Grammaires et Analyse Syntaxique - Cours 6 Introduction à l'analyse ascendante

Ralf Treinen

INFORMATIQUE Sciences Université de Paris

INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

treinen@irif.fr

10 mars 2022

© Ralf Treinen 2020–2022

Analyse descendante

- Nous avons vu au cours 4 et 5 une méthode d'analyse *descendante* (angl. : *top-down*, ou dans le contexte de l'analyse grammaticale *recursive descent*) :
- L'analyse descendante construit l'arbre de dérivation commençant par la racine, et puis en ajoute dans l'arbre des enfants dans un ordre descendant et de gauche à droite.
- L'analyse descendante correspond à la construction d'une dérivation gauche.
- Cette dérivation gauche est construite dans l'ordre naturel : on commence avec l'axiome, et on termine sur le mot d'entrée.
- On a assez peu d'information pour guider la construction : Le non-terminal courant, et le symbole suivant de l'entrée.

Analyse descendante : avantages et inconvénients

- Avantage : facile à implémenter à la main quand la grammaire est LL(1).
- Inconvénient : peut nécessiter des contorsions de la grammaire quand la grammaire n'est pas LL(1).
- Des modifications de la grammaire (comme élimination de la récurrence gauche) bousculent la structure de l'arbre de dérivation.
- Un changement fondamental de la grammaire est gênant car on s'intéresse à la fin aussi à la structure trouvée par l'analyse (l'arbre de dérivation) qui maintenant ne correspond plus à la grammaire initiale.

Analyse ascendante

- L'alternative à l'analyse descendante est l'analyse *ascendante* (angl. : bottom-up) :
- L'analyse ascendante construit l'arbre de dérivation en commençant par les feuilles, et en faisant grandir l'arbre par combinant des arbres existants par des nouveaux nœuds.
- Ça correspond à quel type de dérivation ? Voir la suite !
- Nous avons maintenant beaucoup plus d'informations pour guider la construction de l'arbre : on a déjà des arbres de dérivations qu'il suffit de combiner.
- Pour cette raison l'analyse ascendante peut être plus puissante que l'analyse descendante.

Analyse descendante vs. analyse ascendante

- Exemple : la grammaire

$$E \rightarrow \texttt{(E+E)} \mid \texttt{(E*E)} \mid \texttt{i}$$

- Quand l'analyse *descendante* voit le symbole (, elle ne saura pas quelle production appliquer.
- Nous avons vu au cours 4 que cette grammaire n'est pas LL(1).
- Quand l'analyse *ascendante* voit que des parties de l'entrée sont reconnues comme étant des (, E, +, E,), elle sait que ça correspond à la première production, et elle va combiner ces 5 arbres par un nouveau nœud E.

Analyse ascendante

- Elle lit l'entrée de gauche à droite.
- Le principe est que des parties de l'entrée sont combinées en un morceau d'arbre de dérivation dès que possible.
- Structures de donnée de l'analyseur ascendant :
  - le reste de l'entrée qui reste à consommer
  - une pile de symboles de $\Sigma \cup N$. Elle contient la partie de l'entrée qu'on a déjà consommé.
- Toutes les réductions (applications d'une règle dans le sens ascendant) se font sur la partie haute de la pile.
- Un non-terminal sur la pile est en vérité la racine d'un morceau d'arbre de dérivation.

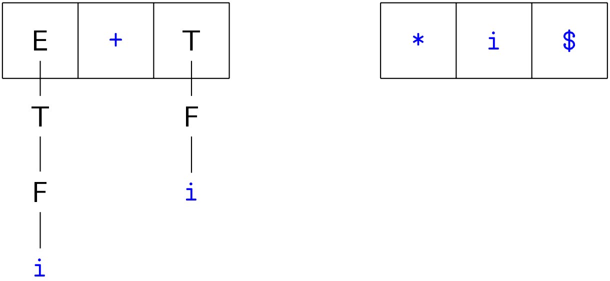
Pile et Entrée pendant l'analyse ascendante

