Outils pour la sûreté de fonctionnement

Vous avez dit « vérification »?

Nicolas Stouls
Université de Lyon, INRIA
INSA-Lyon, CITI, F-69621, France
Nicolas.Stouls@insa-lyon.fr

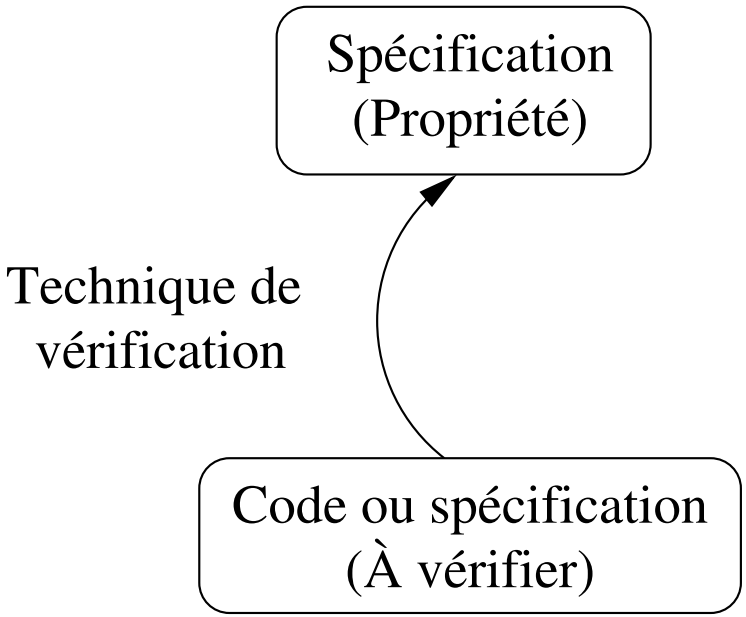
10/8/11

Introduction

Vérification

Que vérifier ?

- Un programme par rapport à une spécification
- Une spécification par rapport à une spécification plus générale



Problèmes

- Comment vérifier ?
- Langage de spécification ?

Distinction

- Génie logiciel « classique »
- Méthodes formelles

Vérification

Génie logiciel

Science de la maîtrise des coûts, de la complexité et de la qualité des développements.

Méthodes formelles

- Méthodes de génie logiciel
- Haut niveau de confiance
- Doit réunir 3 éléments :
  - Langage de spécification *(Expressif + sémantique bien définie)*
  - Technique(s) de vérification *(Règles de raisonnement)*
  - Outil *(Correcte utilisation des règles)*

Spécification ?

- Proposition de définition :

  Description faite sans se préoccuper
  des contraintes de bas niveau

- Spécification $\Leftrightarrow$ modèle
- Attention à la surcharge de « modèle » :
  - Mathématiques:
    Description d'une solution d'un système de contraintes
  - Informatique :
    Description abstraite d'un système ($\Leftrightarrow$ spécification)

Principes des modèles/spécifications

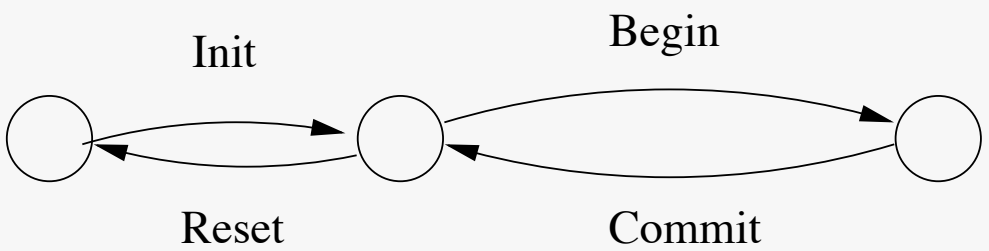
- Modéliser quoi : *un programme, un réseau, la construction d'un voiture ou les propriétés de résistance thermique du choux fleur de Hongrie occidentale.*
- Pourquoi faire ces modèles:
  - Communiquer sur le système
  - Vérifier la conformité modèle-système
- Langage utilisé : *dépend du système modélisé et des objectifs de vérification*
- Méthode utilisée : *dépend du système modélisé et des objectifs de vérification*

