

# Introduction

---

IGL502/IGL752 – Techniques de vérification et de validation

Automne 2025



## Michael Blondin

@ michael.blondin@usherbrooke.ca

🌐 michaelblondin.com

📍 bureau D4-1024-1 au 1<sup>er</sup> étage

### **1<sup>er</sup> cycle** (13 personnes)

- 9 × informatique
- 2 × sciences de l'image et des médias numériques
- 1 × informatique de gestion
- 1 × sciences de l'information quantique

### **Cycles supérieurs** (4 personnes)

- 2 × maîtrise en génie logiciel
- 1 × microprog. en sciences
- 1 × doctorat en informatique

| <b>Mercredi</b> | <b>Jeudi</b>  |
|-----------------|---------------|
| 15h30 – 16h20   | 08h30 – 10h20 |
| D3-2031         | D3-2029       |

Notamment:

- S'initier aux fondements de la vérification formelle algorithmique
- Savoir modéliser des systèmes simples
- Être en mesure de spécifier des propriétés en logique temporelle
- Pouvoir vérifier algorithmiquement qu'un système satisfait sa spécification

## **MAT115 – Logique et mathématiques discrètes**

- Connaissances élémentaires en logique
- Automates finis et expressions régulières

## **IFT436 – Algorithmes et structures de données**

- Notions élémentaires d'algorithmique
- Parcours de graphes

- IGL501/710 – Méthodes formelles en génie logiciel
- IFT313 – Introduction aux langages formels
- IFT503/711 – Théorie du calcul
- Liens avec la planification en IA

- 0. Introduction
- 1. Systèmes de transitions
- 2. Logique temporelle linéaire (LTL)
- 3. Automates de Büchi
- 4. Automates de Büchi et LTL
- 5. Logique temporelle arborescente (CTL)
- 6. Vérification symbolique
- 7. Systèmes à pile
- 8. Systèmes à compteurs
- 9. Systèmes probabilistes

## 0. Introduction

2h

- Motivation
- Vérification formelle
- Exemples

## 1. Systèmes de transitions

1h

- Systèmes de transitions
- Exécutions
- Exemples

## 2. Logique temporelle linéaire (LTL)

6h

- Syntaxe
- Sémantique
- Spécifier des propriétés
- Équivalences

## 3. Automates de Büchi

3h

- Mots infinis
- Langages  $\omega$ -réguliers
- Automates de Büchi déterministes et non déterministes
- Automates de Büchi généralisés

## 4. Automates de Büchi et LTL

3h

- Produit d'automates
- Algorithmes de test du vide
- Vérification de formules LTL

## 5. Logique temporelle arborescente (CTL)

6h

- Syntaxe
- Sémantique
- Spécifier des propriétés
- Équivalences
- Vérification de formules CTL

## 6. Vérification symbolique

3h

- Diagrammes de décision binaire (DDB)
- Manipulation de DDB
- Vérification symbolique de formules CTL

## 7. Systèmes à pile

3h

- Modélisation de la récursion avec automates à piles
- Calcul de configurations accessibles
- Vérification basée sur l'accessibilité

## 8. Systèmes à compteurs

4,5h

- Modélisation de systèmes infinis avec réseaux de Petri
- Propriétés de réseaux de Petri
- Graphes de couverture

## 9. Systèmes probabilistes

4,5h

- Modélisation de systèmes proba. avec chaînes de Markov
- Accessibilité
- Logique temporelle probabiliste

## Cours à saveur théorique

- Pas de programmation
- Quelques démos d'outils de vérification

- Aujourd'hui: premier et dernier cours avec diaporama
- Notes électroniques disponibles en ligne
- Références complémentaires:
  - CHRISTEL BAIER ET JOOST-PIETER KATOEN: *Principles of Model Checking*. The MIT Press, 2008.
  - Articles scientifiques pour certains thèmes
  - Vidéos de 2020

|              |               |
|--------------|---------------|
| Devoirs      | 60% (5 × 12%) |
| Examen final | 40%           |

< 25% à l'examen  $\Rightarrow$  échec au cours

- 5 devoirs
- 1<sup>er</sup> cycle: équipe de deux ou individuel  
(mais au moins deux devoirs individuels)
- 2<sup>e</sup>/3<sup>e</sup> cycles: individuellement
- $\sim$ 2-3 semaines par devoir

