



Compilation

grammaires à attributs

Olivier Ridoux

Plan

- Grammaires à attributs
 - principes
 - mises en œuvre
 - YACC
 - ANTLR

Principes

Calcul dirigé par la syntaxe

- Enchaînement d'opérations dictées par la syntaxe (abstraite)
- Ne pas répondre $m \in \mathcal{L}(G)$
mais $m \in \mathcal{L}(G)$ et $SEM(m) = \dots$

Principes – arbres d'analyse

- Arbre engendré par une grammaire
 - arbre de dérivation
ou arbre de syntaxe abstraite
 - nœud \equiv **instance** de règle
 - si $n = \text{instance}(r)$
alors $\text{label}(n) = \text{tête}(r)$
et $n' \in \text{fils}(n)$ ssi $\text{label}(n') \in \text{corps}(r)$

Principes - attributs

- Attribut = domaine de calcul
entier, chaîne, ensemble, table, ...
- Association (non-)terminal \rightarrow attributs
 $(n)T \rightarrow \text{attr}$
- Association nœud \rightarrow **instance**(attr)
 $n = \text{instance}(r) \rightarrow \text{instance}(\text{attribut}(\text{tête}(r)))$

Principes – règles sémantiques

- Association

règle syntaxique {règle sémantique}



r_{syn}

$E \rightarrow E + E$

$\{r_{\text{sém}}\}$

$\{E.\text{val} = E.\text{val} + E.\text{val}\}$

– quel E ?

– identifier les occurrences !

$E \rightarrow E' + E'' \quad \{E.\text{val} = E'.\text{val} + E''.\text{val}\}$

$E_1 \rightarrow E_2 + E_3 \quad \{E_1.\text{val} = E_2.\text{val} + E_3.\text{val}\}$

Principes – règles sémantiques

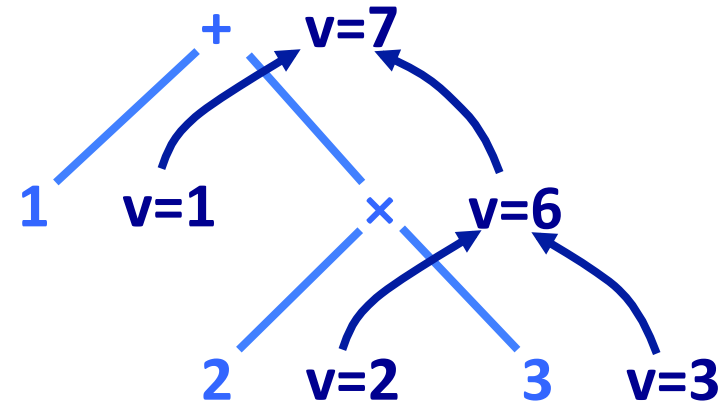
- $\text{Dép}(r_{\text{sém}}) : \mathcal{P}(\text{Attr} \rightarrow \text{Attr})$
 $\{E.\text{val} = E'.\text{val} + E''.\text{val}\} :$
 $\{E'.\text{val} \rightarrow E.\text{val}, E''.\text{val} \rightarrow E.\text{val}\}$
- Dépendance fonctionnelle
 - si $\text{Attr} \rightarrow \text{Attr}' \in \text{Dép}(r_{\text{sém}})$
alors **connaître Attr est nécessaire pour connaître Attr'**

Principes – sémantique de GA

- Soit une grammaire à attributs GA
GA = règles syntaxiques (R_{syn})
+ règles sémantiques ($R_{sém}$)
- Soit un mot $m \in \Sigma_T^*$
 - construire PT l'arbre d'analyse de m par R_{syn}
 - créer les instances des attributs de $R_{sém}$ dans PT
 - évaluer les instances de $R_{sém}$ pour PT

Exemple – une calculatrice

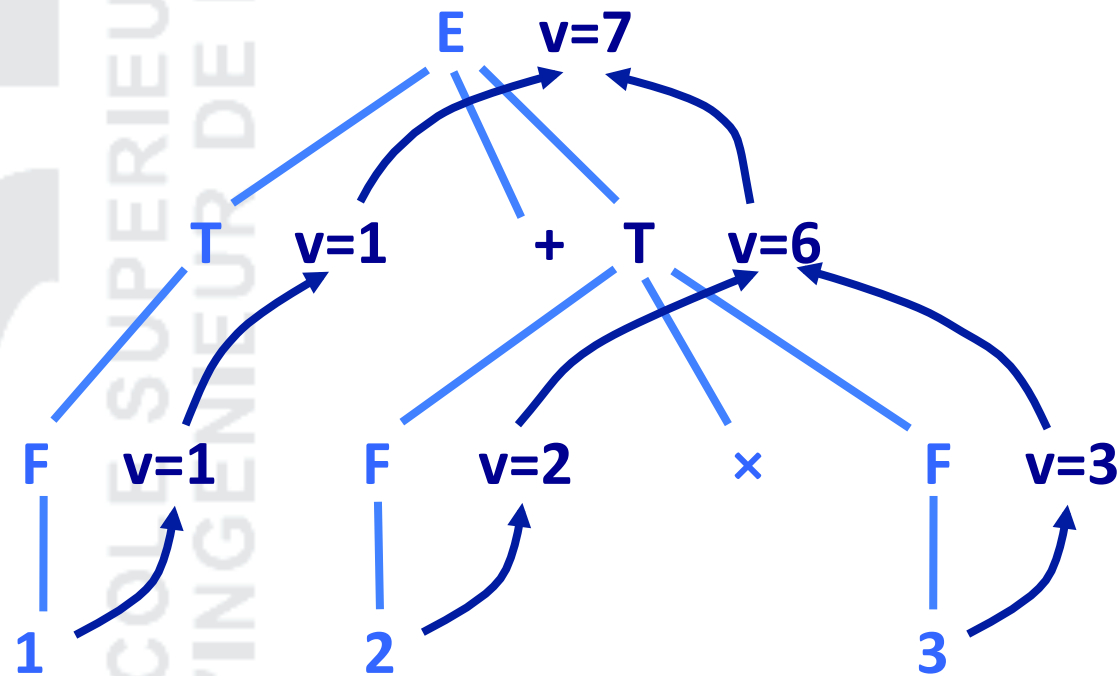
- $E \rightarrow T + T' \{E.v = T.v + T'.v\}$
- $E \rightarrow T \quad \{E.v = T.v\}$
- $T \rightarrow F \times F' \{T.v = F.v * F'.v\}$
- $T \rightarrow F \quad \{T.v = F.v\}$
- $F \rightarrow E \quad \{F.v = E.v\}$
- $F \rightarrow \text{digit} \quad \{F.v = \text{digit}.v\}$



