Introduction aux méthodes formelles

Module Développement Logiciel : Approche Formelle & à Objets

Pascal ANDRE

MIAGE
Université de Nantes

Master Miage M1

Plan

Contexte de l'intervention 1/2

Le contexte

1. Génie logiciel
2. Le développement du logiciel
3. Méthode de développement du logiciel
   - Modèles, produits
   - Processus
   - Validation, vérification
4. Qualité

Contexte de l'intervention 2/2

- De l'idée au code
  - exprimer
  - programmer → MODELES
  - vérifier
- avec méthode.
  - dans le bon ordre PROCESSUS
  - éviter l'anarchie → GESTION DE PROJET
  - travailler en groupe
- et qualité
  - modèles (correct, fiable, évolutif...)
  - processus (efficace, rentable...) → QUALITE

Introduction : méthodes formelles en Miage

1. Master MIAGe M1 Nantes
   - Le développement du logiciel
   - Les méthodes formelles : application à Z
   - Les méthodes à objets : du formel avec OCL
2. Master MIAGe M2 Nantes (Parcours IRSI)
   - B (BTool)
   - Algèbres de processus (LOTOS)

Vers un usage rationnel

- bonnes pratiques de spécification (précision)
- bonnes pratiques de conception (assertions, preuves)
- bonnes pratiques de programmation (tests)
- bonnes pratiques du processus (rigueur)

Introduction : méthodes formelles en Miage

Quelques références

- Développement du logiciel et CSI [AV01a, GMSB96]
- Systèmes formels [Lau86, WL88]
- Z [Spi94, WL88], B [Abr96]
- Méthodes formelles et Z
  [Mey92, BH96, Bow96, Jac97, AV01b, AV04]
- LOTOS, SDL [Tur93], CSP [Hoa85] ou CCS [Mil89]
- Spécifications algériques [Wir90]
- Automates [Arn92], les réseaux de PETRI [Bra83])


Jean-Raymond Abrial.
Spécifier ou comment matérialiser l'abstrait.
Technique et Science Informatique, 3(3):201–219, 1984.


Jean-Raymond Abrial.
The B-Book Assigning Programs to Meanings.
Cambridge University Press, 1996.
ISBN 0-521-49619-5.


André Arnold.
Systèmes de transitions finis et sémantique des processus communiquants.
Collection Etudes et recherches en informatique. Masson, 1992.
ISBN 2-225-82746-X.


Pascal André and Alain Vailly.
Conception de systèmes d'information ; Panorama des méthodes et des techniques, volume 1 of Collection Technosup.
Editions Ellipses, 2001.
ISBN 2-7298-0479-X.


Pascal André and Alain Vailly.
Spécification des logiciels ; Deux exemples de pratiques récentes : Z et UML, volume 2 of Collection Technosup.
Editions Ellipses, 2001.

ISBN 2-7298-0774-8.


Pascal André and Alain Vailly.
Exercices corrigés en langage Z ; Les spécifications formelles par la pratique, volume 4 of Collection Technosup.
Editions Ellipses, 2004.
ISBN 2-7298-1942-8.


Jonathan P. Bowen and Michael G. Hinchey, editors.
Applications of Formal Methods.
International Series in Computer Science. Prentice-Hall, 1 edition, 1996.
ISBN 0-13-832544-8.


Jonathan Bowen.
Formal Specification & Documentation using Z: A Case Study Approach.
International Thomson Publishing, 1996.
ISBN 1-85032-230-9.


G. W. Brams.
Réseaux de Petri : théorie et pratique.
Editions Masson, Paris, 1983.


Marie-Claude Gaudel, Bruno Marre, Françoise Schlienger, and Gilles Bernot.
Précis de génie logiciel.
Enseignement. Masson, 1996.

ISBN 2-225-85189-1.

Charles Antony Richard Hoare.
Communicating Sequential Processes.
International Series in Computer Science. Prentice-Hall, 1985.
ISBN 0-13-153289-8.

Jonathan Jacky.
The Way of Z: Practical Programming with Formal Methods.
Cambridge University Press, 1997.
ISBN 0-521-55041-6.

Cliff B Jones.
VDM, une méthode rigoureuse pour le développement du logiciel.
Masson, Paris, 2e edition, 1993.

Jean.-Louis Laurière.
Résolution de problèmes par l'Homme et la machine.
Eyrolles, 1986.

Bertrand Meyer.
Introduction to the Theory of Programming Languages.
International Series in Computer Science. Prentice Hall, 1992.
ISBN 0-13-498502-8.


Robin Milner.
Communication and Concurrency.
International Series in Computer Science. Prentice-Hall, 1989.
ISBN 0-13-115007-3.


Mike Spivey.
La notation Z.
Collection Méthodologies du logiciel. Editions Masson, 1994.
Traduit de l'anglais par Michel Lemoine, ISBN 2-225-84367-8.


Kenneth J. Turner, editor.
Using Formal Description Techniques, An introduction to Estelle, Lotos and SDL.
Wiley, 1993.
ISBN 0-471-93455-0.


Jeannette M. Wing.
A specifier's introduction to formal methods.
IEEE Computer, 23(9):8–24, 1990.


Martin Wirsing.
Algebraic Specification, volume B of
Handbook of Theoretical Computer Science, chapter 13, pages 675–788.
Elsevier, 1990.

Jan Van Leeuwen, Editor.

J. Woodcock and M. Loomes.
Software Engineering Mathematics.
Addison Wesley, 1988.

Plan

Quelques chiffres sur le GL

- abandon logiciels : 2/6 (de grande taille)
- retard $>$ 50% durée prévue livraison logiciels : 75%
- qualité = problématique coûteuse (60% des coûts totaux - Gardner)
- coût bugs & erreurs des logiciels
  - 59.5 MM$ par an, économie EU (estimation NIST 2002)
  - 175 MM$ en 2002 entreprises US (estimation NRC 2002) pour réparer les dommages causés par des logiciels défectueux
- (quelques) catastrophes :
  - Therac 25, '85-'87 : 6 patients irradiés, 2 morts
  - Syst. Bagages Aéroport Denver, '95 : 16mois, 3.2 Mds$
  - Ariane 5 - vol 88/501, '96: 40s de vol, destr., 850 M$
  - Mars Climate Orbiter & Mars Polar Lander, '99 : destr.

Source: P. Poizat, Borland

Bugs et qualité

- bug = dysfonctionnement logiciel
- problème : tous niveaux
  - processeur, ex: bug virgule flottante des processeurs Pentium
  - codage, ex : bug an 2000 (années codées sur deux chiffres), 2038 (langage C codé sur 32 bits)
  - logiciel, ex : crash Ariane
  - faille sécurité, ex : CB

Source: wikipedia, ...

