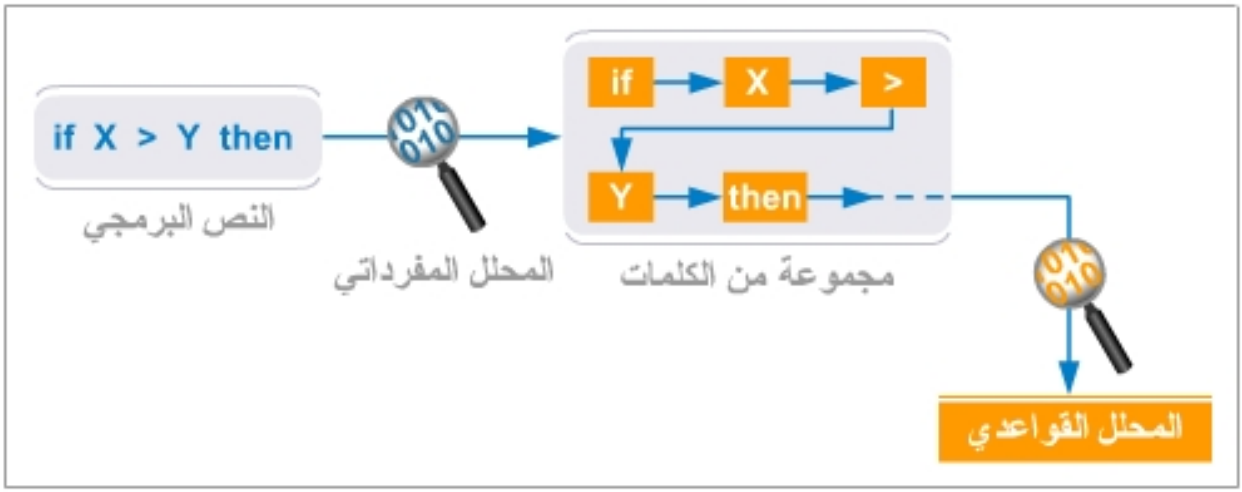
الفصل الثالث

التحليل المفرداتي

مقدمة

يشكل المحلل المفرداتي الجزء الأول من المترجم وينفذ المرحلة الأولى من عمليـة الترجمـة. تـتلخص المهمة الأساسية للمحلل اللفظي بتجميع محارف الدخل، الآتية من النص البرمجي المصدري، بهدف توليد مجموعة من الكلمات التي ستؤلف بدورها جمل يعالجها المحلل القواعدي.



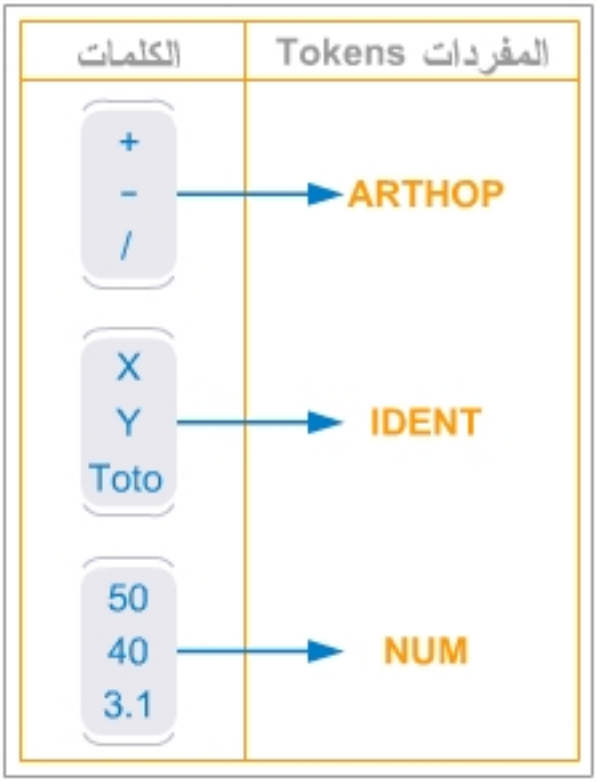
ينفذ المحلل المفرداتي أيضاً مجموعة من المهام الثانوية من أهمها: حذف المحـارف التـي لا دور لهـا: كالفراغ وسلاسل التعليقات، كما يتولى مهمة حفظ أرقام أسطر النص البرمجي التي يمكن الاستعانة بهـا للإشارة إلى مكان الأخطاء الموجودة في النص البرمجي.

