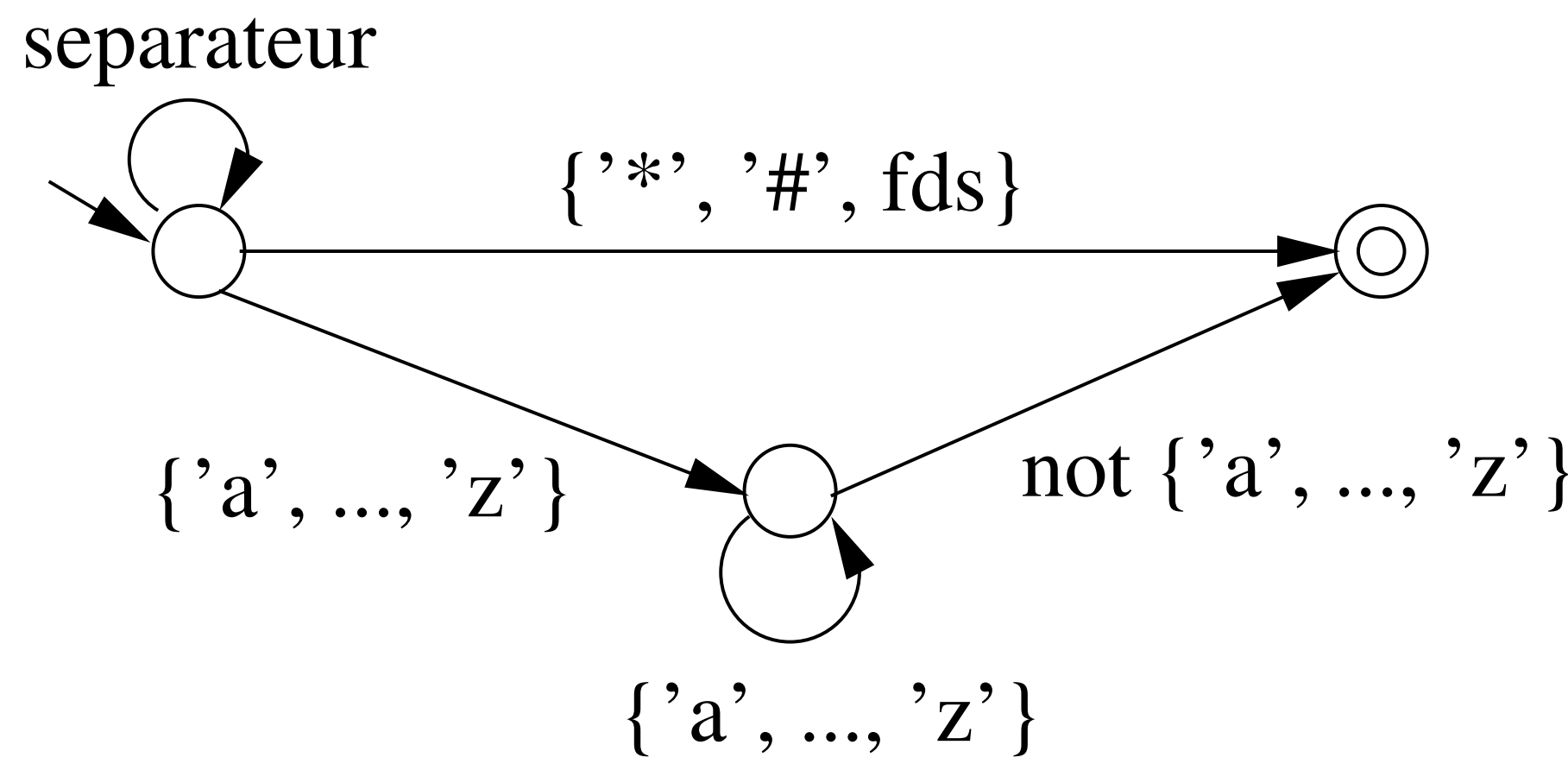
separateur
{'*', '#', fds}
{'a', ..., 'z'}
not {'a', ..., 'z'}
{'a', ..., 'z'}

Q1. Automate de l'analyseur lexical : voir fichier PDF joint

Q2. La phrase

```
      # Pourquoi les flamands * roses * sont-ils * # roses * ? #
```

   contient exactement TROIS erreurs lexicales.

