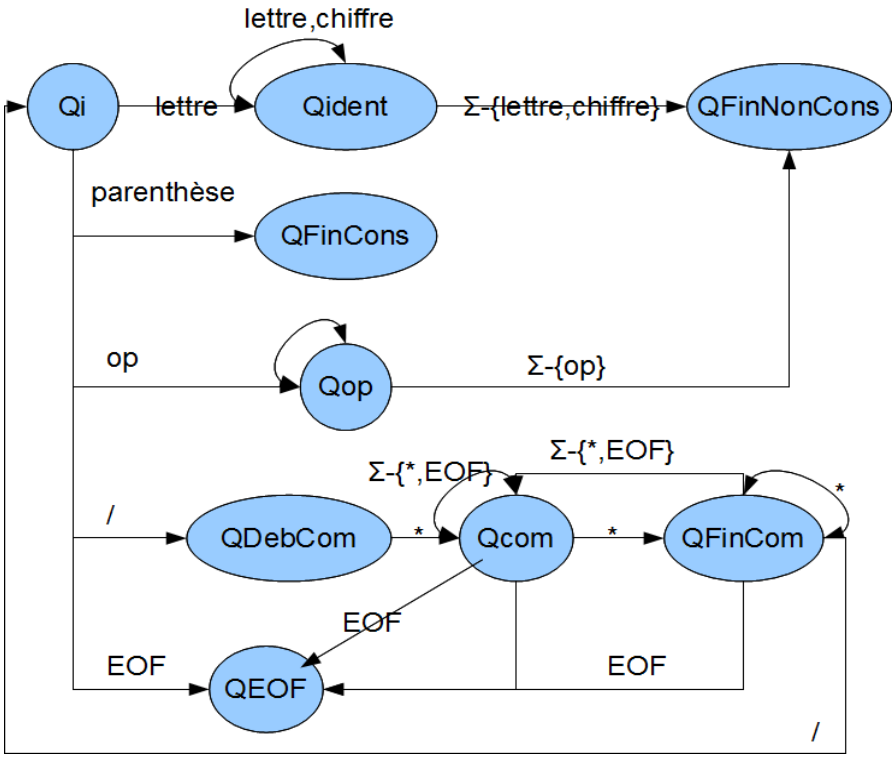
Compte Rendu - Chapitre 1
Analyse lexicale syntaxique

Graignic Guillaume
Olivier Cadoret

I/ Le diagramme de transition implémentant l'analyse lexicale du langage proposé (Grammaire 1,1)



II/ Explications conceptuelles sur les structures de données, les algorithmes et le découpage analyse lexicale/cribles

II-1/Structures de données

→ On utilise les types énumérés de Caml pour codés les états de l'automate, les unités lexicales et les caractères.
→ Une transition de l'automate s'effectue en fonction de l'état et du caractère courant du fichier lu
→ Le crible est séparé de l'analyseur lexicale.
→ Un crible est un lexeme et un état et renvoi une unité lexicale
→ Un automate se compose de 4 état finaux et une transitions
→ Une grammaire contient un automate associé à un crible

II-3/ Analyse lexicale

→ Dans notre cas, on distingue 3 états finaux dans l'automate:
- état final sans consommation du dernier caractère lu
- état final avec consommation du dernier caractère lu
- état final de "End of file"

→ La transition 'trans' effectue l'analyse lexicale du diagramme de transition ci dessus.

II-4/ Cribles

→ On a donc un lexème reconnu par l'automate, que l'on associe ensuite à une unité lexicale avec le crible. Un crible prend un lexème et un état et renvoi une unité lexicale

→ Dans notre cas, le crible est réalisé par mon_crible qui capte aussi les erreurs.

II-5/ Fonctions

→ La fonction 'parcours_automate' lit un fichier en simulant l'automate :

Elle prend en paramètre un in_channel (contenant un fichier et l'endroit ou l'on est arrivé dans le fichier)
un automate
un etat
un string (auquel on concatènera éventuellement le char courant de in_channel)

Elle lit le fichier caractère par caractère et renvoi un couple (etat,lexeme).

→ La fonction get_token :

Elle prend, parmi ses arguments, une description de l'automate
Elle lit le fichier à analyser caractère par caractère
Elle retourne un couple (unité_lexicale, lèxeme

→ La fonction scanner retourne la liste de tous les couples (unite_lexicale, lexème) rendu par la fonction get_token

V/ Réponses aux questions de compréhension

1.1) L'intérêt d'avoir un crible séparé est de simplifier l'automate. En effet, on ne distingue que 3 états finaux dans l'automate:

- état final sans consommation du dernier caractère lu
- état final avec consommation du dernier caractère lu
- état final de "End of file"

On ne distingue les différentes unités lexicales qu'après le parcours de l'automate, à l'aide du crible.

Ainsi, on a un lexème reconnu par l'automate, que l'on associe ensuite à une unité lexicale avec le crible. Ceci permet d'avoir un code plus facilement maintenable et adaptable

1.2) Il est très pratique d'utiliser les types énumérés de CAML. Ils sont simples, et très explicites :

- il est facile d'effectuer un filtrage ("match...with")
- le code gagne en lisibilité (les types énumérés ont des noms très explicites)

1.3) Avec un scanner incrémental, s'il y a une erreur d'analyse, on sait exactement où elle se situe (on aura un résultat au moins jusqu'au crash).
Avec un scanner autre, on n'a se sait pas quand se produit l'erreur, et il est alors beaucoup plus dur de le debugger.

1.4) On considère ident comme étant terminal car son analyse tient de l'analyse lexicale, et non syntaxique. En effet, un ident est un lexème.

1.5) Les seules modifications à apportées sont la création de nouvelles unités lexicales (UL_SI, UL_ALORS, UL_SINON...) et de les introduire dans le crible.
En effet, "si", "alors", "sinon", "fsi", ">=" et "<=" seront reconnus par l'automate qui aboutira sur l'état final sans consommation du dernier caractère.