Slide 1

# Méthodes de spécification et développement formel : l'approche orientée *modèle*

Christian Attiogbé
UFR Sciences Nantes,
Dpt. Informatique

Christian.Attiogbe@univ-nantes.fr

Octobre 1996, maj septembre 2001



Slide 2

## Développement de logiciels

**Nature** des logiciels :

séquentiels, parallèles,
flot de données, transactionnels,

autonomes, centralisés,
répartis, réactifs, temps-réels

embarqués, protocoles, mobiles,
...

**Slide 3**

## Cycles de vie

Ils ont été pendant longtemps le **support méthodologique** du développement du logiciel.

Les plus représentatifs de ces cycles de vie :

– Cycle en V,

– Cycle en cascade,

– Cycle de Balzer



**Slide 4**

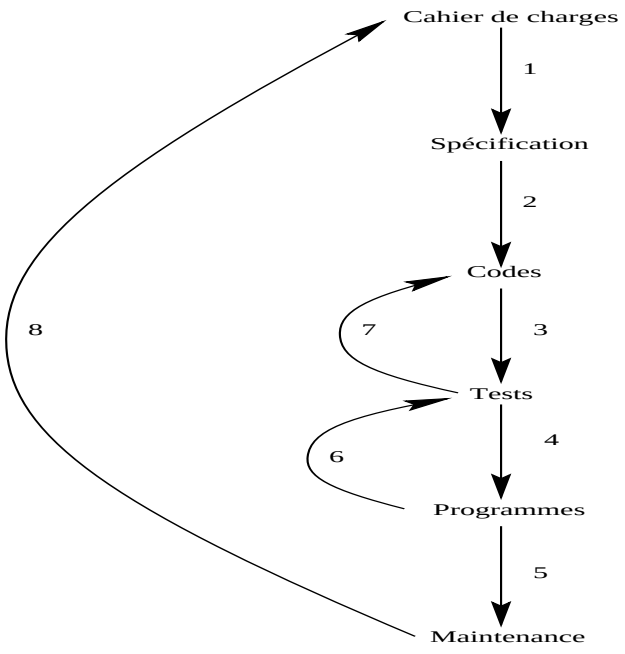



FIG. 1 – Cycle de vie en cascade (Boehm 1977)

**Slide 5**

## Intérêts et limites des méthodes semi-formelles

– SADT, SA-RT, SSADM, ...
– JSD-JSP,
– Merise, Axial, ...
– OOA, OMT, ...
– UML

L'analyse du problème est faite. → une contribution positive même si insuffisante.

Le problème est "dégrossi".



**Slide 6**

## Intérêts et limites des ... (suite)

**Mais** il est **impossible de raisonner/analyser** formellement sur le système en vue.

Il peut y avoir des ambiguités, ...

sources de $BUG$!

Slide 7

## Spécification formelle

**⇒ Expression dans un langage formel du quoi d'un système à développer.**

– Résultat de la phase d'analyse

– Plusieurs formes possibles selon la nature du système

→ langages ou formalismes de spécification formelle :

**Logique, $Z$, Langages de spécification algébriques, algèbres de processus, etc**

Méthodes formelles

Slide 8

## Démarche de spécification

Les aspects *données* ou bien les aspects *opérations* ?

Deux grandes écoles de spécification formelle et de méthodes formelles :

– **Les données d'un système permettent de décrire les états du système**

– **les opérations du système permettent de décrire son fonctionnement ou son comportement par des axiomes**

→ *paradigme des données* et *paradigme des opérations*.

Méthodes formelles

**Slide 9**

Il convient de distinguer :
– les opérations exprimant le comportement d'un système
– des opérations caractérisant les données du système.
Les premières opèrent sur les données du système,

les dernières permettent de construire et exploiter les données du système.



**Slide 10**

Il y a un troisième paradigme qui semble être transversal.

C'est ***le paradigme des processus*** (les algèbres de processus).

Dans ce paradigme des processus les systèmes sont décrits par les équations exprimant leur comportement ou leur états.

Les principales algèbres de processus à la base des autres formalismes sont :
– CSP (Hoare)
– CCS (Milner)
– ACP (Bergstra)

## Spécifications orientées états (model-based approach ou state-based approach)

Slide 11

On privilégie les données du système, on s'intéresse à l'état du système.

– **Principe** Une spécification permet de construire, à l'aide des concepts de base fournis, un modèle du système (logiciel) à développer. Ce modèle doit avoir les propriétés du système. On peut ensuite raisonner sur le fonctionnement du système en utilisant le modèle.



## Spécifications orientées états (suite)

– **Concepts de base** On utilise essentiellement les mathématiques discrètes (Théorie des ensembles), et la Logique.

Slide 12

– **Quelques exemples de formalismes**
Z, VDM, B pour les systèmes séquentiels,
Réseaux de Pétri, CCS, CSP, Unity, ... pour les systèmes concurrents et distribués. (CCS et CSP relèvent du paradigme des processus)

## Spécifications orientées propriétés ou axiomatiques

**(axiomatic approach, algebraic approach, operation oriented)**

**Slide 13**

– **Principe** La spécification permet de construire une axiomatisation qui fournit les propriétés (comportementales) du système à développer en utilisant les concepts de base cités ci-après.



– **Concepts de base**

Logique,

Algèbre,

Domaines définis inductivement.

