Compilers Techniques
Lecture 9
المحاضرة التاسعة

Semantic Analysis التحليل المعنوي

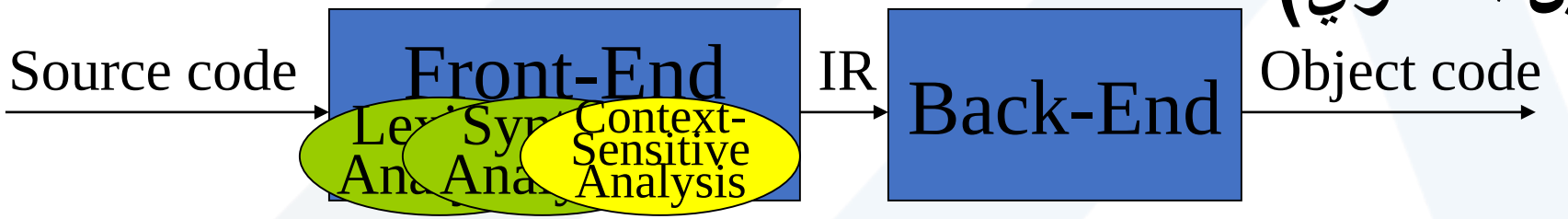
Reference: Aho Pages 304-352

السنة الرابعة –المستوى السابع- الهندسة المعلوماتية

Semantic Analysis التحليل المعنوي

من المحاضرات السابقة:

- بعد التحليل المعجمي وتحليل القواعد وجدنا أن البرنامج المصدري هو جملة تخضع لقواعد اللغة المصدرية.
- خرج مرحلة التحليل النحوي هي شجرة الإعراب: وهي تمثيل رسومي لكيفية اشتقاق سلسة في اللغة انطلاقاً من رمز الدخل start symbol
- *لكن هل تحققنا من كل الأخطاء التي يمكن أن تحدث؟*
- محاضرة اليوم: **قواعد الأنماط والصفة attribute grammars**
- **بشكل عام (التحليل المعنوي)**



What is wrong with the following?

التحليل المعنوي
Semantic Analysis

```
foo(a,b,c,d)
 int a,b,c,d;
{ … }

lala()
{
 int f[3],g[0],h,i,j,k;
 char *p;
 foo(h,i,"ab",j,k);
 k=f*i+j;
 h=g[7];
 printf("%s,%s",p,q);
 p=10;
}
```

في الكود المجاور يمكننا عد 6 أخطاء على الأقل كل منها أخطاء أبعد من موضوع بناء الجملة (القواعد) هذه الأخطاء غير متعلقة بالقواعد حرة السياق

لتوليد الشيفرة علينا أولاً أن نتأكد أنها ذات معنى وتؤدي لنتيجة منطقية

What is wrong with the following?

التحليل المعنوي
Semantic Analysis

لتوليد الكود النهائي لا بد للمترجم أن يفهم الأسئلة التالية:

- هل X هي قيمة وحيدة Scalar؟ هل هي مصفوفة أم تابع؟ هل تم التصريح عنها مسبقاً؟
- هل هناك أسماء لمتغيرات لم يتم التصريح عنها من قبل وتستخدم حالياً؟ هل هناك متحولات صرح عنها لكن لم يتم استخدامها؟
- ما هو تصريح المتحول X الذي يستخدم المرجع وليس القيمة؟
- هل التعبير **`x-2*y`** صحيح من جهة أن `x` و`y` من ذات النوع (متناسقان **type checking**)، المترجم بحاجة لأن يخصص نوع بيانات لكل تعبير يقوم بحسابه (مثلاً `integer`).
- في التصريح **`a[i,j,k]`** هل المقصود أن `a` هي مصفوفة ثلاثية البعد؟
- من هو المسؤول عن تخصيص مساحة للمتحول z؟ لأي وقت نريد أن يتم الاحتفاظ بقيمته؟
- كيف نمثل الرقم 15 في التعبير **`f=15`** ؟
- كم عدد البارامترات التي يأخذها التابع foo() كوسطاء له؟
- هل يمكن أن يشير كلاً من p وq إلى ذات الموقع في الذاكرة؟

كل تلك الأسئلة هي خارج ما تقدمه قواعد النحو حر السياق !

