UNIVERSITÉ PIERRE & MARIE CURIE — SPÉCIFICATION ET CERTIFICATION DU LOGICIEL
MAÎTRISE INFORMATIQUE — *2000-01*

INTERROGATION ÉCRITE
– avec son corrigé –
Novembre *2000*

—○—

QUESTION I – Spécifications algébriques.

1. On suppose connues la sorte `Char` spécifiant l'ensemble des caractères et la sorte `Nat` spécifiant les entiers naturels, donner une spécification algébrique du type `String` (chaînes de caractères).
= *Correction* ==>

Sort: `String`
Uses: `Char, int`
Symbols:
ε : $\rightarrow$ `String`
`"_"`: `Char` $\rightarrow$ `String`
`_^_`: `String, String` $\rightarrow$ `String`
`len` : `String` $\rightarrow$ `int`
Axioms: $\forall$ `c:Char; s1, s2, s3:String.`
ε`^s1 = s1`
`s1^`ε `= s1`
`s1^(s2^s3) = (s1^s2)^s3`
`(len `ε`) = 0`
`(len "c") = 1`
`(len s1^s2) = (len s1)+(len s2)`

<== *Correction* =

2. Donner la spéficication d'une fonction `remdup : Char, String` $\rightarrow$ `String` telle que `(remdup c s)` soit la chaîne obtebue en remplaçant dans `s` toute suite de `c` par un seul `c`.
= *Correction* ==>

Extends: `String`
Symbols:
`remdup: Char, String` $\rightarrow$ `Char`
Axioms: $\forall$ `c1, c2:Char; s:String.`
`(remdup c1 `ε`) = `ε
`c1`$\neq$`c2` $\Rightarrow$ `(remdup c1 "c2"^s) = "c2"^(remdup c1 s)`
`(remdup c1 "c1"^"c1"^s) = (remdup c1 "c1"^s)`
`c1`$\neq$`c2` $\Rightarrow$ `(remdup c1 "c1"^"c2"^s) = "c1"^"c2"^(remdup c1 s)`
`(remdup c1 "c1") = "c1"`

<== *Correction* =

3. Si vous ne l'avez déjà fait, donnez la définition d'une fonction `len` donnant la longueur (nombre de caractères) d'une chaîne. Montrer que:

$$\forall \texttt{s:String}\ \forall \texttt{c:Char}\ \texttt{(len (remdup c s))} \leq \texttt{(len s)}$$

= *Correction* ==>
Par induction sur $n =$ `(len s)`, on montre que

$$\forall \texttt{c:Char}\ \texttt{(len (remdup c s))} \leq \texttt{(len s)}$$

– si $n = 0$ alors `s` $= \varepsilon$, c'est trivial.
– si $n > 0$ on suppose (hypothèse d'induction)

$$\forall \texttt{s':String}\ \texttt{(len s')} < n \Rightarrow \forall \texttt{c:Char}\ \texttt{(len (remdup c s))} \leq \texttt{(len s)}$$

Si `(len s)`> 0 alors il existe c_1`:Char` et s_1`:String` tels que `s` $= c_1$`^`s_1.
– si `c`$\neq c_1$ alors `(remdup c s) = "`c_1`"^(remdup c `s_1`)`. On a, par hypothèse d'induction

$$\texttt{(len (remdup c }s_1\texttt{))} \leq \texttt{(len }s_1\texttt{)}$$

d'où

$$\texttt{(len (remdup c s))} = \texttt{(len (remdup c }s_1\texttt{))+1} \leq \texttt{(len }s_1\texttt{)+1} = \texttt{(len s)}$$