**Slide 14**

– **Exemples de formalisme**

les langages de spécification algébriques : LPG, ASL, ACT ONE, CLEAR, OBJ, PLUSS,...

les algèbres de processus : LOTOS

Slide 15

## Spécifications hybrides (hybrid approach)

– **Principe** La spécification permet de compléter une axiomatisation par un modèle de données ou bien de compléter un modèle de données par une axiomatisation.

– **Concepts de base**
Concepts de base des approches orientées état et propriétés.



Slide 16

## Spécifications hybrides (suite)

– **Exemples de formalisme**
  – RAISE (Combine VDM et CSP)
  – VDM-SL et Réseaux de Petri,
  – Extended LOTOS(combine ACT ONE, CCS et CSP),
  – ZCCS (combine Z et CCS),
  – ...

**Slide 17**

## Méthodes de développement formel

Pour résumer l'essentiel des idées nous reprenons le slogan :

prouver c'est programmer et

programmer c'est prouver !



**Slide 18**

La deuxième façon de résumer est de considérer l'interprétation suivante de l'isomorphisme de Curry-Howard.

**Preuve** $\dfrac{Axiomes}{Theoremes}$ est équivalent à

**Développement** $\dfrac{Specifications}{Programmes}$

## Quelques définitions

**Méthode formelle =**

– Langage formel (syntaxe précise et sémantique précise) et

**Slide 19**

– Système de preuve ou de raisonnement formel.

**Développement formel =**

transformation systématique des spécifications en programmes en utilisant des lois prédéfinies.



**Plusieurs approches dans les méthodes formelles** :

– l'**approche opérationnelle** : le développement est basé sur la description d'un système par un modèle qui a les propriétés du système.

**Slide 20**

– l'**approche axiomatique** : le développement est basé sur l'axiomatisation qui décrit le comportement du système

– l'**approche hybride**

**Slide 21**

## Nouveau cycle de vie

Avec les méthodes formelles les cycles de vie classique ne marchent plus.

Il faut les adapter.

Le cycle de vie de BALZER représente la nouvelle famille de cycles de vie adaptés au développement formel (ou rigoureux !).



