Polytech Paris-Saclay – ET4 Info **2020-2021**

TD – Grammaire et Analyse SLR (suite)

On considère la sous-grammaire des expressions pour un langage fonctionnel inspiré de OCAML. Une grammaire ambiguë des expressions est la suivante:

```
E ::= E + E  | Id | Cste | ( E ) | E , E | E E
```

L'opérateur `'+'` est l'addition sur les entiers. L'opérateur `','` construit des n-uplets (n ≥ 1) comme dans le triplet `x+3, x*y, 5`. Les composants des n-uplets peuvent être de types différents. La règle `E E` correspond à l'application de fonctions. Le premier `E` doit correspondre à une expression dont l'évaluation donne une fonction (soit directement un identificateur de fonction, soit le résultat d'un calcul qui renvoie une fonction) , le second `E` à l'argument de cette fonction. Une fonction prend un unique argument (qui peut être un n-uplet) et donne un résultat qui peut être une fonction à qui l'on fournira un argument ultérieurement. Dans l'exemple ci-dessous on définit de deux manières différentes l'addition de deux nombres. Dans la première version (`somme`) on prend les deux nombres en une fois sous la forme d'un couple (n-uplet à deux éléments) ; dans la seconde version (`add`) les arguments sont passés successivement :

```
let somme = function (x, y) -> x + y;;
let add = function x -> function y -> x + y;;
let a12 = add 12;;
```

On peut écrire une expression comme `add 3 5`, ou avec des parenthèses explicites `(add 3) 5`, qui donnera donc `8`. L'expression `add 12` est aussi correcte et son évaluation renvoie en résultat l'équivalent d'une fonction anonyme à un paramètre `y` qui calcule `12+y`. Dans l'exemple on a mis cette valeur dans la variable `a12` et on pourra ensuite écrire `a12 3` qui s'évaluera donc en `15`. La fonction `somme` prend en argument un couple d'entiers : un appel correct est par exemple `somme(3, 5)` le couple argument de `somme` étant construit avec l'opérateur `','` .

On suppose que l'application de fonction a la précédence la plus élevée, suivie de `'+'`, puis `','`. Tous les opérateurs sont associatifs à gauche.

1. Les expressions `somme 3, 5` et `add 3, 5` et `add(3, 5)` sont-elles syntaxiquement correctes ? Sont-elle correctes du point de vue typage (e.g. on n'additionne pas des fonctions, et un entier ne se retrouve pas dans une position d'appel de fonction) ?

2. Donnez une grammaire **non-ambiguë** pour les expressions du langage qui reflète les précédences et associativités indiquées.

