Formalismes de Spécification Formelle

Support de cours

Master 2 SIA

Dr. ZERARGA Loutfi

Département d'Informatique
Faculté des Sciences Exactes
Université de Bejaia

Année Universitaire : 2023/2024

Table des matières

Avant-Propos

Ce support de cours "formalismes de spécification formelle" est destiné aux étudiants en deuxième année de Master informatique de la spécialité "Système d'Information Avancée". L'objectif de cette matière est l'étude des différentes méthodes formelles de spécification. Ces méthodes ont contribué considérablement aux progrès du génie logiciel. Elles s'appuient sur des formalismes rigoureux, qui permettent de prouver des propriétés des spécifications et de les valider assez tôt dans le cycle de développement.

Conformément au programme en vigueur, ce document est scindé en 6 chapitres qui reprennent l'essentiel, illustrent les cours et aident à la compréhension du contenu de la matière :

Le chapitre 1 est une introduction aux méthodes de spécifications formelles et de ses principaux courants. Ces méthodes se basent sur les mathématiques (Logique, théorie des ensembles ...) qui garantissent une certaine rigourosité et une certaine fiabilité des spécifications.

Le second chapitre présente un rappel du calcul propositionnel, du calcul de prédicats et de la théorie des ensembles. Les notions rappelées dans ce chapitre sont primordiales pour la compréhension des deux langages de spécification, Z et B, présentés dans ce support.

Les chapitres 3 et 4 résument la notation de base du langage Z et présentent les fondements et la démarche de la spécification Z.

Le chapitre 5 est une introduction au langage B. Il présente la démarche globale de spécification, la notation et les machines abstraites du langage B.

Le dernier chapitre introduit les Réseaux de Petri qu'est un outil de modélisation utilisé généralement en phase préliminaire de conception de système (informatique, industriel, ...) pour leur spécification fonctionnelle, modélisation et évaluation.

Chapitre 1

Introduction à la Spécification Formelle

Sommaire

1.1 Introduction

Les méthodes formelles sont des techniques permettant de raisonner rigoureusement, à l'aide des mathématiques, sur des programmes informatiques ou des matériels électroniques, afin de démontrer leur validité par rapport à une certaine spécification. Ces méthodes permettent d'obtenir une très forte assurance de l'absence de bug dans les logiciels, c'est-à-dire d'acquérir des niveaux d'évaluation d'assurance élevés, elles sont basées sur les sémantiques des programmes. Ce chapitre est une introduction aux méthode de la spécification formelle et ces principaux courants[1].

1.2 De l'origine de la spécification formelle

Les méthodes de spécification formelles sont issues des travaux du Programming Research Groupe de l'université d'Oxford (début des années 80). Ce groupe préconisait l'utilisation intensive des mathématiques dans la spécification des systèmes informatiques.

En 1984, J.R.Abrial démontre le bien fondé des mathématiques dans les spécifications. Il décrit la spécification et la construction rigoureuse et systématique d'un algorithme de modification de la mémoire d'un ordinateur. Il procède par étapes, dites de "réalisation successives", de la spécification axiomatique par pré-conditions et post-conditions à l'algorithme du programme, en utilisant la théorie des ensembles et la logique des prédicats.

L'expression des besoins nécessite un formalisme rigoureux pour :

— bien définir les notions du domaine utilisateur et les termes employés.
— éviter les non-sens et la prépondérance (prédominance) des informaticiens dans les décisions.
— éviter les incompréhensions entre les différentes acteurs.

Le choix de la notation est primordial dans la compréhension de la spécification, et la traduction en langage naturel de cette spécification formelle est nécessaire.

1. Les notions présentées dans ce chapitre sont, principalement, extraites du livre "Génie logiciel : Spécification des logiciels deux exemples de pratiques récentes Z et UML" de P. André et A. Vailly[3]

1.3 Les systèmes formels

Les systèmes formels ont été conçus par les logiciens afin de poser et étudier mathématiquement certains problèmes liés au langage mathématique. De ce point de vue on peut les considérer comme des métathéories générales, des théories sur les théories (mathématiques).

Un système formel, décrit les règles de manipulation d'un ensemble de symboles traiter de façon uniquement syntaxique. Il est constitué d'un langage formel et d'un système de déduction (axiomes et règles d'inférences).

Le langage formel comprend deux parties : l'alphabet et la syntaxe. il est décrit par un métalangage (grammaire). Un groupe de symboles est un mot. Les lots accepter par la grammaire sont des formules bien formées (ou construites) (fbf).

— les mots et les termes pour la théorie des langages.
— les formules pour la logique.
— les termes en algèbre.
— les éléments pour la théorie des ensembles.
— les expressions en arithmétique.

Le système d'inférence permet de manipuler les expressions indépendamment de leur interprétations. Il est constitué d'axiomes et de règles d'inférence (inférence et/ou déduction).

Pour être utilisable, les mots produit par le système doivent avoir une signification ou un sens. Une sémantique est alors associée au système d'inférence. Le sens des formules bien formées est donnée en attribuant une interprétation au langage.

Une preuve (ou démonstration) dans un système formel $\mathcal{F}$ est un ensemble structuré d'étapes correctes de raisonnement. Dans une démonstration, chaque étape est soit un axiome, soit l'application d'une règle d'inférence qui permet d'affirmer qu'une formule bien formée, la conclusion, est une conséquence logique d'une ou plusieurs autres formules, les prémisses de la règle. Les prémisses sont soit des axiomes, soit des formules déjà obtenues comme conclusions de l'application d'autres règles. Une formule bien formée qui est la conclusion de l'étape ultime d'une preuve est un théorème (syntaxique). Un lemme est une preuve servant à la démonstration d'un théorème.

Autrement dit, la dérivation d'une formule bien formée W dans un système formel $\mathcal{F}$ à partir d'un ensemble de prémisses$\mathcal{P}$, est une séquence finie de formules de $\mathcal{F}$ se terminant par la conclusion W. Un élément quelconque de la dérivation est soit un axiome, soit une prémisse, soit le résultat d'application d'une règle d'inférence (i.e. théorème ou lemme). On parle aussi de conséquence syntaxique $\mathcal{P} \vdash W$. Une preuve

est un cas particulier de la dérivation (ou la conséquence syntaxique) pour laquelle $\mathcal{P} = \emptyset$.

La construction d'un système formel se fait avec une interprétation sous-jacente. L'interprétation associe une valeur à chaque formule bien formée. On distingue deux cas particulier : la confusion et le rebut. il y a une confusion lorsque l'interprétation affecte la même valeur à des fbcs différentes. Un rebut (Dic. Ce qu'il y a de plus mauvais (dans un ensemble)) est une valeur de l'interprétation associée à aucune formule bien formée. Une interprétation est cohérente s'il n'y a pas de confusions. elle est complète s'il n'y a pas de rebuts. Une contingence[2] est une expression logique qui s'évalue à vrai dans certains contextes et à faux dans d'autres contextes. Une contradiction (ou incohérence) est une expression logique qui s'évalue à faux dans tous les contextes[3]. Une Tautologie est une expression logique qui s'évalue à vraie dans tous les contextes.

Un système est consistant (ou cohérent) pour une interprétation si chaque théorème, provable par le système formel, est interprété par vrai. Plus précisément, pour un ensemble de formules bien formées E on ne peut déduire une contradiction. Une interprétation est complète si chaque valeur *vrai* peut être prouvée par un théorème (Remarque : la plupart des systèmes formels sont incomplets).

