CNAM **Cycle B**

Réseaux de Petri – Exercices (1)

I. Structures algorithmiques

Modéliser à l'aide de RdP :

1) un test « si *C* alors [traitement si] sinon [traitement sinon] »,
2) une boucle « pour *i* = 1, *N* faire [traitement] »,
3) une boucle « tant que *C* faire [traitement] »,
4) une boucle « répéter [traitement] jusqu'à *C* ».

Note : *C* est une condition.

II. Distributeur de friandises

On désire modéliser le fonctionnement d'un distributeur de friandises valant toutes 5 F. Représenter avec un RdP le sous-système de rendu de monnaie. La machine rend de préférence des pièces de 10 F, puis ensuite des pièces de 5 F. Par exemple, sur 20 F, la machine rend de préférence (si le stock de pièces le permet) 10 F + 5 F plutôt que 3 x 5 F.

On dispose d'une information *P* concernant le type de pièce introduite : *P = 5*, pièce de 5 F ; *P = 10*, pièce de 10 F ; *P = 20*, pièce de 20 F ; ainsi que d'une information *S10* donnant l'état du stock de pièces de 10 F. **Ne pas tenir compte du cas où la machine ne peut pas rendre la monnaie.**

III. Expression complètement parenthésée

1. On désire spécifier à l'aide d'un RdP le fonctionnement d'un programme permettant de savoir si une expression est bien une expression complètement parenthésée respectant les règles suivantes.

- Les seuls caractères présents sont les opérateur (+, -, *, /), des lettres et des parenthèses.
- Toute expression commence par '(' et se termine par ')'.

Règles :

1) Une parenthèse ‘(‘ peut être suivie de : ‘(‘, une lettre, ‘)’.
2) Une lettre peut être suivie de : un opérateur, ‘)’.
3) Une parenthèse ‘)’ peut être suivie de : ‘)’, un opérateur, la fin de la chaîne.
4) Un opérateur peut être suivi de : ‘(‘, une lettre.

Exemples d’expressions correctes :

()
(a)
(a+b)
(a+(b))
(a+(b+c))

Exemples d’expressions incorrectes :

(a
(a+)
(a+b
(a+b))
a+b
+b-(b/c)

Indications :

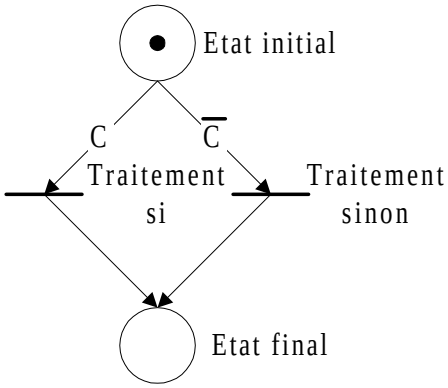
- Utiliser un stock de parenthèses ‘(‘.
- Utiliser un état initial et quatre autres places représentant chacune un des cas 1), 2), 3), 4).

On dispose d’une information *Q* concernant le caractère courant et d’une information *S* concernant l’état du stock de parenthèses ‘(‘. Il est possible de générer les messages suivants a destination du système : *A !* pour incrémenter le compteur courant de 1, *Expression refusée !* lorsque l’expression est incorrecte, *Expression acceptée !* lorsque l’expression est correcte.

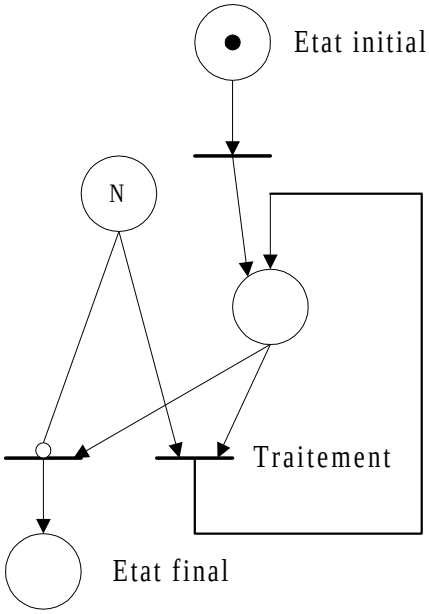
2. Traduire ce RdP dans un langage de programmation comme C.