Règles : $S \to E \ \$ \quad E \to E + T \mid T \quad T \to T * F \mid F \quad F \to (E) \mid i$

Partie de l'entrée déjà lue : `i+i`

Reste à lire : `*i$`

pile (sommet à droite) : entrée :



Actions dans l'analyse ascendante

- Il a quatre types d'action :
  - **shift** : transférer le symbole suivant de l'entrée sur le sommet de la pile ;
  - **reduce** $N \rightarrow \alpha$: remplacer une séquence α qui se trouve en haut de la pile par un N, quand N $\rightarrow \alpha$ est une règle de la grammaire ;
  - **accepter** ;
  - signaler une **erreur**.
- On parle aussi de *shift-reduce parser* car shift et reduce sont les actions essentielles.

Shift

Situation de départ :

pile (sommet à droite) :

x_1	x_2	x_3

entrée :

a_1	a_2	a_3	a_4

Situation après un **shift** :

pile (sommet à droite) :

x_1	x_2	x_3	a_1

entrée :

a_2	a_3	a_4

Reduce

Situation de départ :

pile (sommet à droite) :

x_1	x_2	y_1	y_2	y_3

entrée :

a_1	a_2	a_3

Situation après un **reduce** N $\rightarrow y_1 y_2 y_3$:

pile (sommet à droite) :

x_1	x_2	N

entrée :

a_1	a_2	a_3

Exemple

- La grammaire $G_1 = (\{\mathtt{i}, \mathtt{+}, \mathtt{*}, \mathtt{(}, \mathtt{)}, \mathtt{\$}\}, \{\mathsf{S}, \mathsf{E}, \mathsf{T}, \mathsf{F}\}, \mathsf{S}, P)$, où P est

$$
\begin{array}{rcl}
\mathsf{S} & \rightarrow & \mathsf{E}\ \mathtt{\$} \\
\mathsf{E} & \rightarrow & \mathsf{E}\ \mathtt{+}\ \mathsf{T} \mid \mathsf{T} \\
\mathsf{T} & \rightarrow & \mathsf{T}\ \mathtt{*}\ \mathsf{F} \mid \mathsf{F} \\
\mathsf{F} & \rightarrow & \mathtt{(}\ \mathsf{E}\ \mathtt{)} \mid \mathtt{i}
\end{array}
$$

- Cette grammaire engendre les expressions arithmétiques avec $+$ et $*$, en prenant en compte la priorité de $*$ sur $+$, et l'associativité à gauche de $+$ et de $*$.
- Nous avons vu au cours 5 que cette grammaire n'est pas LL(1), à cause de la récurrence gauche.

Exemple : actions du parseur

- Règles de la grammaire :

$$S \rightarrow E\ \$ \quad E \rightarrow E + T \mid T \quad T \rightarrow T * F \mid F \quad F \rightarrow (\ E\ ) \mid i$$

- Actions : s (shift), r (reduce), a (accept)

pile	*entrée*	*action*
	i+i*i$	*s*
i	+i*i$	*r* F → i
F	+i*i$	*r* T → F
T	+i*i$	*r* E → T
E	+i*i$	*s*
E+	i*i$	*s*
E+i	*i$	*r* F → i
E+F	*i$	*r* T → F

pile	*entrée*	*action*
E+T	*i$	*s*
E+T*	i$	*s*
E+T*i	$	*r* F → i
E+T*F	$	*r* T → T * F
E+T	$	*r* E → E + T
E	$	*s*
E$		*r* S → E$
S		*a*

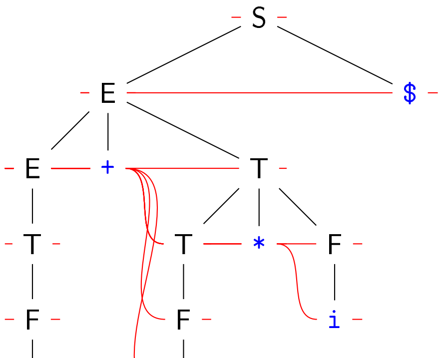
Exemple : quelle est la dérivation produite ?