نظم الأنماط
Type Systems

- نوع القيمة Type هي **مجموعة من الصفات** المعرفة لتلك القيمة.
- يطلق على مجموعة الأنواع في لغة البرمجة بنظم الأنماط Type System.
- يشير مصطلح التحقق من النوع Type Checking إلى إسناد\توقع أنواع التعابير والتحقق من أن عمليات الإسناد تلك صحيحة واستخدمت بشكل مشروع Legal وخاضع للسياق.
- **مكونات نظم الأنماط:**
- **أنماط أساسية** Built-in وهي الأنواع الأساسية المبنية مع اللغة (الأرقام، المحارف، النوع المنطقي boolean).
- **القواعد** اللازمة لبناء أنواع جديدة، تحديد مدى تكافؤ نوعي بيانات، استنتاج نوع بيانات تعابير الملف المصدري.

+	**int**	**real**	**double**	**complex**
int	int	real	double	complex
real	real	real	double	complex
double	double	double	double	illegal
complex	complex	complex	illegal	complex

حساسية السياق

- **التحقق من النوع Type Checking:**
- تعتمد عملية التحقق على تصميم اللغة وكيفية التطبيق.
- في زمن الترجمة Compile Time: يتم التحقق بشكل ستاتيكي (ساكن).

- int x=5; string s=“”; x=s;

- في زمن التشغيل Run Time: تتم عملية التحقق بشكل ديناميكي.
- مثل توقع أن يدخل المستخدم من لوحة المفاتيح رقم صحيح وقام بإدخال محرف! فيتم التحقق من تطابق الأنواع هنا في زمن التشغيل لأن البرنامج يكون قيد التنفيذ.
- لو كانت لغة البرمجة مزودة بفكرة سياسة declare before use «التصريح قبل الاستخدام» عندها تكون عملية استنتاج الأنواع غير صعبة. التحدي يكمن عندما لا يكون هناك أي تصريحات Declarations . لاحظ الجدول التالي:

+	**int**	**real**	**double**	**complex**
int	int	real	double	complex
real	real	real	double	complex
double	double	double	double	illegal
complex	complex	complex	illegal	complex

حساسية السياق

إن الأسئلة في الشريحة 4 هي جزء من تحليل حساسية السياق (التحليل المعنوي)

- تعتمد الإجابة على تلك الأسئلة على ***القيم*** وليس على القواعد أو النحو.
- يمكن أن تتطلب الإجابة على بعض تلك الأسئلة بعض العمليات الحسابية.

كيف نجيب على تلك الأسئلة؟

- **استخدام طرق الصيغ الرياضية Use formal methods:**
  - *باستخدام النحو حساس السياق: هناك العديد من الأسئلة التي من الصعب الإجابة عليها من خلال النحو حساس السياق وسيكون هناك العديد من القواعد اللازمة لضبط كل هذه التساؤلات (مثلاً قصة التصريح قبل الاستخدام).*
  - *باستخدام* قواعد الصفات **attribute grammars.**
- ***باستخدام التقنيات المخصصة ad-hoc:***
  - *جداول الرموز* Symbol tables والترجمة الموجهة قواعدياً syntax-directed translationباستخدام قواعد الصفات attribute grammars.