Correction

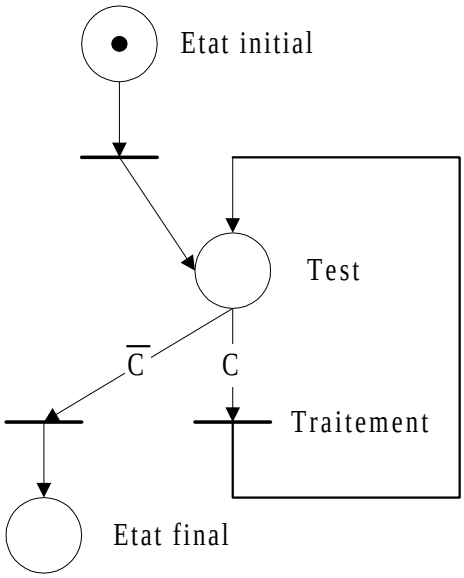
I. 1)



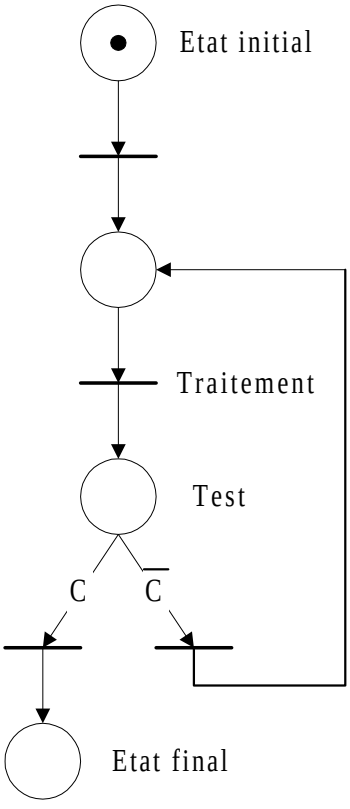
I. 2)



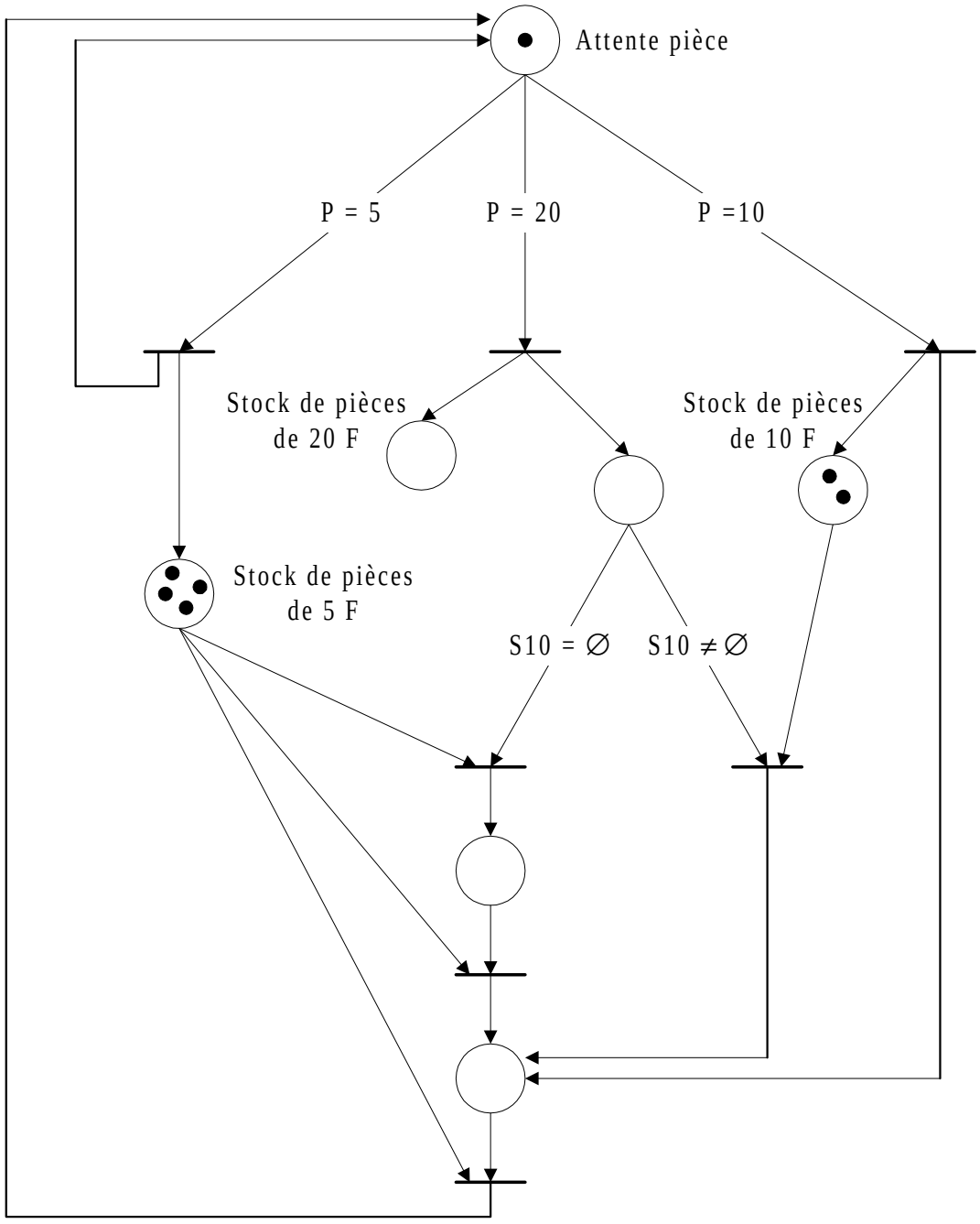
I. 3)