Attention ! Il y a deux langages

$+ \neq +$ et $\times \neq *$

Avec le « vrai » arbre de dérivation



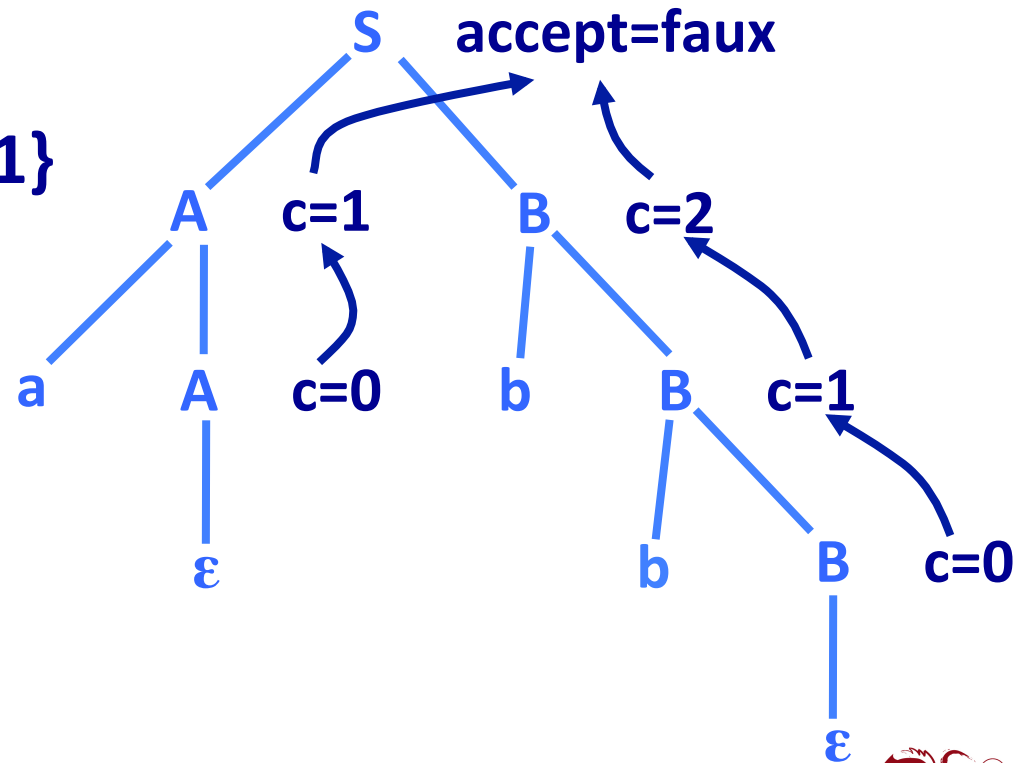
Aparté

Penser global - agir local

- Penser global = imaginer la circulation de l'information à l'échelle de **tous les attributs** de **tous les arbres possibles**
- Agir local = ne programmer la circulation de l'information qu'à l'échelle de la **règle isolée**

Exemple – $a^n b^n$

- $S \rightarrow A B \quad \{S.\text{accept} = (A.c == B.c)\}$
- $A \rightarrow aA' \quad \{A.c = A'.c+1\}$
- $A \rightarrow \epsilon \quad \{A.c = 0\}$
- $B \rightarrow bB' \quad \{B.c = B'.c+1\}$
- $B \rightarrow \epsilon \quad \{B.c = 0\}$



Aparté

Syntaxe vs. sémantique

La Syntaxe

Ce qui dit si un document est correctement composé

Les outils syntaxiques

RE, automates et grammaires

Les outils «sémantiques»

La Sémantique

Ce qui dit la signification d'un document (correctement composé)