5 exemples concrets de langages de spécification

Graphique

- Comportemental *(comportements et espace d'états)*

Textuel

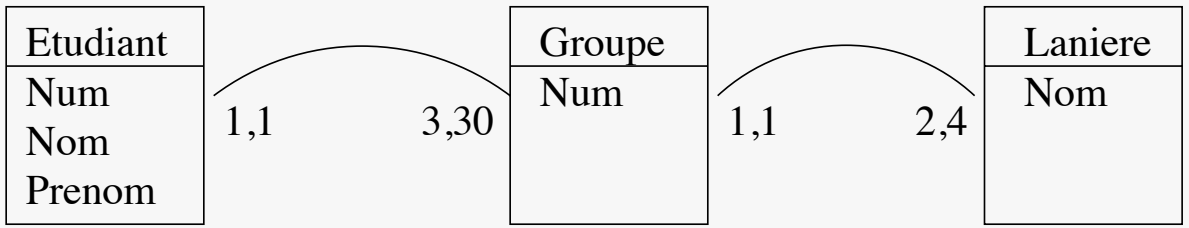


5 exemples concrets de langages de spécification

Graphique

- Comportemental *(comportements et espace d'états)*
- Organisationnel *(dépendance et répartition des données)*

Textuel



5 exemples concrets de langages de spécification

Graphique

- Comportemental *(comportements et espace d'états)*
- Organisationnel *(dépendance et répartition des données)*

Textuel

- Algèbre de processus *(description comportementale)*

$$\mathcal{A} \triangleq x' = x + 1 \ \wedge \ y' = y$$
$$\mathcal{B} \triangleq y' = y + 1 \ \wedge \ x' = x$$
$$\phi \triangleq (x = 0) \ \wedge \ (y = 0) \ \wedge \ \Box[\mathcal{A} \ \vee \ \mathcal{B}]$$

5 exemples concrets de langages de spécification

Graphique

- Comportemental *(comportements et espace d'états)*
- Organisationnel *(dépendance et répartition des données)*

Textuel

- Algèbre de processus *(description comportementale)*
- Fonctionnel *(description de la sortie en terme des entrées)*

$$\backslash\textbf{ensures}\ \forall i \in 1..9 \Rightarrow \backslash result[i] \leq \backslash result[i+1]$$

5 exemples concrets de langages de spécification

Graphique

- Comportemental *(comportements et espace d'états)*
- Organisationnel *(dépendance et répartition des données)*

Textuel

- Algèbre de processus *(description comportementale)*
- Fonctionnel *(description de la sortie en terme des entrées)*
- Transformationnel *(orienté données)*

$$
\begin{array}{l}
res \longleftarrow sort(t) \mathrel{\hat{=}} \textbf{PRE } t \in \mathbb{N} \nrightarrow \mathbb{N} \textbf{ THEN} \\
\quad \textbf{ANY } T2 \textbf{ WHERE } T2 \in \mathrm{dom}(t) \rightarrow \mathrm{ran}(t) \\
\qquad \wedge\ \exists f \cdot (f \in \mathrm{dom}(t) \rightarrowtail \mathrm{dom}(t) \Rightarrow (f; T2) = t) \\
\qquad \wedge\ \forall i, j \cdot (i \in \mathrm{dom}(T2)\ \wedge\ j \in \mathrm{dom}(T2)\ \wedge\ i \leq j \Rightarrow T2[i] \leq T2[j]) \\
\quad \textbf{THEN } res := T2 \textbf{ END END}
\end{array}
$$

Formel/Semi-formel/informel

Langage Formel/Semi-formel/informel ?

- Informel: *Langage contenant des ambigüités*
  Ex : Français, Anglais
- Formel: *Langage mathématiquement bien défini*
  Ex : Z, B
- Semi-formel: *Langage artificiel contenant des ambigüités*
  Ex : Statecharts, C

Méthode formelle ?