Bugs et qualité

- bug = dysfonctionnement logiciel
- problème : tous niveaux
  - processeur, ex: bug virgule flottante des processeurs Pentium
  - codage, ex : bug an 2000 (années codées sur deux chiffres), 2038 (langage C codé sur 32 bits)
  - logiciel, ex : crash Ariane
  - faille sécurité, ex : CB
- Un bug peut être :
  - soit un non-respect de la spécification du système (c'est-à-dire de la définition de ce que le système est censé faire),
  - soit un comportement inattendu non couvert par la spécification (par exemple, cas non prévu de deux actions contradictoires à traiter simultanément, cas du bug de l'an 2000).

Source: wikipedia, ...

Bugs et qualité

- bug = dysfonctionnement logiciel
- problème : tous niveaux
  - processeur, ex: bug virgule flottante des processeurs Pentium
  - codage, ex : bug an 2000 (années codées sur deux chiffres), 2038 (langage C codé sur 32 bits)
  - logiciel, ex : crash Ariane
  - faille sécurité, ex : CB
- Solutions
  - Les règles de programmation.
  - Les techniques de programmation.
  - Les méthodologies de développement.
  - Le support des langages de programmation.
  - Le test.
  - Les méthodes formelles.

Source: wikipedia, ...

Bugs et qualité

- bug = dysfonctionnement logiciel
- problème : tous niveaux
  - processeur, ex: bug virgule flottante des processeurs Pentium
  - codage, ex : bug an 2000 (années codées sur deux chiffres), 2038 (langage C codé sur 32 bits)
  - logiciel, ex : crash Ariane
  - faille sécurité, ex : CB
- Solutions
- Méthodes formelles sont une solution parmi d'autres : *notation/modèle formel + sémantique bien définie + outillage*
  - indispensable pour les systèmes critiques, la certification, ...

Source: wikipedia, ...

Quelques domaines clés

Dans certains domaines d'activité, les spécifications formelles ont acquis un droit de cité incontestable :

- les réseaux de PETRI sont largement utilisés pour spécifier et valider des protocoles de communication, les logiques temporelles en sont une alternative moins répandue.
- les grammaires formelles sont utilisées pour définir et analyser les langages, notamment en théorie des langages de programmation.
- les automates à états finis permettent la modélisation de mécanismes critiques dont la validation formelle est estimée nécessaire (moniteurs temps réel, postes de commande).
- la logique formelle et les algèbres permettent de formaliser et de démontrer des algorithmes essentiels.
- la théorie des ensembles est un support important des bases de données relationnelles et des spécifications abstraites de systèmes.

Pour quoi utiliser ?

- Analyse et spécification :
  - Description formelle des concepts clés du problème (domaine d'application)
  - Spécification formelle de la solution (comportement du système)
  - Prototypage
- Conception architecturale et détaillée :
  - Spécification formelle de l'interface des modules.
  - Méthode rigoureuse de raffinement
- Codification :
  - Preuves de programmes
  - Synthèse automatique du code
  - Utilisation d'assertions
- Tests
  - Génération automatique des tests

Source: G. Tremblay

Bénéfices

- "Devoir en arriver à une meilleure compréhension de l'élément à spécifier en forçant l'analyste à être abstrait tout enétant précis à propos des propriétés du système peut être plus gratifiant que d'avoir le document de spécification lui-même." [Win90].
- Impact positif de l'effort de formalisation.
- Spécifications explicites, précises, non-ambiguës.
- Base pour la mise en oeuvre et pour des vérifications formelles.
- Outils (manipulation, analyse, simulation, génération de tests).
- Base pour le développement des tests.

Source: G. Tremblay

Applications

(quelques) succès et industriels concernés :

- Orly Val, Métro ligne 14 [méthode B]
- U. Oxford et IBM CICS [Z] / qualité + réduction des coût 9%
- NASA, Airbus, IBM, Microsoft, FT R&D, ...

Exemples ayant utilisé des méthodes classiques :

- Réingénierie du système CICS (IBM).
- Composants matériels (Inmos).
- Appareilsélectroniques (Tektronik).
- Outils pour l'analyse structurée.
- Contrôle du métro (Paris).
- Contrôle de trains (SNCF).
- Contrôle du trafic aérien (UK et USA).
- Contrôle de réacteurs nucléaires (Ontario, USA).
- Navette spatiale de la NASA (contrôleur de vol, GPS).
- Contrôle pour machine de radiothérapie.

Source: G. Tremblay, P. Poizat

Application 2/2

Exemples ayant utilisé la vérification de modèles :

-
- Protocole de cohérence de cache IEEE Futurebus+ (identification d'erreurs non détectées auparavant)
- Protocole de télécommunication ISDN/ISUP (122 erreurs détectées)
- Contrôleur de canal HDLC (erreur majeure détectée)
- Système actif de contrôle des structures en génie civil (erreur majeure détectée, qui aurait pû amplifier l'effet des vibrations)
- ...

Source: G. Tremblay

Intérêts

- aux aspects formels :
  - Des propriétés peuvent être établies par raisonnement formel, ce qui n'est pas le cas des autres formes de spécification. Les preuves ne sont plus intuitives mais basées sur une argumentation stricte.
  - Les conséquences d'une spécification peuvent être mises en évidence. Ainsi, les contradictions sont détectées avant la réalisation.
  - Les outils de preuve calculent et vérifient automatiquement et uniquement ce qui a été décrit.

Intérêts

- aux aspects formels :
- à la précision :
  - Le domaine du problème est mieux perçu. A force de réfléchir sur le sujet, on le comprend mieux, les imprécisions sont levées, les contradictions sont mises en valeur. Le problème est mieux compris.
  - Le même langage est pratiqué par différentes personnes.
  - Une conversion en langage naturel facilitera la discussion avec le client, qui lui-même appréhende mieux le problème. On gagne en clarté dans le dialogue.
  - La spécification formelle est un avant-projet de la réalisation. Le programmeur sait exactement ce qu'il doit programmer.

Intérêts

- aux aspects formels :
- à la précision :
- à l'abstraction ("quoi pas comment"),
  - Une abstraction permet de ne retenir que les caractéristiques essentielles,
  - Une description abstraite est plus évolutive et perméable aux changements des besoins, elle améliore la maintenabilité du logiciel son accès.
  - L'abstraction permet une distinction plus nette entre étude et réalisation, pour lesquelles les choix et les décisions sont différents. Une approche multi-language facilite la distinction.
  - Les méthodes formelles mènent à des composants plus faciles à réutiliser et plus fiables. On se rapproche en ce sens des techniques d'autres "génies" (civil, mécanique, électrique).

