Cours "Génie Logiciel II"

Partie2: Méthodes formelles pour la spécification et la vérification de logiciels

Niveau: II2

Enseignante: Rim DRIRA

rim.drira@ensi-uma.tn

AU: 2016/2017

Objectifs du cours

- Introduire la spécification et la vérification formelle dans la démarche de développement d'un logiciel
- Apprendre une méthode formelle: la méthode B
- Apprendre deux techniques de vérification formelle
  - Preuve de théorèmes
  - Vérification de modèles ou model checking

Plan

- **Chapitre1:** Introduction aux méthodes formelles
- **Chapitre2:** La méthode B AMN (Abstract Machine Notation)
- **Chapitre3:** Vérification de modèles (ou model checking)

Pré requis

- Génie Logiciel 1
- Logiques formelles (propositions, prédicats, interprétation, Systèmes formels, axiomes, théorèmes)
- Théorie des langages (automates finis et composition)

Références

- [Abr96] J.-R. Abrial, “The B book”, Cambridge University Press, 1996.
- [Abr10], “Modeling in Event-B : System and Software Engineering”, Cambridge University Press, 2010.
- [Ger06]: Frédéric Gervais , "Combinaison de spécifications formelles pour la modélisation des systèmes d’information« , thèse de doctorat, Université de Sherbrooke, 2006.

http://cedric.cnam.fr/fichiers/RC1103.pdf

- [Jul08]: Jacques Julliand, cours "Spécification, Vérification et Test" , Université Franche-comté, 2008.

http://lifc.univ-fcomte.fr/~julliand/Lecon1_A_8SVT.pdf

- [Mos08]: Olfa Mosbahi, "Développement formel des systèmes automatisés« , thèse de doctorat, Université Tunis-ElManar, 2008.

http://pegase.scd.inpl-nancy.fr/theses/2008_MOSBAHI_O.pdf

- [Ngu98] H.P. Nguyen, "Dérivation de spécifications formelles B à partir de spécifications semi-formelles", Thèse de Doctorat, CNAM, France, 1998.

http://lacl.univ-paris12.fr/laleau/sourcePublis/Before2003/PHD-PHNguyen.pdf

- [SOM92]: Ian Sommerville, "*Le Génie Logiciel »,* Addison-Wesley, 1992. (bibliothèque de l’ENSI)

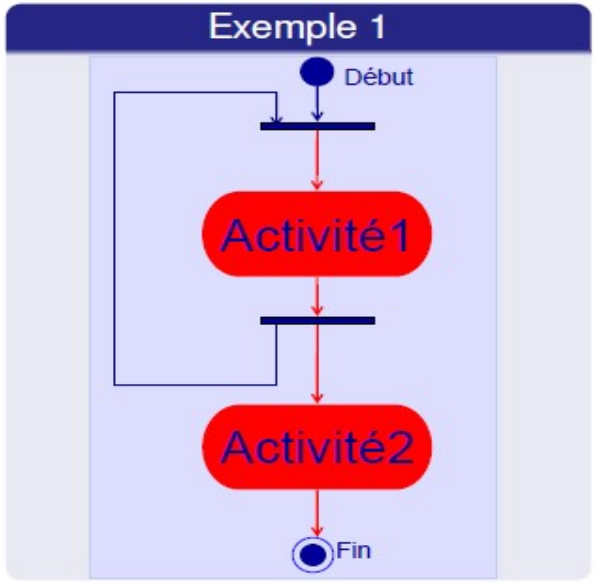
Références

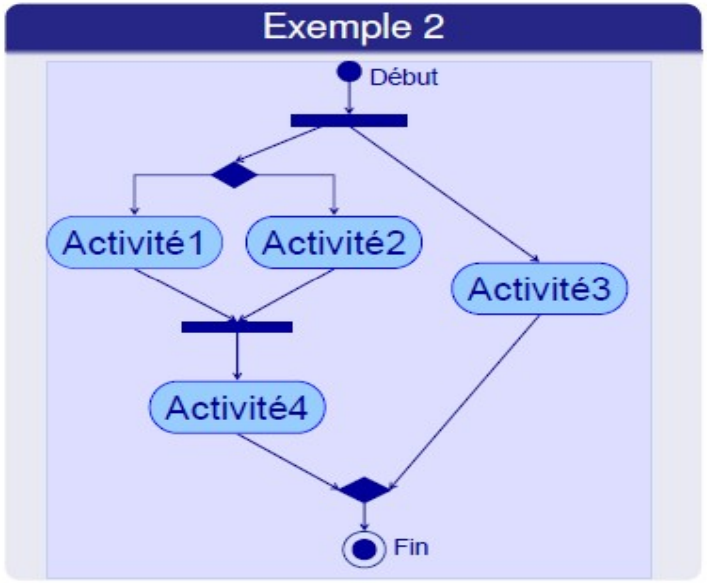
Autres Références Bibliographiques

- David Harel, Michal Politi, Modeling reactive systems with statecharts the statemateapproach, McGraw-Hill, 1998,
- Zohar Manna, Amir Pnueili, The Temporal Logic of Reactive and Concurrent Systems: Specification, Springer, 1992.
- Kevin Lano, The B language and method: a guide to practical formal development, Springer, 1996,
- Philippe Schnoebelen, Vérification de logiciels Techniques et outils du model-checking, Vuibert, 1999.
- Christel Baier, Joost-Pieter Katoen, Principles of Model-Checking, MIT Press Cambridge, London, 2008.

Pourquoi la vérification ?