**Slide 22**

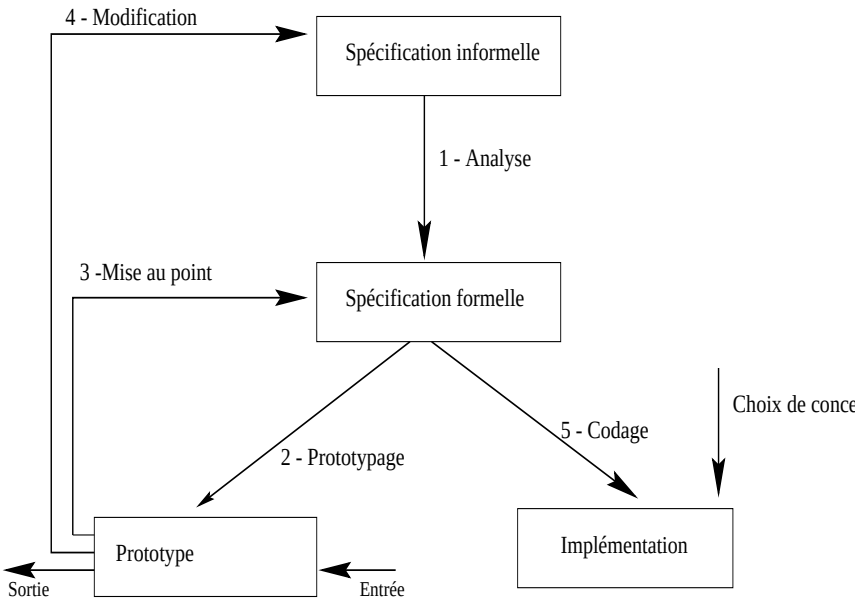



FIG. 2 – Cycle de vie de Balzer

**Slide 23**

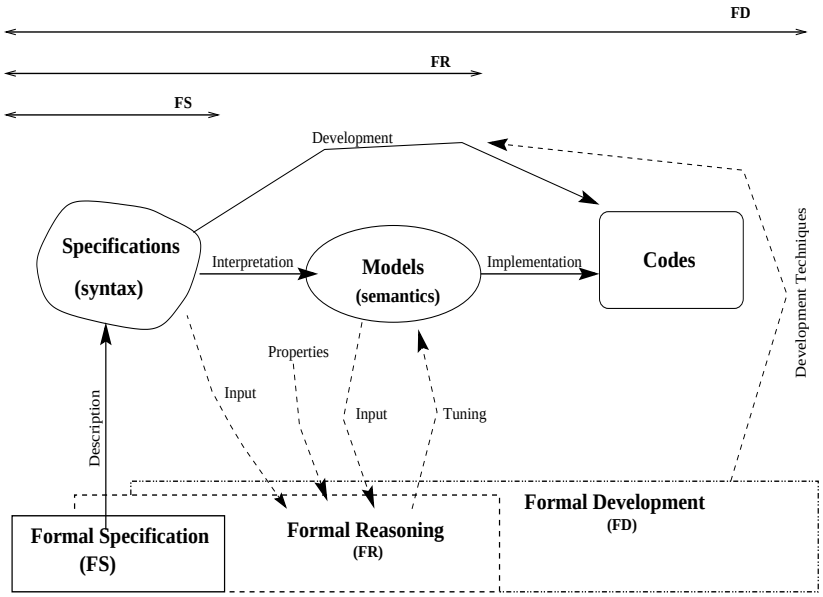



FIG. 3 – Présentation synthétique du développement formel



## Un aperçu des formalismes et méthodes de l'approche orientée modèle

**Slide 24**

$\rightarrow$ les principaux concepts à la base de $VDM$, $Z$ et $B$, etc

**Slide 25**

| | Z | VDM | B |
|---|---|---|---|
| Année de parution | début des années 1980 | fin des 1970 | fin des années 1980 |
| Auteurs et Affiliations | J-R Abrial(FR)<br>puis J. M. Spivey(UK) | C. Jones(UK) | J-R. Abrial (FR) |
| Concepts de base | Théorie des ensembles<br>Logique du 1er ordre | Théorie des ensembles<br>Fonctions partielles | Théorie des ensembles<br>Logique du 1er ordre<br>Transformateurs de prédicats |



**Slide 26**

| | | | |
|---|---|---|---|
| Autres concepts | Schéma<br>Raffinements<br>Calcul de Schémas | pre-post conditions<br>Raffinements | Machine abstraite,<br>Substitutions généralisées,<br>Raffinement |
| Modèle sémantique | Etat Transition | Etat Transition | Etat Transition |
| Phases du cycle de vie | Spécification | Spécification,<br>Conception | Specification,<br>Conception,<br>Implantation |

**Slide 27**

| | Z | VDM | B |
|---|---|---|---|
| Modèles d'exécution | Séquentiel | Séquentiel | Séquentiel, Réparti |
| Domaines appropriés | ++ | ++ | +++ |
| Exemples d'applications industrielles | CICS, INMOS<br>... | IBM, ADA<br>... | KVS, Metro Paris,<br>Metro Calcutta, SACEM (SNCF, GEC-alsthom RATP) |
| Outils domaine public | ZTC, Z/EVES | VDM-Parser | |
| Outils commercialisés | fuzz, Cadiz, Zola | Mural | AtelierB (~~Digilog~~ ClearSy FR),<br>Btoolkit (BCore, UK) |

Méthodes formelles

**Slide 28**

| Variantes | Z++, OOZE, Object-Z | VDM++ | |
|---|---|---|---|
| Standardisation | Tentative ZBase Standard | VDM-SL | ?! |
| Documentation | Spivey + autres (abondantes) | Jones + autres (abondantes) | Abrial (B-Book) + autres (+en, -fr) |
| Pratique | + Accessible | + Accessible | +/- Accessible |

Méthodes formelles

Slide 29

## Fondements communs

– Modélisation des données

  Logique

  Théorie des ensembles

– Modèle sémantique

  Systèmes de transition



Slide 30

## Eléments de logique

## Eléments de logique

Le raisonnement formel utilise les règles d'inférence et les séquents.

### Séquent

**Slide 31**

La preuve d'une conjecture donnée $P$ dépend le plus souvent d'*hypothèses* $HYP$ encore appelées *premisses*.
On dit qu'**on prouve $P$ sous les hypothèses $HYP$.**

La représentation des séquents est de la forme :

$$HYP \vdash P$$



### Règle d'inférence

Les règles d'inférence relient des séquents. Une règle d'inférence définit une transition valide dans le déroulement d'une preuve.

La représentation d'une règle d'inférence est de la forme :

**Slide 32**

$$\frac{\sum_1, \sum_2, \ldots, \sum_n}{\sum}$$

($\sum_1, \sum_2, \ldots, \sum_n$ et $\sum$ sont des séquents).
Les séquents $\sum_1, \sum_2, \ldots, \sum_n$ sont appelés les

**antécédents**,
et le séquent $\sum$ est appelé **conséquent**.

Slide 33

## Principe des preuves

Pour prouver un séquent on peut utiliser les règles d'inférence

- soit comme des règles de dérivations :
  c'est l'**application en avant des règles**,

- soit comme des règles de réduction :
  c'est l'**application en arrière**.



Slide 34

## Les règles de base (RB)

On nomme les règles pour pouvoir les référencer dans les raisonnements formels.

### RB1

On peut toujours dériver le séquent exprimant qu'une assertion donnée $P$ comme conjecture est prouvable sous l'hypothèse $P$ elle même.

$$\frac{}{P \vdash P}$$

Slide 35

## RB2

C'est la **monotonicité des preuves** par rapport aux hypothèses. L'augmentation des hypothèses d'un séquent ne détruit pas les preuves obtenues par ce séquent.

$$\frac{HYP \vdash P, HYP \;\; includans \;\; HYP'}{HYP' \vdash P}$$



Slide 36

## RB3

La conclusion d'un séquent $\sum$ prouvé peut être utilisé comme hypothèse d'un autre séquent avec les mêmes hypothèses de $\sum$

$$\frac{HYP \vdash P \qquad HYP, P \vdash Q}{HYP \vdash Q}$$

Dans la pratique on utilise cette règle en arrière.
Il est souvent plus facile de prouver $Q$ à partir de l'extension de $HYP$ à $P$ (en ayant fait un bon choix de $P$, $HYP \vdash P$ est facile à prouver : c'est une preuve déjà faite ou connue) plutôt que de prouver directement $Q$ à partir de $HYP$.

**Slide 37**

**RB4**

Lorsque qu'une assertion $P$ figure dans une liste d'hypothèses, la preuve de $P$ est immédiate par le séquent ayant la liste comme hypothèses et $P$ comme conclusion.

$$\frac{HY\, P, P}{P}$$



**Slide 38**

## Calcul des propositions

Appelons $P$ une assertion élémentaire. C'est un énoncé $vrai$ ou $faux$ dans un contexte ou une théorie donnée (par exemple la théorie des ensembles).