Un système formel est aussi quelquefois appelé une axiomatique ou une théorie, l'alphabet, supposé fini, est appelé vocabulaire, on y a distingue les constantes, les variables et les opérateurs. On a alors {théorèmes} $\subset$ {mots} $\subset$ {chaine de l'alphabet}

1.4 La spécification formelle

Une spécification formelle est exprimée dans un langage à syntaxe et sémantique précises, construite sur une base théorique *cohérente* et qui permet des validations automatisées. cette base est généralement mathématique. Exemples : Les réseaux de PETRI, les grammaires formelles, les automates à états finis, la logique formelle, l'algèbre, la théorie des graphes, le $\lambda-$calcul.

— Les Réseaux de PETRI : est un modèle mathématique permettant la représentation de systèmes distribués discrets (informatique, industriel). Les Réseaux de PETRI sont largement utilisés pour spécifier et valider des protocoles de communication et les systèmes dynamiques à événements discrets.

2. Dans la philosophie et la logique, la contingence est le statut des propositions qui ne sont ni toujours vraies, indépendamment de toute valeur de vérité, ni toujours fausses. Une proposition contingente n'est donc ni nécessairement vraie, ni nécessairement fausse.

3. contextes : interprétations

— Les grammaires formelles sont utiliser pour définir et analyser les langages (en théorie de langages de programmation).
— Les automates à états finis : modélisation de mécanismes critiques (moniteurs temps réel, postes de commande).
— La logique formelle et les méthodes algébriques permettent de formaliser et de démonter des algorithmes essentiels[4].
— La théorie des ensembles : support de bases de données relationnelles et des spécification abstraites des systèmes.

1.5 L'utilisation des méthodes formelles

L'étape de la spécification est reconnue comme étant primordiale dans le processus de développement du logiciel (les erreurs de spécification sont les plus coûteuses à corriger). La spécification formelle est une bonne réponse à ce problème. Les méthodes formelles sont associées à trois sortes d'activités : (1) écrire des spécifications formelles, (2) prouver des propriétés et (3) implanter le logiciel.

La spécification formelle permet la construction des programmes par diverses approches : la preuve, la transformation et la construction assistée de programmes. Elle met en valeur certain nombre de propriétés que doit respecter le logiciel. Un plan de test peut donc être établi comme pour le test structurel dans les langage de programmation.

Bien qu'une spécification soit destinée à être lue, il se peut que le langage utilisé soit exécutable (E-LOTOS[5]). Dans ce cas On peut obtenir rapidement un prototype, une maquette du système à développer. Le prototype est un élément de communication et de retour d'informations entre les différents acteurs. Ces prototypes peuvent guider le processus de test et celui de conception en facilitant la comparaison des deux.

La réutilisation est l'une des voies que la recherche en génie logiciel explore activement pour bénéficier d'une baisse substentielle des coûts de développement de logiciels. Dans cette voie, les spécifications formelles jouent un rôle fondamentale dans la réutilisation, d'une part en donnant précisément les fonctionnalités d'un module

4. Par rapport aux besoins

5. Language Of Temporal Ordering Specification (LOTOS) est un langage de spécification formel basé sur l'ordre temporel des événements. LOTOS est utilisé pour la spécification des protocoles dans les standards ISO OSI.LOTOS est un langage algébrique qui consiste en deux parties : une partie de description des données et des opérations, basé sur des types abstraits de données, et une partie pour la description des processus concurrents, basé sur l'algèbre de processus.

ou composant réutilisable et d'autre part en facilitant l'intégration d'autres modules et composants lors de raffinement.

1.5.1 Les avantages (ou intérêts)

Les avantages des méthodes formelles sont dûs :

Aux aspects formels :

- contrairement aux autres formes de spécifications, des propriétés peuvent être établies par raisonnement formel. Les preuves formelles ne sont pas intuitives mais basées sur règles strictes.
- les conséquences d'une spécification peuvent être mises en évidence et les contradictions sont détectées avant la réalisation.
- les outils de preuve calculent est vérifient automatiquement uniquement ce qui a été décrit.

A la précision :

- la spécification formel est une pré-réalisation d'un projet, et le programmeur sait exactement ce qu'il doit programmer.
- le domaine du problème est mieux perçu, les imprécision sont levées et le problème est mieux cerné.
- le même langage est pratiqué par différents intervenants dans la réalisation du projet.
- une traduction de la spécification en langage naturel facilitera la discussion avec le client. Améliore, également, la clarté dans le dialogue.

A l'abstraction : on expose ce qui doit être réalisé mais pas comment cela doit l'être.

- une abstraction permet de ne retenir que les caractéristiques essentielles (i.e. plus court et plus lisible).
- une description abstraite est plus évolutive aux changements des besoins.
- elle améliore la maintenabilité du logiciel et facilite l'accès au logiciel de nouveaux acteurs de développement.
- l'abstraction est l'un des concepts clés de l'informatique. Son objectif principal est de gérer la complexité en masquant les détails inutiles à l'utilisateur. Elle permet, alors, une distinction plus nette entre étude et réalisation, pour lesquelles les choix et les décisions sont différents.
- les méthodes formelles mènent à des composants plus facile à réutiliser et plus fiable.

1.5.2 Les inconvénients (ou problèmes)

En pratique : même avec des notations formelles, on peut décrire des spécifications de mauvaise qualité. Le formalisme doit être suffisamment abstrait pour conserver les qualité et suffisamment rigoureux pour permettre les preuves. Voici quelques inconvénients :

— Difficulté : les praticiens n'ont pas toujours la connaissance nécessaire. Les formalismes ne sont pas toujours très lisible et accessibles " *les spécifications formelles exigent un certain niveau d'aptitude mathématique*"
— Diversité : plusieurs standards existent (Z, VDM "Vienna Development Method", Modèles algébriques, etc ...)
— Adéquation : La spécification formelle aide à cerner les oublis et les contradiction, mais on ne peut pas prouver que ce qui a été spécifié.
— Domaine d'application : Les spécification formelles ne sont appliquées qu'à un nombre réduit d'application, certains éléments de langages de programmation (le parallélisme, l'arithmétique en virgule flottante ...) sont encore difficiles à modéliser de manière satisfaisante.
— Structuration : absence d'outils de structuration de la spécification.

Nous abordons maintenant les différentes approches de la spécification formelle.

1.6 La spécification par modèle abstrait

Un type de donnée abstrait (en anglais, abstract data type ou ADT) est une spécification mathématique d'un ensemble de données et de l'ensemble des opérations qu'on peut effectuer sur elles. On qualifie d'abstrait ce type de donnée car il ne spécifie pas comment les données sont représentées ni comment les opérations sont implémentées.

Dans cette approche, il est construit un modèle particulier du type à spécifier. Ce modèle possède un état, défini par une structure de données (représentation concrète). L'état est défini en terme d'autre type de données, et ce récursivement jusqu'à obtenir des combinaisons de types de base (entiers, réels, caractère, booléen). Il s'agit d'une structuration de types. L'approche est constructive. Les constructeurs sont issus des mathématiques (ensembles, produits cartésiens, séquences), des langages de programmation ou algébrique (listes, arbres). Les nouvelles opérations sont définies à partir des opérations des structures de données. Cette spécification procédurale associée peut être opérationnelle si elle est exprimée un calcul (machine abstraite : Turing, auto-

mate fini ou machine à pile, suite de transformation, algorithme), ou axiomatique s'il s'agit de formules logiques (Z, VDM).

Une spécification par modèle abstrait correspond à un système formel dans la syntaxe est donnée par les types et opérations du modèle abstrait et dans la sémantique est donnée par la théorie sous-jacente au modèle abstrait. Exemple : théorie des ensemble, logique du premier ordre, théorie des types (μ-calcul).