## **Pas de retards permis:**

- Solutions présentées en classes
- Calendrier clairement annoncé
- Charge de travail testée depuis plusieurs années

## **Plagiat non toléré:**

- Chaque équipe rédige sa propre copie
- 0% en cas de plagiat + avertissement
- Demandez de l'aide si vous êtes en difficulté

Rétroaction non officielle vers la  
mi-session

| Sujets                                   | Devoirs  |
|------------------------------------------|----------|
| 1: Introduction                          | —        |
| 2: Logique temporelle linéaire (LTL)     | Devoir 1 |
| 3: LTL et systèmes de transitions        |          |
| 4: Automates de Büchi                    | Devoir 2 |
| 5: Automates de Büchi et LTL             |          |
| 6: Logique temporelle arborescente (CTL) | Devoir 3 |
| 7: CTL et systèmes de transitions        |          |
| 8: Semaine sans cours                    |          |
| 9: Relâche                               |          |

| Sujets                                    | Devoirs  |
|-------------------------------------------|----------|
| 10: Vérification symbolique               | Devoir 4 |
| 11: Systèmes à pile                       |          |
| 12: Systèmes à compteurs                  | Devoir 5 |
| 13: Sys. à compteurs / sys. probabilistes |          |
| 14: Systèmes probabilistes                |          |
| 15: Révision                              | —        |
| 16: Examen final                          | —        |
| 17: Examen final                          | —        |

Sur **rendez-vous**: bureau ou en ligne  
et  
**1h / semaine**

 [michaelblondin.com/igl752](https://michaelblondin.com/igl752)

# Introduction

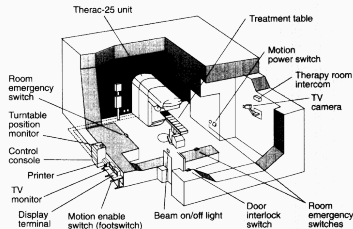
## – **Systèmes informatiques omniprésents**

- Logiciels
- Systèmes embarqués
- Protocoles de communication, etc.

- **Systèmes informatiques omniprésents**
- **Fiabilité de plus en plus cruciale**
  - Systèmes critiques (voitures, médical, centrales, etc.)
  - Failles de sécurité
  - Systèmes de masse, etc.

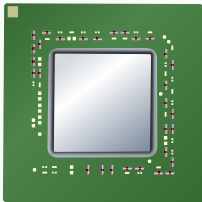
- **Systèmes informatiques omniprésents**
- **Fiabilité de plus en plus cruciale**
- **Systèmes de plus en plus complexes**
  - Systèmes concurrents
  - Systèmes distribués
  - Grande quantité de composantes, etc.

- **Systèmes informatiques omniprésents**
- **Fiabilité de plus en plus cruciale**
- **Systèmes de plus en plus complexes**
- **Erreurs peuvent être coûteuses**
  - Systèmes critiques: vies, sécurité, environnement, etc.
  - Production de masse: \$\$\$, temps



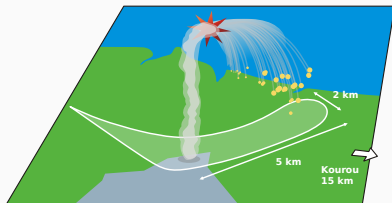
## Therac-25 — radiothérapie (1985-87)

- Concurrence critique lors d'opérations trop rapides
- Doses 100 fois plus grandes de radiations
- Décès et blessures graves chez 6 personnes



## **Bogue FDIV du Pentium (1994)**

- Bogue de l'unité à virgule flottante lors de divisions
- Dû à une table erronée de valeurs précalculées
- Coût d'environ 500 millions de dollars



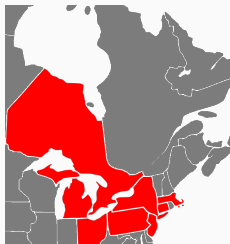
## Vol 501 du lanceur Ariane 5 (1996)

- Détruit après 36 secondes de vol
- Conversion de nombres en virgule flottante de 64 bits vers des entiers 16 bits (code réutilisé d'Ariane 4)
- Centaines de millions d'euros



## **Mars Pathfinder (1997)**

- Mission compromise due à un bogue concurrent
- Bogue corrigé à distance en deux semaines



