1.3 التحليل النحوي(Syntax Analysis)

تُعد مرحلة **التحليل النحوي** المرحلة الثانية في عمل المترجم، وتأتي مباشرة بعد مرحلة **التحليل المعجمي** .**(Lexical Analysis)** فبمجرد أن يتم تحليل البرنامج المصدر واستخراج سلسلة الرموز (Tokens)، تنتقل هذه الرموز إلى **محلل الجمل (Syntax Analyzer)** ليقوم بفحص البنية النحوية للجمل البرمجية.

حيث ان الهدف من التحليل النحوي هو التحقق من أن تسلسل الرموز المولَّدة من المرحلة السابقة يُكوّن **تراكيب صحيحة نحويًا** وفقًا لقواعد اللغة البرمجية المعتمدة .(Grammar) فإذا خالف الترتيب أو أحد الرموز القواعد النحوية، يتم اكتشاف الخطأ في هذه المرحلة.

الآلية المستخدمة:

- يستخدم محلل الجمل **قواعد اللغة النحوية (Grammar)** المحددة مسبقًا، والتي تكون عادة بصيغة **قواعد السياق الحر.(Context-Free Grammar – CFG)**
- يقوم المحلل ببناء **شجرة تحليل نحوي (Parse Tree)** أو ما يُعرف أحيانًا بـ **شجرة بناء .(Syntax Tree)**
- تمثل هذه الشجرة البنية التركيبية للجملة المصدرية، حيث:
  - الجذر يمثل بداية القاعدة.
  - الفروع تمثل تطبيق القواعد.
  - الأوراق (Leaves) تمثل الرموز النهائية.(Tokens)

إذا كانت سلسلة tokens المدخلة يمكن اشتقاقها باستخدام القواعد النحوية المعتمدة (أي يمكن بناء شجرة اشتقاق لها)، فإن السلسلة تكون **صحيحة نحويًا**.

أما إذا لم يمكن بناء هذه الشجرة، فإن محلل الجمل يُصدر **رسالة خطأ نحوي(Syntax Error)** ، ويتم إيقاف المعالجة أو محاولة تصحيح الخطأ حسب نوع المترجم.

1.1.3 لتمييز بين المحلل المعجمي والمحلل النحوي (Lexical Analyzer vs Syntax Analyzer)

عند بناء المترجم، يُقسّم البرنامج المصدر إلى مراحل متعددة. من أهم هذه المراحل:

- **التحليل المعجمي(Lexical Analysis)**
- **التحليل النحوي(Syntax Analysis)**

لفهم الفرق بين دور كل منهما، نُقدّم التعريفات التالية، ثم نوضحها بأمثلة واقعية.

المحلل المعجمي(Lexical Analyzer) هو الوحدة المسؤولة عن فحص **تسلسل الأحرف داخل المفردة الواحدة** (Lexeme)، والتأكد من أنها تُطابق نمطًا معروفًا مثل: معرف(Identifier) ، رقم(Number) ، كلمة محجوزة(Keyword) ، عامل(Operator) ، ...إلخ. • إذا تم التعرّف على المفردة، يُنتج المحلل المعجمي **(Token)** يمثل نوع المفردة وقيمتها. • أما إذا كانت المفردة غير صحيحة من حيث ترتيب الأحرف أو شكلها، يُصدر المحلل المعجمي **خطأ لغوي**.**(Lexical Error)**	**المحلل النحوي / (Syntax Analyzer Parser)** بعد تحويل البرنامج إلى سلسلة من tokens ، يقوم المحلل النحوي بفحص **تسلسل هذه المفردات** وفقًا لقواعد اللغة البرمجية (Grammar). • يتم بناء **شجرة تحليل نحوي Parse) Tree)**لتحديد ما إذا كانت الجملة البرمجية صحيحة من حيث البنية. • إذا كان ترتيب المفردات مخالفًا للقواعد، يُصدر المحلل النحوي **خطأ نحوي (Syntax Error)**.
مثال : int x;	
التحليل المعجمي: int ← كلمة محجوزة(Keyword) x ← معرف(Identifier) ; ← فاصل(Separator) النتيجة: جميع المفردات صحيحة لغويًا	التحليل النحوي الترتيب منطقي وصحيح حسب القواعد: type identifier ; الترتيب صحيح ولا يوجد أي خطأ.
مثال nit x;	
خطا nit ← غير معرفة ضمن الكلمات المحجوزة أو المعرفات المقبولة. النتيجة: خطأ لغوي.(Lexical Error)	لن يتم تنفيذه أصلًا، لأن التحليل المعجمي فشل. سبب الخطأ: تسلسل الأحرف في أول مفردة غير مقبول.
مثال int ;x	
كلمة محجوزة ← int فاصل ← ; معرف ← x .النتيجة: المفردات صحيحة لغويًا	ترتيب المفردات غير مطابق للقواعد (الفاصل قبل المعرف). النتيجة: خطأ نحوي

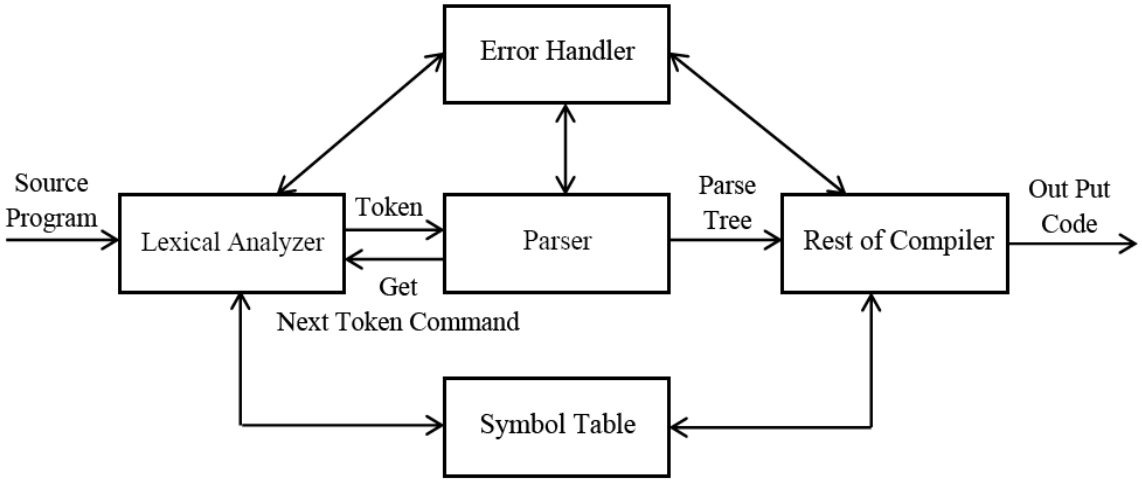
ادناه مقارنه بين التحليل المعجمي والنحوي

	Lexical Analyzer	Syntax Analyzer
المهام	فحص تسلسل الأحرف داخل المفردة (lexeme)	فحص تسلسل المفردات(tokens) ضمن الجملة
نوع البيانات	حرف → lexeme مفردة (token) →	Token → جملة → شجرة تحليل Parse Tree
الخطأ المحتمل	خطأ لغوي (Lexical Error) مثل nit بدلًا من `int`	خطأ نحوي (Syntax Error) مثل: int ;x
المدخلات	البرنامج النصي	سلسلة tokens

2.1.3وظيفة المحلل النحوي(Role of the Parser)

يُعد المُحلل النحوي (Parser) وحدة أساسية ضمن مراحل بناء المترجم، وتتمثل وظيفته في تنفيذ مرحلة التحليل القواعدي.(Syntax Analysis)

يقوم هذا البرنامج بأخذ **سلسلة من tokens (String of Tokens)** التي تم توليدها مسبقًا من قِبل **المحلل المعجمي(Lexical Analyzer)** ، ويقوم بفحص تسلسل هذه tokens للتحقق من مطابقتها للبنية القواعدية للغة البرمجية المستخدمة.



مهام المحلل النحوي:

1. **فحص التركيب النحوي :(Syntactic Structure)** يتأكد من أن tokens الواردة تُكوّن جُملًا صحيحة وفقًا لقواعد اللغة المعرفة.(Grammar)
2. **الاشتقاق من رمز البداية :**يحاول المحلل النحوي اشتقاق سلسلة tokens بدايةً من **رمز البداية (Start Symbol)**وفقًا للقواعد المحددة.(Context-Free Grammar)
3. **بناء شجرة التحليل :(Parse Tree)** إذا كانت السلسلة مقبولة قواعديًا، يقوم المحلل ببناء **شجرة تحليل نحوي**، توضح كيفية اشتقاق السلسلة وفقًا للقواعد.

4. **كشف الأخطاء النحوية:**

في حال لم يمكن اشتقاق السلسلة من القواعد المعطاة، فإن المحلل يصدر **رسالة خطأ نحوي (Syntax Error)**، تُشير إلى وجود خلل في ترتيب أو تركيب tokens.

مثال:

لو أن سلسلة التوكنات هي:

KEYWORD, int> <IDENTIFIER, x> <OPERATOR, => <NUMBER, 5> <SEPARATOR, ;>

يقوم المحلل النحوي بمحاولة مطابقتها مع القاعدة:

declaration → type identifier = expression;

إذا تطابقت، تُبنى شجرة تحليل تمثل هذا التركيب.

إذا لم تتطابق، يُصدر خطأ نحوي يحدد موقع الخلل.

3.1.3توصيف قواعد بناء الجملة(Specification of Syntax)

يتطلب وصف قواعد اللغات البرمجية دقة عالية ووضوحًا تامًا، لذا يُفضّل استخدام **القواعد الخالية من السياق** (*Context-Free Grammar - CFG*)لهذا الغرض.

ويمكن تمثيل القواعد الخالية من السياق بالشكل الرمزي التالي**G = { V, T, S, P }** :

حيث:

- **V**مجموعة الرموز غير النهائية(*Non-Terminal Symbols*)
- **T**مجموعة الرموز النهائية(*Terminal Symbols*)
- **S**الرمز الابتدائي أو رمز البدء(*Start Symbol*)
- **P**مجموعة قواعد الإنتاج(*Production Rules*)

تُكتب قواعد الإنتاج في القواعد الخالية من السياق بالشكل التالي:

$NT \rightarrow \alpha$

حيث أن:

$\alpha \in \{V \cup T \cup \lambda$

أي أن الجزء الأيمن من القاعدة (α) يمكن أن يتكون من رموز غير نهائية، أو رموز نهائية، أو السلسلة الفارغة(λ) ، أو مزيج من هذه العناصر.

4.1.3 تعريف الجملة التعريفية باستخدام القواعد الخالية من السياق (CFG)

لتمثيل الجمل التعريفية (declarative sentences) في لغة برمجة بسيطة باستخدام قواعد خالية من السياق، يمكننا تعريف القواعد كما يلي:

قواعد البناء:(Production Rules)

State → Type| List |Terminator

Type → int | float | char

List → List , id | id
Terminator → ;

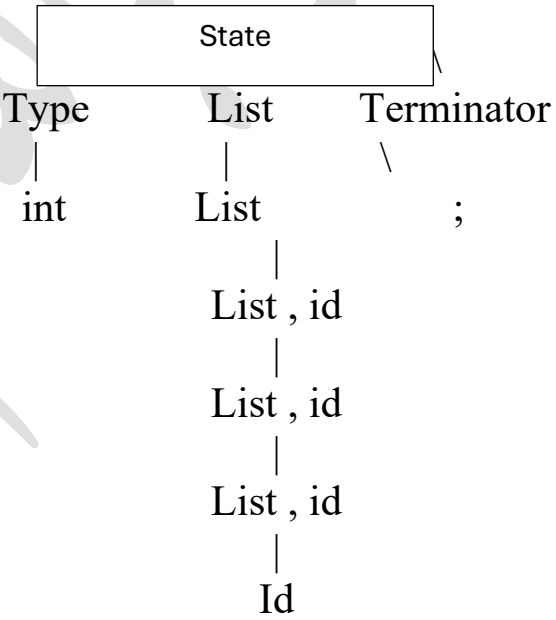
إذا كانت لدينا الجملة التالية W= int id, id, id, id;
فإن هذه الجملة يتم اشتقاقها باستخدام القواعد بالشكل التالي:

State
→ Type List Terminator
→ int List Terminator
→ int List , id Terminator
→ int List , id , id Terminator
→ int List , id , id , id Terminator
→ int id , id , id , id ;

رسم شجرة التحليل النحوي (Syntax Tree) للجملة التعريفية التالية:
int id, id, id, id;

والتي يتم اشتقاقها من القواعد التالية:

State → Type List Terminator
Type → int
List → List , id | id
Terminator → ;



2.3 القواعد الخالية من السياق:(Context-Free Grammars)

يتم وصف **التركيب النحوي** (Syntax) للغة البرمجة باستخدام **القواعد الخالية من السياق.(CFG)**
تتكون القاعدة الخالية من السياق من:

- **مجموعة من الرموز النهائية** :**(Terminals)** وهي العناصر الأساسية في اللغة، مثل الكلمات المحجوزة، المعاملات، المعرفات، إلخ.
- **مجموعة من الرموز غير النهائية** :**(Non-terminals)** وهي رموز تُستخدم لتمثيل التركيبات اللغوية المعقدة، ويمكن استبدالها بمجموعة من الرموز الأخرى وفقًا لقواعد الإنتاج.
- **رمز ابتدائي** :**(Start Symbol)** وهو الرمز غير النهائي الذي يبدأ منه الاشتقاق لتوليد الجملة.
- **مجموعة من قواعد الإنتاج** :**(Productions)** وهي القواعد التي تحدد كيفية استبدال الرموز غير النهائية برموز أخرى (نهائية أو غير نهائية) لتوليد تراكيب صحيحة في اللغة.