سنستعرض في هذا الفصل بنية المحلل المفرداتي وتفاصيل عملية التحليل المفرداتي. تجدر الإشارة إلـى أن مسألة تصميم وتشغيل محلل مفرداتي تكافئ مسألة تصميم برنامج ينفذ عمليات علـى سلاسـل مـن المحارف ويهتم بالتعرف على أشكال وصيغ محددة لهذه السلاسل، وهي مسألة تصب فـي مجـال بنـاء المنظومات التي تساعد في البحث عن المعلومات وتشكل صلب محركات البحث.

في كل نص برمجي تشكل كل مجموعة من الأحرف المتتالية، "كلمة". يكون لكل كلمـة مـن الكلمـات المستخدمة في النص البرمجي شكل يحدد انتماءها إلى نوع من الكلمات أو ما ندعوه "نمـوذج" (Model). ونستخدم لمجموعة الكلمات التي لها نموذج محدد اسم ندعوه "مفردة" (Token).

فعلى سبيل المثال: تعبر الكلمات: (**+** ، **-** ، ***** ، **/** ، **%**) عن عمليات حسابية. يمكن أن نستخدم للتعبيـر عن العمليات الحسابية المفردة ARTHOP (من Arithmetic Operator).

كما تعبر الكلمات (X، Counter، Y، Toto) عن متحولات يستخدمها المبرمج، ويمكن أن نستخدم المفردة IDENT (من Identifier) للتعبير عن المتحولات. كما يكون للمتحولات نموذج يعبر عن شـكل المتحـول وهو ما سنراه لاحقاً (فعلى سبيل المثال يفرض النموذج الذي يعرف شكل المتحولات عدم استخدام أرقـام لتعريف متحول. إذ لا يمكن للمحلل المفرداتي اعتبار 55 متحول، فالمحلل يعتبر 55 قيمة رقمية).



1- التعابير المنتظمة

نستعرض في هذه الفقرة مجموعة من التعاريف والمبادئ الخاصة بنظرية اللغات، وهي تعاريف ومبـادئ مستخدمة بكثافة في عملية التحليل المفرداتي.

تعريف: ندعو "أبجدية" (Alphabet)، مجموعة منتهية غير خالية Σ من الرمـوز. كمـا نـدعو "كلمـة" (Word) كل سلسلة منتهية من عناصر Σ .

نستخدم الرمز ε للدلالة على الكلمة الفارغة. كما نستخدم الرمز Σ^* للدلالة على المجموعة غير المنتهية التي تضم جميع الكلمات الممكنة المبنية اعتباراً من الأبجدية Σ . ونـستخدم الرمـز Σ^+ للدلالـة علـى مجموعة الكلمات غير الفارغة التي يمكن أن نبنيها اعتباراً من Σ . بالنتيجة يكون:

$$\Sigma^+ = \Sigma^* - \{\varepsilon\}$$

يشير الرمز $|m|$ إلى طول الكلمة m، ويعبر عن عدد الأحرف المنتمية إلى الأبجدية Σ والتي تشكل الكلمة m. ونستخدم الرمز Σ^n للدلالة على مجموعة الكلمات المنتمية إلى Σ^* والتي طولها n. بالتالي يمكننـا أن نستنتج أن:

$$\Sigma^* = \sum_{n=0}^{\infty} \Sigma^n$$

$$| \text{ Toto } | = 4$$

$$\text{" Toto "} \in \Sigma^4$$

تعريف: نعرف عملية "الدمج التسلسلي" (Concatenation) والتي نرمز لها بالرمز "•"، بأنها عملية نطبقها على كلمتين كالتالي: لتكن لدينا الكلمة $u=u_1...u_n$ (حيث $u_i\in\Sigma^*$) والكلمة $v=v_1...v_p$ (حيث $v_i\in\Sigma^*$)، يكون حاصل الدمج التسلسلي للكلمتين u و v، الكلمة $u\bullet v=u_1...u_n\,v_1...v_p$. بشكل عام يمكن إهمال رمز الدمج التسلسلي "•" وكتابة uv بدلاً عن $u\bullet v$.

فعلى سبيل المثال، لنأخذ الأبجدية $\Sigma=\{a,b,c\}$، والكلمات: u=aaba، v=bbbacbb، w=c، $t=\varepsilon$، هي كلمات تنتمي إلى Σ^* وبأطوال: 4، 7، 1، 0 على التسلسل. تكون الكلمة $u\bullet v=aababbbacbb$ حاصل الدمج التسلسلي للكلمتين u و v.

خصائص:

1- $|u\bullet v|=|u|+|v|$.

2- $(u\bullet v)\bullet w=u\bullet(v\bullet w)$.

3- $u\bullet\varepsilon=\varepsilon\bullet u=u$.

