Département Informatique de l'IUT de l'Université Bordeaux 1

Cours d'Analyse et Conception des Systèmes d'Information (d'Outils et Modèles pour le Génie Logiciel) : les Méthodes

24 mai 2013

Olivier Guibert

B

Ce document doit beaucoup à www-lsr.imag.fr/users/Didier.Bert/Exams/poly-B-2004.pdf de Didier Bert et de Marie-Laure Potet ainsi qu'à *Spécification formelle avec B* d'Henri Habrias (Éditions Hermès – Lavoisier, 2001) et www.loria.fr/~cansell/MI1.pdf de Dominique Cansell.

Préambule

Un sas de décompression [Olivier Ly]

La machine suivante modélise un sas de décompression qui comporte une porte vers l'extérieur et une porte vers l'intérieur d'une cabine. La pression à l'extérieur est supposée nulle. La pression à l'intérieur doit demeurer constante. La pression est mesurée par un entier, à l'intérieur on suppose que la pression vaut 10 (l'unité de mesure de la pression est arbitraire). Le but de la machine est d'organiser les opérations d'ouverture et de fermeture des portes, ainsi que la commande de mise à niveau de la pression sas, de telle sorte que la pression à l'intérieur ne soit jamais modifiée.

Un sas de décompression

```
MACHINE
   SasDecompression
SETS
   PORTE = { Ouverte, Fermee }
PROPERTIES
   Ouverte /= Fermee
VARIABLES
   pression_sas,  /* pression dans le sas */
   pression_int,  /* pression à l'intérieur */
   porte_sas,     /* porte sas <-> extérieur */
   porte_int      /* porte sas <-> intérieur */
INVARIANT
     pression_sas : NATURAL
   & pression_int : NATURAL
   & porte_sas : PORTE
   & porte_int : PORTE
   & (porte_int = Ouverte => porte_sas = Fermee)
   & (porte_int = Ouverte => pression_sas = 10)
   & (porte_int = Ouverte => pression_int = 10)
   & (pression_int = 10)
```

Un sas de décompression

```
INITIALISATION
   pression_sas, pression_int, porte_sas, porte_int
      := 0, 10, Fermee, Fermee
OPERATIONS
/* opérations de l'environnement */
   /* lorsque la porte sas est ouverte,
      la pression du sas chute à 0 */
   modif_pression_sas =
   PRE porte_sas = Ouverte
   THEN pression_sas := 0
   END;
   /* lorsque la porte intérieure est ouverte,
      la pression du sas et la pression intérieure
      deviennent égales */
   modif_pression_int =
   PRE porte_int = Ouverte
   THEN pression_int := pression_sas
   END;
```

Un sas de décompression

```
/* opération du système */
   /* le système peut augmenter la pression du sas */
   requete_augmentation_pression_sas =
   PRE porte_sas = Fermee & porte_int = Fermee
      & pression_sas < 10
   THEN pression_sas := pression_sas + 1
   END;
/* opérations de l'utilisateur */
   /* ouverture de la porte du sas */
   ouverture_sas =
   PRE porte_int = Fermee
   THEN porte_sas := Ouverte
   END;
   /* fermeture de la porte du sas */
   fermeture_sas =
   THEN porte_sas := Fermee
   END;
```

Un sas de décompression

```
   /* ouverture de la porte intérieure */
   ouverture_int =
   PRE pression_sas = pression_int & porte_sas = Fermee
   THEN porte_int := Ouverte
   END;
   /* fermeture de la porte intérieure */
   fermeture_int =
   THEN porte_int := Fermee
   END;
END
```

Plan

Plan

- Présentation
  - Éléments de classification
  - Bibliographie ; Sites Internet ; Outils
  - Historique ; L’exemple industriel ; Utilisateurs
  - Objectifs
  - Modèles
- Démarche

Plan

- Modèles
  - Logique, Preuve
  - Ensemble
  - Relation, Fonction
  - Entier
  - Suite
  - *Substitution généralisée*
  - Machine abstraite
  - Contrainte dynamique
  - Raffinement (ou Raffinage ou Réification)
  - Implantation (ou Implémentation)
  - Module

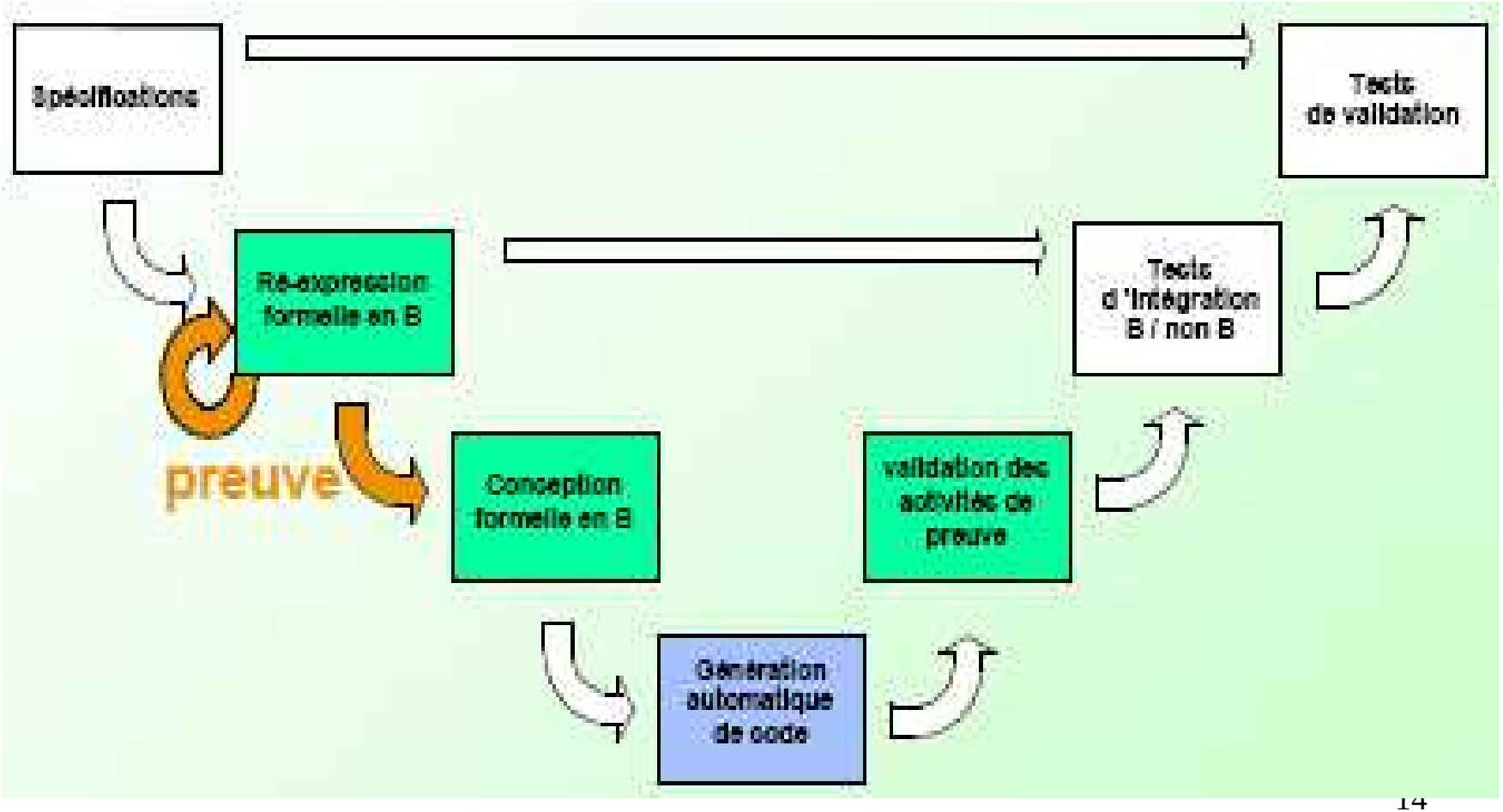
Présentation

Éléments de classification

Éléments de classification

- Fondements théoriques : mathématiques (méthode formelle)
- Dernière Génération
- Domaine d’application : global (intégralité du cycle de développement)
- Démarche linéaire
- Approche descendante

Positionnement relativement au cycle de développement en V



Bibliographie

Sites Internet

Outils

Bibliographie

- Jean-Raymond Abrial, *The B-book: Assigning Programs to Meanings*, Cambridge University Press, 1996
- Jean-Raymond Abrial, *Introduction à la méthode B*, 6 vidéo-cassettes, IUT de Nantes
- Jean-Raymond Abrial, *La méthode B - études de cas*, 6 vidéo-cassettes, IUT de Nantes
- Edsger Dijkstra, *A Discipline of Programming*, Prentice Hall, 1976
- Henri Habrias, *Introduction à la spécification*, Masson, 1993
- Henri Habrias, *Spécification formelle avec B*, Éditions Hermès – Lavoisier, 2001

Bibliographie

- *Software Specification Methods. An Overview Using a Case Study*, Marc Frappier et Henri Habrias (éditeurs), FACIT, Springer Verlag London Ltd., 2001
- Kevin Lano, *The B Language and Method: A guide to Practical Formal Development*, FACIT, Springer Verlag London Ltd., 1996
- Kevin Lano et Howard Haughton, *Specification in B: An Introduction Using the B Toolkit*, Imperial College press, London, 1996
- Carroll Morgan, *Programming from Specifications*, Prentice-Hall, 1996
- Carroll Morgan, *Deriving programs from specifications*, Prentice Hall International, 1990

Bibliographie

- Steve Schneider, *Software Engineering with B*, Palgrave, Cornerstones of Computing series
- Steve Schneider, *The B-Method: an Introduction*, Palgrave, Cornerstones of Computing series, 2001
- *Program Development by Refinement. Case Studies Using the B Method*, Emil Sekerinski and Kaisa Sere (éditeurs), FACIT, Springer Verlag London Ltd., 1999
- Mike Spivey, *La notation Z*, Masson - Prentice Hall, 1992
- John Wordsworth, *Software Engineering with B*, Addison-Wesley, 1996

- Cf. www3.inrets.fr/B@INRETS/B-Bibliography/

Sites Internet

- ftp.irisa.fr/techreports/2001/PI-1424.ps.gz [Mireille Ducassé et Laurence Rozé, IRISA, Rennes]
- http://estas1.inrets.fr:8001/ESTAS/BUG/WWW/BUGhome/BUGhome.html
- www.afm.sbu.ac.uk/b
- www.atelierb.societe.com/ressources/BAtb3.pdf
  www.atelierb.societe.com/ressources/MTB1.pdf
  www.atelierb.societe.com/ressources/manrefb.185.fr.pdf
- www.genie-logiciel.utc.fr/lo19/ [Jean-Louis Boulanger, Université de Technologie de Compiègne]
- www.irisa.fr/NQRT/19avril2001.pdf [Bertrand Russell, IFSIC, Université Rennes 1]

Sites Internet

- www.lirmm.fr/~boks/doc_ens/mait_B/B-Cours.doc
- www.loria.fr/~cansell/MI1.pdf [Dominique Cansell, Université de Metz]
- www.sciences.univ-nantes.fr/info/perso/permanents/habrias/methodb.html
- www3.inrets.fr/ESTAS/B@INRETS/
- www3.inrets.fr/estas/mariano/MSFORMEL/ : notesB.ps, cours_intro_msformel.ps, slides_cours_DESS_B.1p
- www-lsr.imag.fr/B/
- www-lsr.imag.fr/users/Didier.Bert/Exams/methodeB.pdf
  www-lsr.imag.fr/users/Didier.Bert/Exams/poly-B-2004.pdf [Didier Bert et Marie-Laure Potet, LSR-IMAG, Grenoble]

Outils

- Outils commerciaux
  - Atelier B (ClearSy), www.atelierb.societe.com/
  - BToolKit (BCore), www.b-core.com/
- Outils universitaires
  - Another B Tools, www.chez.com/abtools
  - B4free, http://www.b4free.com/
  - BCAML
  - Click'n'Prove (ou Balbulette), http://www.loria.fr/~cansell/cnp.html
  - JBTOOLS, lifc.univ-fcomte.fr/PEOPLE/tatibouet/JBTOOLS/BParser_fr.html
  - Rodin

Historique

L’exemple industriel

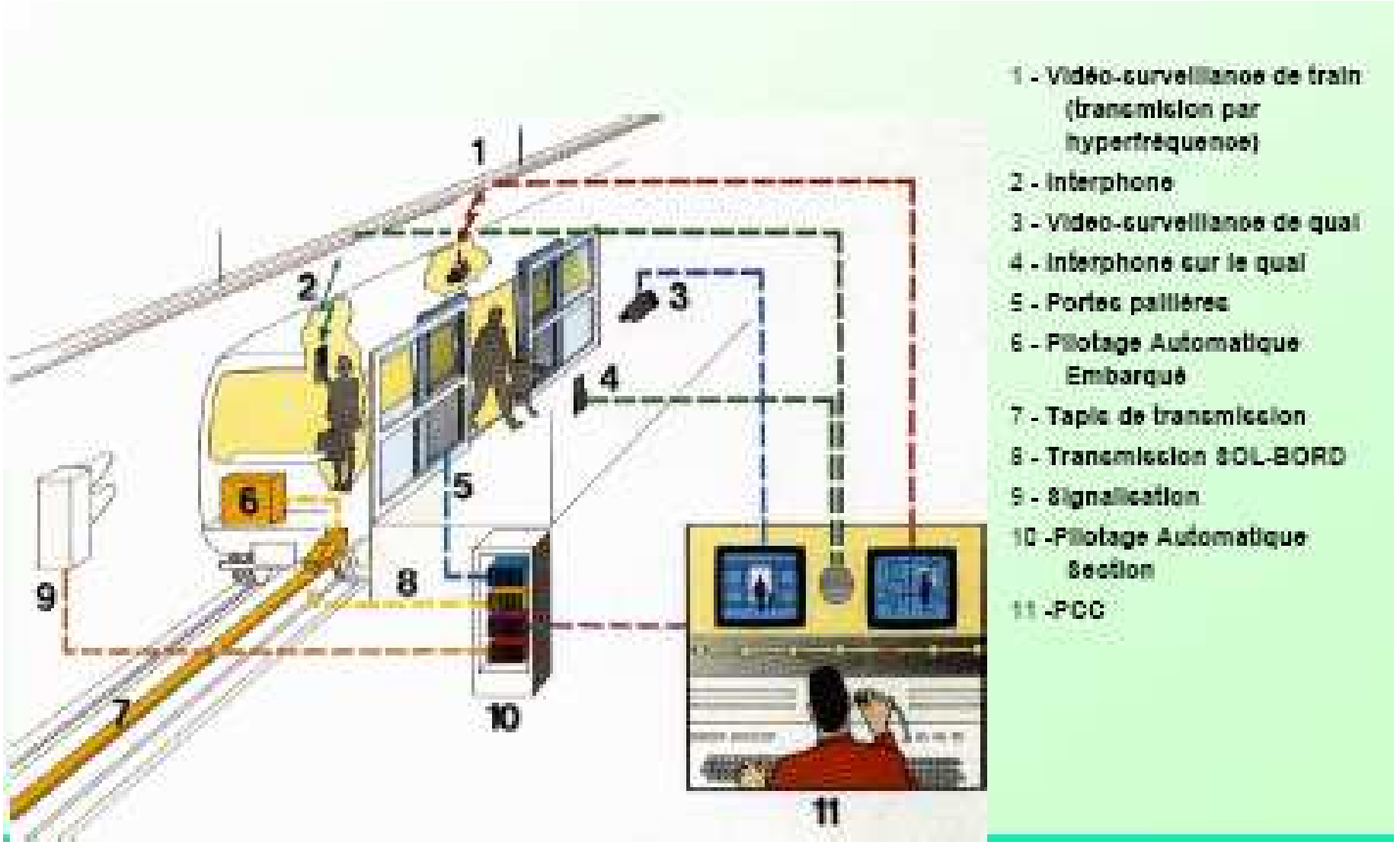
Utilisateurs

Historique

B pour Bourbaki (pseudonyme d'un groupe de mathématicien français)

- 1978 : notation Z [Jean-Raymond Abrial], développée lors de son séjour (1979-1981) au sein du Programming Research Group
- 1988-1991 : méthode B [Jean-Raymond Abrial pour British Petroleum]
- 1994-1995 : Industrialisation de la méthode B (dont Météor)
- 1996 : B-événementiel [Jean-Raymond Abrial]

L’exemple industriel : Météor



L’exemple industriel : Météor

- Logiciel (partie sécuritaire) de la ligne 14 du RER (métro régional parisien)
- 1 075 composants B
  107 000 lignes de code B
  87 000 lignes de code ADA traduites
- 29 000 obligations de preuve
  Prouvé à 100 % (dont plus de 80% automatiquement)
- Opérationnel dès son installation (fin 1998) ; aucune anomalie depuis sa mise en service
- Indice de qualité de service = 99,8 %

Utilisateurs

- Domaine (dans l'industrie) : aérospatial, automobile, commerce électronique, défense, énergie, nucléaire, télécommunication, tertiaire, transport ferroviaire i.e. là où la sûreté de fonctionnement est importante
  Ex. : British Petroleum, RATP, SNCF, Alstom, Matra Transport, etc.
- Temps réel, automatismes industriels, protocoles de communication, protocoles de cryptographie, informatique embarquée
- Thématique de recherche scientifique active (France, Grande-Bretagne, Australie, Finlande)

Objectifs

Objectifs

- Créer un logiciel correct par construction, garanti contractuellement sans défaut
  (« *assigning programs to meanings* » i.e. le logiciel est réalisé à partir de sa sémantique)
- Fournir un cadre homogène pour tout le cycle de développement (de l'analyse, conception et réalisation i.e. spécification, conception et implantation)
- B est un langage de modélisation :
  - Développement d'une abstraction d'un système
  - Prise en compte des événements qui modifient l'état du système
  - Définition des invariants à spécifier et à prouver

Modèles

Modèles

- Logique des propositions et des prédicats : négation, et logique, ou logique, implication, équivalence, quantificateur universel, quantificateur existentiel
- Preuve : séquent, règle d'inférence, stratégie de preuve (par l'avant ou par l'arrière)
- Théorie des ensembles : produit cartésien, parties, définition en extension ou en compréhension, appartenance, inclusion, union, intersection, différence
- Relation : relation, domaine, codomaine, image, inverse, identité, composition séquentielle (et itération, fermeture réflexive, fermeture réflexive transitive), produit direct, produit parallèle, 1re et 2e projection, restriction/corestriction de domaine/codomaine, modification, transformée en relation

Modèles

- Fonction : fonction/injection/surjection/bijection, totale/partielle, lambda-expression, fonction constante, transformée en fonction, évaluation, conversion
- Entier : entiers relatifs, entiers naturels, intervalles, entier représentable, fonction successeur/prédécesseur, opérations arithmétiques, somme/produit d'expressions quantifiées, cardinal, minimum/maximum
- Suite : suites, suites injectives/bijectives, ajout d'un élément à gauche/droite, longueur, concaténation, inversion, restriction à partir du bas/haut, premier/dernier élément, sans premier/dernier élément, somme/produit des éléments, tri, tassage, ordre lexicographique

Modèles

- Substitution généralisée : substitution simple/multiple/sans effet, séquence, parallélisme, pré-condition, garde, variable locale, choix borné/non borné, garde complexe, condition, condition par cas, itération ;
  prédicat avant-après, terminaison/non-terminaison, faisabilité/miracle, forme normale
- Machine abstraite : nom (et paramètres avec leurs contraintes), déclarations (ensembles, constantes et leurs propriétés, variables et l'invariant), initialisation (i.e. substitution), opérations (i.e. substitutions)
  NB : obligation de preuve (l'initialisation doit préserver l'invariant tout comme l'exécution de chaque opération)

Modèles

```
MACHINE
   ADDITION /* addition d'un chiffre à un entier naturel */
VARIABLES
   x
INVARIANT
   x ∈ ℕ
INITIALISATION
   x := 42
OPERATIONS
   ajouter_chiffre(delta) ≙    PRE  delta ∈ 0..9
                               THEN  x := x + delta
                               END
END
```