- Langage formel
- Technique(s) de vérification
- Outil vérifiant la correcte application des règles

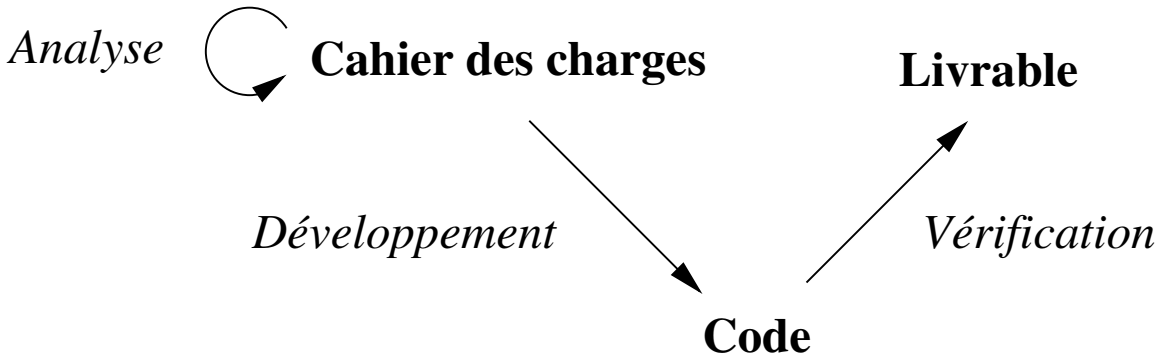
Langages de spécification

Finalement ?

- Sert à décrire un ensemble de propriétés
- Tout langage peut être utilisé
- Seules contraintes :
  - Suffisamment expressif
  - Non ambiguë ? *(≠ non-déterministe)*
- Un sous ensemble de C pourrait être un langage formel
- Le français aussi ....

Vérification/validation

Quand et pourquoi ?



Durant la phase d'analyse

Assistance à la définition des besoins

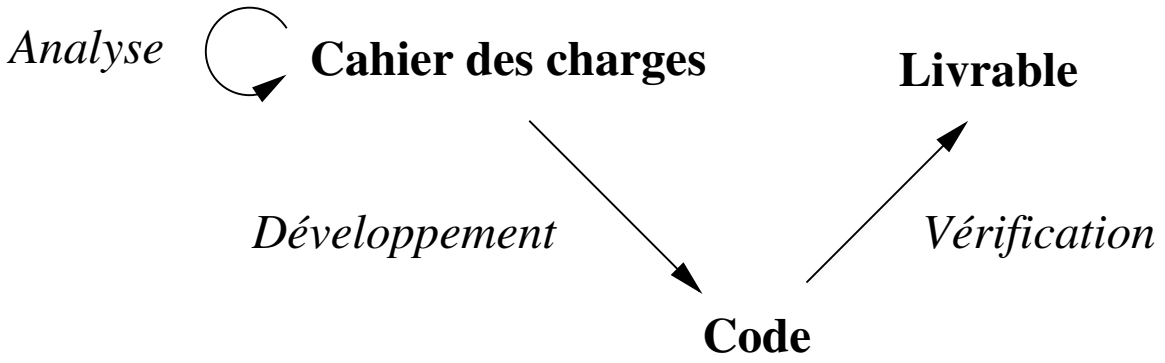
Pré-requis Cahier des charges en langue naturelle

Objectif Lever les ambiguïtés et les incohérences

Exemple GL UML : *description graphique d'un système*

Exemple MF OCL : *propriétés logiques sur diagrammes UML*

Quand et pourquoi ?

