5.1 المقدمة

المحلل القواعدي (syntax analyzer) أو المعرّب (parser) هو برنامج يقوم بتنفيذ التحليل القواعدي (syntax analysis) حيث يتم اخذ سلسلة من المقاطع (tokens) من المحلل اللفظي (lexical analyzer) ويقوم بتدقيق هذه السلسلة فيما إذا كانت مقبولة قواعديا (ذات تركيبة متحققة) في لغة ما أي متوّلدة باعتماد قواعد تلك اللغة وذلك باشتقاق تلك السلسلة ابتداء من رمز البداية لتلك القواعد وهناك طريقتين للإعراب الأولى **من الأعلى إلى الأسفل** (top-down parsing) وفيها يتم الاشتقاق ابتداء من رمز بداية قواعد تلك اللغة إلى نصل إلى تلك السلسلة (الجملة)، أما الثانية فهي **من الأسفل إلى الأعلى** (bottom-up parsing) وفيها يتم الاشتقاق ابتداء من الجملة (سلسلة من المقاطع) إلى أن نصل رمز البداية (Start Symbol) لتلك القواعد.

5.2 الإعراب من الأعلى إلى الأسفل (Top-Down Parsing)

هي محاولة لإيجاد الاشتقاق أقصى اليسار للسلسلة (الجملة) المدخلة w والتي تكافئ إنشاء شجرة إعراب (parse tree) للسلسلة المدخلة w ابتداء من رأس الشجرة (root) وإنشاء عقد شجرة الإعراب بترتيب مسبق اعتمادا على القواعد المعطاة . إن السبب الذي يجعل الإعراب من الأعلى إلى الأسفل ان يقوم بالاشتقاق أقصى اليسار للجملة الدخلة w وليس الاشتقاق أقصى اليمين هو أن الجملة w تقرأ من اليسار إلى اليمين على شكل مقطع في كل مرة (token) وان الاشتقاق أقصى اليسار يولّد أوراق شجرة الإعراب السفلية (leaves) بترتيب من اليسار إلى اليمين والذي يطابق ترتيب عملية فحص الجملة المدخلة .

مثال5.1: لو كانت لدينا القواعد التالية:

$G=(\{ S ,S1 ,D ,A ,B\},\{0 ,1, 2,….,9, + ,-\}, P ,S)$

$P=\{S \rightarrow S1 \mid A$

$S1 \rightarrow D\ S1 \mid D$

D →0 |1 | 2| 3| 4 |….|9

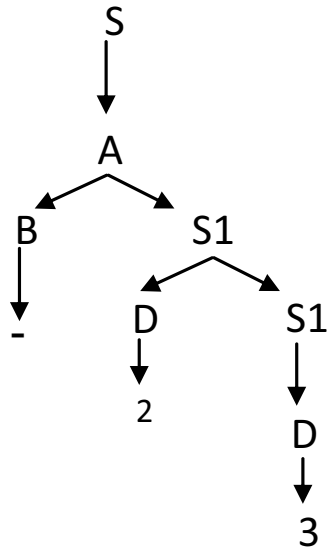
A →B S1

B→+ | - }

وكانت لدينا الجملة **$23-**=w فعند إعراب هذه الجملة بطريقة الإعراب من الأعلى إلى الأسفل نبدأ برمز البداية ونستمر بالاشتقاق أقصى اليسار إلى أن نصل إلى الجملة المراد إعرابها وكما يلي:

S → A → BS1 → -S1 → **-DS1** → -**2**S1 → -2**D** → -**23**

وبذلك تم إعراب الجملة بطريقة الإعراب من الأعلى إلى الأسفل عن طريق البدء برمز البداية وأستمر الاشتقاق إلى أن تم الوصول إلى الجملة المراد إعرابها، وان شجرة الإعراب المتوّلدة من الاشتقاق أقصى اليسار مبيّنة في الشكل (5.1).



شكل (5.1) شجرة الإعراب المتولدة من الاشتقاق أقصى اليسار

مثال5.2: لو كانت لدينا القواعد التالية:

E → E+E | E-E | E*E | E/E | id

وكانت لدينا الجملة w=**id+id*id**$ فعند إعراب هذه الجملة بطريقة الإعراب من الأعلى إلى الأسفل نبدأ برمز البداية ونستمر بالاشتقاق أقصى اليسار إلى أن نصل إلى الجملة المراد إعرابها وكما يلي:

E → **E+E** → **id**+E → id+**E*E** → id+**id***E → id+id***id**

وبذلك تم إعراب الجملة بطريقة الإعراب من الأعلى إلى الأسفل عن طريق البدء برمز البداية واستمرار الاشتقاق إلى أن تم الوصول إلى الجملة المراد إعرابها.

مثال5.3 : لو كانت لدينا القواعد التالية والمكتوبة بصيغة الـ BNF:

<integer>:= <signed > | <unsigned>

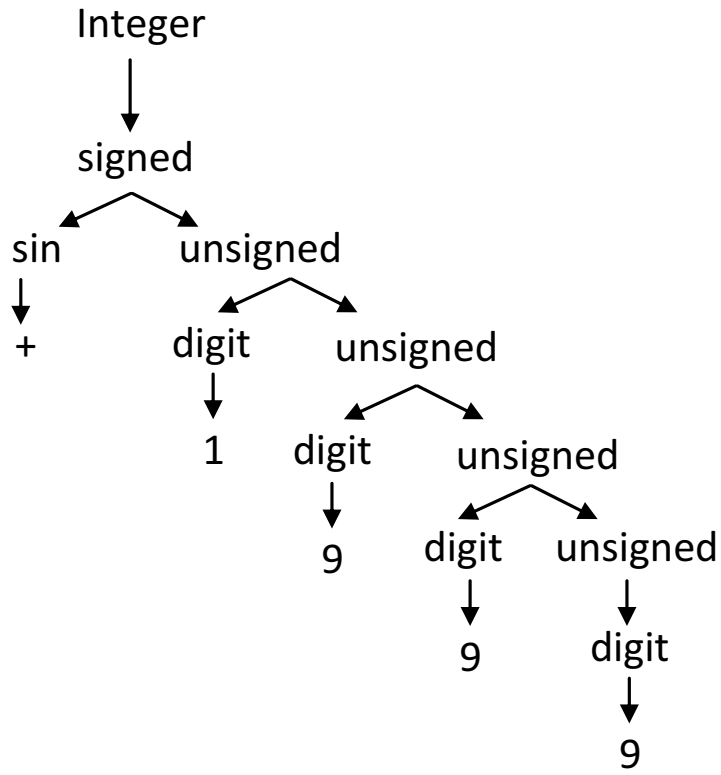
<signed>:=<sin> <unsigned>

<unsigned>:=<digit> <unsigned> | <digit>

<digit>:= 0 |1 | 2| 3| 4 |….|9

<sin>:= + |-

وكانت لدينا الجملة w=**+1999**$ فعند إعراب هذه الجملة بطريقة الإعراب من الأعلى إلى الأسفل نبدأ برمز البداية ونستمر بالاشتقاق إلى أن نصل إلى الجملة المراد إعرابها. الشكل (5.2) يبين شجرة الإعراب المتولدة من الاشتقاق أقصى اليسار. في البداية نكّون شجرة الإعراب حيث يتم توليد هذه الشجرة في هذه المرحلة كما سيتم شرحه لاحقا.



شكل (5.2) شجرة الإعراب المتولدة من الاشتقاق أقصى اليسار للجملة **1999+**

5.2.1 التطبيق (Implementation)

ممكن تطبيق الإعراب من الأعلى إلى الأسفل عن طريق كتابة مجموعة من الإجراءات لمعالجة الجملة المراد إعرابها. ومن طبيعة هذه الإجراءات ما يلي :

- لكل رمز قابل للاشتقاق (NT) يتم بناء إجراء (procedure or function).
- يجب أن تكون هذه الإجراءات ذات طبيعة تكرارية ذاتية (Recursive Procedures).
- أن تكون مرحلة المحلل اللفظي قد نفذّت.