Une proposition est une expression du langage dont la grammaire est la suivante :

$$\begin{aligned} prop \quad &::= \quad P \\ &\mid \quad prop \;\Rightarrow\; prop \\ &\mid \quad prop \wedge prop \\ &\mid \quad \neg prop \end{aligned}$$

Des parenthèses peuvent être utilisées si nécéssaire, la grammaire est alors modifiée en conséquence.

**Slide 39**



## Règles d'inférence du calcul propositionnel

$\wedge\, intr$ $\quad \dfrac{HYP \vdash P \qquad HYP \vdash Q}{HYP \vdash P \wedge Q}$

souvent appliquée en arrière pour diviser un but en deux sous-buts plus simples avec les même hypothèses.

**Slide 40**

$\wedge\, elim$ $\quad \dfrac{HYP \vdash P \wedge Q}{HYP \vdash P \qquad HYP \vdash Q}$

## Règles d'inférence du calcul propositionnel

$\Rightarrow intr$ $\quad \dfrac{HYP,\ P \ \vdash \ Q}{HYP \ \vdash \ P \Rightarrow Q}$

C'est la règle de déduction

**Slide 41**

$\Rightarrow elim$ $\quad \dfrac{HYP \ \vdash \ P \Rightarrow Q}{HYP,\ P \ \vdash \ Q}$

C'est l'anti-déduction



Modus Ponens

$$\frac{HYP \ \vdash \ P \qquad HYP \ \vdash \ P \Rightarrow Q}{HYP \ \vdash \ Q}$$

**Slide 42**

Contradiction : Première règle traitant du $\neg$

$$\frac{HYP,\ \neg Q \ \vdash \ P \qquad HYP,\ \neg Q \ \vdash \ \neg P}{HYP \ \vdash \ Q}$$

$$\frac{HYP,\ Q \ \vdash \ P \qquad HYP,\ Q \ \vdash \ \neg P}{HYP \ \vdash \ \neg Q}$$

**Slide 43**

Deuxième règle du $\neg$



## Procédure de décision dans le calcul propositionnel

**Slide 44**

Dans le calcul propositionnel, on peut tout démontrer à l'aide des règles suivantes :

**Slide 45**

$$\frac{HYP \ \vdash \ P}{HYP \ \vdash \ \neg\neg P}$$

$$\frac{HYP \ \vdash \ P \qquad HYP \ \vdash \ \neg Q}{HYP \ \vdash \ \neg(P \Rightarrow Q)}$$

$$\frac{HYP \ \vdash \ P \Rightarrow \neg Q}{HYP \ \vdash \ \neg(P \wedge Q)}$$

$$\frac{HYP \ \vdash \ P \Rightarrow R}{HYP \ \vdash \ \neg\neg P \Rightarrow R}$$

$$\frac{HYP \ \vdash \ P \Rightarrow (\neg Q \Rightarrow R)}{HYP \ \vdash \ \neg(P \Rightarrow Q) \Rightarrow R}$$



# Procédure décision (suite)

**Slide 46**

$$\frac{HYP \ \vdash \ \neg P \Rightarrow R, \quad HYP \ \vdash \ \neg Q \Rightarrow R}{HYP \ \vdash \ \neg(P \wedge Q) \Rightarrow R}$$

$$\frac{HYP \ \vdash \ \neg P \Rightarrow R, \quad HYP \ \vdash \ Q \Rightarrow R}{HYP \ \vdash \ (P \Rightarrow Q) \Rightarrow R}$$

$$\frac{HYP \ \vdash \ P \Rightarrow (Q \Rightarrow R)}{HYP \ \vdash \ P \wedge Q \Rightarrow R}$$

Slide 47

## Extension du langage à la disjonction et l'équivalence

Le symbole $\equiv$ désigne l'équivalence syntaxique.

Disjonction ($\vee$) :

$$P \vee Q \equiv \neg P \Rightarrow Q$$

Equivalence ($\Leftrightarrow$) :

$$p \Leftrightarrow Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$$

On peut compléter la grammaire donnée plus haut avec ces nouveaux opérateurs.



Slide 48

## Suite : Calcul des prédicats

Slide 49

## Suite : Eléments de la théorie des ensembles



## Références

Slide 50

[Bac89] C. Backhouse. Construction et vérification de programmes. coll. MIM. Masson, Prentice-Hall, 1989.

[GHL95] M. Gondran, J-F. Héry, and J-C. Laleuf. Logique et modélisation. Coll. DER - EDF. Eyrolles, 1995.

[Gri81] David Gries. The science of programming. Springer-Verlag, 1981.

[MB85] Bertrand Meyer and Claude Baudoin. Méthodes de programmation. Coll. DER - EDF. Eyrolles, 1985.

[Mon96] Jean-François Monin. Comprendre les méthodes formelles. coll. CNET-ENST. Masson, Paris, 1996.

Slide 51

## Calcul des prédicats

Calcul des propositions → de vérité absolue.

Le **calcul des prédicats traite des vérités relatives**.
On fait une extension du calcul propositionnel.

Quelle est la valeur de vérité de l'énoncé ?

$$i < j$$

Cet énoncé décrit une propriété de deux entiers $i$ et $j$.
Il est $vrai$ ou $faux$ **selon les valeurs affectées** à $i$ et $j$.



