ANALYSE DESCENDANTE DETERMINISTE METHODE LL(1)

Marie-Laure Potet
Ensimag première année - Cours Théorie des Langages

1er février 2012

1 Rappels

Les rappels ci-dessous sont succincts, pour tout complément se reporter au polycopié *Théorie des Langages*.

1.1 Grammaire hors-contexte

Une grammaire hors-contexte est un quadruplet $G = (V_T, V_N, S, R)$ avec :
- V_T et V_N deux vocabulaires disjoints ;
- S un élément distingué de V_N appelé axiome ;
- R un ensemble de règles de la forme $A \rightarrow \omega$ avec A élément de V_N et ω chaîne sur $V_T \cup V_N$.

Par convention on notera en minuscule les éléments de V_T et en majuscule les éléments de V_N. Lorsque ceci n'est pas précisé le non-terminal apparaissant à gauche de la première règle est l'axiome. Ceci permet de définir une grammaire en ne donnant que ses règles de production.

$\Rightarrow$ dénote la relation de dérivation, $\stackrel{*}{\Rightarrow}$ dénote la fermeture réflexive et transitive de cette relation.

Remarques :

1. La notation de chaîne vide, ε, n'est pas un élément du vocabulaire terminal.
2. On ne considère dans la suite que des grammaires *réduites*, c'est-à-dire dont tous les symboles sont productifs et accessibles (voir polycopié *Théorie des Langages).*

1.2 Reconnaisseurs pour les langages hors-contexte

Le langage engendré par une grammaire $G = (V_T, V_N, S, R)$, noté $L(G)$, est défini par :

$$L(G) = \{\omega \in V_T^* \mid \ S \stackrel{*}{\Rightarrow} \omega\}$$

On montre qu'un mot ω appartient à $L(G)$ en construisant une dérivation de l'axiome vers ω. Une dérivation est souvent représentée sous forme arborescente (arbre de dérivation), cette forme permet de ne pas distinguer des dérivations ne différant que par l'ordre d'applications des règles.

Un reconnaisseur est un automate qui permet, à partir d'une grammaire G et d'une chaine ω de répondre en temps fini à la question "$\omega \in L(G)$?". Répondre *oui* à la question "$\omega \in L(G)$?" revient donc à construire une dérivation $S \overset{*}{\Rightarrow} w$ et répondre *non* à cette même question revient à montrer qu'aucune dérivation ne conduit à ω. Les résultats sont les suivants :

- Le problème de la reconnaissance est décidable pour les langages hors-contexte ;
- Les algorithmes généraux de reconnaissance pour les langages hors-contexte sont non-déterministes (ils peuvent nécessiter des retour-arrières sur la chaîne d'entrée) ;
- Il existe des sous-classes de langages hors-contextes pour lesquelles il est possible d'appliquer des méthodes d'analyse déterministe. Par exemple la sous-classe des langages $LL(1)$ étudiée ici.

Ceci diffère des langages réguliers pour lesquels il existe toujours un reconnaisseur déterministe, par exemple sous la forme d'un automate fini déterministe.

2 Grammaires LL(1)

Les grammaires LL(1) permettent d'effectuer une analyse descendante **déterministe** (ou analyse prédictive) en ne considérant que le symbole courant de la phrase à analyser.

2.1 Analyse Descendante

Le principe général de l'analyse descendante est, partant de l'axiome, de réécrire ce dernier en la phrase à analyser. On utilise pour ce faire les règles de la grammaire pour réécrire le **non-terminal le plus à gauche**.

On décrit, avec les deux règles ci-dessous, la manière d'engendrer systématiquement tous les modèles de phrase.

1. L'axiome S est un modèle de phrase.
2. Soit $\omega_1 A \omega_2$ avec $\omega_1 \in V_T^*$, $A \in V_N$ et $\omega_2 \in (V_T \cup V_N)^*$, un modèle de phrase et soit $A \to \omega$ une règle de la grammaire, alors $\omega_1 \omega \omega_2$ est un modèle de phrase.

On peut imaginer un algorithme non déterministe qui utilise la règle 2 tant que la chaîne à analyser n'a pas été engendrée ou tant qu'on n'a pas construit suffisamment de modèles de phrase pour décider qu'aucun ne conduira à la phrase à analyser. Cette décision dépend de la forme des règles de la grammaire et de la taille de la chaîne à analyser. Il existe toujours une valeur permettant de borner la longueur des dérivations.