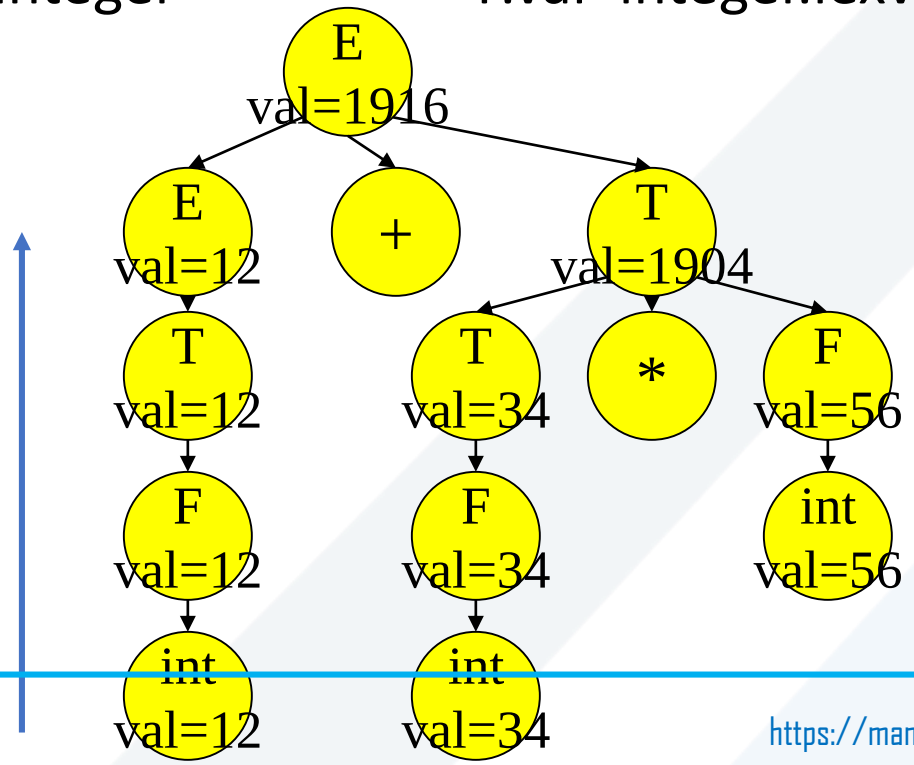
قواعد الصفات
Attribute Grammars

- **الفكرة:**
  - تخصيص كل رمز من رموز النحو بمجموعة من القيم أو الصفات.
  - تخصيص قاعدة معنوية (دلالية) لكل قاعدة نحو تحديد قيمة كل صفة في ضوء وجود بقية الصفات الأخرى (متل العملي)
- **Attribute Grammars (Knuth صيغة ناث):**
  - تزويد النحو حر السياق بمجموعة من القواعد الدلالية (Semantic Actions) كما تعلمنا في العملي.
    - Exp Plus Exp {$$=$1+$3;}
  - لكل رمز في النحو مجموعة من القيم (أو الصفات) تحدد عمله.
  - تحدد القاعدة الدلالية كيفية حساب قيمة لكل صفة.

مثال
(قاعدة دلالية لحساب قيمة تعبير ما)

- **مثال لحساب قاعدة دلالية (معنوية) ما:**

$G \rightarrow E$	print(E.val)
$E \rightarrow E_1+T$	$E.val=E_1.val+T.val$
$\mid T$	E.val=T.val
$T \rightarrow T_1*F$	$T.val=T_1.val \times F.val$
$\mid F$	T.val=F.val
$F \rightarrow integer$	F.val=integer.lexval



ملاحظة:
نعنون التعابير المتطابقة بأرقام مختلفة

ترتيب عملية التقييم:
نبدأ من الأوراق ونصعد باتجاه رمز بداية الإعراب (كما تعلمنا في العملي)

التحليل المعنوي
Semantic Analysis

الاعتماديات بين الصفات:

- صفات مركبة: تشتق قيمها من الأبناء ومن الثوابت.
  - نحصل على نحو مركب S-attribute grammars وفيه يتم التقييم من العمق باتجاه السطح (كما رأينا في المثال السابق).
- الصفات الموروثة: تشتق قيمها من الآباء، الثوابت والأخوة.
  - سياق التعبير المباشر Directly express context.

- **ملاحظة:** القواعد الدلالية الخاصة بعقدة من شجرة الإعراب تحدد الاعتماديات مع القواعد الدلالية الأخرى، لكن ماذا عن ترتيب التقييم في حالات أكثر تعقيداً؟ هذا ما سنراه لاحقاً.