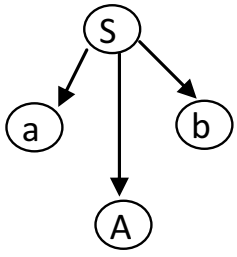
وان يكون هناك مؤشر يؤشر على المدخل لان المعرّب قد يحتاج أن يرجع إلى بداية قاعدة الإنتاج فبالتالي يرجّع المؤشر إلى المكان المقصود من المدخلات في حالة التراجع

(Backtracking). يسمى هذا النوع من المعرّب بالـ(recursive descent parser) .
فلو كانت لدينا القواعد التالية:

S→aAb

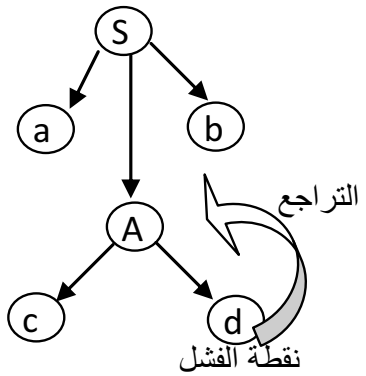
A→cd | c

وكانت لدينا الجملة w=**acb**$ فان المعرّب أعلاه يوّلد شجرة إعراب تتألف من عقدة واحدة بإسم S وان مؤشّر الإدخال (input pointer) يؤشّر على أول مدخل من مخلات الجملة w ألا وهو a . المعرّب يقوم باستخدام قاعدة الإنتاج S→aAb لتوسيع شجرة الإعراب كما مبيّن في الشكل (5.3).



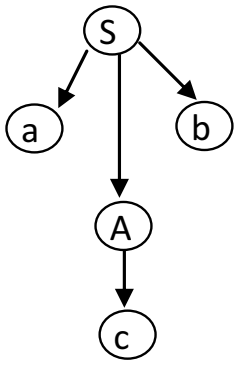
شكل (5.3) المعرّب عندما يستخدم قاعدة الإنتاج S→aAb

إن أقصى يسار ورق الشجرة يطابق أول مدخل a لذا فان المعرّب يقدّم (Advance) مؤشر الإدخال على المدخل التالي c كما يقوم بتوسيع ورقة الشجرة A مستخدما الخيار الأول للـA وهو (A→cd) كما في الشكل (5.4) وبالتالي فان المعرّب سيطابق المدخل c مع ورقة الشجرة c ليقدم مؤشّر الإدخال فيؤشّر على b ولكن لم تحصل مطابقة مع ورقة الشجرة d لذا يحصل ما يسمى بنقطة الفشل كما في الشكل (5.4) ليتم الرجوع (backtracking) إلى A وإرجاع مؤشّر الإدخال إلى المدخل الثاني c ، حينئذ يتم استخدام البديل الثاني للـ A وهو (A→c) فيقوم بتوليد الشجرة الموضّحة في الشكل (5.5)



شكل (5.4) المعرّب عندما يفشل في المطابقة

لتتم مطابقة المدخل c مع ورقة الشجرة c ثم يقدّم مؤشّر الإدخال فيؤشّر على b وتحصل مطابقة مع ورقة الشجرة b وتكون الجملة acb صحيحة قواعديا.



شكل (5.5) شجرة الإعراب للجملة acb

واليك برنامج يقوم بإعراب الجمل في اللغة المقابلة للقواعد أعلاه مع **عدم السماح للتراجع (without backtracking)**.

```
S( )
{     if ( input =='a')
       { advance ( )
```

```
        If ( A( ) != error )
            If ( input == 'b')
                { advance( );
                  If ( input == endmarker)
                      Return (success)
                   Else
                      Return (error);
                }
            Else
                Return (error);
        Else
            Return (error);
}}
A( )
{       if input =='c'
            {       advance( );
                    If (input =='d' )
                        Advance( );
            }
        Else
```

```
        Return (error);
}
Main ( )
{       append the end marker to the string w to be parsed;
        Set the input pointer to the left most token of w;
        If ( s( ) != error )
                Cout<<"عملية الاعراب تمت بنجاح";
        Else
                Cout<< "فشل عملية الاعراب";
}
```

مثال5.4: لو كانت لدينا القواعد التالية:

S→aAb

A→cd | c

واليك برنامج يقوم بإعراب الجمل في اللغة المقابلة للقواعد أعلاه مع **السماح للتراجع (with backtracking)**.

```
S( )
{   if ( input =='a')
        { advance( ) ;
          If (A( ) != error)
           {If (input == 'b')
```

```
                {       advance ( ) ;
                        if (input == end marker)
                                Return (success);
                        Else
                                Return (error);
                    }
                Else
                return (error);
            }
            else        return (error);
}}
A( )
{       save = input pointer;
        if (input =='c')
                {       advance( );
                        If (input =='d' )
                                {    Advance( );
                                     Return(success);
                                }
                          Else
```

```
                    { input pointer = save;
                      If (input =='c')
                            {   advance( );
                                Return(success);
                            }
                    }     Else    return (error);
}}
Main ( )
{       append the end marker to the string w to be parsed;
        Set the input pointer to the left most token of w;
        If ( s( ) != error )
                Cout<<"عملية الاعراب تمت بنجاح  ";
        Else
                Cout<< "فشل عملية الاعراب  ";
}
```

5.3 المعرب التنبؤي (Predictive parsing)

إن هذا النوع من المعرّب هو من نوع الإعراب من الأعلى إلى الأسفل ويحتاج إلى خزّان ادخال(Input buffer) ، مكدس والى جدول إعراب ومن خصائص هذا المعرّب أن تعاني القواعد التي يعمل عليها من مشكلة الرجوع (Backtracking) . في أول خطوة نضع رمز الـ $ داخل المكدس ونضع فوقها رمز البداية (Start symbol) كما سيتم التوضيح من خلال المثال الآتي :

مثال5.5: لو كان لدينا القواعد التالية:

exp →exp or term |term

term→ term and fact |fact

fact → not fact | (exp) |true | false

وكانت لدينا الجملة التالية **W=not(true or false) $** فهل أنها مقبولة قواعديا بالنسبة للغة المقابلة للقواعد أعلاه ؟

كما ذكرنا قبل قليل فان هذه الطريقة في الإعراب تحتاج في عملها إلى جدول الإعراب ولنفترض أن الجدول للمثال الحالي(5.5) معطى كما في الجدول 5.1(سيتم شرح بناء الجدول لاحقا).

كما اشرنا من قبل فان المكدس يحتوي في البداية على رمز الـ $ ورمز البداية وفي مثالنا فانexp هو رمز البداية ، والمدخلات تحتوي على الجملة المراد إعرابها ، أما التحركات فتكون خالية في أول خطوة، الآن يتم التقاطع بين ما موجود أعلى المكدس exp وبين أول مدخل (not) في الجدول أعلاه فنحصل على قاعدة الإنتاج exp→term exp' فيتم التعويض عن exp بما يعادلها من قاعدة الإنتاج term exp'

	not	or	and	(	)	true	false	$
exp	exp→term exp'			exp→term exp'		exp→term exp'	exp→term exp'	
exp'		exp'→term exp			exp'→ ε			exp'→ ε
term	term→ fact term '					term→ fact term'	term→ fact term'	
term'		term' →or fact term '	term' →and fact term '		term' → ε			term' → ε
fact	fact→ not fact			fact→(exp)		fact→true	fact→false	

جدول (5.1) المعرّب التنبؤي للقواعد في المثال5.5

ووضعه في المكدس ولكن بالعكس exp'term وهكذا تستمر العملية لحين الحصول على تطابق بين ما موجود أعلى المكدس مع المدخل فيتم عمل pop أي إخراج ما موجود أعلى المكدس من المكدس والانتقال إلى إعراب المدخل التالي (أي حذف ذلك المدخل من المدخلات) كما يلي :

المكدس	المدخلات	التحركات
$exp	not (true or false)$	
$exp'term	not (true or false)$	exp→ term exp'
$exp'term' fact	not (true or false)$	term→fact term'
$exp'term' fact not	not (true or false)$	fact→not fact
$exp'term' fact	(true or false)$	Pop
$exp'term')exp(	(true or false)$	fact→(exp)
$exp'term')exp	true or false)$	pop
$exp'term')exp'term	true or false)$	exp→ term exp'
$exp'term')exp'term'fact	true or false)$	term→fact term'
$ exp'term')exp'term'true	true or false)$	fact→true
$exp'term')exp'term'	or false)$	pop
$exp'term')exp'term'fact or	or false)$	term'→or fact term'
$exp'term')exp'term'fact	false)$	pop
$exp'term')exp'term'false	false)$	fact→false
$exp'term')exp'term'	)$	pop
$exp'term')exp'	)$	term' →ε
$exp'term')	)$	exp'→ε
$exp'term'	$	pop
$exp'	$	term' →ε
$	$	exp'→ε
		الجملة مقبولة

حتى نصل في الأخير إلى أن يصبح المكدس فارغا والإدخال فارغا وتكون الجملة مقبولة قواعديا في اللغة المقابلة للقواعد أعلاه وإلا فلن تكون الجملة مقبولة .

لبناء جدول المعرب التنبؤي نحتاج الى حساب دالتين الأولى تسمى first( ) والأخرى تسمى follow( ).