3.3 الاشتقاق وشجرة الإعراب(Derivation and Parse Tree)

1.3.3 شجرة الإعراب:(Parse Tree)

شجرة الإعراب هي تمثيل شجري يُظهر كيفية اشتقاق الجملة من رمز البداية باستخدام قواعد الإنتاج.
وهي توضح البنية التركيبية (Syntactic Structure) للجملة.
تبدأ الشجرة برمز البداية في الجذر، وتتفرع عبر تطبيق قواعد الإنتاج حتى تصل إلى أوراق الشجرة التي تمثل الرموز الطرفية، أي كلمات الجملة الفعلية.
كل فرع في الشجرة يُمثّل خطوة اشتقاقية، وتساعد شجرة الإعراب على فهم التركيب البنيوي للجملة وتُستخدم لاحقًا في التحليل الدلالي والترجمة وتنفيذ البرامج.
من أجل التحقق مما إذا كانت الجملة مقبولة **قواعديًا (Syntactically Correct)** ضمن لغة ما، نستخدم طريقة تُعرف باسم **الاشتقاق.(Derivation)**
تعتمد هذه الطريقة على البدء من **رمز البداية (Start Symbol)** في القواعد المعطاة، ثم نقوم بتعويض هذا الرمز باستبداله بما يقابله في **الجانب الأيمن من إحدى قواعد الإنتاج** المناسبة.
نواصل عملية الإحلال أو الاشتقاق خطوة بخطوة حتى نصل إلى سلسلة من **الرموز الطرفية (Terminal Symbols)**تماثل تمامًا الجملة المطلوب التحقق من صحتها.
إذا تمكّنا من الوصول إلى الجملة المستهدفة عبر هذه السلسلة من التعويضات، فإننا نستنتج أن **الجملة صحيحة قواعديًا**.

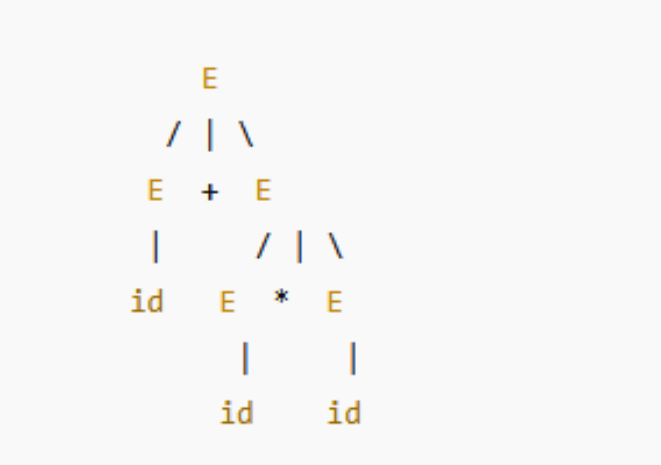
2.3.3 الاشتقاق (Derivation)

يوجد نوعان من الاشتقاق:

1. **الاشتقاق الأيسر:(Leftmost Derivation)**
   يتم في كل خطوة استبدال **أقصى رمز غير نهائي في الجهة اليسرى** من السلسلة بقاعدة إنتاج مناسبة. أي يتم اشتقاق الرموز القابلة للاشتقاق NT الكبيرة التي تقع اقصى اليسار

مثال:

$E \rightarrow E + E \mid E * E \mid id$
W= id + id *id $
E
E + E
id + E
id + E * E
id + id * E
id + id * id

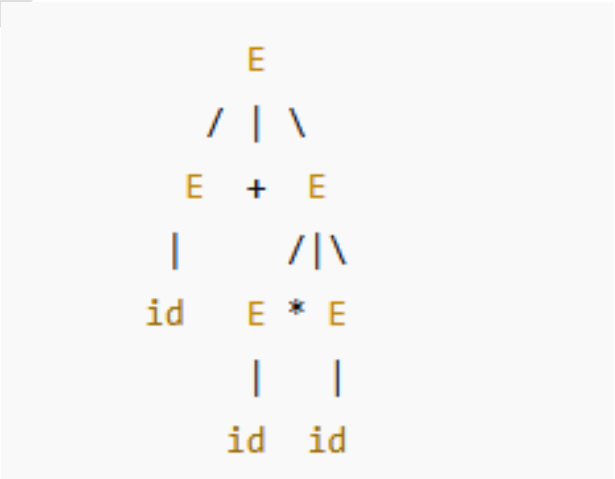


2. الاشتقاق الأيمن:(Rightmost Derivation)

يتم في كل خطوة استبدال **أقصى رمز غير نهائي في الجهة اليمنى** من السلسلة بقاعدة إنتاج مناسبة. أي يتم اشتقاق الرموز القابلة للاشتقاق NT الكبيرة التي تقع اقصى اليمين

مثال :

$E \rightarrow E + E \mid E * E \mid id$
W = id + id * id $
E
E + E
E + E * E
E + E * id
E + id * id
id + id * id



لو جاء تعبير مثل id + * idفهذا غير قابل للاشتقاق لأنه يخالف ترتيب الرموز الصحيح حسب القواعد

مثال: اعتماداً على القواعد التالية قم باشتقاق الكلمة $ W= a*(b+a)

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

نبدأ من Sونشتق التعبير a * (b + a)بتتبع الرموز اليسارية أولًا:

السلسلة

S
E
T
T * F
F * F
P * F
a * 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)

واجب :الحل بطريقة RMD

مثال : اعتماداً على القواعد التالية قم باشتقاق الكلمة **W = aaabbabbba$**

S → aB | bA
A → a | aS | bAA
B → b | bS | aBB

LMD

S
→ aB
→ a aBB
→ a a b B
→ a a b b S
→ a a b b b A
→ a a b b b bAA
→ a a b b b b a A
→ a a b b b b a bAA
→ a a b b b b a b b A
→ a a b b b b a b b a

→aaabbabbba$

RMD

S
→ aB
→ a aBB
→ a a B bS
→ a a b bS
→ a a b b bA
→ a a b b b bAA
→ a a b b b a bAA
→ a a b b b a b bA
→ a a b b b a b b a
→ aaabbabbba
→aaabbabbba$>>

3. 4 انواع المحللات النحوية:(Types of Parsers)

يُصنَّف المحلِّل النحوي بشكلٍ رئيسي إلى فئتين رئيسيتين، هما:

1. المحلِّل من الأعلى إلى الأسفل:(Top-Down Parser)

هو المحلل الذي يبدأ من **رمز البداية Start Symbol** ويحاول بناء شجرة الاشتقاق من الأعلى نزولاً إلى الرموز الطرفية(Terminals) ، باختيار القواعد المناسبة التي تطابق مدخلات البرنامج من البداية. ويتميز بأنه سهل الفهم والتنفيذ، مناسب للغات البسيطة. يستخدم في تقنيات مثل Recursive Descent Parser. ومن العيوب التي يعاني منها هو التكرار الأيسر.(Left Recursion) ويحتاج إلى استرجاع (Backtracking)في بعض الحالات ويكون أقل كفاءة مع اللغات المعقدة مقارنةً بالـ Bottom-Up Parsers.

أنواع Top-Down Parsers:

1. Recursive Descent Parser
   - يعتمد على دوال متكررة.(Recursive Functions)
   - بسيط لكن قد يحتاج استرجاعًا كثيرًا.

2. Predictive Parser (LL Parser)
   - لا يحتاج إلى استرجاع إذا كانت القواعد مناسبة) خالية من التكرار الأيسر والـ Ambiguity).
   - يعتمد على جداول التنبؤ (Parsing Table).

2. المحلِّل من الأسفل إلى الأعلى:(Bottom-Up Parser)

هو المحلل الذي يبدأ من **الرموز الطرفية (input tokens)** ويحاول إعادة بناء شجرة الاشتقاق من الأسفل صعودًا إلى رمز البداية، وذلك عبر تطبيق عمليات الاختزال.(Reduction) ويمتاز بأنه :

1. **أقوى وأكثر عمومية** من Top-Down يستطيع تحليل معظم لغات الـ.(CFG)
2. يتعامل مع **التكرار الأيسر** بدون مشاكل.
3. لا يحتاج Backtracking خصوصًا في.(LR Parsers)
4. يُستخدم فعليًا في المترجمات الحديثة مثل (C, Java بسبب قوته.)

ومن عيوبه أكثر تعقيدًا في الفهم والتنفيذ.

1. يحتاج إلى **جداول Parsing معقدة**)خاصة.(LR
2. صعب التطبيق يدويًا مقارنةً بـ.Top-Down

أنواع Bottom Up Parsers:

1. Shift-Reduce Parser

- أبسط نوع من المحللات السفلية.
- يعتمد على مكدس (Stack) +المدخل.(Input Buffer)

- عملياته الأساسية:
  1. **Shift**: نقل رمز من المدخل إلى المكدس.
  2. **Reduce**: استبدال مجموعة رموز في المكدس بالجانب الأيسر لقاعدة إنتاج.
  3. **Accept**: إذا تحولت الرموز في المكدس إلى S وتم استهلاك المدخل ← السلسلة مقبولة.
  4. **Error**: إذا لم يوجد اختزال أو نقل ممكن.

2. LR Parsers (Left-to-right, Rightmost derivation in reverse)

هذه أقوى وأشهر فئة من الـBottom-Up Parsers ، وتنقسم إلى:

أ. SLR (Simple LR) Parser

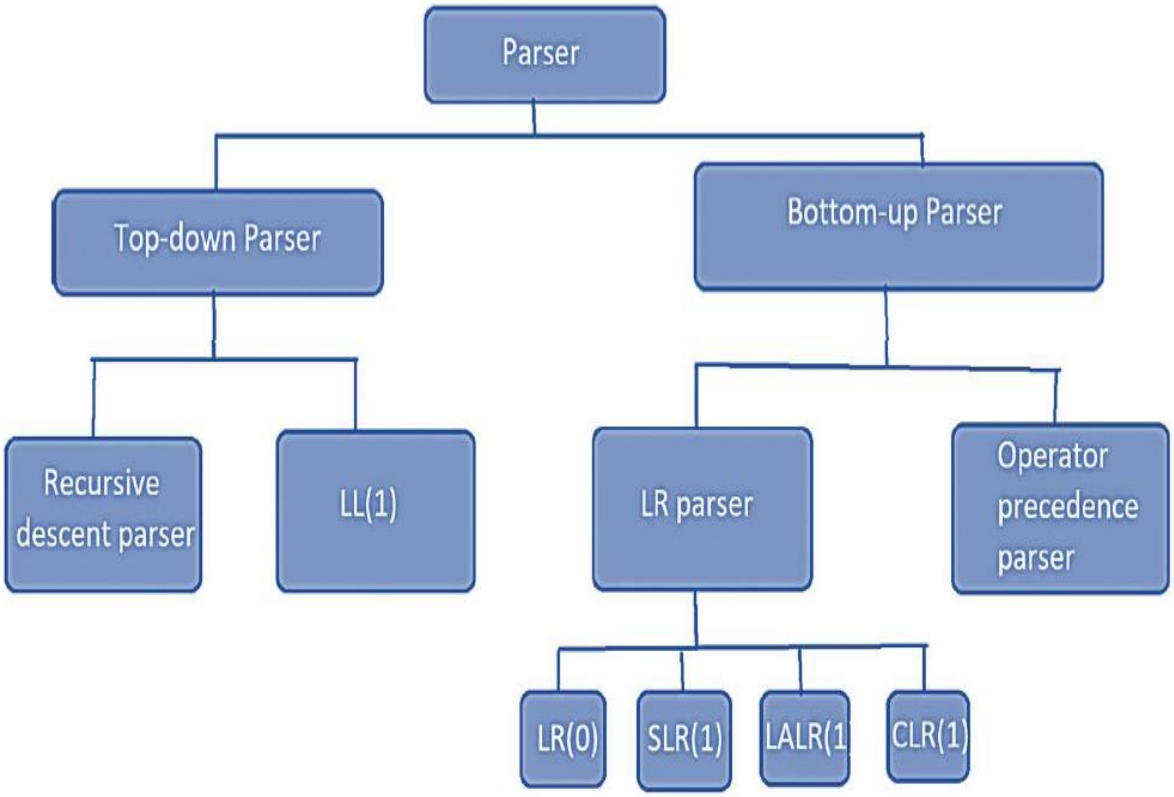
- أبسط أنواع LR.
- يعتمد على جداول **ACTION** و **GOTO**.
- سهل الإنشاء لكن أقل قوة (يفشل مع بعض اللغات).

ب. Canonical LR Parser (CLR)

- الأقوى والأكثر عمومية بين جميع LR.
- يمكنه تحليل معظم لغات **CFG**.
- لكن حجم الجداول الناتجة يكون كبير جدًا ومعقد.