## Panne d'électricité généralisée (2003)

- Concurrence critique qui a coupé une alarme d'urgence
- 10 millions personnes en Ontario et 45 millions aux É.-U.
- De 2 à 14 jours pour corriger la panne

```
// ...  
if ((err = SSLFreeBuffer(&hashCtx)) != 0)  
    goto fail;  
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)  
    goto fail;  
// ...  
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)  
    goto fail;  
    goto fail;  
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)  
    goto fail;  
err = sslRawVerify(...);  
// ...
```

## Bogue SSL d'Apple (2014)

- Bogue de gestion d'erreurs
- Vulnérable à des attaques

## Éviter les bogues

- Choix du langage de programmation
- Méthodes de travail
- Bonnes pratiques de développement

- + **Avantages:** règle le problème à la source
- **Désavantages:** ne détecte pas les bogues, ne garantit pas l'absence de bogues

## Relecture par les pairs

- Analyse statique et *manuelle* du code
- Détecte 31% à 93% des erreurs (médiane: 60%)
- Utilisé dans ~80% des projets

- + **Avantages:** simple, trouve la plupart des bogues
- **Désavantages:** difficile de détecter les erreurs dynamiques (systèmes concurrents, algorithmes, etc.)

## Test

- Test dynamique du code
- 30% à 50% des coûts de développement
- Utilisé dans essentiellement tous les projets

- + **Avantages:** simple, trouve la plupart des bogues
- **Désavantages:** incomplet, généralement manuel, s'applique difficilement aux systèmes concurrents

Permet de raisonner **rigoureusement** sur des systèmes à l'aide de notions mathématiques comme la logique

Permet de raisonner **rigoureusement** sur des systèmes à l'aide de notions mathématiques comme la logique

*« Formal methods should be part of the education of every computer scientist and software engineer, just as the appropriate branch of applied maths is a necessary part of the education of all other engineers. »*

*— Rapport de la NASA et FAA*

## Vérification déductive

- correction décrite en énoncés mathématiques
- on prouve *formellement* ces énoncés
- à l'aide d'outils: assistants de preuve, solveurs logique

+ **Avantages:** complet

– **Désavantages:** fastidieux, en bonne partie manuel, difficile pour les systèmes concurrents

## **Model checking**

- système représenté par un modèle mathématique
- exploration de *tous* les comportements du système
- fait de façon algorithmique

- + **Avantages:** automatique, trouve les bogues et prouve leur absence
- **Désavantages:** plus difficile avec les systèmes de grande taille (ou infinis), dépend de la validité du modèle

## ***Model checking***

- système représenté par un modèle mathématique
- exploration de *tous* les comportements du système
- fait de façon algorithmique

## Sujet de ce cours

## Ensemble de méthodes qui:

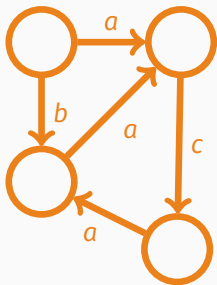
- vérifient si un système satisfait sa spécification
- fonctionnent automatiquement
- prouvent la correction ou identifient un contre-exemple

- **Système:** programme, circuit, protocole, etc.
- **Spécification:** ce que le système doit faire

- **Système:** programme, circuit, protocole, etc.
- **Spécification:** ce que le système doit faire

**Système**      satisfait      **spécification?**

- **Système:** programme, circuit, protocole, etc.
- **Spécification:** ce que le système doit faire

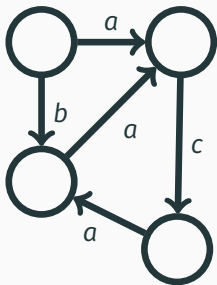


satisfait

**spécification?**

Modélisation

- **Système:** programme, circuit, protocole, etc.
- **Spécification:** ce que le système doit faire



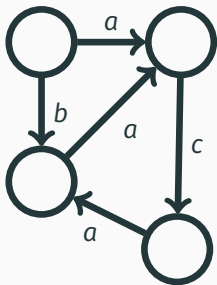
$\models$

$\varphi$

?

Propriété logique

Déterminer à l'aide d'un **algorithme**:



$\models$

$\varphi$

?

### **Avantages**

- s'applique à une grande variété de systèmes
- possible de vérifier seulement certaines propriétés
- ne considère pas seulement les erreurs probables
- contre-exemples lors de détection d'erreur
- automatique!