5.3.1 حساب دالة الـ first( )

لحساب دالة الـ first( ) نتبع ما يلي :

- إذا كانت x رمز غير قابل للاشتقاق (Terminal) فان first(x)={x}.
- إذا كانت x→ε فتضاف ε إلى first(x) .
- إذا كانت $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

فان $First(A) = First(\alpha_1) \cup First(\alpha_2) \cup \ldots \cup First(\alpha_n)$

- إذا كانت $\alpha = XYZ$ فان FIRST(α) يحسب كما يلي :

$$FIRST(\alpha) = FIRST(XYZ) = FIRST(X)$$

إذا كانت X لا تؤدي إلى ε وبمعنى آخر FIRST(X) لا تحتوي على ε .

- إذا كانت FIRST(X) تحتوي على ε فان

$$FIRST(\alpha) = FIRST(XYZ) = FIRST(X) - \{\varepsilon\} \cup FIRST(YZ)$$

أما FIRST(YZ) تحسب بنفس طريقة حساب FIRST(α) أعلاه .

والخوارزمية الرسمية لحساب دالة الـ first( ) هي :

1. If x is a terminal then first(x) is {x}
2. If X ➔ **ε** is a production then add ε to first(X)
3. If X is nonterminal and X ➔ $Y_1Y_2 \dots Y_k$ is a production then place a in first (X) if for some i , a is in first(Y_{i-1}) ; that is $Y_1Y_2 \dots Y_{i-1}$ ➔ **ε**.

If ε is in first (Y_j) for all j = 1, 2, … then add **ε** to first(X) . For example, every thing in first(Y_1) is surely in first(X) . If y_1 does not derive **ε** , then add nothing more to first(X), but if Y_1➔ **ε** , then add first (Y_2) and so on.

مثال5.6: لو كانت لدينا القواعد التالية:

G=({E,E',T,T',F},{+,*,(,),id},P,E).

P={E→TE'

E'→+TE' | ε

T→FT'

T'→*FT' | ε

F→ (E) | id }

F→(E) | id ➔ first(F)={ (, id }

E'→+TE' | ε ➔ first(E')={ +, ε}

T'→*FT' | ε ➔ first(T')={ ε,*}

$T \rightarrow FT'$ ➔ first(T)=first (F) = { (, id }

$E \rightarrow TE'$ ➔ first (E) = first (T) = { (, id }

ولإيضاح عملية حساب دالة الـ first للعناصر القابلة للاشتقاق للقواعد في المثال(5.6) أعلاه : لو أخذنا القاعدة التالية

$E \rightarrow TE'$

فيجب أن نضع First(T) في first(E) ثم نتوقف لان First(T) لا تحتوي على ε.

ولو أخذنا القاعدة التالية: $T \rightarrow FT'$ فيجب أن نضع First(F) في first(T) ثم نتوقف لان First(F) لا تحتوي على ε.

ولو أخذنا القاعدة التالية: $F \rightarrow (E)$ فيجب أن نضع { (} =( )First في first(F) ثم نتوقف لان ( )First لا تحتوي على ε.

ولو أخذنا القاعدة التالية : $F \rightarrow id$ فيجب ان نضع { id } =(id)First في first(F) ثم نتوقف لان (id)First لا تحتوي على ε.

ولو أخذنا القاعدة التالية : $E' \rightarrow +TE'$ فيجب ان نضع { + } =(+)First في first(E') ثم نتوقف لان (+)First لا تحتوي على ε.

ولو أخذنا القاعدة $E' \rightarrow \varepsilon$ فان ε تضاف إلى first(E') حسب القواعد التي وضعناها في البداية .

ولو أخذنا القاعدة $T' \rightarrow *FT'$ فان * تضاف إلى first(T')

ولو أخذنا القاعدة $T' \rightarrow \varepsilon$ فان ε تضاف إلى first(T')

مثال5.7: لو كانت لدينا القواعد التالية:.G=({S,S',E},{i,t,a,b,e},P,S)

S→iEtSS1|a

S1 → eS |ε

E→ b

فان:

First(S)={i , a }.

First(S1)={ e , ε}.

First(E)={ b }.

مثال5.8: لو كانت لدينا القواعد التالية

S→ ABCD

A→ a |ε

B→ b

C→ c | ε

D→ d | ε

فان:

First(S) = {a , b}

First(A) = {a, ε}

First(B) = {b}

First(C) = {c, ε}

First(D) = {d, ε}

لحساب first(S) يجب حساب First(A) ووضعه في first(S) ولان First(A)={a, ε} أي يحتوي على ε فيجب ان لا نتوقف لنحسب بعدها First(B) ونضيفه على first(S) ثم نتوقف لان First(B)={b} ولا تحتوي على ε لذا فان :

First(S) = first(A) - { ε} ∪ first(B)

مثال5.9: لو كانت لديك القواعد التالية

S → ACB | CbB | Ba

A → da | BC

B → g | ε

C → h | ε

فاحسب دالة الـ FIRST لكل الرموز القابلة للاشتقاق .

First (S) = First(ACB) ∪ First (CbB) ∪ First (Ba) …. (I)

First (A) = First (da) ∪ First (BC) = { d } ∪ First (BC) …. (II)

First (B) = First (g) ∪ First (ε) = { g, ε}

First (C) = First (h) ∪ First(ε) = { h, ε}

First(BC) = First(B) – { ε } ∪ First (C)

= {g, ε}-{ ε } ∪ {h, ε} = {g, h, ε}

وبالتعويض في (II) نحصل على:

First(A) = {d} ∪ {g, h, ε} = { d, g, h, ε}

First (ACB) = First (A) – { ε} ∪ First (C) – { ε} ∪ First (B)

= {d, g, h, ε} – {ε} ∪ { h, } – { ε} ∪ {g, ε}

={ d, g, h, ε}

First (CbB) = First (C) – {ε} ∪ First (bB)

= { h, ε} – {ε} ∪ {b} = { h, b}

First (Ba) = First (B) – { ε} ∪ First (a)

= { g, ε} – {ε} ∪ { a } = {g, a }

وبالتعويض في (I) نحصل على:

➔ First (S) = { d, g, h, ε} ∪ {h, b} ∪ {g, a}

= { d, g, h, b, a, ε}

5.3.2 حساب دالة الـ follow( )

ولحساب دالة الـ follow( ) نتبع ما يلي :

1. تضاف علامة الـ $ إلى follow(S) حيث أن S هي رمز البداية للقواعد.
2. إذا كانت لدينا قاعدة إنتاج على الشكل A→αBβ فتضاف first(β) عدا ε إلى follow(B).

3.1 إذا كانت لدينا قاعدة إنتاج على الشكل A→αBβ وكان first(β) يحتوي على ε فان follow(A) تضاف إلى follow(B).

3.2 إذا كانت لدينا قاعدة إنتاج على الشكل A→αB فان follow(A) تضاف إلى follow(B).

حيث ان A و B رموز قابلة للاشتقاق (Non-Terminal) وان *v α,β ε

وتكرر الخطوات السابقة إلى أن لا نحصل على عناصر جديدة تضاف إلى مجاميع الـ Follow .

و**الخوارزمية الرسمية** لحساب دالة الـ follow( ) هي :

for all nonterminal A, apply the following rules until nothing can be added to any follow set.

- Place $ in follow(S) , where S is the start symbol and $ is the input right end marker.
- If there is a production A→αBβ , then every thing in first(β) except for ε is placed in follow(B).
- If there is a production A→αB , or a production A→αBβ where first(β) contains ε (i.e β ➔ε) then every thing in follow(A) is in follow(B).

مثال5.10 : لو كانت لدينا القواعد التالية :G=({E,E',T,T',F},{+,*,(,),id},P,E) وكان

P={ E→ TE'

E'→ +TE' | ε

T→ FT'

T'→ *FT ' | ε

F→ (E) | id }

فان

first(E)={ (, id }

first(E')={ ε , + }

first(T)={ (, id }

first(T')={ ε , * }

first(F)={ (, id }

أما حساب دالة الـ follow( ) للرموز القابلة للاشتقاق للقواعد أعلاه هي :

follow(E) = { $,) }

follow(E') = { $,) }

follow(T) = { + , $,) }

follow(T') = { + , $,) }

follow(F) = { * , + , $,) }

ولتوضيح كيفية حساب دالة الـ follow وحسب القواعد الذي ذكرت آنفا :

أولا يضاف رمز $ إلى follow(E) حيث يعتبر الرمز E هو رمز البداية.