ج. LALR (Look-Ahead LR) Parser

- يجمع بين قوة **Canonical LR** وبساطة **SLR**.
- أصغر في حجم الجداول لكنه يغطي معظم اللغات العملية.
- أكثر نوع مستخدم في بناء المترجمات (مثل أداة **YACC**



العنصر	المحلل من الأسفل إلى الأعلى (Bottom-Up)	المحلل من الأعلى إلى الأسفل (Top-Down)
آلية العمل	يبدأ من المدخل (Tokens) ويحاول اختزاله إلى S.	يبدأ من رمز البداية (S) ويحاول توليد المدخل.
طريقة بناء الشجرة	يبني شجرة التحليل من الأوراق إلى الجذر.	يبني شجرة التحليل من الجذر إلى الأوراق.
التكرار الأيسر (Left Recursion)	يتعامل معه بسهولة.	لا يستطيع التعامل معه.
الـBacktracking	لا يحتاج عادةً.	يحتاج في بعض الحالات) إلا إذا كان Predictive Parser).
التعقيد	أعقد ويتطلب جداول (مثل LR Parsing Table).	سهل الفهم والتنفيذ (Recursive Descentبسيط).
القوة	أقوى – يمكنه تحليل معظم لغات CFG.	أضعف – لا يحلل كل لغات CFG.
الاستخدام العملي	مستخدم في المترجمات التجارية (مثل C, Java).	نادر في المترجمات الكبيرة، مناسب للغات البسيطة.

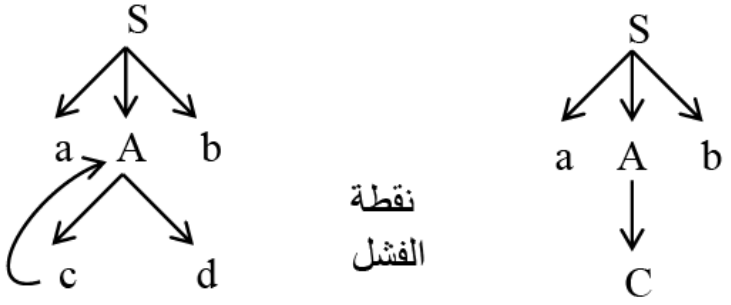
5.3 مشاكل CFG مع Top Down Parser

1.5.3التراجع(Backtracking)

عند محاولة المعرب تحليل جملة ما، قد يختار قاعدة إنتاج لا تتطابق مع إدخال المستخدم، فيضطر إلى **التراجع** والرجوع إلى رمز سابق لإعادة المحاولة باستخدام قاعدة إنتاج مختلفة.
هذا يسبب **زيادة في زمن التحليل**، وقد يؤدي إلى استهلاك موارد أكبر أو حتى الوقوع في حالات لا نهائية.

مثال:

$S \rightarrow aAb$
$A \rightarrow cd \mid c$
$W = acb \$$



إن عملية الفشل والرجوع إلى جهة اليسار من قاعدة الإنتاج تسمى. **Backtracking**
التراجع:(Backtracking) هو حالة في عملية التحليل النحوي **الاعْرابي (Parsing)** حيث يقوم المحلل (Parser) بمحاولة تحليل قاعدة إنتاج معينة من اليسار إلى اليمين، وعندما يصل إلى نقطة معينة **يكتشف فيها أن المسار الذي يسير عليه لا يؤدي إلى تطابق مع الجملة المُدخلة**، فإنه يعود (يتراجع) إلى نقطة سابقة ليختار مساراً بديلاً أو قاعدة إنتاج أخرى.

2.5.3 تكرار الذاتي(Left Recursion)

هو حالة في قواعد اللغة حيث تحتوي قاعدة الإنتاج على نفسها على جهة اليسار من جهة اليمين، مما يعني أن القاعدة تستدعي نفسها مباشرة في بداية بدائلها. وهذا يسبب مشاكل خاصة في المحللات التنبؤية(Predictive Parsers) ، لأنها قد تدخل في حلقة لا نهائية.
وينقسم الى

أ. التكرار الذاتي المباشر(Immediate Left Recursion)

يحدث عندما يكون لديك قاعدة إنتاج تأخذ الشكل التالي:

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

جزء المشكلة — جزء الخالي من المشكلة

نبدأ من الجزء الخالي من المشكلة وبعدها جزء المشكلة

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

حيث:

- Aهو المتغير (non-terminal) نفسه في بداية أحد بدائل الإنتاج (وهذا هو التكرار الذاتي).
- αهو أي سلسلة من الرموز (يمكن أن تكون رموز غير طرفية أو طرفية).\alpha
- βهو بدائل لا تبدأ بالمتغير Aأي لا تحتوي على التكرار الذاتي.\beta

مثال :

S → Sab | Scd | Sef | g | h

Sol:

S → gS′ | hS′

S′ → abS′ | cdS′ | efS′ | ε

مثال :

E → abc | def | Errx

Sol:

E′ → abc E′ | defE′

E′ → rrxE′ | ε

مثال:

S → (L) | a (No left recursion)

L → LcS | S (left recursion)

Sol:

L → SL′

L′ → cSL′ | ε

مثال :

exp → exp or term | term

term → term and factor | factor

factor → not factor | (exp) | true | false

Sol:

exp → term exp′

exp′ → or term exp′ | ε

term → factor term′

term′ → and factor term′ | ε

factor ⟶ not factor ⎪ (exp) ⎪ true ⎪ false

ب. أزالة التكرار الذاتي غير المباشر Not Immediate Left-Recursion Elimination
عبر ترتيب الرموز غير النهائية واستخدام استبدال Productions (Substitution) بشكل منظم.الهدف منها هو تحويل التكرار الذاتي غير المباشر إلى تكرار مباشر، حتى نتمكن من تطبيق الطريقة المعروفة لإزالة التكرار المباشر تكون القواعد على هذا الشكل :

$A \rightarrow B\alpha,$
$B \rightarrow A\beta$

$A \rightarrow B\alpha,$ $B \rightarrow A\beta$

عند تعويض القاعدة الثانية في القاعدة الأولى يتنج تكرار ذاتي مباشر

$S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd \mid \varepsilon$

Sol:

استبدال القاعدة للتخلص من التكرار الذاتي الغير مباشر $S \rightarrow Aa \mid b$
حدوث تكرار ذاتي مباشر $S \rightarrow Aca \mid Sda \mid b|a$
وجود تكرار ذاتي مباشر $A \rightarrow Ac \mid Sd \mid \varepsilon$
يتم التخلص من التكرار الذاتي المباشر
حدوث تكرار ذاتي مباشر $S \rightarrow Aca \mid Sda \mid b|a$
$S \rightarrow AcaS' \mid bS' \mid aS'$
$S' \rightarrow daS' \mid \varepsilon$
$A \rightarrow Ac \mid Sd \mid \varepsilon$
$A \rightarrow Sd \mid \varepsilon$
$A' \rightarrow cA' \mid \varepsilon$

مثال : قم بإزالة التكرار الذاتي المباشر وغير المباشر

$A \rightarrow B\ a \mid d$
$B \rightarrow C\ b \mid e$
$C \rightarrow A\ c \mid f$

تحديد التكرار غير المباشر:

$A \rightarrow B\ a$
$B \rightarrow C\ b$
$C \rightarrow A\ c$
$A \rightarrow B\ a \rightarrow C\ b\ a \rightarrow A\ c\ b\ a$

A يستدعي نفسه عبر B,C ويظهر التكرار الذاتي الغير مباشر

لتحويل التكرار غير المباشر إلى مباشر
خطوة 1 :ترتيب الرموز A, B, C
خطوة 2 :استبدال الرموز السابقة

في قاعدة C استبدال B: B → (A c | f) b | e
B → A c b | f b | e
في قاعدة B استبدال A: A → (A c b | f b | e) a | d
A → A c b a | f b a | e a | d

إزالة التكرار المباشر

A → A → A c b a | f b a | e a | d
A → f b a A' | e a A'| d A'
A' → c b a A' | ε

مثال:

S → AB | BA | a
A → Sa | b
B → bB | AB | a

S → AB | BA | a
A → ABa | BAa | aa | b

*نطبق القاعدة كي نتخلص من الLeft Recursion

A → BAaA' | aaA' | bA'
A' → BaA' | ε

*نعوض من A الناتجة وليس من ال A الموجودة بالسؤال:

B → bB | BAaA'B | aaA'B | bA'B | a
B → bBB' | aaA'BB' | bA'BB' | aB'
B' → AaA'BB' | ε

مثال:

S → SX | SSb | XS | a
X → Sa | Xb | b

*نلغي S الأولى فقط بطريقة.. Left

S → XSS' | aS'
S' → XS' | SbS' | ε

$X \rightarrow XSS'a \mid aS'a \mid Xb \mid b$

$X \rightarrow aSaX' \mid bX'$
$X' \rightarrow SS'aX' \mid bX' \mid \varepsilon$

3.5.3 التحليل من اليسار Left Factoring :هو أسلوب يُستخدم في تحليل قواعد الإنتاج ضمن القواعد النحوية(Grammar Rules) ، ويُطبق عند وجود أكثر من بديل لإحدى القواعد يبدأ بنفس التسلسل من الرموز.
يتمثل الهدف من هذه التقنية في إعادة صياغة القاعدة لتسهيل عملية التنبؤ (Prediction) في التحليل النحوي من الأعلى إلى الأسفل (Top-Down Parsing) ، وخاصة في تحليل(1) LL ، حيث لا يكون بالإمكان اتخاذ قرار فوري بشأن أي بديل يجب اختياره. وقواعده تأخذ الشكل التالي:
$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$
$A \rightarrow \alpha A'$
$A' \rightarrow \beta_1 \mid \beta_2$

مثال :

$S \rightarrow iEts \mid iEtses \mid a$
$E \rightarrow b$
Make the following grammar a left factored grammar
$S \rightarrow iEtsS' \mid a$
$S' \rightarrow es \mid \varepsilon$
($E \rightarrow b$ تكتب كما هي)

تم إدراج الرمز ε لأن:
أحد البدائل بعد استخراج العامل المشترك لا يحتوي على أي شيء إضافي، ولذلك نحتاج إلى بديل يمثل "لا شيء" لإكمال القاعدة بشكل صحيح.

مثال :

$Q \rightarrow aQB \mid aE \mid q$
$E \rightarrow beQ \mid b \mid bB$
$B \rightarrow b \mid \varepsilon$

$Q \rightarrow a\ Q' \mid q$
$Q' \rightarrow QB \mid E$

$E \rightarrow b\ E'$
$E' \rightarrow eQ \mid \varepsilon \mid B$
$B \rightarrow b \mid \varepsilon$

4.5.3 الغموض في القواعد النحوية (Ambiguity in Grammar)

الغموض هو الحالة التي تمتلك فيها جملة معينة في اللغة أكثر من **شجرة اشتقاق واحدة Parse) (Tree**أو أكثر من **اشتقاق نحوي مختلف (Derivation)** باستخدام نفس القواعد النحوية (Grammar).

يُقال عن الـ Grammar أنه **غامض (Ambiguous)** إذا وُجدت جملة واحدة يمكن اشتقاقها بطريقتين مختلفتين على الأقل تؤديان إلى بناء شجرتين مختلفتين من حيث البنية.

هذا النوع من الغموض غير مرغوب فيه، خصوصًا في تصميم المحللات النحوية **من الأعلى إلى الأسفل (Top-Down Parsers)**، مثلLL(1) Parser ، لأنه يؤدي إلى **صعوبة في التنبؤ وتحديد القاعدة المناسبة** أثناء التحليل.

لذلك، يجب **اكتشاف الغموض ومعالجته** أو **إعادة تصميم القواعد النحوية** لإنتاج **Grammarغير غامض.(Unambiguous Grammar)**

القاعدة تأخذ الشكل التالي

NT → NT α NT

E → E + E | E * E | id

تكونت شجرتي اعراب لنفس الجملة أي إن grammarغامض ويمكن معرفة هل grammar غامض أم لا بدون رسم شجرة الاعراب بواسطة بعض الصيغ:

1.NT → NT α NT

S → S + S

S → 2

بمعنى آخر ظهور ال non terminal في جهة اليمين من قاعدة الإنتاج في أقصى اليمين وفي أقصى اليسار، حيث أن ال non terminal تمثل الرموز القابلة للاشتقاق.

NT → NT α | β NT

E → E a | b E

baa

الاشتقاق الأول (التكرار اليساري)

E ⇒ E a

E ⇒ E a ⇒ (b E) a

E ⇒ E a ⇒ b E a ⇒ b (E a) a ⇒ b a a

الاشتقاق الثاني (التكرار اليميني)

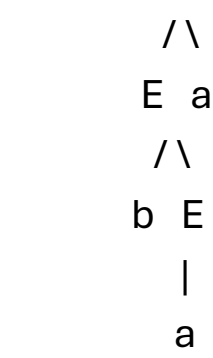
E ⇒ b E

E ⇒ b (E a)

E ⇒ b E ⇒ b (E a) ⇒ b a a

التفسير الثاني (يميني):

لتفسير الأول (يساري)



بمعنى ظهور ال non terminal في جهة اليمين من قاعدة الإنتاج مرة في أقصى اليمين ومرة في أقصى اليسار

5.5.3 إزالة الغموض باستخدام أسبقية وتوزيع العمليات (Disambiguation Using Operator Precedence and Associativity)

عند التعامل مع قواعد نحوية غامضة(Ambiguous Grammar) ، خصوصًا تلك التي تتعلق بالتعابير الرياضية أو المنطقية، من الضروري تطبيق مفهومي **أسبقية العمليات** و**تجميع العمليات** للتوصل إلى قواعد خالية من الغموض.(Unambiguous Grammar)

Left Associative	Right Associative
=	+ , -
↑	* , /
not	OR
	and

- **أسبقية العمليات(Operator Precedence)**

تعني ترتيب تنفيذ العمليات في حال تعددها ضمن التعبير.