Slide 52

| $i$ | $j$ | $i < j$ |
|---|---|---|
| 0 | 1 | Vrai |
| 2 | 1 | Faux |
| 3 | 3 | Faux |

$i$ et $j$ sont des variables.

$i < j$ est un prédicat
(on parle aussi de $formule$ ou de $relation$).

Slide 53

Dans les prédicats on utilise deux sortes de variables :

les *variables libres* et les *variables liées*.
(liberté par rapport aux quantificateurs, portées)

Autres exemples :
Que dire de

$$x + 3 > y$$



Slide 54

## Formation des prédicats

Les prédicats sont construits avec :
- des **constantes** (0,2, ...),
- des **symboles de fonction** (f, g, +, *, ...),
- des **prédicats** (P, Q, R, ...)

et avec des symboles communs à tous les langages du premier ordre :
- les **connecteurs logiques** $\neg$, $\Rightarrow$, $\Leftrightarrow$, $\wedge$, $\vee$ de la logique des propositions,
- les **quantificateurs** : le quantificateur universel noté $\forall$ et le quantificateur existentiel noté $\exists$
- les **variables** (x, y, z, ...) qui forment un ensemble

dénombrable,

- les **parenthèses** ouvrantes et fermantes, parfois les crochets [ et ] et la virgule.

**Slide 55**



## Espaces d'états d'une variable

La valeur associée à une variable $x$ est appelé *état* de $x$. L'ensemble de toutes les variables possibles que $x$ peut avoir est appelé *espace d'états* de $x$.

## Variables libres et liées, substitutions

**Slide 56**

Une *formule* dans laquelle il y a des variables est une **formule non close**.

Une variable peut apparaître plusieurs fois dans une formule, on dit qu'elle a plusieurs **occurrences**.

**Slide 57**

Chaque occurrence d'une variable dans une formule peut être *liée* ou *libre* selon qu'elle apparaît ou non dans le champ d'un quantificateur ($\forall$ ou $\exists$).

Les occurrences (ou places) des variables dans les formules non closes sont dites *libres*. On dit aussi que les variables des formules non closes sont libres dans ces formules.



**Slide 58**

Les variables $x$ et $y$ sont *libres* dans la formule :
$x \ + \ 3 \ > \ y$.

Une variable peut être à la fois libre et liée dans une formule.

$$(x \ < \ 2) \ \vee \ \forall x(x \ + \ 3 \ = \ y)$$

Les variables libres d'une formule $P$ sont les variables qui ont une ou plusieurs occurrences libres dans $P$.

Toutes les occurrences libres de $x$ dans $P$ deviennent des occurrences *liées* dans $\forall x.P$ ou dans $\exists x.P$.

On parle aussi d'occurrence *non-libre*.

La notion d'occurrence liée (non-libre) d'une variable $x$ dans une formule $P$ notée $x\backslash F$ est définie formellement par induction sur la structure de $P$ :

**Slide 59**

| | |
|---|---|
| $x\backslash P$ | si $x$ ne figure pas dans $P$ |
| $x\backslash P \vee Q$ | si $x\backslash P$ et $x\backslash Q$ |
| $x\backslash P \iff Q$ | si $x\backslash P$ et $x\backslash Q$ |
| $x\backslash P \implies Q$ | si $x\backslash P$ et $x\backslash Q$ |
| $x\backslash P \wedge Q$ | si $x\backslash P$ et $x\backslash Q$ |



## Définition occurrence liée, par induction (suite)

**Slide 60**

| | |
|---|---|
| $x\backslash \neg P$ | si $x\backslash P$ |
| $x\backslash \forall y.P$ | si $x\backslash y$ et $x\backslash P$ |
| $x\backslash \forall x P$ | |
| $x\backslash \exists y.P$ | si $x\backslash y$ et $x\backslash P$ |
| $x\backslash \exists x.P$ | |

Une **substitution** est un remplacement des occurrences libres d'une variable dans une expression par une autre expression donnée.

Il y a diverses notations : on trouve souvent

**Slide 61**

$P[v\backslash x]$ la substitution de $v$ à toutes les occurences libres de $x$ dans $P$

$[v\backslash x]P$ la substitution de $v$ à toutes les occurrences libres de $x$ dans $P$.



## Les quantificateurs

Le prédicat $\forall x.P$ dit que tout prédicat obtenu en substituant dans $P$ les occurrences libres de $x$ par une expression est un théorème.

Le prédicat $\exists x.P$ est équivalent à $\neg\forall x.\neg P$.

**Slide 62**

Il exprime qu'il existe au moins une expression $F$ telle que le prédicat obtenu en substituant dans $P$ les occurrences libres de $x$ par $F$ est un théorème.

### Exemples de prédicats

1.

$$\neg(x^2 < 0) \vee \forall y.y + 2 = x$$

**Slide 63**

2. (Exemples extraits de Gondran, Héry et Laleuf)
   Les éléphants à grandes oreilles sont des éléphants d'Afrique

$$\forall x.GrandesOreilles(x) \Longrightarrow Africain(x)$$

Un éléphant a de grandes défenses si ses parents sont d'Afrique

$$\forall x.\forall y.(Parent(x,y) \Longrightarrow Africain(y))$$
$$\Longrightarrow GrandesDefenses(x)$$



**Slide 64**

Les parents d'un éléphant à grandes oreilles ont de grandes oreilles

$$\forall x \forall y.(Parent(x,y) \ \wedge \ GrandesOreilles(x)$$
$$\Longrightarrow GrandesOreilles(y))$$

## Les règles d'inférence du calcul des prédicats

$\forall intr$ $\dfrac{HYP \vdash P}{HYP \vdash \forall x\ P}$

si x n'est pas libre dans les axiomes de $HYP$.