الآن نأخذ كل قاعدة إنتاج ونطبق عليها القواعد التي ذكرت سابقا، فلو أخذنا القاعدة

E→ TE'

- فـان first(E') تضـاف إلـى follow(T) عـدا ε حسـب القاعـدة 2 أعـلاه أي تضاف {+} إلى follow(T) .
- حسب القاعدة 3.1 حيث نماثل E→ TE' بالقاعدة A→αBβ بحيث نجعل T تماثل الـ B ونجعل'E تماثل الـβ ونجعل E تماثل الـA ، والقاعدة تقول إذا كانت first(β) تحتـوي علـى ε فـان follow(A) تضـاف إلـى follow(B).بمـا أن

first(β)=first(E') وان first(E') تحتوي على ε لذا يضاف ما موجود في follow(E) في follow(T) أي تضاف الـ $ إلى follow(T).

- حسب القاعدة 3.2 حيث نماثل E→TE' بالقاعدة A→αB بحيث نجعل T تماثل الـ α ونجعل'E تماثل الـB ونجعل E تماثل الـA ، والقاعدة تقول إن follow(A) تضاف إلى follow(B) لذا يضاف ما موجود في follow(E) إلى follow(E') أي تضاف الـ $ إلى follow(E').

لو أخذنا القاعدة E'→+TE' ونطبق عليها القواعد التي ذكرت سابقا:

- فان first(E') تضاف إلى follow(T) عدا ε حسب القاعدة 2 أعلاه .

- حسب القاعدة 3.1 حيث نماثل E→+TE' بالقاعدة A→αBβ بحيث نجعل T تماثل الـ B ونجعل'E تماثل الـβ ونجعل E تماثل الـA ونجعل الـ + تماثل α ، والقاعدة تقول إذا كانت first(β) تحتوي على ε فان follow(A) تضاف إلى follow(B).بما أن first(β)=first(E') وان first(E') تحتوي على ε لذا يضاف ما موجود في follow(E) في follow(T) .

- حسب القاعدة 3.2 حيث نماثل E→+TE' بالقاعدة A→αB بحيث نجعل T+ تماثل الـ α ونجعل'E تماثل الـB ونجعل E تماثل الـA ، والقاعدة تقول أن follow(A) تضاف إلى follow(B) لذا يضاف ما موجود في follow(E) في follow(E').

أما القاعدة E'→ε فلا تطبق عليها أي قاعدة لأنها لا تمثل بالصيغة A→αBβ

لو أخذنا القاعدة T→ FT' ونطبق عليها القواعد التي ذكرت سابقا:

- فان first(T') تضاف إلى follow(F) عدا ε حسب القاعدة 2 أعلاه أي تضاف * إلى follow(T) .

- حسب القاعدة 3.1 حيث نماثل T→ FT' بالقاعدة A→αBβ بحيث نجعل T تماثل الـ A ونجعل'T تماثل الـβ ونجعل F تماثل الـB ، والقاعدة تقول إذا كانت

first(β) تحتوي على ε فان follow(A) تضاف إلى follow(B).بما أن first(β)=first(T') وان first(T') تحتوي على ε لذا يضاف ما موجود في follow(T) في follow(F) .

- حسب القاعدة 3.2 حيث نماثل T→FT' بالقاعدة A→αB بحيث نجعل F تماثل الـ α ونجعلT' تماثل الـB ونجعل T تماثل الـA ، والقاعدة تقول إن follow(A) تضاف إلى follow(B) لذا يضاف ما موجود في follow(T) في follow(T').

لو أخذنا القاعدة T'→*FT' ونطبق عليها القواعد التي ذكرت سابقا:

- فان first(T') تضاف إلى follow(F) عدا ε حسب القاعدة 2 أعلاه .
- حسب القاعدة 3.1 حيث نماثل T'→*FT' بالقاعدة A→αBβ بحيث نجعل F تماثل الـ B ونجعلT' تماثل الـβ ونجعل T' تماثل الـA ونجعل الـ * تماثل α ، والقاعدة تقول إذا كانت first(β) تحتوي على ε فان follow(A) تضاف إلى follow(B).بما أن first(T')=first(β) وان first(E') تحتوي على ε لذا يضاف ما موجود في follow(T') في follow(F) .
- حسب القاعدة 3.2 حيث نماثل T'→*FT' بالقاعدة A→αB بحيث نجعل F* تماثل الـ α ونجعلT' تماثل الـB ونجعل T' تماثل الـA ، والقاعدة تقول إن follow(A) تضاف إلى follow(B) لذا يضاف ما موجود في follow(T') في follow(T').

أما القاعدة T'→ε فإنها لا تطبق عليها أي قاعدة.

والقاعدة F→(E) فان first(( ))={ ( ) } تضاف إلى follow(E) حسب القاعدة 2 أعلاه أما القاعدتين 3.1 و 3.2 فلا تطبق على القاعدة المذكورة .

والقاعدة F→id لا تنطبق عليها أي قاعدة. **نكرر الخطوات** السابقة لحين عدم الحصول على عناصر جديدة تضاف الى دوال الـfollow.

مثـال5.11 : إذا كانـت لـديك القواعـد التاليـة (G=({S,S',E},{i,t,a,b,e},P,Sفاحسـب دالتي الـ first( ) و follow( ).

P={ S→iEtS S1 | a

S1 → eS | ε

E→b }

First(S)={i , a }.

First(S1)={ e , ε}.

First(E)={ b }.

follow(S)={$, e}

follow(S1)={$, e }

follow(E)={t }

ولتوضيح كيفية حساب دالة الـ follow وحسب القواعد الذي ذكرت آنفا :

أولا تضاف $ إلى follow(S) حيث يعتبر الرمز S هو رمز البداية.

الآن نأخذ كل قاعدة إنتاج ونطبق عليها القواعد التي ذكرت سابقا، فلو أخذنا القاعدة

S→ iEtSS1

إذا اعتبرنا [α= i , B=E , β= tS S1] فان :

First (tSS1) = {t} تضاف إلـى follow (E) أمـا القاعدة 3.1 فـلا تطبـق علـى قاعدة الإنتاج أعلاه.

وعند تطبيق القاعدة 3.2 فان follow(S) تضاف إلى follow(S1).

و إذا اعتبرنا[α= iEt , B=S , β= S1] فان :

First (S1) = {e} تضاف إلى follow(S) حسب القاعدة 2 أعلاه.

أما عند تطبيق القاعدة 3.1 فان Follow(S) تضاف إلى follow(S1).

وعند تطبيق القاعدة 3.2 فان follow(S) تضاف إلى follow(S1).

أما القواعد التالية

S→ a

S1 → eS

S1 → ε

E→b

فإنها لا تطبق عليها أي قاعدة.

5.4 خوارزمية بناء الجدول للمعرّب التنبّؤي (Predictive parser Table)

1- For each production A→α of the grammar do steps 2 and 3

2- For each terminal a in first(α) add A→α to M[A,a]

3- If ε is in first(α) ,add A→α to M[A,b] For each terminal b in follow(A).if ε is in first(α) and $ is in follow(A), add A→α to M[A,$]

4- Make each undefined entry of M be error

مثال5.12 : قم ببناء الجدول الخاص بالمعرّب التنبّؤي (predictive parsing) للقواعد التالية

S→iEtS S1|a

S1 → eS | ε

E→b

لاحظ أن القواعد السابقة لا يوجد فيها تكرار ذاتي أيسر (left recursion)، لذا يسمى المعرّب أعلاه (non-recursive predictive parser).

	I	A	B	e	t	$
S	**S → iEtSS'**	**S → a**				
S'				**S' → eS** **S'→ ε**		**S1 → ε**
E			**E → b**			

ولتوضيح كيفية بناء الجدول أعلاه نطبّق الخوارزمية الخاصة ببناء الجدول على القواعد في المثال(5.12) وكما يلي:

نأخذ القاعدة S→iEtSS' ونماثل الـ S بالـ A ونماثل iEtSS' بالـ α ونحسب first(iEtSS')={i} وحسب الخوارزمية تضاف القاعدة S→iEtSS' إلى الجدول في الموضع M[S,i] .

نأخذ القاعدة S→a ونماثل الـ S بالـ A ونماثل a بالـ α ونحسب first(a)={a} وحسب الخوارزمية تضاف القاعدة S→a إلى الجدول في الموضع M[S,a] .

نأخذ القاعدة S'→es ونماثل الـ 'S بالـ A ونماثل es بالـ α ونحسب first(es)={e} وحسب الخوارزمية تضاف القاعدة S'→es إلى الجدول في الموضع M[S',e].

أما القاعدة S' → ε فلها أسلوب مختلف لان first (α) = {ε} لذا يتم حساب follow(S')={$,e} وتضاف القاعدة S' → ε إلى الجدول في المواضع M[S',e] و M[S',$]

نأخذ القاعدة E→b ونماثل الـ E بالـ A ونماثل b بالـ α ونحسب first(b)={b} وحسب الخوارزمية تضاف القاعدة E→b إلى الجدول في الموضع M[E,b]

مثال5.13 : قم ببناء الجدول الخاص بالمعرب التنبؤي (predictive parsing) للقواعد التالية

E→ T E'

E'→ + T E' | ε

T→ F T'

T'→ * F T' | ε

F→ (E) | id

لاحظ أن القواعد السابقة لا يوجد فيها تكرار ذاتي أيسر (left recursion)، لذا يسمى المعرّب أعلاه (non-recursive predictive parser).