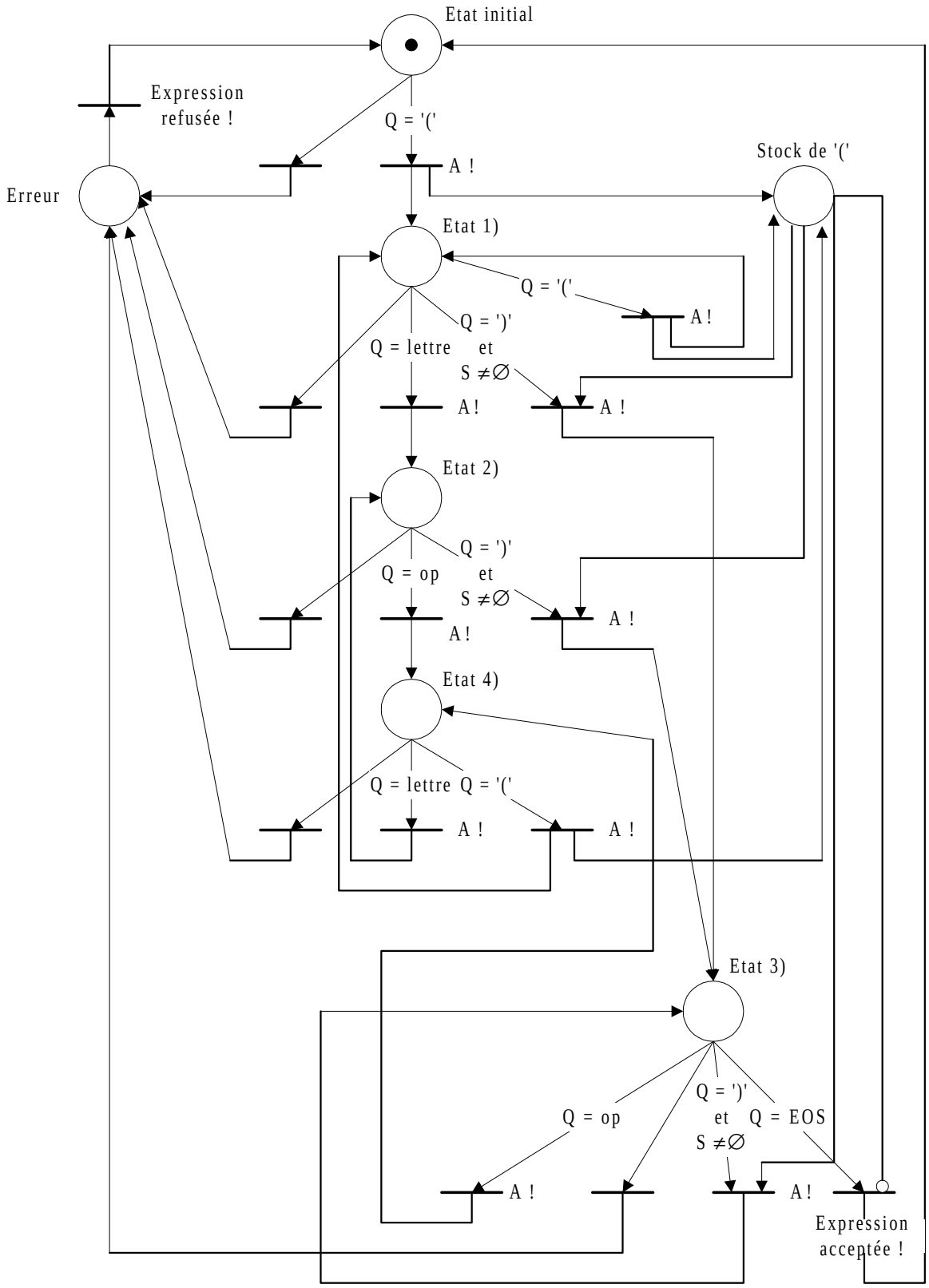
I. 4)



II.