تعريف: ندعو لغة مبنية على أبجدية Σ، كل مجموعة جزئية من Σ^*.

على سبيل المثال، يمكن اعتباراً من الأبجدية $\Sigma=\{a,b,c\}$ تعريف لغة غير منتهية L_1 حيث تتألف هذه اللغة من الكلمات: {ε, a, b, aab, bbcaa, bbcccacccaaaabbbcaa, ... }

1. المفردات

1- التعابير المنتظمة

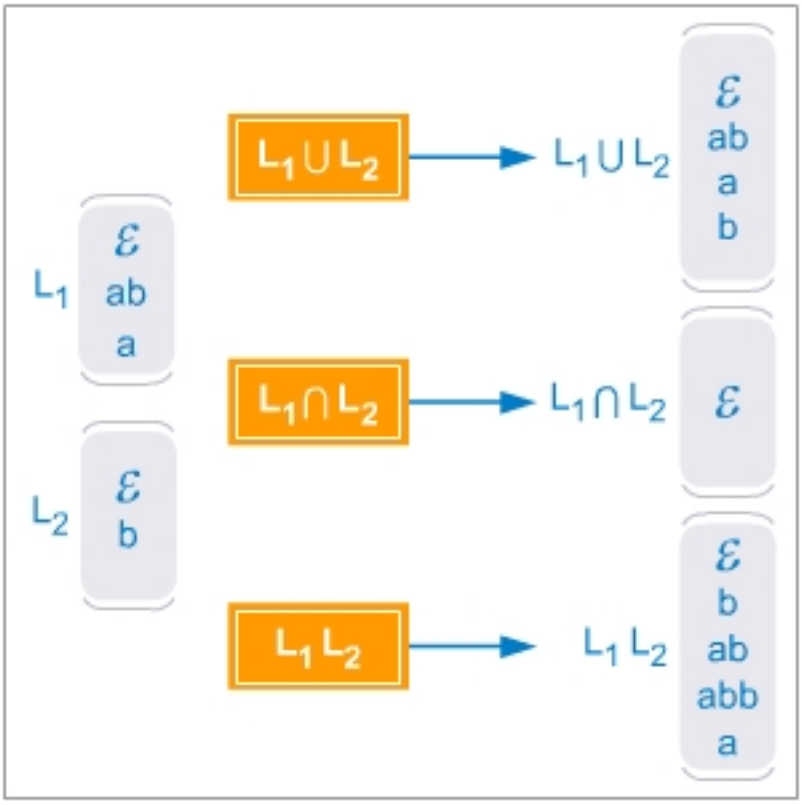
تعريف العمليات على اللغات: **...**

1- الاجتماع: $L_1\cup L_2=\{w: w\in L_1 \quad OR \quad w\in L_2\}$

2- التقاطع: $L_1\cap L_2=\{w: w\in L_1 \quad AND \quad w\in L_2\}$

3- الدمج التسلسلي: $L_1L_2=\{w=w_1w_2: w_1\in L_1 \quad AND \quad w_2\in L_2\}$

4- الإغلاق: $L^*=\sum_{n\geq 0}L^n$



السؤال الذي يطرح نفسه الآن: في حال كان لدينا لغة ما L. كيف يمكن توصيف الكلمات المنتميـة إلـى اللغة؟ وكيف يمكن توصيف هذه اللغة. عموماً، توجد عدة أنماط من اللغات وهو ما سـنراه فـي فـصل لاحق، ولكن سنركز في هذا الفصل على اللغات التي ندعوها لغات منتظمة.

تعريف: نستطيع تعريف "لغة منتظمة" L (Regular Language) على أبجدية Σ بشكل عودي كما يلـي:

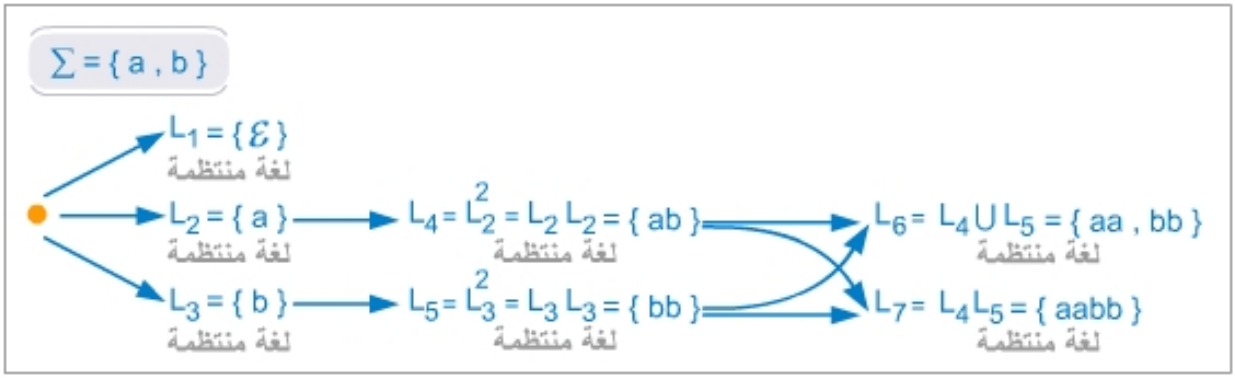
$\{\varepsilon\}$ هي لغة منتظمة على Σ.