1.7 Les spécifications algébriques

Une spécification algébrique de type abstrait de données, ou type abstrait de données (*type abstrait algébrique*), est la donnée d'une signature, notée Σ, et d'un ensemble d'axiomes (formules logiques), noté Σ. La signature comprend des noms, de types (les sortes) et des opérations définies par leur arités i.e. les sortes de leur domaine et co-domaine. La signature permet de construire toutes les valeurs des types de données par application des opérations. Le type abstrait algébrique est dit *équationnel* Si les axiomes sont réduits aux équation. Il est dit *positif conditionnel* si les axiomes sont réduits aux implications de conjonctions d'équation vers une équation (clauses de HORN avec égalité). Ce type de spécification est donc, un système formel dont le langage est donné par la signature et le système d'inférence est basé sur les axiomes et la déduction équationnelle (exemple Fig.1.1), sont interprétation est donné en termes d'algèbres (particulièrement multi-sortes).

$$\frac{s \doteq t \in \mathcal{E}}{s \doteq t}\ \text{(axiome)} \qquad \frac{}{s \doteq s}\ \text{(réflexivité)}$$

$$\frac{s \doteq t}{t \doteq s}\ \text{(symétrie)} \qquad \frac{s \doteq t \quad t \doteq u}{s \doteq u}\ \text{(transitivité)}$$

$$\frac{s \doteq t}{\sigma(s) \doteq \sigma(t)}\ \text{(substitution)} \qquad \frac{s \doteq t}{u[s]_p \doteq u[t]_p}\ \text{(contexte)}$$

FIGURE 1.1 – Règles syntaxiques pour le raisonnement équationnel

Remarque : une algèbre sur un corps commutatif K, ou simplement une K-algèbre, est une structure algébrique $(A, +, *, \times)$ telle que :

— $(A, +, *)$ est un espace vectoriel sur K, ($*$ multiplication par un scalaire) ;

— la loi $\times$ est définie de $A \times A$ dans A (loi de composition interne) ;
Exemple de langages de spécification Algébrique : CLEAR, ASL, ACT-ONE, OBJ2, ...

1.8 Approche dynamique

L'approche dynamique modélise un module par processus. L'importance est donnée à l'interaction entre modules. Exemple : CSP, CCS, les systèmes de transitions (Réseau de petri, Automates), les logiques temporelles (La logique temporelle est une branche de la logique mathématique et plus précisément de la logique modale).

1.9 Les spécifications Mixtes

Les approches mixtes cumulent différents styles de la classification. Le panache peut se faire avec les langages exprimant soit des aspects identiques soit par aspects orthogonaux du système à spécifier. Exemple : LOTOS est un langage basé sur l'algèbre de processus CCS, enrichie par certains mécanismes de CSP. Les données sont décrites par le langage de spécification algébrique ACT-ONE.

1.10 Langages formels : aperçu

Un langage de spécification est dit formel lorsque sa syntaxe et ses notations sont rigoureusement définies par une sémantique précise, c'est-à-dire avec des modèles mathématiques. Une méthode de spécification est dite formelle lorsqu'elle utilise un ou plusieurs langages de spécification formels. Il est possible de classer les méthodes et les langages de spécification formelles en deux groupes :

— Les approches (et langages) basées sur les états : représentent le système à travers deux modèles complémentaires : la partie statique permet de décrire les entités constituant le système et leurs états, tandis que la partie dynamique modélise les changements d'états que le système peut effectuer par l'intermédiaire d'opérations ou d'actions (Exemple : Statechart, Esterel, ASM, Action Systems, VDM, Z, Object-Z et B).

— Les approches (et langages) basées sur les événements : Les approches basées sur les événements représentent le système à travers des processus ou des agents,

qui sont des entités indépendantes qui communiquent entre elles ou avec l'extérieur du système. Ces méthodes permettent de modéliser le comportement du système à l'aide de séquences ou d'arbres d'événements (exemple : les réseaux de Petri, LOTOS, CCS, CSP et EB3).

La section suivante présente quelques exemples de langages formels appartenant à ces deux approches.

Action Systems Les Action Systems sont basés sur les systèmes de transitions. Plusieurs versions existent, comme celles de BACK et UNITY. Un système d'actions est composé de :

— un espace d'états, défini par des variables d'états,
— des actions qui définissent une initialisation et des transitions sur ces variables d'états.

Langage Z Le schéma est la notion de base des spécifications Z. Un schéma est une boîte contenant des descriptions utilisant les notations Z. Les schémas sont utilisés pour décrire les états d'un système, les états initiaux ou bien les opérations. La partie statique permet de définir les états et les relations d'invariant qui sont préservées lors des transitions d'états. Elle est décrite en Z sous la forme d'un schéma d'état. Les aspects dynamiques concernent les opérations, les relations entre les entrées et les sorties, et les changements d'états.

Object-Z est une extension du langage Z qui permet de spécifier des systèmes dans un style orienté objet.

Langage B Le langage B est un langage de spécification formel, basé sur la notion de machine abstraite. Cette dernière représente un état spécifié par une partie statique (à l'aide de variables d'état et des propriétés d'invariance) et une partie dynamique (à l'aide d'opérations). Le langage pour la description de la statique repose sur la théorie des ensembles et sur la logique du premier ordre. Une machine abstraite B est ensuite raffinée. Cette phase permet de passer d'une structure abstraite à une structure proche du code. Le raffinement B se fait en plusieurs étapes successives. Les machines issues du raffinement ne contiennent alors ni précondition, ni indéterminisme.

Calculus of Communicating Systems (CCS) CCS est fondé sur l'observation d'agents qui représentent une partie unitaire d'un système concurrent à modéliser. La notion d'agent peut désigner des parties de granularité variable du système. Un agent peut donc être composé de plusieurs sousagents. Les deux notions fondamentales de CCS sont concurrence et communication. La concurrence est caractérisée en CCS par l'indépendance des actions d'un agent par rapport aux autres agents du système. Quand à la communication entre agents CCS, elle se repose sur deux types : les

actions peuvent agir à l'intérieur d'un agent ou bien interagir avec ses agents voisins. Le comportement d'un système est alors défini par l'observation de ses actions.

Communicating Sequential Processes (CSP) Est une notation utilisée pour décrire des systèmes concurrents. Le langage est supporté par quelques outils, comme FDR, qui permettent d'analyser et de vérifier les spécifications en cours ou existantes. En CSP, les processus sont des entités, indépendantes les unes des autres, mais qui peuvent communiquer entre elles. Un processus peut exécuter des événements (ou actions). Les événements permettent de décrire le comportement des processus. L'ensemble des événements qu'un processus peut exécuter est appelé son alphabet (ou interface).

Chapitre 2

Calcul Propositionnel, Calcul des Prédicats et Théorie des Ensembles

Sommaire

2.1 Introduction

L'objet de ce chapitre est de rappeler des connaissances de premier cycle qui seront fondamentales dans la suite du cours. On rappellera en particulier les notions de base de la logique propositionnelle, de la logique du premier ordre et de la théorie des ensembles *ZF*.

2.2 Calcul Propositionnel

La logique propositionnelle ou le langage formel est construit(e) à partir de variables (propositionnelles) et des opérateurs. Cette notation associée aux règles de calcul permettant de déterminer la valeur de vérité de ses formules (ou expressions). On cherche, alors, la validité d'une formule (vrai/faux).

Les méthodes de décision pour la validité donnent une réponse correcte (oui ou non) a toutes les questions concernant la validité des formules.

Il y a deux aspects principaux dans la description d'un langage (deux types de méthodes de décision) : sa syntaxe et sa sémantique. La syntaxe concerne la structure du langage et la forme de ses énoncés ; la sémantique s'intéresse au "sens" (ici mathématique) à donner au langage.

2.2.1 Les méthodes sémantiques

Basées sur la recherche de modèles : Manipules des valeurs logiques (v .f) (Table de verité ,Tableau sémantique).

Pas praticable en général si le nombre de variable est important .

Exemple table de vérité.