### **Désavantages**

- moins approprié pour systèmes orientés «données» plutôt que «contrôle»
- vérifie un modèle du système, pas le système lui-même
- se bute à des problèmes de complexité/décidabilité pour les systèmes de grande taille ou infinis

- **Vérification du standard IEEE Futurebus+:** détection de bogues après plusieurs années de développement; modèle de plus de  $10^{30}$  états
- **Outils de *model checking* pour C, C++, Java:** par ex. développés et utilisés par Microsoft et la NASA; utilisés pour vérifier des pilotes
- **Matériel:** par ex. 24% des erreurs d'un bus mémoire vérifié par IBM ont été découvertes par *model checking*; utilisé par d'autres grandes compagnies comme Intel

**Outils:** SPIN, NuSMV, PRISM, etc.



 **Infer:**

« Without Infer, multithreading in News Feed would not have been tenable. » — 



Clarke



Emerson



Sifakis

# **Prix Turing 2007**

## Un exemple

### Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

### Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux

# Un exemple

## Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

## Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux

**Possible d'atteindre les  
deux sections critiques simultanément?**

# Un exemple

## Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

## Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux



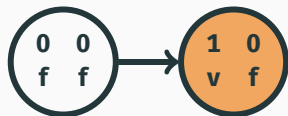
# Un exemple

## Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

## Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux



# Un exemple

## Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

## Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux



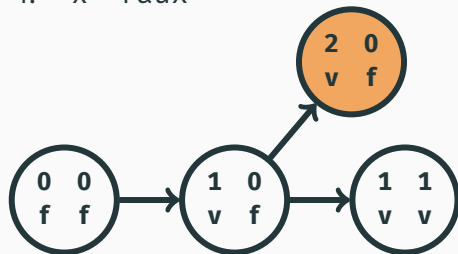
# Un exemple

## Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

## Processus B

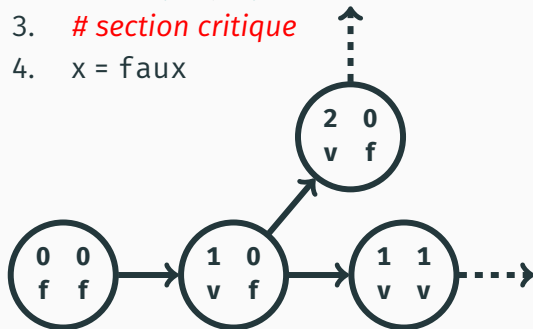
0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux



# Un exemple

## Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux



## Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux

## Un exemple

### Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

### Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux



**non accessible ?**

## Un exemple

### Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

### Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux



# Un exemple

## Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

## Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux



Oui!

## Un exemple

### Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

### Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux

**Processus B peut toujours  
atteindre sa section critique?**

## Un exemple

### Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

### Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux



## Un exemple

### Processus A

0. tant que vrai:
1. x = vrai
2. tant que y: pass
3. *# section critique*
4. x = faux

### Processus B

0. tant que vrai:
1. y = vrai
2. si x alors:
3. y = faux
4. tant que x: pass
5. goto 1
6. *# section critique*
7. y = faux

Non...



# Un exemple

## Processus A

```
0. tant que vrai:
1.  x = vrai
2.  tant que y: pass
3.  # section critique
4.  x = faux
```

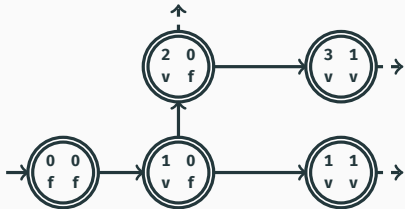
## Processus B

```
0. tant que vrai:
1.  y = vrai
2.  si x alors:
3.    y = faux
4.    tant que x: pass
5.    goto 1
6.  # section critique
7.  y = faux
```



