TD COMPILATION — FEUILLE 4

Gestion des noms et vérification de types.

On s'intéresse à la vérification contextuelle du langage LX décrit par la grammaire suivante :

prog	$\rightarrow$	liste_decl **begin** liste_inst **end**	liste_inst	$\rightarrow$	inst **;** liste_inst | ϵ
liste_decl	$\rightarrow$	decl **;** liste_decl | ϵ	inst	$\rightarrow$	place **:=** exp
decl	$\rightarrow$	**idf :** type	place	$\rightarrow$	**idf** | place [exp]
type	$\rightarrow$	**int** | **array** [**cst_ent**] **of** type	exp	$\rightarrow$	**cst_ent** | place

Les contraintes sont les suivantes :
- Tout identificateur apparaissant dans les instructions doit avoir été déclaré une et une seule fois.
- L'opérateur d'indexation [] ne s'applique que sur des objets de type tableau. La valeur d'indexation ne peut être qu'une valeur de type entier. Comme dans le lanvage C, cette valeur représente le nombre d'éléments des tableaux, ceux-ci étant toujours indicés à partir de 0.
- Dans une affectation le type de la partie droite et le type de la partie gauche de l'affectation doivent être structurellement équivalents.

Exercice 1. Représentation des types du langage LX

On définit le domaine d'attribut `exp_type` qui permet de représenter les types du langage LX :

exp_type $\rightarrow$ **type_int** | **type_tab**(num, exp_type)

num et exp_type désignent le nombre et le type des éléments du tableau. On notera respectivement `est_int`, `est_tab`, `nb_elem` et `type_elem` les fonctions permettant de connaître la nature d'un type (`type_int` ou `type_tab`), le nombre d'éléments d'un tableau et le type de ses éléments.

▷ **Question 1** *Écrire la fonction* `identique(t1,t2 :exp_type)` *qui vérifie si les expressions de type t1 et t2 sont compatibles pour l'affectation. Expliquer comment peut être codée l'affectation de tableaux, sous cette hypothèse.*

▷ **Question 2** *Écrire une fonction* `taille(t :exp_type)` *qui calcule la taille nécessaire à la représentation d'une valeur de type t. On supposera que les entiers et les adresses sont codés sur un mot. Cette question prépare la phase de génération de code.*

Exercice 2. Construction de l'environnement et vérification de type.

On définit maintenant le domaine d'attribut `env` qui permet d'associer à un nom d'identificateur son expression de type. Les environnements sont représentés par des fonctions partielles de la forme :

$$\text{env} : \text{Nom} \rightarrow \text{exp\_type}$$

On pourra utiliser les notations suivantes :
- $dom(e)$ le domaine de la fonction e
- $e(n)$ la valeur de la fonction e au point n. Cette expression n'est définie que si $n \in dom(e)$
- $e1 \oplus e2$ l'union disjointe des environnements $e1$ et $e2$ définie par :
  $e1 \oplus e2 = e1 \cup e2$ si $dom(e1) \cap dom(e2) = \emptyset$ undefined sinon

▷ **Question 1** *Donner la représentation du type associé au nom x par la déclaration :*

```
x :array [10] of array [5] of int
```

▷ **Question 2** *Ajouter sur la grammaire des attributs permettant de construire l'environnement et de vérifier les contraintes de typage.*

Exercice 3. On ajoute les types article avec les règles suivantes :

type	→	**record** liste_champ **end**
liste_champ	→	champ **;** liste_champ | ϵ
champ	→	**idf :** type
place	→	place **. idf**

Dans un type article les noms de champ doivent être disjoints deux à deux.

L'équivalence entre types article impose les mêmes noms de champs, de mêmes types, puisque les noms de champs définissent les opérations applicables sur les articles (**. idf**). On peut adopter les définitions suivantes :

1. deux types articles sont structurellement équivalents s'ils possèdent les mêmes noms de champ de types équivalents.
2. deux types articles sont structurellement équivalents s'ils possèdent les mêmes noms de champ, de types équivalents, et apparaissant dans le même ordre.

▷ **Question 3** *Discuter l'impact du choix d'équivalence structurelle sur le codage de l'affectation entre articles. Proposer, pour les deux cas, une extension du domaine d'attributs `exp_type` pour représenter les types article.*

▷ **Question 4** *Étendre le calcul d'attributs pour traiter le type article (déclaration et opération de sélection).*