- **تجميع العمليات(Operator Associativity)**

تشير إلى اتجاه تنفيذ العمليات المتساوية في الأسبقية، أي من **اليسار إلى اليمين** (Left Associative)أو من **اليمين إلى اليسار** .(Right Associative)

قبل التحويل من ambiguous grammar إلى un ambiguous grammar يجب أن نعرف هل αالموجودة في القاعدة

NT → NT α NT :هل هي Left associative أو Right associative من اليسار إلى اليمين، أو من اليمين إلى اليسار.

والعمليات التي تكون، Left recursion نحولها Left associative العمليات التي تكون Right .
recursion نحوله Right associative

مثال: $E \rightarrow E + E \mid E * E \mid a \mid b$

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow a \mid b$

مثال : $E \rightarrow E + E \mid E - E \mid a \mid b$
$E \rightarrow E + F \mid E - F \mid F$
$F \rightarrow a \mid b$

مثال : $E \rightarrow E + E \mid E - E \mid E * E \mid a \mid b$
$E \rightarrow E + F \mid E - F \mid F$
$F \rightarrow F * R \mid R$
$R \rightarrow a \mid b$

مثال : $R \rightarrow R + R \mid id \mid R - R \mid a \mid r\,z$
$R \rightarrow R + T \mid R - T \mid T$
$T \rightarrow id \mid a \mid r\,z$

مثال : $B \rightarrow B \uparrow B \mid a \mid b$
$B \rightarrow T \uparrow B \mid T$
$T \rightarrow a \mid b$

مثال : $A \rightarrow A * A \mid A \uparrow A \mid A - A \mid A = A \mid a \mid id$
$A \rightarrow T = A \mid T$
$T \rightarrow T - F \mid F$
$F \rightarrow F * Z \mid Z$
$Z \rightarrow R \uparrow Z \mid R$
$R \rightarrow a \mid id$

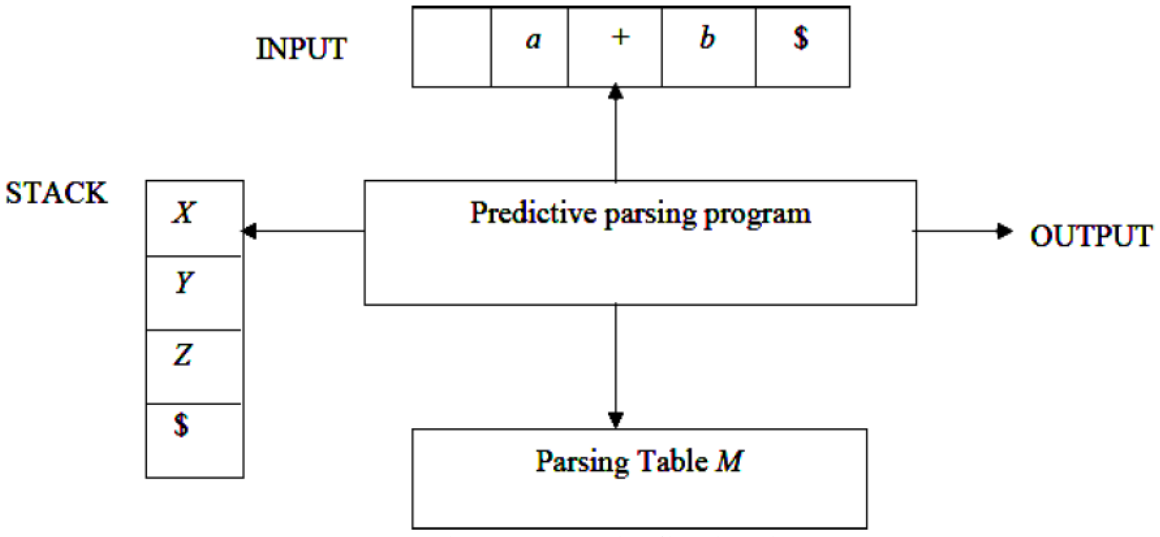
مثال : $E \rightarrow E + E \mid E * E \mid E - E \mid E / E \mid a \mid b$
$E \rightarrow E + T \mid E - T \mid T$
$T \rightarrow T * F \mid T / F \mid F$
$F \rightarrow a \mid b$

الرتبة	العملية	الرمز	نوع التجميع (Associativity)
1 اعلى اسبيقة	NOT (نفي)	!	من اليمين إلى اليسار (Right-to-Left)
2	AND (و)	&&	من اليسار إلى اليمين (Left-to-Right)
3 ادنى اسبقية	OR (أو)	||	من اليسار إلى اليمين (Left-to-Right)

6.3تحليل التنبؤي (المحلل من الأعلى إلى الأسفل)

Predictive Parsing (Top-Down Parser)

- التحليل التنبؤي هو حالة خاصة من التحليل الهابط العودي(Recursive Descent Parsing) ، حيث لا يتطلب وجود التراجع (Backtracking) أثناء عملية التحليل.
- تتمثل المشكلة الأساسية في التحليل التنبؤي في كيفية تحديد قاعدة الإنتاج (Production Rule) المناسبة لتطبيقها على رمز غير نهائي (Non-terminal) عند وجود بدائل متعددة (Alternatives)في قواعد اللغة.
- لتحقيق ذلك، يُستخدم عادةً جدول توقع (Parsing Table) يعتمد على مجموعات التوقع (FIRST Sets)ومجموعات التتبع(FOLLOW Sets) ، مما يمكّن المحلل من اختيار قاعدة الإنتاج الصحيحة بناءً على الرمز القادم (Lookahead symbol) دون الحاجة للرجوع أو التراجع.
- بهذا الأسلوب، يُصبح التحليل أكثر كفاءة وسرعة، ويُمكن تنفيذه بطريقة خطية دون الحاجة إلى تجريب عدة احتمالات، ما يجعله مناسبًا للغات خالية من الغموض والتي تحقق شروط مثل .LL(1)
- تحليل **LL(1)**هو تحليل تنبؤي من الأعلى إلى الأسفل، يعتمد على قراءة الإدخال من اليسار لليمين، ويختار قواعد الاشتقاق بناءً على أول رمز قادم فقط، مما يسمح ببناء جدول إعراب فعال وبسيط بدون رجوع.



هيكلية المحلل التنبؤي غير الرجوعي

- محلل التنبؤي غير الرجوعي هو نوع من المحللات النحوية (parsers) التي تعتمد على أسلوب التحليل التنبؤي من الأعلى إلى الأسفل (Top-Down Predictive Parsing)، ولكنه لا يستخدم الاستدعاء الذاتي (recursion) في تنفيذ خطواته، بل يستخدم مكدس (stack) وجدول إعراب (parsing table) لاتخاذ القرارات بشكل مباشر وفعال.

مكونات الهيكلية:

1. **مكدس:(Stack)**
   - يبدأ المكدس بوضع رمز النهاية $ ورمز البداية للغة S أو الرمز البادئ للقواعد.
   - يحتوي المكدس على الرموز التي يجب أن تُحلل لاحقًا (غير نهائية أو نهائية).
2. **مدخل الإدخال:(Input Buffer)**
   - يحتوي على سلسلة الرموز (tokens) التي يجب تحليلها، متبوعة برمز النهاية .$
3. **جدول الإعراب التنبؤي:(Parsing Table)**
   - جدول ثنائي الأبعاد يربط كل رمز غير نهائي (Non-terminal) مع رمز إدخال متوقع(Lookahead symbol) ، ويحتوي على قاعدة الإنتاج المناسبة التي يجب تطبيقها.
4. **وحدة التحكم:(Control Unit)**
   - تقرأ رمز الإدخال الحالي ورمز القمة في المكدس.
   - تقارن الرمزين وتتخذ القرار: هل تطابق الرمزين (في حالة رموز نهائية) أم تستبدل الرمز غير النهائي بقواعد إنتاج (من جدول الإعراب).

المزايا:

- لا يحتاج إلى استدعاءات دوال متكررة(non-recursive) ، مما يقلل استهلاك الذاكرة.
- فعال وسريع في اللغات التي تحقق شروط .**LL(1)**
- يسهل تنفيذها ببرمجة بسيطة اعتمادًا على المكدس والجدول.

1.6.3بناء معرب تنبؤي من نوع(1)LL Construction of Predictive LL(1) Parser

لبناء معرب تنبؤي من نوع **LL(1)**، يجب اتباع الخطوات الأساسية التالية:

1. حساب دالتي FIRST و FOLLOW

- دالة :FIRST تمثل مجموعة الرموز الطرفية التي يمكن أن تبدأ بها سلاسل الاشتقاق لأي رمز غير نهائي.
- دالة :FOLLOW تمثل مجموعة الرموز الطرفية التي يمكن أن تأتي مباشرة بعد رمز غير نهائي في اشتقاق الجملة.
- يتم حساب هاتين الدالتين بدقة للغة المعطاة بناءً على قواعد الإنتاج الخاصة بها

.

2. بناء جدول الإعراب التنبؤي(Parsing Table)

- يتم بناء جدول الإعراب باستخدام نتائج دالتي **FIRST**و .**FOLLOW**
- لكل رمز غير نهائي Aورمز طرفيa :

- إذا كان a في مجموعة **FIRST** للبديل α في قاعدة الإنتاج A → α، نضع قاعدة الإنتاج A → α في خانة (A, a) من الجدول.
- إذا كانت مجموعة **FIRST(α)** تحتوي على ε (الفراغ)، فإننا نضع قاعدة الإنتاج A → α في كل خانة (A, b) حيث b ينتمي إلى **FOLLOW(A)**.

- هذا الجدول يسمح للمحلل باختيار القاعدة المناسبة دون الحاجة إلى التراجع.

3. إعراب جملة الإدخال باستخدام جدول الإعراب التنبؤي

- يبدأ المحلل بقراءة أول رمز من جملة الإدخال.
- يستخدم جدول الإعراب التنبؤي لتحديد القاعدة التي يجب تطبيقها على الرمز غير النهائي الحالي في المكدس، بناءً على رمز الإدخال الحالي.
- تستمر العملية حتى يتم إعراب الجملة بالكامل أو يحدث خطأ نحوي.

2.6.3تعريف دالة First

First(α) هي مجموعة الرموز الطرفية (**Terminal symbols**) التي يمكن أن تظهر **أولاً** في أي سلسلة يمكن اشتقاقها من التعبير α.
وقد يحتوي التعبير α على رموز نهائية (**Terminals**) أو غير نهائية (**Non-terminals**) أو كلاهما

القوانين المستخدمة لحساب دالة First

1. اذا كانت x رمزا طرفيا ير قابل للاشتقاق (terminal) فأن First(X) = {x}
   E → a Frist(E) →{a}
2. اذا كانت القاعده تحتوي على ε (الرمز الفارغ) نفس القاعدة الأولى مضافا اليها ε
   X → α | ε
   أي ان Frist يكون First (X)+ ε

3. إذا كانت القاعدة تحتوي على بدائل متعددة .
. X → α1 | α2 | .. |αn

فنأخذ أول ظهور Terminal لكل حد

First(A) = First(α_1) ∪ First(α_2) ∪ ... ∪ First(α_n)
S → iEt | $\boldsymbol{\lambda}$
First (S) = {i, $\boldsymbol{\lambda}$}.

4. إذا كانت **XYZ** = α فإنَّ First (α) : يحسب بالشكل التالي
   - إذا كانت X لا تؤدي إلى $\boldsymbol{\lambda}$ بمعنى آخر First (X) لا يحتوي على $\boldsymbol{\lambda}$ فإنَّ:

First (α) = First (XYZ) ≈ First (X).

إيجاد دالة First لمجموعه من الأمثلة

مثال 1

$E \rightarrow a$
First(E) = {a}

مثال 2

$E \rightarrow a \mid \varepsilon$
First(E) = {a, ε}

مثال 3

$S \rightarrow iEt \mid \varepsilon$
First(S) = {i, ε}

مثال 4

$S \rightarrow XYZ$
$X \rightarrow x \mid \varepsilon$
$Y \rightarrow y$
$Z \rightarrow z$
First(S) = {x, y}

مثال 5

$S \rightarrow XYZ$
$X \rightarrow x \mid \varepsilon$
$Y \rightarrow y \mid \varepsilon$
$Z \rightarrow z \mid \varepsilon$
First(S) = {x, y, z, ε}

مثال 6

$S \rightarrow iEtSS1 \mid a$
$S1 \rightarrow eS \mid \varepsilon$
$E \rightarrow b$
First(S) = {i, a}
First(S1) = {e, ε}
First(E) = {b}

مثال 7

$S \rightarrow XYZ$
$X \rightarrow Y$
$Y \rightarrow y \mid \varepsilon$
$Z \rightarrow z \mid \varepsilon$

First(S) = {y, z, ε}
First(X) = {y, ε}
First(Y) = {y, ε}
First(Z) = {z, ε}

مثال 8

S → XYZa
X → Y
Y → y | ε
Z → z | ε
First(S) = {y, z, a}

مثال 9

S → XaYZ
X → Y
Y → y | ε
Z → z | ε
First(S) = {y, a}

مثال 10

G = ({E, E', T, T', F}, {+, *, (,), id}, E, P)
E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | id

First(E) = {(, id}
First(E') = {+, ε}
First(T) = {(, id}
First(T') = {*, ε}
First(F) = {(, id}

3.6.3 تعريف دالة Follow

Followللرمز غير النهائي Xهي مجموعة الرموز الطرفية (Terminals) التي يمكن أن تظهر مباشرة بعد Xفي أي شكل اشتقاقي (Sentential Form) في اللغة الناتجة من القواعد النحوية.