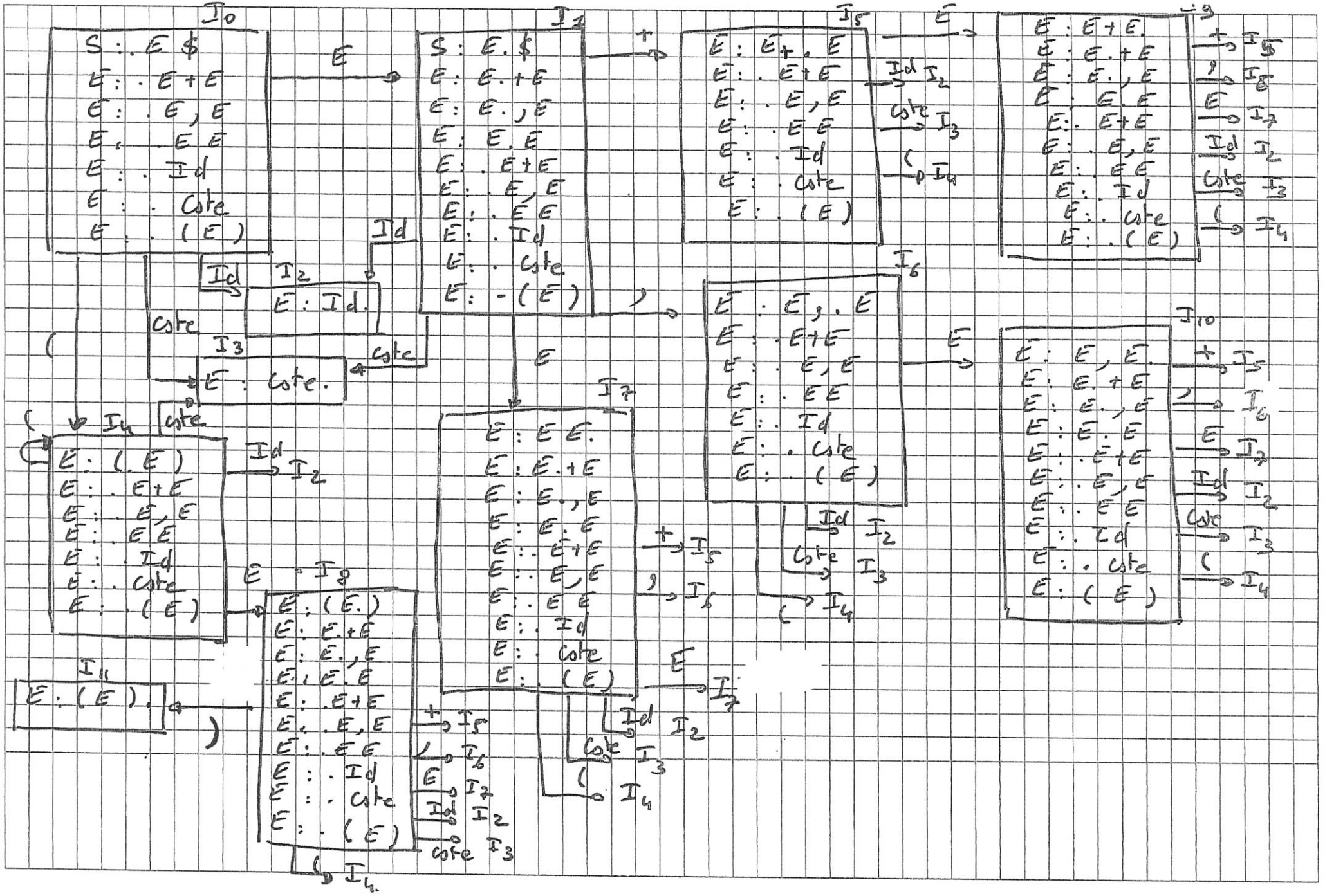
3. Donnez les arbres syntaxiques des expressions `add 3 5 + 2` et `add 3 + 5 2` dans **votre** grammaire en considérant `add` comme une instance de `Id` et `3` et `5` comme des instances de `Cste`. Laquelle des deux correspondra à une expression correcte du point de vue du typage ?

4. Donnez les ensembles *First* et *Follow* **pour la grammaire de départ** augmentée de S ::= E $.

5. A l'aide de l'automate LR(0) fourni en annexe, indiquez **tous** les conflits (état, type de conflit et caractère d'avance) et **montrez comment les résoudre** pour avoir un analyseur correct.

6. Indiquez les étapes dans l'analyse syntaxique de l'expression `Id Cste Cste + Cste $`.

- ANNEXE -


I0
S : . E $
E : . E + E
E : . E , E
E : . E E
E : . Id
E : . cste
E : . (E)
E
I1
S : E . $
E : E . + E
E : E . , E
E : E . E
E : . E + E
E : . E , E
E : . E E
E : . Id
E : . cste
E : . (E)
+
I5
E : E + . E
E : . E + E
E : . E , E
E : . E E
E : . Id
E : . cste
E : . (E)
Id
I2
cste
I3
(
I4
E : E + E .
E : E . + E
E : E . , E
E : E . E
E : . E + E
E : . E , E
E : . E E
E : . Id
E : . cste
E : . (E)
,
I6
E : E , E .
E : E . + E
E : E . , E
E : E . E
E : . E + E
E : . E , E
E : . E E
E : . Id
E : . cste
E : . (E)
I7
E : E E .
E : E . + E
E : E . , E
E : E . E
E : . E + E
E : . E , E
E : . E E
E : . Id
E : . cste
E : . (E)
I8
E : (E .)
E : E . + E
E : E . , E
E : E . E
E : . E + E
E : . E , E
E : . E E
E : . Id
E : . cste
E : . (E)
)
I9
E : (E) .
I10