Le non déterminisme de l'algorithme, esquissé ci-dessus, résulte uniquement du choix de la règle à utiliser pour réécrire le non terminal le plus à gauche. Les grammaires LL(1) permettent d'effectuer ce choix de façon déterministe en consultant le symbole courant de la phrase à analyser. Dans le sigle LL(1) le premier L (pour Left) signifie qu'on lit la chaîne à analyser de gauche à droite, le second L qu'on construit une dérivation la plus à gauche et le 1 qu'on consulte un seul symbole sur la chaîne à analyser. Cette méthode peut bien sûr s'étendre en autorisant la consultation de k symboles en avance. Avant d'étudier comment déterminer si une grammaire est LL(1) ou non, regardons quelques exemples.

Exemple 1 Soit G_1 la grammaire suivante :

$$
\begin{array}{lcll}
(1) & S & \rightarrow & a \\
(2) & S & \rightarrow & Ac \\
(3) & A & \rightarrow & bAa \\
(4) & A & \rightarrow & c
\end{array}
$$

avec $V_T = \{a, b, c\}$.

Dans cet exemple le choix de la règle à appliquer ne pose pas de problème. En effet si on doit réécrire le non-terminal S alors soit le caractère courant est a et on applique la première règle, soit le caractère courant est b ou bien c et on applique la seconde règle, puisque la réécriture de A produira b ou c (règles 3 et 4). De la même manière, lorsqu'il s'agit de réécrire le non-terminal A, la consultation du caractère courant nous permet de choisir entre les règles (3) et (4). On peut donc construire un tableau qui décrit quelle règle choisir en fonction du non-terminal à réécrire et du caractère courant. On désignera par *erreur* l'impossibilité de choisir une règle. Pour la grammaire G_1 ce tableau se présente sous la forme :

	a	b	c
S	(1)	(2)	(2)
A	*erreur*	(3)	(4)

Pour chaque non-terminal et chaque terminal courant une règle au plus peut être choisie, la grammaire G_1 est donc LL(1).

Exemple 2 Ajoutons à la grammaire G_1 la règle $S \rightarrow bS$. La grammaire G_2 obtenue est :

$$
\begin{array}{lcll}
(1) & S & \rightarrow & a \\
(2) & S & \rightarrow & Ac \\
(3) & S & \rightarrow & bS \\
(4) & A & \rightarrow & bAa \\
(5) & A & \rightarrow & c
\end{array}
$$

Cette grammaire n'est pas LL(1) puisque, par exemple lorsqu'un mot débute par b, on peut construire deux dérivations qui produisent toutes deux le terminal b en tête, en utilisant deux façons différentes de réécrire S ($S \Rightarrow bS$ et $S \Rightarrow Aa \Rightarrow bAaa$). Lorsque le non-terminal à réécrire est S et l'élément courant est b, on ne sait donc pas choisir entre les règles 2 et 3.

Exemple 3 Sur des grammaires ne contenant pas de ε-règle, il est assez facile d'intuiter si une grammaire est LL(1) ou non. Examinons maintenant des exemples contenant des ε-règles. Remplaçons la dernière règle de la grammaire G_1 par $A \rightarrow \varepsilon$. La grammaire G_3 obtenue est :

$$
\begin{array}{lcll}
(1) & S & \rightarrow & a \\
(2) & S & \rightarrow & Ac \\
(3) & A & \rightarrow & bAa \\
(4) & A & \rightarrow & \varepsilon
\end{array}
$$

Nous devons maintenant pouvoir choisir entre les 2 règles $A \rightarrow bAa$ et $A \rightarrow \varepsilon$. L'utilisation de la dernière règle a pour effet de transformer un modèle de phrase de la forme $\omega_1 A \omega_2$ en $\omega_1 \omega_2$ et

donc l'élément courant de la chaîne à reconnaître est un élément du vocabulaire terminal qui peut être placé derière le non-terminal A.

Par exemple si on veut reconnaître le mot c on va construire la dérivation $S \Rightarrow Ac \Rightarrow c$, l'élément courant au moment du choix entre les deux règles $A \rightarrow bAa$ et $A \rightarrow \varepsilon$ est alors c. Dans la grammaire G_3 les éléments qui peuvent suivre le non-terminal A dans une dérivation à partir de l'axiome sont a et c, à cause des règles $A \rightarrow bAa$ et $S \rightarrow Ac$. La règle $A \rightarrow bAa$, quant à elle, n'est choisie que lorsque le caractère courant est b, il n'y a donc pas de problème de choix entre les règles 3 et 4.