Attente pièce
P = 5
P = 20
P =10
Stock de pièces de 20 F
Stock de pièces de 10 F
Stock de pièces de 5 F
S10 = ∅
S10 ≠ ∅

III. 1)


Etat initial
Expression refusée !
Q = '('
A !
Stock de '('
Erreur
Etat 1)
Q = '('
A !
Q = ')'
Q = lettre et S ≠∅
A !
A !
Etat 2)
Q = ')'
Q = op et S ≠∅
A !
A !
Etat 4)
Q = lettre
Q = '('
A !
A !
Etat 3)
Q = op
Q = ')' et S ≠∅
Q = EOS
A !
A !
Expression acceptée !

III. 2)

```
#include <stdio.h>

#define CHAINEMAX 50

main() {

  // Declarations

  char chaine[CHAINEMAX];
  int a;
  int etat;
  int nb_par;
  int res;

  // Saisie chaine

  printf("Saisir une chaine (max. %d caracteres) : ",CHAINEMAX);
  scanf("%s",chaine);

  // Initialisation

  a=0;
  nb_par=0;
  etat=0;
  res=0;

  // Analyse

  printf("\nDebut analyse...\n");

  while (etat!=9) {

    printf("Automate dans l'etat %d, caractere courant =
%c\n",etat,chaine[a]);

    switch (etat) {

      // Etat initial

      case 0: {
        if (chaine[a]=='(') {
          nb_par++;
          etat=1;
        } else {
          res=1;
          etat=5;
        }
        break;
      }

      // Etat 1 (apres '(')

      case 1: {
        if (chaine[a]=='(') {
          nb_par++;
          etat=1;
        } else if ((chaine[a]==')') && (nb_par>0)) {
          nb_par--;
          etat=3;
```

```
  } else if ((chaine[a]>='a') && (chaine[a]<='z')) {
    etat=2;
  } else {
    res=2;
    etat=5;
  }
  break;
}

// Etat 2 (apres lettre)

  case 2: {
  if ((chaine[a]==')') && (nb_par>0)) {
    nb_par--;
    etat=3;
  } else if ((chaine[a]=='+') || (chaine[a]=='-') ||
             (chaine[a]=='*') || (chaine[a]=='/')) {
    etat=4;
  } else {
    res=3;
    etat=5;
  }
  break;
}

// Etat 3 (apres ')')

case 3: {
  if ((chaine[a]==')') && (nb_par>0)) {
    nb_par--;
    etat=3;
  } else if ((chaine[a]=='+') || (chaine[a]=='-') ||
             (chaine[a]=='*') || (chaine[a]=='/')) {
    etat=4;
  } else if (chaine[a]=='\0') {
    res=0;
    etat=9;
  } else {
    res=4;
    etat=5;
  }
  break;
}

// Etat 4 (apres op)

case 4: {
  if (chaine[a]=='(') {
    nb_par++;
    etat=1;
  } else if ((chaine[a]>='a') && (chaine[a]<='z')) {
    etat=2;
  } else {
    res=5;
    etat=5;
  }
  break;
}

// Etat 5 (erreur)
```

```
      case 5: {
        printf("ERREUR a la position %d ! ",a);
        switch(res) {
          case 1: printf("'(' attendue\n");break;
          case 2: printf("lettre ou '(' ou ')' attendue ou pb de parenthe-
se\n");break;
          case 3: printf("operateur ou ')' attendu(e) ou pb de parenthe-
se\n");break;
          case 4: printf("operateur ou ')' ou EOS attendu(e) ou pb de pa-
renthese\n");break;
          case 5: printf("lettre ou ')' attendue\n");break;
        }
        etat=9;
      }

    }

    a++;

  }

  // Resultat

  if (res) printf("\nExpression refusee !\n\n");
  else printf("\nExpression acceptee !\n\n");

}
```