Processus B peut toujours  
atteindre sa section critique

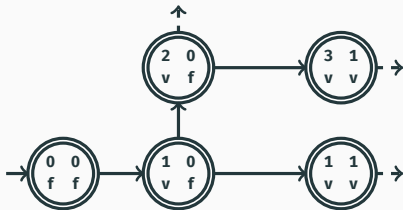
## Un exemple



$\models$

Processus B peut toujours  
atteindre sa section critique

## Un exemple

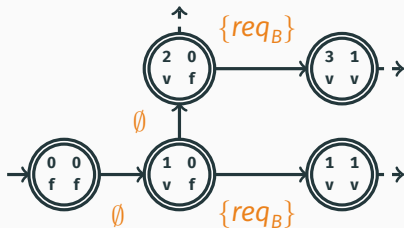


$\equiv$

Processus B peut toujours  
atteindre sa section critique

$$P = \{req_B, crit_B\}$$

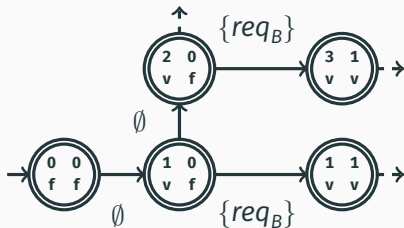
## Un exemple



Processus B peut toujours  
atteindre sa section critique

$$P = \{req_B, crit_B\}$$

## Un exemple

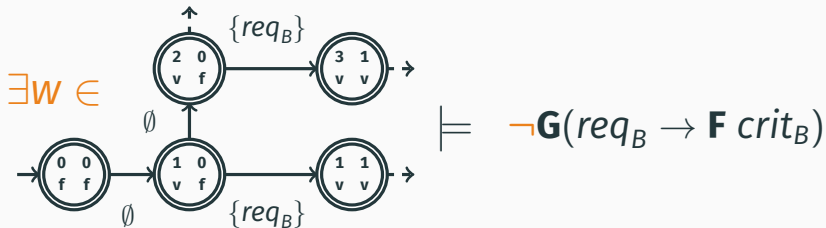


$\models$

$G(req_B \rightarrow F crit_B)$

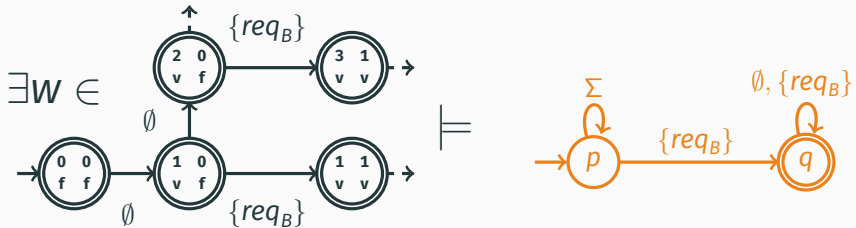
$$P = \{req_B, crit_B\}$$

## Un exemple



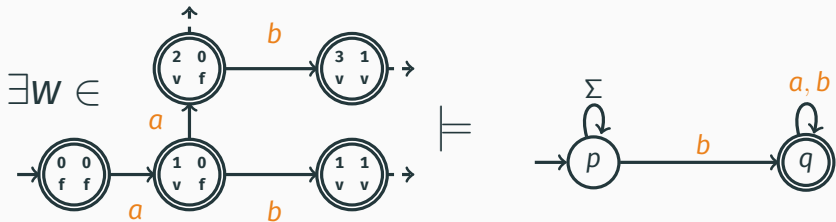
$$P = \{req_B, crit_B\}$$

## Un exemple



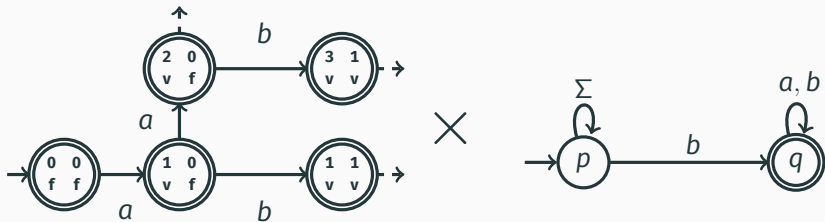
$$\Sigma = \{\emptyset, \{req_B\}, \{crit_B\}, \{req_B, crit_B\}\}$$

## Un exemple



$$\Sigma = \{a, b, c, d\}$$

## Un exemple



Système ne satisfait pas la propriété



Les deux automates acceptent un mot en commun

# Démonstration

- **Modélisation:** systèmes de transitions (sem. 2), sys. à pile (sem. 11), réseaux de Petri (sem. 12–13), sys. proba. (sem. 13–14)
- **Spécification:** LTL (sem. 2–3), CTL (sem. 6–7), PCTL (sem. 15)
- **Vérification:**
  - **Algorithmes:** LTL via automates de Büchi (sem. 4–5), CTL via saturation (sem. 7), sys. à piles (sem. 11), réseaux de Petri (sem. 13–14), chaînes de Markov (sem. 14–15)
  - **Struct. de données:** diagrammes de décision binaire (sem. 10)

**À mardi prochain!**