De la même manière il faut pouvoir choisir entre les deux règles permettant de réécrire le non-terminal S. La règle $S \rightarrow Ac$ peut s'appliquer lorsque le caractère courant est soit b soit c. En effet on peut construire les deux dérivations $S \Rightarrow Ac \Rightarrow bAac$ et $S \Rightarrow Ac \Rightarrow c$. On peut donc choisir les règles à appliquer de la manière suivante :

	a	b	c
S	(1)	(2)	(2)
A	(4)	(3)	(4)

Exemple 4 Si on considère maintenant les règles $S \rightarrow Ac$, $A \rightarrow \varepsilon$ et $A \rightarrow c$ la grammaire obtenue n'est pas LL(1). En effet on peut alors construire les deux dérivations $S \Rightarrow Ac \Rightarrow cc$ et $S \Rightarrow Ac \Rightarrow c$ qui produisent toutes deux une chaîne débutant par c, en utilisant des règles différentes ayant même non-terminal en partie gauche.

2.2 Conditions LL(1)

Nous donnons maintenant une méthode systématique pour décider si une grammaire est LL(1) ou non. Pour cela on calcule pour chaque règle l'ensemble des "symboles courants" qui peuvent être produits par cette règle, dans une dérivation quelconque à partir de l'axiome. Cet ensemble est appelé l'ensemble des symboles **directeurs** associés à une règle. La fonction *Directeur* associe à chaque règle son ensemble de symboles directeurs.

$$Directeur : R \longrightarrow \mathcal{P}(V_T \cup \{\$\})$$

$$\text{avec : } Directeur(A \rightarrow \omega) =$$

$$\{x \in V_T \cup \{\underline{\$}\} : \exists\ \omega_1,\ \omega_2,\ \omega_3\ S\underline{\$} \overset{*}{\Rightarrow} \omega_1 A \omega_2 \Rightarrow \omega_1 \omega \omega_2 \overset{*}{\Rightarrow} \omega_1 x \omega_3\}$$

Remarques :

1. Le modèle de phrase initial est $S\underline{\$}$ (où \$ est un nouveau symbole n'appartenant pas à V_T), ceci afin que la fonction *Directeur* soit toujours définie. On a alors $Directeur(A \rightarrow \omega) \neq \emptyset$, $\forall A \rightarrow \omega$.
2. La notation de chaîne vide ε n'est évidemment jamais un symbole directeur de règle (ε n'est pas élément de V_T).
3. Lorsque ω ne dérive pas vers ε (notation $\omega \overset{*}{\not\Rightarrow} \varepsilon$) on peut simplifier la définition en $Directeur(A \rightarrow \omega) = \{x \in V_T : \exists \omega_1\ \omega \overset{*}{\Rightarrow} x\omega_1\}$. En effet, dans ce cas, la réécriture de la partie droite de règle produit toujours un élément de V_T.

On peut alors donner la définition suivante :

Définition 1 *Grammaires LL(1)*

Une grammaire $G = (V_T, V_N, S, R)$ est LL(1) si et seulement si elle vérifie la condition :

$$\forall\ A, \omega_1, \omega_2\ \ (A \to \omega_1 \in R \wedge\ A \to \omega_2 \in R\ \wedge\ \omega_1 \neq \omega_2 \Rightarrow Directeur(A \to \omega_1) \cap Directeur(A \to \omega_2) = \emptyset)$$

On énonce ici le fait que, lorsqu'on doit réécrire un non-terminal A et que celui-ci peut être réécrit de différentes manières, on peut toujours choisir la règle à utiliser en fonction de l'élément courant de la phrase à reconnaître. Remarquons que les comparaisons entre ensembles de directeurs ne portent que sur des règles ayant même non-terminal en partie gauche.

Exemple :

Dans l'exemple 2 de la section précédente on peut construire les deux dérivations suivantes :

- $S\underline{\$} \Rightarrow Ac\underline{\$} \Rightarrow bAac\underline{\$}$ (par les règles $S \to Ac$ et $A \to bAa$)
- $S\underline{\$} \Rightarrow bS\underline{\$}$ (par la règle $S \to bS$)

On a donc $b \in Directeur(S \to Ac)$ et $b \in Directeur(S \to bS)$. L'intersection de ces deux ensembles est non vide, la grammaire G_2 n'est donc pas LL(1).