La méthode des tableaux est un algorithme qui consruit un arbre(tableau)permettant de répondre de façon systématique a la question : la formule A est elle valide ?

Principe : si une formule A est une satisfiable, sa négation est insatisfiable

2.2.2 Les méthodes syntaxique (déductive)

A pour objet de calculer les conséquences logiques par l'application de règles d'inférence.

Pour cela on construit des systèmes formels d'inférence composés d'axiomes (formules)

et de règles d'inférence Produire des formules qui sont des conséquences logiques des formules de départ

Exemples : système de Hilbert (un grand nombre d'axiomes logiques exprimant les principales propriétés de la logique que l'on combine au moyen de quelques règles, Axiomes pour l'implication, Axiomes pour la conjonction ...), système de déduction naturelle (où les règles de déduction des démonstrations sont proches des façons naturelles de raisonner ≠système de Hilbert) , principe de résolution de Robinson (est une règle d'inférence logique qui généralise le modus ponens, à la base du langage Prolog),

2.2.3 Formules bien formées

Définition (formule) : L'ensemble FOR des formules (ou formules bien formées) de la logique propositionnelle est le plus petit ensemble de mots construits sur l'alphabet tel que :

— si A est une formule atomique alors A est une formule
— $(\neg A)$ est une formule si A est une formule
— $A \wedge B$ est une formule si A et B sont des formules
— $A \vee B$ est une formule si A et B sont des formules
— $A \Rightarrow B$ est une formule si A et B sont des formules
— $A \Leftrightarrow B$

Une proposition atomique est appelée variable propositionnelle.

2.2.4 Déduction naturelle

Des règles d'inférence (ou de déduction) viennent enrichir la manipulation des propositions. L'ensemble de ces règles forme le système de déduction naturelle. Il n'y a pas d'axiomes dans ce système.
Une règle d'inférence se présente sous la forme $\frac{Prémisses}{conclusion}[nom]$

Soit A, B et C des variables propositionnelles, et S des formules déjà dérivées.

TABLEAU 2.1: Règles de déduction du calcul des propositions

Nom	Règle d'inférence

$\wedge-$introduction	$\frac{A;B}{A\wedge B}$ ou $\frac{A;B}{B\wedge A}$
$\wedge-$élimination	$\frac{A\wedge B}{A}$ ou $\frac{A\wedge B}{B}$
$\vee-$introduction	$\frac{A;B}{A\vee B}$ ou $\frac{A;B}{B\vee A}$
$\vee-$élimination	$\frac{S,A\vdash C;S,B\vdash C;A\vee B}{C}$ Sous hypothèse A,B
$\neg-$introduction	$\frac{S,A\vdash B;S,A\vdash\neg B}{\neg A}$ A est une hypoyhèse
$\neg-$élimination	$\frac{\neg\neg A}{A}$
$\Rightarrow-$introduction	$\frac{S,A\vdash B}{A\Rightarrow B}$, A est une hypothèse
$\Rightarrow-$élimination	$\frac{A;A\Rightarrow B}{B}$
$\Leftrightarrow-$introduction	$\frac{A\Rightarrow B;B\Rightarrow A}{A\Leftrightarrow B}$
$\Leftrightarrow-$élimination	$\frac{A\Leftrightarrow B}{A\Rightarrow B}$ $\frac{A\Leftrightarrow B}{B\Rightarrow A}$

TABLEAU 2.2: Quelques propriétés et théorèmes du calcul des propositions.

Nom	Propriété	Proposition
P_1	Commutativité	$P \wedge Q \Leftrightarrow Q \wedge P$ et $P \vee Q \Leftrightarrow Q \vee P$
P_2	Associativité	$P \vee (Q \vee R) \Leftrightarrow (P \vee Q) \vee R$ $P \wedge (Q \wedge R) \Leftrightarrow (P \wedge Q) \wedge R$
P_3	Distributivité	$P \wedge (Q \vee R) \Leftrightarrow (P \wedge Q) \vee (P \wedge R)$ $P \vee (Q \wedge R) \Leftrightarrow (P \vee Q) \wedge (P \vee R)$
P_4	Lois de DE MORGAN formes normales	$\neg(P \wedge Q) \Leftrightarrow (\neg P \vee \neg Q)$ $\neg(P \vee Q) \Leftrightarrow (\neg P \wedge \neg Q)$
P_5	Forme normale	$P \Rightarrow Q \Leftrightarrow \neg P \wedge Q$
P_6	Absorption	$P \wedge (P \vee Q) \Leftrightarrow P$ $P \vee (P \wedge Q) \Leftrightarrow P$ $faux \wedge P \Leftrightarrow p$ $vrai \vee P \Leftrightarrow vrai$
P_7		$P \Rightarrow (Q \Rightarrow R) \Leftrightarrow (P \wedge Q) \Rightarrow R$

P_8	$\Leftrightarrow$ –élimination	$(P \Leftrightarrow Q) \Leftrightarrow (P \Rightarrow Q) \wedge (Q \Rightarrow P)$
P_9		$(P \Rightarrow Q) \Leftrightarrow (\neg Q \Rightarrow \neg P)$ $(P \Rightarrow Q) \Leftrightarrow (P \Leftrightarrow (P \wedge Q)$ $(Q \Rightarrow P) \Leftrightarrow (P \Leftrightarrow (P \vee Q)$
P_{10}	élément neutre	$vrai \wedge P \Leftrightarrow P$ $faux \vee p \Leftrightarrow P$ $vrai \Rightarrow P \Leftrightarrow p$
P_{11}		$faux \Rightarrow P \Leftrightarrow vrai$
P_{12}		$P \Rightarrow vrai \Leftrightarrow vrai$
P_{13}		$P \Rightarrow faux \Leftrightarrow \neg P$
P_{14}	Tiers exclus	$P \vee \neg P \Leftrightarrow vrai$
P_{15}	Contradiction	$P \wedge \neg P \Leftrightarrow faux$
P_{16}	$\wedge-$ simplification	$P \wedge P \Leftrightarrow P$
P_{17}	$\vee-$ simplification	$P \vee P \Leftrightarrow P$

2.3 Le Calcul des Prédicats

Le langage des prédicats est celui des propositions enrichi par des variables et des quantificateurs ∃ et ∀. La sémantique des quantificateurs dépend des valeurs que peut prendre la variable quantifiée. La table ci-dessous présente la syntaxe de base du calcul de prédicats et la sémantique des quantificateurs.

TABLEAU 2.3: La sémantique des quantificateurs

$P(x)$	Prédicat paramétré par la variable x, dite libre
$\forall x\ P(x)$	x est liée universellement, P est vrai pour toute valeur de x
$\exists x\ P(x)$	x est liée existentiellement, il y'a au moins une valeur de x tel que P soit vrai
$\exists! x\ P(x)$	x est liée par unicité, c-a-d il y'a une seule valeur de x tel que P soit vrai

Le quantificateur d'unicité est défini formellement par :

$$\exists x\ P(x) \wedge \neg(\exists y\ x \neq y \wedge P(y))$$

2.3.1 Déduction

Le calcul des prédicats s'enrichit des règles d'inférence suivantes :

TABLEAU 2.4: Règle de déduction du calcul des prédicats

$\forall-$introduction	$\frac{P(a)}{\forall x\ P(x)}$ où a est un terme arbitraire

$\forall$−élimination	$\frac{\forall x\ P(x)}{P(a)}$ où a est un terme arbitraire
$\exists$−introduction	$\frac{P(a)}{\exists x\ P(x)}$ où a un terme donné
$\exists$−élimination	$\frac{\exists x\ P(x), \forall x(P(x) \Rightarrow Q)}{Q}$ où Q est une proposition