Inconvénients

- difficulté : les praticiens n'ont pas toujours la connaissance nécessaire. Les formalismes ne sont pas toujours très lisibles ou accessibles. "*Malgré les progrès significatifs accomplis ces dernières années pour rendre les notations formelles plus compréhensibles et utilisables, la rédaction et l'emploi de spécifications formelles exigent toujours un certain niveau d'aptitude mathématique, ainsi qu'un effort important.*" [Mey92].
- diversité : plusieurs standards existent (Z, VDM, modèles algébriques, OSDL, etc.).

Inconvénients

- difficulté :
- diversité :
- pauvreté des processus et des outils support : peu de méthodes présentent un processus clair et réaliste, un environnement complet et convivial fait défaut même si des prototypes existent. Les preuves sont souvent données de façon anecdotique dans les ouvrages. Le formalisme doit permettre autant que possible d'automatiser la preuve.

Inconvénients

- difficulté :
- diversité :
- pauvreté des processus et des outils support :
- difficultés de modélisation : manque de structuration, erreurs, exceptions, concurrence. On a besoin d'outils et de méthodes pour construire les spécifications complexes. La modélisation passe aussi par la capitalisation du savoir (modèles, méthodes utilisées).

Inconvénients

- difficulté :
- diversité :
- pauvreté des processus et des outils support :
- difficultés de modélisation :
- adéquation : la spécification formelle aide à cerner les oublis et les contradictions, mais on ne peut prouver que ce qui a été spécifié, pas ce qui était (implicitement) souhaité par le client.

Inconvénients

- difficulté :
- diversité :
- pauvreté des processus et des outils support :
- difficultés de modélisation :
- adéquation :
- preuve : des outils de preuve existent (calcul de schémas en Z, réécriture dans les algèbres), mais ils sont insuffisants. Des études sont en cours pour les enrichir et donner les algorithmes de test et de preuve.

Inconvénients

- difficulté :
- diversité :
- pauvreté des processus et des outils support :
- difficultés de modélisation :
- adéquation :
- preuve :
- domaine d'application : actuellement, les spécifications formelles ne sont appliquées qu'à un nombre réduit d'applications : "*Nous avons présenté dans ce texte une certaine manière d'aborder une petite classe de problèmes informatiques, ...*" [Abr84]. "*Certains éléments des langages de programmation (le parallélisme, l'arithmétique en virgule flottante, les structures de données complexes) sont encore difficiles à modéliser de manière satisfaisante.*" [Mey92]. Le lecteur trouvera dans [BH96] un certain nombre d'application "réaliste" de méthodes formelles.

Inconvénients

- difficulté :
- diversité :
- pauvreté des processus et des outils support :
- difficultés de modélisation :
- adéquation :
- preuve :
- domaine d'application :
- raffinage : la théorie du raffinage ou réification est relativement simple à comprendre : on transforme petit à petit les structures de données et de contrôle jusqu'à atteindre les langages de programmation. Il faut noter que le nombre d'étapes de raffinage peut être important et que la preuve du bien-fondé de la transformation est souvent très lourde. "*Une relation bien fondée ne contient pas de cycle*" [Jon93].

Inconvénients

- difficulté :
- diversité :
- pauvreté des processus et des outils support :
- difficultés de modélisation :
- adéquation :
- preuve :
- domaine d'application :
- raffinage :
- structuration : on a besoin d'outils de structuration des spécifications pour en simplifier l'écriture, l'accès la preuve et la réutilisation. Certains outils existent (dérivation de shéma en Z, sous-typage et relation d'importation dans les spécifications algébriques) mais ne sont pas toujours suffisants.

Plan

Exemple introductif

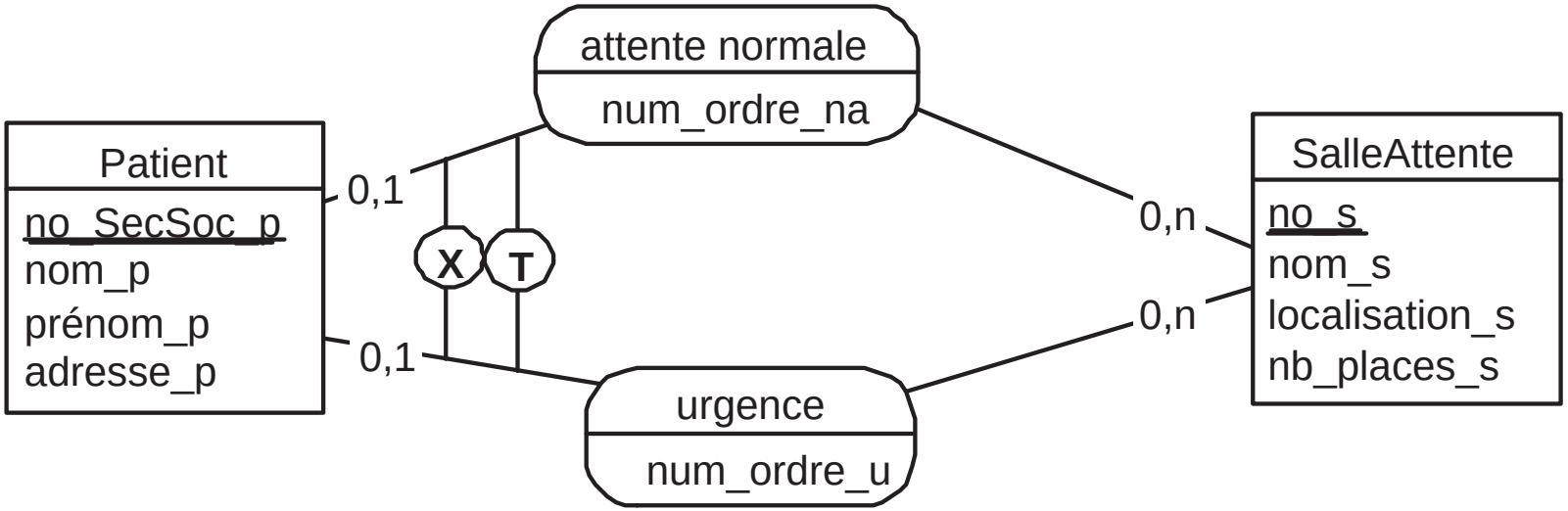
1. Description informelle
2. Modélisation avec Merise
   - MCD
   - MCT
   - Critique
3. Modélisation avec Z
   - Données
   - Traitements
   - Critique
4. Bilan