De la même manière dans l'exemple 4 on peut construire les deux dérivations :

- $S\underline{\$} \Rightarrow Ac\underline{\$} \Rightarrow c\underline{\$}$ (par la règle $A \to \varepsilon$)
- $S\underline{\$} \Rightarrow Ac\underline{\$} \Rightarrow cc\underline{\$}$ (par la règle $A \to c$).

On a $c \in Directeur(A \to c) \cap Directeur(A \to \varepsilon)$ et donc la condition LL(1) n'est pas vérifiée.

2.3 Calcul des conditions LL(1)

Nous allons maintenant étudier comment calculer les ensembles de directeurs. La définition donnée dans la section précédente n'est pas constructive dans le sens où elle fait intervenir l'ensemble des dérivations possibles de la forme $S\underline{\$} \stackrel{*}{\Rightarrow} \omega_1 A \omega_2$, cet ensemble est en général infini. Nous allons donner ci-dessous une méthode constructive permettant de ramener le calcul des directeurs à un calcul fini. Pour cela nous définissons ci-dessous deux fonctions.

Soient $V_T^{'} = V_T \cup \{\underline{\$}\}$ et $V = V_N \cup V_T^{'}$.

- $Premier : V^* \to \mathcal{P}(V_T^{'})$
  avec : Premier$(\omega) = \{\mathbf{x} \in V_T^{'} : \exists\ \omega_1 \in V_T^{'*},\ \omega \stackrel{*}{\Rightarrow} \mathbf{x}\omega_1\}$
- $Suivant : V_N \to \mathcal{P}(V_T^{'})$
  avec : Suivant(A) $= \{\mathbf{x} \in V_T^{'} : \exists\ \omega_1 \in V_T^*, \exists\ \omega_2 \in V_T^{'*},\ S\underline{\$} \stackrel{*}{\Rightarrow} \omega_1 A \mathbf{x} \omega_2\}$

Remarques :

1. Les définitions de *Premier* et *Suivant* ne sont pas constructives puisqu'elles utilisent aussi la notion de dérivations de longueur quelconque.
2. On peut avoir $Premier(\omega) = \emptyset$ si et seulement si $\mathcal{L}(\omega) = \{\varepsilon\}$.
3. On a toujours $Suivant(A) \neq \emptyset$, puisque $\mathbf{x}\omega_2$ contient au moins $\underline{\$}$.

On peut redéfinir la fonction *Directeur* à l'aide de ces nouvelles fonctions.

Définition 2 *Calcul des ensembles Directeur*

$$\begin{array}{c} Directeur(A \rightarrow \omega) \\ = \\ Premier(\omega) \ \cup \ (\texttt{si} \ \omega \stackrel{*}{\Rightarrow} \varepsilon \ \texttt{alors} \ Suivant(A) \ \texttt{sinon} \ \emptyset) \end{array}$$

Rappelons que $\omega \stackrel{*}{\Rightarrow} \varepsilon$ signifie $\varepsilon \in \mathcal{L}(\omega)$ et non pas $\mathcal{L}(\omega) = \{\varepsilon\}$ et que $\omega \stackrel{*}{\not\Rightarrow} \varepsilon$ signifie que ε n'est pas élément de $\mathcal{L}(\omega)$.

La définition ci-dessus est équivalente à la définition 1 si la grammaire ne contient pas de symboles inaccessibles. En effet si on considère la règle $A \rightarrow a$, A étant inaccessible, par la première définition nous aurons $Directeur(A \rightarrow a) = \emptyset$ et par la définition ci-dessus $Directeur(A \rightarrow a) = \{a\}$. La restriction sur les grammaires provient du fait que nous transformons une définition basée sur les dérivations en une définition basée sur les règles. Pour cela on doit garantir qu'il y a correspondance entre les dérivations et les règles et donc qu'il n'y a pas de "parasites" dans la grammaire.

Nous allons maintenant donner une méthode constructive permettant de calculer les fonctions *Premier* et *Suivant*, à partir des règles de la grammaire. Les mêmes restrictions que précédemment sont faites. La méthode considérée consiste à définir les fonctions *Premier* et *Suivant* comme les plus petites solutions d'équations. Ces fonctions peuvent être définies par cas sur les chaînes et les définitions obtenues peuvent être récursives, en raison de la récursivité qui peut apparaître dans la grammaire.