Idée de la preuve de la machine abstraite : *ADDITION*

Initialisation

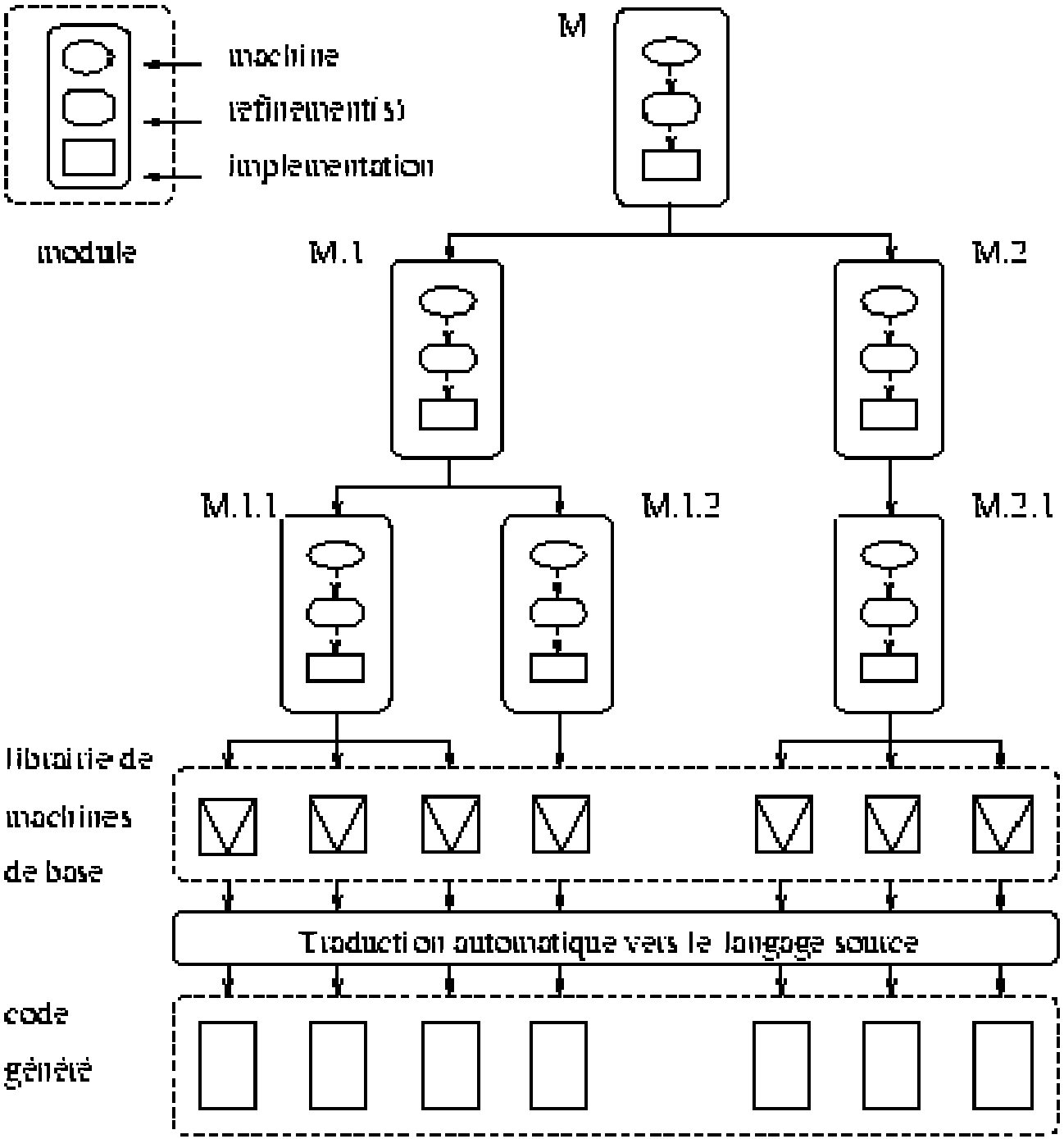
42 est un entier naturel ($\mathbb{N}$)

Opération *ajouter_chiffre*

la somme ($+$) de deux entiers naturels (x et $delta$) est un entier naturel ($\mathbb{N}$)

Modèles

- Contraintes dynamiques
- Raffinement : d'une machine ou d'un raffinement
  NB : passage progressif d'un modèle abstrait à une réalisation concrète
  Objectifs : diminution du non déterminisme, affaiblissement des pré-conditions
- Implantation : d'une machine ou d'un raffinement
- Module : composé de composants i.e. une machine, son(ses) aucun/un/plusieurs raffinement(s), son implantation, son code exécutable
  NB : assemblage (inclure, importer, voir et utiliser) de composants


machine
raffinement(s)
implementation
module
M
M.1
M.2
M.1.1
M.1.2
M.2.1
librairie de
machines
de base
Traduction automatique vers le langage source
code
généré

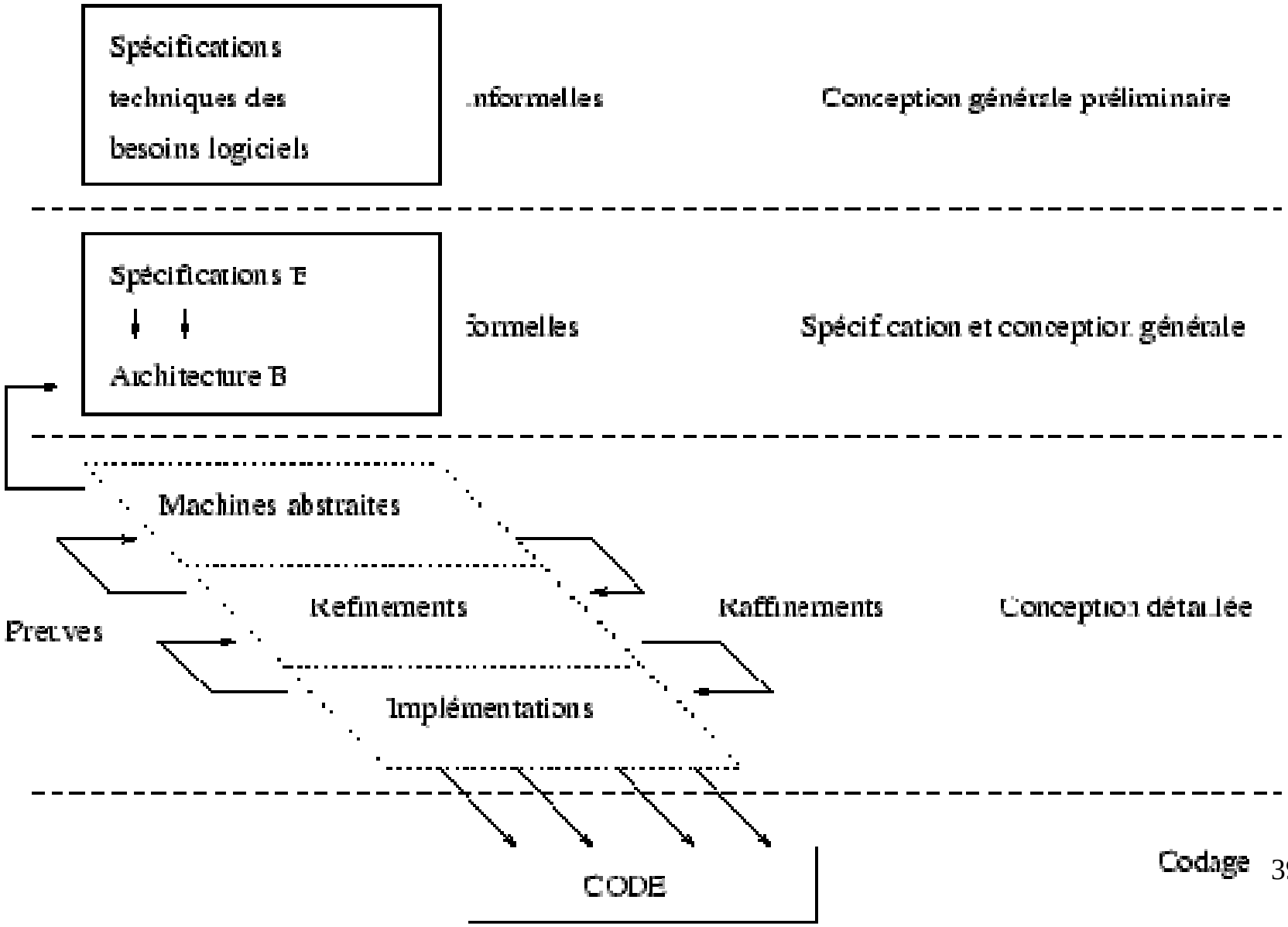
Démarche

Démarche

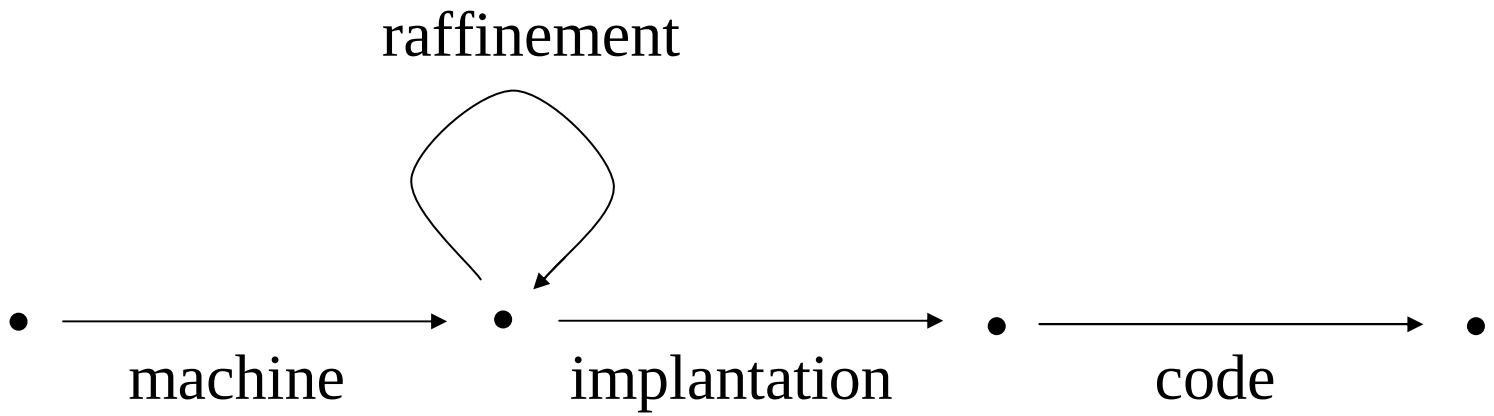
Description d'un cycle de développement

- Cela débute par la construction d’un modèle (décrit avec la notation B) reprenant toutes les descriptions du besoin. D'autres modèles sont ensuite élaborés par étapes (toujours à l’aide du langage B) jusqu’à l’obtention du programme exécutable.
- La cohérence des modèles aux différentes étapes et la conformité du programme au modèle initial, sont garanties par des preuves.
  À l’ultime étape, les programmes réalisés sont corrects par construction et rendent inutiles les tests destinés à éliminer les erreurs de programmation ou à vérifier leur conformité au modèle.
  Seuls subsistent les tests d'intégration du logiciel dans son environnement matériel et logiciel.

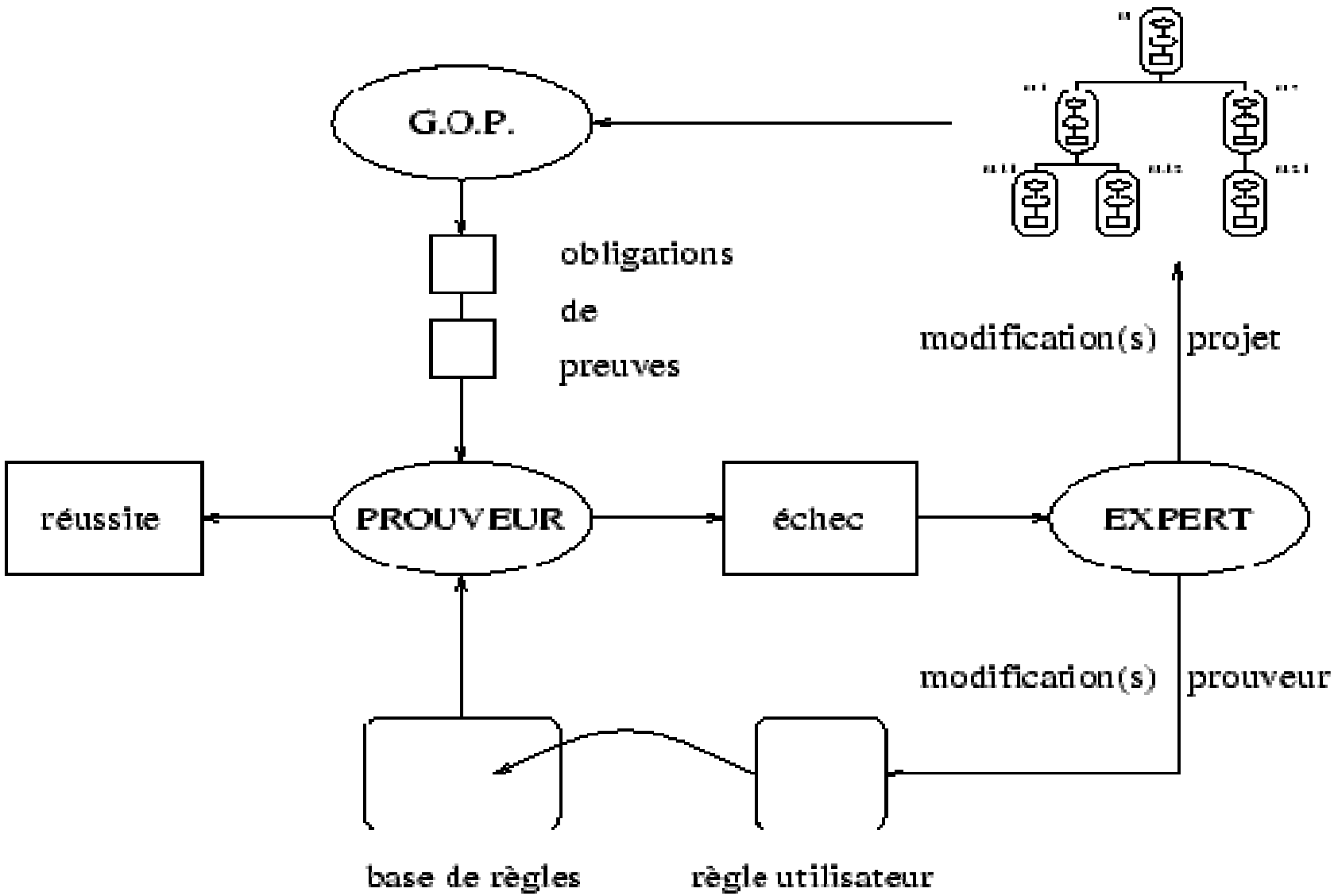
Démarche : Processus B



Démarche : Raffinement(s)



Démarche : Dynamique du processus de preuve

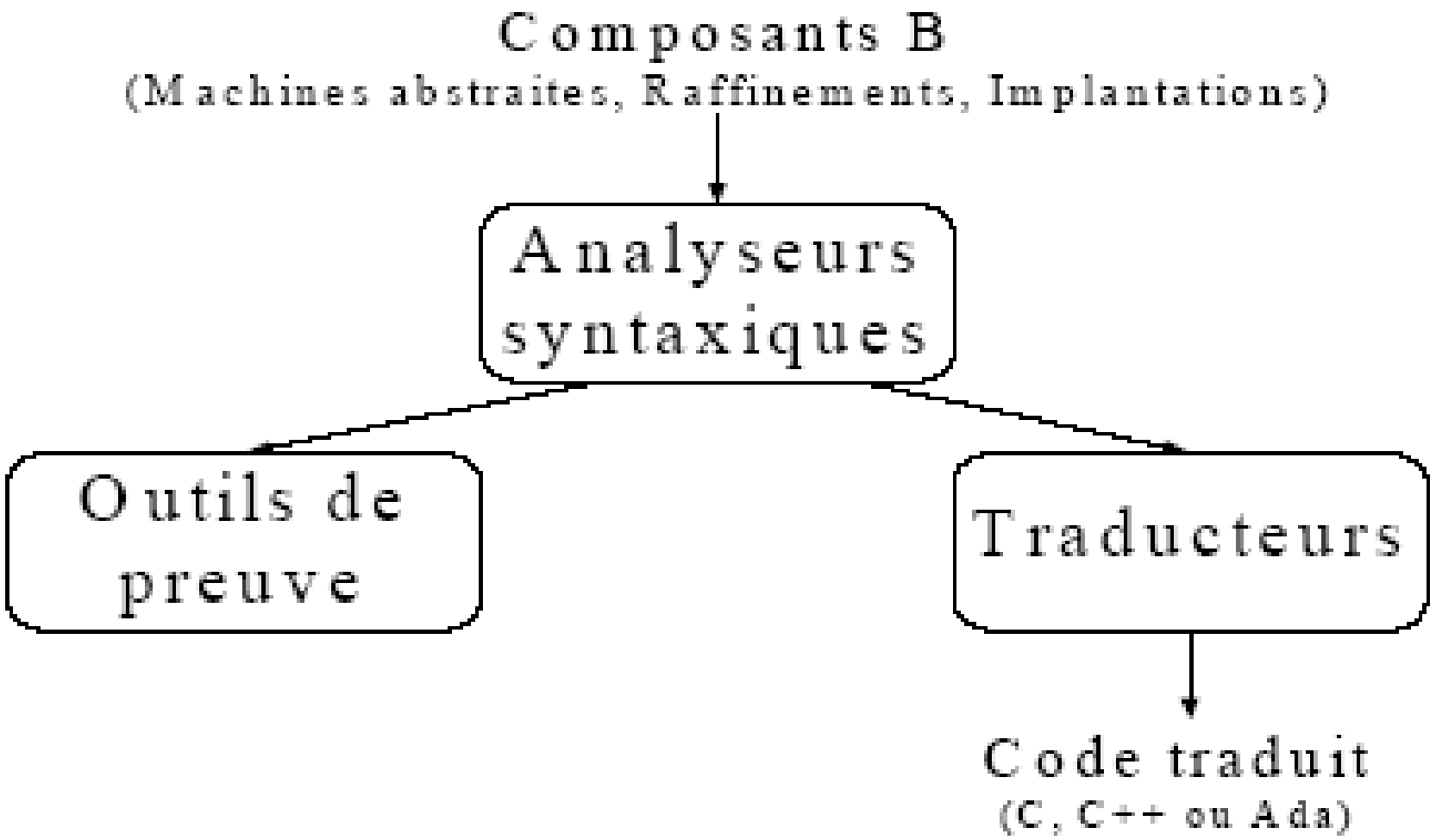


Démarche

- 80 % à 90 % des preuves sont prouvées automatiquement : restent les plus difficiles
  Ex. : sur les 2200 règles du prouveur de l'atelier B, 300 l'ont été par l'homme (les autres par un prouveur de prédicat) … et des erreurs ont été découvertes !