- On regarde la suite des reductions sur la concaténation de pile et entrée :
  i+i*i$ – F+i*i$ – T+i*i$ – E+i*i$ – E+F*i$ – E+T*i$ – E+T*F$ – E+T $ – E$ – S
- Inverser l'ordre de cette séquence :
  S – E$ – E+T $ – E+T*F$ – E+T*i$ – E+F*i$ – E+i*i$ – T+i*i$ – F+i*i$ – i+i*i$
- C'est une dérivation droite !
- Donc, la parseur shift-reduce construit une dérivation droite dans un ordre inversé.
- Regardons cela maintenant avec construction de l'arbre de dérivation.

$S \rightarrow E\ \$ \quad E \rightarrow E + T \mid T \quad T \rightarrow T * F \mid F \quad F \rightarrow (\ E\ ) \mid i$

Entrée complète : i+i*i$

Situation initiale Après *Shift* Après *Reduce* $F \rightarrow i$ Après *Reduce* $T \rightarrow F$ Après *Reduce* $E \rightarrow T$ Après *Shift* Après *Shift* Après *Reduce* $F \rightarrow i$ Après *Reduce* $T \rightarrow F$ Après *Shift* Après *Shift* Après *Reduce* $F \rightarrow i$ Après *Reduce* $T \rightarrow T * F$ Après *Reduce* $E \rightarrow E + T$ Après *Shift* Après *Reduce* $S \rightarrow E\ \$$



Efficacité

- Pour appliquer une opération de *reduce* $N \rightarrow \alpha$ il faut chercher une occurrence de α.
- L'intérêt de la construction d'une dérivation droite construite à l'envers est qu'il suffit de chercher α dans la pile.
- En plus, si on réussit à éviter des *shift* prématurés, il suffit de regarder seulement le haut de la pile !

Prendre fausse route

- Attention le parseur peut a priori aussi prendre une fausse route.
- Dans une situation où il y a en haut de la pile α et N $\rightarrow \alpha$ est une règle :
  - On peut faire un **reduce** $N \rightarrow \alpha$, ou un **shift**
  - Quand il y une autre règle $M \rightarrow \beta$, et β est également en haut de la pile (c'est possible quand α est un suffixe de β ou l'inverse) :
    on peut faire **reduce** $N \rightarrow \alpha$ ou **reduce** $M \rightarrow \beta$
- Comment éviter que notre parseur prenne fausse route ?

Deux Reduce possibles

Pile de départ :

x_1	x_2	y_1	y_2	y_3

1. Soit **reduce** N $\rightarrow y_1 y_2 y_3$:

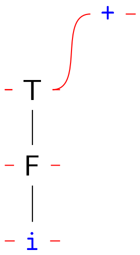
x_1	x_2	N

2. Soit **reduce** M $\rightarrow y_2 y_3$:

x_1	x_2	y_1	M

Exemple 1 : un shift prématuré

$$S \to E\ \$ \quad E \to E + T \mid T \quad T \to T * F \mid F \quad F \to (\ E\ ) \mid i$$

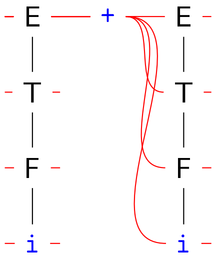


Le problème dans l'exemple 1

- Dans l'exemple 1 on a obtenu en haut de la pile : T +
- Même si ce qui suit dans l'entrée se réduit vers T on aura en haut de la pile : T + T
- On a bien $S \rightarrow E+T \rightarrow T+T$
- Mais ce n'est **pas** une dérivation droite !
- On fait on n'a **pas** que $S \xrightarrow{d}{}^{*} T+T$
- On aurait du réduire le premier T à E avant de shifter le +