Définition 3 *Calcul des ensembles Premier*

$$\begin{array}{lcll} Premier(\varepsilon) & = & \emptyset & \\ Premier(x) & = & \{x\} & \text{si } x \in V_T^{'} \\ Premier(A) & = & \bigcup_{A \rightarrow \omega_i} Premier(\omega_i) & \text{si } A \in V_N \\ Premier(X.\omega) & = & Premier(X) \cup & \\ & & (\texttt{si} \ X \stackrel{*}{\Rightarrow} \varepsilon \ \texttt{alors} \ Premier(\omega) \ \texttt{sinon} \ \emptyset) & \text{si } X \in V \ \texttt{et} \ \omega \in V^* \end{array}$$

Définition 4 *Calcul des ensembles Suivant*

Pour tout non-terminal A :

$$Suivant(A) \ = \ \bigcup_{B \rightarrow \omega_1 A \omega_2} (Premier(\omega_2) \ \cup_{si \ \omega_2 \stackrel{*}{\Rightarrow} \varepsilon} Suivant(B) \ \cup_{si \ A \ est \ l'axiome} \{\$\}$$

avec $A, B \in V_N$ et $\omega_1, \omega_2 \in V^*$

Pour calculer *Premier* et *Suivant* on écrit les équations à l'aide des définitions ci-dessus, puis on les transforme par substitution jusqu'à obtenir un système ne contenant en partie droite que

des fonctions portant sur des terminaux et des non-terminaux de la grammaire. On peut ici, étant donnée la forme linéaire des équations, résoudre simplement les équations récursives. Une équation de la forme $X = X \cup B$ a pour plus petite solution $X = B$.

Exemple 5 Soit la grammaire dont les règles de production sont $A \to AB$, $A \to \varepsilon$ et $B \to a$. On veut calculer $Premier(A)$. En appliquant la définition on obtient $Premier(A) = Premier(AB) \cup Premier(\varepsilon)$, c'est à dire :

$$\begin{aligned} Premier(A) &= Premier(A) \cup Premier(B) \\ Premier(B) &= Premier(a) \end{aligned}$$

La première équation provient du fait que $A \stackrel{*}{\Rightarrow} \varepsilon$. On obtient alors les équations récursives suivantes :

$$\begin{aligned} Premier(A) &= Premier(A) \cup \{a\} \\ Premier(B) &= \{a\} \end{aligned}$$

On résout la première équation en $Premier(A) = \{a\}$, qui est la plus petite solution de cette équation. Par exemple {a, b} est aussi solution de cette équation, mais ce n'est pas la plus petite. On voit sur cet exemple que retenir la plus petite solution garantit qu'on n'obtient que des éléments atteignables par dérivation.

Appliquons maintenant ces calculs aux exemples 3 et 4 de la section **2.1**.

Exemple 6 Soit la grammaire :

$$\begin{aligned} S &\to a \\ S &\to Ac \\ A &\to bAa \\ A &\to \varepsilon \end{aligned}$$

On a alors :

$$\begin{aligned} Directeur(S \to a) &= Premier(a) = \{a\} \\ Directeur(S \to Ac) &= Premier(A) \cup Premier(c) \\ &= Premier(A) \cup \{c\} \\ Directeur(A \to bAa) &= Premier(b) = \{b\} \\ Directeur(A \to \varepsilon) &= Premier(\varepsilon) \cup Suivant(A) = Suivant(A) \end{aligned}$$

On doit donc calculer $Premier(A)$ et $Suivant(A)$. En appliquant les définitions on obtient $Premier(A) = Premier(bAc) \cup Premier(\varepsilon) = \{b\}$. Pour calculer $Suivant(A)$ on considère les deux règles ayant A en partie droite, c'est à dire $S \to Ac$ et $A \to bAa$. On a donc $Suivant(A) = Premier(c) \cup Premier(a) = \{a, c\}$.

$$\begin{aligned} Directeur(S \to a) &= \{a\} \\ Directeur(S \to Ac) &= \{b, c\} \\ Directeur(A \to bAa) &= \{b\} \\ Directeur(A \to \varepsilon) &= \{a, c\} \end{aligned}$$

Les deux premiers ensembles sont disjoints, les deux derniers aussi donc la grammaire est LL(1).

Exemple 7 Rajoutons maintenant la règle $A \rightarrow c$ (exemple 4 précédent). On a alors $Directeur(A \rightarrow \varepsilon) = \{a, c\}$ et $Directeur(A \rightarrow c) = \{c\}$. La grammaire n'est donc pas LL(1).