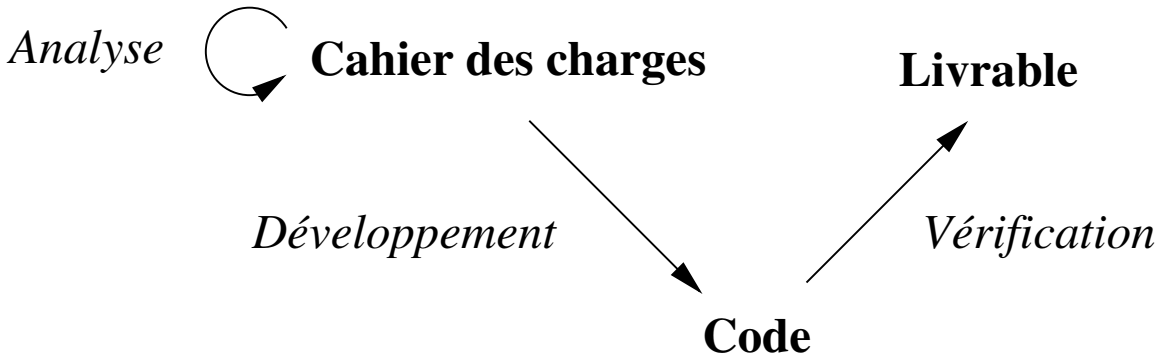


Durant la phase de développement

Logiciels corrects par construction

Pré-requis	Spécification *(pas de code)*
Objectif	Produire un code correspondant à la spécification
Exemple GL	IDM : *dérivation de code à partir d'un modèle*
Exemple MF	Raffinement : *description incrémentale*

Quand et pourquoi ?



Durant la phase de vérification

Vérification a posteriori

Pré-requis	Spécification et code
Objectif	Vérifier la cohérence code/spec
Exemple GL	Tests
Exemple MF	Simulation symbolique

Principales techniques de vérification

Exemple de 3 approches

- Model checking
- Interprétation abstraite
- Preuve de programme

Model checking

Principe : Existe-t-il un contre exemple ?

1. Construction de la représentation concrète du programme
2. Construction du négatif de la propriété
3. Faire produit synchrone des 2
4. Résultat : ensemble des traces violant la propriété

Bilan

Difficultés Taille de la représentation **concrète** du programme

Garanties Preuve de correction (si le processus termine)

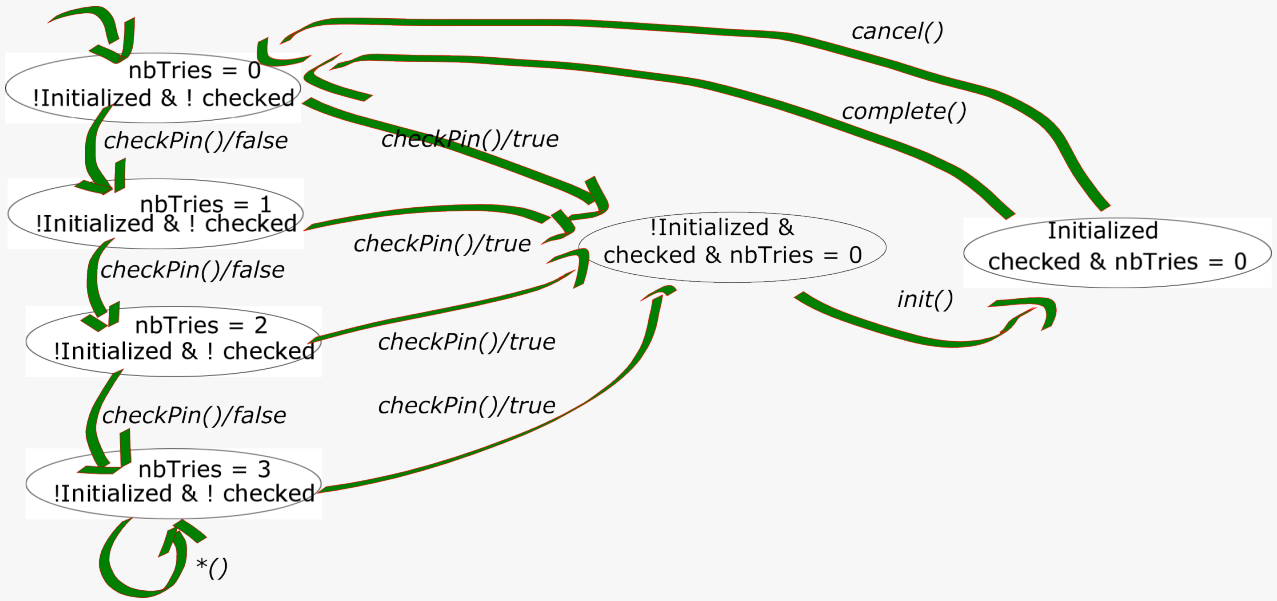
Model checking : exemple

Code à vérifier

```
if(terminalAuthentication()){
  int amount=amountChoice() ;
  int nbTries=0 ;
  while(nbTries<3 && !checkPIN()) {
    nbTries++ ;
  }
  if(nbTries<3){
    if(initTransaction(amount)){
      completeTransaction() ;
    } else {
      cancelTransaction() ;
    }
  }
}
```