Description informelle

L'administration veut gérer automatiquement les salles d'attente à l'hôpital. Chaque salle a une capacité d'accueil limitée. Elles portent un nom et sont situées à un emplacement précis de l'hôpital. Les patients sont admis normalement ou en urgence. Les patients admis en urgence sont appelés prioritairement vis-à-vis des patients en admission normale. L'ordre d'appel est l'ordre d'arrivée, compte-tenu des éventuelles urgences. Un patient appelé est reçu et soigné par le docteur. Les soins ne sont pas interrompus par l'arrivée d'un patient en urgence. On considère trois cas :

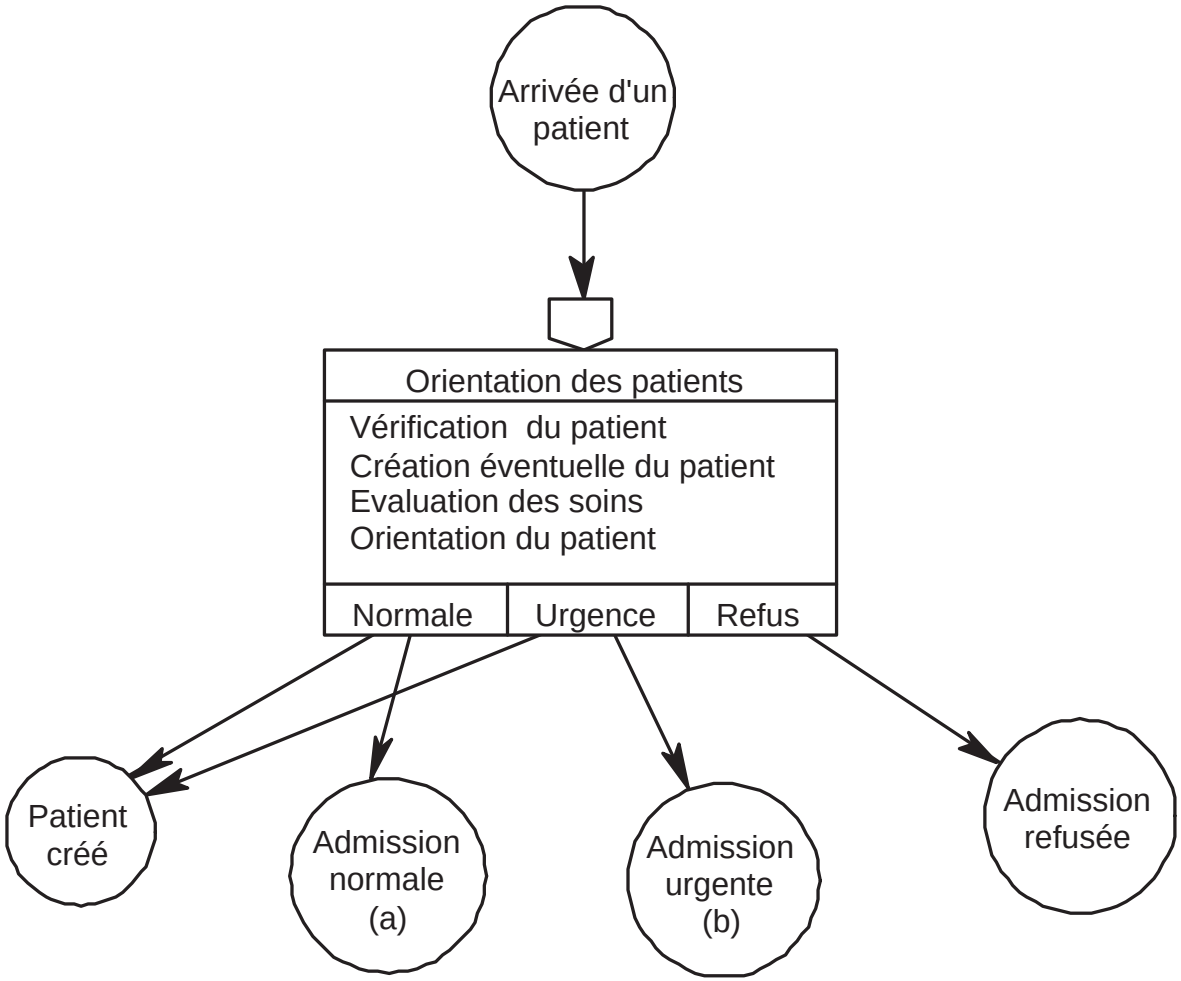
1. une seule salle d'attente et pas d'urgences,
2. une seule salle d'attente et traitement des urgences,
3. plusieurs salles d'attente et traitement des urgences.

Modélisation avec Merise (MCD)

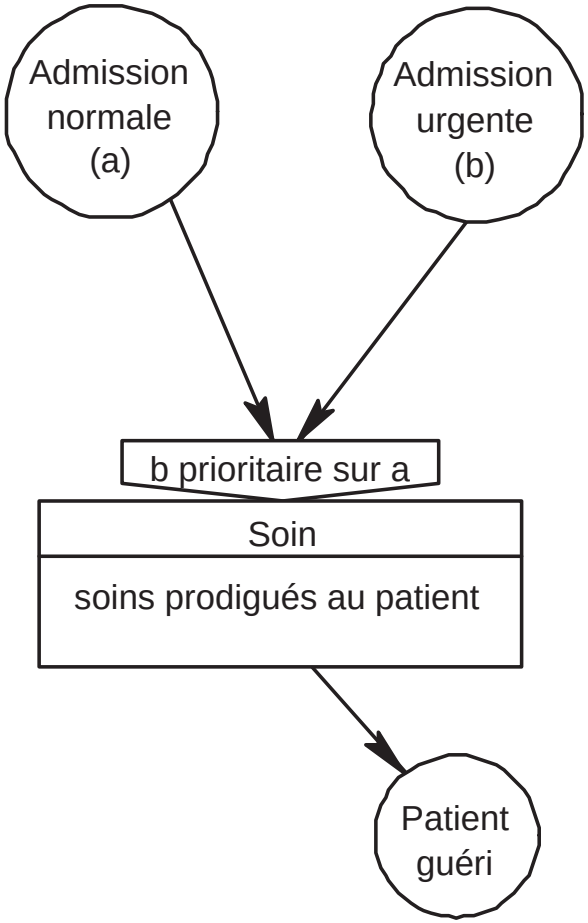


+ dictionnaire des données + explications

Modélisation avec Merise (MCT)



Modélisation avec Merise (MCT)



+ explications

Critique (MCD)

+ Plus concis et moins soumis à interprétation que la description informelle.

Critique (MCD)

+ Plus concis et moins soumis à interprétation que la description informelle.
+ MCD presque complet (cardinalités, contraintes).