GA, règles de déduction, ..., programmation à la main

Exemple

Déclarations / utilisations

$S \rightarrow \{'\} Ds ; Is '\}$ $\{Is.ts = Ds.ts ; S.ok = Is.ok\}$

$Ds \rightarrow d ; Ds'$ $\{Ds.ts = \{d.def\} \cup Ds'.ts\}$

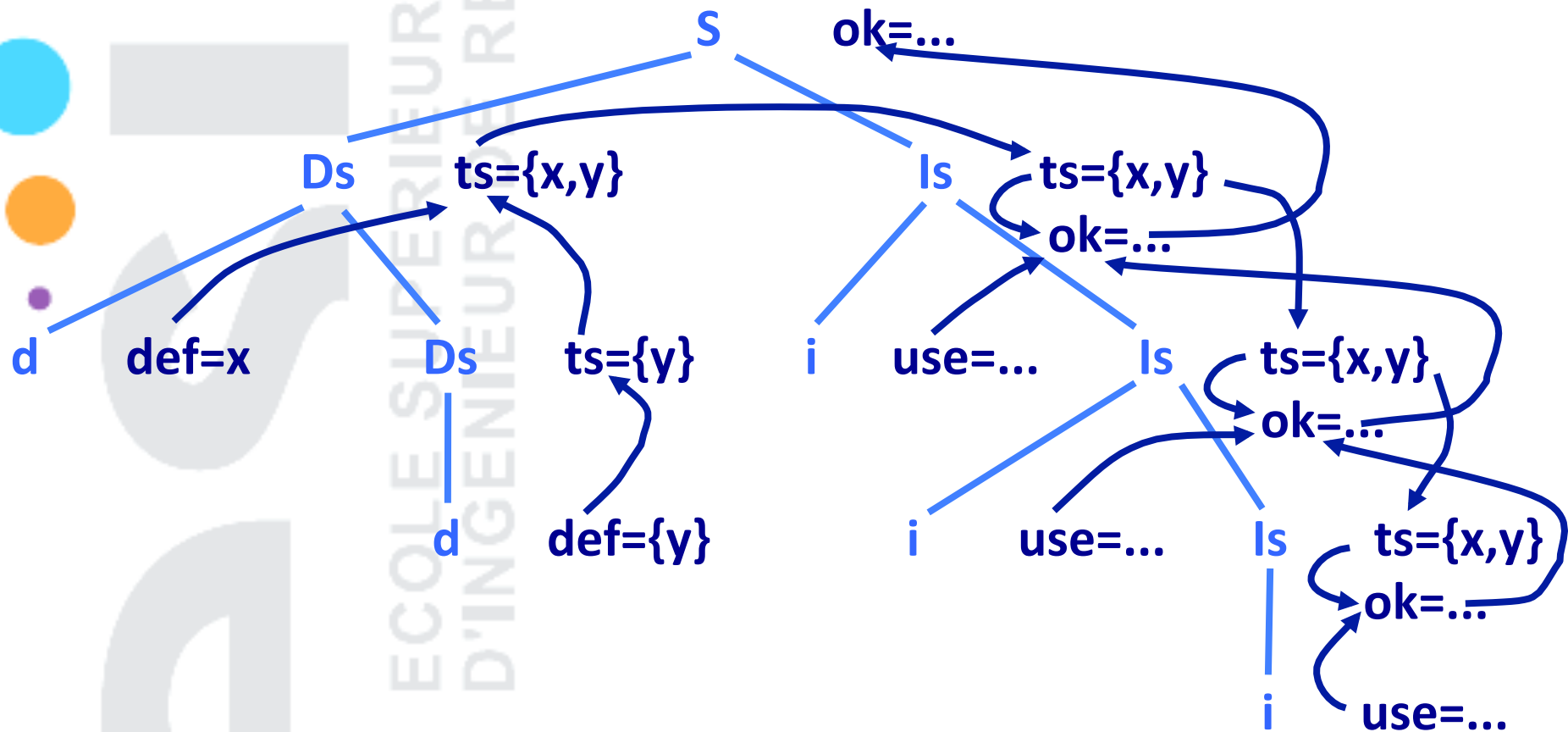
$Ds \rightarrow d$ $\{Ds.ts = \{d.def\}\}$

$Is \rightarrow i ; Is'$ $\{Is'.ts = Is.ts ;$
 $Is.ok = (i.use \subset Is.ts) \wedge Is'.ok\}$

$Is \rightarrow i$ $\{Is.ok = i.use \subset Is.ts\}$

Exemple

Déclarations / utilisations



Pragmatique des AG

- Faire en sorte que le résultat soit un attribut de l'axiome
- Penser global – agir local
- Ne pas oublier les initialisations
- Penser les (non-)terminaux comme à des prises orientées

Évaluation des attributs

- Deux difficultés
 - possibles dépendances circulaires entre règles sémantiques
 - difficulté d'entrelacer
 - construction de T
 - création des instances d'attribut
 - évaluation des règles sémantiques
 - (dans la partie mise en œuvre)**

Dépendances circulaires

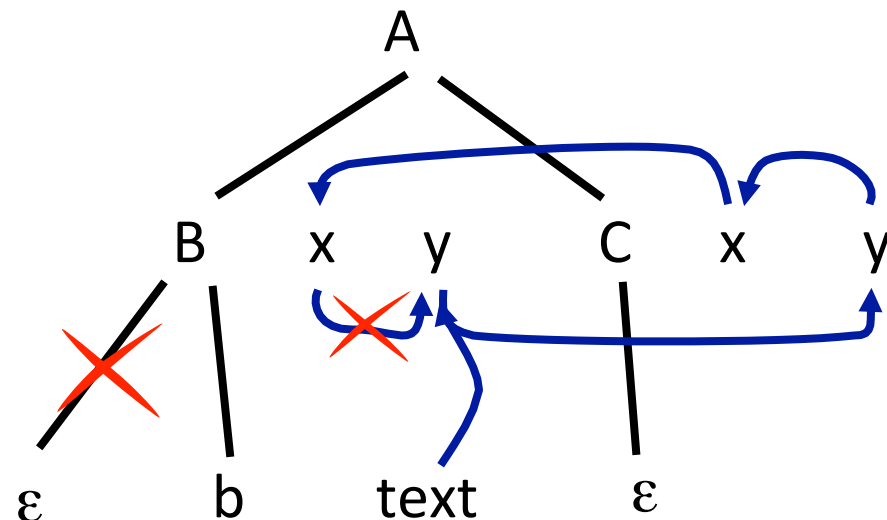
- Exemple

$A \rightarrow B C \quad \{B.x = C.x ; C.y = B.y\}$

$B \rightarrow b \quad \{B.y = b.text\}$

$B \rightarrow \varepsilon \quad \{B.y = B.x\}$

$C \rightarrow \varepsilon \quad \{C.x = C.y\}$



Détecter la circularité après la construction de l'arbre

- Algorithme du tri topologique
 - entrée : toutes les instances d'attributs, Attr
 - sortie : **un ordre total < sur Attr, tel que**
$$\forall a, a' \in \text{Attr}. a \rightarrow a' \Rightarrow a < a'$$

ou bien **échec**
 - complexité : $O(|\text{Attr}|)$

Détecter ...

après la construction de l'arbre

- Semble efficace
- Mais que faire d'un échec lors de l'analyse ?