Model checking : exemple

Flot de contrôle



Triche

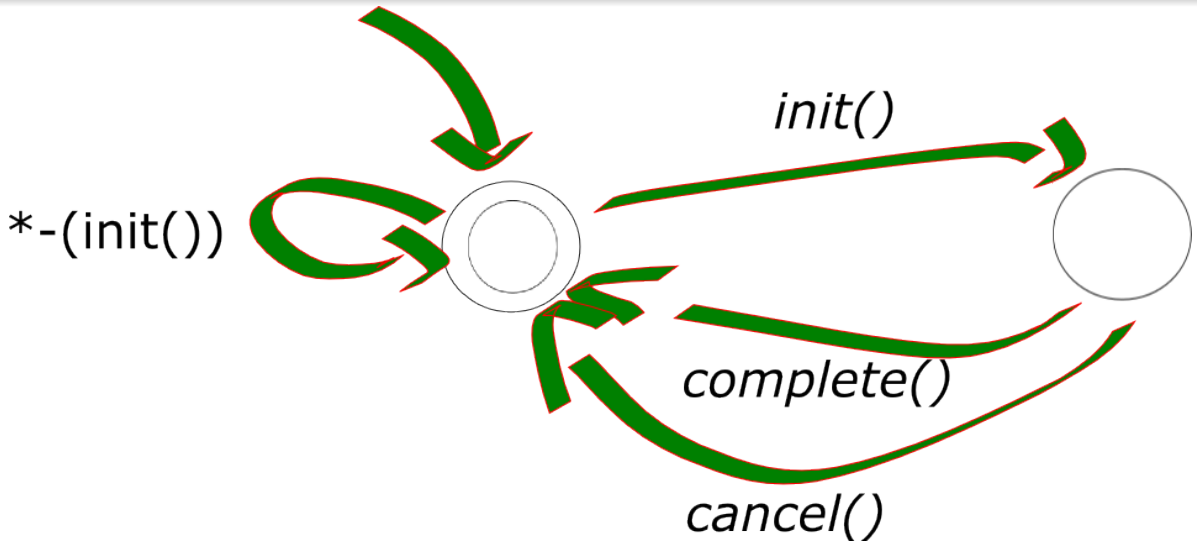
amount non représenté

Model checking : exemple

Propriété de protection contre l'arrachement

$$\varphi \triangleq \Box(\mathit{Init} \Rightarrow \bigcirc(\mathit{Comp} \vee \mathit{Cancel}))$$

Pour toujours, si *initTransaction* survient, alors il doit être immédiatement suivi de *completeTransaction* ou *cancelTransaction*.