	+	*	(	)	Id	$
E			**E→TE'**		**E→TE'**	
E'	**E'→+TE'**			**E'→ε**		**E'→ε**
T			**T→FT''**		**T→FT''**	
T'	**T'→ε**	**T'→*FT'**		**T'→ε**		**T'→ε**
F			**F→(E)**		**F→id**	

نأخذ القاعدة E→TE' ونحسب first(TE')={(,id} وحسب الخوارزمية تضاف القاعدة E→TE' إلى الجدول في الموضع M[E,id] وفي الموضع M[E,(].

نأخذ القاعدة E'→+TE' ونحسب first(+TE')={+} وحسب الخوارزمية تضاف القاعدة E'→+TE' الى الجدول في الموضع M[E',+] .

أما القاعدة E' → ε فلها أسلوب مختلف لان first (α) = {ε} لذا يتم حساب follow(E')={ $,) } وتضاف القاعدة E' → ε إلى الجدول في المواضع M[E',)] و M[E',$]

نأخذ القاعدة T→FT' ونحسب first(FT')={ (, id } وحسب الخوارزمية تضاف القاعدة T→FT' إلى الجدول في الموضع M[T,id] وفي الموضع M[T, ( ].

نأخذ القاعدة T'→*FT' ونحسب first(*FT')={*} وحسب الخوارزمية تضاف القاعدة T'→*FT' إلى الجدول في الموضع M[T',*].

أما القاعدة T' → ε فلها أسلوب مختلف لان first (α) = {ε} لذا يتم حساب follow(T')={ +, $,) } وتضاف القاعدة T' → ε إلى الجدول في المواضع M[T',)] و M[T',$] وفي الموضع M[T',+].

ولو نأخذ القاعدة F→(E) ونحسب first((E))={ (} وحسب الخوارزمية تضاف القاعدة F→(E) إلى الجدول في الموضع M[F,(].

ولو نأخذ القاعدة F→id ونحسب first(id)={ id } وحسب الخوارزمية تضاف القاعدة F→id إلى الجدول في الموضع M[F, id].

5.5 القواعد من نوع LL(1) (Left to right Left most derivation)

إن هذا النوع من القواعد يعتبر جزء من قواعد السياق الحر (CFG) والتسمّية بالـ LL(1) هي مختصر لـ **(Left to right Left most derivation)** ومعناها إعراب الجملة من اليسار إلى اليمين باستخدام الاشتقاق أقصى اليسار والرقم 1 يعني الإعراب مقطع (token) واحد في كل مرة، ويتميز بان كل خلية من خلايا الجدول المتولّد (parser table) فيها قاعدة إنتاج واحدة فقط ويشترط في هذا النوع من القواعد شرطين:

- إذا كان في القواعد قاعدة تأخذ الشكل A → α | β فان

$$FIRST(\alpha) \cap FIRST(\beta) = \varphi$$

- إذا كان FIRST(β) يحتوي على ε وكان FIRST(α) لا يحتوي على ε فان

$$FIRST(\alpha) \cap FOLLOW(A) = \varphi$$

إذا كانت أي خلية من خلايا جدول الإعراب المتولّد من القواعد تحتوي على أكثر من قاعدة إنتاج فان تلك القواعد لا تعتبر من نوع LL(1) ويسمى المعرّب (parser) حينذاك بالمعرّب غير المحدد للغة المقابلة لتلك القواعد (non-deterministic recognizer of L(G)) ، أما المعرّب (parser) الذي يعمل على

إعراب الجمل في اللغة المقابلة للقواعد التي من نوع LL(1) فانه يسمى بالمعرّب المحدد للغة المقابلة لتلك القواعد (deterministic recognizer of L(G)) .

مثال5.14: اختبر فيما إذا كانت القواعد التالية من نوع LL(1) ؟ ثم قم ببناء الجدول الخاص بالمعرّب التنبؤي (Predictive Parser)

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$

لتحقيق شروط القواعد من نوع LL(1) نقوم بما يلي :

$First(S) = first(AaAb) \cup first(BbBa)$

$= First(A) - \{\varepsilon\} \cup first(aAb) \cup first(B) - \{\varepsilon\} \cup first(bBa)$

$= \{\varepsilon\} - \{\varepsilon\} \cup \{a\} \cup \{\varepsilon\} - \{\varepsilon\} \cup \{b\} = \{a, b\}$

$First(S) = \{a,b\}.$

$First(A) = \{\varepsilon\}.$

$First(B) = \{\varepsilon\}.$

أما حساب دوال الـ follow :

$Follow(S) = \{\$\}$

$Follow(A) = \{a, b\}$

$Follow(B) = \{a, b\}$

لو ماثلنا AaAb بـ α وماثلنا BbBa بـ β في شروط القواعد المطلوبة فان :

First (AaAb) ∩ first (BbBa) = {a} ∩ {b} = φ

أما الشرط الثاني فهو متحقق أصلا .

إذن فان القواعد أعلاه من نوع LL(1).ويتم إنشاء الجدول التالي:

	A	**B**	**$**
S	**S → AaAb**	**S → BbBa**	
A	**A → ε**	**A → ε**	
B	**B → ε**	**B → ε**	

مثال5.15: اختبر فيما إذا كانت القواعد التالية من نوع LL(1) ؟

S → iEtSS' | a

S' → es | ε

لو ماثلنا iEtSS' بـ α وماثلنا a بـ β في شروط القواعد المطلوبة فان :

S → iEtSS' | a

First (iEtSS') ∩ first (a) = {i} ∩ {a} = φ

ولو ماثلنا eS بـ α وماثلنا ε بـ β في شروط القواعد المطلوبة فان :

S' → es | ε

first(eS) ∩ first(ε) = { e} ∩ { ε } = { e } ≠ φ

لذا فان القواعد أعلاه ليست من نوع LL(1) .

مثال5.16: اختبر فيما إذا كانت القواعد التالية من نوع LL(1) ؟ ثم قم ببناء الجدول الخاص بالمعرّب التنبؤي (Predictive Parser)

S1→ S #

S→ q A B C

A→ a | b b D

B→ a | ε

C→ b | ε

D→ c | ε

لتحقيق شروط القواعد من نوع LL(1) نقوم بما يلي :

First(S1) = { q } follow(S1) = {#}

First(S) = { q } follow(S) = {#}

First(A) = { a , b } follow(A) = {a, b, #}

First(B) = { a , ε } follow(B) = {b, #}

First(C) = { b , ε } Follow(C) = {#}

First(D) = { c , ε } follow(D) = {a , b, #}

لو ماثلنا a بـ α وماثلنا bbD بـ β في شروط القواعد المطلوبة فان:

A→ a | bbD

First (a) ∩ first (bbD) = {a} ∩ {b} = φ

والشرط الثاني متحقق أصلا .

ولو أخذنا القاعدة B→ a | ε

1. First (a) ∩ first (ε) = {a} ∩ {ε} = φ

2.first(a) ∩ follow(B) = { a } ∩ { b,# } = φ

ولو أخذنا القاعدة C→ b | ε

1. first(b) ∩ first(ε) = { b } ∩ { ε } = φ

2. first(b) ∩ follow(C) = { b } ∩ { # } = φ

ولو أخذنا القاعدة D→ c | ε

1. first(c) ∩ first(ε) = { b } ∩ {ε } = φ

2. first(c) ∩ follow(D) = { c } ∩ { a , b, # } = φ

اذن القواعد من نوع LL(1) ، ويتم إنشاء الجدول التالي:

	q	A	b	c	#
S	**S → S#**				
S	**S → qABC**				
A		**A → a**	**A → bbD**		
B		**B → a**	**B → ε**		**B → ε**
C			**C → b**		**C → ε**
D		**D → ε**	**D → ε**	**D → c**	**D → ε**

ولتوضيح عملية بناء الجدول نتبع الخطوات الآتية :

القاعدة S1→S# نحسب first(S#)={q} نضيف S1→S# الى M[S1,q]

القاعدة S→qABC نحسب first(qABC)={q} نضيف S→qABC إلى M[S,q]

القاعدة C → ε نحسب follow(C)={#} نضيف C → ε إلى M[C,#]

القاعدة A→a نحسب first(a)={a} نضيف A→a الى M[A,a] وهكذا بالنسبة للبقية.