Les diagrammes suivants contiennent des erreurs





L'entrée dans l'activité 4 dépond de la fin des deux activités 1 et 2 or une d'entre elles peut etre entrée (ou exclusif

Vérification d'absence de blocage

Utilisation d'une technique de vérification formelle

Le premier se bloque depuis le début de l'exécution (l'entrée dans l'activité 1 dépond de sa fin

Pourquoi la vérification ?

Calcul de la somme de deux nombres entiers en langage C

```
Int somme(int x, int y)
{
  int z;
  z = x+y;
  return z;
}
```

Ce code admet un problème :
si $x = 2^{31} - 1$ **et** $y = 1$

2^{31} ***n'existe pas***, **seulement** $2^{31}-1$ **en entier donc un débordement peut avoir lieu**

Pourquoi la vérification ?

Exemple3. On a besoin de vérifier que la saisie d'un utilisateur est une séquence alphanumérique qui se termine par ab1.
Le système de saisie à développer doit pouvoir vérifier la saisie et répondre positivement si la propriété est respectée et demande une autre saisie dans le cas contraire ou arrêter la saisie.

Ici le système à développer est un contrôleur. On doit développer un automate fini qui reconnait les séquences valides, vérifier cette reconnaissance et implémenter le parcours de cet automate.

Pourquoi la vérification ?

Exemple4. Le système informatique est un système de contrôle d'un chauffage
La propriété est liée à la température ambiante qui ne doit pas déborder d'un intervalle.
L'environnement est la température.
Le composant physique (commandé) est l'appareil de chauffe
Le contrôleur (qui est le logiciel que nous avons à développer) reçoit en entrée la valeur de la température et agit sur le composant physique si nécessaire pour chauffer ou refroidir
Le composant logique (le contrôleur) et le composant physique (contrôlé) doivent se comporter de façon à ce que la propriété soit vérifiée (température dans un intervalle).

Pourquoi la vérification ?

Exemple5. Un système de contrôle d'une barrière passage à niveau
Le composant physique est la barrière
Le composant de contrôle qui est le système informatique à développer agit sur la barrière pour qu'elle soit baissée ou relevée.
L'environnement est composé par les trains (avec leurs états arrive ou part ou rien)
Le contrôleur reçoit une information sur l'environnement et si un train arrive alors il doit commander pour baisser la barrière si elle est initialement relevée. Si tous les trains quittent alors le système commande la levée de la barrière.

Pourquoi la vérification ?

Exemple6. Accès à une section critique
Etant donnée une section critique (SC), un algorithme d'exclusion mutuelle doit satisfaire :
1- exclusion mutuelle (EM) sureté : si un processus est en SC alors aucun processus ne peut y être.
2- progrès (P) vivacité: si un groupe de processus demande d'entrer en SC alors l'un d'eux doit obtenir l'accès
3- equite (E) vivacité : tout processus demandant d'entrer en SC doit y entrer (au bout d'un certain temps) – pas de famine

Que faire alors ?

- **Développer des systèmes sûrs de fonctionnement.**
  - Ce sont des systèmes qui doivent répondre dans toutes les exécutions possibles aux propriétés exigées par l'utilisateur : absence de blocage, de traitement infini, de division par zéro…
  - Ce sont les **propriétés sémantiques dynamiques.**

- **Remarque:** L'analyseur sémantique dans un compilateur est plutôt limité **aux propriétés sémantiques statiques** : compatibilité des types, portée d'un identificateur, un sous programme….).

Que faire alors ?

- Le développement classique de tels systèmes nécessite des tests qui peuvent à la fin obliger le développeur à refaire le développement.

- D'autre part, les tests ne garantissent en aucun cas la fiabilité du système.…

- L'utilisation des méthodes formelles permet de remédier à ce problème. En effet ce sont des méthodes basées sur un fondement mathématique offrant la précision, l'absence d'ambiguité…..

- On peut ainsi partir des besoins spécifiés au départ, développer un modèle formel pour ce système (décrivant l'aspect comportemental, fonctionnel et structurel), exprimer formellement les exigences et effectuer la preuve.

Que faire alors ?

Le Système	satisfait-il	des exigences
Technique Sémantique		
Modèle D'automates	? |==	Formule logique
Ou bien		
Technique Syntaxique		
Formules Logiques	? |--	Formule logique

Il faut qu'on arrive, après ce cours, à modéliser, formaliser et vérifier

Chapitre1: Introduction aux méthodes formelles

Objectifs

- Sensibiliser à l'intérêt d'introduire les méthodes formelles dans le cycle de développement d'un logiciel
- Comprendre comment on peut introduire les méthodes formelles dans le cycle de développement d'un logiciel
- Définir les termes clés dans le domaine des méthodes formelles
- Présenter quelques formalismes de spécification formelle
- Introduire les techniques de la vérification formelle