Example (variable declarations) cf. Aho1, p.283, Ex: 5.3

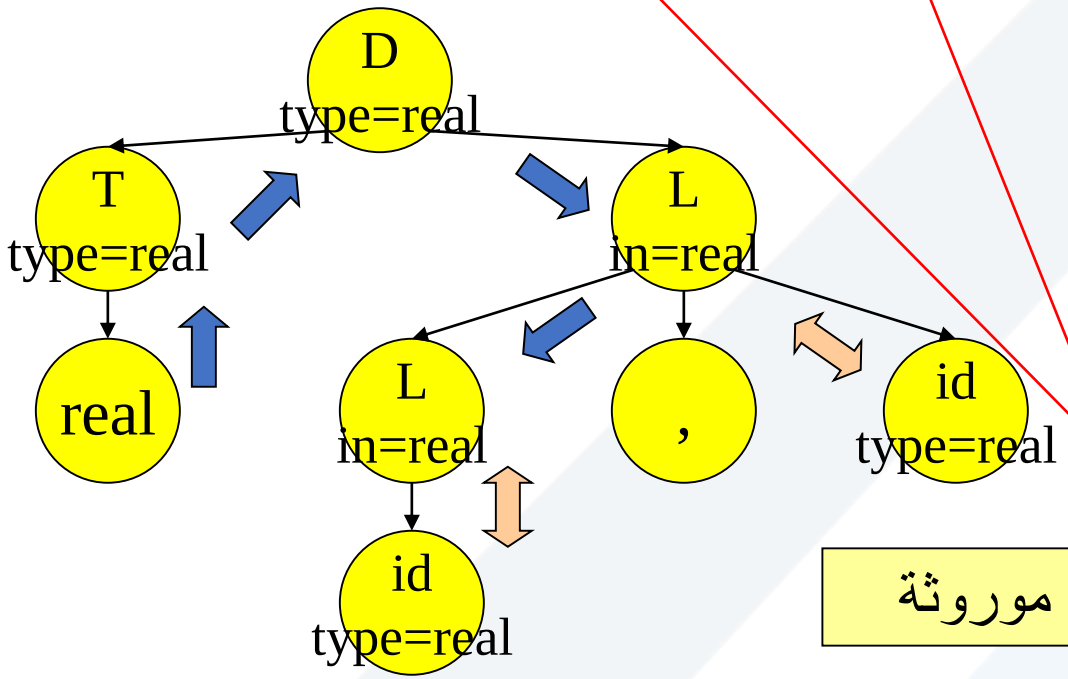
التحليل المعنوي Semantic Analysis

D→T L	L.in=T.type
T→int	T.type=integer
| real	T.type=real
L→L_1, id	L_1.in=L.in; id.type=L.in
| id	id.type=L.in