Démarche

Fonctionnalités de l'atelier B (ClearSy)



Démarche

Fonctionnalités de l'atelier B (ClearSy)

- Analyseur syntaxique
  - *Type checker* = vérification grammaticale et vérifications contextuelles (contrôles de type et de portée des identificateurs)
  - B0 *checker* = vérification spécifique du langage B0 utilisé dans les implantations
  - Vérifieur de projet i.e. de l'ensemble de ses composants
- Outils de preuve
  - Génération (et visualisation) automatique des obligations de preuve
  - Preuves automatiques (base de plus de 2200 règles totalement démontrée par un comité interindustriel d'experts français de B)
  - Preuves interactives i.e. le programmeur guide le prouveur dans sa démonstration d'une obligation de preuve
  - Prouveur de prédicats
- Traducteurs
  - Des implantations de B en langage cible (C, C++, ADA)

Modèles

Logique

Preuve

Logique

P	Q	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
vrai	vrai	vrai	vrai	vrai	vrai
vrai	faux	faux	vrai	faux	faux
faux	vrai	faux	vrai	vrai	faux
faux	faux	faux	faux	vrai	vrai

- $\neg$ *prédicat* négation
- *prédicat* $\wedge$ *prédicat* et logique
- *prédicat* $\vee$ *prédicat* ou logique
  NB : $P \vee Q \equiv \neg (\neg P \wedge \neg Q)$
- *prédicat* $\Rightarrow$ *prédicat* implication
  NB : $P \Rightarrow Q \equiv \neg P \vee Q$
- *prédicat* $\Leftrightarrow$ *prédicat* équivalence
  NB : $P \Leftrightarrow Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$

Logique

- [*variable* := *expression*] *prédicat* substitution (d'un prédicat en proposition)
  Ex. : [h:=Socrate](h est homme $\Rightarrow$ h est mortel) devient Socrate est homme $\Rightarrow$ Socrate est mortel
  Ex. : du prédicat $n \in \mathbb{N}^* \Rightarrow n \geq 1$ en opérant la substitution x:=8 à la proposition $8 \in \mathbb{N}^* \Rightarrow 8 \geq 1$
- $\forall$ *identificateurs* • *prédicat* quantificateur universel (pour tout)
  Ex. : $\forall$ h • (h $\in$ HOMME $\Rightarrow$ h $\in$ MORTEL)
- $\exists$ *identificateurs* • *prédicat* quantificateur existentiel (il existe)
  NB : $\exists$ x • P $\equiv \neg \forall$ x • $\neg$ P

Preuve

- *prédicat* ⊢ *prédicat* séquent

  H ⊢ P
  se lit « H entraîne P » ou « prouver P sous les hypothèses H » ou « à partir de H on veut déduire P »

  H : 0, 1 ou plusieurs hypothèses
  P : la conjecture

  Ex. :

  – 8>5,5>0 ⊢ 8>0
  – §₪½,½₪å,½₪♣ ⊢ §₪å,§₪♣

Preuve

- $\dfrac{\textit{séquents}}{\textit{séquent}}$ règle d'inférence

  $\dfrac{\Sigma_1, \ldots, \Sigma_n}{\Sigma}$ antécédents ($n \in \mathbb{N}$)

  Σ le conséquent

  se lit « des preuves de $\Sigma_1, \ldots, \Sigma_n$ on peut déduire la preuve de Σ »

  Ex. : $\dfrac{\vdash 5>0 \quad 8>5,5>0 \vdash 8>0}{8>5 \vdash 8>0}$

Preuve

- Remarque : un théorème est une conjecture prouvée (par exemple quand il n'y a pas d'hypothèses)
- Remarque : un axiome est une règle sans antécédent et dont le conséquent n'a pas d'hypothèses
- Remarque : on parle de collection d'hypothèses contradictoires si les deux séquents $H \vdash P$ et $H \vdash \neg P$ sont prouvables

Preuve

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

- $$\frac{}{H,P \vdash P}\ \text{HYP (hypothèse)}$$

Preuve

- $$\frac{H \vdash P}{H,Q \vdash P}\ \text{MON (monotonie)}$$

- $$\frac{H \vdash P \quad H,P \vdash Q}{H \vdash Q}\ \text{CUT (coupure)}$$

Preuve

- $\dfrac{H \vdash P \quad H \vdash Q}{H \vdash P \wedge Q}$ R1 ou CNJ (conjonction)

- $\dfrac{H \vdash P \wedge Q}{H \vdash P}$ R2 (conjonction)

- $\dfrac{H \vdash P \wedge Q}{H \vdash Q}$ R2’ (conjonction)

Preuve

- $\dfrac{H,P,Q \vdash R}{H,P \wedge Q \vdash R}$ CNJH (conjonction sous hyp.)

  règle dérivée

- $\dfrac{H,P \vdash Q}{H \vdash P \Rightarrow Q}$ DED ou R3 (déduction, implication)

- $\dfrac{H \vdash P \Rightarrow Q}{H,P \vdash Q}$ R4 (implication)

Preuve

- $\dfrac{H \vdash P \quad H \vdash P \Rightarrow Q}{H \vdash Q}$ MP (*modus ponens*)

  règle dérivée (R4 et CUT)

Preuve que MP *se dérive de* R4 *et* CUT *:*

$$\dfrac{\dfrac{H \vdash P \quad H \vdash P \Rightarrow Q}{H,P \vdash Q}\ R4}{H \vdash Q}\ CUT$$

Preuve

- $\dfrac{H,P,Q \vdash R}{H,P \Rightarrow Q,P \vdash R}$ MPH (*modus ponens* sous hypoth.)

Preuve

- $\dfrac{H,\neg Q \vdash P \quad H,\neg Q \vdash \neg P}{H \vdash Q}$ R5 (négation)

- $\dfrac{H,Q \vdash P \quad H,Q \vdash \neg P}{H \vdash \neg Q}$ R6 (négation)

- Raisonnement par l'absurde : pour prouver Q sous H, supposer ¬Q et montrer qu'il y a contradiction

Preuve

- $\dfrac{H,\neg Q \vdash P}{H,\neg P \vdash Q}$ CTR1 règle dérivée

- $\dfrac{H,Q \vdash \neg P}{H,P \vdash \neg Q}$ CTR2 règle dérivée

- $\dfrac{H,Q \vdash P}{H,\neg P \vdash \neg Q}$ CTR3 règle dérivée

Preuve

- Utilisation de la règle d'inférence des antécédents vers le conséquent (usage par l'avant) : $\Sigma_1, \ldots, \Sigma_n$ sont prouvés ; on en déduit (dérive) la preuve de Σ

- Preuve d'un séquent initial : stratégie par l'avant

  Tant que le séquent initial n'est pas prouvé et il y a au moins une règle dont les antécédents sont tous prouvés Faire
  Choisir l'une quelconque des règles dont les antécédents sont tous prouvés
  Ajouter le conséquent de la règle aux séquents prouvés
  // le séquent initial est-il prouvé ?

Preuve

Du théorème $P \vdash Q$ on en dérive $\dfrac{H \vdash P}{H \vdash Q}$

<u>Preuve</u> : (stratégie par l'avant)

On a $P \vdash Q$ et on considère $H \vdash P$

On applique MON à $P \vdash Q$ et on obtient $P,H \vdash Q$

On applique CUT à $H \vdash P$ et $P,H \vdash Q$ et on obtient $H \vdash Q$

<u>Ex.</u> : de $P \vdash \neg\neg P$ on dérive $\dfrac{H \vdash P}{H \vdash \neg\neg P}$ D1

Preuve

- $$\frac{H \vdash P}{H \vdash \neg\neg P}\ D1$$

- $$\frac{H \vdash P \wedge \neg Q}{H \vdash \neg(P \Rightarrow Q)}\ D2$$

- $$\frac{H \vdash P \Rightarrow \neg Q}{H \vdash \neg(P \wedge Q)}\ D3$$

Preuve

- $$\frac{H \vdash P \Rightarrow Q}{H \vdash \neg\neg P \Rightarrow Q}\ D4$$

- $$\frac{H \vdash P \Rightarrow (\neg Q \Rightarrow R)}{H \vdash \neg(P \Rightarrow Q) \Rightarrow R}\ D5$$

- $$\frac{H \vdash (\neg P \Rightarrow R) \wedge (\neg Q \Rightarrow R)}{H \vdash \neg(P \wedge Q) \Rightarrow R}\ D6$$

Preuve

- $$\frac{H \vdash (\neg P \Rightarrow R) \wedge (Q \Rightarrow R)}{H \vdash (P \Rightarrow Q) \Rightarrow R}\ D7$$

- $$\frac{H \vdash P \Rightarrow (Q \Rightarrow R)}{H \vdash P \wedge Q \Rightarrow R}\ D8$$

Preuve

- $$\frac{}{H,P \vdash \neg P \Rightarrow Q}\ CT1$$ règle dérivée

- $$\frac{}{H,\neg P \vdash P \Rightarrow Q}\ CT2$$ règle dérivée

- $$\frac{H,P \vdash R \quad H,Q \vdash R}{H,P \vee Q \vdash R}\ CAS$$ (disjonction des cas)

Preuve

- Utilisation de la règle d'inférence du conséquent vers les antécédents (usage par l'arrière) : on veut prouver Σ qu'on réduit à prouver $\Sigma_1, \ldots, \Sigma_n$

- Preuve d'un séquent initial : stratégie par l'arrière

Placer le séquent initial dans les séquents à prouver
Tant qu'il y a des séquents à prouver et il y a au moins une règle dont le conséquent fait partie des séquents à prouver Faire
    Choisir l'une quelconque des règles dont le conséquent fait partie des séquents à prouver
    Ajouter les antécédents de la règle aux séquents à prouver
    Enlever le conséquent de la règle des séquents à prouver
// reste-t-il des séquents à prouver ?

Preuve

- Exemple : on veut prouver $H \vdash P \Rightarrow \neg\neg P$

  Preuve : (stratégie par l'arrière)
  On place $H \vdash P \Rightarrow \neg\neg P$ dans les séquents à prouver
  On choisit DED et on obtient $H,P \vdash \neg\neg P$ comme séquent à prouver
  Soit on choisit R6 et on obtient $H,P,\neg P \vdash R$ et $H,P,\neg P \vdash \neg R$ comme séquents à prouver ; or, R est P ; on choisit HYP pour chacun des deux séquents à prouver et il n'y a plus de séquents à prouver
  Soit on choisit CTR2 et on obtient $H,\neg P \vdash \neg P$ comme séquent à prouver ; on choisit HYP pour $H,\neg P \vdash \neg P$ et il n'y a plus de séquents à prouver

Preuve

- Tactique de preuve : on applique en stratégie par l'arrière uniquement et dans l'ordre (en recommençant à chaque étape) CT1, CT2, HYP, D1, …, D8, CNJ, DED
- Remarque : c'est une procédure de décision (en calcul des propositions) car on arrive à un résultat qui dit que ce que l'on veut démontrer est ou n'est pas un théorème (on parle de semi-décision lorsque la procédure produit une réponse uniquement dans le cas où ce que l'on veut démontrer est un théorème … et ne s'arrête pas lorsque ce que l'on veut démontrer n'est pas un théorème)
  La logique des propositions (sans variable) est décidable tandis que la logique des prédicats (avec variables) est semi-décidable

Preuve

- $$\frac{H \vdash P}{H \vdash \forall\, x \bullet P}$$ GEN ou R7 (généralisation)

  x lié (non libre) dans tout Q de H

- $$\frac{H \vdash \forall\, x \bullet P}{H \vdash [x:=E]\, P}$$ R8 (spécialisation, substitution)

- $$\frac{H, \forall\, x \bullet P, [x:=E]\, P \vdash Q}{H, \forall\, x \bullet P \vdash Q}$$ INST (instanciation)

  règle dérivée

Preuve

- $\dfrac{H \vdash [x:=E]\, P}{H \vdash \exists\, x \bullet P}$ D13 règle dérivée

- …/… (D9, D10, D11, D12, D14, D15, R9, E10, EQ1, EQ2, EQL)

Ensemble

Ensemble

- $\varnothing$ ensemble vide
- *ensemble* $\times$ *ensemble* produit cartésien
- {*expressions*} ensemble défini en extension
  Ex. : AMIS = {Pierre, Paul, Jacques}
- {*identificateurs* | *prédicat*} ensemble défini en compréhension

Ensemble

- $\mathbb{P}$(*ensemble*) parties (i.e. sous-ensembles)
  Remarque : un ensemble de n éléments a 2^n parties
- $\mathbb{P}_1$(*ensemble*) parties sauf l'ensemble vide
- $\mathbb{F}$ (*ensemble*) parties finies
- $\mathbb{F}_1$(*ensemble*) parties finies sauf l'ensemble vide

Ensemble

- *expression* $\in$ *ensemble* appartenance

- *expression* $\notin$ *ensemble* non appartenance
  NB : $e \notin E \equiv \neg (e \in E)$

Ensemble

- *ensemble* $\subseteq$ *ensemble* inclusion
  NB : $S \subseteq E \equiv S \in \mathbb{P}(E)$

- *ensemble* $\nsubseteq$ *ensemble* non inclusion
  NB : $S \nsubseteq E \equiv \neg (S \subseteq E)$

- *ensemble* $\subset$ *ensemble* inclusion stricte
  NB : $S \subset E \equiv S \subseteq E \wedge S \neq E$

- *ensemble* $\not\subset$ *ensemble* non inclusion stricte
  NB : $S \not\subset E \equiv \neg (S \subset E)$

Ensemble

- *ensemble* $\cup$ *ensemble* union
  NB : $E1 \cup E2 \equiv \{x \mid x \in E \wedge (x \in E1 \vee x \in E2)\}$

- *ensemble* $\cap$ *ensemble* intersection
  NB : $E1 \cap E2 \equiv \{x \mid x \in E \wedge (x \in E1 \wedge x \in E2)\}$

- *ensemble* – *ensemble* différence
  NB : $E1 - E2 \equiv \{x \mid x \in E \wedge (x \in E1 \wedge x \notin E2)\}$

Ensemble

- `union`(*ensemble_d'ensembles*) union généralisée
  NB : `union`(E) ≡ {x | x ∈ F ∧ ∃ S • (S ∈ E ∧ x ∈ S)}
  Ex. : `union`({{1,2},{2,3}}) ={1,2,3}

- `inter`(*ensemble_d'ensembles*) intersection généralisée
  NB : `inter`(E) ≡ {x | x ∈ F ∧ ∀ S • (S ∈ E ⇒ x ∈ S)}

Ensemble

- $\bigcup$ *identificateurs* • (*prédicat* | *ensemble*) union quantifiée

  NB : $\bigcup x \bullet (P \mid E) \equiv \{y \mid y \in F \wedge \exists x \bullet (P \wedge y \in E)\}$

  Ex. : $\bigcup x \bullet (x \in \{1,3\} \mid \{y \mid y \in \mathbb{N}^* \wedge x\text{-}1 \leq y \leq x+1\}) = \{1,2\} \cup \{2,3,4\} = \{1,2,3,4\}$

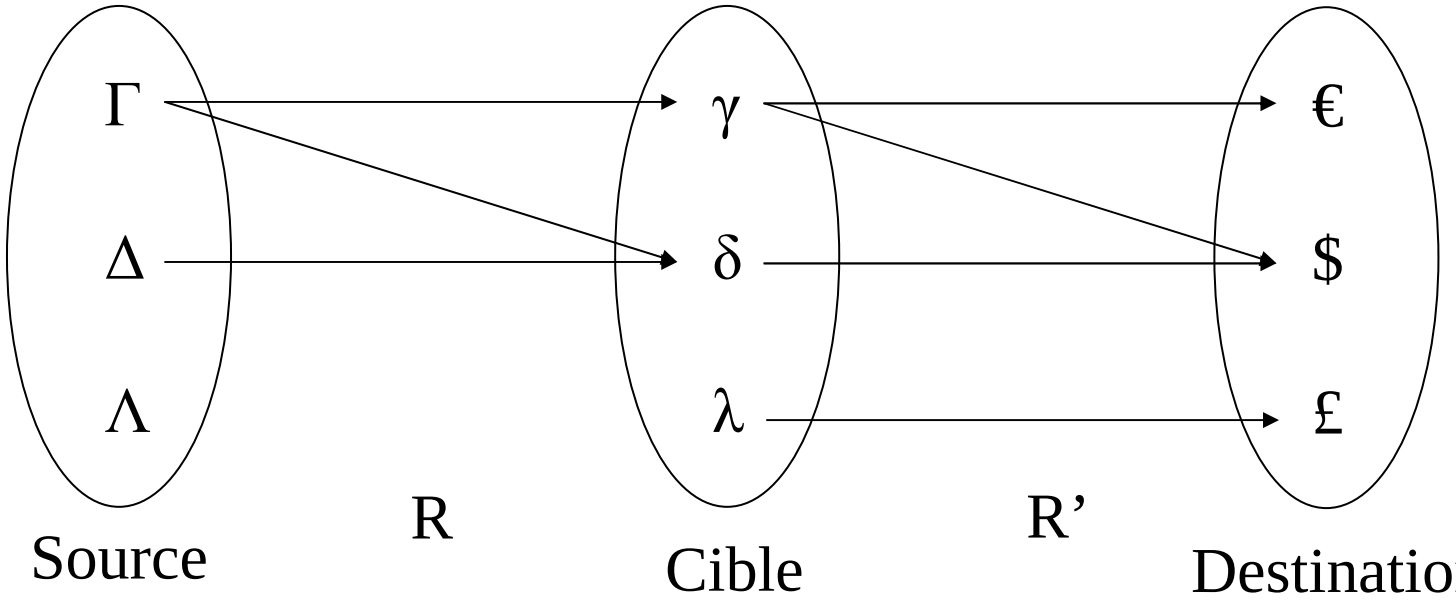
- $\bigcap$ *identificateurs* • (*prédicat* | *ensemble*) intersection quantifiée

  NB : $\bigcap x \bullet (P \mid E) \equiv \{y \mid y \in F \wedge \forall x \bullet (P \Rightarrow y \in E)\}$

Relation

Fonction

Relation



- *ensemble* ↔ *ensemble* relation (<- -> en ASCII)
  NB : E1 ↔ E2 ≡ ℙ(E1×E2)
  Attention : E1 ↔ E2 ≠ E2 ↔ E1
  Ex. : R = {(Γ,γ), (Γ,δ), (Δ,δ)} = {Γ ↦ γ, Γ ↦ δ, Δ ↦ δ}

Relation

- dom(*relation*) domaine
  NB : soit $R \in E1 \leftrightarrow E2$. $\mathrm{dom}(R) \equiv \{x \mid x \in E1 \wedge \exists y \bullet (y \in E2 \wedge (x \mapsto y) \in R)\}$
  Ex. : $\mathrm{dom}(R) = \{\Gamma, \Delta\}$

- ran(*relation*) codomaine (*range*)
  NB : soit $R \in E1 \leftrightarrow E2$. $\mathrm{ran}(R) \equiv \{y \mid y \in E2 \wedge \exists x \bullet (x \in E1 \wedge (x \mapsto y) \in R)\}$
  Ex. : $\mathrm{ran}(R) = \{\gamma, \delta\}$

- *relation* [*sous-ensemble*] image
  NB : soient $R \in E1 \leftrightarrow E2$ et $S \subseteq E1$.
  $R[S] \equiv \{y \mid y \in E2 \wedge \exists x \bullet (x \in S \wedge (x \mapsto y) \in R)\}$
  Ex. : $R[\{\Delta\}] = \{\delta\}$

Relation

- *relation*$^{-1}$ relation inverse
  NB : soit $R \in E1 \leftrightarrow E2$.
  $R^{-1} \equiv \{y,x \mid (y \mapsto x) \in E2 \times E1 \wedge (x \mapsto y) \in R)\}$
  Ex. : $R^{-1} = \{\gamma \mapsto \Gamma, \delta \mapsto \Gamma, \delta \mapsto \Delta\}$

- `id`(*ensemble*) identité
  NB : `id`$(E) \equiv \{x,y \mid (x \mapsto y) \in E \times E \wedge x = y\}$
  Ex. : `id`$(Source) = \{\Gamma \mapsto \Gamma, \Delta \mapsto \Delta, \Lambda \mapsto \Lambda\}$

- *relation* ; *relation* composition séquentielle
  NB : soient $R1 \in E1 \leftrightarrow E2$ et $R2 \in E2 \leftrightarrow E3$.
  $R1 ; R2 \equiv \{x,z \mid x,z \in E1 \times E3 \wedge \exists y \bullet (y \in E2 \wedge (x \mapsto y) \in R1 \wedge (y \mapsto z) \in R2)\}$
  Ex. : R ; R' = $\{\Gamma \mapsto €, \Gamma \mapsto \$, \Delta \mapsto \$\}$

Relation

- *relation*^{n} itération n fois d'une relation

  NB : soit $R \in E \leftrightarrow E$.

  $R^0 \equiv \mathtt{id}(E)$.