échec \Rightarrow mot mal formé



échec \Rightarrow évaluation impossible

échec \Rightarrow grammaire mal formée

cela ne regarde pas l'utilisateur final !

- Détecter la circularité + en amont !

Détecter la circularité avant la construction de l'arbre

- Ne doit pas dépendre d'un mot m
- Montrer que

$\forall m \in \Sigma_T^*$. le graphe de dépendance
produit par l'analyse de m
ne contient pas de cycle

Détecter la circularité avant la construction de l'arbre

- Algorithme de Knuth (1968) et variantes
- Idée : représenter la fermeture transitive des dépendances connues de chaque non-terminal
- Complexité exponentielle

Algorithme de Knuth

- Chaque non-terminal peut avoir plusieurs ensembles de dépendances
 - plusieurs règles pour un même non-terminal
- Chaque non-terminal d'une règle influence le non-terminal de tête

Algorithme de Knuth

$A \rightarrow B C \quad \{B.x=C.x, C.y=B.y\}$

$B \rightarrow b \quad \{B.y=b.text\}$

$B \rightarrow \varepsilon \quad \{B.y=B.x\}$

$C \rightarrow \varepsilon \quad \{C.x=C.y\}$

$A \rightarrow B C \quad \{C.x \rightarrow B.x, B.y \rightarrow C.y\}$

$B \rightarrow b \quad \emptyset$

$B \rightarrow \varepsilon \quad \{B.x \rightarrow B.y\}$

$C \rightarrow \varepsilon \quad \{C.y \rightarrow C.x\}$

Algorithme de Knuth

	A	B	C
$r_{\text{sém}} + \text{dép.}$	$\{\{C.x \rightarrow B.x, B.y \rightarrow C.y\}\}$	$\{\{B.x \rightarrow B.y\}, \emptyset\}$	$\{\{C.y \rightarrow C.x\}\}$
ferm. trans.	$\{\{C.x \rightarrow B.x, B.y \rightarrow C.y\}\}$	$\{\{B.x \rightarrow B.y\}, \emptyset\}$	$\{\{C.y \rightarrow C.x\}\}$
$r_{\text{sém}} + \text{dép.}$	$\{\{C.x \rightarrow B.x, B.y \rightarrow C.y, B.x \rightarrow B.y, C.y \rightarrow C.x\},$ $\{C.x \rightarrow B.x, B.y \rightarrow C.y, C.y \rightarrow C.x\}\}$	-	-
ferm. trans.	$\{\{C.x \rightarrow B.x, B.y \rightarrow C.y, B.x \rightarrow B.y, C.y \rightarrow C.x, B.x \rightarrow B.x\},$ $\{C.x \rightarrow B.x, B.y \rightarrow C.y, C.y \rightarrow C.x, B.y \rightarrow B.x\}\}$	-	-

Biographie

- Donald Knuth (1938, États-Unis - ...)

- Champion de l'algorithmique

The Art of Computer Programming

- Auteur de TeX, METAFONT et Computer Modern
- N'a plus de mail depuis 1990

Rendre impossible la circularité

- Donner des critères de forme de grammaire qui rendent toute circularité impossible

Critères de non-circularité (1)

- Distinguer les attributs **synthétisés**

corps \rightarrow tête

- et les attributs **hérités**

tête \rightarrow corps

corps \rightarrow corps

Critères de non-circularité (2)

- Si **synthétisé seulement**,
alors **pas de circularité** (YACC)
- Si **hérité (et synthétisé)**,
alors **risque de circularité**
- Si **hérité (et synthétisé)**, mais **héritage seulement gauche**→**droite**,
alors **pas de circularité**

Bilan tests non-circularité

- Tri topologique

- G et m connus

trop tardif

- Algorithme de Knuth

- G connue

- conclusion pour tout m

trop complexe

- Critères de non-circularité

- sous-langage de G connu

- conclusion pour toute G du sous-langage et tout m

très simple

Bilan grammaires à attributs

- Grande flexibilité des GA
- Risque de dépendances circulaires
 - **traitement tardif efficace, mais embarrassant**
 - **traitement précoce convaincant, mais complexe**
 - **traitement préventif très facile**

Méthodologie GA - étude

- Identifier ce qu'on veut calculer
- Élaborer des cas d'étude
- Tracer les arbres de syntaxe
- Imaginer la circulation de l'information nécessaire au calcul
- Définir les attributs qui opèrent la circulation
- Les identifier comme synthétisés ou hérités

Méthodologie GA - contraintes

- Jamais de variables globales
- Jamais de propagation à distance

(pas de magie !)

- Un même type de nœud...

(les instances d'une même règle)

...fait toujours la même opération

- Respecter l'orientation hérité/synthétisé

Méthodologie GA - réalisation

- Lister les attributs synthétisés
- Lister les attributs hérités
- Définir les règles sémantiques
- Penser global, agir local
- Faire attention aux initialisations
- Tester la non-circularité