2.3.2 Propriétés

TABLEAU 2.5: Propriétés du calcul des prédicats

	$\neg(\forall x\ P(x)) \Leftrightarrow (\exists x\ \neg P(x))$
	$\neg(\forall x\ \neg P(x)) \Leftrightarrow \neg(\exists x\ P(x))$
	$\neg(\forall x\ P(x)) \Leftrightarrow \neg(\exists x\ \neg P(x))$
	$\neg(\forall x\ \neg P(x)) \Leftrightarrow (\exists x\ P(x))$
Distribution de $\forall$ sur $\wedge$	$\forall x\ (P(x) \wedge Q(x)) \Leftrightarrow (\forall x P(x) \wedge \forall x Q(x))$
Distribution de $\forall$ sur $\vee$	$\forall x\ (P(x) \vee Q(x)) \Leftrightarrow (\forall x P(x) \vee \forall x Q(x))$
Distribution de $\exists$ sur $\wedge$	$\exists x\ (P(x) \wedge Q(x)) \Leftrightarrow (\exists x P(x) \wedge \exists x Q(x))$

Distribution de $\exists$ sur $\vee$	$\exists x\ (P(x) \vee Q(x)) \Leftrightarrow (\exists x P(x) \vee \exists x Q(x))$
Simplification	$\forall x\ A \Leftrightarrow \exists x\ A \Leftrightarrow A$
des variables	$\forall x\ (P(x) \wedge A) \Leftrightarrow (\forall x\ P(x)) \wedge A$
libres	$\forall x\ (P(x) \vee A) \Leftrightarrow (\forall x\ P(x)) \vee A$
	$\exists x\ (P(x) \wedge A) \Leftrightarrow (\exists x\ P(x)) \wedge A$
	$\exists x\ (P(x) \vee A) \Leftrightarrow (\exists x\ P(x)) \vee A$
Simplification	$\forall x\ (P(x) \Rightarrow A) \Leftrightarrow (\exists x\ P(x)) \Rightarrow A$
de $\Rightarrow$	$(\forall x\ A\ \Rightarrow P(x)) \Leftrightarrow (A \Rightarrow (\forall x\ P(x)))$
	$\exists x\ P(x) \Rightarrow N) \Leftrightarrow ((\forall x\ P(x)) \Rightarrow A)$
	$\exists x\ N \Rightarrow P(x)) \Leftrightarrow (A \Rightarrow (\exists x\ P(x))$

2.4 La Théorie des Ensembles ZF

Le contexte est celui d'une théorie où tous les objets sont des ensembles, en particulier A est un ensemble d'ensembles, sinon il faut le préciser Ces axiome sont écrit dans le langage formel de la théorie des ensembles, qui est un langage égalitaire du premier ordre avec la relation d'appartenance comme seul symbole primitif non logique. On peut définir formellement l'inclusion :

$$A \subseteq B \ ::: \ \forall x(x \in A \Rightarrow x \in B)$$

Remarque *: l'appartenance est un prédicat.*

2.4.1 Axiome d'extensionalité

Deux ensembles sont égaux, si et seulement si ils ont les mêmes éléments. En formule,

$$\forall A \ \ \forall B \ \ [\forall x \ \ (x \in A \Leftrightarrow x \in B)) \Rightarrow (A = B)]$$

2.4.2 Axiome de la paire

Pour tous ensembles A et B, on peut construire un ensemble C dont les seuls éléments sont A et B. Autrement dit, on permet ici de construire $C = \{A, B\}$ En formule,

$$\forall A \ \ \forall B \ \ \exists C \ \ [\forall x \ \ (x \in C) \Leftrightarrow (x = A \ \ \vee \ \ x = B)]$$

2.4.3 Axiome de la réunion (ou axiome de la somme)

Pour tout ensemble A, on peut construire un ensemble B dont les seuls éléments sont tous ceux qui sont éléments des éléments de A. Autrement dit, on permet ici la construction de l'ensemble :

$$B = \bigcup_{x \in A} x$$

En formule :

$$\forall A \ \ \exists B \ \ \forall C[C \in B \Leftrightarrow \exists D(D \in A \wedge C \in D)]$$

La clause placée entre parenthèses et faisant intervenir D sert à déclarer que C est élément d'un certain ensemble, lui-même élément de A.

2.4.4 Axiome de l'ensemble des parties

Pour tout ensemble A, on peut construire l'ensemble B des sous-ensembles de A. Autrement dit, on permet ici la construction de l'ensemble

$$B = \{ x \mid x \subseteq A \}$$

En formule :

$$\forall A \, \exists B \, \forall x \, (x \in B) \Leftrightarrow (x \subseteq A)$$

2.4.5 Axiome de l'infini

Il énonce qu'il existe un ensemble infini. Dans le langage formel de l'axiomatique de Zermelo-Fraenkel, l'axiome s'écrit :

$$\exists \, \omega \, (\emptyset \in \omega \wedge \forall \, x \, (\, x \in \omega \Rightarrow x \cup \{x\} \in \omega \,))$$

Ou en d'autres termes : il existe un ensemble ω ; tel que l'ensemble vide $\emptyset$ appartienne à ω et tel que toutes les fois où x est un élément de ω, l'ensemble formé en prenant l'union de x avec son singleton $\{x\}$ est également un élément de ω.

Pour comprendre cet axiome, appelons tout d'abord $x \cup \{x\}$ le successeur de x. Notons que l'axiome de la paire et l'Axiome d'extensionnalité nous permettent de construire le singleton $\{x\}$, et l'axiome de la réunion nous sert à former l'union. Les successeurs sont utilisés pour définir et coder les entiers dans la théorie des nombres entiers naturels. Dans le codage des entiers, zéro est l'ensemble vide ($0 = \emptyset$), et 1 est le successeur de 0 :

$$1 = 0 \bigcup \{0\} = \emptyset \bigcup \{\emptyset\} = \{\emptyset\} = \{0\}$$

De même, 2 est le successeur de 1 :

$$2 = 1 \bigcup \{1\} = \{0\} \bigcup \{1\} = \{\emptyset, \{\emptyset\}\} = \{0, 1\}$$

2.4.6 Schéma d'axiomes de compréhension (de séparation)

Pour tout ensemble A, on peut construire le sous-ensemble B des éléments de A qui satisfont une propriété P (exprimée dans le langage de la théorie des ensembles).

Autrement dit, on permet ici la construction de

$$B = \{x \in A \mid P(x)\}$$

En formule,

$$\forall y_1 \ldots \forall y_n \forall A \exists B \forall x[(x \in B) \Leftrightarrow (x \in A \wedge P(x; y_1; \ldots; y_n))$$

Les y_i sont ici simplement des paramètres auxiliaires dont on pourrait avoir besoin pour formuler plus facilement la propriété P. On dit qu'on a un ***schéma*** d'axiomes, parce qu'il y a un axiome pour chaque choix de P. On parle aussi de schéma de séparation, car il permet de séparer dans A les éléments qui vérifient la propriété P pour définir un nouvel ensemble.

Exemple : Dans les mathématiques usuelles, le schéma de compréhension est utilisé sous la forme de la notation en compréhension, par exemple on définira l'ensemble des nombres premiers comme :

$$\{p \in N \mid p > 1 \; et \; \forall x \in N \; [x divise p \Rightarrow (x = 1 ou x = p)]\}$$

2.4.7 Schéma d'axiomes de remplacement (ou substitution)

Pour tout ensemble A et toute relation fonctionnelle F, on a un ensemble

$$B := \{y \mid x \in A \; et \; F(x, y)\}$$

Pour exprimer ceci en formule (simplifiée), rappelons d'abord que F est une relation fonctionnelle, on écrit $Fonct(F)$ ssi

$$\forall x \; \forall y_1 \; \forall y_2 \; [(F(x, y_1) \; et \; F(x, y_2)) \Rightarrow (y_1 = y_2)]$$

Alors l'axiome se présente comme :

$$Fonct(F) \Rightarrow \forall A \; \forall B \; \forall y \; [y \in B \Leftrightarrow \exists x \; (x \in A \; et \; F(x, y)\;)]$$

Informellement, le schéma de remplacement énonce que, un ensemble étant donné, les images de ses éléments par une relation fonctionnelle forment un ensemble. L'intérêt majeur de l'axiome de remplacement, par rapport à l'axiome de compréhension (qui est néanmoins plus simple à mettre en oeuvre et couvre beaucoup de cas), c'est

qu'il permet de construire des ensembles qui ne sont pas forcément des sous-ensembles d'ensembles déjà identifiés (mais des images d'ensembles identifiés).