  $R^n \equiv R ; R^{n-1}$ si $n>0$ i.e. $R^n \equiv \{x,y \mid \exists\, y_0,y_1,\ldots,y_n \bullet (x=y_0 \wedge y_{i-1} \mapsto y_i \in R \;\forall\; 1\leq i\leq n \wedge y_n=y)\}$

  Ex. : soient $E = \{1,2,3,4\}$ et $R = \{1 \mapsto 1, 1 \mapsto 2, 2 \mapsto 1, 2 \mapsto 3\}$. $R^0 = \{1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3, 4 \mapsto 4\}$, $R^1 = R$, $R^2 = \{1 \mapsto 1, 1 \mapsto 2, 1 \mapsto 3, 2 \mapsto 1, 2 \mapsto 2\}$, $R^3 = \{1 \mapsto 1, 1 \mapsto 2, 1 \mapsto 3, 2 \mapsto 1, 2 \mapsto 2, 2 \mapsto 3\}$, $R^i = R^3 \;\forall\; i>3$

Relation

- *relation*$^{+}$ fermeture transitive

  NB : $R^{+} \equiv \bigcup n \bullet (n \in \mathbb{N}^{*} \mid R^{n})$
  Ex. : soient E = {1,2,3,4} et R = {1 ↦ 1, 1 ↦ 2, 2 ↦ 1, 2 ↦ 3}. R^{+} = {1 ↦ 1, 1 ↦ 2, 1 ↦ 3, 2 ↦ 1, 2 ↦ 2, 2 ↦ 3}

- *relation** fermeture réflexive transitive

  NB : $R^{*} \equiv \bigcup n \bullet (n \in \mathbb{N} \mid R^{n}) \equiv R^{0} \cup R^{+}$
  Ex. : soient E = {1,2,3,4} et R = {1 ↦ 1, 1 ↦ 2, 2 ↦ 1, 2 ↦ 3}. R^{*} = {1 ↦ 1, 1 ↦ 2, 1 ↦ 3, 2 ↦ 1, 2 ↦ 2, 2 ↦ 3, 3 ↦ 3, 4 ↦ 4}

Relation

- *relation* $\otimes$ *relation* produit direct
  NB : soient $R1 \in E1 \leftrightarrow E2$ et $R2 \in E1 \leftrightarrow E3$.
  $R1 \otimes R2 \equiv \{x,(y,z) \mid x,(y,z) \in E1 \times (E2 \times E3) \wedge (x \mapsto y) \in R1 \wedge (x \mapsto z) \in R2\}$
  Ex. : $R^{-1} \otimes R' = \{\gamma \mapsto (\Gamma,€), \gamma \mapsto (\Gamma,\$), \delta \mapsto (\Gamma,\$), \delta \mapsto (\Delta,\$)\}$

- *relation* $\|$ *relation* produit parallèle
  NB : soient $R1 \in E1 \leftrightarrow E2$ et $R2 \in E3 \leftrightarrow E4$.
  $R1 \| R2 \equiv \{(x,z),(y,t) \mid (x,z),(y,t) \in (E1 \times E3) \times (E2 \times E4) \wedge (x \mapsto y) \in R1 \wedge (z \mapsto t) \in R2\}$

Relation

- $\texttt{prj}_1$(*ensemble*,*ensemble*) 1re projection
  NB : $\texttt{prj}_1(E1,E2) \equiv \{(x,y),z \mid (x,y),z \in (E1 \times E2) \times E1 \wedge z = x\}$
  Ex. : $\texttt{prj}_1(\{1,2\},\{a,b\}) = \{(1 \mapsto a) \mapsto 1, (1 \mapsto b) \mapsto 1, (2 \mapsto a) \mapsto 2, (2 \mapsto b) \mapsto 2\}$

- $\texttt{prj}_2$(*ensemble*,*ensemble*) 2e projection
  NB : $\texttt{prj}_2(E1,E2) \equiv \{(x,y),z \mid (x,y),z \in (E1 \times E2) \times E2 \wedge z = y\}$
  Ex. : $\texttt{prj}_2(\{1,2\},\{a,b\}) = \{(1 \mapsto a) \mapsto a, (1 \mapsto b) \mapsto b, (2 \mapsto a) \mapsto a, (2 \mapsto b) \mapsto b\}$

Relation

- *sous-ensemble* $\lhd$ *relation* restriction de domaine (<| en ASCII)

  NB : soient $R1 \in E1 \leftrightarrow E2$ et $S1 \subseteq E1$.

  $S1 \lhd R1 \equiv \{x,y \mid (x \mapsto y) \in R1 \wedge x \in S1\}$

  Ex. : $\{\Gamma,\Lambda\} \lhd R = \{\Gamma \mapsto \gamma, \Gamma \mapsto \delta\}$

- *sous-ensemble* $\lhd\!\!\!-$ *relation* corestriction (i.e. soustraction) de domaine (<<| en ASCII)

  NB : soient $R1 \in E1 \leftrightarrow E2$ et $S1 \subseteq E1$.

  $S1 \lhd\!\!\!- R1 \equiv \{x,y \mid (x \mapsto y) \in R1 \wedge x \notin S1\}$

  Ex. : $\{\Gamma\} \lhd\!\!\!- R = \{\Delta \mapsto \delta\}$

Relation

- *relation* $\triangleright$ *sous-ensemble* restriction de codomaine ($|>$ en ASCII)

  NB : soient $R1 \in E1 \leftrightarrow E2$ et $S2 \subseteq E2$.

  $R1 \triangleright S2 \equiv \{x,y \mid (x \mapsto y) \in R1 \wedge y \in S2\}$

  Ex. : $R \triangleright \{\gamma,\lambda\} = \{\Gamma \mapsto \gamma\}$

- *relation* $\triangleright\!\!>$ *sous-ensemble* corestriction (i.e. soustraction) de codomaine ($|>>$ en ASCII)

  NB : soient $R1 \in E1 \leftrightarrow E2$ et $S2 \subseteq E2$.

  $R1 \triangleright\!\!> S2 \equiv \{x,y \mid (x \mapsto y) \in R1 \wedge y \notin S2\}$

  Ex. : $R \triangleright\!\!> \{\gamma\} = \{\Gamma \mapsto \delta, \Delta \mapsto \delta\}$

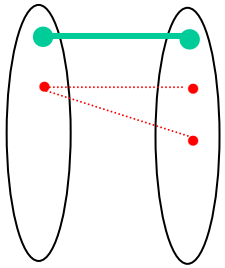
Relation

- *relation* $\lhd$- *relation* modification (i.e. surcharge)

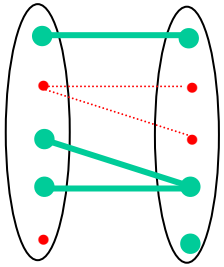
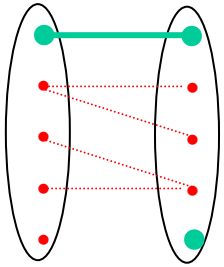
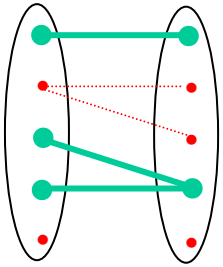
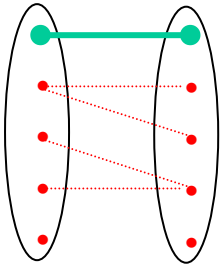
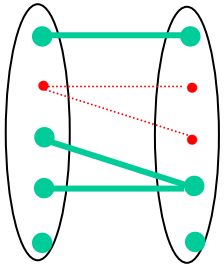
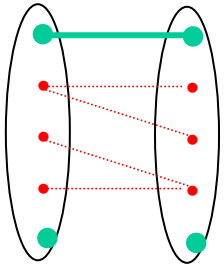
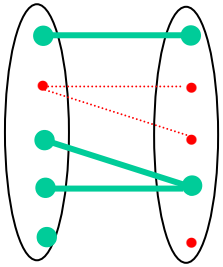
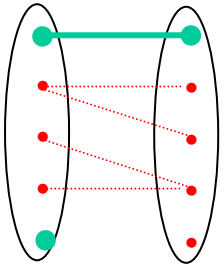
  NB : soient $R1 \in E1 \leftrightarrow E2$ et $R2 \in E1 \leftrightarrow E2$.

  $R1 \lhd\text{-} R2 \equiv \{x,y \mid (x \mapsto y) \in E1 \times E2 \wedge (((x \mapsto y) \in R1 \wedge x \notin \mathtt{dom}(R2)) \vee (x \mapsto y) \in R2)\}$

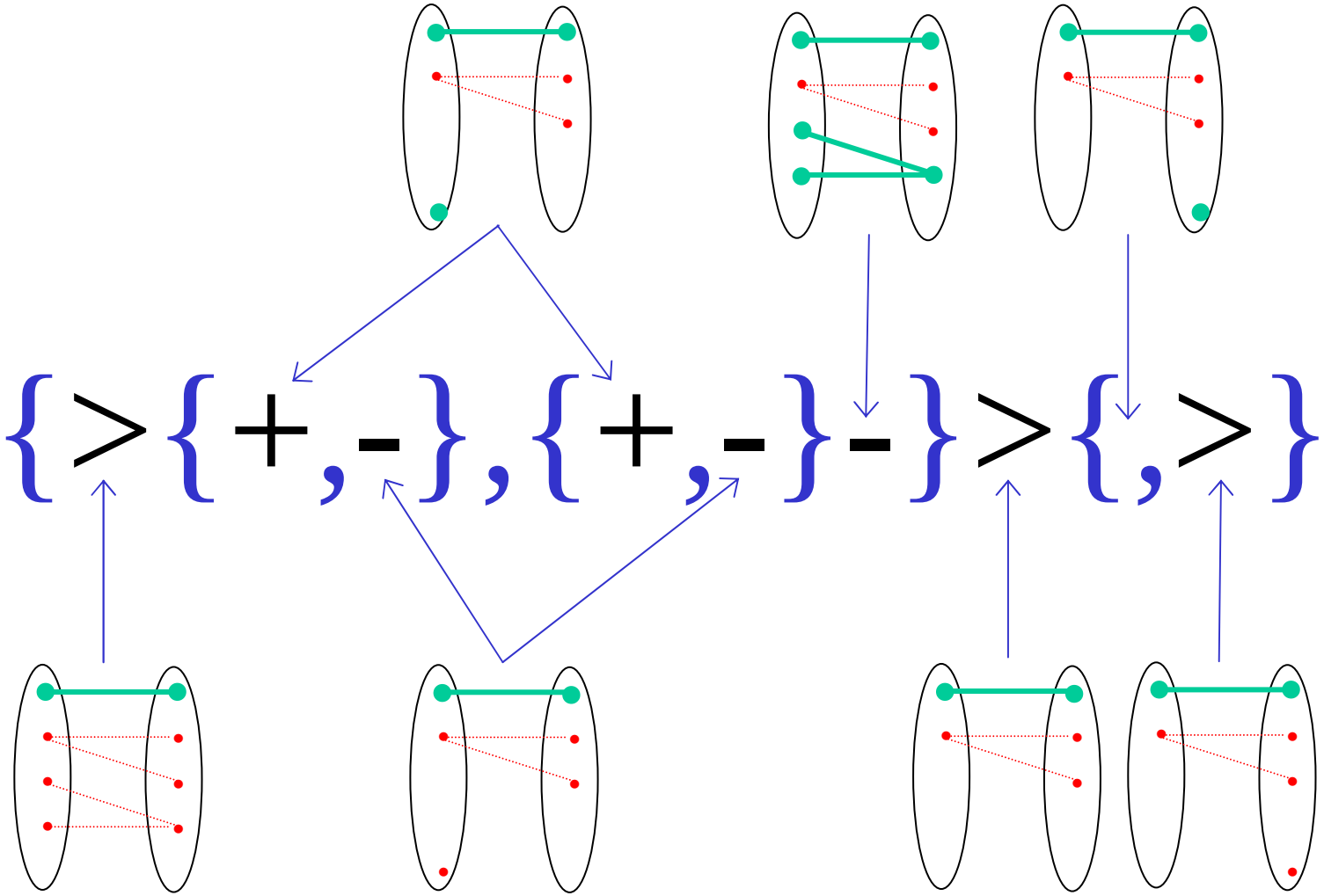
  Ex. : $R1 \lhd\text{-} \{\Gamma \mapsto \gamma, \Lambda \mapsto \delta\} = \{\Gamma \mapsto \gamma , \Delta \mapsto \delta, \Lambda \mapsto \delta\}$



Fonction (notation ASCII)

	fonction	injection	surjection	bijection
totale	--> 	>-> 	-->> 	>->> 
partielle	+-> 	>+> 	+->> 	>+>> 

Fonction (notation ASCII)



Fonction

- *ensemble* --> *ensemble* fonction totale
  NB : E1 --> E2 ≡ E1 → E2 ≡
  {R | R ∈ E1 +-> E2 ∧ dom(R) = E1}

- *ensemble* +-> *ensemble* fonction partielle
  NB : E1 +-> E2 ≡
  {R | R ∈ E1 ↔ E2 ∧ ∀ x,y,z • (((x ↦ y) ∈ R
  ∧ (x ↦ z) ∈ R) ⇒ y = z)}

Fonction

- *ensemble* >-> *ensemble* injection totale
  NB : E1 >-> E2 ≡ E1 ↣ E2 ≡
  (E1 >+> E2) ∩ (E1 --> E2)

- *ensemble* >+> *ensemble* injection partielle
  NB : E1 >+> E2 ≡
  $\{R \mid R \in E1 \text{ +-> } E2 \wedge R^{-1} \in E2 \text{ +-> } E1\}$

Fonction

- *ensemble* --\>\> *ensemble* surjection totale
  NB : E1 --\>\> E2 ≡ E1 ↠ E2 ≡
  (E1 +-\>\> E2) ∩ (E1 --\> E2)

- *ensemble* +-\>\> *ensemble* surjection partielle
  NB : E1 +-\>\> E2 ≡
  {R | R ∈ E1 +-\> E2 ∧ ran(R) = E2}

Fonction

- *ensemble* >->> *ensemble* bijection totale
  NB : E1 >->> E2 ≡
  (E1 >-> E2) ∩ (E1 -->> E2)

- *ensemble* >+>> *ensemble* bijection partielle
  NB : E1 >+>> E2 ≡
  (E1 >+> E2) ∩ (E1 +->> E2)

Fonction

- *fonction* $\circ$ *fonction* composition de fonctions
  NB : $(f \circ g)(x) \equiv g(f(x)) \equiv z \mid \exists y : f(x)=y \wedge g(y)=z$ avec $f \in X \rightarrow Y$ et $g \in Y \rightarrow Z$
  Propriétés :
  - généralement non commutative :
    $f \circ g \neq g \circ f$
  - associative :
    $f \circ (g \circ h) = (f \circ g) \circ h$
  - généralement non distributive (sur un opérateur $\blacksquare$ quelconque) :
    $f \circ (g \blacksquare h) \neq (f \circ g) \blacksquare (f \circ h)$

Relation et Fonction

- λ *identificateurs* • (*prédicat* | *expression*) lambda-expression
  Ex. : f(3)=5 pour $f = \lambda\, x \bullet (x \in \mathbb{N} \mid x+2)$

- *expression* $\times$ { *expression* } fonction constante
  Ex. : $\{x \in \mathbb{N}^* \wedge x \leq 3\} \times \{0\} = \{1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0\}$

- *expression* (*expression*) évaluation de fonction

Relation et Fonction

- `fnc`(*expression*) transformée en fonction
  Ex. : `fnc`(R) = {$\Gamma \mapsto \{\gamma, \delta\}, \Delta \mapsto \{\delta\}$}

- `rel`(*expression*) transformée en relation
  Ex. : `rel`({$\gamma \mapsto \{€,\$\}, \delta \mapsto \{\$\}, \lambda \mapsto \{£\}$}) = R'

- `bool`(*prédicat*) conversion d'un prédicat en une valeur booléenne de BOOL = {TRUE, FALSE}

Entier

Entier

- $\mathbb{Z}$ ensemble des entiers relatifs
- $\mathbb{N}$ ensemble des entiers positifs ou nul
  NB : $\mathbb{N} \equiv \{x \mid x \in \mathbb{Z} \wedge x \geq 0\}$
- $\mathbb{N}_1$ ensemble des entiers strictement positifs
  NB : $\mathbb{N}_1 \equiv \{x \mid x \in \mathbb{Z} \wedge x > 0\}$
- $n..m$ ensemble des entiers compris entre n et m
  NB : $n..m \equiv \{x \mid x \in \mathbb{Z} \wedge n \leq x \leq m\}$

Entier

- `minint` entier minimum représentable
- `maxint` entier strictement positif maximum représentable
- `INT` entiers relatifs représentables
  NB : `INT` ≡ `minint..maxint`
- `NAT` entiers positifs ou nul représentables
  NB : `NAT` ≡ 0`..maxint`
- $\texttt{NAT}_1$ entiers strictement positifs représentables
  NB : $\texttt{NAT}_1$ ≡ 1`..maxint`

Entier

- `succ`(*entier*) fonction successeur
- `pred`(*entier*) fonction prédécesseur

Entier

- - *entier* opposé
- *entier* + *entier* addition
- *entier* - *entier* soustraction
- *entier* * *entier* multiplication

Entier

- *entier* `/` *entier_non_nul* quotient de la division entière
- *entier* `mod` *entier_strictement_positif* reste de la division entière
- $entier\_positif\_ou\_nul^{entier\_positif\_ou\_nul}$ puissance

Entier

- $\sum$ *identificateurs* • (*prédicat* | *expression*)
  somme d'expressions quantifiées
  Ex. : $\sum x \bullet (x \in \{0,1,2,3\} \mid 2*x+1) = 16$

- $\prod$ *identificateurs* • (*prédicat* | *expression*)
  produit d'expressions quantifiées
  Ex. : $\prod x \bullet (x \in \{1,2,4,8\} \mid x) = 64$

Entier

- `card`(*expression*) cardinal (nombre d'éléments) d'un ensemble
- `min`(*expression*) minimum d'un ensemble fini non vide d'entiers
- `max`(*expression*) maximum d'un ensemble fini non vide d'entiers

Suite (*Sequence*)

Suite

- seq(*ensemble*) suites finies d'éléments d'un ensemble

  NB : $\mathsf{seq}(E) \equiv \bigcup n \bullet (n \in \mathbb{N} \mid 1..n \rightarrow E)$
  Ex. : $[\alpha, \beta, \gamma, \delta] = \{1 \mapsto \alpha, 2 \mapsto \beta, 3 \mapsto \gamma, 4 \mapsto \delta\}$
  Ex. : $[] \equiv 1..0 \rightarrow E$

Suite

- $\texttt{seq}_1$(*ensemble*) suites non vides
  NB : $\texttt{seq}_1\,(E) \equiv \texttt{seq}(E) - \varnothing$

- `iseq`(*ensemble*) suites injectives
  NB : $\texttt{iseq}\,(E) \equiv \texttt{seq}(E) \cap (\mathbb{N}_1 \mathrel{>\!\!+\!\!>} E)$

- $\texttt{iseq}_1$(*ensemble*) suites injectives non vides
  NB : $\texttt{iseq}_1\,(E) \equiv \texttt{iseq}(E) - \varnothing$

- `perm`(*ensemble*) suites bijectives (permutations)
  NB : $\texttt{perm}\,(E) \equiv \texttt{seq}(E) \cap (\mathbb{N}_1 \mathrel{>\!\!+\!\!>\!\!>} E)$

Suite

- *élément* `->` *suite* ajout d'un élément à gauche
  NB : soient $e \in E$ et $S \in \texttt{seq}(E)$. $e \texttt{ -> } S \equiv \{1 \mapsto e\}=[e]$ si $S=[]$, $\{1 \mapsto e\} \cup (\texttt{pred}; S)$ sinon
  En effet, `pred` est la relation des $\{i+1 \mapsto i\}$ que l'on compose avec $\{i \mapsto i^{e}$ élément de $S\}$.
  Ex. : $(\alpha \texttt{ -> } (\beta \texttt{ -> } (\gamma \texttt{ -> } (\delta \texttt{ -> } [])))) = (\alpha \texttt{ -> } (\beta \texttt{ -> } (\gamma \texttt{ -> } [\delta]))) = (\alpha \texttt{ -> } (\beta \texttt{ -> } [\gamma, \delta])) = (\alpha \texttt{ -> } [\beta, \gamma, \delta]) = [\alpha, \beta, \gamma, \delta]$

- *suite* `<-` *élément* ajout d'un élément à droite
  NB : $[] \texttt{ <- } e2 \equiv e2 \texttt{ -> } []$, $(e1 \texttt{ -> } S) \texttt{ <- } e2 \equiv e1 \texttt{ -> } (S \texttt{ <- } e2)$ pour une suite non vide
  Ex. : $((((([] \texttt{ <- } \alpha) \texttt{ <- } \beta) \texttt{ <- } \gamma) \texttt{ <- } \delta) = (\alpha \texttt{ -> } (\beta \texttt{ -> } (\gamma \texttt{ -> } (\delta \texttt{ -> } [])))) = [\alpha, \beta, \gamma, \delta]$

Suite

- `size`(*suite*) longueur de suite
  NB : `size` ([]) = 0, `size` (e `->` S) = `size`(S)+1
  Ex. : `size` ([α, β, γ, δ]) = 4

- *suite* ∩ *suite* concaténation de deux suites
  NB : S1 ∩ S2 ≡ S2 si S1=[], e `->` (S ∩ S2) si S1 = (e `->` S)
  Ex. : [α, β, γ, δ] ∩ [ε, φ] = [α, β, γ, δ, ε, φ]

- `conc`(*suite*) concaténation généralisée
  NB : `conc` ([]) = [], `conc`(S1 `->` S2) = S1 ∩ `conc`(S2)
  Ex. : `conc`([[α, β, γ, δ],[],[ε, φ]]) = [α, β, γ, δ, ε, φ]