Mise en œuvre

grammaires à attributs

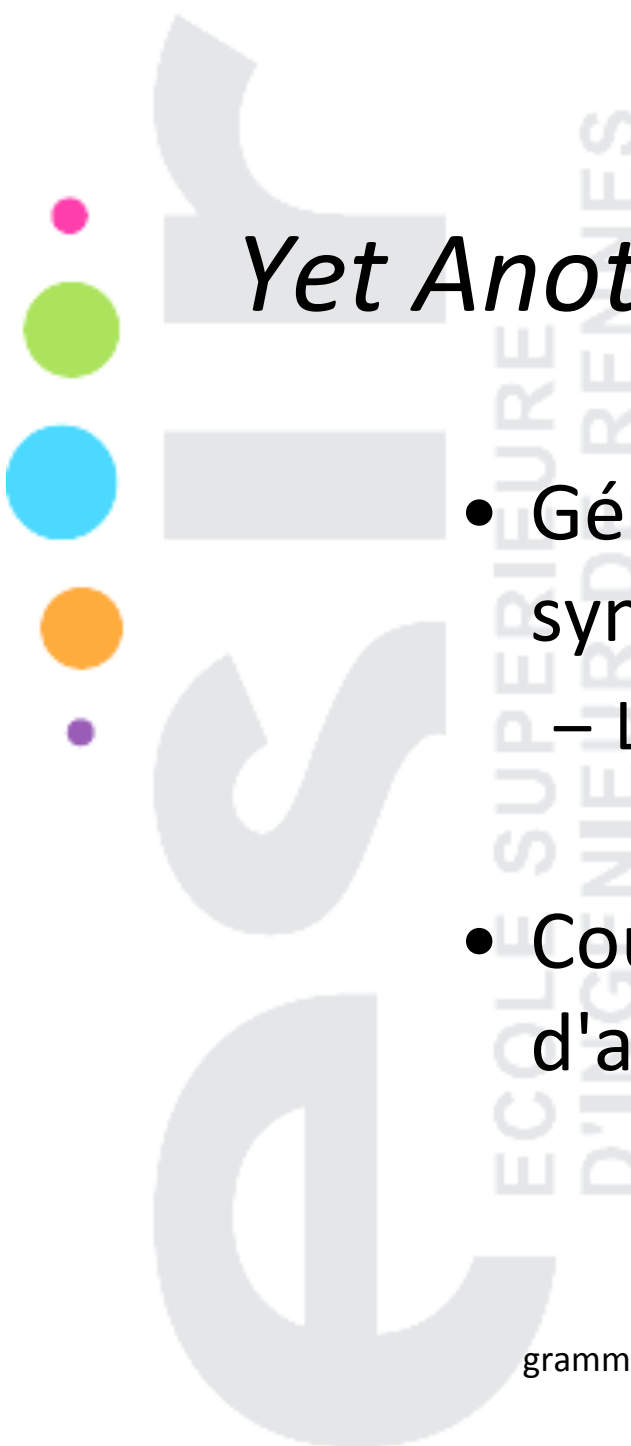
Mise en œuvre

- Entrelacer
 - construction de T
 - création des instances d'attribut
 - évaluation des règles sémantiques

YACC

- Ne pas entrelacer

ANTLR



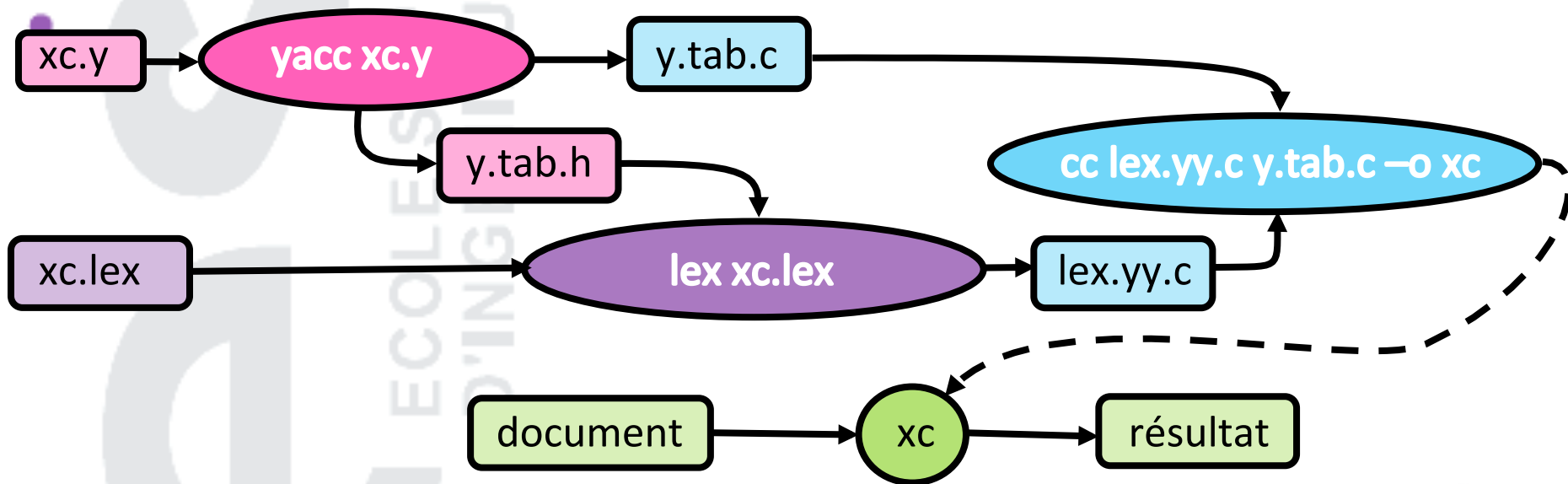
YACC (1975)