5- إذا كان a حرف من حروف الأبجدية Σ، تكون {a} لغة منتظمة على Σ.

6- إذا كانت L لغة منتظمة على Σ، تكون كلٌ من L^n و L^* لغات منتظمة على Σ.

7- إذا كان كلٌ من L_1 و L_2 لغات منتظمة، تكون كلٌ من $L_1 \cup L_2$ و $L_1 L_2$ لغات منتظمة على Σ.

8- لا توجد لغات منتظمة أخرى على Σ.



تعريف: يمكن توصيف اللغـات المنتظمـة، باسـتخدام أداة نـدعوها "التعـابير المنتظمـة" (Regular Expressions). نعطي فيما يلي تعريف عودي للتعابير المنتظمة:

1- ε هو تعبير منتظم يوصف اللغة $\{\varepsilon\}$.

2- إذا كان a حرف من حروف الأبجدية Σ، يكون a تعبير منتظم يوصف اللغة {a}.

3- إذا كان r تعبير منتظم يوصف اللغة L فإن كلاً من $(r)^*$ و $(r)^+$ عبارة عن تعبيـرين منتظمـين يوصفان اللغتين L^* و L^+ على الترتيب.

4- إذا كان r_1 و r_2 تعبيرين منتظمين يوصفان اللغتين L_1 و L_2 علـى الترتيـب، فـإن كـلاً مـن $(r_1)|(r_2)$ و $(r_1)(r_2)$ عبارة عن تعبيرين منتظمين يوصفان اللغـتين $L_1 \cup L_2$ و L_1L_2 علـى الترتيب.

5- لا توجد تعابير منتظمة أخرى.

أمثلة عن التعابير المنتظمة:

1- يوصف $(a|b)^*$ مجموعة الكلمات (اللغة) المؤلفة من a و b، أو الكلمة الفارغة.

$$L = \{ \varepsilon, a, b, aa, bb, ab, ba, \dots \}$$

2- يكون $(a|b)^* = (b|a)^*$.

3- يوصف $(a|b)^*bbb(a|b)^*$ مجموعة الكلمات (اللغة) التي تحتوي على السلسلة bbb فيها.

$$L = \{ bbb, abbb, bbbb, abbbb, abbba, aabbbaba, \dots \}$$

2. تنفيذ التحليل المفرداتي

تتألف من العمليات التالية:

1- توصيف المفردات.

2- تحديد الرموز.

3- تحليل كلمات النص البرمجي.

1- توصيف المفردات

توفر لنا التعابير المنتظمة أداة لوصف الكلمات التي لها نموذج محدد. يتم التعبير عن هذا النموذج بمفردة. فعلى سبيل المثال، جميعنا يعلم أن استخدام اسم أي متحول في نص برمجي يخضع لقواعد محددة:

أن يبدأ الاسم بحرف ومن ثم نستخدم أي حرف، أو رقم، بالإضـافة إلـى المحـرف "_" (underscore)، ودون أن يكون هناك حدود لطول اسم المتحول يمكن اسـتخدام الأحـرف الـصغيرة (small letters) أو الكبيرة (capital letters).

وعليه يمكننا أن نعرف المفردة IDENT المعبرة عن المتحولات، وذلك اعتماداً على تعبير منـتظم، كمـا يلي:

IDENT = (a|b|c … |y|z|A|B|C … |Y|Z) (a|b|c … |y|z|A|B|C … |Y|Z|0|1|2|3|4|5|6|7|8|9)*""

لاختصار العمليات يمكننا أن نعرف ما يلي:

Letter = a – z | A – Z
Digit = 0 – 9
Sep = _
EndOfString =""

وبالتالي يصبح تعريف IDENT على الشكل التالي:

IDENT = Letter (Letter | Digit | Sep)* EndOfString

تجدر الإشارة إلى عدم إمكانية تمثيل كل نماذج الكلمات (وبالتالي عدم إمكانية تعريف المفردات المعبـرة عنها) باستخدام التعابير المنتظمة. فعلى سبيل المثال، في حال كان لدينا في لغة معرفة على أبجدية {a, b} نموذج كلمات له الشكل $a^n b^n$ يتبع القاعدة التالية: "تبدأ الكلمة بالحرف a الذي يتكرر n مرة ومـن ثـم يتبعها نفس العدد من b الذي سيظهر n مرة أيضاً من أجل أي n). فإننا ببـساطة لا نـستطيع اسـتخدام التعابير المنتظمة للتعبير عن هذا النموذج (سيتم توضيح السبب في فصل لاحق).