Suite

- `rev`(*suite*) inversion de suite
  NB : `rev`([]) = [], `rev`(e `->` S) = `rev`(S) `<-` e
  Ex. : `rev`([α, β, γ, δ]) = [δ, γ, β, α]

- *suite* ↑ *entier* restriction à partir du bas (on garde les premiers éléments)
  NB : soit n ∈ 0..`size`(S). S ↑ n ≡ (1..n) ◁ S
  Ex. : [α, β, γ, δ] ↑ 3 = [α, β, γ]

- *suite* ↓ *entier* restriction à partir du haut (on supprime les premiers éléments)
  NB : soit n ∈ 0..`size`(S).
  S ↓ n ≡ λ i • (i ∈ 1..(`size`(S) - n) | S(n+i))
  Ex. : [α, β, γ, δ] ↓ 3 = [δ]

Suite

- `first`(*suite_non_vide*) premier élément
  NB : `first`(S) = S(1)
  Ex. : `first` ([α, β, γ, δ]) = α

- `last`(*suite_non_vide*) dernier élément
  NB : `last`(S) = S(`size`(S))
  Ex. : `last` ([α, β, γ, δ]) = δ

Suite

- `tail`(*suite_non_vide*) suite sans premier élément
  NB : `tail`(S) = S ↓ 1
  Ex. : `tail`([α, β, γ, δ]) = [β, γ, δ]

- `front`(*suite_non_vide*) suite sans dernier élément
  NB : `front`(S) = S ↑ (`size`(S) - 1)
  Ex. : `front`([α, β, γ, δ]) = [α, β, γ]

Suite

- `sum`(*suite*) somme des éléments d'une suite
  Ex. : `sum` ([2, 8, 1, 4]) = 15

- `prod`(*suite*) produit des éléments d'une suite
  Ex. : `prod` ([2, 8, 1, 4]) = 64

- `sort`(*suite*) tri des éléments d'une suite
  Ex. : `sort` ([2, 8, 1, 4]) = [1, 2, 4, 8]

- `sortset`(*ensemble*) tri des éléments d'un ensemble
  Ex. : `sort` ({2, 8, 1, 4}) = [1, 2, 4, 8]

Suite

- `squash`(*fonction*) tassage d'une fonction de domaine d'entiers
  NB : `squash` (F) ≡ `sortset`(`dom`(F)) ; F
  Ex. : `squash`($\{4 \mapsto \gamma, 8 \mapsto \beta, 2 \mapsto \delta, 5 \mapsto \alpha\}$) = `sortset`(`dom`($\{4 \mapsto \gamma, 8 \mapsto \beta, 2 \mapsto \delta, 5 \mapsto \alpha\}$)) ; $\{4 \mapsto \gamma, 8 \mapsto \beta, 2 \mapsto \delta, 5 \mapsto \alpha\}$ = `sortset`($\{4, 8, 2, 5\}$) ; $\{4 \mapsto \gamma, 8 \mapsto \beta, 2 \mapsto \delta, 5 \mapsto \alpha\}$ = $[2, 4, 5, 8]$; $\{4 \mapsto \gamma, 8 \mapsto \beta, 2 \mapsto \delta, 5 \mapsto \alpha\}$ = $\{1 \mapsto 2, 2 \mapsto 4, 3 \mapsto 5, 4 \mapsto 8\}$; $\{4 \mapsto \gamma, 8 \mapsto \beta, 2 \mapsto \delta, 5 \mapsto \alpha\}$ = $[\delta, \gamma, \alpha, \beta]$

Suite

- *suite* $\prec$ *suite* suite lexicographiquement strictement plus petite qu'une autre suite
  NB : soient S1 $\in$ `seq`($\mathbb{Z}$), S2 $\in$ `seq`($\mathbb{Z}$) et `size`(S1)=`size`(S2).
  S1 $\prec$ S2 $\equiv$ S1 $\neq$ S2 $\wedge$ S1(i)<S2(i) où i = `min`({n | n $\in$ `dom`(S1)} $\wedge$ S1(i) $\neq$ S2(i)})
  Ex. : [2, 3, 5, 8] $\prec$ [2, 4, 5, 7]

- *suite* $\preccurlyeq$ *suite* suite lexicographiquement égale ou strictement plus petite qu'une autre suite
  NB : soient S1 $\in$ `seq`($\mathbb{Z}$), S2 $\in$ `seq`($\mathbb{Z}$) et `size`(S1)=`size`(S2).
  S1 $\preccurlyeq$ S2 $\equiv$ S1 = S2 $\vee$ S1 $\prec$ S2

Substitution généralisée

Substitution généralisée

- P{S}R [Hoare] logique des programmes
  Si la pré-condition P (prédicat) est vraie et que l'instruction S se termine, alors la post-condition R (prédicat) est vraie après l'exécution de S

- On peut renforcer la pré-condition et/ou affaiblir la post-condition sans changer la validité d'une formule logique de programme i.e.

$$\frac{P' \Rightarrow P \quad P\{S\}R \quad R \Rightarrow R'}{P'\{S\}R'}$$

Substitution généralisée

- $wp_S(R)$ ou $wp(S,R)$ [Dijkstra] plus faible pré-condition (*weakest precondition*)
  Pour une post-condition R et une instruction S, quelle est la plus faible pré-condition P qui assure que S termine et que R est vraie après S ?

- On a $P\{S\}R \equiv P \Rightarrow wp_S(R)$ pour S qui termine

Substitution généralisée

- La sémantique des constructions de programme peut se définir comme la composition d'opérateurs *wp* (vu comme un transformateur de prédicat) et des expressions de prédicats
  Ex. : $wp(S1;S2) \equiv wp(S1) \circ wp(S2)$

- Toutes les constructions du langage (pour la spécification) fondé sur la logique sont des analogues à des substitutions : c'est le langage des substitutions généralisées
  Ex. : $wp_{x:=E}(R) \Leftrightarrow x:=E\ (R)$

Substitution généralisée

- *substitution* structuration des substitutions
  S ≡
  ```
  BEGIN
  S
  END
  ```
- *variable* := *expression* substitution simple
  Ex. : x := y+1
- *variables* := *expressions* substitution multiple
  $x_1,\ldots,x_n := E_1,\ldots,E_n$
- `skip` substitution sans effet
- *substitution* ; *substitution* séquence
  S1 ; S2

Substitution généralisée

- *substitution* || *substitution* parallélisme
  S1 || S2
- *prédicat* | *substitution* pré-condition
  P | S ≡
  ```
  PRE P
  THEN S
  END
  ```
  Ex. : x>0 | y:=y+x
- *prédicat* ⇒ *substitution* garde
  P ⇒ S ≡
  ```
  SELECT P
  THEN S
  END
  ```
  Ex. : x>0 ⇒ y:=y+x

Substitution généralisée

- P | P ⇒ S ≡
  ```
  ASSERT P
  THEN S
  END
  ```
- *@ variable • substitution* variable locale

  @ x • S ≡
  ```
  VAR x IN S
  END
  ```
  Ex. : @ x,y • d:=x-y
- *substitution* [] *substitution* choix borné (ou choix fermé)

  S1 [] S2 [] … [] Sk ≡
  ```
  CHOICE S1 OR S2 OR … OR Sk
  END
  ```
  Ex. : x:=1 [] y:=x+2

Substitution généralisée

- *@ variable • prédicat* $\Rightarrow$ *substitution* choix non borné (ou choix libre)

  @x•(P$\Rightarrow$S) ≡

  ```
  ANY x
  WHERE P
  THEN S
  END
  ```

  Ex. : @ x • x*(x+1)=2 $\Rightarrow$ x_1,x_2:=1,-2

Substitution généralisée

- LET $x_1,\ldots,x_n$
  BE $x_1=E_1 \wedge \ldots \wedge x_n=E_n$
  IN S
  END
  ≡
  ANY $x_1,\ldots,x_n$
  WHERE $x_1=E_1 \wedge \ldots \wedge x_n=E_n$
  THEN S
  END
  variables locales

Substitution généralisée

- SELECT P1 THEN S1
  WHEN P2 THEN S2
  ELSE S3
  END
  ≡
  CHOICE SELECT P1
  THEN S1
  END
  OR SELECT P2
  THEN S2
  END
  OR SELECT $\neg$ P1 $\wedge$ $\neg$ P2
  THEN S3
  END
  END
  garde complexe

Substitution généralisée

- $P \Rightarrow S1 \; [] \; \neg P \Rightarrow S2 \equiv$

```
IF P
THEN S1
ELSE S2
END
```

  condition

- ```
  IF P
  THEN S
  END
  ```

  $\equiv$

  ```
  IF P
  THEN S
  ELSE skip
  END
  ```
  ```

Substitution généralisée

- ```
  IF     P1 THEN S1
  ELSIF  P2 THEN S2
  ELSE           S3
  END
  ≡
  IF P1
  THEN S1
  ELSE IF P2
       THEN S2
       ELSE S3
       END
  END
  ```
  ```

Substitution généralisée

- CASE E OF
  EITHER C1 THEN S1
  OR C2 THEN S2
  …
  OR Ck THEN Sk
  ELSE S
  END
  ≡
  SELECT E ∈ {C1} THEN S1
  WHEN E ∈ {C2} THEN S2
  …
  WHEN E ∈ {Ck} THEN Sk
  ELSE S
  END
  condition par cas

Substitution généralisée

Ex. :

```
CASE ch OF
    EITHER 1 THEN
           x:=1
    OR [1,2,3] THEN
           y:=123
    ELSE
           z:=0
END
```

≡

```
SELECT ch=1 THEN
    x:=1
OR ch ∈ {1,2,3} THEN
    y:=123
ELSE
    z:=0
END
```

≡

```
IF ch=1 THEN
    x:=1
ELSIF ch ∈ {1,2,3}
THEN
    y:=123
ELSE
    z:=0
END
```

≡

```
IF ch=1
THEN x:=1
ELSE IF ch ∈ {1,2,3}
       THEN y:=123
       ELSE z:=0
       END
END
```

Substitution généralisée

- *W(prédicat,substitution,invariant,variant)* itération

  $W(P,S,I,V) \equiv$

  ```
  WHILE P
  DO S
  INVARIANT I
  VARIANT V
  END
  ```

  Ex. : $W(i<m,i:=i+1,i \in 1..m,m-i)$

- *variable* $:\in$ *expression* choix dans un ensemble

  $x :\in E \equiv$

  ```
  ANY x' WHERE x' ∈ E
  THEN x := x'
  END
  ```

Substitution généralisée

- *variable* : *prédicat* choix selon un prédicat
  x : P ≡
  ```
  ANY x' WHERE [x$0,x:=x,x'] P
  THEN x := x'
  END
  ```
  (où x$0 vaut x avant la substitution et x vaut x' après la substitution)
- f(x) := E ≡ f := f ⩤ {x ↦ E} surcharge d'une fonction
  Ex. : soit f={1 ↦ 2, 2 ↦ 3}. f(2):=0 donne f={1 ↦ 2, 2 ↦ 0}

Substitution généralisée

- $wp([x:=E]\ R) \equiv R(E/x)$ E renomme les x libres dans R
- $wp([x_1,x_2:=E_1,E_2]\ R) \equiv [x_1:=E_1]\ [x_2:=E_2]\ R$
- $wp([\text{skip}]\ R) \equiv R$
- $wp([P \mid S]\ R) \equiv P \wedge [S]\ R$
- $wp([P \Rightarrow S]\ R) \equiv P \Rightarrow [S]\ R$
- $wp([S1\ []\ S2]\ R) \equiv [S1]\ R \wedge [S2]\ R$
- $wp([@\ x \bullet S]\ R) \equiv \forall\ x \bullet [S]\ R$
- $wp([@\ x \bullet (P \Rightarrow S)]\ R) \equiv \forall\ x \bullet (P \Rightarrow [S]\ R)$
- $wp([S1\ ;\ S2]\ R) \equiv [S1]\ ([S2]\ R)$
- $wp([W(P,S,I,V)]\ R) \equiv$
  $I \wedge \forall\ x \bullet ((I \wedge P) \Rightarrow [S]\ I) \wedge \forall\ x \bullet (I \Rightarrow V \in \mathbb{N}) \wedge \forall\ x \bullet$
  $((I \wedge P) \Rightarrow [n:=V]\ [S]\ (V<n)) \wedge \forall\ x \bullet ((I \wedge \neg\ P) \Rightarrow R)$

Substitution généralisée

$S1=S2 \equiv \forall\ P \bullet [S1]\ P \Leftrightarrow [S2]\ P$

- $(S1\ ;\ S2)\ ;\ S3 = S1\ ;\ (S2\ ;\ S3)$
- $skip\ ;\ S = S\ ;\ skip = S$
- $(P \mid S1)\ ;\ S2 = P \mid (S1\ ;\ S2)$
- $S1\ ;\ (P \mid S2) = [S1]\ P \mid (S1\ ;\ S2)$
- $(P \Rightarrow S1)\ ;\ S2 = P \Rightarrow (S1\ ;\ S2)$
- $(S1\ []\ S2)\ ;\ S3 = (S1\ ;\ S3)\ []\ (S2\ ;\ S3)$
- $S1\ ;\ (S2\ []\ S3) = (S1\ ;\ S2)\ []\ (S1\ ;\ S3)$
- $(x:=E)\ ;\ (P \Rightarrow S) = [x:=E]\ P \Rightarrow (x:=E\ ;\ S)$

Substitution généralisée

- $(@\ x \bullet S1)\ ;\ S2 = @\ x \bullet (S1\ ;\ S2)$
  x lié dans S2
- $S1\ ;\ (@\ x \bullet S2) = @\ x \bullet (S1\ ;\ S2)$
  x lié dans S1
- $S1 \parallel S2 = S2 \parallel S1$
- $x_1 := E_1 \parallel x_2 := E_2 = x_1, x_2 := E_1, E_2$
  $x_1 \neq x_2$
- $skip \parallel S = S$
- $(P \mid S1) \parallel S2 = P \mid (S1 \parallel S2)$
- $(P \Rightarrow S1) \parallel S2 = trm(S2) \mid P \Rightarrow (S1 \parallel S2)$
- $(S1\ []\ S2) \parallel S3 = (S1 \parallel S3)\ []\ (S2 \parallel S3)$
- $(@\ x \bullet S1) \parallel S2 = @\ x \bullet (S1 \parallel S2)$
  x lié dans S2

Substitution généralisée

- <u>Propriété</u> (sur le parallélisme) :
  $[S1]\ P1\ ,\ [S2]\ P2 \quad \vdash \quad [S1 \| S2]\ (P1 \wedge P2)$

- $\mathtt{prd}_{variable}(substitution)$ prédicat avant-après (ou prédicat relationnel d'une substitution)
  $\mathtt{prd}_x(S) \equiv \neg\ [S]\ (x \neq x')$
  x' est l'une des valeurs possibles de x après la substitution

Substitution généralisée

- `trm`(*substitution*) terminaison
  `trm`(S) ≡ [S] TRUE
  Ex. : substitution pré-conditionnée
  `PRE` *prédicat* `THEN` *substitution* `END`
  Pour ¬ *prédicat*, le résultat (état post) est imprédictible (i.e. il ne vérifie aucun prédicat i.e. l'opération ne se termine pas)

- `abt`(*substitution*) non-terminaison ou crash (*abort*)
  `abt`(S) ≡ ¬ `trm`(S)

Substitution généralisée

- $\texttt{trm}(x{:=}E) \Leftrightarrow \text{TRUE}$
- $\texttt{trm}(\texttt{skip}) \Leftrightarrow \text{TRUE}$
- $\texttt{trm}(P \mid S) \Leftrightarrow P \wedge \texttt{trm}(S)$
- $\texttt{trm}(P \Rightarrow S) \Leftrightarrow P \Rightarrow \texttt{trm}(S)$
- $\texttt{trm}(S1 \,[]\, S2) \Leftrightarrow \texttt{trm}(S1) \wedge \texttt{trm}(S2)$
- $\texttt{trm}(@\, x \bullet S) \Leftrightarrow \forall\, x \bullet \texttt{trm}(S)$
- $\texttt{trm}(x :\in E) \Leftrightarrow \text{TRUE}$
- $\texttt{trm}(x : P) \Leftrightarrow \text{TRUE}$
- $\texttt{trm}(S1 ; S2) \Leftrightarrow \texttt{trm}(S1) \wedge \forall\, x' \bullet (\texttt{prd}_x(S1) \Rightarrow [x{:=}x']\, \texttt{trm}(S2))$
- $\texttt{trm}(W(P,S,I,V)) \Leftrightarrow$

  $I \wedge$
  $\forall\, x \bullet ((I \wedge P) \Rightarrow [S]\, I) \wedge$
  $\forall\, x \bullet (I \Rightarrow V \in \mathbb{N}) \wedge$
  $\forall\, x \bullet ((I \wedge P) \Rightarrow [n{:=}V]\, [S]\, (V<n))$

Substitution généralisée

- `fis`(*substitution*) faisabilité (*feasibility*)
  `fis`(S) ≡ ¬ [S] FALSE
  Ex. : substitution gardée
  `SELECT` *prédicat* `THEN` *substitution* `END`
  Pour ¬ *prédicat*, le résultat (état post) est vide
  Propriété : `fis`(S) ⇔ ∃ x' • (`prd`$_x$(S))

- `mir`(*substitution*) miracle
  `mir`(S) ≡ ¬ `fis`(S)

Substitution généralisée

- $\texttt{fis}(x:=E) \Leftrightarrow \text{TRUE}$
- $\texttt{fis}(\texttt{skip}) \Leftrightarrow \text{TRUE}$
- $\texttt{fis}(P \mid S) \Leftrightarrow P \Rightarrow \texttt{fis}(S)$
- $\texttt{fis}(P \Rightarrow S) \Leftrightarrow P \wedge \texttt{fis}(S)$
- $\texttt{fis}(S1 \,[]\, S2) \Leftrightarrow \texttt{fis}(S1) \vee \texttt{fis}(S2)$
- $\texttt{fis}(@\, x \bullet S) \Leftrightarrow \exists\, x \bullet \texttt{fis}(S)$
- $\texttt{fis}(x :\in E) \Leftrightarrow E \neq \varnothing$
- $\texttt{fis}(x : P) \Leftrightarrow \exists\, x' \bullet [x\$0, x:=x, x']\, P$
- $\texttt{fis}(S1 ; S2) \Leftrightarrow$
  $\texttt{trm}(S1) \Rightarrow \exists\, x' \bullet (\texttt{prd}_x(S1) \wedge [x:=x']\, \texttt{fis}(S2))$
- $\texttt{fis}(W(P,S,I,V)) \Leftrightarrow$
  $\texttt{trm}(W(P,S,I,V)) \Rightarrow \exists\, x' \bullet ([x\$0, x:=x, x']\, (I \wedge \neg\, P))$

Substitution généralisée

- S = P | @ x' • (Q ⇒ x:=x') avec x lié dans P
  forme normale (toute substitution généralisée peut se mettre sous la forme normalisée)
  S ≡
  ```
  PRE P
  THEN ANY x' WHERE Q
       THEN x := x'
       END
  END
  ```
  S = trm(S) | @ x' • ($\texttt{prd}_x$(S) ⇒ x:=x')

  wp([S] R) ⇔ P ∧ ∀ x' • (Q ⇒ [x:=x'] R)

  trm(S) ⇔ P

  fis(S) ⇔ P ⇒ ∃ x' • Q

  $\texttt{prd}_x$(S) ⇔ P ⇒ Q

Substitution généralisée

- $[S]\,(P1 \wedge P2) \Leftrightarrow [S]\,P1 \wedge [S]\,P2$
  distributivité
- $\forall\, x \bullet (P1 \Rightarrow P2) \Rightarrow ([S]\,P1 \Rightarrow [S]\,P2)$
  monotonie
- $\mathtt{prd}_x(S) \vee \mathtt{trm}(S)$
  totalité
- $[S]\,R \Rightarrow \mathtt{trm}(S)$
  terminaison

Substitution généralisée

- Interprétation ensembliste : tout ce qui porte sur les prédicats peut être transposé sur les ensembles
  On fait correspondre au prédicat P son support $\underline{P} = \{x \mid x \in STATE \wedge P\}$ où STATE représente l'ensemble des états
- Soit $x \in E \wedge \mathtt{trm}(S) \Rightarrow [S]\,(x \in E)$
- $\mathtt{pre}(S) \equiv \{x \mid x \in E \wedge \mathtt{trm}(S)\}$
  ensemble des éléments pour lesquels S termine
- $\mathtt{rel}(S) \equiv \{x,x' \mid x,x' \in E \times E \wedge \mathtt{prd}_x(S)\}$ relation reliant les valeurs avant avec les valeurs après de S
- $\mathtt{dom}(S) \equiv \{x \mid x \in E \wedge \mathtt{fis}(S)\}$
  domaine de $\mathtt{rel}(S)$
- $S = x \in \mathtt{pre}(S) \mid @\,x' \bullet ((x,x') \in \mathtt{rel}(S) \Rightarrow x := x')$
  forme normale

