# Théorie des langages et compilation – Examen

- **Durée : 1h30.**
- **Documents, calculatrices et dictionnaires autorisés.**
- **Les exercices sont indépendants.**
- **Toute réponse non justifiée ne rapportera pas de point.**

## Exercice 1 – Langages réguliers

On considère le langage $L$ des mots sur l'alphabet $\{a, b\}$ où tous les $a$ sont précédés de 1 ou 3 $b$.

1. Donner une expression régulière reconnaissant $L$.
2. Transformer cette expression régulière en un automate sans $\varepsilon$-transition.
3. Déterminiser cet automate.

## Exercice 2 – Un langage pas régulier

1. Donner une grammaire pour le langage $L = \{a^i b^j c^k, i + j = k\}$.
2. Montrer que $L$ n'est pas régulier.

## Exercice 3 – Analyse LL

Sur l'alphabet $\{x, y, z, \$\}$, on considère la grammaire

$$S \to A\$ \qquad A \to xCBy \qquad B \to z \mid \varepsilon \qquad C \to y \mid Bx$$

où $A, B, C, S$ sont des non-terminaux ($S$ étant l'axiome).

1. Calculer les ensembles Premier et Suivant associés à chaque non-terminal.
2. Construire la table d'analyse $LL(1)$. Cette grammaire est-elle $LL(1)$ ?
3. Effectuer l'analyse du mot $xxzy\$$.

## Exercice 4 – Typage

On considère les fonctions TCL suivantes :

```
auto f(auto a) {
    return f({a});
}
```

```
auto g(auto x, auto y, auto z) {
    if (x[0] > x[y]) {
        return z[y];
    }
    return x;
}
```

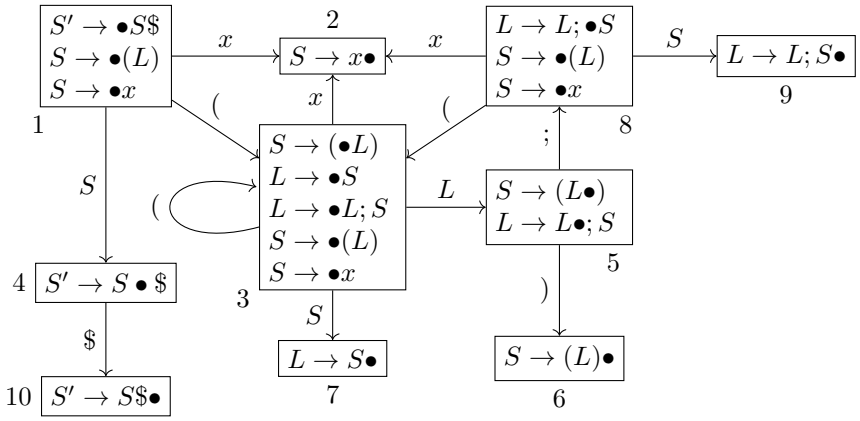
1. Donner la liste des paires de types à unifier lors du typage de chacune des fonctions.
2. Effectuer l'analyse de types : pour chaque fonction, donner son type final si l'analyse a réussi, ou préciser quelle erreur est renvoyée dans le cas contraire.

## Exercice 5 – Analyse LR

Sur l'alphabet $\{(, ), ;, x, \$\}$, on considère la grammaire

$$S' \to S\$ \qquad S \to x \mid (L) \qquad L \to S \mid L; S$$

d'axiome $S'$. L'analyse LR de cette grammaire a permis de construire l'automate des items suivant :




Effectuer l'analyse du mot $(x; (x; x))\$$.

## Exercice 6 – Groupe 3

La portion de code TCL suivant a été traduite en assembleur par le groupe 2.

```
x=12;
while (z < 3) {
    if (y > 0) {
        z=4+y;
    } else {
        t=x-y+z;
    }
    y=3+x;
}
x=t+8
```

```
        XOR R0 R0 R0
        ADDi R1 R0 12
label0: ADDi R2 R1 3
        JSEQ R3 R2 label1
        JIEQ R4 R0 label5
        ADDi R5 R4 4
        ADDi R3 R5 0
        JMP label4
label5: SUB R5 R1 R4
        ADD R6 R5 R3
        ADDi R7 R6 0
label4: ADDi R8 R1 3
        ADDi R4 R8 0
        JMP label0
label1: ADDi R9 R7 8
        ADDi R1 R9 0
```

1. Construire le graphe de contrôle à l'échelle des instructions assembleur.
2. Calculer $LV_{entry}$ et $LV_{exit}$ pour chaque instruction.
3. Dessiner le graphe de conflits et le colorer.
4. Transformer le code assembleur pour minimiser le nombre de registres utilisés.
5. Comment transformer le code pour le faire fonctionner sur un processeur à 4 registres ?