2.4.8 Axiomes de fondation

Pour tout ensemble A non vide, il existe un ensemble B, appartenant à A, qui n'a aucun élément en commun avec A, c'est-à-dire que

$$A \bigcap B = \emptyset$$

En formule,

$$\forall A[(A \neq \emptyset) et \exists B(B \in A et A \cap B = \emptyset)]$$

2.4.9 Axiomes du choix

Pour tout ensemble A, d'ensembles non vides, le produit cartésien des éléments de A est non vide. En formule,

$$[\forall\, x \in A(x \neq \emptyset)] \Rightarrow \prod_{x \in A} x \neq \emptyset)$$

2.5 Exercices

Exercice 1 : (Logique propositionnelle) Soit la formule P définie comme $(p \Rightarrow (q \Rightarrow r)) \Rightarrow (r \vee \neg p)$.

1. Donner la table de vérité de la formule P.
2. Est-ce que la formule P est valide, satisfiable, insatisfiable ?
3. La formule P a-t-elle un modèle ? si oui lequel ?
4. Donner la forme normale conjonctive et la forme normale disjonctive de la formule P.

Exercice 2 : (Barre de Sheffer) On définit le connecteur de **Sheffer** noté "|" (barre de Sheffer,ou NAND) par : $p \mid q \equiv \neg(p \wedge q)$

1. Donner la table de vérité de la formule $(p \mid q)$.
2. Donner la table de vérité de la formule $((p \mid q) \mid (p \mid q))$

3. On veut maintenant exprimer les différents connecteurs en utilisant la barre de Sheffer, et rien qu'elle.

   3.1 Donner la table de vérité de la formule $(p \mid p)$ et en déduire que le connecteur $\neg$ peut être défini en n'utilisant que la barre de Sheffer.

   3.2 Trouver des formules équivalentes à $p \wedge q$ et $p \vee q$, qui n'utilisent que la barre de Sheffer.

Exercice 3 : (Raisonnement) Soit P,Q et R trois propositions. Démontrer la propriété suivante par différentes méthodes (table de vérité, Raisonnement par hypothèse, Raisonnement par déduction et méthode de résolution) :

$$P \Rightarrow (Q \Rightarrow R) \Leftrightarrow (P \wedge Q) \Rightarrow R$$

Exercice 4 : (Logique des Prédicats) Représenter la phrase "Tout nombre entier naturel x a un successeur qui est inférieur ou égal à tout entier strictement supérieur à x" par une formule logique en utilisant les prédicats suivants :

— $entier(x)$: "x est un entier naturel"
— $successeur(x, y)$: "x est successeur de y"
— $inf(x, y)$: "x est inférieur ou égal à y"

Exercice 5 : (Logique des Prédicats) On modélise sommairement le système solaire par le modèle suivant :
Le domaine $\mathcal{D}$: contient les neuf planètes et le Soleil
Les constantes C : s_0 : le Soleil, m_1 : Mercure, v : Vénus, t : la Terre, l : la Lune, m_2 Mars, j : Jupiter ; s_1 : Saturne ; u : Uranus ; n : Neptune, p : Pluton. Cet ordre présente la proximité relative par rapport au Soleil : Pluton est la plus éloignée et Mercure la plus proche
Les tailles relatives des planètes : Lune < Mercure < Mars < Pluton < Vénus < Terre < Neptune < Uranus < Saturne < Jupiter.
On considère les prédicats suivants :

— $P(x)$: x est une planète (du système solaire) ;
— $T(x)$: x tourne autour de la Terre ;
— $M(x, y)$: x est plus petit (ou aussi grand) que y ;
— $S(x, y)$: x est plus proche (ou à égale distance) du Soleil que y ;

Traduire les phrases suivantes en logique des prédicats :

1. Vénus est une planète.
2. Le Soleil n'est pas une planète.
3. Le Soleil tourne autour de la Terre.
4. Certaines planètes sont plus petites que la Terre.
5. Toutes les planètes sont plus petites que Saturne.
6. Rien n'est plus petit que la Lune.
7. Mercure est la planète la plus proche du Soleil.
8. Mercure est la planète la plus proche du Soleil.
9. Mars est plus loin du Soleil que Pluton.
10. Si quelque chose est plus éloigné du Soleil que Pluton, alors ce n'est pas une planète.
11. Si le Soleil tourne autours de la Terre, alors il est plus petit que celle-ci.
12. S'il n'y a pas de planète plus grande que la Terre, alors la Terre est plus grande que Jupiter.
13. La Lune est une planète mais certaines choses ne sont pas des planètes.
14. Toutes les planètes ne tournent pas autour de la Terre.
15. Aucune planète n'est plus petite que Mercure.
16. Il n'y a pas de planète qui soit plus grande que la Terre tout en étant plus proche du Soleil qu'elle.
17. Il existe une planète telle que tout objet plus proche du Soleil qu'elle, est plus petit qu'elle.
18. Aucune planète n'est à la fois plus petite qu'Uranus et plus éloignée du Soleil qu'elle
19. Si toutes les planètes tournent autour de la Terre, alors Neptune aussi.

Exercice 6 : (Logique des Prédicats)

1. Quand dit-on qu'une variable est libre dans une formule ?

2. Considérant le langage du premier ordre $\mathcal{L} = \{R, S, f, a\}$ où R et S désignent deux symboles de prédicat respectivement unaire et binaire, f désigne un symbole de fonction unaire et a désigne un symbole de constante.
   Soit F la formule suivante :

$$(\forall x\ \exists y\ R(f(x), f(y))) \wedge ((\forall z\ (R(x, z))) \Rightarrow S(x))) \tag{2.1}$$

   Énumérez les termes qui apparaissent dans F.
3. Les variables x et y sont-elles libres dans la formule F ? Justifiez votre réponse.
4. Transformez la formule F précédente de manière à ce que variables liées et variables libres ne portent pas le même nom.
5. Donnez la formule $F[x/t]$ (c'est-à-dire la substitution de t à la variable x dans F) quand t est le terme $f(y)$
6. Donnez la formule $F[x/t]$ (c'est-à-dire la substitution de t à la variable x dans F) quand t est le terme $f(z)$.

Exercice 7 : (Logique des Prédicats) On considère le langage du premier ordre composé d'un symbole de fonction f d'arité 2, du symbole binaire de l'égalité $=$ et d'un symbole de prédicat R d'arité 2. Les variables sont notées $x, y, z \ldots$. Soit l'interprétation suivante :

— le domaine est $\mathbb{Z}$,
— l'interprétation de f, soit f_I, est l'addition sur $\mathbb{Z}$,
— l'interprétation de R, soit R_I, est la relation $<$,
— l'interprétation de $=$, soit $=_I$, est l'égalité sur $\mathbb{Z}$.