Machine abstraite

Machine abstraite

MACHINE

Partie entête : *nom de la machine avec ses paramètres, contraintes sur les paramètres* (CONSTRAINTS)

Partie statique : *déclaration d'ensembles* (SETS), *déclaration de constantes* (CONCRETE_CONSTANTS, ABSTRACT_CONSTANTS ou CONSTANTS), *propriétés des constantes* (PROPERTIES), *définitions* (DEFINITIONS), *variables i.e. état* (CONCRETE_VARIABLES, ABSTRACT_VARIABLES ou VARIABLES), *invariant i.e. caractérisation de l'état* (INVARIANT), *assertions supplémentaires* (ASSERTIONS)

Partie dynamique : *initialisation de l'état* (INITIALISATION), *opérations* (OPERATIONS)

END

Machine abstraite

- `CONSTRAINTS :` *contraintes*
  prédicat qui spécifie les valeurs des paramètres
- `PROPERTIES :` *propriétés des constantes*
  prédicat qui spécifie les constantes par des axiomes
- `INVARIANT :` *invariant*
  prédicat qui donne le typage des variables et les contraintes qu'elles doivent satisfaire
- `ASSERTIONS :` *assertions*
  prédicats qui sont des conséquences logiques des autres axiomes et de l'invariant

Machine abstraite

- `INITIALISATION :` *initialisation*
  substitution généralisée
- `OPERATIONS :` *opérations*
  liste de la forme
  variable ← *opération(paramètres)* =
  `PRE` *prédicat pré-condition*
  `THEN` *substitution généralisée*
  `END`

Machine abstraite

- Écriture des obligations de preuve
  - Pour chaque assertion :
    © ∧ *invariant* ∧ *(k-1) premières assertions* ⇒ *k^e assertion*
  - © ⇒ [*initialisation*] *invariant*
  - Pour chaque opération :
    © ∧ *invariant* ∧ *assertions* ∧ *pré-condition* ⇒ [*substitution généralisée*] *invariant*

avec © ≡ *contraintes* ∧ *paramètre* (contraintes éventuelles sur leurs domaines) ∧ *propriétés* ∧ *ensemble* (contraintes éventuelles sur leurs domaines)

Machine abstraite

- Trois étapes pour résoudre une obligation de preuve
  - Écriture de l'obligation de preuve (pour une assertion ou l'initialisation ou une opération)
  - Substitution (de l'initialisation pour l'initialisation ou de la substitution généralisée pour une opération) dans l'invariant
  - Preuve (les prémisses doivent impliquer la conséquence)

Exemple de machine abstraite :

réservation

MACHINE
RÉSERVATION /* réservation de places */
CONSTANTS
nb_max, *SIÈGES*
PROPERTIES
$nb\_max \in 1..\texttt{maxint} \wedge SIÈGES = 1..nb\_max$
VARIABLES
occupés, *nb_libre*
INVARIANT
$occupés \subseteq SIÈGES \wedge nb\_libre \in 0..nb\_max \wedge$
$nb\_libre = nb\_max - \texttt{card}(occupés)$
/* Ex. : *nb_max*=8, *occupés*={2,3,7}⊆1..8, *nb_libre*=5*/

Exemple de machine abstraite :

réservation

```
INITIALISATION
   occupés,nb_libre := ∅,nb_max
OPERATIONS
   /* réserver une place quelconque */
   place ← réserver ≙
        PRE  nb_libre≠0
        THEN ANY  p  WHERE  p ∈ SIÈGES-occupés
                THEN  place,occupés,nb_libre :=
                        p,occupés∪{p},nb_libre-1
                END
        END
   ;
```

Exemple de machine abstraite :

réservation

```
/* libérer une place précise */
r ← libérer(place) ≙
    PRE  place ∈ SIÈGES
    THEN IF  place ∈ occupés
         THEN  r,occupés,nb_libre :=
                    TRUE,
                    occupés-{place},
                    nb_libre+1
         ELSE  r := FALSE
         END
    END
END
```

Exemple de machine abstraite :

réservation (preuve de l'initialisation)

L'obligation de la preuve de l'initialisation

$$nb\_max \in 1..\mathtt{maxint} \wedge SIÈGES = 1..nb\_max \Rightarrow [\ occupés,nb\_libre := \varnothing,nb\_max\ ]\ occupés \subseteq SIÈGES \wedge nb\_libre \in 0..nb\_max \wedge nb\_libre = nb\_max - \mathtt{card}(occupés)$$

Exemple de machine abstraite :

réservation (preuve de l'initialisation)

Se réécrit en (après substitution)

$$nb\_max \in 1..\texttt{maxint} \wedge SIÈGES = 1..nb\_max \Rightarrow \varnothing \subseteq SIÈGES \wedge nb\_max \in 0..nb\_max \wedge nb\_max = nb\_max - \texttt{card}(\varnothing)$$

Exemple de machine abstraite :

réservation (preuve de l'initialisation)

Est prouvée car

- $\varnothing \subseteq SIÈGES$ puisque *SIÈGES* est un ensemble et contient donc $\varnothing$
- $nb\_max \in 0..nb\_max$ puisque *nb_max* est dans les bornes
- $nb\_max = nb\_max - \texttt{card}(\varnothing)$ puisque $\texttt{card}(\varnothing) = 0$

Exemple de machine abstraite :

réservation (preuve de *réserver*)

L'obligation de la preuve de l'opération *réserver*

$$nb\_max \in 1..\texttt{maxint} \wedge SIÈGES = 1..nb\_max \wedge occupés \subseteq SIÈGES \wedge nb\_libre \in 0..nb\_max \wedge nb\_libre = nb\_max - \texttt{card}(occupés) \wedge nb\_libre \neq 0 \Rightarrow [\ \texttt{ANY}\ p\ \texttt{WHERE}\ p \in SIÈGES - occupés\ \texttt{THEN}\ place,occupés,nb\_libre := p,occupés \cup \{p\},nb\_libre-1\ ]\ occupés \subseteq SIÈGES \wedge nb\_libre \in 0..nb\_max \wedge nb\_libre = nb\_max - \texttt{card}(occupés)$$

Exemple de machine abstraite :

réservation (preuve de *réserver*)

Se réécrit en (après substitution)

$$nb\_max \in 1..\texttt{maxint} \wedge SIÈGES = 1..nb\_max \wedge$$
$$occupés \subseteq SIÈGES \wedge nb\_libre \in 0..nb\_max \wedge$$
$$nb\_libre = nb\_max - \texttt{card}(occupés) \wedge$$
$$nb\_libre \neq 0 \wedge p \in SIÈGES\text{-}occupés \Rightarrow$$
$$occupés \cup \{p\} \subseteq SIÈGES \wedge nb\_libre\text{-}1 \in$$
$$0..nb\_max \wedge nb\_libre\text{-}1 = nb\_max -$$
$$\texttt{card}(occupés \cup \{p\})$$

Exemple de machine abstraite :

réservation (preuve de *réserver*)

Est prouvée car

- $occupés \subseteq SIÈGES \wedge p \in SIÈGES\text{-}occupés \Rightarrow occupés \cup \{p\} \subseteq SIÈGES$
- $nb\_libre \in 0..nb\_max \wedge nb\_libre \neq 0 \Rightarrow nb\_libre - 1 \in 0..nb\_max$
- $occupés \subseteq SIÈGES \wedge p \in SIÈGES\text{-}occupés \Rightarrow card(occupés \cup \{p\}) = card(occupés) + card(\{p\}) = card(occupés) + 1$

  $nb\_libre = nb\_max - card(occupés) \wedge card(occupés \cup \{p\}) = card(occupés) + 1 \Rightarrow nb\_libre - 1 = nb\_max - card(occupés \cup \{p\})$

Exemple de machine abstraite :

réservation (preuve de *libérer*)

Exercice : faites la preuve de l'opération *libérer*.

Preuve de machine abstraite :

addition (preuve de l'initialisation)

L'obligation de preuve

$$\Rightarrow [x := 42]\ x \in \mathbb{N}$$

se réécrit en (après substitution)

$$\Rightarrow 42 \in \mathbb{N}$$

est prouvée car

$$42 \in \mathbb{N}$$

Preuve de machine abstraite :

addition (preuve de l'opération *ajouter_chiffre*)

L'obligation de preuve

$$x \in \mathbb{N} \land delta \in 0..9 \Rightarrow [x := x + delta]\, x \in \mathbb{N}$$

se réécrit en (après substitution)

$$x \in \mathbb{N} \land delta \in 0..9 \Rightarrow x + delta \in \mathbb{N}$$

est prouvée car

$$x \in \mathbb{N} \land delta \in 0..9 \subseteq \mathbb{N} \Rightarrow x + delta \in \mathbb{N}$$

Exemple de machine abstraite : division

```
MACHINE
   DIVISION   /* division entière */
OPERATIONS
   /* a=dividende, b=diviseur, q=quotient, r=reste */
   /* Ex. : diviser(16,5) retourne 3,1 */
   q,r ← diviser(a,b) ≙
        PRE  a ∈ ℕ ∧ b ∈ ℕ1
        THEN
              ANY  s,t  /* s et t sont en fait uniques */
              WHERE  s ∈ ℕ ∧ t ∈ ℕ ∧ a = b * s + t ∧ t < b
              THEN  q,r := s,t
              END
   END
END
```

Exemple de machine abstraite :

tri

MACHINE

TRI /* tri */

CONCRETE_CONSTANTS

n

PROPERTIES

$n \in \mathbb{N}_1$ /* n est un entier (strictement) positif */

VARIABLES

T

INVARIANT

$T \in 1..n \rightarrow \mathbb{N}$ /* T est une fonction totale */

INITIALISATION

$T :\in 1..n \rightarrow \mathbb{N}$

Exemple de machine abstraite :

tri

```
OPERATIONS
   v ← valeur_T(i) ≙
        PRE  i ∈ 1..n
        THEN  v := T(i) /* T est accédé comme un tableau */
        END
   ;
   affecter_T(i,v) ≙
        PRE  i ∈ 1..n ∧ v ∈ ℕ
        THEN  T(i)  := v /* T est affecté comme un tableau */
        END
   ;
```

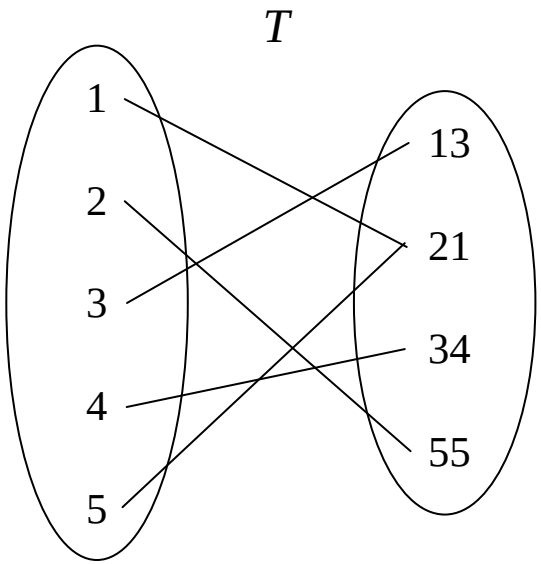
Exemple de machine abstraite :

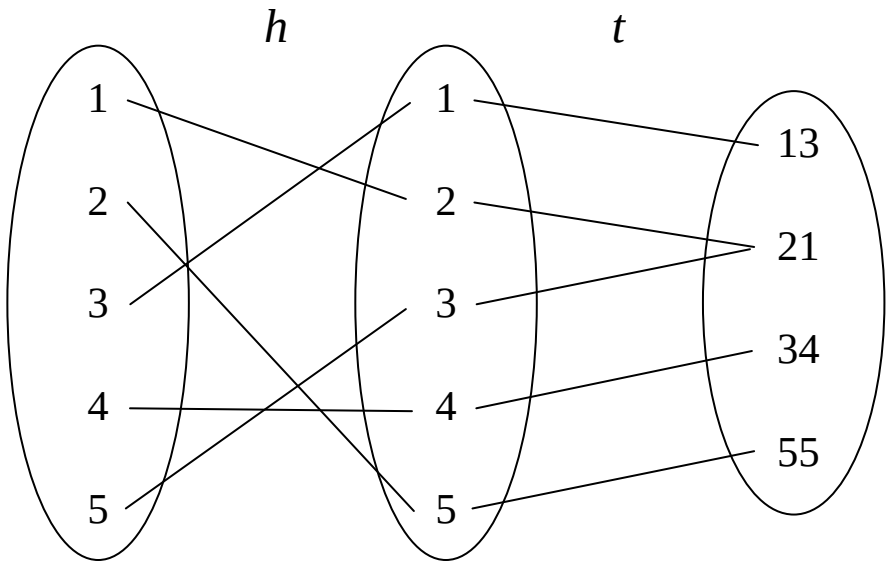
tri

trier ≙
ANY *t*,*h* /* *(t,h)* est unique ssi *T* est une injection */
WHERE *t* ∈ 1..*n* → ℕ ∧ /* *t* fonction totale */
$(\forall(i_1,i_2) \bullet (i_1 \in 1..n \wedge i_2 \in 1..n \wedge i_1<i_2$
$\Rightarrow t(i_1) \leq t(i_2))) \wedge$
/* l'ordre partiel des images de *t*
est compatible avec
l'ordre total des indices de *t* */
h ∈ 1..*n* >->> 1..*n* ∧
/* *h* est une bijection totale sur les
éléments de 1 à *n* */
(*h*;*t*)=*T* /* la composition doit rendre *T* */
THEN *T* := *t*
END
END

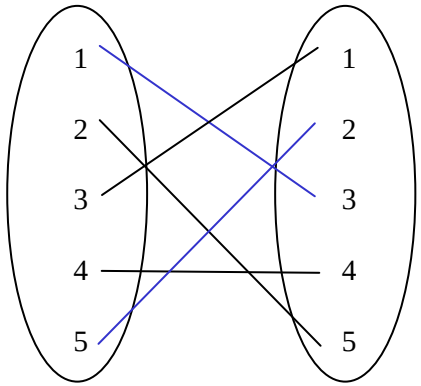
Exemple de machine abstraite :

tri (opération *trier*)





Autre solution pour *h* :



Exemple de machine abstraite :

interrupteur

```
MACHINE
   INTERRUPTEUR   /* interrupteur */
SETS
   ÉTATS = {allumé,éteint}
VARIABLES
   é
INVARIANT
   é ∈ ÉTATS
INITIALISATION
   é := éteint
OPERATIONS
   changer ≙
   IF  é = éteint  THEN  é := allumé  ELSE  é := éteint  END
END
```

Exemple de machine abstraite :

un livre est prêté à 0, 1 ou plusieurs lecteurs (fonction et injection totale)

MACHINE

PRÊTS_LIVRES_À_LECTEURS_FonctionInjectionTotale

SETS

LIVRES ; LECTEURS

VARIABLES

prêts, *lecteurs*

INVARIANT

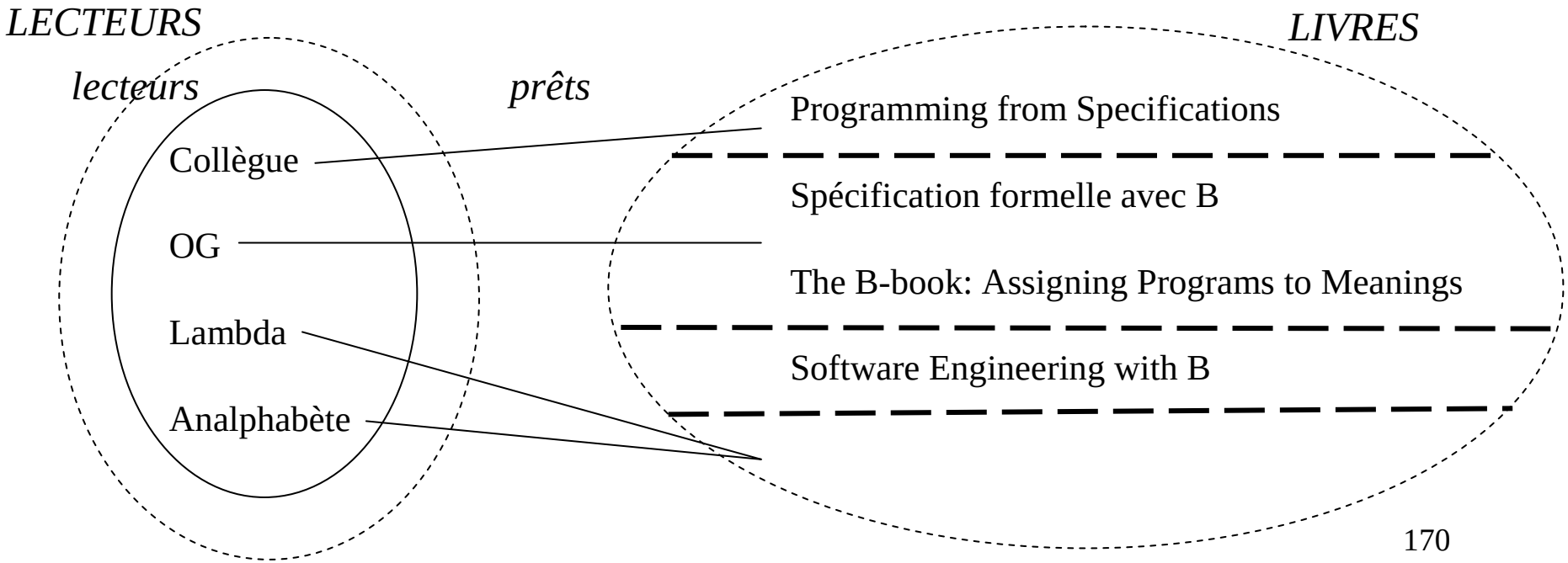
$lecteurs \subseteq LECTEURS \wedge$

$prêts \in lecteurs \rightarrow \mathbb{P}(LIVRES) \wedge$ /* fonction totale */

$(prêts \rhd \emptyset) \in lecteurs >\!\!\rightarrow \mathbb{P}(LIVRES)$ /* *prêts*, sans les lecteurs qui n'ont emprunté aucun livre, est une injection totale entre les *lecteurs* et les *parties des LIVRES* */

Exemple de machine abstraite :
un livre est prêté à 0, 1 ou plusieurs lecteurs (fonction et injection totale)

Ex. : *prêts* = { Collègue ↦ { Programming from Specifications } , OG ↦ { Spécification formelle avec B , The B-book: Assigning Programs to Meanings } , Lambda ↦ ∅ , Analphabète ↦ ∅ }



Exemple de machine abstraite :
un livre est prêté à 0, 1 ou plusieurs lecteurs (relation et fonction partielle)