مثال5.17 : لو كان لديك القواعد التالية التي من نوع LL(1) ، هل أن الجملة w=acbgfh$ مقبولة قواعديا في اللغة المقابلة للقواعد مستخدما معرّب LL(1) ؟

S → a B D h

B→ c C

C → b C | ε

D → E F

E → g | ε

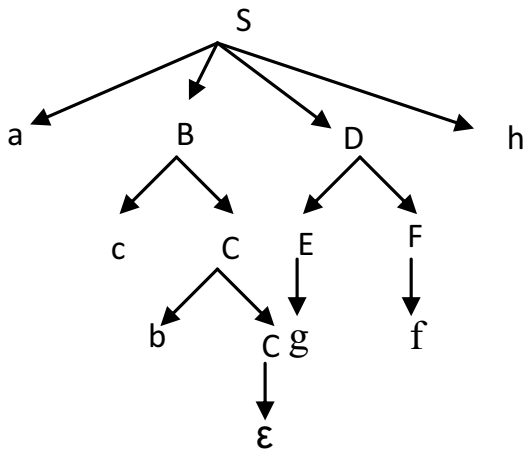
F → f | ε

	a	b	c	g	f	h	$
S	**S→aBDh**						
B			**B →**				
C		**C →**		**C → ε**	**C → ε**	**C → ε**	
D				**D →**	**D → EF**	**D → EF**	
E				**E → g**	**E → ε**	**E → ε**	
F					**F → f**	**F → ε**	

لو تم اشتقاق الجملة أعلاه باستخدام الاشتقاق أقصى اليسار (left most derivation) لحصلنا على ما يلي:

S → **a B D h** → a **c C** D h →a c **b C** D h → a c b D h → a c b **E F** h → a c b **g** F h → a c b g **f** h.

ويتم توليد شجرة الإعراب كما في الشكل (5.6).



شكل (5.6) شجرة الإعراب للجملة acbgfh

لهذا سمي هذا النوع من المعرّب بـ LL(1) لأنه يحتاج إلى عملية الاشتقاق أقصى اليسار ويعرب الجملة مبتدأ من اليسار إلى اليمين كما ذكرنا من قبل.

المكدس	المدخلات	التحركات
$S	acbgfh$	S→aBDh
$hDBa	acbgfh$	Pop
$hDB	cbgfh$	B →cC
$hDCc	cbgfh$	Pop
$hDC	bgfh$	C → bC
$hDCb	bgfh$	Pop
$hDC	gfh$	C → ε
$hD	gfh$	D → EF
$hFE	gfh$	E → g
$hFg	gfh$	Pop
$hF	fh$	F→ f
$hf	fh$	Pop
$h	h$	Pop
$	$	Accept

لاحظ في التحركات أن الاشتقاق للعناصر القابلة للاشتقاق يتم بطريقة الاشتقاق أقصى اليسار ففي البداية تم استخدام قاعدة الإنتاج S→aBDh ليتم بعدها اشتقاق B عن طريق استخدام قاعدة الإنتاج B →cC وان الـ B تؤدي الى C لذا يتم اشتقاقها باستخدام

قاعدة الإنتاج $C \rightarrow bC$ وأن الـ C تؤدي الى C لذا يتم اشتقاقها باستخدام قاعـدة الإنتـاج $C \rightarrow \varepsilon$ وهكذا بالنسبة للبقية.

5.6 استنباط المعاني من التركيب القواعدي (Syntax directed semantics)

في مرحلـة التحليل القواعدي يتم بنـاء شـجرة الإعـراب التـي من خلالهـا يمكن استنباط المعاني.

مثال5.18: لو كانت لدينا القواعد التالية:

$exp \rightarrow exp + op \mid exp - op \mid op$

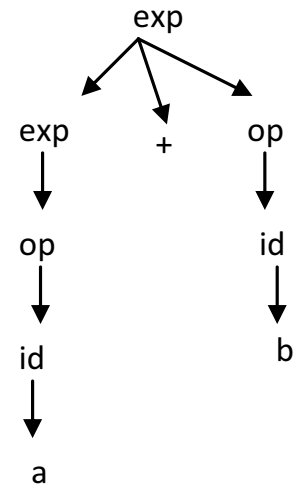
$op \rightarrow id \mid (exp)$

$id \rightarrow a \mid b \mid c$

وكانت لدينا الجملة التالية :

W=a + b$

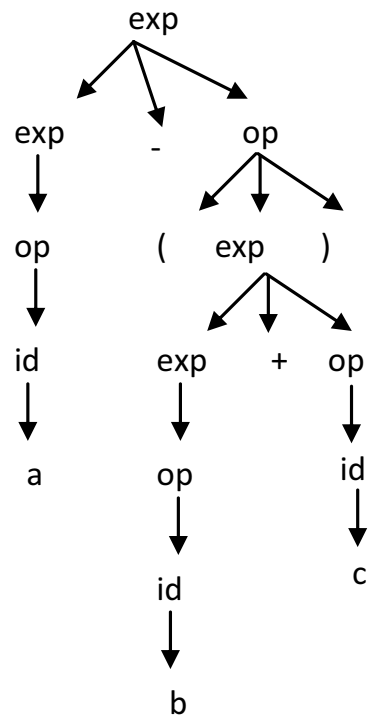
فتتكون لدينا في مرحلة التحليل القواعدي شجرة الإعراب كما في الشكل (5.7) :



شكل (5.7) شجرة الإعراب للجملة a + b

ومن الشجرة البسيطة أعلاه فانه من الواضح معنى الجملة حيث يوجد فرعين للشجرة للمعاملات وبينهما فرع للعملية وان الشجرة توضح معنى عملية الجمع بين المعاملين a و b.

أما الشجرة المقابلة للجملة W=**a-(b+c)**$ فتفسر نفسها من نفسها من خلال الأوراق المتدلية منها وهي المعاملات b و c وبينهما علامة الـ **+** ثم تحصر بين قوسين ثم تطرح من المعامل a كما مبين في شجرة المبينة في الشكل (5.8).



شكل (5.8) شجرة الإعراب للجملة a-(b+c)

مثال5.19 : لو كان لدينا القواعد التي فيها {S ,E ,F ,P ,R ,T ,L}=NT و {a =T
{ ^, / , *, -, +, (,), b, وكان S=S أما

P={S→ E| +E |-E |E-T

T→F |T *F

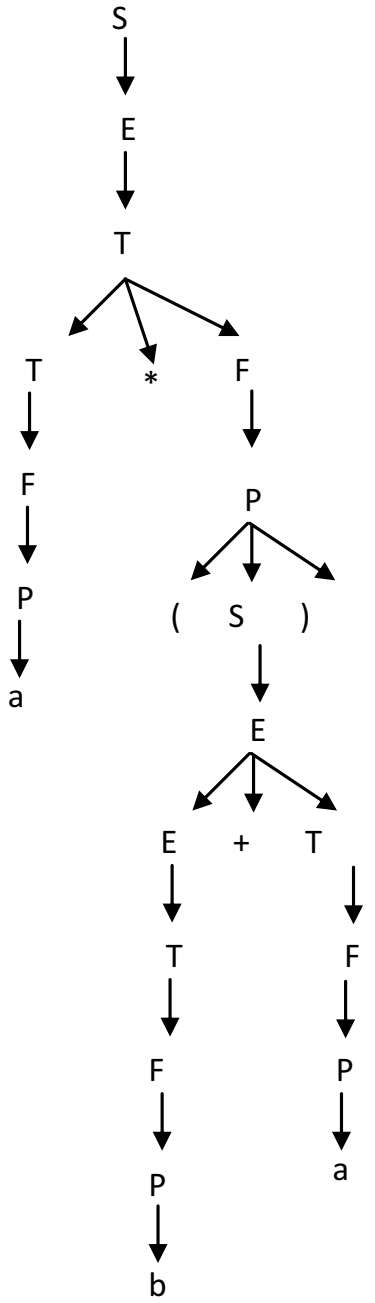
F→P| F^P

P→ b| a |(S)

E →E+T| T |T/F

L →S

R→ a(L) }



شكل (5.9) شجرة الإعراب للجملة a*(b+a)

هل أن الجملة $**a*(b+a)** مقبولة باستخدام الاشتقاق أقصى اليسار والاشتقاق أقصى اليمين وباستخدام شجرة الإعراب كيف يتم استنباط المعاني من تلك الشجرة ؟

الاشتقاق أقصى اليسار:

S → **E** → **T** → **T*F** → **F***F → **P***F→ **a***F → a***P** → a***(S)** → a*(**E**) → a*(**E+T**) → a*(**T**+T) → a*(**F**+T) → a*(**P**+T) → a*(**b**+T) → a*(b+**F**) → a*(b+**P**) → a*(b+**a**)

الاشتقاق أقصى اليمين:

S→ **E** → **T** → **T*F** → T***P** → T***(S)** → T*(**E**) → T*(**E+T**) → T*(E+**F**) → T*(E+**P**) →T* (E+**a**) → T*(**T**+a) → T*(**F**+a) → T*(**P**+a) → T*(**b**+a) → **F***(b+a) → **P***(b+a) → **a***(b+a)

أما شجرة الإعراب فهي مبينة في الشكل (5.9) وتبين عملية الجمع بين a وb وإجراء عملية الضرب بين ناتج الجمع و a.لذا فان الجملة $a*(b+a) مقبولة قواعديا في اللغة المقابلة للقواعد أعلاه .