2- تحديد الرموز

تنقسم رموز لغة برمجة إلى:

1- الكلمات المفتاحية الأساسية مثل if، while، … بالإضافة إلى الرموز البسيطة كالعلاقات الحسابية (+، -، *، /، %) أو علاقات المقارنة (<، >، =>، =<) وغيرها، فإننا نكتفي بتعريف المفـردة المعبرة عنها مثل ARTHOP التي تعبر عن العلاقات الحسابية.

2- بالنسبة للرموز التي تحتاج لقواعد محددة لكتابتها (لها نموذج) مثل المتحـولات، فإننـا (وكمـا أوضحنا في الفقرة السابقة) نحدد قواعد تعريف نموذجها ونعطي تعريف للمفردات المعبرة عنهـا كما هو الحال مع IDENT. بنفس الطريقة يمكن أن نستخدم المفردة REAL للتعبير عن عدد حقيقي (مثل 5.3) والذي تكون قاعدة كتابته كما يلي:

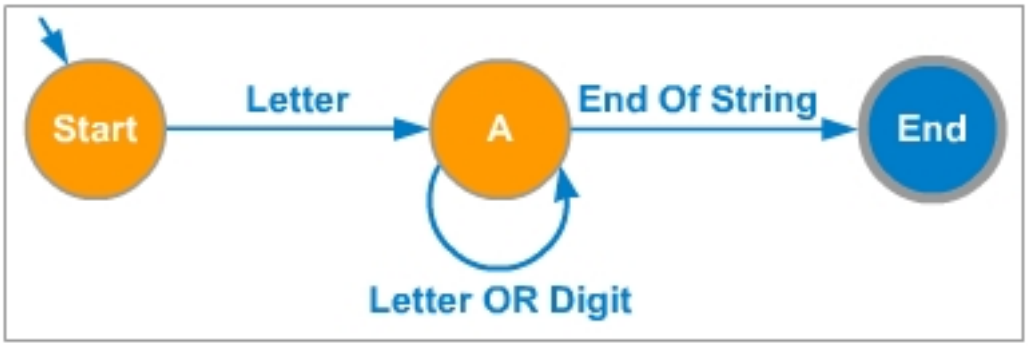
POINT = "."
REAL = (Digit)* POINT (Digit)*

3- تحليل كلمات النص البرمجي

لنفترض أننا نحاول كتابة برنامج للتعرف على أسماء المتحولات بحيث يأخذ البرنامج في دخله اسم ويعيد قيمة 1 أو 0 للدلالة على أن الكلمة تعبر عن اسم متحول أو لا تعبر على الترتيب. لنفترض أن اسم المتحول يتبع للنموذج الذي تمثله المفردة IDENT التي حددناها سابقاً.

نظرية: لكل لغة منتظمة أوتومات منته.
نتيجة: كل أوتومات منته يكافئ برنامج حاسوبي.

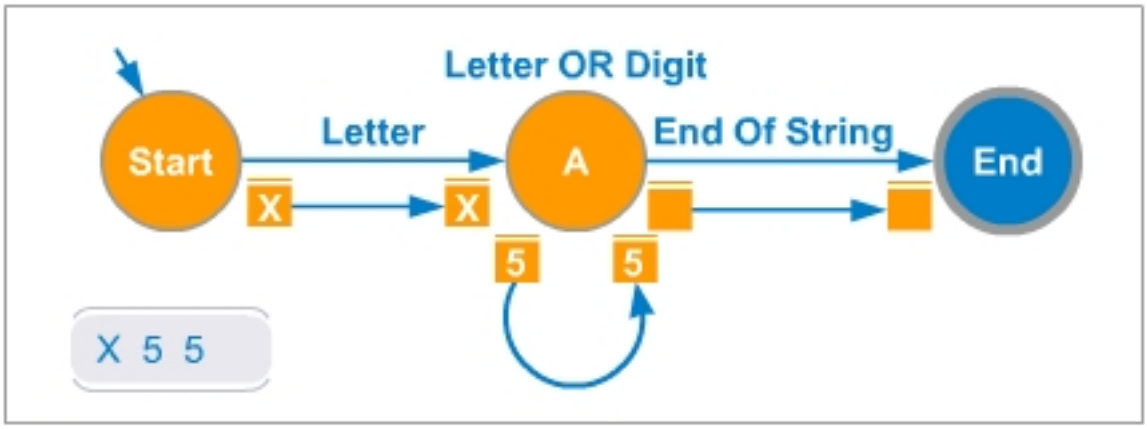
فعلى سبيل المثال، يمكن تمثيل التعبير المنتظم الذي يعرف المفردة IDENT على الشكل التالي:



تعريف: يمكننا تعريف أوتومات منته بشكل أولي (وسنعود لهذا الموضوع في فصل لاحق) على أنها بيان من مجموعة منتهية من الحالات والوصلات.

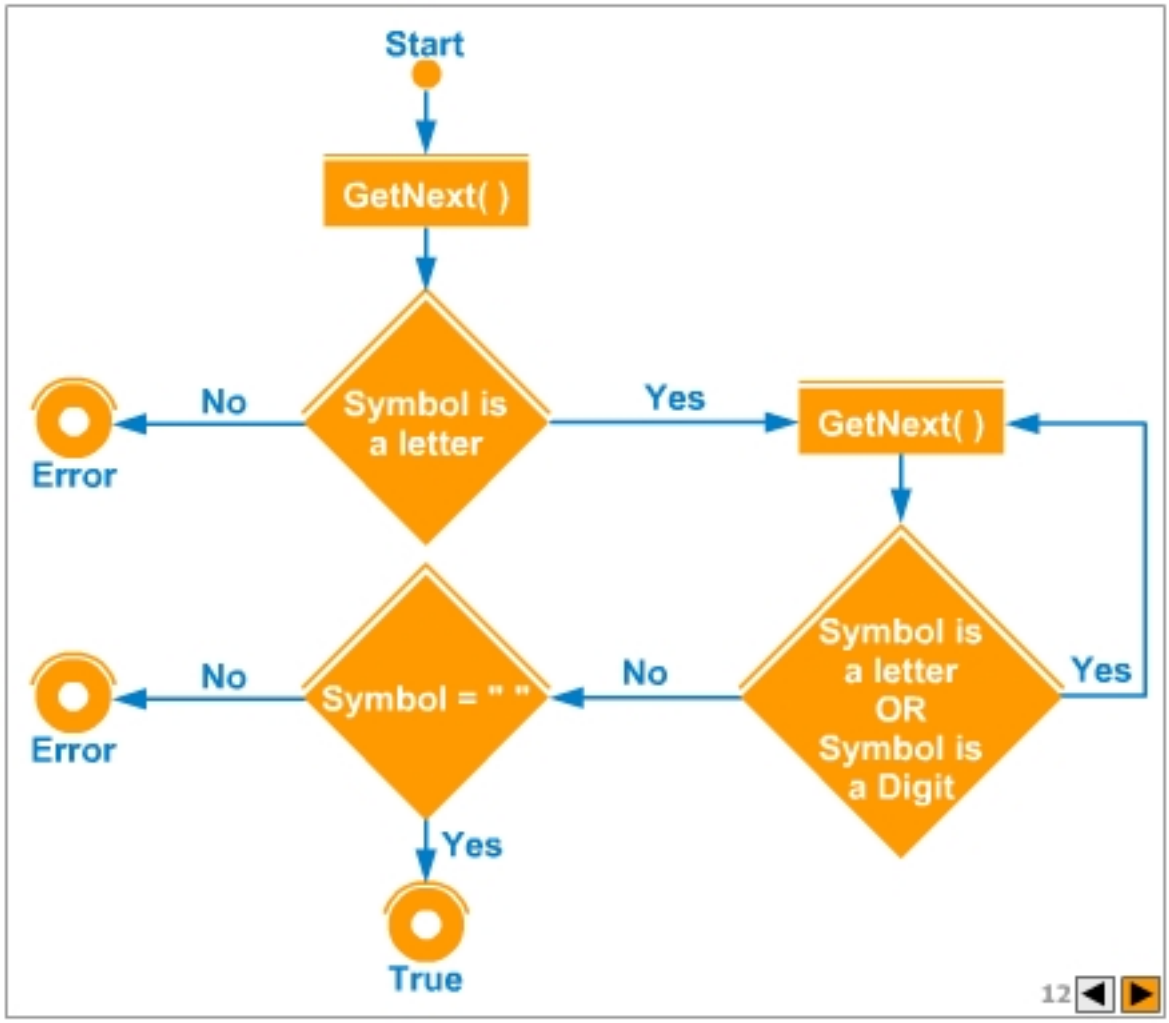
1- هناك أوتومات تتميز بوضع لاصقات على الحالات لتعريفها، وهناك أوتومات تتميز بوضع اللاصقات على الناقلات. في كلا الحالتين نحصل على نفس النتيجة.
2- لكل أوتومات حالة ابتدائية.
3- لكل أوتومات مجموعة من الحالات النهائية التي يمثل الوصول إليها انتهاء عمل الأوتومات.
4- يتم تعريف اللاصقات بالاعتماد على مجموعة من الرموز ومجموعة من العمليات.
5- يبدأ عمل الأوتومات اعتباراً من الحالة الابتدائية وتتبع، من أجل كل رمز مقروء على مدخلها، وصلة مرتبطة بها وذات لاصقة تتكون من الرمز المقروء (في حال كانت اللاصقات مثبتة على النواقل).
6- في حال تمثيل الأوتومات لتعبير نظامي نقول عن كلمة أنها مقبولة من أوتومات إذا استطعنا الوصول (انطلاقاً من الحالة الابتدائية) إلى حالة نهائية باستخدام رموز وحروف الكلمة.