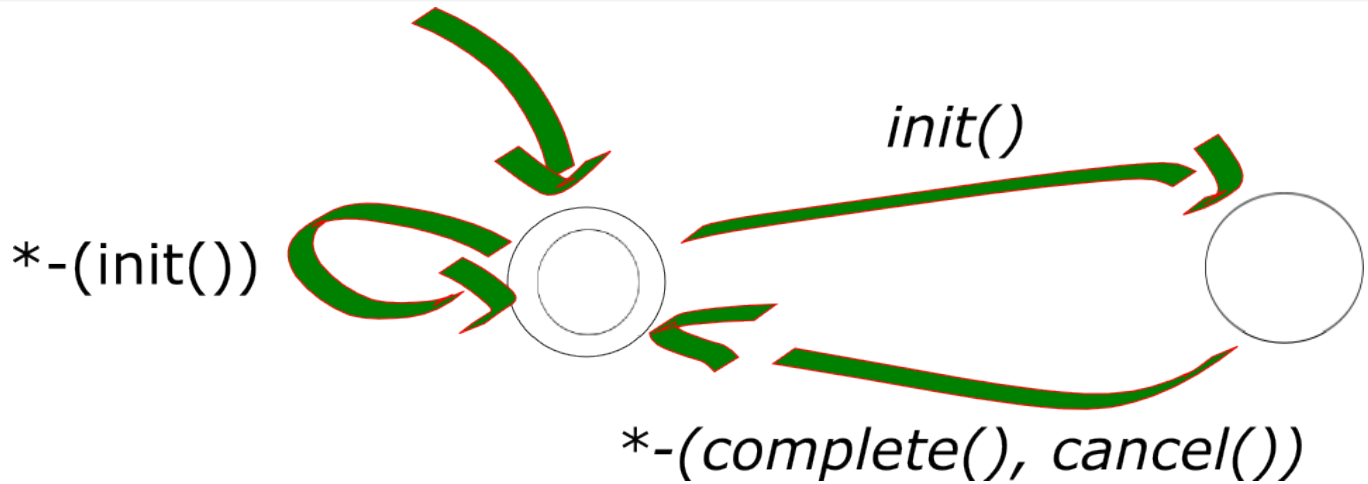


Model checking : exemple

Négation de la propriété

$$\neg\varphi \triangleq \diamond(\mathit{Init} \wedge \bigcirc(\neg\mathit{Comp} \wedge \neg\mathit{Cancel}))$$

Forcément un jour *initTransaction* surviendra et il ne sera immédiatement suivi ni de *completeTransaction*, ni de *cancelTransaction*.



Model checking : exemple

Vérification

- Toute trace d'exécution du programme vérifie la propriété ?
- Existe-t-il une trace vérifiant $\neg\varphi$?
  Si oui : c'est un contre exemple

Interprétation abstraite

Principe : Le même problème en plus simple ?

1. Construction d'une abstraction du programme
2. Vérification de la propriété sur l'abstraction

Bilan

Difficultés Choix de **bonnes** fonctions d'abstraction et de concrétisation (connexions de Galois), nécessite une seconde technique de vérification

Garanties Correction du programme par rapport à la propriété

Interprétation abstraite : exemple (non débordement)

Étude d'intervalle sur un tri par insertion

```
int[] T={...} ;
for(int i=1 ; i<T.length ; i++){
  int p=T[i] ;
  int j=i-1 ;
  while(j>=0 && T[j]>p){

    T[j+1]=T[j] ;
    j-- ;

  }

  T[j+1]=p ;
}
```

Interprétation abstraite : exemple (non débordement)

Étude d'intervalle sur un tri par insertion

```
int[] T={...} ;
for(int i=1 ; i<T.length ; i++){  // i ∈ [1..T.length − 1]
  int p=T[i] ;
  int j=i-1 ;
  while(j>=0 && T[j]>p){

    T[j+1]=T[j] ;
    j−− ;

  }

  T[j+1]=p ;
}
```

Interprétation abstraite : exemple (non débordement)

Étude d'intervalle sur un tri par insertion

```
int[] T={...} ;
for(int i=1 ; i<T.length ; i++){  // i ∈ [1..T.length − 1]
  int p=T[i] ;
  int j=i-1 ;  // i ∈ [1..T.length − 1] ∧ j ∈ [0..T.length − 2]
  while(j>=0 && T[j]>p){

    T[j+1]=T[j] ;
    j−− ;

  }

  T[j+1]=p ;
}
```

Interprétation abstraite : exemple (non débordement)

Étude d'intervalle sur un tri par insertion

```
int[] T={...} ;
for(int i=1 ; i<T.length ; i++){  // i ∈ [1..T.length − 1]
  int p=T[i] ;
  int j=i-1 ;  // i ∈ [1..T.length − 1] ∧ j ∈ [0..T.length − 2]
  while(j>=0 && T[j]>p){
    // i ∈ [1..T.length − 1] ∧ j ∈ [0..T.length − 2]
    T[j+1]=T[j] ;
    j−− ;

  }

  T[j+1]=p ;
}
```