مثال5.20: لو كان لدينا القواعد التالية:

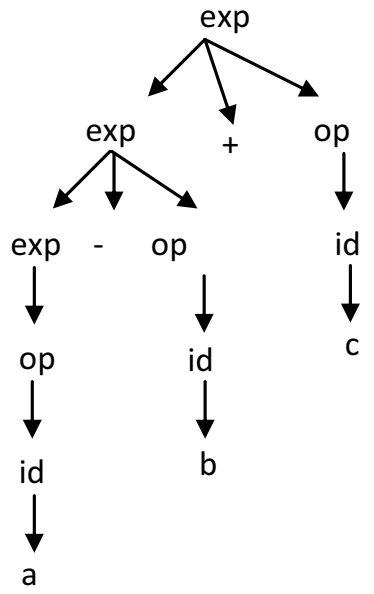
exp→ exp + op | exp –op |op

op → id | (exp)

id → a | b | c

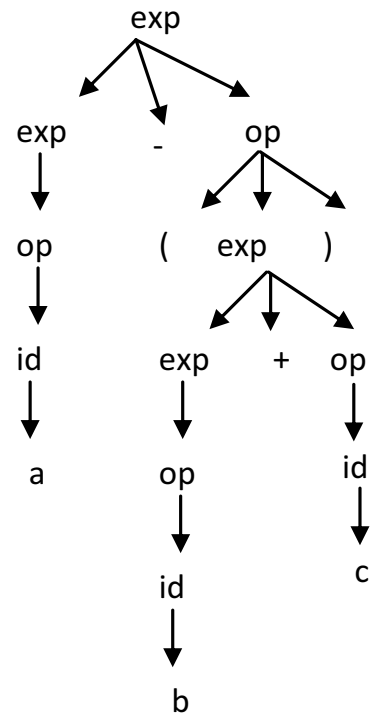
وكانت لديك الجملة $ W=**a–b+c**

ممكن أن تتكون شجرتي إعراب لنفس الجملة كما في الشكلين (5.10) و(5.11).



شكل (5.10) شجرة إعراب للجملة (a-b)+c

ان المعنى المتأتي من الشجرة أعلاه هو عملية الطرح بين المعاملين a و b ثم الجمع بين ناتج الطرح السابق مع المعامل c والأقواس في عنوان الشكل (5.10) تبين المعنى فقط .أما الشجرة المبينة في الشكل (5.11) فتبين عملية الجمع بين المعاملين b و c ثم طرح ناتج الجمع من المعامل a .



شكل (5.11) شجرة الإعراب للجملة a-(b+c)

5.7 معرّب أسبقية المعاملات (Operator precedence parser)

يتعامل هذا النوع من المعرّبات مع نوع خاص من القواعد يسمى قواعد المعاملات (Operator Grammar) لا تأخذ قواعد إنتاجه الشكلين التاليين:

1. $A \rightarrow \varepsilon$ قاعدة إنتاج تؤدي إلى ε
2. $A \rightarrow BC\alpha$ قاعدة إنتاج تؤدي إلى أكثر من رمز قابل للاشتقاق بشكل متجاور

مثال5.21: هل أن القواعد التالية من نوع قواعد المعاملات (Operator Grammar) ؟

$E \rightarrow EAE \mid (E) \mid -E \mid id$

$A \rightarrow + \mid - \mid * \mid /$

إن القواعد أعلاه ليست من نوع قواعد المعاملات (Operator Grammar) لان قاعدة الإنتاج $E \rightarrow EAE$ تحتوي من جهة اليمين في قاعدة الإنتاج الأولى في القواعد أعلاه على أكثر من رمز قابل للاشتقاق (EAE) بشكل متجاور.

يمكن للقواعد أعلاه أن تصبح من نوع قواعد المعاملات (Operator Grammar) إذا تم الاستعاضة عن A بما تؤدي إليه فتصبح القواعد كما يلي :

$E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid (E) \mid -E \mid id$

ولا يوجد فيها قواعد إنتاج من التي تم استثناؤها من قبل. من مساوئ هذا النوع من المعرّبات هو انه يتعامل مع مجموعة قليلة من القواعد (قواعد المعاملات Operator Grammar) كما انه يعاني من مشكلة معالجته للـ (-) حيث أن هناك نوعين من (-) أحداهما يحتاج الى معامل واحد ويسمى (Unary Operator) مثل E- والآخر يحتاج إلى معاملين ويسمى (Binary Operator) مثل E+E. إن هذا النوع

من المعرّب يحتاج إلى التعامل مع الأسبقيات بين المعاملات فهناك ثلاث أنواع من الأسبقيات :

<. , .> , =.

a أسبقيتها أقل من أسبقية b	a <. b
a أسبقيتها أكبر من أسبقية b	a .> b
a أسبقيتها مساوية لأسبقية b	a=.b

و لتحديد الأسبقية هناك طريقتان:

1. توزيع وأسبقية المعاملات Associatively and precedence of operator

من خلال المثال التالي سيتم شرح هذه الطريقة.

مثال5.22: لو كانت لدينا القواعد التالية:

E→E+E | E-E | E*E | E/E | (E)|-E |id

ولدينا الجملة التالية :

W=id + id*id$

وكانت هناك أسبقيات معطاة **خاصة باللغة** المقابلة للقواعد أعلاه ممثلة بالجدول الأتي:

	id	+	*	**$**
id		.>	.>	.>
+	<.	.>	<.	.>
*	<.	.>	.>	.>
$	<.	<.	<.	

فان عناصر الجملة تأخذ الأسبقيات الآتية وحسب الجدول أعلاه :

$<. id .> + <. id .> * <. id .> $

Handle Handle Handle

E + E * E

حيث يعتبر كل جزء محصور بين العلامتين (<. .>) بمثابة مقود تقليص (Handle) ويتم تحديده عن طريق البدء من أقصى يسار الجملة إلى يمينها حتى الوصول إلى أول علامة <. ثم الرجوع لليسار (للخلف) إلى أن نحصل على أول علامة .> ، وجاءت هذه التسمية من عملية تسلق الشجرة من الأسفل إلى الأعلى في كل مرة عبر مقود يقود إلى أعلى الشجرة ، كما أن كل مقود يقود إلى العنصر الأعلى ومن خلال قاعدة الإنتاج الخاصة به ، وفي مثالنا فان المقاود الثلاثة أعلاه متشابهة (<. id .>) تقود إلى العنصر الأعلى وهو E على أساس قاعدة الإنتاج (E→id) لنحصل على E+E*E ولو أخذنا العناصر الغير قابلة للاشتقاق فقط حيث أن **الأسبقيات عادة تكون بين العناصر الغير قابلة للاشتقاق** لحصلنا على ما يلي :

$ <. + <. * .> $

Handle

وتم الحصول على مقود أصله (E*E) يقود إلى E بناءا على قاعدة الإنتاج (E→E*E) فتصبح الجملة بعد التقليص E+E وإذا استبعدنا الرموز القابلة للاشتقاق كما فعلنا من قبل ونضع الأسبقيات لحصلنا على مقود أصله (E+E) يقود إلى الرمز E (رمز البداية) بناءا على قاعدة الإنتاج (E+E→E) كما يلي :

$<. + .> $

Handle

وبالتالي نكون قد بدأنا بالجملة id + id*id وانتهينا برمز البداية E .