Quelle est la valeur de vérité de chacune des 3 formules ci-dessous dans cette interprétation ? (justifier votre réponse)

F_1 : $\forall x\ \exists z\ (f(z, y) = x)$

F_2 : $\exists x\ (R(x, y) \wedge R(y, f(x, x)))$

F_3 : $\forall x\ (R(x, y) \Rightarrow R(f(x, x), y))$

Chapitre 3

Langage Z : Notation

Sommaire

3.1 Introduction

Le langage Z a été développé à l'Université d'Oxford à la suite des travaux de Jean Raymond Abrial. C'est un langage formel qui utilise : la théorie des ensembles, le calcul des propositions ($\vee$,$\wedge$, $\neg$, $\Rightarrow$,...), le calcul des prédicats (les prédicats et les quantificateurs $\exists$ $\forall$), les relations (partie du produit cartésien de plusieurs ensembles), les fonctions et les séquences ou suites (fonctions des entiers naturels dans un autre ensemble pour imposer un ordre aux valeurs). Ce chapitre présente la notation de base de ces éléments en langage Z[1].

3.2 Langages des prédicats en langage Z

Une variable est définie en Z par un nom, son type (un ensemble de valeurs que peut prendre cette variable) et un espace où elle est définie, sa portée. L'introduction d'une nouvelle variable s'appelle la déclaration de variable. La portée d'une déclaration, dépend de l'endroit où se trouve cette déclaration dans la spécification Z. La syntaxe du calcul de prédicats en Z est la suivante :

— $P(x)$: Prédicat paramétré par la variable x, dite libre ;
— $\forall\, x \bullet P(x)$: x est une variable liée universellement, P est vrai pour toute valeur de x ;
— $\exists\, x \bullet P(x)$: x est une variable liée existentiellement, il y'a au moins une valeur de x pour laquelle le prédicat P soit vrai ;
— $\exists_! \, x \bullet P(x)$ ou $\exists_1 \, x \bullet P(x)$: x est liée par unicité, il y'a une seule valeur de x pour laquelle le prédicat P soit vrai.

Le symbole $\widehat{=}$ est l'égalité par définition dans Z, il est noté également ==. Le symbole $\bullet$ appelé boulet, est facultatif.

Exemple 3.2.1. *Voici quelques exemples de prédicats.*

1. La notation utilisée dans ce chapitre est basée sur le livre "Génie logiciel : Spécification des logiciels deux exemples de pratiques récentes Z et UML" de P. André et A. Vailly[3]

$Etudiant(x)$	*la variable x est libre*
$Etudiant(djamel)$	*est une proposition*
$Personne(djamel)$	*est une proposition*
$Jeune(Said)$	*est une proposition*
$\forall\, x \bullet Jeune(x) \Rightarrow Etudiant(x)$	*x est liée*
$Personne(S) \Rightarrow Jeune(J)$	*S est une constante*
$\forall\, x \bullet Personne(x) \Rightarrow Jeune(x) \vee Etudiant(x)$	*x est liée*

3.3 Les ensembles dans le langage Z

Ensemble Un ensemble est une collection bien définie d'objets (de valeurs, de termes), appelés éléments ou membres.
Le prédicat $x \in E$ désigne l'appartenance d'un élément x à un ensemble E. L'ensemble $\emptyset$ (noté aussi $\{\}$ est l'ensemble vide. Un ensemble contenant un seul élément est appelé singleton.

Type Un type est un ensemble de termes. En Z les types sont des ensembles disjoints.

Déclaration Une déclaration est l'association d'un type (un ensemble) à une variable telle que la variable peut prendre les valeurs (les termes) de ce type (déclaration de domaine de variable).
Les deux syntaxe $x : T$ et $\forall x \bullet x \in T$ sont équivalente, sous réserve que T soit un type.

Égalité Un même objet peut avoir différents noms (différents termes syntaxiques). Par exemple : $9, 5{+}4, \sqrt{81}, 3^2$ représentent la même valeur. Le prédicat $egal(a, b)$ teste l'égalité de valeur de deux termes a et b.

Égalité par définition L'égalité de deux termes x et y peut être fixée par définition. On le note $x \widehat{=} y$ ou encore $(x == y)$. On parle aussi d'abréviation. ($\widehat{=}$ et $==$ pour les définitions par compréhension)
Soient T un type, t_1 et t_2 des termes et x_i des variables. La syntaxe complète des déclarations et des prédicats est la suivantes :

$x : T$		Déclaration de la variable x de type T
$x_1 : T_1, x_2 : T_2, \dots, x_n : T_n$		Déclaration des variables x_i de type T_i
$x_1, x_2, \dots, x_n : T$	$\widehat{=}$	$x_1 : T, x_2 : T, \dots, x_n : T$
$t_1 = t_2$		égalité entre termes
$t_1 \neq t_2$	$\widehat{=}$	$\neg(t_1 = t_2)$ différence entre termes
$\forall x : T \mid P(x) \bullet Q(x)$	$\widehat{=}$	$(\forall x : T \bullet P(x) \Rightarrow Q(x))$
$\exists x : T \mid P(x) \bullet Q(x)$	$\widehat{=}$	$(\forall x : T \bullet P(x) \wedge Q(x))$

Une paire (ordonnée) d'éléments s'écrit (x, y). Un $n-uplet$ est la généralisation de la paire à une liste ordonnée d'éléments, on le note $(x_1, x_2, \dots, x_n)$. L'égalité sur les $n-uplets$ est l'égalité sur chaque élément :

$$((x_1, x_2, \dots, x_n) = (y_1, y_2, \dots, y_n)) \Leftrightarrow ((x_1 = y_1) \wedge (x_2 = y_2) \wedge \dots \wedge (x_n = y_n)$$

Déclarations globales toutes les variables sont typées en Z. Ce type est précisé dans une déclaration.

$$\mid x : \mathbb{Z}$$

Le caractère "|" est facultatif. Si cette déclaration se trouve telle quelle dans une spécification, alors elle déclare une variable x de portée globale et de type entier. Une déclaration contrainte par un prédicat s'écrit :

$$\frac{\mid x : \mathbb{Z}}{\mid 0 \leq x}$$

Cette déclaration peut être simplifiée[2] en

$$\mid x : \mathbb{N}$$

3.3.1 Déclaration des types et des ensembles

On distingue quatre sortes de types en Z

Les types de base : (given set) sont les ensembles dont les éléments ne sont connus. Par convention les types de base sont notés en majuscule.

2. même si le langage Z autorise de tels raccourcis, le type de la variable reste bien $\mathbb{Z}$

Les types prédéfinis : Sont des ensembles définis par Z, souvent infinis, dont les valeurs et les opérations sont implicites.

Les types libres : Sont des ensembles définis par énumération des éléments.

Les types défini : correspondent à des définitions (déclarations) d'ensembles ou de schémas[3].

Un ensemble est défini par :

— extension en énumérant ses éléments $S_1 \widehat{=} \{elt_1, elt_2, \dots, elt_n\}$

— compréhension par un prédicat $S_2 \widehat{=} \{x : T \mid P(x) \bullet m(x)\}$. S_2 est l'ensemble des termes de types T vérifiant le prédicat P et de forme (motif) m.