Exemple 2 : un reduce prématuré

$$S \to E\ \$ \quad E \to E + T \mid T \quad T \to T * F \mid F \quad F \to (\ E\ ) \mid i$$

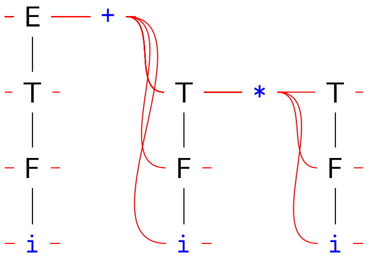


Le problème dans l'exemple 2

- Dans l'exemple 2 on a obtenu en haut de la pile : E + E
- Or, E+ se réduit seulement si on a après un T.
- Peut importe le mot $w \in \Sigma^*$ qu'on trouve après, on ne peut pas avoir que E$w \rightarrow^*$ T.
- On n'aurait pas du réduire le deuxième T à E mais shifter le $*$.

Exemple 3 : mauvais choix de la règle dans une réduction

$$S \to E\ \texttt{\$} \qquad E \to E\ \texttt{+}\ T \mid T \qquad T \to T\ \texttt{*}\ F \mid F \qquad F \to \texttt{(}\ E\ \texttt{)} \mid \texttt{i}$$



Le problème dans l'exemple 3

- Dans l'exemple 3 on a obtenu en haut de la pile : T $*$ T
- Or, T$*$ se réduit seulement si on a après un F.
- Peut importe le mot $w \in \Sigma^*$ qu'on trouve après, on ne peut pas avoir que T$w \to^*$ F.
- On n'aurait pas du réduire par la règle T $\to$ F mais réduire par la règle T $\to$ T$*$F.

Éviter la mauvaise route

- Nous avons vu que notre analyseur ascendant peut prendre mauvaise route.
- Nous allons maintenant caractériser ce que ça veut dire mauvaise route.
- Une fois qu'on a trouvé et compris cette caractérisation on cherche un moyen pour détecter efficacement le cas de mauvaise route (la semaine prochaine).
- Quand on sait comment détecter les mauvais cas on verra aussi comment les éviter dans l'analyse ascendante.

Comment caractériser les cas de mauvaise route

Définition

$\alpha \in (\Sigma \cup N)^*$ est *préfixe réductible* ssi

- il y a une décomposition $\alpha = \alpha_1\alpha_2$
- et une règle $\mathrm{N} \rightarrow \alpha_2$
- et un mot $w \in \Sigma^*$ tel que
- $S \xrightarrow{d}^* \alpha_1 \mathrm{N} w \xrightarrow{d} \alpha_1\alpha_2 w = \alpha w$

C'est à dire

- Si α est un préfixe réductible, il y encore de l'espoir : on peut encore avoir une continuation w de l'entrée tel que αw fait partie d'une dérivation droite. w peut être vide.
- Mais ce n'est pas un préfixe quelconque : α se termine précisément avec le résultat de la dernière application de règle.

Comment caractériser les cas de mauvaise route

- Cette notion de préfixe réductible nous sert à distinguer les bonnes et mauvaises routes :
  - Si le contenu de la pile est un préfixe réductible alors on peut faire une opération **reduce** sans de naviguer dans une impasse.
  - Si le contenu de la pile n'est pas un préfixe réductible alors il faut qu'on puisse atteindre par des opérations **shift** un préfixe réductible.
- Regardons cette notion de préfixe réductible sur les exemples que nous avons vu.

L'exemple du mauvais choix de règle

$$S \to E\ \$ \quad E \to E + T \mid T \quad T \to T * F \mid F \quad F \to (\ E\ ) \mid i$$

- On avait obtenu la pile : E+T*T
- Or il n'y aucun mot $w \in \Sigma^*$ tel que $S \xrightarrow{d}^* \text{E+T*T}w$
- Comment s'en convaincre ? On peut essayer toutes les possibilités, ou on utilise la méthode que nous verrons la semaine prochaine.
- On aurait pas du réduire le F à T mais plutôt T*F à T car E+T est un préfixe réductible.

