INF3140 — Modélisation et spécification formelles de logiciels
Examen intra (Automne 2008)

Durée: 13h30 – 16h30 (3 heures) **Documentation autorisée:** Toute documentation personnelle.

Nom: _____

Code permanent: | | | | | | | | | | | | |

Directives:

a. Répondez aux questions directement sur le questionnaire dans les espaces prévus à cette fin.

b. Utilisez les versos pour vos brouillons. Si vous devez utiliser un verso pour compléter une réponse, indiquez le au recto.

c. N'écrivez rien dans la table à droite (réservée au correcteur).

1	/10
2	/15
3	/15
4	/15
5	/10
Bonus	/10
Total	/65

1. Littéraux et types (10 pts)

Indiquez le type de chacun des littéraux suivants.

```
CONCEPT v1:

  WHERE
   v1 = [ {"bac", "def"}, {"abc", ""}, {}, {} ]



CONCEPT v2:

  WHERE
   v2 = { [{10}, {2}], [{0}, {}], [{3, 19}, {343}] }




CONCEPT v3:

  WHERE
   v3 = { [{10}, {2}], [{0}, {}], [{3, 19}, {343}]; {} }



CONCEPT v4:

  WHERE
   v4 = { ["abc", {[1, 'a'], [2, 'b'], [3, 'c']; '?'}],
          ["xx",  {[1, 'x'], [2, 'x']; '?'}],
          ["u",   {[1, 'u']; '?'}];
          {; '?'} }
```

2. Types abstraits `Spec` (15 pts)

[10] a) Soit les deux valeurs suivantes :

```
CONCEPT s1: sequence{sequence{string}}
  WHERE s1 = [ ["bac", "def"], ["abc", ""], [] ]

CONCEPT d1: map{set{integer}, nat}
  WHERE d1 = { [{-10, 20}, 2], [{0}, 1], [{3..5}, 3]; 0 }
```

Pour chacune des expressions suivantes, donnez le type de l'expression (après «`expression_x:`») ainsi que la valeur résultante (après «`# Valeur =`») :

```
CONCEPT expression_a:
  WHERE
    expression_a = SUM( i: nat SUCH THAT i IN domain(s1) :: length(s1[i]) )

# Valeur =



CONCEPT expression_b:
  WHERE
    expression_b = SOME( s: sequence{string} SUCH THAT s IN s1 :: "" IN s )

# Valeur =



CONCEPT expression_c:
  WHERE
    expression_c = add( [], remove( [], s1 ) )[2][2]

# Valeur =



CONCEPT expression_d:
  WHERE
    expression_d = MAXIMUM( s: set{integer} SUCH THAT s IN d1 :: 10 * d1[s] )

# Valeur =



CONCEPT expression_e:
  WHERE
    expression_e = bind( {0}, 0, remove( {3, 4, 5}, d1 ) )

# Valeur =
```

[5] b) Pour les deux expressions suivantes, reprises de celles plus haut, en utilisant les opérations de la bibliothèque Java `TypesSpec`, donnez le type Java approprié ainsi que l'expression équivalente (pas la valeur!) exprimée en Java (comme dans le labo 1 et le devoir 1) :

```
// expression_b = SOME( s: sequence{string} SUCH THAT s IN s1 :: "" IN s )

TypeJava =

expr =
```

```
// expression_c = add( [], remove( [], s1 ) )[2][2]

TypeJava =

expr =
```

3. Concepts pour des programmes Java (15 pts)

Soit les concepts suivants, définis dans un module `DEFINITION`, qui définissent diverses caractéristiques et propriétés d'un programme Java — vous pouvez détacher cette feuille, pour la consulter plus facilement :

```
DEFINITION programmes_java
  IMPORT Subtype FROM type

  -- L'entite programme.
  CONCEPT programme: type

  -- Attributs d'un programme.
  CONCEPT classes         ( p: programme ) VALUE( cs: set{classe} )
  CONCEPT classe_de_depart( p: programme ) VALUE( c: classe )
     -- Classe specifiee explicitement a l'appel de la machine virtuelle Java.
     WHERE c IN classes(p)

  -- Types auxiliaires.
  CONCEPT classe: type
  CONCEPT nom          ( c: classe ) VALUE( n: nom_classe )
  CONCEPT methodes     ( c: classe ) VALUE( ms: set{methode} )


  CONCEPT methode: type
  CONCEPT nom       ( m: methode ) VALUE( n: nom )
  CONCEPT est_public( m: methode ) VALUE( b: boolean )
  CONCEPT arguments ( m: methode ) VALUE( as: sequence{type_java} )
  CONCEPT resultat  ( m: methode ) VALUE( t: type_java )


  -- Types de base representant les types de Java.
  CONCEPT type_java: type
    WHERE
       Subtype( type_java, nom ),
       ALL( tj: type_java :: tj IN nom_classe | tj IN nom_type_primitif )

  CONCEPT nom_classe: type
     WHERE
       Subtype( nom_classe, nom )

  CONCEPT nom_type_primitif: type
    WHERE
      Subtype( nom_type_primitif, nom )

   CONCEPT void: nom_type_primitif WHERE void = "void"
   CONCEPT int:  nom_type_primitif WHERE int  = "int"
   CONCEPT bool: nom_type_primitif WHERE bool = "bool"

   CONCEPT nom: type
     WHERE nom = string
END
```

Complétez les concepts suivants :

```
CONCEPT est_main( m: methode ) VALUE( b: boolean )
# Une methode est_main si son nom est "main", elle est public, son
# resultat est de type void et tous ses arguments sont de type String
```

```
CONCEPT contient_main( c: classe ) VALUE( b: boolean )
# Une classe contient_main si l'une de ses methodes est_main.
```

```
CONCEPT methodes_privees( c: classe ) VALUE( ms: set{methode} )
# Retourne les methodes de c qui ne sont pas des methodes "public".
```

4. Expressions régulières et automates (15 pts)

Rappel : Une expression de la forme «`[`$c_1 c_2 \ldots c_k$`]`» dénote un ensemble contenant les caractères $c_1, c_2, \ldots, c_k$. Une expression comme «`[0-9]`» est donc simplement une abréviation de «`[0123456789]`».