MACHINE

PRÊTS_LIVRES_À_LECTEURS_RelationFonct°Partielle

SETS

LIVRES ; LECTEURS

VARIABLES

prêts, lecteurs

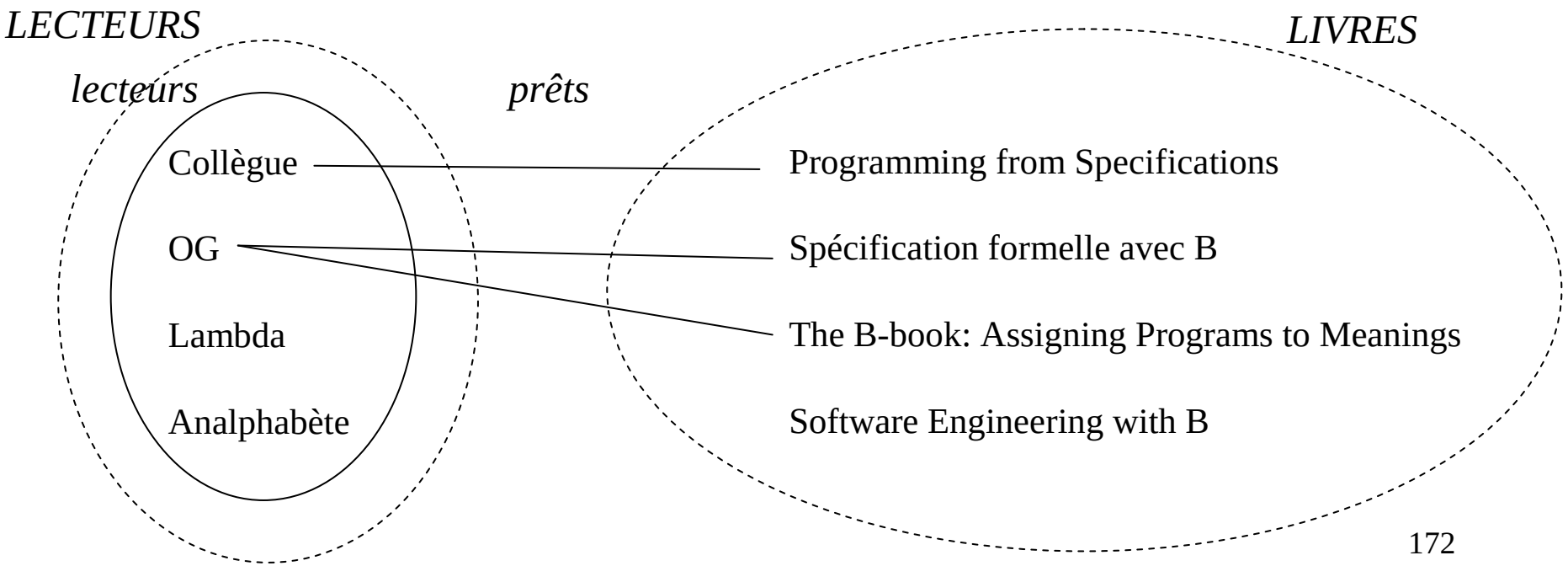
INVARIANT

$lecteurs \subseteq LECTEURS \wedge \quad prêts \in lecteurs \leftrightarrow LIVRES \wedge$

$prêts^{-1} \in lecteurs \; +\!\!> LIVRES$ /* la relation inverse de *prêts* est une fonction partielle entre les *lecteurs* et les *LIVRES* */

Exemple de machine abstraite :
un livre est prêté à 0, 1 ou plusieurs lecteurs (relation et fonction partielle)

Ex. : *prêts* = { Collègue ↦ Programming from Specifications , OG ↦ Spécification formelle avec B , OG ↦ The B-book: Assigning Programs to Meanings }



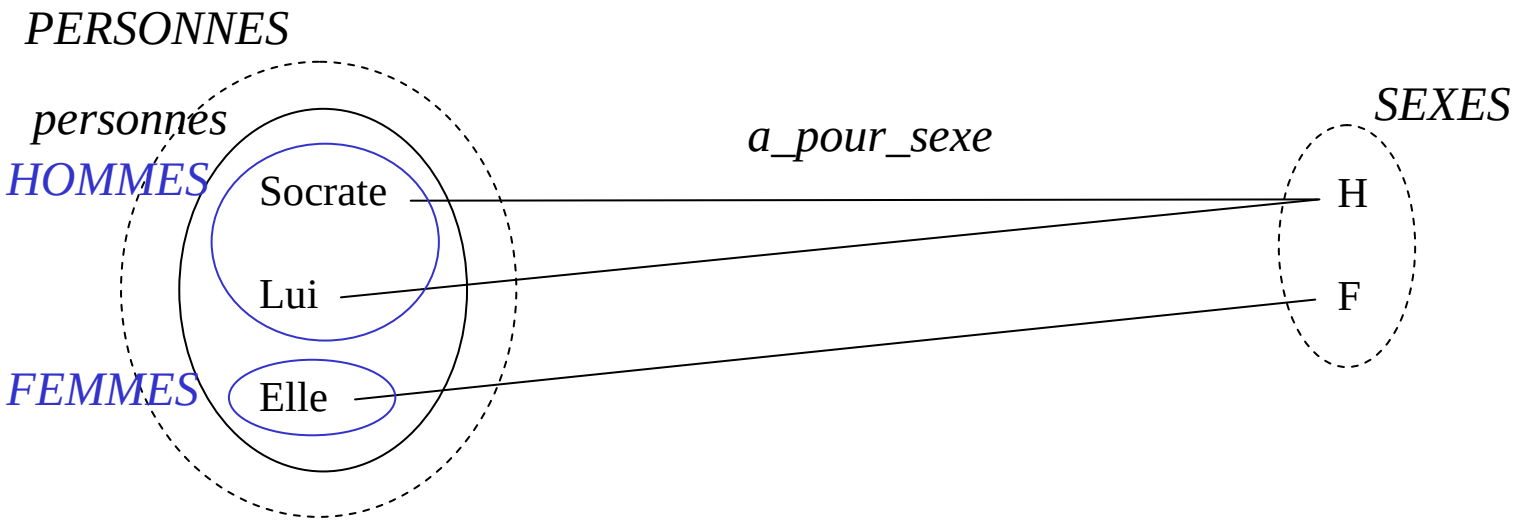
Exemple de machine abstraite :

une personne est soit un homme, soit une femme (fonction totale)

MACHINE
PERSONNES_FonctionTotale
SETS
PERSONNES ; *SEXES* = {H,F}
VARIABLES
personnes, *a_pour_sexe*
DEFINITIONS /* i.e. informations calculables */
HOMMES ≜ $a\_pour\_sexe^{-1}[\{H\}]$;
FEMMES ≜ $a\_pour\_sexe^{-1}[\{F\}]$
INVARIANT
$personnes \subseteq PERSONNES \wedge$
$a\_pour\_sexe \in personnes \rightarrow SEXES$ /* fonction totale */

Exemple de machine abstraite :

une personne est soit un homme, soit une femme (fonction totale)



Exemple de machine abstraite :

une personne est soit un homme, soit une femme (3 ensembles)

MACHINE

PERSONNES_3ensembles

SETS

PERSONNES

VARIABLES

personnes, *hommes*, *femmes*

INVARIANT

$personnes \subseteq PERSONNES \wedge$

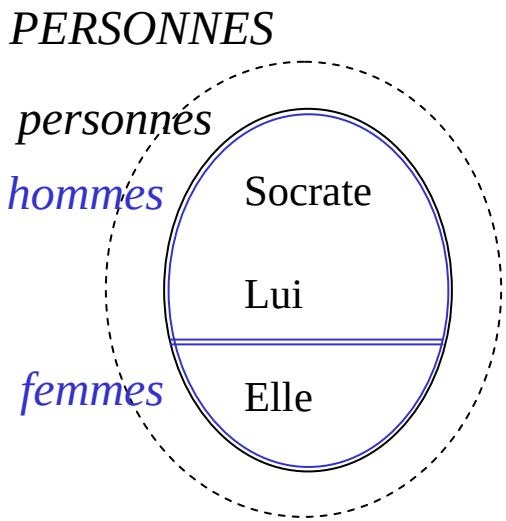
$hommes \subseteq PERSONNES \wedge femmes \subseteq PERSONNES \wedge$

$hommes \cap femmes = \varnothing \wedge$

$hommes \cup femmes = personnes$

Exemple de machine abstraite :

une personne est soit un homme, soit une femme (3 ensembles)



Exemple de machine abstraite :

une personne est soit un homme, soit une femme (3 ensembles et 2 fonctions)

MACHINE

PERSONNES_3ensembles_2fonctions

SETS

PERSONNES ; HOMMES ; FEMMES

VARIABLES

personnes, hommes, femmes, fctH, fctF

INVARIANT

$personnes \subseteq PERSONNES \wedge$

$hommes \subseteq HOMMES \wedge \ femmes \subseteq FEMMES \wedge$

$fctH \in personnes >\!\!+\!\!\gg hommes \wedge$

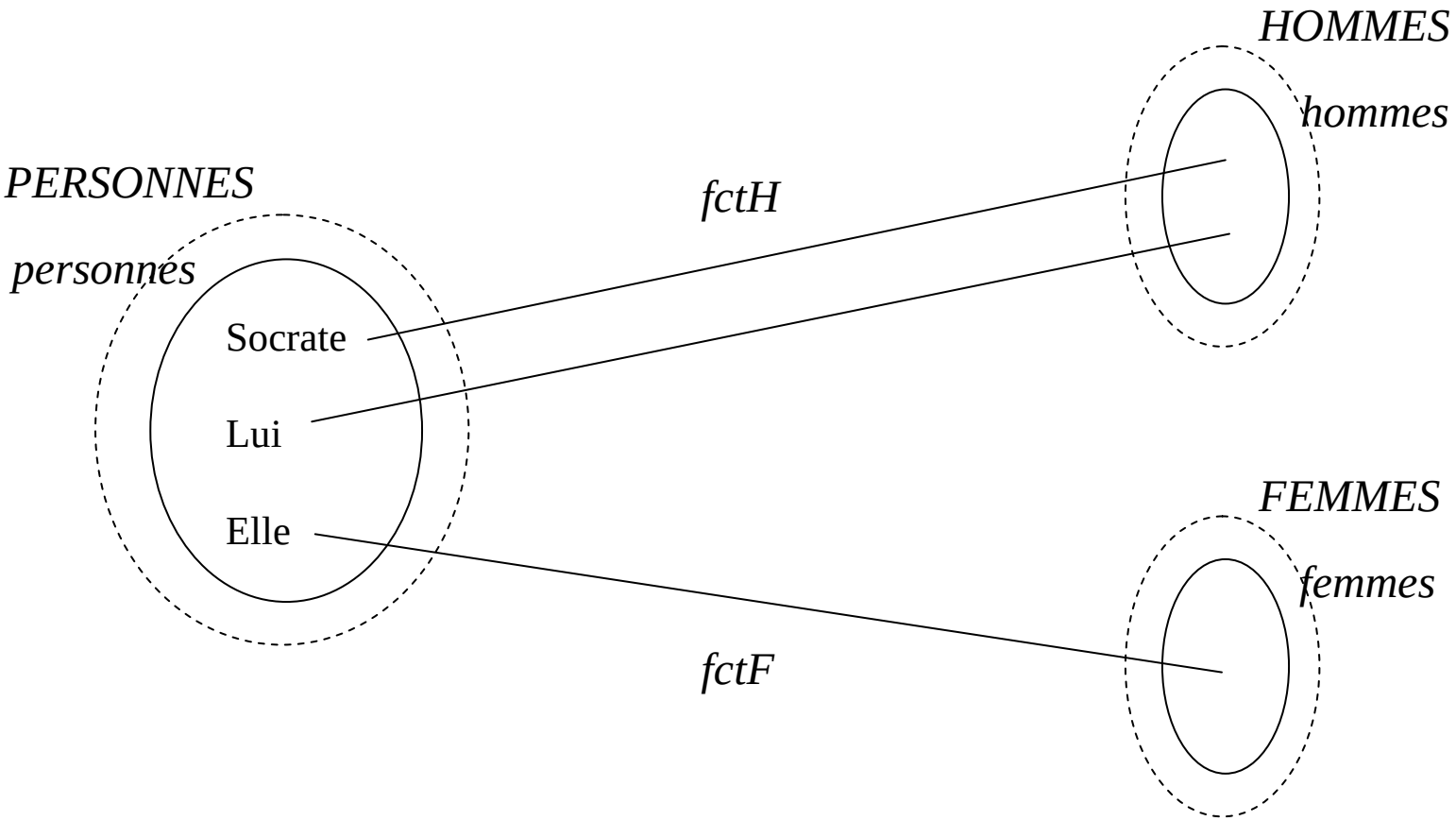
$fctF \in personnes >\!\!+\!\!\gg femmes \wedge$

$\mathtt{dom}(fctH) \cap \mathtt{dom}(fctF) = \varnothing \wedge$

$\mathtt{dom}(fctH) \cup \mathtt{dom}(fctF) = personnes$

Exemple de machine abstraite :

une personne est soit un homme, soit une femme (3 ensembles et 2 fonctions)



Exemple de machine abstraite :
tout étudiant doit émarger pour un examen (avec inclusion d'ensembles)

MACHINE

ÉMARGER_EXAMEN_AvecInclusionEnsembles

SETS

ÉTUDIANTS ; EXAMENS

VARIABLES

étudiants, examens, a_émargé, a_une_note

INVARIANT

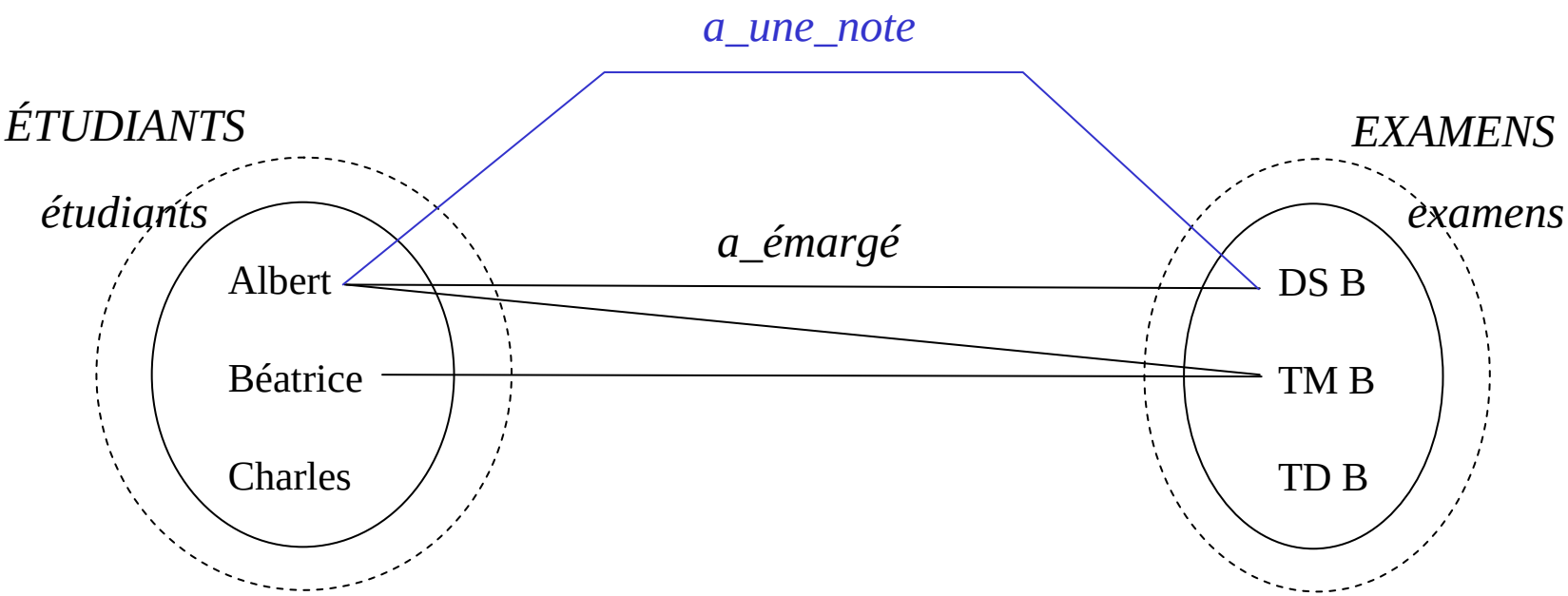
$étudiants \subseteq ÉTUDIANTS \land examens \subseteq EXAMENS \land$

$a\_une\_note \in étudiants \leftrightarrow examens \land$

$a\_émargé \in étudiants \leftrightarrow examens \land$

$a\_une\_note \subseteq a\_émargé$

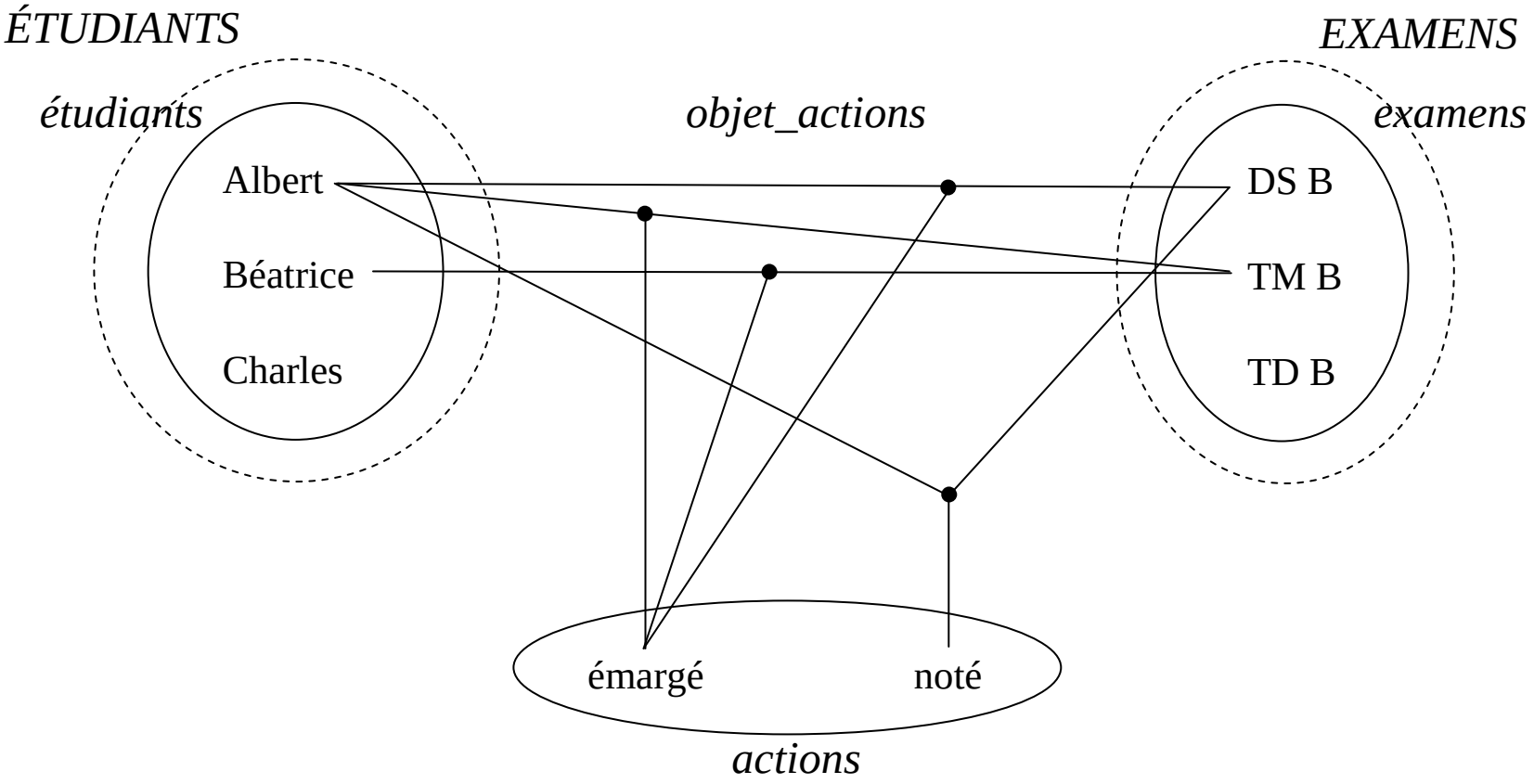
Exemple de machine abstraite :
tout étudiant doit émarger pour un examen (avec inclusion d'ensembles)



Exemple de machine abstraite :
tout étudiant doit émarger pour un examen (sans inclusion d'ensembles)

MACHINE
ÉMARGER_EXAMEN_SansInclusionEnsembles
SETS
ÉTUDIANTS ; EXAMENS
VARIABLES
étudiants, *examens*, *actions*, *objet_actions*
INVARIANT
$étudiants \subseteq ÉTUDIANTS \wedge examens \subseteq EXAMENS \wedge$
$actions = \{émargé,noté\} \wedge$
$objet\_actions \in (étudiants \times examens) \leftrightarrow actions \wedge$
$(\forall (é,e) \bullet (é,e) \in \mathtt{dom}(objet\_actions) \wedge$
$\mathtt{card}(objet\_actions [\{(é,e)\}])=1)$
$\Rightarrow objet\_actions[\{(é,e)\}] = \{émargé\})$

Exemple de machine abstraite :
tout étudiant doit émarger pour un examen (sans inclusion d'ensembles)



Exemple de machine abstraite :
vie privée des personnes

MACHINE

PERSONNES

SETS

PERSONNES ; ÉTATS = {célibataire,marié,décédé}

CONSTANTS

transitions = {(célibataire, mariage) ↦ marié, (marié, divorce) ↦ célibataire , (célibataire, décès) ↦ décédé, (marié, décès) ↦ décédé}

