Analyse syntaxique ascendante (par décalage/réduction)

- On construit l'arbre d'analyse pour ω en commençant par les feuilles et en remontant vers la racine (ordre GDR)
- Analyseur par décalage/réduction :
  on construit la dérivation droite inverse

  Principe :

  on lit la phrase de gauche à droite

  en recherchant une chaîne α correspondant à une partie droite de règle $A \rightarrow \alpha$ (le « *manche* »)

  et on « *réduit* » α en A

  ceci jusqu'à ce qu'on arrive à S

Manches

- Si $\gamma = \alpha\beta\omega$ est une protophrase droite
  obtenue par une dérivation droite $S \Rightarrow_d^* \alpha A\omega \Rightarrow_d \alpha\beta\omega$
  alors, on dit que $A \rightarrow \beta$ est un manche de $\gamma = \alpha\beta\omega$
- Abus de langage : on dit souvent que β est un manche pour $\alpha\beta\omega$ si la production est claire dans le contexte
- Rem : si G n'est pas ambiguë, il existe une seule dérivation droite pour γ et donc un et un seul manche

exemple

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

Ex. de dérivation droite :
$S \Rightarrow_d aABe \Rightarrow_d aAde \Rightarrow_d aAbcde \Rightarrow_d abbcde$

Protophrase droite	Manche
abbcde	$A \rightarrow b$
aAbcde	$A \rightarrow b$ n'est pas un manche car $S \neq>_d^* aAAcde \Rightarrow_d aAbcde$
aAbcde	$A \rightarrow Abc$ car $S \Rightarrow_d^* aAde \Rightarrow_d aAbcde$
aAde	$B \rightarrow d$
aABe	$S \rightarrow aABe$

« Réduction » du manche

On peut ainsi obtenir une dérivation droite inverse par réduction du manche :

Si $S \Rightarrow_d \gamma_1 \Rightarrow_d \dots \Rightarrow_d \gamma_{n-1} \Rightarrow_d \gamma_n = \omega$

alors on peut reconstruire la dérivation droite inverse à partir de ω :

- On repère le manche $A_n \rightarrow \beta_n$ dans γ_n et on le réduit. On obtient γ_{n-1}
- Etc.

Jusqu'à ce qu'on arrive à S

$S \overset{S \rightarrow aABe}{\Rightarrow_d} aABe \overset{B \rightarrow d}{\Rightarrow_d} aAde \overset{A \rightarrow Abc}{\Rightarrow_d} aAbcde \overset{A \rightarrow b}{\Rightarrow_d} abbcde$

- Difficulté : repérer le manche

Implémentation à l'aide d'une pile

On utilise :

- Un tampon « entrée » : partie non encore traitée du texte
- Une pile : partie déjà traitée (avec réductions effectuées)

Au départ : pile = \$ entrée = ω \$

Actions possibles :

- On n'a pas encore repéré le manche, on « décale » un symbole de l'entrée vers la pile
- Le manche est en sommet de pile, on « réduit » le manche
- On a réduit toute l'entrée en S, on « accepte » la phrase
- On a détecté une « erreur »

- Rem : les *symboles de la pile* suivis des *symboles non encore traités* de l'entrée forment *une protophrase droite* γ_i.
  L'analyseur décale les symboles de l'entrée jusqu'à ce que le manche soit en tête de pile.
  Il effectue alors une réduction qui mène à la protophrase précédente γ_{i-1}.
- Problème : être capable de repérer le manche, afin de décider quand il faut « décaler » et quand il faut « réduire »

95

Préfixes viables

- Un préfixe viable est un préfixe d'une protophrase droite qui ne s'étend pas au delà du manche

⇒ Ce sont les préfixes qui peuvent apparaître sur la pile d'un analyseur par décalage/réduction

- Le problème devient : comment choisir les actions (décaler, réduire) de façon à ce que la pile contienne toujours un préfixe viable ?
- Une réponse : les analyseurs LR

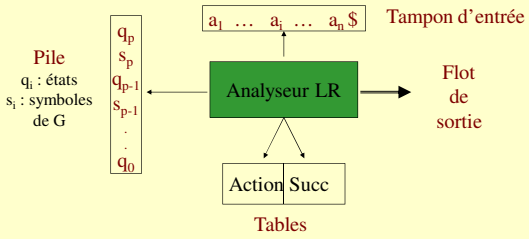
96

Analyseurs LR

- Analyse LR(k) : méthode d'analyse par décal/réduct
  - Left : parcours de l'entrée de gauche à droite
  - Right : construction d'une dérivation droite inverse
  - k : nombre de symboles de LA
  - LR signifie LR(1)
- Intérêt :
  - Méthode très générale et très efficace (déterministe)
  - Peut reconnaître toute construction de langage de prgmt°
  - Grammaires LR sont un sur-ensemble strict des gramm. LL
  - Erreurs de syntaxe détectées très tôt lors de la lecture du prg
- Inconvénient : construction « à la main » très lourde
  Mais, des outils logiciels construisent automatiquement des analyseurs LR à partir de GNC (ex : Yacc)