En particulier les trois lexèmes ci-dessous sont incorrects :

```
      "Pourquoi"   (contient une lettre majuscule)
      "sont-ils"   (contient le caractère '-')
      "?"      (n'est pas un lexème)
```

Q3. La phrase suivante contient une erreur syntaxique :

```
      * boujour #
```

Q4. En première lecture, on peut considérer ce langage comme NON RÉGULIER ...
    Intuitivement il faut en effet vérifier la bonne imbrication des blocs de texte en gras et en italique,
    ce qui peut nécessiter de compter les caractères qui délimitent ces structures
    (comme dans le cas des parenthèses).

    Dans ce cas particulier, comme c'est le même caractère qui délimite les début et les fin de blocs,
     il suffit de vérifier que les # et les * apparaissent en nombre "pair". Cela peut se faire via
     un automate à deux états uniquement. Ce langage est donc finalement REGULIER.

Q5. On s'appuie sur la structure de la grammaire :

```
analyser (char *nom_fichier) =
      demarrer (nom_fichier) ;
      rec_texte ;


rec_texte =
      rec_portion_texte ;
      rec_suite_texte ;
      if (LC.nature != FDS) alors
            Erreur ;

rec_portion_texte =
      selon LC.nature
            cas ETOILE:
                  avancer ;
                  rec_portion_texte ;
                  rec_suite_texte ;
                  si LC.nature = ETOILE alors avancer sinon Erreur ;
            cas DIESE:
                  avancer ;
                  rec_portion_texte ;
                  rec_suite_texte ;
                  si LC.nature = DIESE alors avancer sinon Erreur ;
            cas MOT:
                  avancer ;
            sinon :
                  Erreur ;

rec_suite_texte =
```

```
      selon LC.nature
            cas ETOILE, DIESE, MOT :
                  rec_portion_texte ;
                  rec_suite_texte ;
            sinon :
                        // epsilon
```

Q6.

```
int verifie_gi(Ast A) =
      si A != NULL alors
            selon A.nature
                  cas N_SEP :
                              // on verifie les fils gauches et droits
                        return (verifie_gi(A.gauche) && verifie_gi(A.droit)) ;
                  cas N_MOT :
                        return 1 ; // ce sous-arbre est toujours correct
                  cas N_GRAS :
                              // ok si :
                              //   - il n'y a pas de sous-arbre italique
                              //   - et le fils gauche est correct
                        return (
                              !contient_ital(A.droit) && verifie_gi(A.droit)
                               )
                  cas N_ITAL :
                              // ok si :
                              //   - il n'y a pas de sous-arbre gras
                              //   - et le fils gauche est correct
                        return (
                              !contient_gras(A.droit) && verifie_gi(A.droit)
                               )
      sinon
            return 1 ; // un arbre vide est correct
```

avec les deux fonctions auxilliaires ci-dessous :

```
int contient_gras (AST A) =
      si A != NULL alors
            selon A.nature
                  cas N_SEP :
                        // vrai si l'un ou l'autre des fils contient un mot en
gras
                        return (contient_gras(A.gauche) ||
contient_gras(A.droit)) ;
                  cas N_MOT :
                        return 0 ; // ne contient pas d'elements en gras
                  cas N_GRAS :
                        return 1 ; // contient un element en gras
                  cas N_ITAL :
                        vrai si le fils gauche contient un element en gras
                        return contient_gras(A.droit)
      sinon
            return 0 ; // un arbre vide ne contient pas d'elements en gras
```

Idem pour la fonction  contient_ital en inversant les rôles de N_GRAS et N_ITAL :

```
int contient_ital (AST A) =
      si A != NULL alors
            selon A.nature
                  cas N_SEP :
```

```
                        return (contient_ital(A.gauche) ||
contient_ital(A.droit)) ;
                  cas N_MOT :
                        return 0 ; // ne contient pas d'elements en gras
                  cas N_ITAL :
                        return 1 ; // contient un element en gras
                  cas N_GRAS :
                        return contient_ital(A.droit)
      sinon
            return 0 ; // un arbre vide ne contient pas d'elements en gras
```