– si `c`$\neq c_1$, on raisonne par cas sur s_1.
– si $s_1 = \varepsilon$ on a alors `(remdup c s) = "`c_1`"` et c'est trivial.
– sinon, il existe il existe c_2`:Char` et s_2`:String` tels que `s = "c"^`s_1 `= "c"^`c_2`"^`s_2.
– si `c`$\neq c_2$ alors `(remdup c s) = "c"^"`c_2`"^(remdup c `s_2`)`. Et, en utilisant l'hypoth
d'induction, on a bien

$$\texttt{(len (remdup c s))} = \texttt{(len (remdup c }s_2\texttt{))+2} \leq \texttt{(len }s_2\texttt{)+2} = \texttt{(len s)}$$

– si `c`$= c_2$ alors `(remdup c s) = (remdup c "c"^`s_2`)`. Par hypothèse d'induction, o

$$\texttt{(len (remdup c "c"^}s_2\texttt{))} \leq \texttt{(len "c"^}s_2\texttt{)}.$$

D'où

$$\texttt{(len (remdup c s))} \leq \texttt{(len }s_2\texttt{)+1} < \texttt{(len s)}$$

<== *Correction* =

QUESTION II – Spécifications ensemblistes. On veut modéliser le fond d'une bibliothèque et son mécani de gestion des emprunts.
Voici une première description informelle des ressources de la bibliothèque :

La bibliothèque dispose d'un certain nombre de titres dont chaque exemplaire est indentifé par une cote. La bibliothèque gère l'emprunt et la restitution des ouvrages auprés d'un ensemble d'abonnés. On connaît, à chaque instant, le nombre d'exemplaires disponibles d'un titre donné ainsi que la liste des ouvrages empruntés par chaque abonné.

L'analyse de l'organisation et du fonctionnement de la bibliothèque a fait resortir les contraintes suivan
– *Une cote est associée à un titre au plus.*
– *Un abonné peut emprunter plusieurs titres, mais il ne peut emprunter plusieurs exemplaires d'un m titre.*
– *Un même exemplaire ne peut être emprunté par plusieurs abonnés.*

1. Déterminer et définir lorsque c'est possible, les diverses ressources (i.e. ensembles) gérées.
= *Correction* ==>
On se donne trois ensembles abstraits

A : ensemble des abonnés
T : ensembles de titres
C : ensemble des cotes

On se donne l'ensemble E des exemplaires comme une relation entre les cotes et les titres

$$E \in C \leftrightarrow T$$

La liste des ouvrages empruntés par un abonnés est donné par la fonction

$$AL \in A \rightarrow \mathcal{P}(E)$$

Il faut se donner, bien que non explicitement demandé, l'ensemble L est exemplaires disponibles. On peut le définir comme

$$L = E \setminus \bigcup_{x \in \mathrm{ran}(AL)} x$$

Le nombre d'exemplaires disponibles pour chaque titre est donné par la fonction

$$\begin{array}{rl} N: & T \to I\!N \\ & t \mapsto \#\{c \in C \mid (c,t) \in L\} \end{array}$$

où $\bigcup_{x \in \mathrm{ran}(AL)} x$ est l'ensemble des exemplaires empruntés.
On a besoin
<== *Correction* =

2. Exprimer formellement les contraintes dégagées.
= *Correction* ==>
   - en fait, la relation E est fonctionnelle et partielle
   $$E \in C \nrightarrow T$$
   - On scinde l'énoncé en deux parties
     - un abonné peut emprunter plusieurs titres ; c'est déjà dit dans le fait que AL associe un ensemble d'exemplaires à chaque abonné.
     - mais un abonné ne peut emprunter qu'un seul exemplaire d'un même titre
     $$\forall a \in A. \forall t \in T. \forall c_1, c_2 \in C. (c_1, t) \in AL(a) \wedge (c_2, t) \in AL(a) \Rightarrow c_1 = c_2$$
   - un même exemplaire ne peut être emprunté par deux abonnés différents
   $$\forall a_1, a_2 \in A. a_1 \neq a_2 \Rightarrow AL(a_1) \cap AL(a_2) = \emptyset$$