بمعنى آخر، إذا كان هناك اشتقاق ممكن من الرمز الابتدائي Sيؤدي إلى سلسلة تحتوي Xمتبوعًا برمز طرفي a، فإن aسيكون عنصرًا في .**Follow(X)** كما يُضاف رمز النهاية $إلى **Follow** للرمز الابتدائي إذا كان يمكن أن يظهر في نهاية السلسلة.

القوانين المستخدمة لحساب دالةFollow

رمز البداية:

1. ضع رمز النهاية $ في Follow(S)حيث Sهو الرمز الابتدائي.(أي نضع $ في القاعده الرئيسة)

A → aBz
Follow (A) = {$}

2. إذا كانت لدينا قاعدة انتاج على الشكل A → αBβ : فسوف تضاف ال First)β(عدا ε الى(Follow(B.

A → aBz
Follow (B) = {z}

3. إذا كانت لدينا قاعدة انتاج بالشكل A → αBβ :وكان First)β(يحتوي على ε فإنَّ Follow(B) تضاف إلى Follow(A)

A → aBC
C → d | ε
Follow (B) = {d, $}

ملاحظه داله Follow لا يمكن ان تحتوي علىε

4. إذا كانت لدينا قاعدة انتاج بالشكل A → αB : فإ ن Follow(A) :تضاف إلى Follow(B) حيث أ ن A و B هي رموز غير قابلة للاشتقاق.

A → aB
Follow (B) = {$}

إيجاد دالة Follow لمجموعه من الأمثلة

مثال :

S → iEtSS1 | a
Follow(S) = { $, e }
S1 → eS | ε
Follow(S1) = { $, e }
E → b
Follow(E) = { t }

مثال :

G= ({E, E', T, T', F}, {+, *, (,), id}, E, P)
E → TE'
E' → +TE' | λ
T → FT'
T' → *FT' | λ
F → (E) | id

Non-Terminal	FOLLOW
E	{ $,) }
E'	{ $,) }
T	{ +, $,) }
T'	{ +, $,) }
F	{ *, +, $,) }

مثال : حساب دالتي ال First & Follow

S → A | B C D
A → B e | E B
B → b E c | C b | D c | ε
C → c | ε
D → a | C b
E → a | b E S e | ε

Non-Terminal	FIRST	FOLLOW
S	{a, b, c, e, | ε }	{ $, e }
A	{a, b, c, e, | ε }	{ $,e }
B	{a, b, c, ε}	{c, a, b, $, e}
C	{c, ε}	{a, b, c}
D	{a, b, c}	{c, $,e}
E	{a, b, ε}	{c, a, b, $, e}

بعد ان تم استخراج ال FIRST و FOLLOW نبدأ ببناء جدول الاعراب

7.3 بناء جدول الاعراب التنبؤي Productive Parser table LL(1)

جدول الإعراب التنبؤي (LL (1) Parsing Table) هو أداة أساسية في المحلل النحوي من النوع التنبؤي.(Predictive Parser)
وظيفته يساعد المحلل النحوي على اختيار أي قاعدة إنتاج (Production) يجب استخدامها أثناء عملية التحليل، يعتمد على دالتي FIRST و FOLLOW لبناء الجدول.
يسمى LL (1) لأنه:

- L نقرأ الإدخال من اليسار إلى اليمين.
- L نُنتج المشتقة اليسرى.(Leftmost derivation)
- (1) نستخدم رمز واحد فقط من الأمام (Lookahead) لاتخاذ القرار.

1.7.3خطوات بناء جدول(1)LL

1. احسب دالة FIRST لكل قاعدة

دالة FIRST(α) تعطي مجموعة الرموز الطرفية (Terminals) التي يمكن أن تبدأ بها السلسلة المشتقة من.α

2. احسب دالة FOLLOW لكل رمز غير نهائي

دالة FOLLOW(A) تعطي مجموعة الرموز الطرفية التي يمكن أن تأتي مباشرة بعد A في أي اشتقاق.

3. املأ الجدول

- لكل قاعدة إنتاج من الشكل $A\rightarrow\alpha$ نتبع القواعد التالية:
  لكل $a\in FIRST(\alpha)$ ضع $A\rightarrow\alpha$ في الخانة M[A,a]
- إذا كانت :$\varepsilon \in FIRST(\alpha)$ لكل $b\in FOLLOW(A)$ ، ضع $A\rightarrow\alpha$ في M[A,b]
- إذا لم تنطبق أي حالة، تبقى الخانة فارغة أو نضع فيها خطأ.(Error)

مثال :

$S\rightarrow AB$
$A\rightarrow aA|\varepsilon$
$B\rightarrow bB|c$

1. حساب:FIRST

FIRST(A): { a, ε }
FIRST(B): { b, c }
FIRST(S): = {a, ε }
$A \rightarrow aA \Rightarrow a$ إذا
$A \rightarrow \varepsilon \Rightarrow FIRST(B) = \{ b, c \}$ إذا
FIRST(S) = { a, b, c } إذن

2 .حساب:FOLLOW

FOLLOW(S): { $ } ($ دائما القاعده الرئيسيه نضيف فيها (نهاية السلسلة

FOLLOW(A): بما أن S → A B ⇒ إذنB كل ما يبدأ به ، FOLLOW(A) = FIRST(B) = { b, c }

FOLLOW(B): لأن B آخر في S ⇒ FOLLOW(B) = FOLLOW(S) = { $ }

3 .بناء الجدولM

	a	b	c	$
S	S→AB	S→AB	S→AB	
A	A → aA	A → ε	A → ε	
B		B→bB	B→ c	

ملاحظات مهمة:

- هذه الخوارزمية تضمن أن:
  1. كل قاعدة إنتاجية تظهر في الجدول وفق الرموز الطرفية التي يمكن أن تبدأ.α
  2. إذا α يمكن أن ينتجε ، فإن القاعدة تظهر أيضًا في الخانات التابعة لـ.Follow(A)
- هذه الطريقة تمنع التعارضات إذا كانت اللغة **LL(1)**، أي أن كل خلية تحتوي على قاعدة واحدة فقط.

مثال:

S → iEtSS1 | a

S1 → eS1 | ε

E → b

- FIRST:
  - FIRST(E) = { b }
  - FIRST(S1) = { e, ε }
  - FIRST(S) = { i, a }
- FOLLOW:
  - FOLLOW(S) = { $ }
  - FOLLOW(S1) = { $ }
  - FOLLOW(E) = { t }
- بناء الجدول

	i	a	b	e	t	$
S	S → iEtSS1	S → a				
S1				S1 → eS1 S1→ ε		S1 → ε
E			E → b			

لوجود اكثر من قاعده انتاج في نفس الخلية معناته ان هذه القواعد ليست .LL1

مثال :

E → AB | e
A → a | λ
B → b | λ

حساب FIRST:
- FIRST(E) = { a, b, e, ε }
- FIRST(A) = { a, ε }
- FIRST(B) = { b, ε }

حساب FOLLOW:
- FOLLOW(E) = { $ }
- FOLLOW(A) = { b, $ }
- FOLLOW(B) = { $ }

الرمز	FIRST	FOLLOW
E	{ a, b, e, ε }	{ $ }
A	{ a, ε }	{ b, $ }
B	{ b, ε }	{ $ }

جدول الإعراب التنبؤي LL(1):

	a	b	e	$
E	E → AB	E → AB	E → e	E → AB
A	A → a	A → ε		A → ε
B		B → b		B → ε

بما ان لا يوجد قاعدتين في نفس الخلية معناته ان هذه القواعد هي LL1

8.3 قواعد LL(1)

هي جزء من قواعد السياق الحر(Context-Free Grammar - CFG) ، وتستخدم لتحليل الجملة من اليسار إلى اليمين باستخدام الاشتقاق الأقصى لليسار.(Leftmost Derivation)

- Lالأول :قراءة الإدخال من اليسار إلى اليمين.
- Lالثاني :إنتاج المشتقة اليسرى أولاً.(Leftmost derivation)
- 1:استخدام رمز طرفي واحد (Single Token Lookahead) لاتخاذ القرار المناسب لكل قاعدة إنتاج.

1.8.3 خصائص قواعد LL(1)

1. جدول الإعراب التنبؤي:(Parsing Table)
   - كل خلية في الجدول تحتوي على قاعدة إنتاج واحدة فقط.
   - أي تعارض في الخلية يعني أن القواعد ليست LL(1).
2. سهولة التحليل التنبؤي:(Predictive Parsing)
   - تسهل عملية بناء محلل Recursive Descentأو Table-Driven Parser.

2.8.3 الشروط الأساسية لقواعد LL(1)

1. **Unambiguous غير غامضة**
   - القواعد يجب أن تكون واضحة دون أي غموض في الاشتقاق.
   - أي جملة يمكن اشتقاقها بطريقة واحدة فقط باستخدام القواعد المعطاة.
2. **No Left Recursion لا يحتوي على تكرار ذاتي أيسر**
   - القواعد لا يجب أن تحتوي على إنتاج من الشكل:
   - التكرار الذاتي الأيسر يولد حلقة لا نهائية أثناء التحليل التنبؤي.
3. **Left Factoring لا يحتوي على عامل مشترك**
   - إزالة أي عوامل مشتركة في بداية قواعد الإنتاج لتجنب التعارض في الجدول.

3.8.3 الشروط الرياضية للتحقق من أن القواعد LL(1)

للتأكيد أن **القواعد Grammar من نوع LL(1)** وأنها لا تعاني من الغموض أو التكرار الذاتي الأيسر أو العامل المشترك، هناك شرطان أساسيان لكل قاعدة إنتاج من الشكل:

$A \rightarrow \alpha|\beta$

<u>الشرط الأول</u>

- التحقق من التداخل بين مجموعات:FIRST

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

معنى ذلك لا يوجد رمز طرفي مشترك بين بداية كل إنتاج منA , هذا يضمن أن المحلل يستطيع اختيار الإنتاج المناسب باستخدام رمز طرفي واحد فقط

<u>الشرط الثاني</u>

- إذا كان FIRST(β)يحتوي على ε وكان (FIRST(α لا يحتوي علىε ، عندها يجب التحقق من:

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

- معنى ذلك:

الرموز التي يمكن أن تبدأ α لا تتداخل مع الرموز التي يمكن أن تأتي بعد A في أي اشتقاق هذا يمنع أي تعارض في الجدول عند التعامل مع.ε

$S \rightarrow iEtSS1 \mid a$
$S1 \rightarrow eS1 \mid \varepsilon$
$E \rightarrow b$

- FIRST:

- FIRST(E) = { b }
- FIRST(S1) = { e, ε }
- FIRST(S) = { i, a }

- FOLLOW:

- FOLLOW(S) = { $,e}
- FOLLOW(S1) = { $,e{e }
- FOLLOW(E) = { t }

S:{i} ∩ {a} = ∅
{e}∩{$,e} = e≠∅

مثال :

$A \rightarrow cdB \mid \varepsilon$
$B \rightarrow b \mid Ca$
$C \rightarrow c$

. حساب FIRST:

- FIRST(A) = { c, ε }
- FIRST(B) = { b, c }
- FIRST(C) = { c }

حساب FOLLOW:

- FOLLOW(A) = { $ }
- FOLLOW(B) = { $ }
- FOLLOW(C) = { a }

جدول الإعراب التنبؤي LL(1):

	c	b	d	a	$
A	A → cdB				A → ε
B	B → Ca	B → b			
C	C → c				

اذن القواعد من نوع LL1

3.9.3 تجهيز عناصر الإعراب(Parsing Setup)

- **Stack**: يحتوي على **Start Symbol** في الأعلى متبوعًا بعلامة $ التي تمثل نهاية الإدخال.
- **Input**: تحتوي على الجملة المطلوب إعرابها، متبوعة بعلامة $.
- **Parse Table**: الجدول الذي تم إنشاؤه مسبقًا بناءً على **FIRST** و**FOLLOW**.
- **Output**: لتسجيل قواعد الإنتاج المستخدمة أثناء الإعراب.

4.9.3 خطوات الإعراب(Top-Down Parsing Algorithm)

1. ضع Start Symbol في الـStack ، وابدأ بـ Input للجملة المراد إعرابها.
2. Top of Stack = X
   - إذا كان X :Terminal
     - تحقق إذا كان X = a (رمز في Input.)
     - "عند تحقق الشرط، احذف العنصر من الـ Stack وحرّك مؤشر الإدخال إلى العنصر التالي".
     - إذا لم يتحقق الشرط ⇒ الجملة Not Accepted.
   - إذا كان X :Non-Terminal
     - ابحث في Parse Table[X, a] حيث a هو العنصر الحالي في Input.
     - إذا وجد إنتاج Pop X :من Stack و Push عناصر الإنتاج بالترتيب العكسي (Rightmost element on top).
     - إذا لم يوجد إنتاج ⇒ الجملة Not Accepted.
3. كرر العملية حتى:
   - Stack = $ و Input = $ ⇒ الجملة Accepted.
   - أي تعارض أو عدم تطابق ⇒ الجملة **Not Accepted**.