Plan

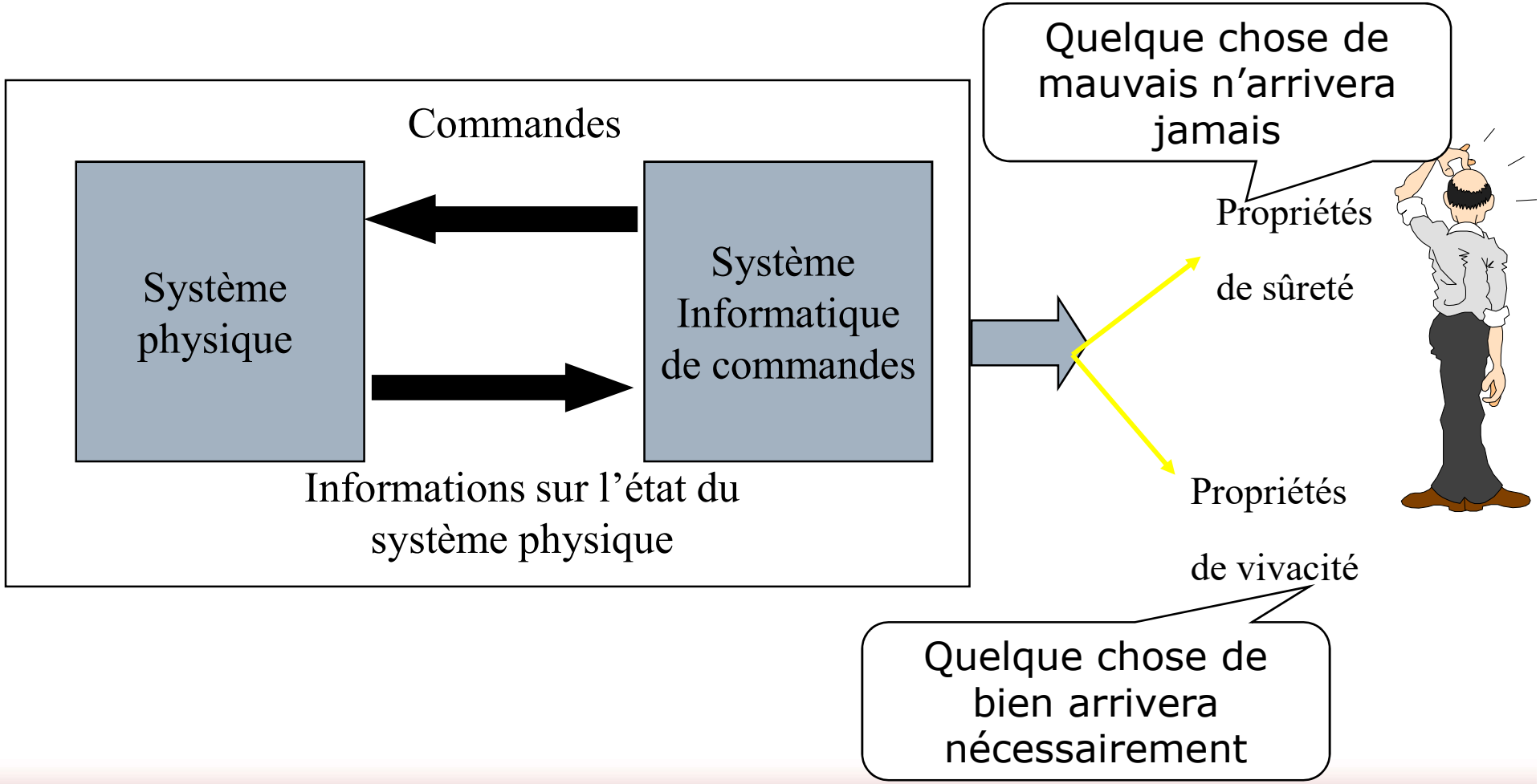
1. Systèmes automatisés sûrs de fonctionnement
2. Nécessité des méthodes formelles
3. Méthodes formelles et cycle de développement
4. Comportement, environnement, propriétés
5. Techniques de Vérification

Systèmes automatisés sûrs de fonctionnement

Systèmes automatisés sûrs de fonctionnement

- Un système est dit automatisé s'il exécute toujours le même cycle de travail pour lequel il a été programmé
- Les systèmes automatisés sûrs de fonctionnement exigent un niveau de sûreté et de fiabilité élevé
- Ce sont des systèmes qui doivent répondre dans toutes les exécutions possibles aux propriétés exigées par l'utilisateur

Systèmes automatisés sûrs de fonctionnement



Exemple d'un passage à niveau

- *Le système physique est la barrière*
- *Le composant de contrôle (le système informatique à développer) agit sur la barrière pour qu'elle soit baissée ou relevée*
- *Une propriété attendue de ce système (sûreté):* Le système ne doit pas permettre un accès simultané au trafic ferroviaire et au trafic routier