<== *Correction* =

3. Spécifier les opérations d'emprunt et de restitution.

Bib

$AL : A \to \mathcal{P}(E)$
$N : T \to I\!N$
$L : \mathcal{P}(E)$

Emprunt

ΔBib
$a? : A$
$t? : T$
$c : C$

$N(t?) > 0 \wedge$
$(t, c) \in L \wedge$
$AL' = AL \oplus \{a \mapsto (c, t)\} \wedge$
$N' = N \oplus \{t \mapsto N(t) - 1\} \wedge$
$L' = L \setminus \{(c, t)\}$

ΔBib

Bib
Bib′

Restitution

ΔBib
$a? : A$
$(c, t)? : E$

$AL' = AL \oplus \{a \mapsto AL(a) \setminus \{(c, t)\}\} \wedge$
$N' = N \oplus \{t \mapsto N(t) + 1\} \wedge$
$L' = L \cup \{(c, t)\}$

En fait, il faudrait aussi donner l'état initial de la bibliothèque

InitBib

Bib

$\forall a \in A. AL(a) = \emptyset \wedge$
$\forall t \in T. N(t) = \#\{c \in C \mid (c, t) \in L\} \wedge$
$L = E$

QUESTION III – Logique de Hoare-Floyd. Soient trois constantes `c1`, `c2` et `c3` telles que `c1` $\leq$ `c2` $\leq$ `c3`. S
`t` un tableau ne contenant que ces constantes et dont l'intevalle d'indice est $[1..L]$. Soit alors le programm

```
i, j, k := 1, 1, L;
While j <= k do
 if t(j) = c1 then
  begin t(i), t(j) := t(j), t(i); i, j := i+1, j+1 end
 else if t(j) = c2 then
  j := j+1
 else
  begin t(j), t(k) := t(k), t(j); k := k-1 end
done
```

Soient Montrez que ce programme statisfait :

$$\forall\ \texttt{i,j} \in [\texttt{1..L}]\ (\texttt{i} \leq \texttt{j} \Rightarrow \texttt{t(i)} \leq \texttt{t(j)})$$

= *Correction* ==>
Posons

$$\begin{array}{rcl} \texttt{init} & \widehat{=} & \texttt{i, j, k := 1, 1, L} \\ \texttt{loop} & \widehat{=} & \texttt{While j <= k do ... done} \\ C & \widehat{=} & \texttt{\{ c1, c2, c3 \}} \\ P & \widehat{=} & \texttt{t} \in C\ \textit{seq} \wedge \texttt{(len t) = L} \\ Q & \widehat{=} & \forall\ \texttt{i,j} \in [\texttt{1..L}].(\texttt{i} \leq \texttt{j} \Rightarrow \texttt{t(i)} \leq \texttt{T(j)} \\ I & \widehat{=} & \forall\ \texttt{x} \in [\texttt{1..i}[.\ \texttt{t(x)=c1} \wedge \\ & & \forall\ \texttt{x} \in [\texttt{i..j}[.\ \texttt{t(x)=c2} \wedge \\ & & \forall\ \texttt{x} \in]\texttt{k..L}].\ \texttt{t(x)=c3} \wedge \\ & & \texttt{j} \leq \texttt{k+1} \\ B & \widehat{=} & \texttt{j} \leq \texttt{k} \\ \neg B & \widehat{=} & \texttt{k} < \texttt{j} \end{array}$$

On applique le schéma de preuve de boucle :