الشرط الأساسي للإعراب بطريقة (Top-Down) هو خلو القواعد من الرجوع الخلفي (Backtracking).
فإذا كانت القواعد تحتوي على الرجوع الخلفي فلا بد من التأكد من نوع الرجوع الخلفي فيما إذا كان من النوع المباشر (Immediate Backtracking) أو غير المباشر (Not-Immediate Backtracking) لكي يتم معالجته وفق الطرق التي تم شرحها مسبقاً.
نحتاج في هذه الخوارزمية إلى وجود **Stack** والعمليات الخاصة بها والتي تمثل **Push & Pop**.
يتم حساب قيمة (**First and Follow**) من أجل تكوين جدول (**Parse Table**) بعدد من الأسطر والأعمدة، حيث أن عناصر الأسطر تمثل عناصر **Non-Terminal** أما قيم أو عناصر الأعمدة فتمثل عناصر **Terminal**، حسب ما تم التطرق إليه مسبقاً.

مثال :

$A \rightarrow cdB \mid \varepsilon$

$B \rightarrow b \mid Ca$

$C \rightarrow c$

Non-Terminal / Terminal	FIRST	FOLLOW
A	c, ε	$
B	b,c	$
C	c	a

جدول LL(1) .

Non-Terminal / Terminal	c	b	a	$
A	A → cdB			A → ε
B	B → Ca	B → b		
C	C → c			

الجملة المطلوب إعرابها **cdca$**

خطوات الاعراب باستخدام Output و Input و Stack

#	Top of Stack	Input	Output
1	$A	cdca$	
2	A	cdca$	
3	cdB	cdca$	A → cdB
4	c	cdca$	A → cdB
5	d	dca$	A → cdB
6	B	ca$	A → cdB
7	C	ca$	A → cdB, B → Ca
8	c	ca$	A → cdB, B → Ca, C → c
9	a	a$	A → cdB, B → Ca, C → c
10		$	A → cdB, B → Ca, C → c
11	$	$	A → cdB, B → Ca, C → c

واجب : اعرب الكلمة التالية **$(id)** اعتمادا على القواعد ادناه

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \boldsymbol{\lambda}$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \boldsymbol{\lambda}$
$F \rightarrow (E) \mid id$

10.3(Bottom-up Parsing) التحليل من الأسفل

يُعرف التحليل من الأسفل، أو ما يسمى أيضًا بالتحليل التصاعدي، على أنه أسلوب لبناء شجرة الاشتقاق (Parse Tree)يبدأ من الرموز الطرفية (Terminals) في جملة الإدخال، ثم يعمل على تجميعها تدريجيًا نحو الأعلى حتى الوصول إلى رمز البداية (Start Symbol) للقواعد.
تتمثل الفكرة الأساسية للتحليل من الأسفل في تقليص (Reduction) جملة الإدخال w خطوة بعد أخرى، وذلك بالبحث عن جزء من السلسلة المدخلة يطابق الجهة اليمنى من إحدى قواعد الإنتاج Production) .(Rulesوعند إيجاد هذا التطابق، يتم استبداله بالجهة اليسرى للقاعدة، والتي تمثل رمزًا غير طرفي .(Non-Terminal)هذه العملية تمثل اشتقاقًا عكسيًا(Reverse Derivation) ، أي عكس عملية الاشتقاق الأمامي.
وبذلك فإن التحليل التصاعدي يُكافئ إجراء الاشتقاق الأيمن (Rightmost Derivation) ولكن بالاتجاه العكسي، حيث يبدأ من أوراق شجرة الاشتقاق (الرموز الطرفية) ويتجه نحو جذر الشجرة (رمز البداية).

1.10.3 خصائص التحليل من الأسفل

- يبدأ من الرموز الطرفية وينتهي عند رمز البداية.
- يعتمد على عمليات التقليص (Reduction) التدريجية حتى الوصول إلى الهدف.
- يعادل بناء شجرة الاشتقاق من الأسفل إلى الأعلى.
- أكثر عمومية من التحليل من الأعلى(Top-down Parsing) ، لأنه قادر على التعامل مع قواعد نحوية أعقد.

2.10.3 أشهر طرق التحليل من الأسفل

يعتمد التحليل من الأسفل غالبًا على محللات تعتمد أسلوب الإزاحة والتقليص (Shift-Reduce Parsing)، ومن أشهرها:

- LR Parser (Canonical LR)
- SLR Parser (Simple LR)
- LALR Parser (Look-Ahead LR)

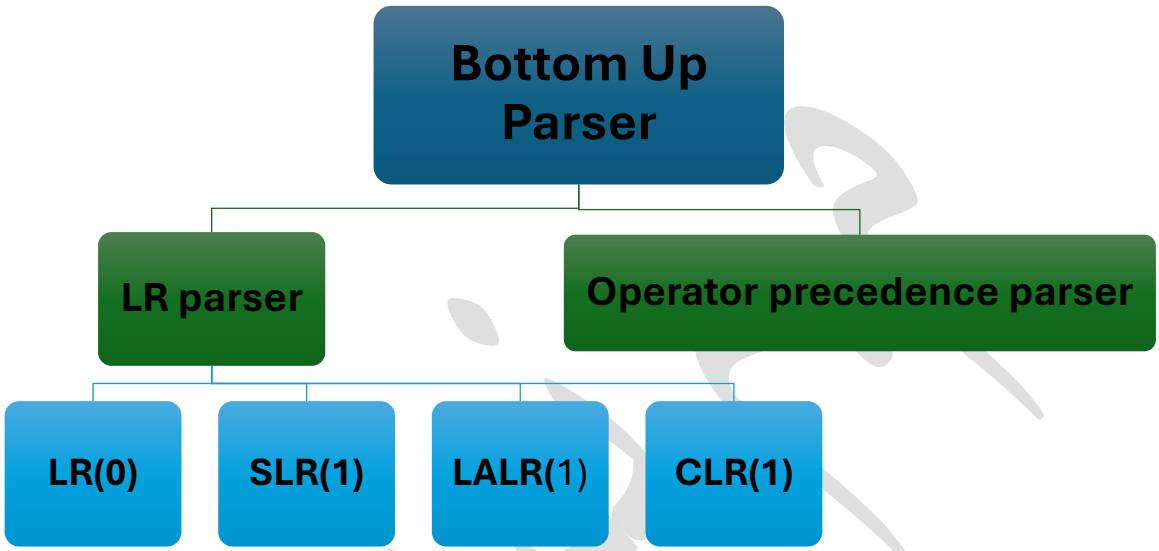
3.10.3أهمية التحليل من الأسفل

- يتميز بمرونته وقدرته على التعامل مع لغات واسعة النطاق.
- يُستخدم على نطاق واسع في بناء المترجمات الحديثة.
- يوفر أساسًا عمليًا لتصميم محللات نحوية تعتمد على الجداول(Parsing Tables)

.

نحتاج مثال

Bottom Up Parser (Shift-Reduce Parser)



Handle:11.3

هو المقطع (Substring) في جملة الإدخال الذي يطابق الجهة اليمنى لقاعدة إنتاج (Production Rule) في القواعد (Grammar) ، والذي عند إجراء التقليص (Reduction) يُستبدل بالرمز غير الطرفي (Non-terminal symbol, NT) الموجود في جهة اليسار من نفس قاعدة الإنتاج. بعبارة أخرى يعتبر Handle هو المفتاح الذي يستخدمه المحلل من الأسفل للأعلى لتطبيق قواعد الإنتاج وتقليص الجملة تدريجيًا نحو الجذر. حيث بدون تحديد الـ Handle بدقة، لا يمكن لمحلل Bottom-Up أن يعرف أي جزء من الإدخال يجب تقليصه أولًا.

يوجد أسلوب عام من التحليل من الأسفل يُسمى التحليل بالإزاحة والتقليص (Shift-Reduce Parsing)، والذي يقوم على مبدأ:

- Shift :إدخال الرموز من جملة الإدخال تدريجيًا في المكدس.(Stack)
- Reduce: تقليص مقطع من المكدس إذا كان يطابق قاعدة إنتاج (أي أنه Handle).

مثال : اعتمادا على القواعد ادناه اعرب الجملة (id+id*id) باستخدام طريقة shift reduce

E → E + E
E → E * E
E → (E)
E → id

Step	Stack	Input	Action	Production Used
1		(id+id*id) $	Shift '('	
2	(	id+id*id) $	Shift 'id'	
3	(id	+id*id) $	Reduce id → E	E → id
4	(E	+id*id) $	Shift '+'	
5	(E+	id*id) $	Shift 'id	
6	(E+id	*id) $	Reduce id → E	E → id
7	(E+E	*id) $	Shift '*'	
8	(E+E*	) $	Shift 'id	
9	(E+E*id	) $	Reduce id → E	E → id
10	(E+E*E	) $	Reduce E * E → E	E → E * E
11	(E+E	) $	Reduce E + E → E	E → E + E
12	(E	) $	Shift ')	
13	(E)		Reduce (E) → E	E → (E)
14	E		Accept	

في التحليل من الأسفل(Bottom-Up Parsing) ، يقوم المحلل بفحص أعلى عنصر في المكدس (Top of Stack)للتحقق مما إذا كان يشبه أي Handle على جهة اليمين لقواعد الإنتاج.
إذا كان العنصر الموجود في أعلى المكدس يتطابق مع كامل الـHandle ، يقوم المحلل بعملية Reduction:أي إزالة الـ Handle من المكدس (Pop) ثم دفع (Push) الرمز غير الطرفي(Non-terminal, NT)المقابل من جهة اليسار للقاعدة.
إذا كان العنصر الموجود في أعلى المكدس يتطابق مع آخر رمز في جهة يمين الـ Handle فقط) أي نهاية الـ(Handle ، يبدأ المحلل في تحديد النهاية اليسرى للـHandle ، وذلك في حالة كون الـ Handle مكونًا من أكثر من رمز واحد، لضمان تطبيق التقليص بشكل صحيح.

12.3محلل أسبقية المشغل(Operator Precedence Parser)

محلل أسبقية المشغل هو نوع من المحللات من نوع Shift-Reduce Parser، يعتمد على تقنية التحليل من الأسفل إلى الأعلى(Bottom-Up Parsing) ، ويُستخدم بشكل خاص مع فئة محدودة من القواعد تُعرف باسم قواعد المعاملات.(Operator Grammar)
المميزات:

- يستخدم **تقنية التحليل من الأسفل إلى الأعلى.(Bottom-Up Parsing Technique)**
- يعمل على قواعد من نوع **Operator Grammar**، أي قواعد خاصة بالمعاملات.
- يعتمد على **آلية Shift-Reduce** لتقليص المقاطع (Handles) إلى رموز غير طرفية(Non-terminals).

- مناسب للتعامل مع التعابير الحسابية والعمليات التي لها أسبقية محددة.
- يسهّل تحديد **متى يتم تطبيق عملية التقليص (Reduce)** بناءً على أولوية المشغل.

1.12.3خصائص قواعد المعاملات(Operator Grammar)

- لا تحتوي على قواعد:(No ε-rules) ε

أي أنه لا يوجد إنتاج من الشكل: $A \rightarrow \varepsilon$

هذا لضمان وضوح الـ **Handle**أثناء التحليل من الأسفل إلى الأعلى.

- يجب أن تكون قواعد من نوع:Operator Grammar

$A \rightarrow BC\propto$

أي لا تحتوي على رموز غير طرفية متجاورة (No Adjacent Non-terminals) في جهة اليمين.
هذا يسمح بتحديد أسبقية المشغل وتطبيق Shift-Reduce Parsingبسهولة

مثال

$E \rightarrow E\ A\ E / (E) / -E / id:$
$A \rightarrow + | - | * | /$

ليست من نوع قواعد المعاملات، لأنه يوجد أكثر من NT متجاورين، لكن إذا قمنا بتعويض عن ال NT الذي هو (A) بما يقابله تصبح القواعد بالشكل التالي:

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

نلاحظ بعد التعويض لا يوجد أكثر من NT متجاورين، بهذه الحالة تصبح من نوع قواعد المعاملات.

2.12.3التحليل بالانتقال والتقليص — (Shift-Reduce Parsing) محلل أسبقية المشغل

يُعد التحليل بالانتقال والتقليص أحد أنواع التحليل من الأسفل إلى الأعلى(Bottom-Up Parsing) ، حيث يحاول هذا الأسلوب بناء شجرة اشتقاق (Parse Tree) لسلسلة الإدخال ابتداءً من الأوراق (Leaves)في الأسفل، ثم التدرج نحو الجذر (Root) في الأعلى.
تعتمد هذه الطريقة على عمليتين أساسيتين:

1. الانتقال :(Shift) نقل الرمز التالي من سلسلة الإدخال إلى المكدس.
2. التقليص :(Reduce) استبدال مقطع (Handle) في المكدس برمز غير طرفي(Non-terminal)وفقًا لقاعدة إنتاج.

وبتكرار عمليتي Shiftو Reduce، يتمكن المحلل من تحويل سلسلة الإدخال إلى الرمز الابتدائي (Start Symbol) للجملة، مما يدل على صحة البنية النحوية (Syntactic Correctness) وفق القواعد المحددة.