Exemples de mots qui sont des préfixes réductibles

- Rappel de la grammaire :

  $$\mathrm{S} \to \mathrm{E}\ \$ \quad \mathrm{E} \to \mathrm{E} + \mathrm{T} \mid \mathrm{T} \quad \mathrm{T} \to \mathrm{T} * \mathrm{F} \mid \mathrm{F} \quad \mathrm{F} \to (\ \mathrm{E}\ ) \mid \mathtt{i}$$

- E+T, car $S \xrightarrow{d} \mathrm{E}\$ \xrightarrow{d} \mathrm{E+T}\$$.
  Donc, $w = \$$.
- E+i, car $S \xrightarrow{d} \mathrm{E}\$ \xrightarrow{d} \mathrm{E+T}\$ \xrightarrow{d} \mathrm{E+F}\$ \xrightarrow{d} \mathrm{E+}\mathtt{i}\$$.
  Donc, $w = \$$.
- (T, car $S \xrightarrow{d} \mathrm{E}\$ \xrightarrow{d} \mathrm{T}\$ \xrightarrow{d} \mathrm{F}\$ \xrightarrow{d} (\mathrm{E})\$ \xrightarrow{d} (\mathrm{T})\$$.
  Donc, $w =)\$$.
- E\$, car $S \xrightarrow{d} \mathrm{E}\$$.
  Donc $w = \epsilon$.

Préfixes réductibles et préfixes viables

- Attention quand vous regardez des autres cours/livres :
- Souvent l'explication de l'analyse shift-reduce est basée sur une autre notation, celle d'un *préfixe viable*.
- La différence entre les deux notions est qu'un préfixe viable est un préfixe quelconque d'un préfixe réductible.
- Dans ce cours nous utilisons la notion d'un *préfixe réductible* qui est plus simple.

La grande question qui reste

- Comment savoir si un α est un préfixe réductible ou pas ?
- Ça semble compliqué : ne faut-il pas trouver la bonne continuation w et la complétion de l'arbre de dérivation pour αw ? N'est ce pas encore plus difficile que notre problème de départ ?
- La grande surprise est : non ! On peut décider si α est un préfixe réductible en temps **linéaire** dans la longueur de α.
- Les automates du L2 nous viennent à la rescousse : on verra la semaine prochaine la construction d'un automate qui reconnaît les préfixes réductibles.

Comment est-ce possible ?

- Regardons un exemple très simple.
- Grammaire $G_2 = (\{a, b\}, \{S\}, S, \{S \to aSb \mid \epsilon\})$
- $\mathcal{L}(G_2) = \{a^n b^n \mid n \geq 0\}$: pas reconnaissable.
- $L' = \{\alpha \in \{a, b, S\}^* \mid S \xrightarrow{d}{}^* \alpha\} =$
  $\{a^n b^n \mid n \geq 0\} \cup \{a^n S b^n \mid n \geq 0\}$: pas reconnaissable.
- Les préfixes de L' : $\{a^n b^m \mid m \leq n\} \cup \{a^n S b^m \mid m \leq n\}$: toujours pas reconnaissable.
- Comment est il possible que le langage des préfixes réductibles est reconnaissable ?

Comment est-ce possible ?

- Il faut se rappeler ce que c'est un préfixe *réductible* !
- Un préfixe réductible est une séquence dans L' *qui se termine sur le résultat d'une application d'une règle*.
- Regardons une dérivation, les préfixes réductibles soulignés :

$$S \to \underline{aSb} \to \underline{aaSb}b \to \underline{aaaSb}bb \to \underline{aaaaSb}bbb \to \underline{aaaa}bbbb$$

- Donc, l'ensemble des préfixes réductibles est

$$\{a^n Sb \mid n \geq 1\} \cup \{a^n \mid n \geq 0\}$$

et cet ensemble est reconnaissable ! L'expression rationnelle est $a^*(aSb \mid \epsilon)$