فعلى سبيل المثال يتم التأكد من أن الكلمة x55 مقبولة من الأوتومات السابقة الموضحة في الشكل بتمريرها حرفاً حرفاً ابتداءاً من الحالة الابتدائية. نلاحظ أنها تصل بعد ثلاث انتقالات إلى الحالة النهائية:

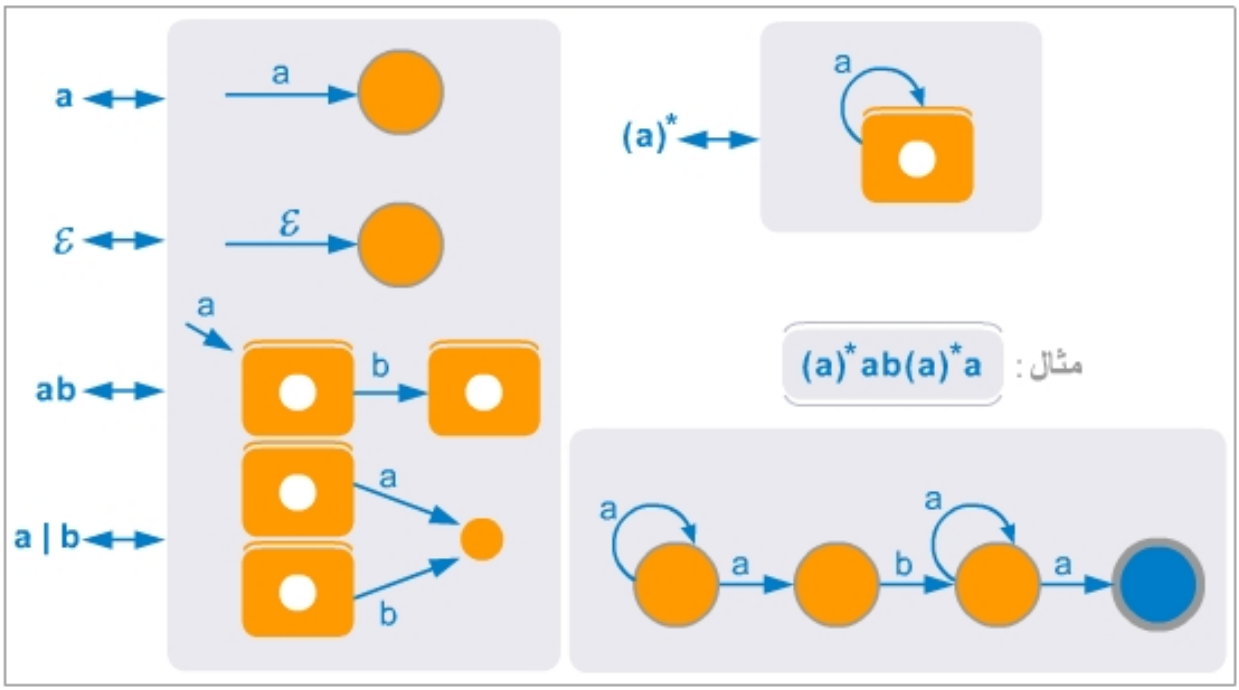


وبما أن كل أوتومات تكافئ برنامج حاسوبي، فيمكننا الآن افتراض وجود إجرائية GetNext() تقرأ الحرف التالي من سلسلة الحروف وتضع هذا الحرف في متحول Symbol. ولنكتب إجرائية **IsIdentifier()** التـي تقوم بالتحقق من كون الكلمة المقروءة هي عبارة عن اسم متحول:

```
int IsIdentifier( )
{
        GetNext( );
        If (Symbol is a Letter)
        {
                GetNext( );
                while ((Symbol is a Letter) | | (Symbol is a Digit)) /*Case Switch*/
                        GetNext( );
                if (Symbol = = '  ')              /* Space*/
                        return TRUE;
                else
                        return Error( );
        }
        else
                return Error( );
}
```



لنلق نظرة الآن على قواعد تحويل تعبير منتظم إلى أوتومات والتي يمكن التعبير عنها بخوارزمية تحويل والتي نعرضها هنا بشكل بياني فقط:



بالنتيجة، نستطيع تحويل تعبير منتظم إلى أوتومات منته حتمي، مما يعني أنه سـيكون مكـافئ لبرنـامج حاسوبي وأننا سنستطيع توليد برنامج حاسوبي. وعليه بإمكاننا أتمتة عملية كتابـة المحلـل مـن خـلال الخطوات التالية:

1- تعريف التعابير المنتظمة المعبرة عن مفردات اللغة التي تمتلك نماذج.
2- استخدام خوارزمية تحويل تعبير منتظم إلى أوتومات.
3- توليد البرنامج الحاسوبي المكافئ للأوتومات وهو البرنامج الذي سيقوم بمعالجة النص البرمجـي المصدري وقراءته وتحديد أنواع كلماته.

3. أخطاء المفردات

تكون علاقة المحلل المفرداتي مع الأخطاء قاصرة. إذ يمكن للمحلل المفرداتي التعرف على عدد محـدود من الأخطاء:

1- خطأ في كتابة رمز من الرموز التي لا تمتلك نموذج ولا توجد قاعدة لتعريفها، كالرموز البسيطة أو الكلمات المفتاحية.
2- خطأ في تطبيق قاعدة من قواعد تعريف المفردات، ككتابة متحول له الشكل 0.55X، حيث يظهـر وضوحاً أن هذا الرمز ليس كلمة مفتاحية، ولا رمز بسيط، ولا يخضع لا لقاعدة تعريف IDENT (فهو ليس متحول)، ولا لقاعدة تعريف REAL (فهو ليس عدد حقيقي).

تتم معالجة الأخطاء عموماً اعتماداً إما على طريقة تسجيل الحرف الذي سبب الخطأ وإهماله والمتابعة كما سنرى لاحقاً عند التطرق لمعالجة الأخطاء في المحلل القواعدي، حيث سننظر للأمر علـى نحـو أكثـر عمومية. أو بمحاولة تصحيح بعض الأخطاء في الرموز البسيطة والكلمات المفتاحية وهو أمـر تحـاول بعض محررات النصوص المرتبطة بالمترجمات تقديمه، كأن يتم تصحيح whle بكتابة while في حال أتت هذه الكلمة في بداية الجملة وتلتها جملة تعبر عن حلقة تكرارية. إذ يعتبر محرر النصوص في هذه الحالة أن الكلمة الصحيحة هي الكلمة المفتاحية التي تحتاج لأقل عدد ممكن من التغييرات حتى تتحول من كلمـة خطأ إلى كلمة صحيحة في المكان الذي تظهر فيه.

4. تمارين

السؤال (1)

ماهي اللغات التي توصفها التعابير المنتظمة التالية المعرفة على الأبجدية {a, b}:

1- $a\ (a \mid b)^{*}\ b$

2- $aab\ (aa \mid bb)^{+}$

3- $(aa)^{*}\ a$

الجواب

1- اللغة التي تبدأ جميع كلماتها بالحرف a وتنتهي بالحرف b.

2- اللغة التي تبدأ جميع كلماتها بالسلسلة aab ومن ثم يليها تكرار واحد على الأقل لثنائية aa أو من bb.

3- اللغة التي لا تحتوي كلماتها إلا على حرف a ويكون طول كلماتها مفرداً.

السؤال (2)

ما هي التعابير المنتظمة التي يمكن أن تعرف اللغات التالية:

1- اللغة على الأبجدية {a, b, c} والتي تبدأ جميع كلماتها بالحرف a.

2- الأعداد الصحيحة من مضاعفات 5.

الجواب

1- $a\ (a \mid b \mid c)^{*}$

2- $(0 \mid 5)\ (0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)^{*}$