1. $[\ P\ ]$ `init` $[\ I\ ]$: c'est trivial.
2. $\neg B \wedge I \Rightarrow Q$: on a $\neg B \wedge I \Rightarrow$ `j=k+1`. On vérifie alors que
   $I \wedge$ `j=k+1` $\Rightarrow \forall\ \texttt{x} \in [\texttt{1..i}[.\ \texttt{t(x)=c1} \wedge \forall\ \texttt{x} \in [\texttt{i..k}].\ \texttt{t(x)=c2} \wedge \forall\ \texttt{x} \in]\texttt{k..L}].\ \texttt{t(x)=c3}$
   alors à vérifier que quelque soit le couple d'indice considéré, on obtient Q (laissé en exercice.
3. $[\ I \wedge B\ ]$ ... $[\ I\ ]$, où les trois points représentent le corps de la boucle : pour alléger, on éc
   `t[a..b[=c` à la place de $\forall$`x`$\in$`[a..b[. t(x)=c` (resp. `t]a..b]=c`, pour $\forall$`x`$\in$`]a..b]. t(x)=c`);
   plus, on ne retiendra de l'invariant que les éléments pertinents.
   On raisonne par cas sur `t(j)`.
   - si `t(j)=c1`, on traite la séquence `t(i), t(j) := t(j), t(i); i, j := i+1, j+1`. Il f
     considérer deux sous-cas : `i=j` ou `i`$\neq$`j`.
     - si `i=j`, on a (en écrivant verticalement les triplets

$$\begin{array}{l} [\ \texttt{i=j} \wedge \texttt{t[1..i[=c1} \wedge \texttt{t[i..j[=c2} \wedge \texttt{t(j)=c1}\ ] \\ \qquad\qquad \texttt{t(i), t(j) := t(j), t(i)} \\ [\ \texttt{i=j} \wedge \texttt{t[1..i[=c1} \wedge \texttt{t[i..j[=c2} \wedge \texttt{t(i)=c1}\ ] \end{array}$$

comme

`i=j ∧ t[1..i[=c1 ∧ t[i..j[=c2 ∧ t(i)=c1 ⇒ t[1..i+1[=c1 ∧ t[i+1..j+1[=c2`

on retrouve

```
[ t[1..i+1[=c1 ∧ t[i+1..j+1[=c2 ]
        i, j := i+1, j+1
  [ t[1..i[=c1 ∧ t[i..j[=c2 ]
```

– sinon, `i≠j` et alors, nécessairement `i<j`. On a que

```
i<j ∧ t[1..i[=c1 ∧ t[i..j[=c2 ∧ t(j)=c1
⇒ i<j ∧ t[1..i[=c1 ∧ t(i)=c2 ∧ t[i+1..j[=c2 ∧ t(j)=c1
```

D'où

```
[ i<j ∧ t[1..i[=c1 ∧ t(i)=c2 ∧ t[i+1..j[=c2 ∧ t(j)=c1 ]
          t(i), t(j) := t(j), t(i)
[ i<j ∧ t[1..i[=c1 ∧ t(j)=c2 ∧ t[i+1..j[=c2 ∧ t(i)=c1 ]
```

Et comme

```
i<j ∧ t[1..i[=c1 ∧ t(j)=c2 ∧ t[i+1..j[=c2 ∧ t(i)=c1
⇒ t[1..i+1[=c1 ∧ t[i+1..j+1[=c2
```

on retrouve

```
[ t[1..i+1[=c1 ∧ t[i+1..j+1[=c2 ]
        i, j := i+1, j+1
  [ t[1..i[=c1 ∧ t[i+1..j[=c2 ]
```

– si `t(j)=c2`, on traite l'incrément `j := j+1`. On a

`t[1..i[=c1 ∧ t[i..j[=c2 ∧ t(j)=c2 ⇒ t[1..i[=c1 ∧ t[i..j+1[=c2.`

Et

```
[ t[1..i[=c1 ∧ t[i..j+1[=c2 ]
        j := j+1
  [ t[1..i[=c1 ∧ t[i..j[=c2 ]
```

– si `t(j)=c3`, on traite la séquence `t(j), t(k) := t(k), t(j); k := k-1`. On a

`[ t]k..L]=c3 ∧ t(j)=c3 ] t(j), t(k) := t(k), t(j) [ t]k..L]=c3 ∧ t(k)=c3 ]`

Comme on a

```
t]k..L]=c3 ∧ t(k)=c3 ⇒ t]k-1..L]=c3
```

on retrouve

```
[ t]k-1..L]=c3 ] k := k-1 [ t]k..L]=c3 ]
```

<== *Correction* =