Le passage à niveau

Autres exemples

Le distributeur de billets

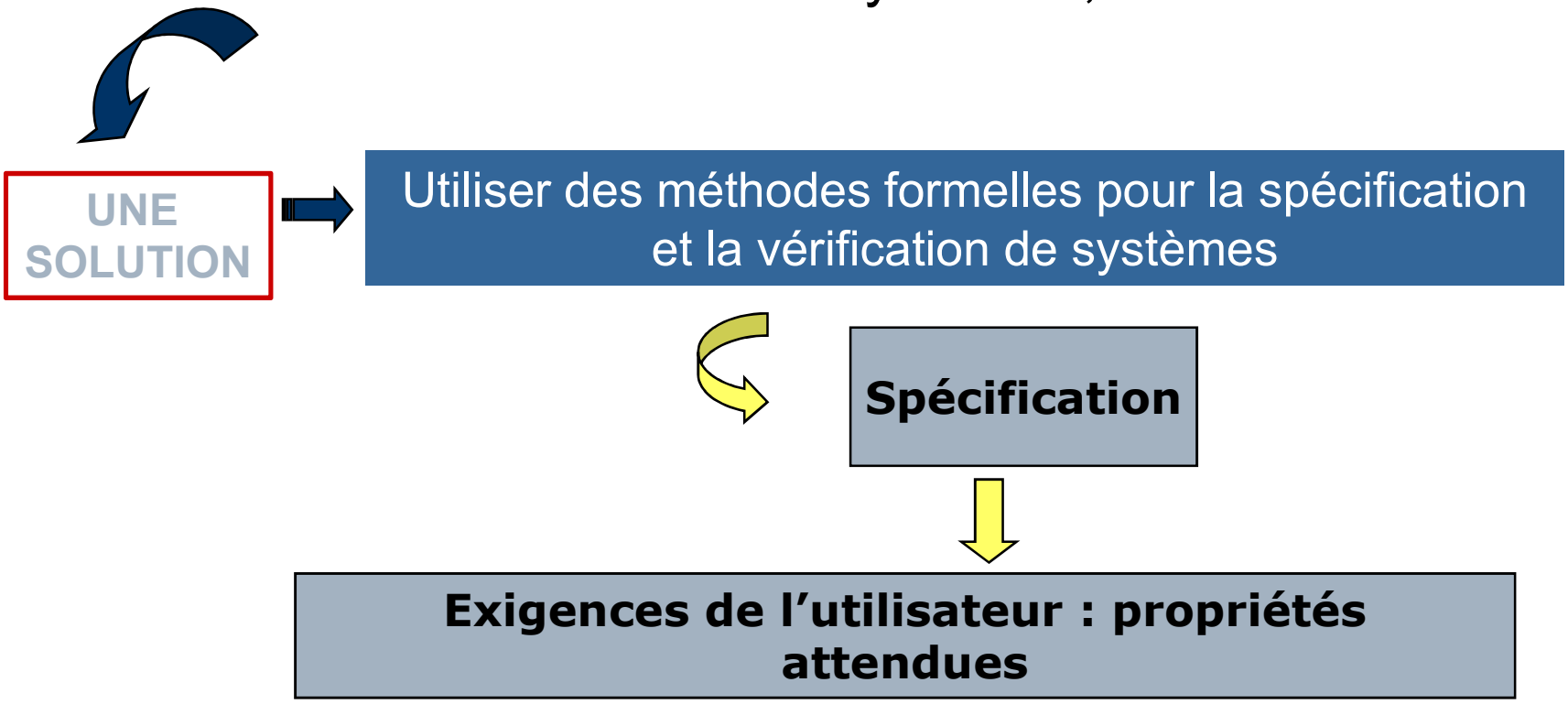
Les robots

Les feux de carrefour

La barrière de parking

Systèmes automatisés sûrs de fonctionnement

- Afin de réduire la complexité et assurer un bon fonctionnement de ces systèmes,



Nécessité des méthodes formelles

Bref historique

- Avant dans les années 70, l'utilisation des méthodes formelles se réalisait après l'implémentation.
  - Le programme est traduit en automate et la vérification se fait sur l'automate.
  - Cependant, la découverte d'une erreur à ce stade nécessite une mise à jour de toutes les étapes qui précèdent. Ce qui est, en, général, très couteux.

- Avec la complexité du code, les développeurs ont pensé à introduire la vérification dans une phase très avancée du cycle de développement.
  - Les modifications deviennent moins couteuses

- Les logiques (propositionnelles et de prédicats) n'étaient pas suffisantes pour décrire un comportement. On a pensé à introduire les logiques temporelles.

- Avec l'évolution des bases de données, la théorie des ensembles et la théorie des fonctions ont été introduites dans les approches formelles dans les années 90.

Nécessité des méthodes formelles

- Pour réduire la complexité et assurer un bon fonctionnement, l'utilisation des méthodes formelles apparaît comme une solution principale

- Elle consiste à Introduire une phase de **spécification formelle** dans le cycle de développement d'un logiciel
  - Le système est **ainsi spécifié et vérifié avant qu'il soit implémenté**
  - Le document de référence devient le document formel élaboré par la spécification

Rappel: Techniques de spécification

- Selon le degré de formalisation :
  - **Informel** : basée sur le langage naturel
  - **Semi-formel** : basée sur un langage structuré (graphique et/ou textuel) dont la sémantique est faible
  - **Formel** : basée sur un langage formel dont le vocabulaire, la syntaxe et la sémantique sont formels

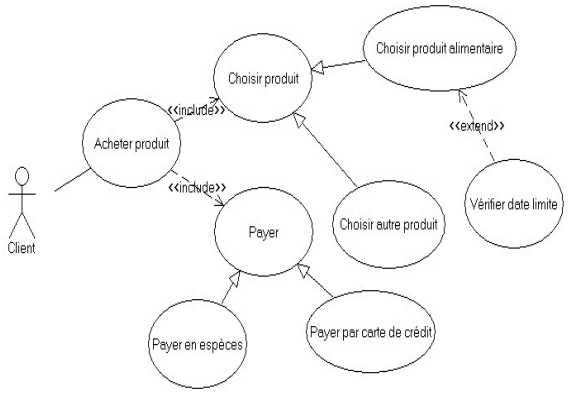
Spécification informelle

- **Exemple:** La vérification de la validité de la carte consiste à vérifier que la carte introduite par un utilisateur provient d'une banque reconnue, qu'elle est à jour, et qu'elle contient des informations appropriées ainsi que des détails sur les dates et les montants des précédents retraits.