Rappel : Le méta-caractère «`\s`» indique un espace.

[6] a) Voici le contenu d'un fichier — les numéros font partie des lignes (font partie du contenu du fichier) mais servent aussi à identifier chacune des lignes dans les questions qui suivent :

```
1. Voici un bout de texte avec des mots aleatoires.
2. Oui, ce texte contient des choses qui n'ont pas de sens et qui sont erronnes.
3. De meme, il contient un ou des chiffres a la premiere position.
4. Ces nombres sont presents ou
5. pas
6. Voici une series de mots sans sens et sans ordre particulier
7. aa ab ac ad ae af
```

Quelle-s ligne-s sera-seront *matchée-s* par les expressions régulières suivantes — par exemple, utilisées comme argument pour le programme `Grep.java`. Indiquez simplement les numéros de ces lignes :

a. `"s.n.+"`

b. `"^[0-9]..[acAC]+[^ac]+"`

c. `"\.[0-9]*$"`

[4] b) Écrivez une expression régulière qui va permettre de trouver toutes les lignes qui contiennent des mots, de un ou plusieurs caractères, formés entièrement *de voyelles*, mots qui sont entourés d'au moins un espace de chaque coté (un ou plusieurs devant, un ou plusieurs après).

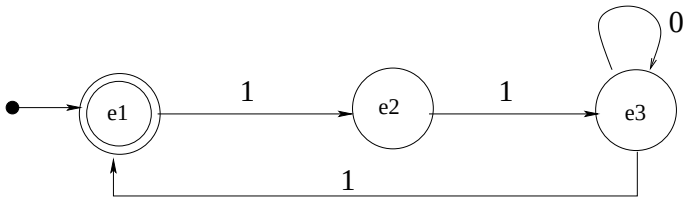
[5] c) *Dessinez* un automate qui accepte les mots décrits dans la sous-question précédente — une ou plusieurs voyelles entourées d'un ou plusieurs espaces.

5. Automates et formalisation en Spec (10 pts)

[5] a) Soit le concept auxiliaire suivant défini dans le module `automate.Spec` :

```
CONCEPT est_transition( t: transition, so: etat, ci: etat, sy: set{char} ) VALUE( b: boolean )
  WHERE
    b <=> source(t) = so & cible(t) = ci & symboles(t) = sy
```

Soit l'automate suivant, sur l'alphabet des bits (`0` ou `1`) :



Complétez le module Spec suivant pour qu'il définisse cet automate — et qu'il soit accepté par l'analyseur Spec, mais pas nécessairement par le programme `interpreter.rb`.

```
DEFINITION automate_question_5_a
  INHERIT automate{zero_ou_un}

  CONCEPT zero:       set{char}  WHERE zero       = { '0' }
  CONCEPT un:         set{char}  WHERE un         = { '1' }
  CONCEPT zero_ou_un: set{char}  WHERE zero_ou_un = {'0', '1'}

  CONCEPT a: automate
     WHERE

       etats(a) =

       transitions(a) =

       etat_initial(a) =

       etats_finaux(a) =

END
```

[5] b) Soit le concept suivant tiré de la solution du devoir 1 :

```
CONCEPT accepte( a: automate, c: chaine ) VALUE( b: boolean )
-- L'automate a, a partir de son etat initial, accepte la chaine c.
```

Formalisez les concepts suivants sur les automates modélisés en Spec (comme dans le devoir 1, dans le fichier `automate.Spec`), en indiquant explicitement le type du résultat :

a. Un concept qui permet de déterminer si un automate `a1` `simule` un automate `a2`. On dit que `a1` «simule» `a2` si pour chaque chaine acceptée par `a2`, `a1` accepte aussi cette chaine.

```
CONCEPT simule( a1 a2: automate ) VALUE( b:                          )
WHERE
  b <=>
```

b. On veut un concept qui permet de déterminer les `etats_ordinaires` d'un automate `a`. Un état est «ordinaire» s'il n'est ni l'état initial de `a`, ni un de ses états finaux (acceptants).

```
CONCEPT etats_ordinaires( a: automate ) VALUE( es:                              )
 WHERE
```

Bonus (10 pts)

Pour ces questions, vous pouvez (devez!) introduire des concepts auxiliaires. Si vous manquez d'espace, vous pouvez utiliser le recto de la feuille.

[5] a) En vous basant sur les concepts de la question 3 (programmes Java), définissez un concept qui détermine si une méthode `m` `est_constructeur_sans_arg` pour une classe `c`. C'est le cas lorsque la méthode `est_constructeur` pour `c` et qu'elle n'a aucun argument. On dira qu'une méthode `m` `est_constructeur` pour une classe `c` si elle est l'une des méthodes de `c`, que le type de son résultat est le nom de la classe lui-même et que le nom de la méthode est vide (égal à `""`).

[5] b) En vous basant sur les concepts des automates, définissez un concept qui produit la `trace` d'un automate `a` pour une chaine `c`. Plus précisément, la `trace` est la séquence des états traversés par `a` lors de son analyse de `c`. Par exemple, la trace de l'automate reconnaissant des nombres points fixes avec ou sans signe (Figure 6, p. 8 des notes de cours sur les automates et expressions régulières) sur la chaine `"+124.78"` serait la séquence `[e1, e2, e3, e3, e3, e4, e5, e5]`.