Yet Another Compiler Compiler

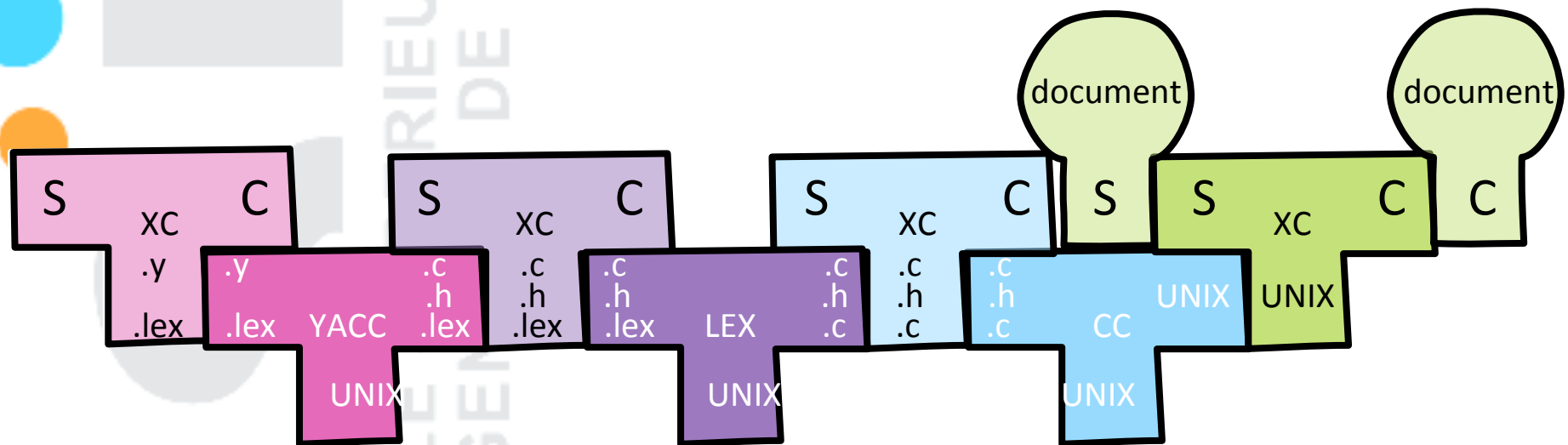
- Générateur d'analyseur syntaxique ascendant
 - LALR(1)
- Couplé avec le générateur d'analyseur lexical LEX

Génération du compilateur

- Fichiers lex et yacc
 - fichiers c
 - compilateur exécutable



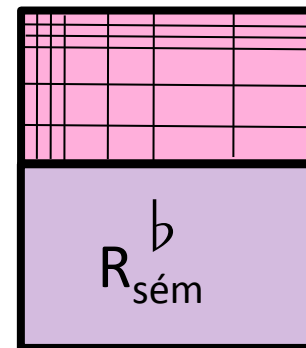
YACC en T



Génération du compilateur

- Règles syntaxiques
→ automate (table des transitions δ)
- Règles sémantiques
→ actions de l'automate

R_{syn}
+
 $R_{\text{sém}}$



Génération de $R_{\text{sém}}^b$

- Désignation des (non-)terminaux
 - $\$i$ désigne le i ème (non-)terminal du corps de la règle