مثال : اعتمادا على القواعد ادناه أعرب الكلمة **abbcde**

$S \rightarrow aABe$
$A \rightarrow Abc | b$
$B \rightarrow d$

RIGHTMOST DERIVATION

$S \rightarrow aABe$
$\rightarrow aAde$
$\rightarrow aAbcde$
$\rightarrow abbcde$

REDUCTION (LEFTMOST)

abbcde $(A \rightarrow b)$
aAbcde $(A \rightarrow Abc)$
aAde $(B \rightarrow d)$
aABe $(S \rightarrow aABe)$
S

3.12.3الإجراءات في خوارزمية التحليل بالانتقال والتقليص:(Shift-Reduce Parsing)

- الانتقال :(Shift) يتم نقل (إزاحة) الرمز التالي من جملة الإدخال إلى أعلى المكدس.(Stack)
- التقليص :(Reduce) يقوم المحلل باستبدال المقبض (Handle) الموجود داخل المكدس بالرمز غير الطرفي (Non-terminal) المقابل له وفقًا لقاعدة الإنتاج المناسبة.
- القبول :(Accept) يعلن المحلل عن نجاح عملية التحليل عند اكتمالها بشكل صحيح.
- الخطأ :(Error) يكتشف المحلل وجود خطأ نحوي في جملة الإدخال، ويستدعي روتين معالجة الأخطاء.(Error Recovery Routine)

قيمة المكدس الابتدائية $
قيمة الإدخال الابتدائية: الجملة المراد تحليلها.
قيمة الإجراء الابتدائية: الانتقال(Shift)

معنى المقبض:(Handle)
المقبض هو مقطع (Substring) في جملة الإدخال يحقق الشرطين الآتيين:

- يطابق الجهة اليمنى لإحدى قواعد الإنتاج في القواعد.(Grammar)
- إن استبداله بالرمز غير الطرفي الموجود في الجهة اليسرى من نفس القاعدة يُعَدّ خطوة ضمن خطوات الاشتقاق الأيمن العكسي (Reverse of a Rightmost Derivation).

- الشرط الأساسي للإعراب بطريقه(Bottom-Up) هو خلو القواعد من (ε) Empty word وان تكون من نوع (Operator grammar) أي عدم وجود عناصر متجاورة من نوع Non-Terminal.
- لا تهتم هذه الطريقة بوجود أو عدم وجود رجوع خلفي في القواعد المطلوب التعامل معها.
- تحتاج هذه الخوارزمية إلى وجود Stack والعمليات الخاص بها والتي تمثلPush & Pop

4.12.3خطوات الخوارزمية

- تكوين جدول بثلاثّ أعمدة-:

1.العمود الأول يمثل. Stack

2.العمود الثاني يمثل عناصر الجملة المطلوب أعرابها بالكامل(. Input)
3.العمود الثالث والأخير يمثل Action والذي يمثل عمليتين أساسيتن هما . Shift & Reduce

- ❖ القيم الابتدائية للعمود الأول(Stack) تحتوي فقط عل. $
- ❖ القيم الابتدائية للعمود الثاني (Input) الجملة المطلوب إعرابها.
- ❖ القيم الابتدائية للعمود الثالث والأخير تكون Shift وتمثل عملية Push للعنصر الموجود في أقصى يسار العمود الثاني ودفع العنصر في. Stack
- ❖ لابد من تطبيق Right Most Derivation عل القواعد المعطاة.
- ❖ بعد الخطوة السابقة مباشرة وبالاعتماد عليها يتم تحديد ما يسمى (Handle) والتي سوف يعتمد عليها قيم العمود الثالث (Action)
- ❖ اشتقاق القواعد باستخدام (. Tree)
- ❖ أول مرحلة تمثل حالة إضافة العنصر الموجود في أقصى يسار الجملة المطلوب إعرابها وإضافته الى (.Top of Stack)
- ❖ ملا حّظة إذا كوان العنصر الذي تم إضافته الى) (Top of Stack)ٍفي الخطوة السابقة هل هو (Handle)ام لا إذا كان(Handle) فيتم إرجاع العنصر إل أصله وإذا لم يكن (Handle) فيتم اضافته الى(.Top of Stack)
- ❖ نستمر بالخطوات السابقة الى ان تكون قيمة الحقل الأول ($Stack=Start Symbol .)

مثال : اعتمادا على القواعد ادناه قم باشتقاق الكلمة التالية $id×id+id

$S \rightarrow S×S / S+S / id$

1. أوجد (أو استخرج) اشتقاقًا أيمنًا لهذه القواعد (النحو)".

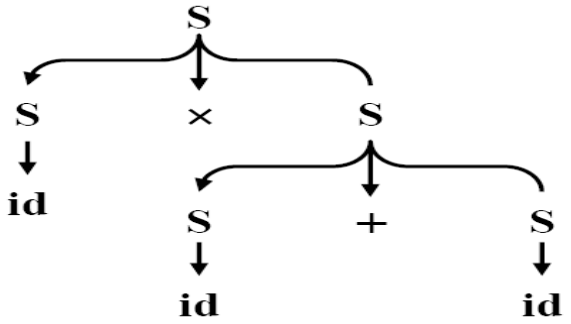
$S \rightarrow S×S \rightarrow S×S+S \rightarrow S×S+id \rightarrow S×id+id \rightarrow id×id+id$

Input = id×id+id$

2. "حدّد المقابض (Handles) بالاعتماد على الاشتقاق أعلاه".

$S \rightarrow \underline{S×S} \rightarrow S× \underline{S+S} \rightarrow S×S+ \underline{id} \rightarrow S× \underline{id} +id \rightarrow \underline{id} ×id+id$

3. بناء شجرة التحليل



4. بناء جدول الاعراب

Stack	Input	Action
$	id×id+id$	Shift
$ id	×id+id$	Reduce S →id
$ S	×id+id$	Shift
$ S×	id+id$	Shift
$ S×id	+id$	Reduce S →id
$ S×S	+id$	Shift
$ S×S+	id$	Shift
$ S×S+id	$	Reduce S →id
$ S×S+S	$	Reduce S →S+S
$ S×S	$	Reduce S →S×S
$ S	$	Accept

مثال: اعتمادا على القواعد ادناه قم باشتقاق الكلمة التالية $ (id×(id-id) /id) –

E → T / E+T / E-T / -T

T → F / T×F/ T/F

F → (E) / id

Input = -(id×(id-id) / id)$

_____ :-

E → -T

→ -F

→ -(E)

→ -(T)

→-(T/F)

→-(T/ id)

→-(T×F / id)

→-(T× (E) /id)

→ -(T×(E-T) /id)

→ -(T×(E - F) /id)

→ -(T×(E - id) /id)

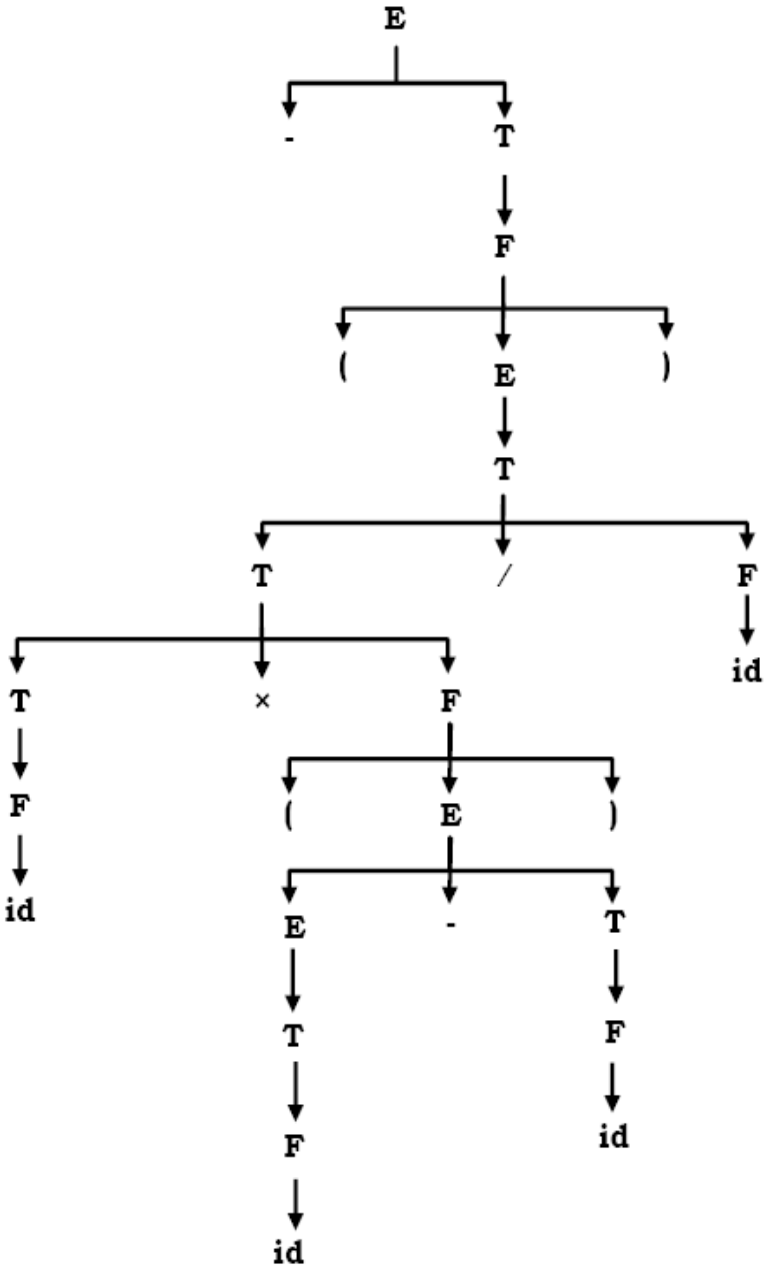
→ -(T×(T - id) /id)

→ -(T×(F - id) /id)

→ -(T×(id - id) /id)

→ -(F × (id - id) /id)

→ -(id × (id - id) /id)



Stack	Input	Action
$	-(id×(id-id)/id)$	Shift
$-	(id×(id-id)/id)$	Shift
$-(	id×(id-id)/id)$	Shift
$-(id	×(id-id)/id)$	Reduce F →id
$-(F	×(id-id)/id)$	Reduce T →F
$ -(T	×(id-id)/id)$	Shift
$ -(T×	(id-id)/id)$	Shift
$ -(T×(	id-id)/id)$	Shift
$ -(T×(id	-id)/id)$	Reduce F →id
$ -(T×(F	-id)/id)$	Reduce T →F
$ -(T×(T	-id)/id)$	Reduce E →T
$ -(T×(E	-id)/id)$	Shift
$ -(T×(E-	id)/id)$	Shift
$ -(T×(E-id	)/id)$	Reduce F →id
$ -(T×(E-F	)/id)$	Reduce T →F
$ -(T×(E-T	)/id)$	Reduce E →E-T
$-(T×(E	)/id)$	Shift
$-(T×(E)	/id)$	Reduce F → (E)
$-(T×F	/id)$	Reduce T →T×F
$-(T	/id)$	Shift
$-(T/	id)$	Shift

$-(T/id	)$	Reduce F →id
$-(T/F	)$	Reduce T →T/F
$-(T	)$	Reduce E → T
$-(E	)$	Shift
$-(E)	$	Reduce F → (E)
$-F	$	Reduce T →F
$-T	$	Reduce E → -T
$E	$	Accept

LR

13.3 Parsing (التحليل LR)

LR Parserهو محلل نحوي (Syntax Parser) يقرأ جملة الإدخال من اليسار إلى اليمين-(Left-to-Right)، ويقوم ببناء اشتقاق أيمن عكسي.(Rightmost Derivation in Reverse)
يحتاج هذا المعرب في عمله إلى أوتوماتا منتهية حتمية(DFA) ، ولغرض بناء هذا الـ DFA نحتاج إلى نوع خاص من القواعد يُسمّى القواعد الموسّعة.**(**Augmented Grammar)

1.13.3 Augmented Grammar

هي شكل مُحسّن من القواعد الأصلية (Grammar) يُستخدم خصيصًا عند بناء محللLR ، حيث نضيف قاعدة جديدة تحتوي على رمز بداية جديد 'Sلإنتاج الرمز الابتدائي الأصلي .S

إذا كانت لدينا القواعد G={V,T,P,S}

فإن القواعد الموسّعة تكون:

G'={V∪{S'}, T, P∪{S'→S}, S'}

- الغرض من إضافة القاعدة الجديدة 'S→'Sهو تسهيل بناء DFA اللازم للتحليل.
- الرمز الجديد 'Sيُسمّى **Start Symbol** الموسّع.
- هذه الخطوة ضرورية عند تصميم **LR(0) items** وبناء جداول التحليل)

مثال :

$E \rightarrow E + E$
$E \rightarrow id$
Sol:
$E' \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow id$

وهناك ثلاث تقنيات

التقنية	LALR	CLR(1) = LR(1)	SLR(1)
الاختصار	Look Ahead LR(1)	Canonical LR(1)	Simple LR(1)
حجم القواعد المستخدمة	يعمل على حجم متوسط من القواعد	يعمل على مجموعة كاملة من القواعد	يعمل على أصناف صغيرة من القواعد
عدد الحالات (States)	نفس عدد الحالات في المحلل((LR(1)	يولد عدد كبير من الحالات (States)، لذا جدول الإعراب يكون كبير ومعقد	عدد الحالات يكون قليل، لذا جدول الإعراب يكون صغير
البناء	متوسط التعقيد	معقد وبسيط	بسيط وسريع
الكفاءة والأداء	متوسط الفعالية، والكلفة له أقل من LR(1) وأعلى من SLR(1)	أعلى كفاءة وأكثر فعالية	أقل كفاءة وأقل فعالية
نوع الـ Item المستخدم لبناء DFA	يستخدم LR(1) Item	يستخدم LR(1) Item	يستخدم LR(0) Item

