Consignes

- Le seul document autorisé est une feuille A4 manuscrite recto-verso.
- Vous devez rendre le sujet avec votre copie. Assurez-vous d'avoir écrit votre nom en haut de cette feuille.
- Sauf mention explicite du contraire, vos programmes Heptagon doivent être causaux, bien initialisés et utiliser les horloges de façon cohérente.

1 Oscillateur de Van der Pol (9 points)

L'oscillateur de Van der Pol est un système physique simple dont la dynamique est gouvernée par l'équation différentielle ordinaire suivante.

$$\frac{d^2x(t)}{dt^2} - \mu(1 - x^2(t))\frac{dx(t)}{dt} + x(t) = 0$$

Cette équation décrit l'évolution de la position $x(t)$ de l'oscillateur en fonction d'un paramètre fixé μ (prononcé "mu"). Le but de l'exercice est d'écrire un programme Heptagon simple qui simule ce système de façon numérique, à la manière du pendule inversé ou du véhicule du projet.

1. (2 points) On va commencer par écrire un *intégrateur*, qui est un nœud capable de calculer numériquement la valeur de $x(t)$ en fonction de $dx(t)/dt$, pour n'importe quel flot x.

   ```
   node itgr(dx, x0, dt : float) returns (x : float)
   ```

   Implémentez le nœud ci-dessus pour implémenter l'intégration d'Euler "en avant". Elle est identique à celle vue en cours ou durant le projet, à ceci près que sa sortie `x` doit dépendre de `dx` de façon *retardée* plutôt qu'instantanée.

2. (4 points) Utilisez le nœud `itgr` pour compléter le code ci-dessous de sorte à ce que les flots `x`, `dx` et `d2x` transportent respectivement les valeurs $x(t)$, $dx(t)/dt$ et $d^2x(t)/dt^2$. On supposera que la durée du pas d'intégration est fixe et de 0.01 seconde, et que $x(0) = 0$. Ce nœud *n'a pas à être causal*, cf. question suivante.

   ```
   node vanderpol(mu : float) returns (x : float)
   var dx, d2x : float; ...
   ```

3. (3 points) Le nœud précédent n'est a priori pas causal. Pourquoi ? Quelles solutions pouvez-vous suggérer ?

2 Les ascenseurs de la Gallerie des Grains (7 points)

L'Université Huppée souhaite sous-traiter la rénovation des quatre ascenseurs d'un de ses bâtiments, la Gallerie des Grains. Dans ce but, elle a établi un contrat avec Rinci, une grosse entreprise spécialisée dans le BTP, qui doit en particulier fournir le code Heptagon du contrôleur des ascenceurs. Ce dernier doit implémenter l'interface suivante.

```
type dstatus = Open | Closed
type mstatus = Up | Down | Idle
node elevator(c : bool^5) returns (d : dstatus; m : mstatus)
```

L'entrée `c` précise à chaque cycle si l'ascenseur est appelé à l'un des cinq étages de la Gallerie des Grains. La sortie `d` contrôle l'ouverture de la porte de l'ascenseur. La sortie `m` contrôle le mouvement de l'ascenseur (montée / descente / inactif). Contrairement à un véritable ascenseur, on ne gère pas le choix de l'étage destination.

L'Université Huppée souhaite s'assurer que Rinci obéit bien à son cahier des charges. Pour cela, elle vous sous-traite l'écriture d'un certain nombre d'*observateurs*. Un observateur est un nœud Heptagon qui observe l'interface (entrées et sorties) d'un autre nœud, et renvoie un flot booléen qui vaut vrai tant qu'une propriété de sûreté est vérifiée. Dans le cas qui nous occupe, un observateur a donc la signature suivante.

```
node obs(c : bool^5; d : dstatus; m : mstatus) returns (ok : bool)
```

Attention : il suffit qu'un observateur renvoie faux une seule fois pour que la propriété soit considérée comme falsifiée. Vos observateurs ne doivent donc pas nécessairement maintenir leur sortie à faux en cas d'erreur.

1. (1 point) Écrivez un observateur `obs1` pour la propriété “la porte d’un ascenseur en mouvement est fermée”.
2. (3 points) Écrivez un observateur `obs2` pour la propriété “l’ascenseur ouvre ses portes à l’étage d’un appel en dix cycles au plus après cet appel”. On supposera qu’un ascenseur commence l’exécution au rez-de-chaussée.
3. (3 points) Écrivez un nœud principal `building` qui instancie quatre copies du nœud `elevator` et s’assure que chaque copie vérifie les propriétés `obs1` et `obs2`. L’entrée `c` correspond aux boutons respectifs des ascenseurs.

   ```
   node building(c : bool^5^4) returns (ok : bool)
   ```

   (Vous supposerez que le nœud `elevator` vous a été fourni.)

3 Élimination des automates (4 points)

Le but de cet exercice est de traduire un nœud utilisant des automates vers du code purement équationnel en utilisant de façon judicieuse les opérateurs **`when`** et **`merge`**. Le nœud à traduire est le suivant.

```
node auto(x : int :: .) returns (o : int :: .)
var cpt : int :: .;
let
  cpt = 0 fby (cpt + 1);
  automaton
    state A
      do o = 0 fby x
      until cpt >= 3 then B

    state B
      do o = (0 fby o) + 1
      until o % 3 = 0 then A
  end
tel
```

1. (1 point) Complétez le chronogramme ci-dessous. (Vous écrirez directement sur le sujet.)

x	1	2	4	5	1	5	8	1	4	8	0	…
o												…

2. (3 points) Réécrivez le nœud `auto` pour en supprimer la construction **`automaton`** tout en préservant son interface exacte (entrées et sorties, types et horloges compris). Vous pouvez utiliser les constantes, les opérations arithmétiques, les opérateurs de sélection (**`when`**) et d’entrelacement (**`merge`**). Quelques suggestions :
   1. introduire une nouvelle variable qui correspond à l’état courant de l’automate dans le code initial,
   2. introduire, *pour chaque état*, deux variables distinctes correspondant :
      (a) au prochain état de l’automate et
      (b) à la définition de o locale à cet état,
   3. définir la sortie o en utilisant les variables introduites précédemment.

   On peut éventuellement définir un type énumérés pour les états de l’automate, ou bien utiliser le type **`bool`**.

   Attention : votre programme doit être correct du point de vue des horloges. Nous vous suggérons fortement de spécifier l’horloge de chaque variable locale lors de sa déclaration, comme dans le code original (où toutes les variables sont calculées sur l’horloge de base, ce qui ne devrait pas être le cas dans le code traduit).