ex. $E : E '+' T \quad \{\$\$ = \$1 + \$3\}$

- Traduction en adressage dans la pile de l'automate

ex. $[\text{top}-1], [\text{top}-12]$

Analyse ascendante

$E \rightarrow E + T$

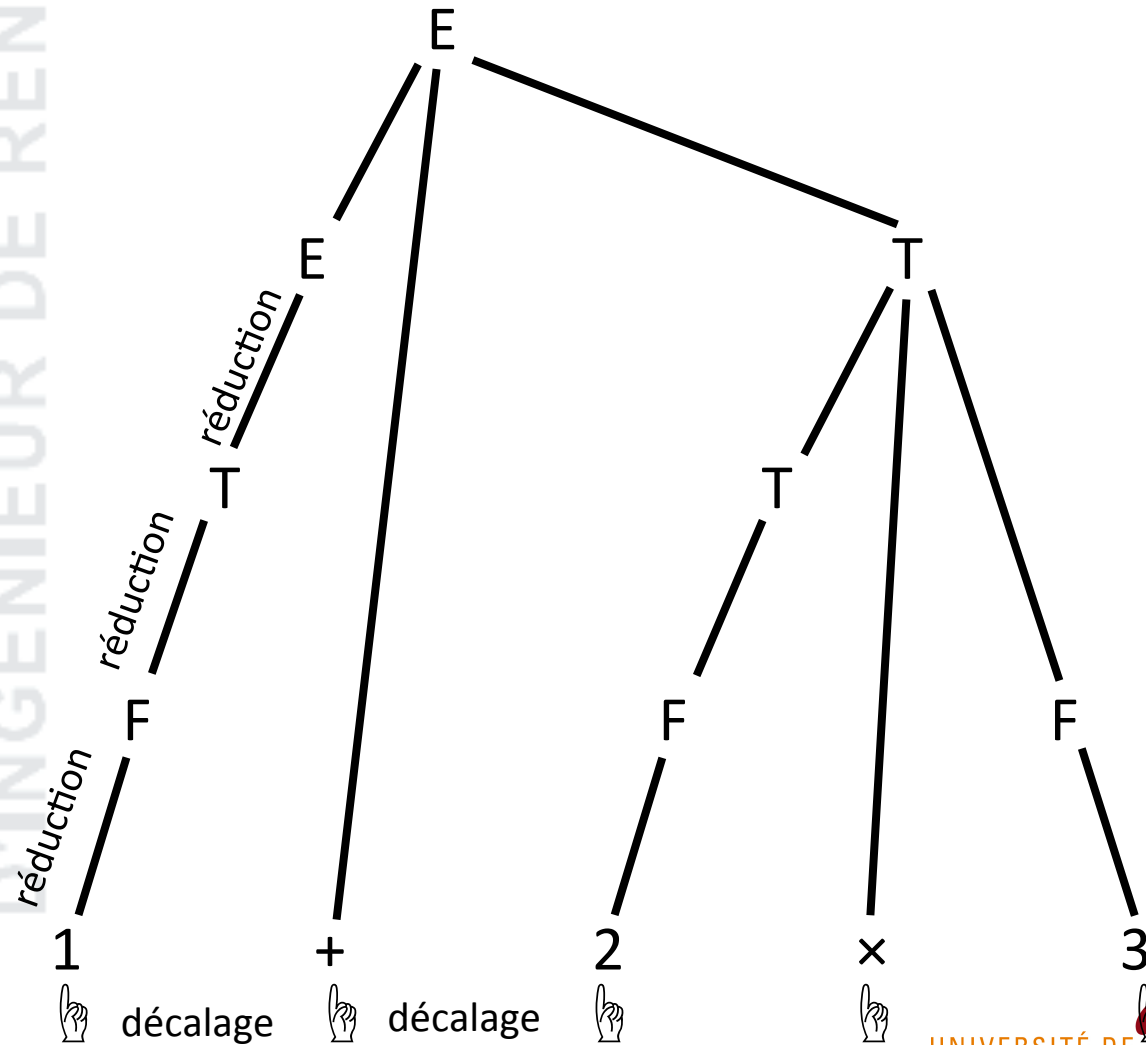
$E \rightarrow T$

$T \rightarrow T \times F$

$T \rightarrow F$

$F \rightarrow '(' E ')'$

$F \rightarrow \text{digit}$



grammaires à attributs

43

UNIVERSITÉ DE
RENNES 1

Évaluation entrelacée

$E \rightarrow E + T \{ \$\$ = \$1 + \$3 \}$

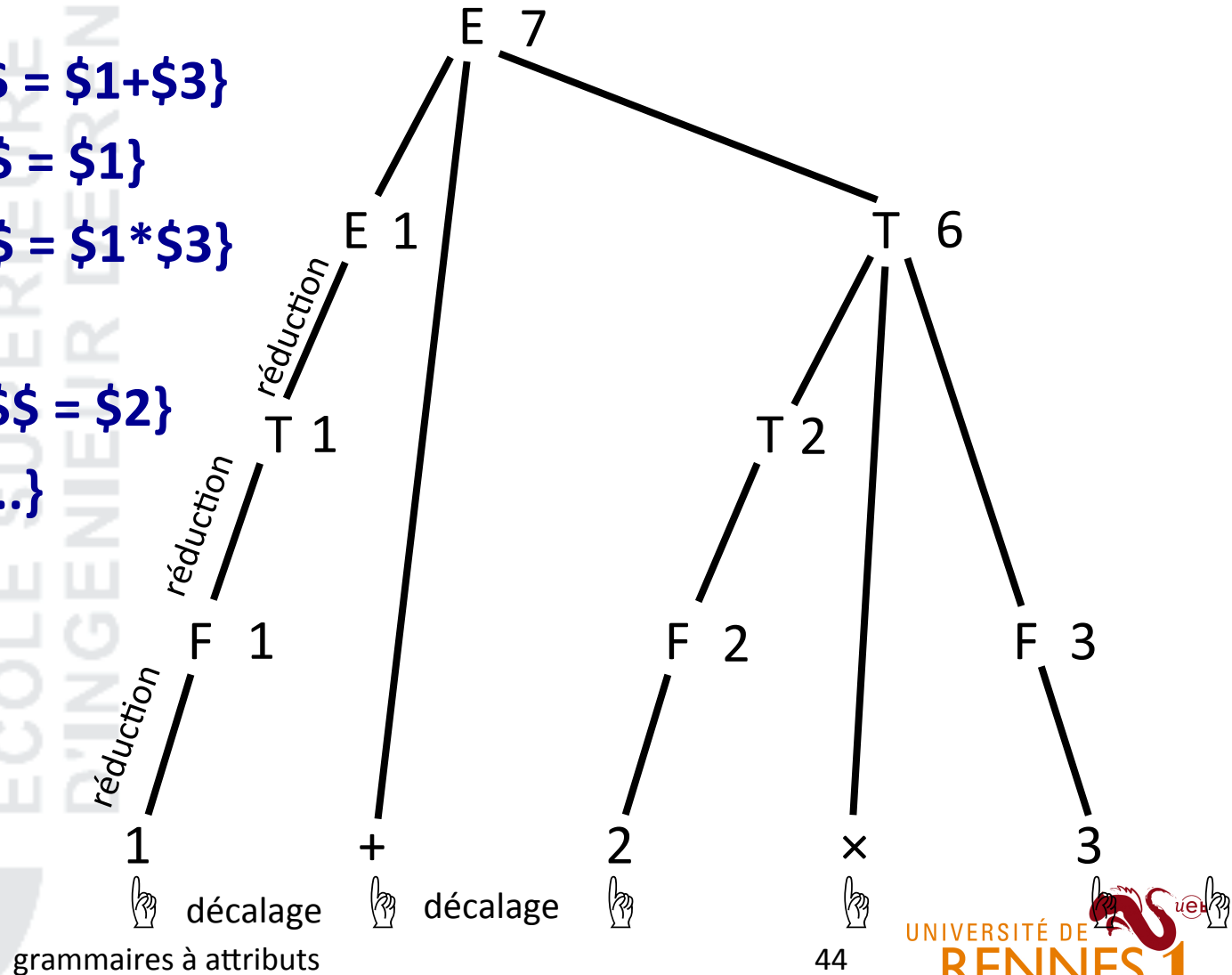
$E \rightarrow T \{ \$\$ = \$1 \}$

$T \rightarrow T \times F \{ \$\$ = \$1 * \$3 \}$

$T \rightarrow F$

$F \rightarrow '(' E ') \{ \$\$ = \$2 \}$

$F \rightarrow \text{digit} \{ \dots \}$



Analyse ascendante et évaluation entrelacée

- Marche très bien pour attributs synthétisés

Ne marche que pour attributs synthétisés !