97

Modèle d'analyseur LR



- L'état courant q_p et le symbole courant a_i déterminent l'action à effectuer (table Action)
- Action[q_p, a_i]=décaler(q) ou réduire(A→α) ou accepter ou erreur
- Succ[q,A] = q'

98

idée

On détermine les préfixes viables par un AFD

- Tous les états sont acceptants pour les préfixes viables
- L'analyse commence en q_0
- L'action à effectuer est déterminée par
  - l'état courant q_p
  - et le symbole courant a_i de ω
  - Décaler(q_l) s'il y a une transition $q_p - a_i \rightarrow q_l$
  - Réduire(A→α) si q_p accepte le préfixe γα et γA est un préfixe viable (≈ pour SLR)
    ($\gamma A a_i$ est préfixe d'une protophrase droite ≈ pour LR)
- L'analyseur fonctionne à partir de 2 tables Action et Succ qui reprennent les informations de l'AFD

99

Algo général d'analyse LR

- Données : - une chaîne à analyser ω
  - les tables Action et Succ
- État initial : - la pile contient q_0
  - l'entrée est ω$
  - le symbole courant $a_i = \omega(1)$
- À chaque étape (si q_j est le sommet de pile et a_i le symbole courant) :
  - Si Action[q_j, a_i] = décaler(q_k) alors - empiler a_i et q_k
    - lire le symbole suivant a_{i+1}
  - Si Action[q_j, a_i] = réduire($A \rightarrow \alpha_1\alpha_2\dots\alpha_p$) alors
    - Dépiler 2p symboles (p états et p symboles de G)
    - Empiler A et Succ[q_{j-p},A] où q_{j-p} est le nouveau sommet de pile
    + émettre en sortie ($A \rightarrow \alpha_1\alpha_2\dots\alpha_p$)
  - Si Action[q_j, a_i] =accept alors FIN
  - Sinon erreur (procédure de récupération sur erreur)

100

Tous les analyseurs LR fonctionnent selon ce principe.

Mais il existe plusieurs méthodes, + ou – « fines », pour construire les tables Action et Succ

- SLR (Simple LR) : la + simple, et la – fine
  - Facile à mettre en œuvre
  - Fonctionne avec moins de grammaires que les autres
- LR canonique : + complexe, et + fine
  - Plus lourde à mettre en œuvre
  - Fonctionne avec beaucoup de grammaires NC
- LALR (Look Ahead LR) :
  - Coût et puissance intermédiaires

Construction des tables d'analyse SLR

- L'analyse SLR (Simple LR) est la plus facile à implémenter mais la moins puissante des méthodes LR
- Les grammaires SLR sont un sous-ensemble strict des grammaires LR
- Démarche de construction d'un analyseur pour une grammaire G:
  - Construire un AFD qui reconnaît les préfixes viables des protophrases de G (construction non étudiée ici)
  - Construire les tables Action et Succ

Propriétés de l'AFD construit

- Il reconnaît tous les préfixes viables
- Quand transition(I_i, a)=I_j avec a ∈ V_T, l'action correspondante Action[I_i, a] = décaler(I_j)
- Quand une action réduire(A→α) est définie, les symboles de prévision possibles sont tous les a ∈ suivant(A)
- L'action accepter est utilisée seulement si le symbole de prévision est $
- Si plusieurs actions sont possibles en un état pour un même symbole de prévision, un conflit se produit
- S'il n'y a pas de conflit, la grammaire est SLR(1)

Traduction dirigée par la syntaxe

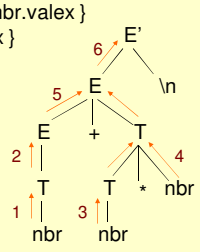
(le retour)

- En pratique, on fait souvent des calculs sémantiques en même temps que l'analyse syntaxique
- On a vu que l'utilisation d'attributs synthétisés permet de faire l'évaluation sémantique au fur et à mesure de l'analyse LR
  (par contre, toute dépendance père → fils est proscrite)

exemple

E' → E \n { écrire E.val }
E → E + T { E.val := E_1.val + T.val }
E → T { E.val := T.val }
T → T * nbr { T.val := T_1.val * nbr.valex }
T → nbr { T.val := nbr.valex }

Ici, tous les attributs sont synthétisés. On peut les calculer lors de l'analyse LR (correspond à un parcours GDR)



- Modification de l'analyseur LR pour effectuer la traduction dirigée par la syntaxe :
  lors des réductions :
  - On évalue les règles sémantiques associées
  - On empile les attributs du NT avec le symb. NT lui-même

- Ex : Analyser la phrase 5+3*4 en effectuant la traduction. On suppose que nbr.valex est la valeur (entière) du nombre.