Slide 65

$\forall elim$ $\dfrac{HYP \vdash \forall x\ P}{HYP \vdash (T/x)\ P}$

si $T$ est un terme substituable à x dans $P$.



## Les règles d'inférence du calcul des prédicats(...)

$\exists intr$ $\dfrac{HYP \vdash (T/x)\ P}{HYP \vdash \exists x\ P}$

si $T$ est librement substituable à $x$ dans $P$

Slide 66

$\exists elim$ $\dfrac{HYP \vdash \exists x\ P \qquad HYP \vdash P, \vdash Q}{HYP \vdash Q}$

si $x$ n'est pas libre ni dans $Q$ ni dans $HYP$

Slide 67

## Programmes :

## Construction et raisonnement formel
(DIJKSTRA, FLOYD, HOARE)

Un petit langage pour construire des programmes :
– l'affectation

`:=`

– la séquence

`;`

– la conditionnelle

`if... then... else...`

– l'itération

`while ... do...end`

Méthodes formelles

Slide 68

## Notion de commandes, instructions (statements) : $S$

**Les états :** ensemble des valeurs des variables.

**La notation de** HOARE

HOARE a axiomatisé la technique de FLOYD et proposé une notation où $P$ et $Q$ sont des prédicats, S un programme.

$\{P\}\ S\ \{Q\}$ exprime que toute exécution de $S$ qui commence dans un état satisfaisant $P$ se termine dans un état satisfaisant $Q$ **si** $S$ se termine.

$P$ est appelé **précondition**, $Q$ est appelé

Méthodes formelles

**postcondition**.

**Slide 69**



$\{P\}\ S\ \{Q\}$ exprime la **correction partielle** de $S$.

La correction est partielle parce qu'elle suppose que la préconditon $P$ garantit la terminaison de $S$.

**Slide 70**

La correction **devient totale quand on prouve la terminaison**.

## Exemples

$\{x = 1\}\ x := x + 1\ \{x = 2\}$

(vrai ou faux ?)

Slide 71

$\{x = 1\}$

```
while  true
do     SKIP
end
```

$\{y = 2\}$

(vrai ou faux ? que dire de la boucle ?)

Remarque : $SKIP$ représente l'instruction vide.



## Les plus faibles préconditions

Soient $S$ une commande et $R$ un prédicat qui décrit le résultat (voulu) de l'exécution de $S$.

Slide 72

Soit un autre prédicat $wp(S, R)$ qui représente : l'ensemble de tous les états tels que toute exécution de $S$ commençant par un d'entre eux **se termine** en un *temps fini* dans un état satisfaisant $R$ (où $R$ est vrai),

$wp(S, R)$ est la *plus faible précondition* de $S$ par rapport à $R$.

## Quelques exemples

Soient $S$ une affectation et $R$ le prédicat $i \leq 1$

$$wp(i := i + 1, i \leq 1) \ = \ (i \leq 0)$$

Soient $S$ la conditionnelle suivante :

**Slide 73**

`if` $x \geq y$ `then` $z := x$ `else` $z := y$

et $R$ le prédicat $z = max(x, y)$

$$wp(S, R) \ = \ Vrai$$



## Exercices

Soit $S$ une commande, que représente $wp(S, Vrai)$ ?

Calculer

**Slide 74**

$wp($

```
while 0 <> n
      do
         n := n - 1
      end
```

$, vrai)$

(on trouve $0 \ \leq \ n$)

**Slide 75**

Le sens de $wp(S, R)$ peut être précisé par **deux propriétés** :

– $wp(S, R)$ est une précondition garantissant $R$ après l'exécution de $S$, c'est à dire que :

$$\{wp(S, R)\}\ S\ \{R\}$$

– $wp(S, R)$ est la plus faible de telles préconditions, c'est à dire que :
si $\{P\}\ S\ \{R\}$ alors $P\ \Rightarrow\ wp(s, R)$



**Slide 76**

Souvent un programme a pour but d'établir une postcondition $R$.
On est souvent intéressé par les préconditions qui permettent d'établir $R$.
Plus précisément on est intéressé par une partie de $wp(S, R)$ représentée par **une condition plus forte** $P$ car le calcul de $wp(S,\ R)$ peut être parfois très difficile.

Il suffit ensuite de prouver que $P\ \Rightarrow\ wp(S,\ R)$.
Cependant la technique des **plus faibles préconditions** est plus intéressante à cause de la **guarantie de terminaison**.

$wp$ est une fonction à deux arguments :
– une instruction (ou programme) $S$ et
– un prédicat $R$.

Slide 77

Pour un programme quelconque fixé $S$ on peut voir $wp(S,\ R)$ comme une fonction à un seul argument $wp_S(R)$.
La fonction $wp_S$ est appelé **transformateur de prédicats**.

C'est la fonction qui associe à tout prédicat $R$ la plus faible précondition telle que $\{P\}\ S\ \{R\}$.



On détaillera les paragraphes suivants lorsqu'on va utiliser les formalismes de spécifications (Z, VDM, B).