- *Avantage :*
  - liberté pour l'auteur
- *Inconvénients* :
  - Ambiguïté
  - Flexibilité excessive
  - Modularité difficile

Spécification semi-formelle

- Se base sur une notation graphique :
  - Introduit un aspect formel mais diagrammes annotés généralement par du texte informel (pour cette raison, on dit semi-fomelle).
- Ces notations sont particulièrement pratiques pour fournir une vue d'ensemble, statique ou dynamique, d'un système ou d'un sous-système.
- Exemple: UML



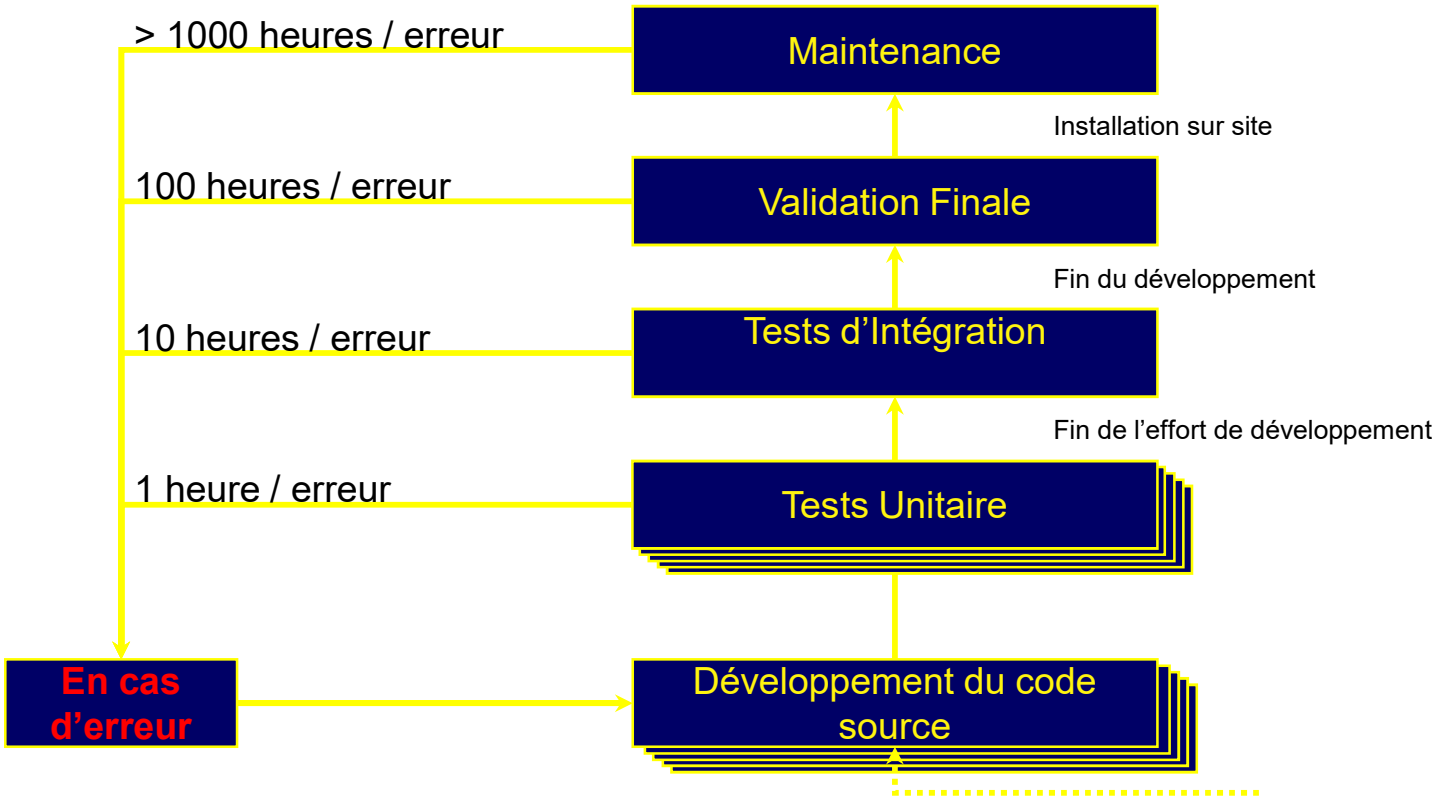
- ***Avantage :***
  - Compromis lisibilité-formalisme
  - Communication facilitée
- ***Inconvénient* :**
  - Vérification partielle (complétude, cohérence)
  - Sémantique faible

Difficultés rencontrées

- Validation difficile à réaliser
  - Vérifier que la spécification respecte les exigences utilisateurs (exemple: vérifier le respect des propriétés de sûreté)
- Vérification limitée (cohérence entre les étapes du développement)
- Difficultés du test qui prouve que l'application réalise l'énoncé
- Impossibilité de garantir l'absence d'erreurs: Non exhaustivité des cas envisagés lors des tests
- Coût élévé de la correction des erreurs surtout quand elles sont détectées tard dans le cycle de développement.
- **Ces difficultés s'accentuent et les conséquences sont plus catastrophiques pour les systèmes automatisés qui exigent une sûreté de fonctionnement**

Remarque: Coût d'une erreur 1/2

- Plus les erreurs sont détectées tard dans le cycle de développement, plus les coûts de correction sont élevés.



Remarque: Coût d'une erreur 2/2

- Plus une erreur est découverte tard plus elle coûte cher:
  - En phase de validation
    - L'équipe de développement n'est plus disponible
    - L'installation est reportée
  - En phase d'installation
    - Le produit ne fonctionne pas correctement
      - → perte du service
    - Le produit ne fonctionne pas du tout
      - → perte de la mission
    - Le produit cause des atteintes à la vie humaine
      - → Atteinte à l'image de l'entreprise et perte financière

Une solution: Spécification formelle

- Une spécification d'un logiciel est formelle si elle est exprimée avec un langage qui possède:
  - un vocabulaire et une syntaxe formellement définis;
  - une sémantique basée sur les mathématiques.
- Expression dans un langage formel du quoi d'un système à développer
- Offre une description **claire, précise et non ambiguë** du système sans référence aux détails d'implémentation:
  - La modélisation du fonctionnement du système (spécification du système).
  - La formulation rigoureuse des exigences sous la forme de propriétés attendues du système