الخاصية in هنا تعتمد على العقدة الأخ لذا هي موروثة

الخاصية type مركبة

الخاصية type الخاصة بـ id هنا هي موروثة



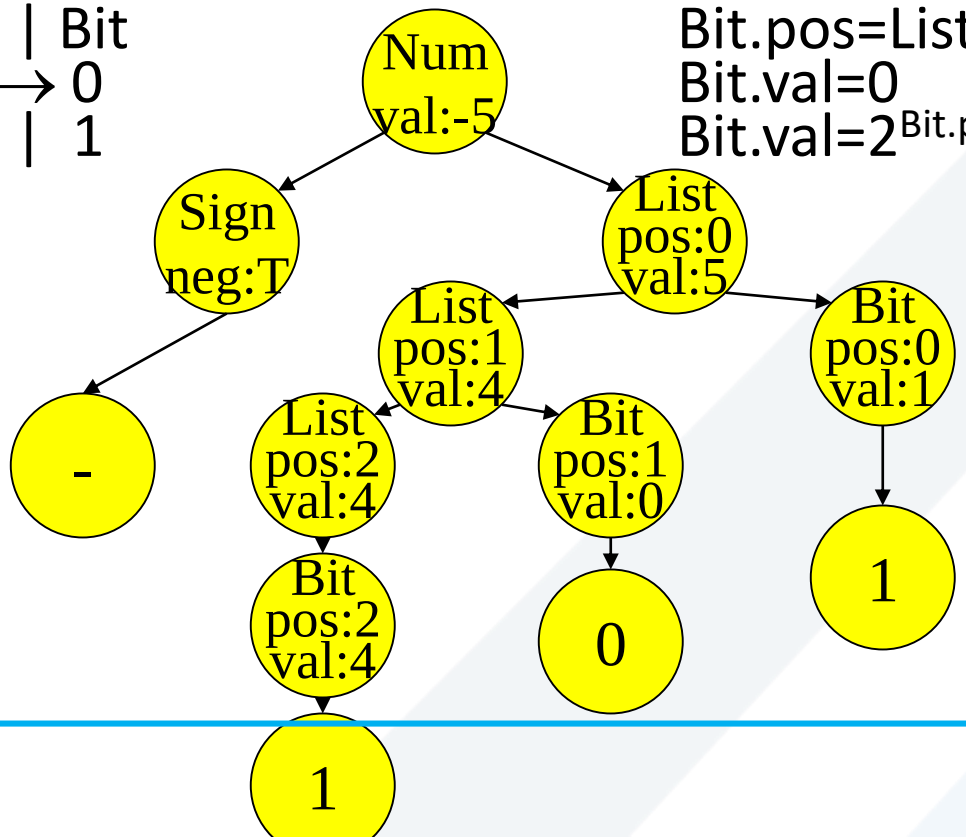
عملياً، لسنا بحاجة لكل هذا العمل عند التعامل مع التصريحات فقط يمكننا بناء جدول رموز وكلما صادفنا رمز في التصريح نضيفه إلى جدول الرموز (وهذا ما سنراه في محاضرة تحليل المعاني في القسم العملي).

addtype(id.type, L.in)

نوع الخاصية "type" هي موروثة نوع الخاصية "in" هي موروثة

مثال آخر
(الأرقام الثنائية ذات الإشارة)

Number→Sign List	List.pos=0; If Sign.neg then Number.val=-List.val else Number.val=List.val
Sign→ + | − List→List$_1$ Bit	Sign.neg=false Sign.neg=true List$_1$.pos=List.pos+1; Bit.pos=List.pos; List.val= List$_1$.val+Bit.val
| Bit Bit → 0 | 1	Bit.pos=List.pos; List.val=Bit.val Bit.val=0 Bit.val=$2^{Bit.pos}$



- الأرقام والإشارات لها خاصية واحدة فقط.
- List و Bit لكل منها خاصيتان.
- Val و neg هي خاصية مركبة
- Pos هي خاصية موروثة
- ماهو ترتيب التقييم في هذه الحالة المختلطة؟
- حسب طريقة ناث Knuth

اقترح ناث نموذج تدفق بيانات data- flow model لعملية التقييم مع الصفات المستقلة أولاً (غير معتمدة على غيرها) وهذا يتطلب منا رسم مخطط الاعتمادية

مخطط اعتمادية الصفات
Attribute dependence graph

في حال اعتمدت خاصية ما في عقدة ما على خاصية عقدة أخرى عندها نقيم الأولى أولاً ثم الثانية.