**Preuve utilisant les plus faibles préconditions**

Slide 78

**Inductions et invariants**

**Construction des boucles**

Slide 79

## Eléments de la théorie des ensembles



## Eléments de la théorie des ensembles

**En guise d'introduction**

**Exercice :** on nous demande de développer un programme qui donne :

la liste de tous les pays européens membres de l'OTAN,

Slide 80

la liste des pays à la fois membres de l'Union européenne et membres de l'OTAN ,

les pays membres de l'union européenne ou membres de l'OTAN,

...

**Solution :** construire un modèle (ensembliste).

## Les bases

Cantor,

*Par ensemble on entend un groupement en un tout d'objets bien distincts de notre intuition ou de notre pensée.*

dans la suite le symbole $==$ pour $...est\ défini\ par...$

Slide 81

### Définition en extension

On définit un ensemble en donnant entre accolades, la liste de ses éléments séparés par des virgules.

### Exemple

$E_e == \{1,\ 3,\ 5,\ 7,\ 9\}$ permet de définir les 5 premiers entiers impairs.



## Définition en compréhension

On définit un ensemble en donnant un prédicat, en fonction d'une variable $x$, qui exprime la condition nécessaire et suffisante pour que $x$ soit élément de l'ensemble.

Slide 82

### Exemple

$E_c == \{x : I\!N / x\ mod\ 2\ =\ 1\}$ permet de définir l'ensemble des entiers impairs.

Slide 83

## Propriétés principales des ensembles

Dans un ensemble, chaque élément n'apparaît qu'**une seule fois** (il ne peut y avoir de répétition).

L'**ordre d'apparition** des éléments n'est **pas significatif**.

### Quelques exemples

$\{\}$ ou $\emptyset$ l'ensemble vide.

$UE == \{Belgique, France, Allemagne, Espagne, Italie, Luxembourg, Hollande, Danemark, Grece, Irlande, Portugal, Royaume-uni\}$



Slide 84

### Exemples (suite)

$OTAN == \{Belgique, Canade, Danemark, Islande, Italie, Luxembourg, Hollande, Norvege, Portugal, Royaume-uni, Etas-unis, Grece, Turquie, Espagne, Allemagne\}$

$Scandinavie == \{Danemark, Finlande, Norvege, Suede, Islande\}$

$Benelux == \{Belgique, Hollande, Luxembourg\}$

Slide 85

## Appartenance d'un élément

Le symbole de l'appartenance est $\in$.

On parle aussi de la relation *est membre de.*

**Exemples**

$$France \in OTAN$$

$$Bresil \in UE$$

**Remarque :** On peut chercher à savoir si un énoncé est vrai ou faux.



Slide 86

## La non-appartenance

Le symbole utilisé est $\notin$.

## Quelques exemples

$$Bresil \notin UE$$

$$France \notin UE$$

Slide 87

## Opérateurs constructeurs d'ensemble

On peut construire de nouveaux ensembles en combinant des ensembles donnés avec des opérateurs de base.

Les opérateurs de base sont :

$\cap(intersection), \cup(union), \setminus(difference)$

### Union de deux ensembles

Soient $A$ et $B$ deux ensembles,

$$A \cup B == \{x \,/\, (x \in A) \vee (x \in B)\}$$



Slide 88

### Intersection de deux ensembles

Soient $A$ et $B$ deux ensembles,

$$A \cap B == \{x \,/\, (x \in A) \wedge (x \in B)\}$$

### Différence ensembliste

Soient $A$ et $B$ deux ensembles,

$$A \setminus B == \{x \,/\, (x \in A) \wedge (x \notin B)\}$$

Slide 89

## Produit cartésien de deux ensembles

Soient $A$ et $B$ deux ensembles, le produit cartésien de $A$ et $B$ noté $A \times B$ est l'ensemble de toutes les *paires ordonnées* dont le premier élément est membre de $A$ et le deuxième est membre de $B$.

## Ensemble des parties d'un ensemble

Soit $A$ un ensemble, l'ensemble des parties de $A$ noté $\mathcal{P}(A)$ est l'ensemble de tous les sous-ensembles de $A$.



Slide 90

## Cardinal d'un ensemble

Soit $A$ un ensemble, le cardinal de $A$ noté $card(A)$ ou $\#A$ est le nombre d'éléments de $A$.

**Slide 91**

## Quelques exemples

$EU \cup OTAN =$
$\{Belgique, Danemark, France, Italie, Luxembourg, Hollande,$
$Norvege, Portugal, Royaume{-}uni, Etats{-}unis, Grece,$
$Turquie, Espagne, Allemagne, Irlande\}$

$EU \cap OTAN =$
$\{Belgique, Daneark, France, Italie, Luxembourg, Hollande,$
$Portugal, Royaume-uni, Grece, Espagne, Allemagne\}$



**Slide 92**

$OTAN \setminus UE =$
$\{Etats{-}unis, Canada, Islande, Norvege, Turquie\}$

$EU \setminus OTAN = \{Ireland\}$

## Opérateurs $\subseteq, \subset, =, \neq, \ldots$

**Inclusion d'un ensemble dans un autre**

Soient $A$ et $B$ deux ensembles, on dit que $A$ est inclu dans $B$ si :

$$\forall x,\ x \in A \Rightarrow x \in B$$

Slide 93

On note $A \subseteq B$ et ont dit aussi $A$ *est un sous-ensemble* de $B$ ou $A$ *est une partie de* $B$.