Une solution: Spécification formelle

- **Avantages :**
  - Rigueur et précision des spécifications
    - Faciliter la validation
  - Automatiser la vérification
  - Prévenir les erreurs
- **Inconvénients :**
  - Nécessite une certaine qualification du client, utilisateurs et développeurs
  - Difficulté de communication

Spécification formelle et validation 1/3

- La validation consiste à se demander si le texte formel traduit le cahier des charges
- La spécification formelle permet de poser les bonnes questions et d'être plus rigoureux
  - ↳ La validation est facilitée
- La validation ne peut pas être automatisée

Spécification formelle et validation 2/3

Remarque:

- Lors du passage du cahier des charges vers la Spécification formelle, on risque toujours de:
  - Mal interpréter les besoins à la fois dans les propriétés et le système
  - Ne pas pouvoir formaliser toutes les propriétés qui couvrent la totalité des besoins,

Spécification formelle et validation 3/3

- **Quelques voies pour éviter ces problèmes:**
  - **Complémentarité entre validation et vérification**: Compléter la vérification formelle par de la validation à partir de jeux de tests,
  - **Hétérogénéité des langages formels** pour augmenter le pouvoir d'expression et combiner l'efficacité des méthodes de vérification,

    Pour plus de détails, voir [Ger06]

Méthodes Formelles

- Les méthodes formelles consistent à utiliser les mathématiques pour le développement de logiciels.
- **Les principales activités sont :**
  - l'écriture d'une spécification formelle ;
  - la preuve de certaines propriétés de cette spécification ;
    - Corriger la spécification si besoin
  - la construction de programmes en manipulant mathématiquement la spécification (Raffiner, transformer) ;
  - la vérification à l'aide de raisonnements mathématiques.
- 3.

Comparaison

Spécifications semi-formelles

- ☺ Modèles graphiques et intuitifs,
- ☺ Support de communication
- ☺ Editeurs visuels
- ☹ Absence d'une sémantique précise,
- ☹ Pas de vérification de propriétés (sûreté, vivacité, etc.)

Spécifications formelles

- ☺ Sémantique précise et rigoureuse
- ☺ Possibilité d'automatisation de la vérification de propriétés
- ☺ Faciliter la validation
- ☹ Difficiles à utiliser par les non familiarisés
- ☹ Manque de support de communication

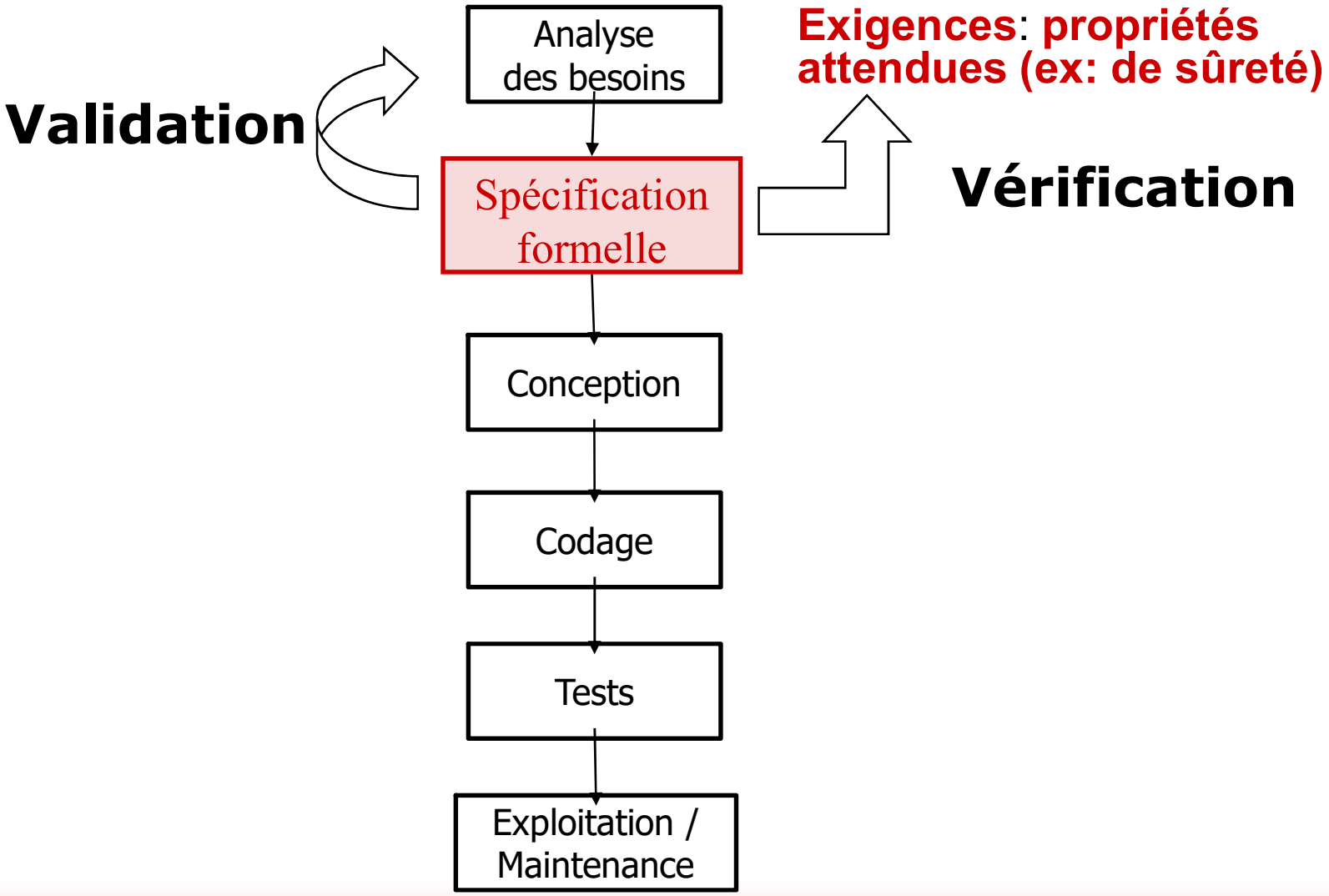
Voir [Ngu98] pour des détails sur les possibilités de dériver des spécifications formelles à partir de spécifications semi-formelles