Critique (MCD)

+ Plus concis et moins soumis à interprétation que la description informelle.
+ MCD presque complet (cardinalités, contraintes).

Mais ...

Critique (MCD)

+ Plus concis et moins soumis à interprétation que la description informelle.
+ MCD presque complet (cardinalités, contraintes).

Mais ...

- Manque de précision

Critique (MCD)

+ Plus concis et moins soumis à interprétation que la description informelle.
+ MCD presque complet (cardinalités, contraintes).

Mais ...

- Manque de précision
  - règles de gestion

Critique (MCD)

+ Plus concis et moins soumis à interprétation que la description informelle.
+ MCD presque complet (cardinalités, contraintes).

Mais ...

- Manque de précision
  - règles de gestion
  - contraintes entre associations

Critique (MCD)

\+ Plus concis et moins soumis à interprétation que la description informelle.

\+ MCD presque complet (cardinalités, contraintes).

Mais ...

- Manque de précision
  - règles de gestion
  - contraintes entre associations
  - contraintes informelles

Critique (MCD)

+ Plus concis et moins soumis à interprétation que la description informelle.
+ MCD presque complet (cardinalités, contraintes).

Mais ...

- Manque de précision
  - règles de gestion
  - contraintes entre associations
  - contraintes informelles
- Manque de sémantique formelle

Critique (MCD)

+ Plus concis et moins soumis à interprétation que la description informelle.
+ MCD presque complet (cardinalités, contraintes).

Mais ...

- Manque de précision
  - règles de gestion
  - contraintes entre associations
  - contraintes informelles
- Manque de sémantique formelle
  - Comment vérifier ?

Critique (MCD)

+ Plus concis et moins soumis à interprétation que la description informelle.
+ MCD presque complet (cardinalités, contraintes).

Mais ...

- Manque de précision
  - règles de gestion
  - contraintes entre associations
  - contraintes informelles
- Manque de sémantique formelle
  - Comment vérifier ?
  - Comment prouver la correction ?

Critique (MCD)

+ Plus concis et moins soumis à interprétation que la description informelle.
+ MCD presque complet (cardinalités, contraintes).

Mais ...

- Manque de précision
  - règles de gestion
  - contraintes entre associations
  - contraintes informelles
- Manque de sémantique formelle
  - Comment vérifier ?
  - Comment prouver la correction ?
  - Pas d'oublis ?

Critique (MCT)

+ Enchaînements des opérations.

Critique (MCT)

+ Enchaînements des opérations.
+ Vue globale des traitements.

Critique (MCT)

+ Enchaînements des opérations.
+ Vue globale des traitements.

Mais ...

Critique (MCT)

+ Enchaînements des opérations.
+ Vue globale des traitements.

Mais ...

- Manque de précision

Critique (MCT)

+ Enchaînements des opérations.
+ Vue globale des traitements.

Mais ...

- Manque de précision
  - règles de gestion

Critique (MCT)

\+ Enchaînements des opérations.

\+ Vue globale des traitements.

Mais ...

- Manque de précision
  - règles de gestion
  - détails des opérations

Critique (MCT)

+ Enchaînements des opérations.
+ Vue globale des traitements.

Mais ...

- Manque de précision
  - règles de gestion
  - détails des opérations
  - contraintes sur les événements

Critique (MCT)

+ Enchaînements des opérations.
+ Vue globale des traitements.

Mais ...

- Manque de précision
  - règles de gestion
  - détails des opérations
  - contraintes sur les événements
- Manque de sémantique formelle

Critique (MCT)

+ Enchaînements des opérations.
+ Vue globale des traitements.

Mais ...

- Manque de précision
  - règles de gestion
  - détails des opérations
  - contraintes sur les événements
- Manque de sémantique formelle
  - Comment vérifier ?

Critique (MCT)

+ Enchaînements des opérations.
+ Vue globale des traitements.

Mais ...

- Manque de précision
  - règles de gestion
  - détails des opérations
  - contraintes sur les événements
- Manque de sémantique formelle
  - Comment vérifier ?
  - Comment prouver la correction ?

Critique (MCT)

+ Enchaînements des opérations.
+ Vue globale des traitements.

Mais ...

- Manque de précision
  - règles de gestion
  - détails des opérations
  - contraintes sur les événements
- Manque de sémantique formelle
  - Comment vérifier ?
  - Comment prouver la correction ?
  - Pas d'oublis ?

Exemple introductif

1. Description informelle
2. Modélisation avec Merise
   - MCD
   - MCT
   - Critique
3. Modélisation avec Z
   - Données
   - Traitements
   - Critique
4. Bilan

Modélisation avec Z (Données)

Définitions de types, déclarations globales

$[PATIENT]$

$MAX \mathrel{\widehat=} 20$

Définitions de l'état du système (schéma)

SalleAttenteHôpital

$$patients : \operatorname{seq} PATIENT$$

$$\#patients \leq MAX$$

$$\forall i, j : 1 \mathinner{..} \#patients \mid i \neq j \bullet patients(i) \neq patients(j)$$

Modélisation avec Z (Traitements)

Définitions des opérations de modification (schéma)

Arrivée

$$\Delta SalleAttenteHôpital$$
$$p? : PATIENT$$

$$patients' = patients \frown \langle p? \rangle$$
$$\#patients < MAX \land \forall i : 1 \,..\, \#patients \bullet patients(i) \neq p?$$

Sortie

$$\Delta SalleAttenteHôpital$$
$$p! : PATIENT$$

$$patients = \langle p! \rangle \frown patients'$$
$$patients \neq \langle \rangle$$

Modélisation avec Z (Traitements)

Définitions des préconditions des opérations (schéma)

$$\mathrm{pre}\ Arrivé \mathrel{\widehat=} \exists SalleAttenteHôpital' \bullet Arrivée$$

$$\begin{array}{|l} \mathrm{pre}\ Arrivée \\ \hline SalleAttenteHôpital \\ p? : PATIENT \\ \hline \#patients < MAX \land \\ \quad (\forall i : 1 \mathinner{..} \#patients \bullet patients(i) \neq p?) \\ \hline \end{array}$$

Modélisation avec Z (Traitements)

Définitions des opérations de consultation (schéma)

$$\frac{EnAttente}{\Xi SalleAttenteHôpital \quad n! : \mathbb{N} \quad | \quad n! = \#patients}$$

Combien de patients en attente ?

