

Développement d'un analyseur syntaxique - 2

Le module `earley.c`

L'objectif de ce TP est de programmer l'analyseur d'Earley, en particulier la structure d'article, la structure des ensembles d'articles, et les trois opérations de création d'articles (prédiction, complétion et scan). L'implémentation de l'algorithme d'analyse nécessite la représentation de la phrase à analyser. On trouvera dans les deux fichiers `phrase.h` et `phrase.c` une structure de donnée permettant de représenter une phrase, ainsi qu'une fonction de lecture d'une phrase sur l'entrée standard. Ces deux fichiers se trouvent à l'emplacement suivant : `/home/anars/FIL/TP2`
Note : pour réaliser ce TP il est impératif d'avoir parfaitement compris l'algorithme d'Earley.

1 Représentation des articles en mémoire

Le principe de stockage des articles en mémoire est proche de celui de la grammaire, vu au TP 1 : tout article est associé à un entier, appelé code de l'article, qui sert d'indice dans différents tableaux dans lesquels sont stockés les différents éléments des articles : règle, position du point, indice de début et indice de fin.

L'ensemble des articles est matérialisé par la structure `articles` qui permet de représenter `MAX_ARTICLES` articles. De plus, les articles sont regroupés en ensembles ordonnés. Ces ensembles sont représentés sous la forme de liste chaînées : la tête de la liste de l'ensemble `i` est accessible depuis la case `i` du tableau `debut_ens`. Le code de l'article suivant l'article `a` dans le même ensemble est stocké dans la case `a` du tableau `suivant`. Enfin, la queue de la liste de l'ensemble `i` est accessible depuis la case `i` du tableau `fin_ens`.

Voici le détail de la structure `articles` :

```
typedef struct{
    int regle[MAX_ARTICLES] ;      int premier_libre ;
    int pos[MAX_ARTICLES] ;        int debut_ens[MAX_LONG_PHRASE] ;
    int debut[MAX_ARTICLES] ;      int fin_ens[MAX_LONG_PHRASE] ;
    int fin[MAX_ARTICLES] ;        int succes ;
    int suivant[MAX_ARTICLES] ;    } articles ;
```

- `int regle[MAX_ARTICLES]` ; associe à tout article le code de la règle correspondante.
- `int pos[MAX_ARTICLES]` ; associe à tout article la position du point.
- `int debut[MAX_ARTICLES]` ; associe à tout article l'indice de début.
- `int fin[MAX_ARTICLES]` ; associe à tout article l'indice de fin.
- `int suivant[MAX_ARTICLES]` ; associe à tout article le code de l'article suivant dans le même ensemble.
- `int premier_libre` ; indique l'indice de la première case libre dans les différents tableaux.
- `int debut_ens` ; permet d'accéder au code du premier article de chaque ensemble d'articles.
- `int fin_ens` ; permet d'accéder au code du dernier article de chaque ensemble d'articles.
- `int succes` ; à l'issue de l'analyse d'une phrase, la valeur 1 est affectée à l'entier `succes` si l'analyse est réussie, sinon, il est égal à 0.

2 Fonctions à réaliser

- `int initialise_articles(articles *A, grammaire *G)`
initialise les champs de la structure pointée par `A` qui ont besoin de l'être.

- void ajoute_article(int a, articles *A, grammaire *G, phrase *P)
le principe d'ajout d'un article est le suivant. L'article à ajouter, a est mis à la position premier_libre. S'il existe déjà un article égal à a dans l'ensemble ordonné d'articles correspondant à a, alors on ne fait rien, sinon, on incrémente premier_libre et l'article est ajouté à l'ensemble général des articles. Il ne reste alors plus qu'à l'ajouter à la fin de l'ensemble ordonné d'articles correspondant à a. La fonction vérifiera que le nombre maximal d'articles n'a pas été atteint.
- int article_present(int a, articles *A)
retourne 1 si l'article a est déjà présent dans l'ensemble des articles, 0 sinon.
- int prediction(int a, articles *A, grammaire *G, phrase *P)
réalise l'opération de prédiction sur l'article a.
- int scanning(int a, articles *A, grammaire *G, phrase *P)
réalise l'opération de scanning sur l'article a.
- int completion(int a, articles *A, grammaire *G, phrase *P)
réalise l'opération de complétion sur l'article a.
- void analyse(phrase *P, articles *A, grammaire *G)
réalise l'analyse de la phrase P à l'aide de la grammaire G. Si l'analyse réussit, le champ succes de A est positionné à 1.

3 Ajout du champ intervalle aux articles

De la même manière qu'est associé à un article a un intervalle de la phrase à analyser (materialisé par les deux champs debut et fin), on peut associer à tout symbole X de la partie droite de la règle de a un intervalle, représentant la partie de la phrase pouvant être dérivée à partir de X.

Un article a, complet, peut maintenant être représenté de la manière suivante :

W[0..10] -> X[0..3] Y[3..5] Z[5..10]

qui explicite l'intervalle associé à tout symbole de la règle de l'article.

Pour prendre en compte cette information nouvelle, on ajoute à la définition de la structure articles le tableau bidimensionnel suivant :

```
int intervalle[MAX_ARTICLES][MAX_PARTIE_DROITE];
```

Nous aurons ainsi, pour l'article précédent a :

```
debut[a] = 0          fin[a] = 10
intervalle[a][0] = 3  intervalle[a][1] = 5  intervalle[a][2] = 10
```

La prise en compte de l'intervalle des symboles de la partie droite nécessite d'effectuer un certain nombre de modifications dans les fonctions déjà programmées. En particulier :

- int article_present(int a, articles *A)
doit maintenant prendre en compte, lors de la comparaison de deux articles, les valeurs des intervalles des symboles de la partie droite.
- int scanning(int a, articles *A, grammaire *G, phrase *P) et
int completion(int a, articles *A, grammaire *G, phrase *P)
doivent, lors de la création de nouveaux articles à partir de a, recopier les valeurs des intervalles de a.

QUELQUES RAPPELS

Etant donné une grammaire $G = \langle \Sigma, N, P, S \rangle$ et une phrase $a_1..a_n$, on appelle article valide un triplet $\langle i, A \rightarrow \alpha \cdot \beta, j \rangle$ tel que :

- $S \xRightarrow{*} \gamma A \delta$
- $\gamma \xRightarrow{*} a_1 \dots a_i$
- $\alpha \xRightarrow{*} a_{i+1} \dots a_j$

On définit les trois opérations suivantes :

<i>scanning</i>	<i>prédiction</i>	<i>completion</i>
		$i \leq k \leq j$
$\frac{\langle i, A \rightarrow \alpha \cdot a_j \beta, j-1 \rangle}{\langle i, A \rightarrow \alpha a_j \cdot \beta, j \rangle}$	$\frac{\text{si } B \rightarrow \gamma \in P \quad \langle i, A \rightarrow \alpha \cdot B, j \rangle}{\langle j, B \rightarrow \cdot \gamma, j \rangle}$	$\frac{\langle i, A \rightarrow \alpha \cdot B \beta, k \rangle \quad \langle k, B \rightarrow \gamma \cdot, j \rangle}{\langle i, A \rightarrow \alpha B \cdot \beta, j \rangle}$