- Attributs hérités → variables globales

ANTLR (~1995)

- Séparation possible de la construction de l'arbre syntaxique et de l'évaluation des attributs
- Officialisation de la notion de syntaxe abstraite
- Grammaire d'arbre de syntaxe abstraite

Attributs synthétisés

≡

résultats d'appel de procédure

- Toute règle peut produire un résultat
≡ attributs synthétisés

expr returns [int v] :

term '+' expr { \$v = \$term.v + \$expr.v } ;

Attributs hérités

≡

paramètres d'appel de procédure

- Toute règle peut dépendre de paramètres
≡ attributs hérités

expr [TS * ts] returns [int v] :

term [ts] '+' expr [ts] {\$v = \$term.v + \$expr.v} ;

Construction de l'arbre vs. évaluation

- Analyse par descente récursive
 - chaque règle \equiv procédure
 - expansion \equiv appel de procédure
 - héritage
 - réduction \equiv retour de procédure
 - synthèse
- Entrelacement possible, **mais...**



Officialisation de la syntaxe abstraite

- Annotation

\wedge et !

- Réécriture

$\rightarrow \wedge(\dots)$

- Grammaire d'arbre de syntaxe abstraite

$\text{expr returns [int v]} :$

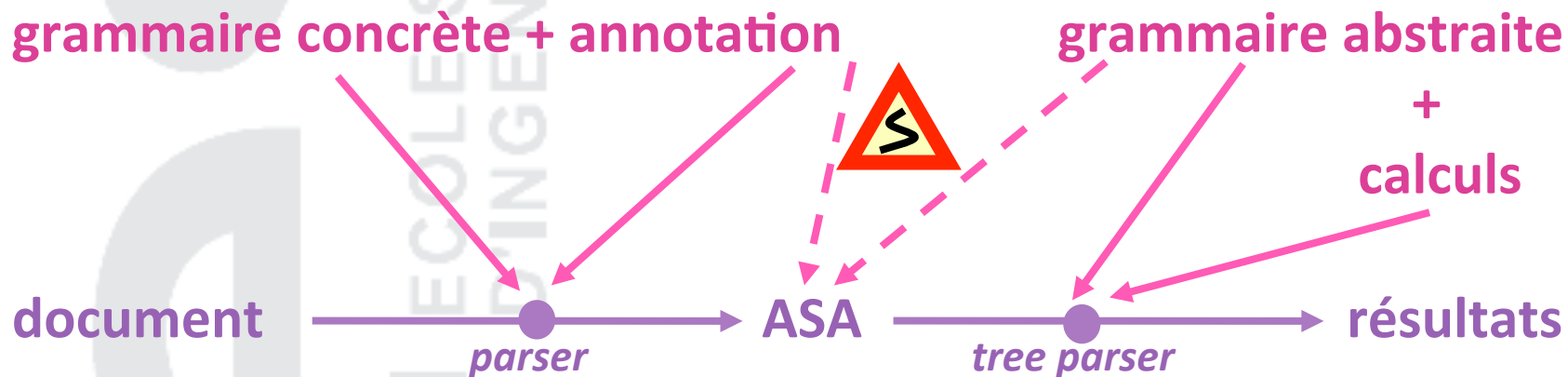
$\wedge('+' a=\text{expr } b=\text{expr}) \{ \$v = a+b; \} ;$

Stratégie recommandée

- Limiter l'usage des attributs de la *grammar* aux contrôles syntaxiques
- Utiliser les attributs de la *tree grammar* pour les calculs plus sémantiques

Aparté - Situation assez bizarre

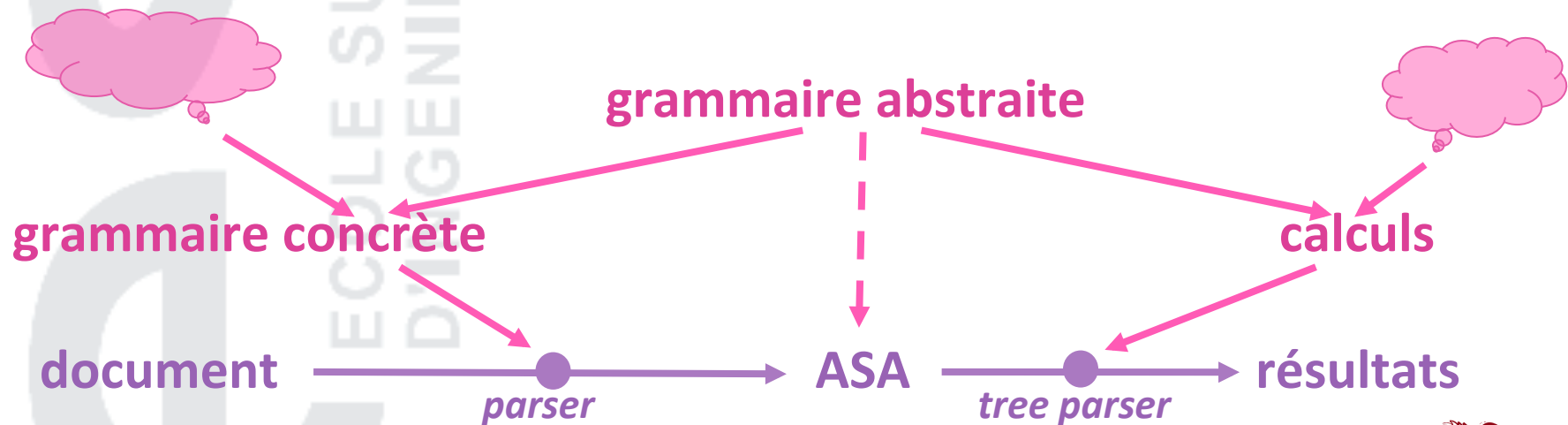
- Grammaire concrète d'abord...
- ...puis arbre de syntaxe abstraite
- ...puis (!) grammaire abstraite
- ...puis calculs



grammaires à attributs

Amélioration nécessaire

- Commencer par la grammaire abstraite...
...en déduire une grammaire concrète
...y [la GA] accrocher des calculs



Conclusion

- GA est l'outil de base pour les calculs dirigés par la syntaxe
- Penser global – agir local
- Concevoir en termes de
 - grammaire abstraite
 - attribut synthétisé/hérité

**S'adapter dans un second temps
selon la mise en œuvre**