Modélisation avec Z (Traitements)

$n^{\text{ième}}$ patient à consulter

Sorties

$\Delta SalleAttenteHôpital$

$p! : PATIENT$

$n? : \mathbb{N}_1$

$n? \leq \#patients$

$\exists sa_1 : \operatorname{seq} PATIENT \mid \#sa_1 = n? - 1 \bullet$
$\quad patients = sa_1 \frown \langle p! \rangle \frown patients'$

Modélisation avec Z (Validation)

Prouver des

- propriétés générales *e.g.*cohérence et complétude, équité, la vivacité . . .

Modélisation avec Z (Validation)

Prouver des

- propriétés générales *e.g.*cohérence et complétude, équité, la vivacité . . .
- propriétés attendues du système.

Modélisation avec Z (Validation)

Prouver des

- propriétés générales *e.g.*cohérence et complétude, équité, la vivacité ...
- propriétés attendues du système.
- propriétés liées à la pratique de Z (ou VDM) (obligations de preuves)

Modélisation avec Z (Validation)

Prouver des

- propriétés générales *e.g.*cohérence et complétude, équité, la vivacité ...
- propriétés attendues du système.
- propriétés liées à la pratique de Z (ou VDM) (obligations de preuves)
  - Existence d'un état initial.

Modélisation avec Z (Validation)

Prouver des

- propriétés générales *e.g.*cohérence et complétude, équité, la vivacité ...
- propriétés attendues du système.
- propriétés liées à la pratique de Z (ou VDM) (obligations de preuves)
  - Existence d'un état initial.
  - Préservation des invariants dans les opérations.

Modélisation avec Z (Validation)

Prouver des

- propriétés générales *e.g.*cohérence et complétude, équité, la vivacité ...
- propriétés attendues du système.
- propriétés liées à la pratique de Z (ou VDM) (obligations de preuves)
  - Existence d'un état initial.
  - Préservation des invariants dans les opérations.
  - Preuve de raffinement de données ou d'opérations.

Modélisation avec Z (Validation)

- Les propriétés attendues pour le système salle d'attente sont :

Modélisation avec Z (Validation)

- Les propriétés attendues pour le système salle d'attente sont :
  - La capacité de la salle n'est pas dépassée.

Modélisation avec Z (Validation)

- Les propriétés attendues pour le système salle d'attente sont :
  - La capacité de la salle n'est pas dépassée.
  - Il y a équité (forte).

Modélisation avec Z (Validation)

- Les propriétés attendues pour le système salle d'attente sont :
  - La capacité de la salle n'est pas dépassée.
  - Il y a équité (forte).
  - L'ordre de sortie est l'ordre d'arrivée.

Modélisation avec Z (Validation)

- Les propriétés attendues pour le système salle d'attente sont :
  - La capacité de la salle n'est pas dépassée.
  - Il y a équité (forte).
  - L'ordre de sortie est l'ordre d'arrivée.
  - Conservation du nombre pour (arrivée + sortie ...)

Modélisation avec Z (Validation)

- Les propriétés attendues pour le système salle d'attente sont :
  - La capacité de la salle n'est pas dépassée.
  - Il y a équité (forte).
  - L'ordre de sortie est l'ordre d'arrivée.
  - Conservation du nombre pour (arrivée + sortie ...)
- Les obligations de preuves sont :

Modélisation avec Z (Validation)

- Les propriétés attendues pour le système salle d'attente sont :
  - La capacité de la salle n'est pas dépassée.
  - Il y a équité (forte).
  - L'ordre de sortie est l'ordre d'arrivée.
  - Conservation du nombre pour (arrivée + sortie ...)
- Les obligations de preuves sont :
  - Existence d'un état initial.
    $\exists SalleAttenteHôpital \bullet InitSalleAttenteHôpital$;

Modélisation avec Z (Validation)

- Les propriétés attendues pour le système salle d'attente sont :
  - La capacité de la salle n'est pas dépassée.
  - Il y a équité (forte).
  - L'ordre de sortie est l'ordre d'arrivée.
  - Conservation du nombre pour (arrivée + sortie ...)
- Les obligations de preuves sont :
  - Existence d'un état initial.
    $\exists SalleAttenteHôpital \bullet InitSalleAttenteHôpital$;
  - Préservation de l'invariant dans les opérations.
    $\exists SalleAttenteHôpital'; sorties! \bullet SchémaOp.$

Modélisation avec Z (Raffinement)

De la spécification au code...

Deux types de raffinement sont distingués en Z :

- Le raffinement des opérations. Les structures de contrôle du langage de programmation cible sont introduites progressivement *e.g.*séquence, conditionnelles, itérations pour la programmation impérative.
- Le raffinement des données. Les structures de données du langage de programmation cible sont introduites progressivement *e.g.*structures, tableaux, pointeurs pour la programmation impérative.

Modélisation avec Z (Raffinement *)

$[PATIENT]$
$MAX \mathrel{\widehat=} 20$

SalleConcret

$file : \operatorname{seq} PATIENT$
$premier : \mathbb{N}$

$\#file \leq MAX \land premier \leq MAX$
$\forall i, j : 1 \mathinner{..} premier \mid i \neq j \bullet file(i) \neq file(j)$

InitSalleConcret

SalleConcret

$premier = 0$

Modélisation avec Z (Raffinement **)

ArrivéeC

$\Delta SalleConcret$
$p? : PATIENT$

$premier < MAX \land \forall i : 1 \,..\, premier \bullet file(i) \neq p? \land$
$premier' = premier + 1 \land file' = \langle p? \rangle \frown \text{front } file$

pre *ArrivéeC*

$SalleConcret$
$p? : PATIENT$

$premier < MAX \land \forall i : 1 \,..\, premier \bullet file(i) \neq p?$

Modélisation avec Z (Raffinement ***)

SortieC

$\Delta SalleConcret$
$p! : PATIENT$

$premier > 0$
$premier' = premier - 1$

pre *SortieC*

$SalleConcret$
$p! : PATIENT$

$premier > 0$

Modélisation avec Z (Raffinement ****)

EnAttenteC

$\Xi SalleConcret$

$n! : \mathbb{N}$

$n! = premier$

SortiesC

$\Delta SalleConcret$

$p! : PATIENT$

$n? : \mathbb{N}_1$

$n? \leq premier \land p! = file(n?) \land$

$premier' = premier - n? \land file' = file$

Modélisation avec Z (Raffinement *****)

Abs

SalleAttenteHôpital
SalleConcret

$patients = \mathrm{rev}\ ((1\,..\,premier) \lhd file)$

Preuves du raffinement

- de l'état initial
- des opérations

Exemple introductif

1. Description informelle
2. Modélisation avec Merise
   - MCD
   - MCT
   - Critique
3. Modélisation avec Z
   - Données
   - Traitements
   - Critique
4. Bilan