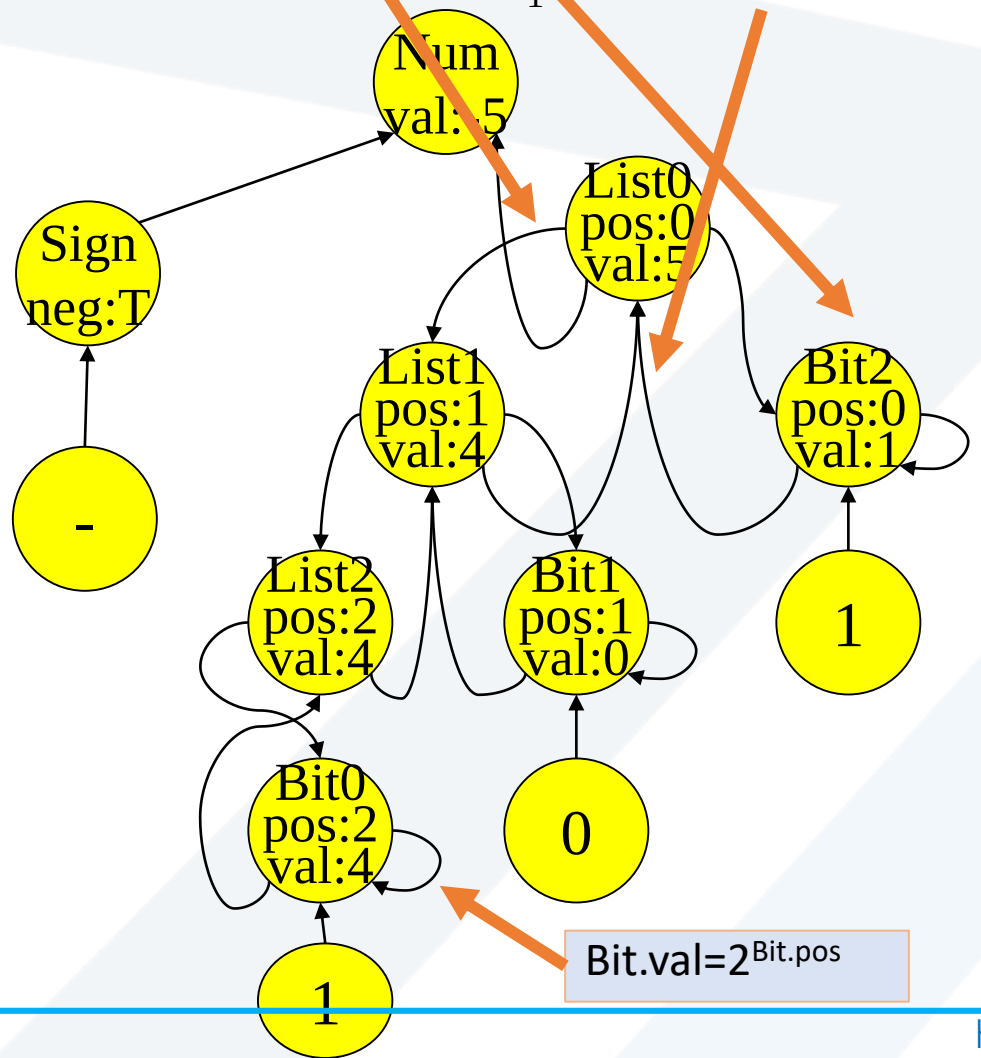
Attribute Dependence Graph مخطط اعتمادية الصفات:

- العقد تمثل الصفات أما الحواف تمثل تدفق القيم.
- مخطط الاعتمادية موازي لشجرة الإعراب (يمكن رسمهما جنباً إلى جنب، حجم المخطط معتمد على حجم شجرة الإعراب).
- **ترتيب عملية التقييم Evaluation order:**
  - الطريقة المعتمدة على شجرة الإعراب: استخدام ترتيب طبولوجي (ترتيب العقد حيث الحواف تنطلق فقط من العقد السابقة إلى اللاحقة كما يلي: نرتب المخطط، نوجد القيم المستقلة، نمشي عبر حواف المخطط). المخططات الدائرية تفشل في هذه الطريقة.
  - الطريقة المعتمدة على القواعد الدلالية Rule-based methods هنا الترتيب محدد مسبقاً.
  - الطريقة غير المعتمدة على القواعد الدلالية (المنسية) Oblivious methods : حيث اختيار طريقة غير معتمدة على القواعد الدلالية لعملية التقييم ضمن الشجرة
- أهم المشاكل التي تصادفنا في هذه الطرق هي الحلقات والاعتماديات المعقدة