Méthodes formelles et cycle de développement

Méthodes formelles et cycle de développement

- Introduire une phase de **spécification formelle**
  - Le document de référence devient le document formel élaboré par la spécification
- Intégrer **la vérification formelle** dans les démarches de conception de logiciels.
  - Le système est ainsi spécifié et vérifié avant qu'il soit implémenté

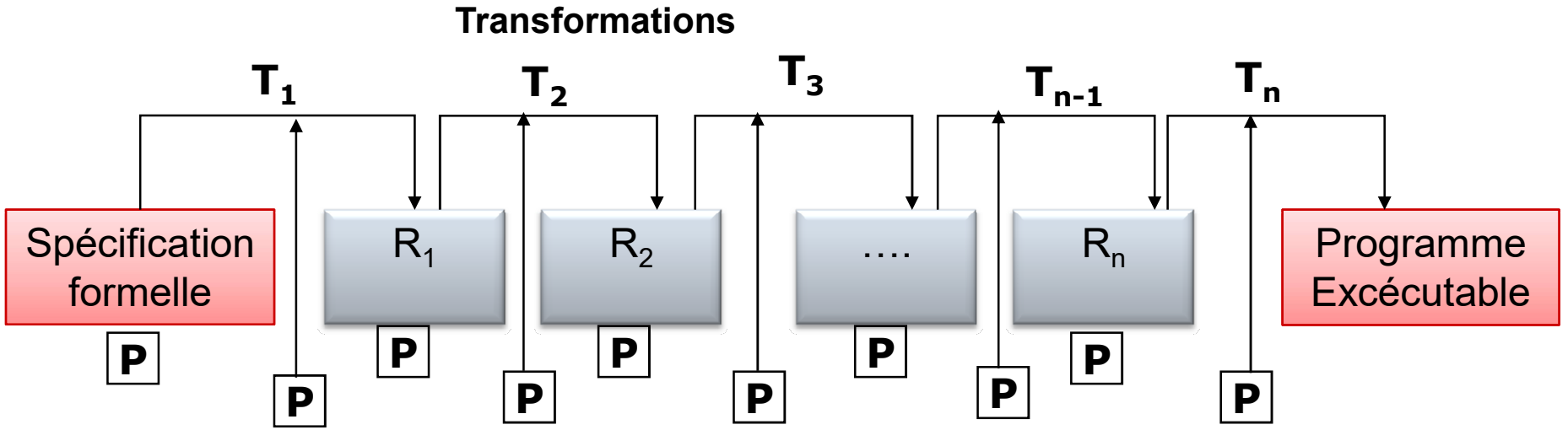
Méthodes formelles et cycle de développement



Méthodes formelles et cycle de développement

- **Les méthodes formelles ne sont pas limitées à la spécification mais couvent aussi:**
  - **Conception:** par transformation formelle on passe de la spécification à du pseudo-code
  - **Codage:** on peut générer automatiquement du code exécutable
  - **Preuve:** On prouve à chaque étape le respect des spécifications initiales

Développement par transformation formelle [Som92]



Si la spécification satisfait les propriétés et l'implémentation traduit la spécification alors l'implémentation satisfait aussi les propriétés

Formalismes pour représenter des spécifications

- les langages dérivés de la logique classique pour exprimer des systèmes transformationnels,
- les logiques temporelles pour exprimer des propriétés dynamiques de sûreté et de vivacité des systèmes réactifs,
- les langages logico-ensemblistes comme Z, VDM et B pour décrire et vérifier des propriétés statiques sur les états des systèmes,
- les réseaux de Petri, les automates communicants, LOTOS pour modéliser des systèmes concurrents avec ou sans partage de variables,
- les automates temporisés pour modéliser des systèmes temps réels

Modélisation des exigences: Propriétés attendues

Types de propriétés

- Sûreté (Safety) Invariants
  - Quelque chose de mauvais n'arrivera jamais
    - pas d'accès simultané de A et B à la section critique
- Vivacité (Liveness)
  - Quelque chose de bien arrivera nécessairement
    - si A veut entrer à la section, alors inévitablement il aura cet accès

Types de propriétés

Fatalité

- Énonce que, sous certaines conditions quelque chose de bien finira par avoir lieu au moins une fois à partir d'un certain état.
- Dans ce cas, on se situe dans la classe des propriétés de vivacité.

Equité

- Énonce que, sous certaines conditions, quelque chose aura lieu un nombre infini de fois.

Atteignabilité

- Ces propriétés énoncent qu'une certaine situation peut être atteinte.
- Nous pouvons exprimer aussi que quelque chose n'est jamais atteignable et nous parlons alors **d'inatteignabilité**.
  - Dans ce cas, on se situe dans la classe des propriétés de sûreté.