Interprétation abstraite : exemple (non débordement)

Étude d'intervalle sur un tri par insertion

```
int[] T={...} ;
for(int i=1 ; i<T.length ; i++){  // i ∈ [1..T.length − 1]
  int p=T[i] ;
  int j=i-1 ;  // i ∈ [1..T.length − 1] ∧ j ∈ [0..T.length − 2]
  while(j>=0 && T[j]>p){
    // i ∈ [1..T.length − 1] ∧ j ∈ [0..T.length − 2]
    T[j+1]=T[j] ;
    j−− ;
    // i ∈ [1..T.length − 1] ∧ j ∈ [−1..T.length − 3]
  }

  T[j+1]=p ;
}
```

Interprétation abstraite : exemple (non débordement)

Étude d'intervalle sur un tri par insertion

```
int[] T={...} ;
for(int i=1 ; i<T.length ; i++){  // i ∈ [1..T.length − 1]
  int p=T[i] ;
  int j=i-1 ;  // i ∈ [1..T.length − 1] ∧ j ∈ [0..T.length − 2]
  while(j>=0 && T[j]>p){
    // i ∈ [1..T.length − 1] ∧ j ∈ [0..T.length − 2]
    T[j+1]=T[j] ;
    j−− ;
    // i ∈ [1..T.length − 1] ∧ j ∈ [−1..T.length − 3]
  }
  // i ∈ [1..T.length − 1] ∧ j ∈ [−1..T.length − 2]
  T[j+1]=p ;
}
```

Preuve de programme (Vue globale)

Principe :

- La propriété est elle cohérente ?
- La propriété est elle une abstraction du programme ?

Bilan

Difficultés	Indécidabilité de la logique du premier ordre (SAT)
Garanties	Correction du programme par rapport à la propriété

Preuve de programme : différentes approches

Le code existe, on veut le vérifier (Ex : Frama-C)

- Propriétés décrites dans le programme
- Chaque fonction / variable globale est annotée

Extraction du programme à partir de la preuve (Ex : Coq)

- La fonction est un théorème
- La preuve est le programme

Description par étape du programme (Ex : Méthode B)

- Descriptions de plus en plus précises du programme

Le code existe, on veut le vérifier (Frama-C)

- Propriétés décrites dans le programme
- Chaque fonction / variable globale est annotée
- On vérifie pour chaque fonction :
  - qu'elle respecte sa spécification
  - qu'elle ne génère pas d'erreur à l'exécution
- Exemple : Frama-C

Extraction du programme à partir de la preuve (Coq)

Principe

1. Description d'un théorème : c'est la fonction
2. preuve du théorème : c'est le programme

Approche constructiviste

- La preuve doit être en logique constructiviste
  - $(\neg\neg A) \not\equiv A$
  - Pas de raisonnement par l'absurde
- Isomorphisme de Curry-Howard
  - Équivalence programme/preuve
  - Traduction automatique de la preuve en programme