Binary Numbers example: dependency graph

List→List$_1$ Bit

List$_1$.pos=List.pos+1;
Bit.pos=List.pos;
List.val=List$_1$.val+Bit.val



الترتيب الطبولوجي

1. Sign.neg
2. List0.pos
3. List1.pos
4. List2.pos
5. Bit0.pos
6. Bit1.pos
7. Bit2.pos
8. Bit0.val
9. List2.val
10. Bit1.val
11. List1.val
12. Bit2.val
13. List0.val
14. Num.val

استخدام نحو الصفات عملياً

- نحو القواعد العامة غير مستخدم عملياً بشكل كبير
  - معالجة السياق المحلي المعقد سهلة.
  - تحتاج القضايا الحساسة للسياق غير المحلية العديد من قواعد الدعم. عادة، يمكن التفكير ببناء جداول عامة. لكن هذا يأخذ وقت.
- ولاتزال الأبحاث لحل هذه المسألة مستمرة وقد ظهرت نتائجها في بعض تطبيقات محررات النصوص البنيوية، كتمثيل الوثائق البنيوية وغيرها (قواعد الصفات وXML)
- وعملياً يتم اللجوء إلى استخدام الأفكار البسيطة مثل:

الترجمة المخصصة المعتمدة على القواعد ***ad hoc* syntax directed**:

- **قواعد الصفة من النوع S-attribute grammars**:
  - تستخدم الصفات المركبة فقط (وليس الموروثة).
  - تتدفق القيم باتجاه واحد فقط (من الأوراق إلى الجذر).
  - عملية التقييم فعالة وهي تتم من الأسفل للأعلى (من الأوراق للجذر) ومناسبة لمعرب bottom-up المشروح في محاضرات سابقة.
- **قواعد الصفة من النوع L-attribute grammars**:
  - يستخدم الصفات المركبة والموروثة.
  - هناك قواعد محددة تحدد استخدام الصفات الموروثة.

عودة لمثال المحاضرة السابقة

1. *Goal* → *Expr*
2. *Expr* → *Expr* + *Term*
3. | *Expr* – *Term*
4. | *Term*
5. *Term* → *Term* * *Factor*
6. | *Term* / *Factor*
7. | *Factor*
8. *Factor* → *number*
9. | *id*

باستخدام الإعراب (من الأسفل للأعلى) LR (من المحاضرة السابقة) ونتيجةً لحساب التخفيضات الخاصة أيضاً. بالسلسلة 2*8-19 يتم استخدام القواعد التالية بالترتيب:

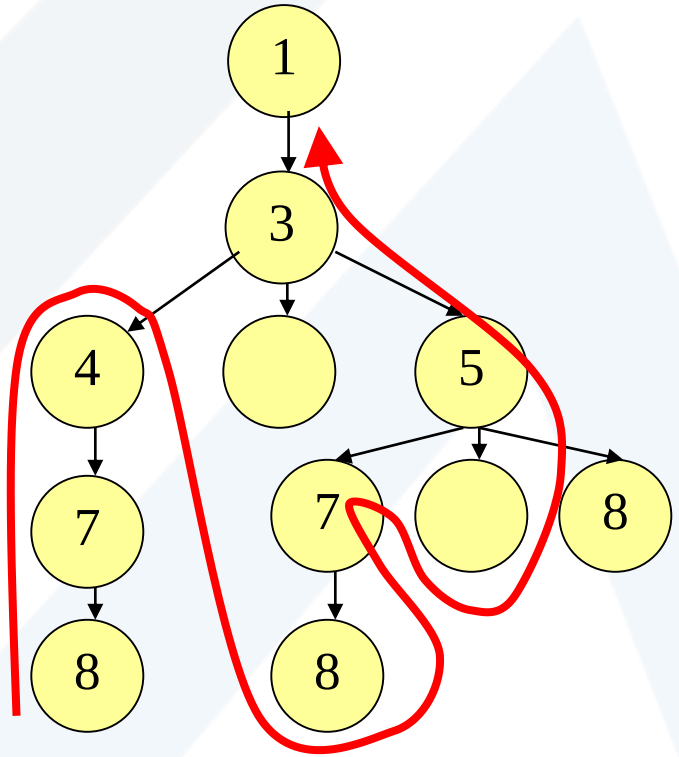
8, 7, 4, 8, 7, 8, 5, 3, 1

يتم حساب صفات الجزء اليساري بسهولة بهذا الترتيب

Parse tree:

يشير السهم الأحمر إلى تدفق البيانات,

وهو ذات الترتيب الذي تكتشف فيه القواعد

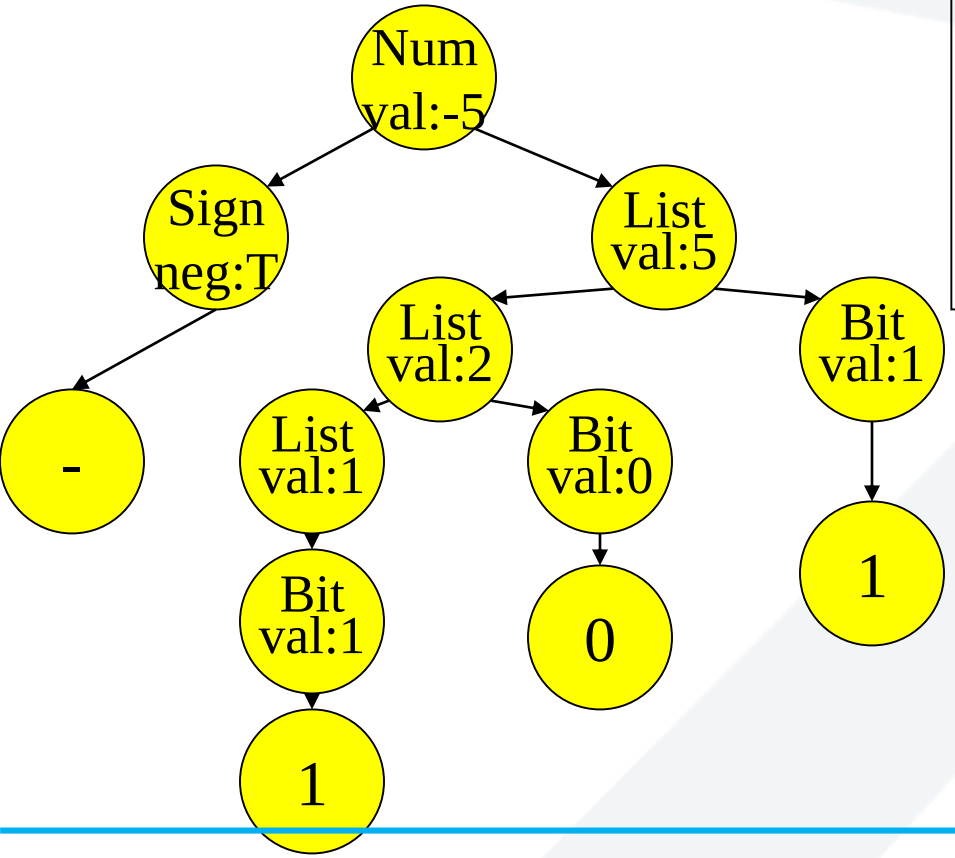


أفترض أننا خصصنا لكل رمز الخاصية Val لحساب قيمة التعبير

القواعد الدلالية حذفت لكنها واضحة

مثال الأرقام الثنائية ذات الإشارة مجدداً باستخدام S-Attribute هذه المرة

Number→Sign List	Number = Sign.neg*List.val
Sign→ +	Sign.neg=1
| –	Sign.neg=–1
List→List$_1$ Bit	List.val=2*List1.val+Bit.val
| Bit	List.val=Bit.val
Bit → 0	Bit.val=0
| 1	Bit.val=1



لاحظ حذفنا pos ولم يبقى سوى خاصية واحدة هي Val وهي من النوع المركب (يأخذ قيمه من الابن)