Exemple : Contrôleur de feux de circulation d'un carrefour

- **Problème:** spécifier le fonctionnement d'un contrôleur de feux tricolores d'un carrefour.
- Les feux peuvent être :
  - Hors service (hs) : tous les feux sont au jaune
  - En service (es) : ils évoluent selon

    rouge ⇝ vert ⇝ jaune ⇝ rouge
  - Lorsque le feu est au rouge sur une voie, les véhicules de cette voie ne peuvent s'engager dans le carrefour

Exemple : Contrôleur de feux de circulation d'un carrefour

- **Propriétés souhaitées:**
  - **Condition de sûreté (safety) :** les véhicules ne peuvent s'engager dans les deux voies simultanément.
  - **Condition de vivacité (liveness) :** les véhicules ne sont pas bloqués infiniment sur l'une des (ou les deux) voies.
- Ces propriétés sont décrites par :

$((feuA=rouge)\wedge(feuB\neq rouge))\vee((feuA\neq rouge)\wedge(feuB=rouge))$

Techniques de vérification

Techniques de vérification

Deux approches de vérification:

- Preuve de théorèmes (vérification syntaxique)
- Vérification de modèles ou model checking (vérification sémantique)

Techniques de vérification

Preuve

- consiste à prouver des propriétés à partir des axiomes du système et d'un ensemble de règles d'inférence.
- très puissantes
- elles traitent des systèmes à nombre d'états infinis
- elles sont indécidables dans le cas général,
  - ☞ difficiles à automatiser.

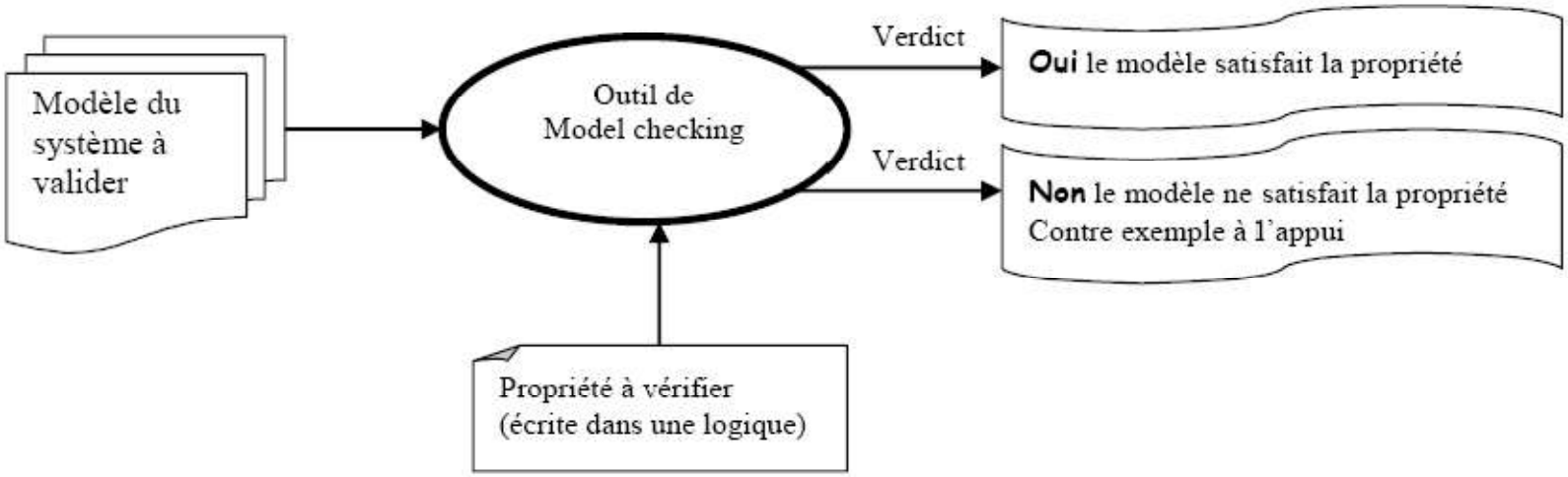
Techniques de vérification

Model checking

- permet de savoir si un automate donné vérifie une formule temporelle donnée
- consiste à parcourir exhaustivement l'espace d''etats d'un modèle fini du système à vérifier.
- Les propriétés à vérifier sont généralement exprimées en logique temporelle.
- Un contre exemple est généré lorsque la propriété n'est pas vérifiée par le modèle considéré.
- entièrement automatiques
- limitées aux systèmes ayant un nombre fini d'états
  - ☞ posent des problèmes en temps de traitement et en espace mémoire liés à l'explosion combinatoire en nombre d'états.

Techniques de vérification

Principe simplifié du model-checking



Une précision: TEST vs Preuve

- Un test qui mène à une exécution incorrecte apporte une information mais un test correct n'apporte aucune information

- Une preuve correcte permet d'indiquer qu'une propriété P est vérifiée, alors que l'impossibilité de faire une preuve ne renseigne pas sur l'anomalie

Comparaison

Preuve

- ☺ Applicable aux systèmes à nombre infini d'états
- ☹ Difficulté d'automatiser les preuves de propriétés dynamiques
- ☹ En cas d'échec de la preuve automatique, l'utilisateur doit déterminer si c'est une erreur ou une difficulté de preuve.
- ☹ L'utilisateur doit guider le prouveur.

Model checking

- ☺ Simple à utiliser
- ☺ Automatisation de la vérification des propriétés dynamiques
- ☹ Limité par l'explosion combinatoire du graphe d'état (problème de place mémoire et de temps de calcul).
- ☹ Limité aux automates à nombre fini d'états.

Fin chapitre1