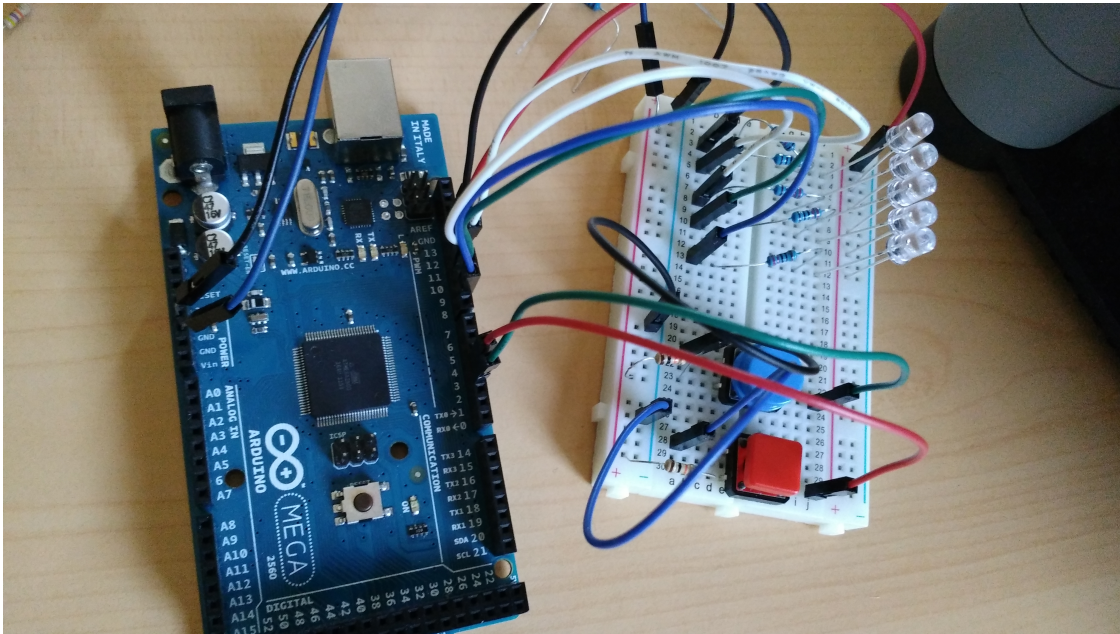


Du Lustre pour des jolies leds

Programmation et validation de systèmes embarqués

Laboratoire Verimag – Faites de la science !



1 Introduction

Au laboratoire Verimag, nous cherchons à aider les ingénieurs informaticiens à prouver automatiquement que leurs programmes sont corrects.

- La correction des programmes est cruciale, en particulier pour les **systèmes embarqués**, car des vies sont souvent en jeu.
- Mais elle est difficile à cause de l'**explosion combinatoire**. Une bonne illustration de ce phénomène est donnée par l'échiquier de Sissa, comportant 1 grain de riz sur case 1, 2 grains sur la case 2, 4 sur la case 3, 8 sur la case 4, etc. Un tel échiquier comporte $2^{64} - 1$ grains de riz, c'est à dire plus de 18 milliards de grains, soit l'équivalent de 1000 ans de production mondiale de riz actuellement. Or un programme de 64 cases (ou bits) est un petit programme.

Plusieurs approches sont possibles pour faire face à cette difficulté. Au laboratoire Verimag, les axes de recherche visant à résoudre ce type de problèmes concernent :

- la conception de **Langages** de programmation empêchant les programmeurs de faire certaines erreurs;
- l'**Analyse de programmes** : conception d'algorithmes (des programmes aussi) qui essaient de **prouver** qu'un autre programme est « **correct** ».
- l'automatisation des **tests** : conception d'algorithmes qui essaient de prouver qu'un programme n'est **pas** correct.

Pour illustrer tous ces axes de recherche, nous proposons une petite démonstration permettant de montrer

1. quelques exemples de programmes écrit en Lustre, un langage inventé au laboratoire et conçu pour mettre en oeuvre des systèmes embarqués critiques ;
2. comment un tel programme peut-être embarqué sur un système (ici, une carte Arduino) ;
3. comment prouver automatiquement des propriétés sur ces programmes ;
4. comment tester automatiquement ces programmes.

Il s'agit donc de valider est donc un petit système, composé d'un programme Lustre et d'une carte Arduino, comportant 2 entrées, contrôllées par un bouton rouge et un bouton bleu, et 5 sorties, représentées par 5 leds.

La propriété de sureté que l'on veut vérifier est « il est impossible d'allumer les 4 leds ». Pour chaque programme qu'on embarquera sur la carte, la question sera donc de savoir s'il est possible d'allumer les 5 leds (signifiant qu'il y a un problème) en jouant sur les entrées. Nous montreront ensuite comment des outils peuvent automatiser la preuve de correction ou de non correction.

2 Puzzle 1

- Bouton bleu : permute circulairement l'allumage des (jolies) leds
- Bouton rouge : inverse l'état des leds 2 et 4

```
node puzzle1 (red,blue:bool) returns (led1,led2,led3,led4,led5:bool);
let
  led1 = true  -> if blue then pre(led5) else pre(led1) ;
  led2 = false -> if blue then pre(led1) else if red then not(pre(led2)) else pre(led2) ;
  led3 = false -> if blue then pre(led2) else pre(led3) ;
  led4 = false -> if blue then pre(led3) else if red then not(pre(led4)) else pre(led4) ;
  led5 = true  -> if blue then pre(led4) else pre(led5) ;
tel
```

3 Puzzle 2

- Bouton bleu : idem
- Bouton rouge : inverse l'état de la led 5

4 Puzzle 3

- Bouton bleu : idem
- Bouton rouge : inverse l'état d'une led qui circule à chaque pression de boutons

5 Puzzle 4

- Bouton bleu : idem
- Bouton rouge : inverse l'état d'une led qui circule à chaque pression mais dans un sens différent selon la couleur du bouton pressé

6 Puzzle 5

```
true xor false = false xor true = true
false xor false = true xor true = false

true and true = true
true and false = false and true = false and false = false
```

- Bouton bleu => la led courante devient le « xor » des 2 leds autour
- Bouton red => la led courante devient le « and » des 2 leds autour
- la led courante est la première au début, puis la deuxième, etc.

7 Puzzle 6

- Bouton rouge => inverse l'état de la led 1
- Bouton bleu => allume la led suivant la première led éteinte (en partant de la led1)

```
node puzzle6(red,blue:bool)
returns (led1,led2,led3,led4,led5:bool);
var
  -- Xi = je suis la led qui suit la plus petite led éteinte
  X2, X3, X4, X5 : bool;
let
  X2, X3, X4, X5 = (true, F, F, F) ->
    if not(pre(led1)) then (true, F, F, F) else
    if not(pre(led2)) then (F, true, F, F) else
    if not(pre(led3)) then (F, F, true, F) else
    if not(pre(led4)) then (F, F, F, true) else (F, F, F, F);

  led1 = F -> if red then not(pre(led1)) else pre(led1);
  led2 = F -> if blue and X2 then not(pre(led2)) else pre(led2);
  led3 = F -> if blue and X3 then not(pre(led3)) else pre(led3);
  led4 = F -> if blue and X4 then not(pre(led4)) else pre(led4);
  led5 = F -> if blue and X5 then not(pre(led5)) else pre(led5);
tel
```