VARIABLES

personnes, *histoires*

Exemple de machine abstraite :

vie privée des personnes

INVARIANT

$personnes \subseteq PERSONNES$

$\wedge\ histoires \in personnes \rightarrow \mathtt{seq}(ÉTATS)$

/* Ex. : *histoires* = { Instable ↦ [célibataire , marié , célibataire , marié , célibataire] , NouveauNé ↦ [célibataire] */

INITIALISATION

$personnes, histoires := \emptyset, \emptyset$

OPERATIONS

ajouter_personne ≙

PRE $PERSONNES - personnes \neq \emptyset$ THEN

ANY *pers* WHERE $pers \in (PERSONNES - personnes)$

THEN $personnes , histoires(pers) :=$

$personnes \cup \{pers\} , histoires(pers) <- célibataire$

END

END ;

Exemple de machine abstraite :

vie privée des personnes

/* Ex. : soit (λ ↦ [célibataire , marié]) ∈ *histoires* ; alors, *transiter(*λ,divorce*)* donne λ ↦ [célibataire , marié , célibataire] car (marié,divorce) ↦ célibataire ∈ *transitions* */

transiter(*pers*,*événement*) ≙
PRE *pers* ∈ *personnes* ∧
(last(*histoires*(*pers*)),*événement*) ∈ dom(*transitions*)
THEN *histoires*(*pers*) := *histoires*(*pers*)<-
transitions(last(*histoires*(*pers*)),*événement*)
END ;

/* Ex. : soit (λ ↦ [célibataire , marié]) ∈ *histoires* ; alors, *état_courant(*λ*)* retourne marié */

état_courant ← *afficher_état_courant*(*pers*) ≙
PRE *pers* ∈ *personnes*
THEN *état_courant* := last(*histoires*(*pers*))
END
END

Exemple de machine abstraite :
fichier

MACHINE

FICHIER(max_enreg,CHAMPS,VALEURS)

CONSTRAINTS

$max\_enreg \in \mathbb{N}_1$

VARIABLES

nb_enreg, *fichier*

DEFINITIONS

$ENREGS \triangleq 1..max\_enreg$

INVARIANT

$fichier \in CHAMPS \rightarrow (ENREGS \rightarrow VALEURS)$

INITIALISATION

$nb\_enreg, fichier := 0, CHAMPS \times \varnothing$

Exemple de machine abstraite :

fichier

OPERATIONS

affecter(*champ*,*enreg*,*val*) ≙

PRE *champ* ∈ *CHAMPS* ∧ *enreg* ∈ *ENREGS*

∧ *val* ∈ *VALEURS*

THEN *fichier*(*champ*)(*enreg*) := *val*

END

;

v ← *rechercher*(*champ*,*enreg*) ≙

PRE *champ* ∈ *CHAMPS* ∧ *enreg* ∈ *ENREGS*

THEN *v* := *fichier*(*champ*)(*enreg*)

END

END

Exemple de machine abstraite :

exemple « jouet »

MACHINE

EXEMPLE_JOUET

SETS

ÉTUDIANTS ; DIPLÔMES ; VOITURES

VARIABLES

étudiants, *diplômes*, *voitures*, *avoir_obtenu*, *être_possédée_par*

INVARIANT

$étudiants \subseteq ÉTUDIANTS \land diplômes \subseteq DIPLÔMES \land voitures \subseteq VOITURES \land avoir\_obtenu \in étudiants \leftrightarrow diplômes \land \mathtt{ran}(avoir\_obtenu) = diplômes \land être\_possédée\_par \in voitures \rightarrow étudiants$

Exemple de machine abstraite :

exemple « jouet »

/* *être_possédée_par* ∈ *voitures* → *étudiants*
Ex. : *voitures* = {3333BX33, 4040NT40 , 4747LA47}, *étudiants* = {Durand, Martin, Leroi}, *être_possédée_par* = {3333BX33 ↦ Durand, 4040NT40 ↦ Durand, 4747LA47 ↦ Martin} */

DEFINITIONS

posséder1 ≙ *être_possédée_par*$^{-1}$ /* = {Durand ↦ 3333BX33, Durand ↦ 4040NT40, Martin ↦ 4747LA47} ∈ *étudiants* ↔ *voitures* */
;
posséder2 ≙ fnc(*être_possédée_par*$^{-1}$) /* = {Durand ↦ {3333BX33,4040NT40}, Martin ↦ {4747LA47}} ∈ *étudiants* +-> ℙ(*voitures*) */ ;

Exemple de machine abstraite :

exemple « jouet »

posséder3 ≙ fnc(*être_possédée_par*$^{-1}$) ∪ ((*étudiants*-ran(*être_possédée_par*)) × {∅}) /* = {Durand ↦ {3333BX33,4040NT40}, Martin ↦ {4747LA47}, Leroi ↦ ∅} ∈ *étudiants* → ℙ(*voitures*) */

OPERATIONS

enregistrer(*étud*,*dipl*) ≙

PRE *étud* ∈ *étudiants* ∧ *dipl* ∈ *diplômes*

THEN *avoir_obtenu* := *avoir_obtenu* ∪ {*étud* ↦ *dipl*}

END

END

Contrainte dynamique

Contrainte dynamique

- Spécification du système (clause `SYSTEM` au lieu de `MACHINE`) bénéficiant d'une clause supplémentaire : `DYNAMICS`
- Ex. : croissance stricte d'une variable (la valeur après doit être supérieure à la valeur avant)

`SYSTEM`	*CROISSANCE*
`VARIABLES`	x
`INVARIANT`	$x \in \mathbb{N}$
`DYNAMICS`	$x < x'$
`END`	

Raffinement (*Refinement*)

Raffinement

- *substitution* ⊑ *substitution* raffinement
  S1 ⊑ S2 se lit « S1 est raffiné par S2 »
  NB : S1 ⊑ S2 ≡ ∀ R • ([S1] R ⇒ [S2] R)
  Remarques sur S1 ⊑ S2 :
  - pre(S1) ⊆ pre(S2)
  - rel(S2) ⊆ rel(S1)

  Ex. : (x:=1 [] x:=2*y) ⊑ x:=1
  Ex. : (x>4 | x:=x+1) ⊑ (x≥0 | x:=x+1)
  Objectifs :
  - diminution du non déterminisme
  - affaiblissement des pré-conditions

Raffinement

- $R \in E2 \leftrightarrow E1 \wedge \mathtt{dom}(R)=E2$ Condition C-raffinement (relation entre les deux ensembles de valeurs E1 et E2 définissant x1 et x2 leurs espaces de variables respectifs)
- *substitution* $\sqsubseteq_{relation}$ *substitution* raffinement avec changement de variables (caractérisé par une relation satisfaisant la condition C-raffinement)
  Remarques sur $S1 \sqsubseteq_R S2$:
  - $R^{-1}[\mathtt{pre}(S1)] \subseteq \mathtt{pre}(S2)$
  - $R^{-1} ; \mathtt{rel}(S2) \subseteq \mathtt{rel}(S1) ; R^{-1}$
- $S1 \sqsubseteq_{R1} S2 \wedge S2 \sqsubseteq_{R2} S3 \Rightarrow S1 \sqsubseteq_{R2;R1} S3$ transitivité

Raffinement

- *substitution* $\sqsubseteq_{prédicat}$ *substitution* raffinement avec changement de variables en terme de substitution généralisée
  NB : Soient R une relation satisfaisant la condition C-raffinement et $R = \{(x2,x1) \mid P\}$. $S1 \sqsubseteq_P S2 \equiv P \wedge \mathtt{trm}(S1) \Rightarrow [S2] \neg [S1] \neg P$

Raffinement (de machine)

REFINEMENT

Partie entête : *nom du raffinement avec ses paramètres (identiques à ceux de la machine raffinée), nom de la machine raffinée* (REFINES)

Partie statique : *déclaration d'ensembles* (SETS), *déclaration de constantes* (CONCRETE_CONSTANTS, ABSTRACT_CONSTANTS ou CONSTANTS), *propriétés des constantes* (PROPERTIES), *définitions* (DEFINITIONS), *variables i.e. état* (CONCRETE_VARIABLES, ABSTRACT_VARIABLES ou VARIABLES), *invariant i.e. caractérisation de l'état* (INVARIANT), *assertions supplémentaires* (ASSERTIONS)

Partie dynamique : *initialisation de l'état* (INITIALISATION), *opérations* (OPERATIONS)

END

Raffinement (de machine)

- Les raffinements sont donc des incréments de spécifications de plus haut niveau.
  Cela assure le passage progressif d'un modèle abstrait à une réalisation concrète
- On peut aussi associer une machine abstraite à un raffinement

- Obligations de preuve (idem que pour les machines abstraites)

Exemple de raffinement (de machine) : réservation

REFINEMENT

RÉSERVATION_Raffinement

REFINES

RÉSERVATION

VARIABLES

états, *nb_libre*

INVARIANT

$états \in SIÈGES \rightarrow BOOL \wedge occupés = états^{-1}[\{\text{TRUE}\}]$

INITIALISATION

$états := SIÈGES \times \{\text{FALSE}\} \;||\; nb\_libre := nb\_max$

Exemple de raffinement (de machine) : réservation

```
OPERATIONS
   /* réserver une place quelconque */
   place ← réserver ≙
   ANY p1 WHERE p1 ∈ états⁻¹[{FALSE}]
   THEN place,états(p1),nb_libre :=
          p1,TRUE,nb_libre-1
   END
   ;
```

Exemple de raffinement (de machine) : réservation

```
/* libérer une place précise */
r ← libérer(place) ≙
BEGIN
IF états(place)
THEN r,nb_libre := TRUE,nb_libre+1
ELSE r := FALSE
END
||
états(place) := FALSE
END
END
```

Implantation

Implantation

- variable concrète = variable de type concret
  type concret = ensemble simple concret (entier représentable, booléen, énumération) ou tableau (fonction totale, surjective ou injective ou bijective, d'un e.s.c. ou d'un produit d'e.s.c.vers un e.s.c.)
- ensemble et constante concrets : clause VALUES
  nécessitant une obligation de preuve
- prédicats valides : ¬ ∧ ∨
  opérateurs relationnels valides : < ≤ = ≠ ≥ >
  opérations arithmétiques valides : + - * /
- Instruction = substitution déterministe de l'implantation
- Opérations locales (non visibles de l'extérieur) : spécification et implantation

Exemple d'implantation :
réservation

IMPLEMENTATION

RÉSERVATION_Implantation

REFINES

RÉSERVATION_Raffinement

VALUES

nb_max = 4 ; *SIÈGES* = 1..4

CONCRETE_VARIABLES

états, *nb_libre*

INITIALISATION

états := *SIÈGES* × {FALSE} ; *nb_libre* := *nb_max*

Exemple d'implantation :

réservation

```
OPERATIONS
   /* réserver une place quelconque */
   place ← réserver ≙
   VAR ind,val
   IN ind := 1 ; val := états(ind) ;
   WHILE val
         DO ind := ind + 1 ; états(ind)
         INVARIANT ind ∈ 1..nb_max
                   ∧ val = états(ind)
                   ∧ états[1..ind-1] ⊆ {TRUE}
         VARIANT nb_max-ind
         END
   ;
   états(ind) := TRUE ; nb_libre := nb_libre-1 ; place := ind
   END
END
```

Exemple d'implantation :
division

```
IMPLEMENTATION
   DIVISION_Implantation
REFINES
   DIVISION
OPERATIONS
   /* a=dividende, b=diviseur, q=quotient, r=reste */
   q,r ← diviser(a,b) ≙
   VAR s,t
   IN s := 0 ; t := a ;
   WHILE t ≥ b DO s := s + 1 ; t := t - b
          INVARIANT s ∈ NAT ∧ t ∈ NAT ∧
                        a = b * s + t ∧ t ≥ 0
          VARIANT t END
   ; q := s ; r := t
   END
END
```

Exemple d'implantation :

tri

IMPLEMENTATION
TRI_Implantation /* tri par insertion */
REFINES
TRI
VALUES
$n = 100$
CONCRETE_VARIABLES
T
LOCAL_OPERATIONS
$k \leftarrow trouver\_min(j) \triangleq$
PRE $j \in 1..n\text{-}1 \wedge T \in 1..n \rightarrow$ NAT
THEN ANY p WHERE $p \in j..n \wedge T(p) = \min(T[j..n])$
THEN $k := p$ END
END
.../...

Exemple d'implantation :

tri

```
k ← trouver_min(j) ≙
  VAR p,l,a IN p := j+1 ; l := j ; a := T(l) ;
  WHILE p < n DO
    VAR tmp_p IN tmp_p := T(p) ;
    IF tmp_p < a THEN a := tmp_p ; l := p END
    END
    ; p := p+1
    INVARIANT p ∈ j+1..n ∧ l ∈ j..n-1 ∧ a ∈ NAT ∧
                    a = min(T$0[j..p-1]) ∧ a = T$0(l)
    VARIANT n-p+1 END ;
    VAR tmp_n IN tmp_n := T(n) ;
    IF tmp_n < a THEN k := n ELSE k := l END
    END
  END
```

Exemple d'implantation :

tri

trier ≙
VAR *j*,*l* IN *j* := 1 ; *l* := *n*-1 ;
WHILE *j* < *l* DO
VAR *k*,*tmp*
IN *k* ← *trouver_min(j)* ; *tmp* := *T(j)* ;
T(j) := *T(k)* ; *T(k)* := *tmp* END ;
; *j* := *j*+1
INVARIANT
$j \in 1..n \ \wedge \ T \in 1..n \rightarrow$ NAT $\wedge$
$(\forall(i_1,i_2) \bullet (i_1 \in 1..j\text{-}1 \wedge i_2 \in 1..j\text{-}1 \wedge i_1<i_2 \Rightarrow T(i_1) \leq T(i_2))) \ \wedge$
$(\forall(i_3,i_4) \bullet (i_3 \in 1..j\text{-}1 \wedge i_4 \in j..n \Rightarrow T(i_3) \leq T(i_4))) \ \wedge$
$(\exists h \bullet (h \in 1..n >\!\!\!\rightarrow\!\!\!> 1..n \wedge (h;T)=T\$0))$
VARIANT *n*-*j*
END
END

Module

Module

- Développement complet = projet dont l'objectif est de produire un programme exécutable conforme à sa spécification
- Projet = ensemble d'instances de modules
- Instance de module = copie d'une machine
- Module : composé de composants
- Module abstrait = machine sans raffinement ni implantation ni code
- Module développé = machine, son(ses) aucun/un/plusieurs raffinement(s), son implantation et son code exécutable
- Module/Machine de base = machine (sans raffinement ni implantation) codée dans le langage cible (C, ADA, etc. mais pas B)
  NB : la validation doit être spécifique

Module

- Interface = liste des opérations (avec profils) d'un module
- Services fournis par une machine : déclarations statiques, constantes concrètes et abstraites spécifiées par des propriétés, variables concrètes et abstraites spécifiées par un invariant, initialisation, opérations visibles
- Composant = machine ou raffinement ou implantation ou code exécutable
- Combinaison = liste (ou ensemble) de machines
- Composant « raffinable » = machine ou raffinement
- Composant raffinant = raffinement ou implantation

- Un composant « raffinable » ne peut pas être raffiné plus d'une fois
- Un composant raffinant raffine un seul autre composant

Module

- Dans un composant, les clauses d'assemblage servent à inclure (clause `INCLUDES`), importer (clause `IMPORTS`), voir (clause `SEES`) et utiliser (clause `USES`) une combinaison
- `INCLUDES` est une relation entre les spécifications tandis que `IMPORTS` et `SEES` sont des relations d'architecture de modules (i.e. que les programmes générés respectent cette modularité)
- `INCLUDES` et `IMPORTS` sont transitives contrairement à `SEES` et `USES` (pour les objets autres que les opérations)
- Dans un composant, on peut promouvoir (i.e les rendre visible dans son interface) une liste d'opérations d'une combinaison incluse ou importée (clause `PROMOTES`) ou toutes les opérations d'une combinaison (clause `EXTENDS`)

Module

- Les clauses d'assemblage permettent de préserver les invariants sans avoir à faire de preuve supplémentaire
- Un module ne peut être inclus qu'une fois dans un projet
- Un module ne peut être importé qu'une fois dans un projet
- Cas d'utilisation typiques selon une architecture :
  - sans partage : `INCLUDES` et `IMPORTS`
  - avec partage [de variables en consultation ou d'ensembles, de constantes] : `SEES` et `USES`
    (1 écrivain et plusieurs lecteurs)

Module

Restriction d'accès aux services de la combinaison :

- déclarations statiques et constantes sans restriction

- `INCLUDES`
  - variables en lecture
  - opérations sans appels simultanés modifiant la même machine incluse
- `IMPORTS`
  - variables concrètes en lecture ; variables abstraites modifiables uniquement avec les opérations de la combinaison (encapsulation)
  - opérations utilisables
- `SEES`
  - variables et opérations en lecture
- `USES`
  - variables en lecture
  - interdiction d'utiliser les opérations

Exemple de module :
instances multiples

MACHINE *À_INSTANCIER*
SETS *ENS* = 0..9
CONSTANTS *const*
PROPERTIES *const* = 2
VARIABLES *var*
INVARIANT *var* ⊆ *ENS*

MACHINE *CRÉE_DEUX_INSTANCES*
INCLUDES *INSTANCE_1. À_INSTANCIER ;*
INSTANCE_2. À_INSTANCIER
INVARIANT card(*INSTANCE_1.var*)+ *const* =
card(*INSTANCE_2.var*)

Exemple de module : réservation

```
MACHINE
   MAIN_RÉSERVATION
OPERATIONS
   main ≙ skip
END
```

```
IMPLEMENTATION
   MAIN_RÉSERVATION_Implantation
REFINES
   MAIN_RÉSERVATION
IMPORTS
   RÉSERVATION, BASIC_IO
```

Exemple de module : réservation

```
main ≙ VAR essai,code,pl
  IN essai := maxint ; code := 1 ; tmp := 1 ;
  WHILE 0 < essai ∧ code ≠ 0
    DO essai := essai - 1 ;
       STRING_WRITE("Tapez 0=Quitter, 1=Réserver,
                              2=Libérer, 3=Place libre\n") ;
       code ← INTERVAL_READ(0,3) ;
    CASE code OF
      EITHER 0 THEN skip
      OR 1 THEN
        pl ← place_libre ;
        IF pl = 0
        THEN STRING_WRITE("Plus de place\n")
        ELSE pl ← réserver ;
             STRING_WRITE("Réservation place ") ;
             INT_WRITE(pl) ; STRING_WRITE("\n")
        END
```

Exemple de module : réservation

```
      OR 2 THEN ...
      OR 3 THEN ...
      END
   INVARIANT
      essai ∈ NAT  ∧ code ∈ NAT  ∧ pl ∈ NAT
      ∧ occupés ⊆ SIÈGES ∧ nb_libre ∈ 0..nb_max
      ∧ nb_libre = nb_max-card(occupés)
    VARIANT essai
    END
END
```

Exemple de module :

interrupteur (INCLUDES)

MACHINE
  INCLUDES_INTERRUPTEUR
INCLUDES
  INTERRUPTEUR
VARIABLES
  lumière
INVARIANT
  lumière ∈ BOOL ∧ *(lumière* ⇔ *é* = allumé*)*
INITIALISATION
  lumière := FALSE
OPERATIONS
  basculer ≙
    BEGIN *lumière* := ¬ *lumière* || *changer* END
END

Exemple de module :
interrupteur (SEES)

MACHINE	*SEES_CUISINE_INTERRUPTEUR*
SEES	*INTERRUPTEUR*
CONCRETE_VARIABLES	*bouton_cuisine*
INVARIANT	*bouton_cuisine* $\in$ *ÉTATS*

MACHINE	*SEES_CHAMBRE_INTERRUPTEUR*
SEES	*INTERRUPTEUR*
VARIABLES	*bouton_chambre*
INVARIANT	*bouton_chambre* $\in$ *ÉTATS*
INITIALISATION	*bouton_chambre* := éteint

Exemple de module :
vie professionnelle des personnes (USES)

MACHINE *USES_TRAVAILLER*
SETS *ENTREPRISES*
USES *PERSONNES*
VARIABLES *travailler*
INVARIANT
$travailler \in PERSONNES \leftrightarrow ENTREPRISES$
$\wedge\ personnes \subseteq \mathtt{dom}(travailler)$
INITIALISATION $travail := \varnothing$

MACHINE *INCLUDES_USES_TRAVAILLER*
INCLUDES *PERSONNES*, *USES_TRAVAILLER*