Coq récompensé par l'ACM

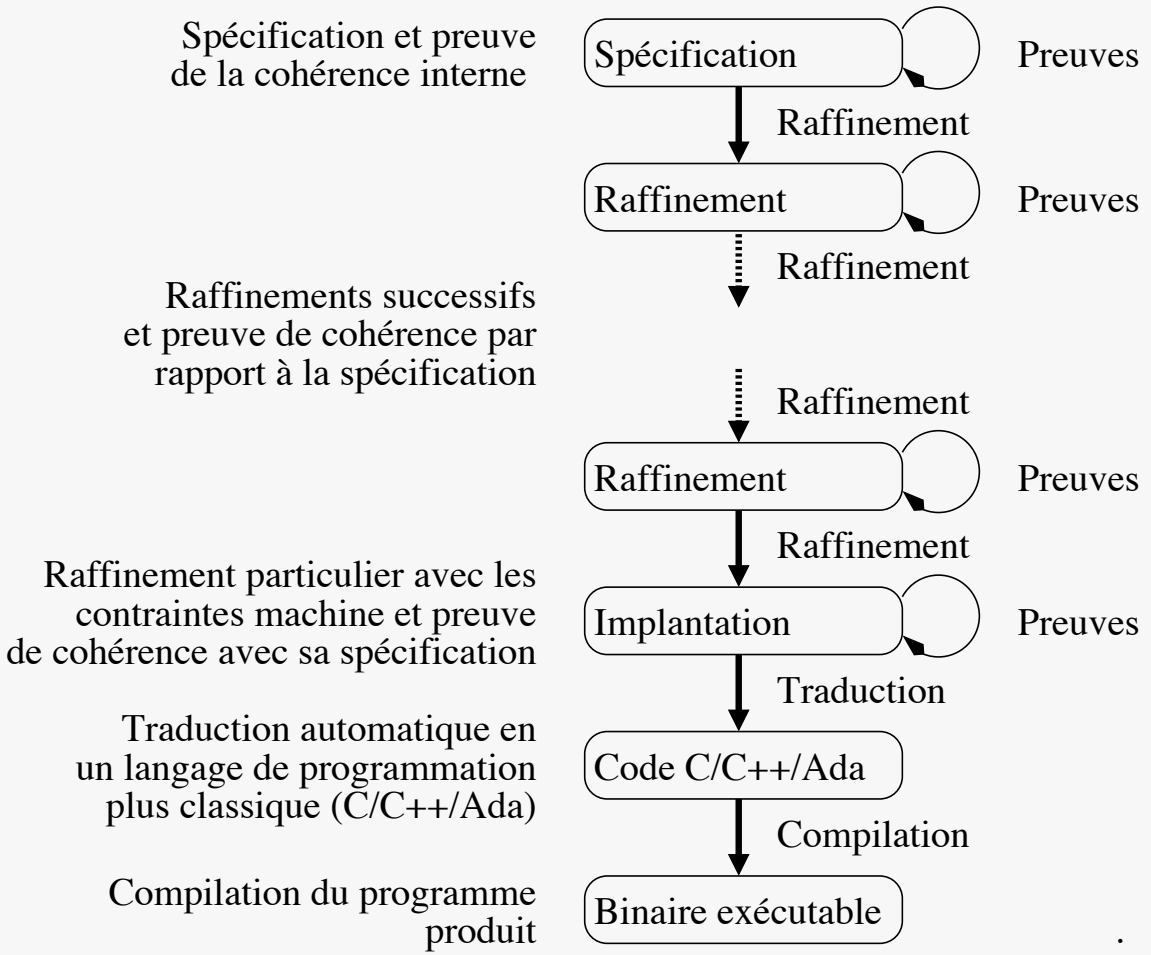
ACM ?

- Association for Computing Machinery
- Association à but non lucratif
- Mission : soutenir recherche & innovation en informatique

ACM SIGPLAN Programming Languages Software 2013 award

- Plus haute distinction de l'ACM (Soutenue par IBM)
- Exemple d'anciens récipiendaires de cette distinction :
  - Unix
  - TeX
  - TCP/IP
  - Java
  - Make

Description par étapes du programme : Méthode B



Bilan

Bilan

Bilan

- Modèle: *description de référence d'un système*
- Modèle $\Leftrightarrow$ spécification
- Formel / informel / semi-formel
- Confiance peut être acquise à tout moment
- Confiance peut être plus ou moins forte
- Qu'est-ce qui a été vérifié ? À définir.

Dans le cadre de ce cours

- Preuve (Méthode B)

Devoir à la maison 1/2

Travail à la maison : définition des règles

- Choisir un article parmi ceux proposés
- Faire l'analyse du travail : Résumé + avantages/inconvénients + 1 alternative possible (comment ? + bien/pas bien)
- Format imposé :
  - 2 à 3 pages
  - en anglais
  - en latex
  - au format *IEEE Transactions*[a].
  - au moins 2 articles dans les références

21/11/14 Rapport à envoyer à `nicolas.stouls@insa-lyon.fr`

a. `http://www.ieee.org/publications_standards/`

Devoir à la maison 2/2

Installez l'atelier B sur votre machine perso !

- `http://www.atelierb.eu/`
- apportez votre machine aux 2 prochains cours.