2.13.3 تحليل LR (LR Parsing) والأوتوماتا المحددة(DFA)

يعتمد **تحليل LR** على استخدام **الأوتوماتا المحددة – Deterministic Finite Automata (DFA)**. تتكوَّن هذه الأوتوماتا من عدد محدود من الحالات، ولإيجاد هذه الحالات نتبع الخطوات التالية:

1. استخدام القواعد الموسَّعة(Augmented Grammar)

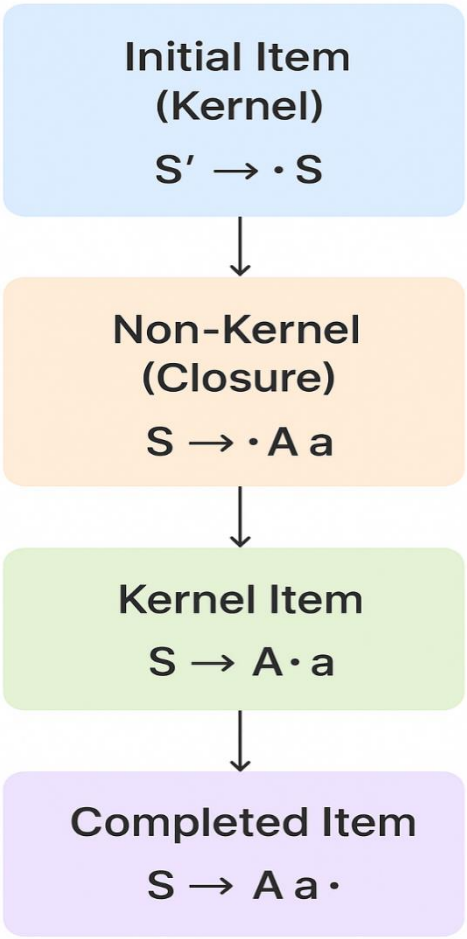
لإعداد DFA ، يتم أولًا تحويل القواعد الأصلية إلى **قواعد موسَّعة**، وذلك بإضافة قاعدة ابتدائية جديدة تسمى **'S**، بحيث يكون الشكل:

$S' \rightarrow S$

2. عناصر (LR(0) Items)

لتكوين حالاتDFA ، نستخدم مفهوم **LR(0) Items**، والذي يتمثل في وضع **نقطة (·)** داخل قواعد الإنتاج لتحديد موقع التحليل الحالي في الطرف الأيمن للقاعدة .(RHS) هذه النقطة تساعد في تمييز حالة التحليل لكل عنصر.

3.أنواع عناصرLR(0)



Initial Item
(Kernel)
Non-Kernel
(Closure)
Kernel Item
Completed
Item

- **Kernel Items العناصر الأساسية:**
هي كل العناصر التي لا تكون النقطة فيها في بداية الطرف الأيمن، بالإضافة إلى **initial item** .(S' → ·S)
أمثلة:

S→A·a,

- **Non-Kernel Items العناصر غير الأساسية:**
هي العناصر التي تكون النقطة فيها عند بداية الطرف الأيمن مباشرةً، **باستثناء** العنصر الابتدائي.S' → ·S
- **Completed Items العناصر المكتملة:**
هي العناصر التي تقع النقطة فيها عند نهاية الطرف الأيمن للقاعدة، أي عندما يكتمل مطابقة.RHS
مثال:

X→ab·

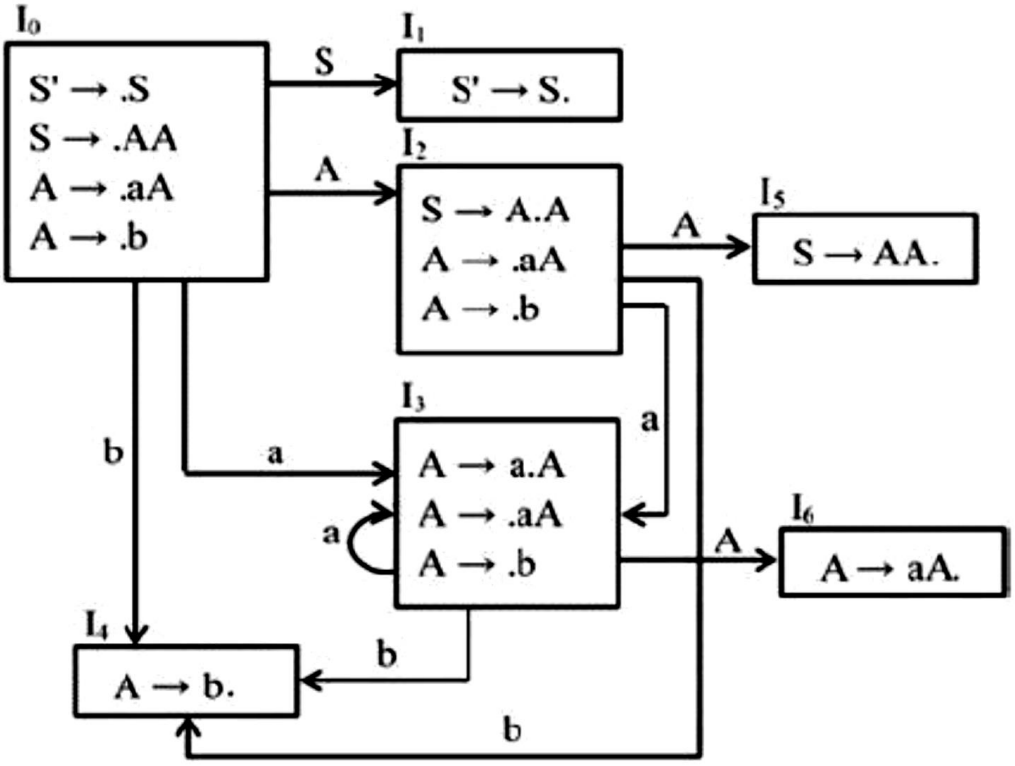
مثال : اعتمادا على القواعد التالية اوجد عناصر LR(0)

S→ AA
A → aA
A→ b

S' → S
S → AA
A → aA
A → b

ملاحظة: عندما تأتي Non-terminal بعد النقطة نستمر بإضافة القواعد الخاصة لها (نستمر بالاشتقاق بالاعتماد على القواعد الأصلية الثانية بعد الترقيم) واذا كان Terminal نتوقف، وتكون

إضافة القواعد من ال grammar وليس من التكرار السابق، لأن التكرار لا يعتمد على التكرار الذي قبله أبدًا .



وبعدها يتم اعراب الجملة وهل تكون مقبولة ام لا (حالات الاعراب للاطلاع)

3.13.3 SLR(1) Parser (Simple LR Parser)

يعتبر **SLR(1)** نوعًا من **Bottom-Up Parsers** يعتمد على **LR(0) Items** مع إضافة معلومات الـ **Follow** لتحديد أماكن التقليص (Reduce) بدقة أكبر.

خطوات بناء محلل SLR(1)

حساب مجموعات First و Follow

- **First(X)** مجموعة الرموز الطرفية التي يمكن أن تبدأ بها أي سلسلة مشتقة من X.
- **Follow(A)** مجموعة الرموز الطرفية التي يمكن أن تأتي مباشرة بعد الرمز غير الطرفي A في أي جملة صالحة للقواعد.

إيجاد مجموعة Items (Closure)

- **Item** قاعدة إنتاج مع نقطة (•) تحدد مدى تقدم التحليل.
- **Closure(I)** العملية التي تضيف جميع القواعد الممكنة إلى الـ Item إذا كان هناك نقطة أمام رمز غير طرفي.

ملاحظة Items في SLR(1) هي نفسها المستخدمة في.LR(0)
وبعدها يتم اعراب الجملة وهل تكون مقبولة ام لا (حالات الاعراب للاطلاع)

مثال : اعتمادا على القواعد

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

Augmented Grammar:

0) $E' \rightarrow E$
1) $E \rightarrow E + T$
2) $E \rightarrow T$
3) $T \rightarrow T * F$
4) $T \rightarrow F$
5) $F \rightarrow (E)$
6) $F \rightarrow id$

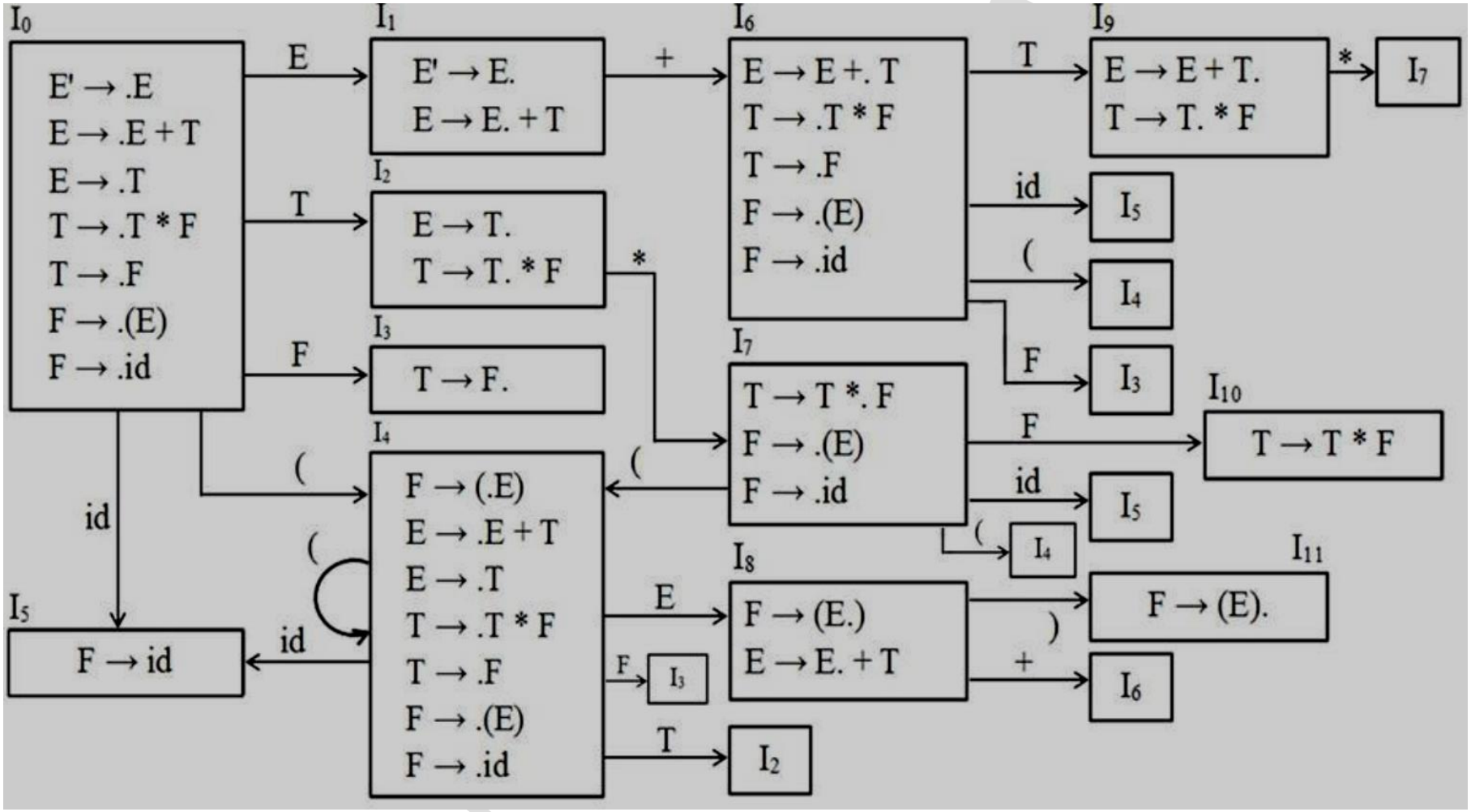
First and Follow Sets:

Non-terminal	First	Follow
F	(, id	+ ,) , $
T	(, id	+ , * ,) , $
F	(, id	+ , * ,) , $

LR(0) Items (Closure):

State	Items
I0	$E' \rightarrow \bullet E$ $E \rightarrow \bullet E + T$ $E \rightarrow \bullet T$ $T \rightarrow \bullet T * F$ $T \rightarrow \bullet F$ $F \rightarrow \bullet (E)$ $F \rightarrow \bullet id$ |
I1	$E' \rightarrow E \bullet$ $E \rightarrow E \bullet + T$ |

| I2 | E → T •
T → T • * F |
| I3 | T → F • |
| I4 | F → (• E)
E → • E + T
E → • T
T → • T * F
T → • F
F → • (E)
F → • id |
| I5 | F → id •


I0
E' → .E
E → .E + T
E → .T
T → .T * F
T → .F
F → .(E)
F → .id
E
T
F
id
(
I1
E' → E.
E → E. + T
I2
E → T.
T → T. * F
I3
T → F.
I4
F → (.E)
E → .E + T
E → .T
T → .T * F
T → .F
F → .(E)
F → .id
I5
F → id
id
(
id
I6
E → E + . T
T → .T * F
T → .F
F → .(E)
F → .id
+
T
id
(
F
I7
T → T * . F
F → .(E)
F → .id
*
(
F
id
(
I8
F → (E.)
E → E. + T
E
F
T
I9
E → E + T.
T → T. * F
*
I7
I10
T → T * F
F
I11
F → (E).
)
+
I6
I2
I3
I4
I5