2. الطريقة التجريبية

إذا كانت لدينا جملة تأخذ الشكل التالي:

a Handle b

فان a<. Handle وان Handle .> b وإذا كان مقود التقليص(Handle) مكون من أكثر من رمز فان الأسبقية بين رموزه هي .= حيث سيتم شرح هذه الطريقة من خلال المثال الأتي:

مثال5.23: إذا كان لدينا القواعد التالية:

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

وكانت لدينا الجملة التالية :

W = id + id $

أولا يتم اشتقاق الجملة أعلاه اشتقاق أقصى اليمين كما يلي :

$E \rightarrow \mathbf{E+T} \rightarrow E+\mathbf{F} \rightarrow E+\mathbf{id} \rightarrow \mathbf{T}+id \rightarrow \mathbf{F}+id \rightarrow \mathbf{id}+id$

ثانيا يتم تحديد مقاود التقليص (Handles) ونبدأ بالاشتقاق أعلاه ولكن من الأخير وننظر الاختلاف بين الاشتقاق الأخير id+id والذي قبله F+id فنلاحظ أن F قد أصبحت id لذا فان تلك الـ id تصبح مقود تقليص ويوضع تحتها خط للدلالة على أنها مقود وهكذا بالنسبة للبقية ، لذا نحصل على الاشتقاق أعلاه ولكن بالشكل التالي الذي يوضح عملية إيجاد مقاود التقليص:

$E \rightarrow \underline{\mathbf{E+T}} \rightarrow E+\underline{\mathbf{F}} \rightarrow E+\underline{\mathbf{id}} \rightarrow \underline{\mathbf{T}}+id \rightarrow \underline{\mathbf{F}}+id \rightarrow \underline{\mathbf{id}}+id$

وحسب القاعدة التي تم ذكرها انفا (**a Handle b**) سيتم تحديد الأسبقيات وكما يلي :

Id .> + , F.> + , T.> + , +<. id , +<. F , E =. + , +=. T

وان في هذه الطريقة التجريبية تكون (أي عنصر.>$) وان ($.< أي عنصر).

من الملاحظ انه إذا كان E =. + , +=. T فان E=.T علما أن الاستفادة من الأسبقيات أعلاه هي فقط الأسبقيات بين العناصر الغير قابلة للاشتقاق.

أما **الخوارزمية** القانونية لطريقة إعراب أسبقيات المعاملات هي :

1. initially the stack contains $ and input buffer contains W$.
2. setup input buffer pointer to the first symbol in the W$
3. repeat forever
4. if $ on top the stack and input pointer to $ then stop. Else
5. begin
6. let a be the topmost symbol on the stack and let b be the symbol pointed to by the input pointer
7. if a<.b or a =. b then begin (* shift *)
8. push b onto the stack
9. advance input pointer to the next symbol end
10. else if a.>b then (* Reduce *)
11. repeat

12. pop the stack until the top stack terminal is related <. To the terminal most recently poped

13. else error

إن هذه الطريقة في الإعراب تستخدم أيضا عمليتي الـ Shift والـ Reduce التي تم شرحها سابقا لكن بشكل مختلف قليلا وان هذه الطريقة تحتاج إلى مكدس توضع في بدايته علامة الـ $ وهي علامة نهاية الجملة (end marker) المراد إعرابها وخزان توضع فيه المدخلات (الجملة المراد إعرابها) والعلاقة (Relation) التي تحدد العلاقة بين ما موجود أعلى المكدس والمدخل وأخيرا تحتاج هذه الطريقة إلى الفعل (Action) الذي سيكون إما الـ Shift أوالـ Reduce فإذا كانت العلاقة بين ما موجود أعلى المكدس مع المدخل .> او .= فان الفعل (Action) سيكون الـ Shift وإذا كانت العلاقة بين ما موجود أعلى المكدس مع المدخل <. فان الفعل (Action) سيكون الـ Reduce كما موضح في المثال التالي :

مثال5.24: إذا كانت لديك القواعد التالية :

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

وكانت لديك الجملة التالية :

W= id + id $

فاستخدم طريقة إعراب أسبقيات المعاملات لإعرابها فيما إذا كانت مقبولة في اللغة المقابلة للقواعد أعلاه أم لا ؟

$E \rightarrow \underline{\mathbf{E+T}} \rightarrow E+\underline{\mathbf{F}} \rightarrow E+ \underline{\mathbf{id}} \rightarrow \underline{\mathbf{T}}+ id \rightarrow \underline{\mathbf{F}}+ id \rightarrow \underline{\mathbf{id}} + id$

id .> + , F .> + , T .> + , + <. Id , + <. F , E =. + , + =. T

المكدس	المدخلات	العلاقة	الفعل
$	id + id $	$ <. Id	shift id
$ id Id	+ id $	id .>+	reduce by F→ id
$. . Id	+ id $	$ <. +	shift +
$. + . + Id	Id $	+<. Id	shift id
$. + id . + id Id	$	id<. $	reduce by F→ id
$. + . . + . Id id	$	+ .> $	reduce byE→E+T
$ . . . + . . + . id + id	$		مقبولة

مثال5.25: إذا كانت لديك القواعد التالية :

$E \rightarrow U$

$U \rightarrow T$

$T \rightarrow U + T$

$T \rightarrow V$

$V \rightarrow F$

$V \rightarrow V*F$

$F \rightarrow i$

$F \rightarrow (E)$

وكانت لديك الجملة التالية :

$W = i * (i + i) \$$

فاستخدم طريقة إعراب أسبقيات المعاملات لإعرابها فيما إذا كانت مقبولة في اللغة المقابلة للقواعد أعلاه أم لا ؟

نشتق أولا اشتقاق أقصى اليمين ثم نحدد مقاود التقليص كما يلي :

$E \rightarrow \underline{U} \rightarrow \underline{T} \rightarrow \underline{V} \rightarrow \underline{V*F} \rightarrow V*\underline{(E)} \rightarrow V*(\underline{U}) \rightarrow V*(\underline{T}) \rightarrow V*(\underline{U+T}) \rightarrow V*(U+\underline{V}) \rightarrow V*(U+\underline{F}) \rightarrow V*(U+\underline{i}) \rightarrow v*(\underline{T}+i) \rightarrow V*(\underline{V}+i) \rightarrow V*(\underline{F}+i) \rightarrow V*(\underline{i}+i) \rightarrow \underline{F}*(i+i) \rightarrow \underline{i}*(i+i)$

ثم نجد الأسبقيات على أساس مقاود التقليص التي تم الحصول عليها من الخطوة السابقة:

i .> * , F .> * , (<. i , i.>+ , (<. F , F .> + , (<. V

V .> + , (<. T , T .> + , + <. i , i.>) ,+ <. F , F .>)

+ <. V , V .>) , (<. U , U =. + ,+ =.T ,T .>) , (<.U, U .>) , (<.T, T .>), *<. (,(=. E , E =.) , V =. * , *=.F

ثم نطبق الخوارزمية الخاصة بذلك كما يلي:

الفعل	العلاقة	المدخلات	المكدس

$	i*(i + i) $	$ <. i	shift i
$i i	*(i + i) $	i.>*	reduce by F→i
$. . i	*(i + i) $	$ <. *	shift *
$. * . * I	(i + i) $	*<. (	shift (
$. * (. * (I	i+i)$	(<. i	shift i
$. * (i . * (i i	+i)$	i.> +	reduce F→i
$. * (. . * (. i i	+i)$	(<. +	shift +
$. * (. + . * (. + i i	i)$	+<. i	shift i

$. * (. + i . * (. + i i i	)$	i.>)	reduce F→i
$. * (. + . . * (. + . i i i	)$	+.>)	reduce T→U+T
$. * (. . * (. i . + . . + . i i	)$	(=)	shift)
$. * (.) . * (.) i . + . . + . i i	$	) .>$	reduce F→(E)
$. * . . * . i (.) . + . . + . i i	$	* .>$	reduce V→V*F

مقبولة		$	$ . . * . . * . i (.) . + . . + . i i

مثال5.26 : إذا كانت لديك القواعد التالية :

S→L=R

S→R

L→*R

R→L

L→id

وكانت لديك الجملة التالية :

W= *id = *id $

فاستخدم طريقة إعراب أسبقيات المعاملات لإعرابها فيما إذا كانت مقبولة في اللغة المقابلة للقواعد أعلاه أم لا ؟

S→ **L=R** → L=**L** → L=***R** → L=***L** → L=***id** →***R**=*id → ***L**=*id → ***id**=*id

*<.id , id.>= , *<.L , L.>= , *=.R , R.>= , *<.id

<.L , =<. ,* =. R , = <.L , L=. =, = =. R

المكدس	المدخلات	العلاقة	الفعل
$	*id = *id $	$ <. *	shift *
$* *	id = *id $	*<.id	shift id
$* id * id	= *id $	id.>*	reduce L→ id
$* . * . id	= *id $	*.>=	reduce L→ *R
$ . . * . * . id	= *id $	$<.=	shift =
$. = . = * . * .	*id $	=<.*	shift *

id			
$. = * . = * * . * . id	id $	*<.id	shift id
$. = * id . = * id * . * . id	$	id.>$	reduce L→ id
$. = * . . = * . * . id * . id	$	*.>$	reduce L→ *R
$. = . . = . * . * . * . * . id id	$	=.>$	reduce S→ L=R
$.	$		مقبولة

. . = . . = . * . * . * . * . id id			