On distingue l'inclusion stricte en la notant $\subset$.

Deux ensembles sont disjoints si leur intersection est l'ensemble vide.



## Un peu plus sur les ensembles : relations, fonctions

**Les relations** Une relation binaire entre deux ensembles $A$ et $B$ est un sous-ensemble du produit cartésien $A \times B$. (c'est le sous-ensemble dont les éléments vérifient un certain prédicat défini).

Slide 94

**Exemples :** Soient

$A == \{Alice,\ Pierre,\ Patrice,\ Rodrique,\ Yves\}$ et

$B == \{velo,\ voiture,\ tricycle\}$

Slide 95

## Exemples (suite)

$\{(Alice,\ velo),\ (Patrice,\ voiture),\ (Rodrigue,\ velo)\}$ définit une relation binaire entre $A$ et $B$.

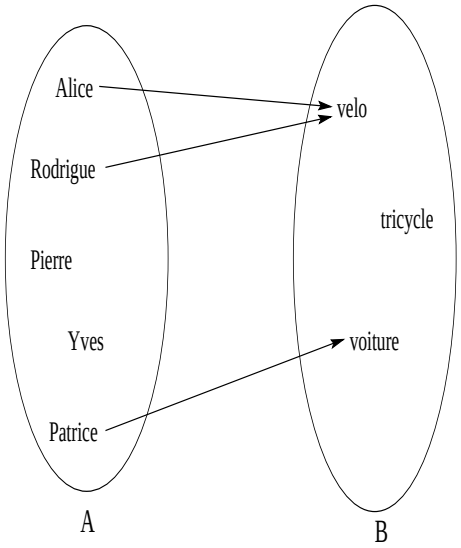
On peut la **nommer comme un ensemble**, mais généralement on donne un nom qui donne **intuitivement la propriété qui relie les éléments** des couples de la relation ; par exemple *conduit*.

C'est un ensemble de *couples* dont le premier élément appartient à $A$ et le deuxième appartient à $B$.



Slide 96

## Exemple de relation




On note

$conduit\ :\ A \longrightarrow B$

**Attention :** *conduit* est une relation de $A$ vers $B$. Il y en a d'autres, $A \longrightarrow B$ représente **un ensemble de relations de $A$ vers $B$. $conduit(Alice) = velo$ ou Alice *conduit* velo.**

$A$ est appelé **ensemble de départ** de la relation.
$B$ est appelé **ensemble d'arrivée** de la relation.
$\{Alice,\ Pierre,\ Rodrigue\}$ est appelé **domaine** de la relation. C'est le sous-ensemble, de l'ensemble de départ, dont les éléments sont dans la relation.

**Slide 97**

$\{velo,\ voiture\}$ est appelé **image** ou **codomaine** de la relation. C'est le sous-ensemble, de l'ensemble d'arrivée, dont les éléments sont dans la relation.
On parle aussi d'**image pour un élément** de $B$ relié avec un élément de $A$. velo est l'image de Alice.



## Les fonctions

Une relation de $A$ vers $B$ est une fonction lorsque **chaque élément a au plus une image** par la relation.

**Slide 98**

Une fonction de $A$ vers $B$ est une **fonction totale** lorsque le domaine de la fonction est $A$ tout entier.
(Une fonction totale est aussi appelée **application**)

Une fonction de $A$ vers $B$ est une **fonction partielle** lorsque le domaine de la fonction ne contient pas tous les éléments de $A$.

Slide 99

## Fonction injective

Soit $f$ une fonction de $A$ vers $B$, notée $f : A \rightarrow B$
$f$ est une injection (ou fonction injective) lorsque :
$\forall\, x,\, y \in A : (x \neq y) \longrightarrow (f(x) \neq f(y))$

## Fonction surjective

Soit $f$ une fonction de $A$ vers $B$, notée $f : A \rightarrow B$
$f$ est une surjection (ou fonction surjective) lorsque :
l'image de $f = B$.



Slide 100

## Fonction bijective

Soit $f$ une fonction de $A$ vers $B$, $f$ est une bijection (ou fonction bijective) lorsque $f$ est à la fois injective et surjective.

Pour conclure :

**Une fonction est une relation est un ensemble**

**Slide 101**

⇒ "hériter" des propriétés des ensembles dans la modélisation à l'aide des relations et des fonctions.

...et celles des relations pour les fonctions



## Modélisation avec les ensembles

(Exercices, TD)

**Slide 102**

## Notes bibliographiques

*Ce cours a été préparée à l'aide de quelques livres de référence en matière de construction de programmes et de raisonnement formel.*

***Nous recommandons de les lire ou de les relire.***

Slide 103

***Le livre de Gries*** *[Gri81] est une très bonne référence pour les concepts de base utilisés dans les méthodes formelles abordées dans le cadre de ce cours.*

***Le livre de Backhouse*** *[Bac89] (proche de celui de Gries) est un excellent cours pour les concepts de base de la construction de programmes, il est traduit en français.*



Slide 104

***Le livre de Monin*** *[Mon96] dresse un panorama interessant pour une vision d'ensemble.*

*La modélisation en logique peut être approfondie avec* ***le livre de Gondran, Hery et Laleuf*** *[GHL95].*

*Dans le* ***livre plus ancien de Meyer et Baudoin*** *[MB85] vous trouverez quelques bases interessantes pour la construction de programmes.*
*Il en existe de nombreux autres facilement accessibles.*