# Programmation synchrone (PSYN9)
# TP4 (bonus) : Vérification de programmes synchrones

Cette séance de travaux pratiques concerne la vérification automatique (*model-checking*) de programmes synchrones. On va pour cela utiliser l'outil Kind 2 [1], qui prend en entrée un langage différent mais voisin d'Heptagon. Vous pouvez installer Kind 2 via OPAM.

```
$ opam install kind2
```

Cette ligne suppose que vous avez installé OPAM correctement, comme requis par Heptagon. En guise d'alternative, le manuel de Kind 2 vous expliquera comment utiliser Docker ou une interface en ligne.

**Exercice 1** – *Mise en jambe*

1. Essayer de vérifier le programme suivant avec Kind 2. Que constatez-vous ? Proposer une correction au problème et vérifier que celle-ci fonctionne correctement.

```
node half1(x: bool) returns (o: bool)
var y: bool; let y = true -> pre (not y); o = x and y; tel

node half2(x: bool) returns (o: bool)
var y: bool; let y = false -> pre (true -> pre y); o = x and y; tel

node main_half1_vs_half2(x: bool) returns (ok: bool)
let
  ok = half1(x) = half2(x);
  --%PROPERTY ok;
tel
```

2. Vérifier que la sortie du nœud `f` ci-dessous est toujours égale à `false` pour toute entrée `x` fausse durant les trois premiers instants.

```
node f(x : bool) returns (o : bool)
let
  automaton
    state Init:
      unless if x resume KO end;
      var cpt : int;
      let o = true; cpt = 0 -> (pre(cpt) + 1); tel
      until if cpt >= 1 resume OK end;
    state KO: let o = false; tel
    state OK: let o = true; tel
  returns o;
tel
```

1. https://kind.cs.uiowa.edu

**Exercice 2** – *Détecteur d'intrusion*

Cet exercice se base sur le détecteur d'intrusion réalisé lors des séances précédentes. Il vous faudra très probablement adapter son code pour qu'il soit accepté par Kind2.

1. Vérifier qu'en l'absence totale de mouvements, l'alarme ne se déclenche jamais.

2. Vérifier qu'en la présence d'un flot `mvt` qui ne contient jamais deux `true` consécutifs, l'alarme ne se déclenche jamais.

3. Vérifier qu'en la présence de mouvements durant 50 ms consécutives, l'alarme se déclenche.

**Exercice 3** – *Four à micro-ondes*

On se propose dans cet exercice de programmer un contrôleur de four à micro-ondes. Celui-ci doit prendre la forme d'un nœud à l'interface donnée ci-dessous.

```
node microwave_oven(start, stop, turbo, door : bool; heat, duration : int)
          returns (wave: int)
```

Ce nœud doit obéir à la spécification suivante.

- L'entrée `start` (resp. `turbo`) est vraie lorsque le bouton de démarrage de la cuisson en mode normal (resp. turbo) est pressé. La cuisson en mode normal doit chauffer les aliments durant `duration` secondes à l'intensité `heat` watts. La cuisson en mode normal doit chauffer les aliments durant 30 secondes à 750 watts.
- L'entrée `stop` est vraie lorsque le bouton d'interruption de la cuisson est pressé.
- L'entrée `door` est vraie lorsque la porte du four à micro-ondes est ouverte.
- La sortie `wave` contrôle la puissance du magnétron ; on considère qu'une puissance de zéro signifie que le magnétron est éteint.

On considèrera qu'il s'écoule une durée fixe de 10 millisecondes entre deux instants logiques.

1. Programmer un contrôleur obéissant à la spécification donnée ci-dessus. La cuisson doit être mise en pause lorsque la porte est ouverte.

2. Vérifier que le magnétron est toujours inactif lorsque la porte est ouverte.

3. Vérifier qu'il est impossible de démarrer le magnétron sans avoir vu au moins une pression sur l'un des deux boutons de démarrage.

4. Vérifier qu'un appui sur le bouton d'arrêt provoque toujours l'arrêt du magnétron jusqu'à la prochaine pression sur le bouton de démarrage ou turbo.