Bilan

Les "spec formelles"

+ Formalisme (théories, logiques et calculs)

Bilan

Les "spec formelles"

+ Formalisme (théories, logiques et calculs)
+ Rigueur (types, contraintes, invariants, traitements)

Bilan

Les "spec formelles"

+ Formalisme (théories, logiques et calculs)
+ Rigueur (types, contraintes, invariants, traitements)
+ Abstraction (vis à vis des langages de programmation)

Bilan

Les "spec formelles"

- $+$ Formalisme (théories, logiques et calculs)
- $+$ Rigueur (types, contraintes, invariants, traitements)
- $+$ Abstraction (vis à vis des langages de programmation)
- $\Rightarrow$ Réduction des erreurs tardives, réutilisabilité en confiance

Bilan

Les "spec formelles"

+ Formalisme (théories, logiques et calculs)
+ Rigueur (types, contraintes, invariants, traitements)
+ Abstraction (vis à vis des langages de programmation)

$\Rightarrow$ Réduction des erreurs tardives, réutilisabilité en confiance

- pratique délicate (lisibilité, expérience)

Bilan

Les "spec formelles"

+ Formalisme (théories, logiques et calculs)
+ Rigueur (types, contraintes, invariants, traitements)
+ Abstraction (vis à vis des langages de programmation)

$\Rightarrow$ Réduction des erreurs tardives, réutilisabilité en confiance

- pratique délicate (lisibilité, expérience)
- lourdeur de modélisation

Bilan

Les "spec formelles"

+ Formalisme (théories, logiques et calculs)
+ Rigueur (types, contraintes, invariants, traitements)
+ Abstraction (vis à vis des langages de programmation)

$\Rightarrow$ Réduction des erreurs tardives, réutilisabilité en confiance

- pratique délicate (lisibilité, expérience)
- lourdeur de modélisation
- temps de spécification et preuve

Bilan

Les "spec formelles"

+ Formalisme (théories, logiques et calculs)
+ Rigueur (types, contraintes, invariants, traitements)
+ Abstraction (vis à vis des langages de programmation)

$\Rightarrow$ Réduction des erreurs tardives, réutilisabilité en confiance

- pratique délicate (lisibilité, expérience)
- lourdeur de modélisation
- temps de spécification et preuve
- domaine d'application

Bilan

Les "spec formelles"

+ Formalisme (théories, logiques et calculs)
+ Rigueur (types, contraintes, invariants, traitements)
+ Abstraction (vis à vis des langages de programmation)

$\Rightarrow$ Réduction des erreurs tardives, réutilisabilité en confiance

- pratique délicate (lisibilité, expérience)
- lourdeur de modélisation
- temps de spécification et preuve
- domaine d'application

$\Rightarrow$ Application limitée

Usage

Usage des "spec formelles"

- réfléchir sur des spécifications avant d'implanter = raisonnement abstrait

Usage

Usage des "spec formelles"

- réfléchir sur des spécifications avant d'implanter = raisonnement abstrait
- prouver des propriétés avant de tester

Usage

Usage des "spec formelles"

- réfléchir sur des spécifications avant d'implanter = raisonnement abstrait
- prouver des propriétés avant de tester
- prouver la correction de programmes

Usage

Usage des "spec formelles"

- réfléchir sur des spécifications avant d'implanter = raisonnement abstrait
- prouver des propriétés avant de tester
- prouver la correction de programmes
- automatiser la construction de programmes

Usage

Usage des "spec formelles"

- réfléchir sur des spécifications avant d'implanter = raisonnement abstrait
- prouver des propriétés avant de tester
- prouver la correction de programmes
- automatiser la construction de programmes
- se dégager des contraintes d'implantation

Usage

Usage des "spec formelles"

- réfléchir sur des spécifications avant d'implanter = raisonnement abstrait
- prouver des propriétés avant de tester
- prouver la correction de programmes
- automatiser la construction de programmes
- se dégager des contraintes d'implantation
- documenter la réutilisation

Pour aller plus loin

- Les systèmes formels, définitions, utilisation des spécifications formelles, Intérêts et limites, Classification
  $\Longrightarrow$ [AV01b] Chapitre 1
- What is this Theory Good for?

  `www.cs.auc.dk/~luca/SV/use.html`
- Palshikar, Girish Keshav (2001) Applying Formal Specifications to Real-World Software Development, IEEE Software, Novembre/Décembre, pp. 89-97

  `www.seg.etsmtl.ca/sylvie/LOG310/Textes/Palshikar.pdf`
- van Lamsweerde, Axel (2000) Formal Specification: a Roadmap, ACM Future of Software Engineering Conference, Limerick, Irelande, pp. 149-159.

  `www.seg.etsmtl.ca/sylvie/LOG310/Textes/p147-lamsweerde.pdf`

Plan

Les systèmes formels

Système formel =

- langage formel

Les systèmes formels

Système formel =

- langage formel
  - alphabet
  - syntaxe → META-LANGAGE

Les systèmes formels

Système formel =

- langage formel
  - alphabet
  - syntaxe → META-LANGAGE

mots, expressions, termes, éléments, formules

→ acceptés ou pas

Les systèmes formels

Système formel =

- langage formel
- système formel
  - axiomes
  - règles d'inférence
    règles de déduction ($\rightarrow$ méta-langage d'inférence))

Les systèmes formels

Système formel =

- langage formel
- système formel
  - axiomes
  - règles d'inférence
    règles de déduction ($\rightarrow$ méta-langage d'inférence))

construction des mots, expressions, termes, éléments, formules
$\rightarrow$ acceptés

Les systèmes formels

Utilisation d'un système formel $\Rightarrow$

- sémantique (langage naturel, dénotationnelle, opérationnelle...)