Exemple 8 Finissons sur un exemple plus conséquent : une grammaire LL(1) des expressions arithmétiques définies sur le vocabulaire $V_T = \{num, add, mult, (,)\}$. Soit la grammaire :

$$\begin{array}{rcl} E & \rightarrow & T\ SE \\ SE & \rightarrow & add\ T\ SE \\ SE & \rightarrow & \varepsilon \\ T & \rightarrow & F\ ST \\ ST & \rightarrow & mult\ F\ ST \\ ST & \rightarrow & \varepsilon \\ F & \rightarrow & (E) \\ F & \rightarrow & num \end{array}$$

On a alors :

$$\begin{array}{lcl} Directeur(E \rightarrow T\ SE) & = & Premier(T) \\ Directeur(SE \rightarrow add\ T\ SE) & = & \{add\} \\ Directeur(SE \rightarrow \varepsilon) & = & Suivant(SE) \\ Directeur(T \rightarrow F\ ST) & = & Premier(F) \\ Directeur(ST \rightarrow mult\ F\ ST) & = & \{mult\} \\ Directeur(ST \rightarrow \varepsilon) & = & Suivant(ST) \\ Directeur(F \rightarrow (E)) & = & \{(\} \\ Directeur(F \rightarrow num) & = & \{num\} \end{array}$$

Nous devons donc calculer $Premier(T)$, $Premier(F)$, $Suivant(SE)$ et $Suivant(ST)$. Commençons par les Suivants :

$$\begin{array}{rcl} Suivant(SE) & = & Suivant(SE) \cup Suivant(E) \\ Suivant(E) & = & \{), \underline{\$}\} \\ Suivant(ST) & = & Suivant(ST) \cup Suivant(T) \\ Suivant(T) & = & Premier(SE) \cup Suivant(SE) \cup Suivant(E) \end{array}$$

Ce qui donne :

$$\begin{array}{rcl} Suivant(SE) & = & \{), \underline{\$}\} \\ Suivant(E) & = & \{), \underline{\$}\} \\ Suivant(ST) & = & Suivant(T) \\ Suivant(T) & = & Premier(SE) \cup \{), \underline{\$}\} \end{array}$$

Il reste maintenant à calculer $Premier(T)$, $Premier(SE)$ et $Premier(F)$. On a $Premier(T) =$

$Premier(F)$, $Premier(SE) = \{add\}$ et $Premier(F) = \{(, num\}$. Et donc :

$$
\begin{array}{lcl}
Directeur(E \rightarrow T\ SE) & = & \{(, num\} \\
Directeur(SE \rightarrow add\ T\ SE) & = & \{add\} \\
Directeur(SE \rightarrow \varepsilon) & = & \{), \underline{\$}\} \\
Directeur(T \rightarrow F\ ST) & = & \{(, num\} \\
Directeur(ST \rightarrow mult\ F\ ST) & = & \{mult\} \\
Directeur(ST \rightarrow \varepsilon) & = & \{add,), \underline{\$}\} \\
Directeur(F \rightarrow (E)) & = & \{(\} \\
Directeur(F \rightarrow num) & = & \{num\}
\end{array}
$$

Il n'y a pas d'ambiguïté sur le choix des règles pour les non-terminaux SE, ST et F donc la grammaire est LL(1). En effet E et T ne posent pas de problème puisqu'il n'y a qu'une seule règle attachée à chacun de ces non-terminaux.

3 Résultats et Limitations

On étend les propriétés des grammaires aux langages. On dit donc qu'un langage est LL(1) si et seulement si il existe une grammaire LL(1) qui l'engendre. On sait décider si une grammaire est $LL(1)$ mais on ne sait pas décider si un langage est $LL(1)$. Il n'existe pas d'algorithme prenant en entrée une grammaire hors-contexte et calculant si il existe une grammaire LL(1) équivalente, c'est-à-dire engendrant le même langage. On peut néanmoins donner quelques critères :

1. Une grammaire ambiguë n'est pas $LL(1)$
2. Une grammaire récursive à gauche n'est pas $LL(1)$
3. Un langage régulier est toujours LL(1)

Certains langages sont intrinséquement ambigus, il existe donc des langages qui ne sont pas LL(1) (il y en a d'autres ...). Lorsqu'une grammaire est récursive à gauche il est inutile de se lancer dans des calculs. La méthode d'élimination de la récursivité à gauche dans les grammaires peut dans certains cas rendre la grammaire LL(1), mais pas toujours. En particulier si la grammaire est ambiguë elle le restera. La remarque 3 n'est pas très intéressante puisqu'on dispose déjà d'une méthode d'analyse déterministe pour les langages réguliers.