Les systèmes formels

Utilisation d'un système formel $\Rightarrow$

- sémantique (langage naturel, dénotationnelle, opérationnelle...)
- interprétation

  $\rightarrow$ associer un sens (une valeur) aux expressions

Les systèmes formels

Utilisation d'un système formel $\Rightarrow$

- sémantique (langage naturel, dénotationnelle, opérationnelle...)
- interprétation
- calcul
  - preuve = suite d'inférence
  - théorème = expression prouvée

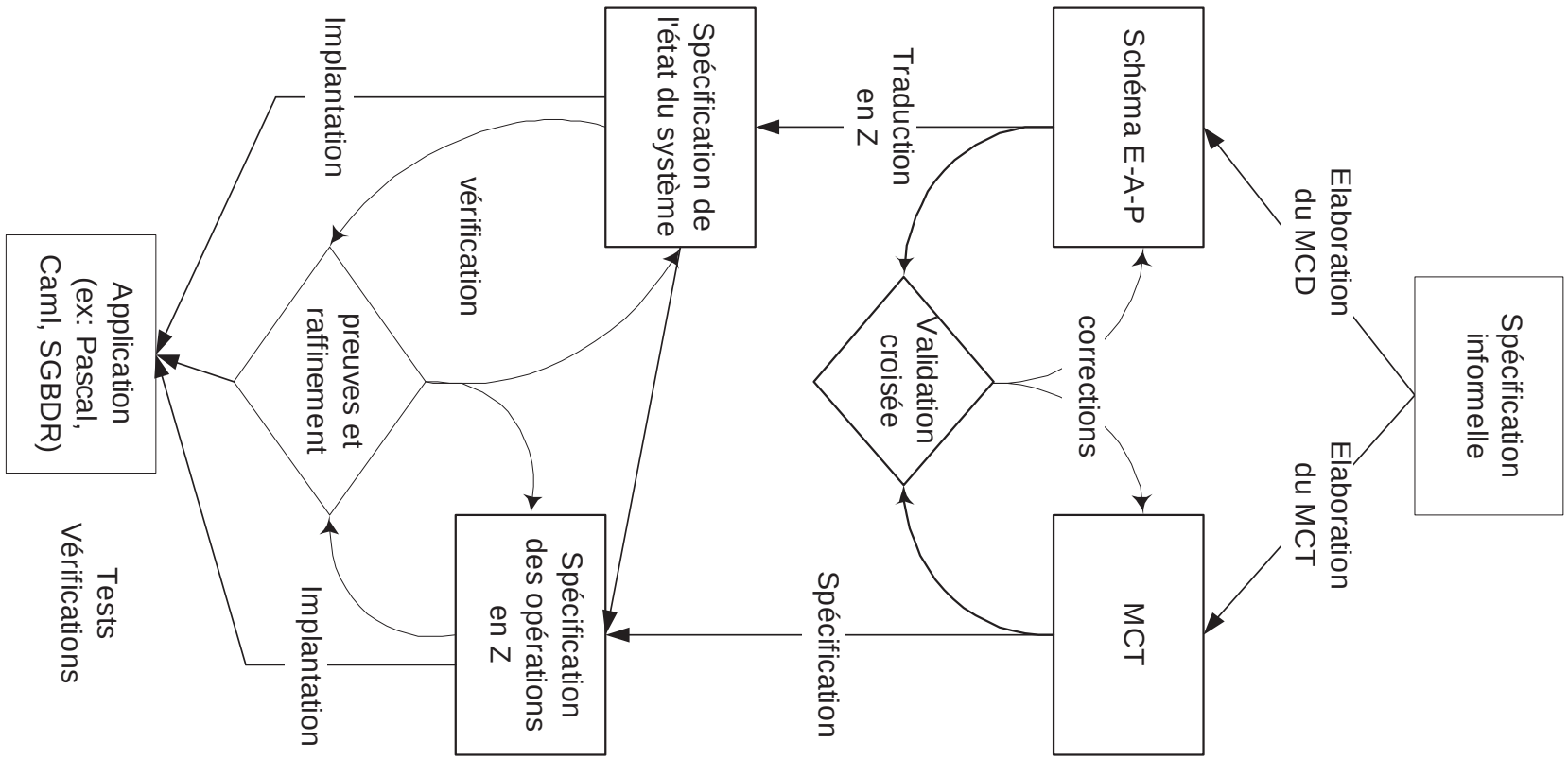
Les systèmes formels

Utilisation d'un système formel $\Rightarrow$

- sémantique (langage naturel, dénotationnelle, opérationnelle...)
- interprétation
- calcul
- propriétés
  - cohérence $\Rightarrow$ deux expressions $\neq$ ont des valeurs différentes
    en logique : on ne peut déduire vrai $=$ faux
  - complétude $\Rightarrow$ pas de valeur non-atteignables par les expressions
    en logique : chaque valeur 'vrai' est prouvée par un théorème

Plan

La spécification formelle



Plan

Une classification des méthodes formelles

Il existe de nombreuses approches plus ou moins parcellaires avec de nombreuses variantes.

Site de référence :
http://archive.comlab.ox.ac.uk/formal-methods.html
maintenant http://formalmethods.wikia.com/wiki/VL

une centaine de méthodes référencées !!!

Une classification des méthodes formelles

Il existe de nombreuses approches plus ou moins parcellaires avec de nombreuses variantes.

Site de référence :
http://archive.comlab.ox.ac.uk/formal-methods.html
maintenant http://formalmethods.wikia.com/wiki/VL

une centaine de méthodes référencées !!!

- difficile de classer les approches
  $\rightarrow$ subjectif

Une classification des méthodes formelles

Critères possibles :

- type de raisonnement

Une classification des méthodes formelles

Critères possibles :

- type de raisonnement :
  - structure (par modèle, opérationnel) :
    - modèle mathématique : Z, B, VDM, AMN, automates, réseaux de Petri...
    - modèle algorithmique : CLU
    - théorie des types : Coq,
    - algèbres de processus : CSP, CCS, π-calcul, ...
    - files d'attentes : SDL, QNAP, ...
  - propriétés (axiomatique) : réécriture, LP, spécifications algébriques (LPG, ASL, CLEAR, OBJ, PLUSS, LARCH, CASL), λ-calcul, logiques temporelles (TLA, LTL...), logiques diverses...

Pour chaque cas, il existe des approches hybrides (LOTOS, COLD, RSL...)

Une classification des méthodes formelles

Critères possibles :

- type de raisonnement
- aspect du système :
  - statique : Z, B, VDM, AMN...
  - dynamique : automates, réseaux de Petri, SDL, CSP, CCS, TLA, π-calcul ...
  - fonctionnel : spécifications algébriques, λ-calcul

Pour chaque cas, il existe des approches hybrides (LOTOS, COLD, RSL...)

Une classification des méthodes formelles

Critères possibles :

- type de raisonnement
- aspect du système
- langage :
  - général : Z, B, VDM, spécifications algébriques, CLU...
  - dédié : automates, réseaux de Petri, TLA, SDL, CSP, CCS, π-calcul, λ-calcul...

Pour chaque cas, il existe des approches hybrides (LOTOS, COLD, RSL...)

Une classification des méthodes formelles

Critères possibles :

- type de raisonnement
- aspect du système
- langage

En général, les approches s'accompagnent d'outils de preuve.
Les approches sont des méthodes si : méthode + outils de développement.

Plan

Exemples

1. Z
2. Spécification algébrique
3. Spécification algorithmique
4. B
5. LOTOS

ex.intro.pdf