Parfois, chez certains auteurs, le type de base et types prédéfinis sont regrouper sous l'appellation *type de base*.
Ci-après la syntaxe et quelques exemples de types et d'ensembles :

$[FACULTE]$	définition de type de base FACULTE
$\mathbb{Z}$	ensemble prédéfini des entiers
$\mathbb{N}$	ensemble prédéfini des entiers naturels
	$\widehat{=}\{x : \mathbb{Z} \mid x \geq 0\}$ (définition par compréhension)
$\mathbb{N}^+$	ensemble prédéfini des entiers non nuls
$\mathbb{R}$	ensemble des réels
$n..m$	Intervalle d'entiers
	$\widehat{=}\{k : \mathbb{Z} \mid m \leq k \land k \leq n\}$
Bit ::= 0| 1	type libre Bit
$\emptyset$	ensemble vide $\widehat{=}\{\forall x : X \mid x \notin \emptyset\}$
nat_impair	$\widehat{=}\{x : \mathbb{Z} \mid x \geq 0 \bullet 2 * x + 1\}$
	définition des entiers naturels pairs en compréhension

La définition des types libres par constructions de type (produit et somme) est autorisée, par le langage Z, en permettant de définir des types récursifs par des équations. Exemple :

$$nat ::= zero \mid succ\langle\langle nat\rangle\rangle$$

3. à voir plus tard

La définition récursive des entiers naturels

$$arbreBinaire ::= Feuille\langle\langle E\rangle\rangle \mid Noeud\langle\langle arbreBinaire \times arbreBinaire\rangle\rangle$$

Où E (expression d'ensemble) est un ensemble dénombrable[4]
Des types mutuellement récursifs peuvent, également, être définis :

$$pair ::= zero \mid SuccImpair\langle\langle impair\rangle\rangle$$

$$impair ::= SuccPair\langle\langle pair\rangle\rangle$$

L'ensemble $\emptyset$ est **polymorphe** , i.e. il a plusieurs types $(\emptyset_{\mathbb{N}}, \emptyset_{\mathbb{R}}, \emptyset_{PERSONNE}, \emptyset_{FACULTE})$.
Les opérations entières prédéfinies sont entre autres :$+$, $-$, $*$, div, mod

3.3.1.1 Type Booléen

Le type booléen n'est pas défini dans Z, on peut le définir par un type libre[5] (Les mots-clés true et false sont réservés)

$$bool ::= vrai \mid false$$

3.3.2 Propriétés

Soit X et Y deux ensembles, x et y deux variables et t un motif :

Appartenance :	$\forall x \bullet x \in \{x\}$
Ensemble vide :	$\neg\, \exists x \bullet \in \{\}$
Extension :	$(X = Y) \Leftrightarrow (\forall x \bullet x \in X \Leftrightarrow x \in Y)$
Prédicat :	$x \in \{y : X \mid P(y)\} \Leftrightarrow (x \in X \wedge P(x))$
Motif :	$x \in \{y : X \bullet t(y)\} \Leftrightarrow (\exists y \bullet y \in X \wedge x = t(y))$

3.3.3 Opérations ensemblistes

A l'instar des prédicats, les ensembles ont des opérateurs. Il est possible de définir des ensembles d'ensembles avec l'opérateur $\mathbb{P}$ (puissance) ou des ensembles des n−uplets d'éléments (produit cartésien d'ensembles) par l'opérateur $\times$. Soit T, S et X

4. En mathématique un ensemble est dit dénombrable lorsque ses éléments peuvent être listés sans omission ni répétition dans une suite indexée par les entiers.
5. On peut utiliser ce type dans les fonctions.

trois ensembles, Soit x, y et z des variables. Ci-dessous les opérateurs sur les ensembles quelques propriétés :

$$
\begin{aligned}
\#S &\widehat{=} |S| : \text{cardinal, nombre d'éléments;} \\
x \notin S &\quad \neg(x \in S) : \text{non appartenance;} \\
S \subseteq T &\widehat{=} \forall x : S \bullet x \in T) : \text{inclusion;} \\
S \subset T &\widehat{=} (S \subseteq T \wedge S \neq T) : \text{inclusion stricte;} \\
\mathbb{P}\, S &\quad \text{puissance, partie de } S; \\
\mathbb{F}\, S &\widehat{=} \{t : \mathbb{P}\, S | fini(T)\} : \text{ensemble de sous-ensemble finis;} \\
S \times T &\widehat{=} \{x : S; y : T \bullet (x, y)\} : \text{produit cartisien;} \\
S \cap Y &\widehat{=} \{x : X | x \in S \wedge x \in T\} : \text{intersection;} \\
S \cup Y &\widehat{=} \{x : X | x \in S \vee x \in T\} : \text{union;} \\
S \setminus Y &\widehat{=} \{x : X | x \in S \wedge x \notin T\} \widehat{=} S - T : \text{différence;} \\
min\ S &\quad min\ S \in S \wedge (\forall x : S \bullet x \geq min\ S) \text{ avec } S : \mathbb{FN} | S \neq \emptyset : \text{minimum ;} \\
max\ S &\quad max\ S \in S \wedge (\forall x : S \bullet x \leq max\ S) \text{ avec } S : \mathbb{FN} | S \neq \emptyset : \text{maximum ;} \\
S \in \mathbb{P}T &\Leftrightarrow \forall x \bullet x \in S \Rightarrow x \in T : \text{puissance;} \\
x \in (S \times T) &\Leftrightarrow \exists y, z \bullet (y \in S \wedge z \in T \wedge x = (y, z)) : \text{produit cartésien.}
\end{aligned}
$$

Exemples :

— $ens \widehat{=} \{1, 2, 3\}$
— $alpha ::= a|b$
— $1 \in ens$
— $4 \notin ens$
— $\#ens = 3$
— $ens \subseteq \mathbb{N}$ et $ens \subset \mathbb{N}$
— $\mathbb{P}\, ens = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$
— $ens \times alpha = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}$
— $ens \bigcup \mathbb{N} = \mathbb{N}$
— $ens \bigcap \mathbb{N} = ens$

La théorie des ensembles est la base formelle des description en langage Z, et le calcul des prédicats est la base formelle des preuves.

3.4 Les relations

Les relations sont un élément clé de la modélisation de données avec Z, elle sont une abstraction au niveau des ensembles des liaisons existants entre les éléments.

Définition 3.4.1. *Voici les définitions de base sur les relations :*

- Lien : *Une paire d'éléments indique un lien entre ces deux éléments.*
- Le produit cartésien : *est l'abstraction des liens entre éléments au niveau de leurs ensembles d'appartenance (de leur type). Il désigne tous les liens possibles entre éléments de deux ensembles.*
- Relation : *la relation est un ensemble de liens entre éléments, c'est un sous ensemble du produit cartésien. Formellement, pour deux ensembles X et Y, l'ensemble des relations entre X et Y est définie par $X \leftrightarrow Y \widehat{=} \mathbb{P}(X \times Y)$.*
  Si le produit cartésien se généralise pour les $n-$uplets, ce n'est pas le cas aux relations qui restent binaire. De même que les paires sont ordonnées, la relation est ordonnée de l'ensemble de départ vers l'ensemble d'arrivée. Un élément de la relation est appelé lien, il s'écrit $x \mapsto y$, x est l'antécédent et y est l'image.
- Domaine et codomaine d'une relation : *le domaine d'une relation est l'ensemble des éléments de l'ensemble du départ qui ont un lien. C'est l'ensemble des antécédents. Le codomaine (range en anglais) d'une relation est l'ensemble des éléments de l'ensemble d'arrivée qui ont un lien par la relation. C'est l'ensemble des images.*

3.4.1 Opérations relationnelles

Soit X, Y et Z des ensembles ; $x : X$, $y : Y$ et $R : X \leftrightarrow Y$.

$$
\begin{array}{rcl}
x \mapsto y & \widehat{=} & (x, y) \text{ paire d'une relation ;} \\
x\ R\ y & \widehat{=} & (x, y) \in R \widehat{=} x \underline{R} y \text{ membre de la relation } R ; \\
dom\ R & \widehat{=} & \{x : X | (\exists y : Y \bullet xRy)\} \text{ domaine de la relation } R ; \\
ran\ R & \widehat{=} & \{y : Y | (\exists x : X \bullet xRy)\} \text{ codomaine de la relation } R ; \\
id\ R & \widehat{=} & \{x : X \bullet x \mapsto x\} \text{ identité ;} \\
R^{-1} & \widehat{=} & \{x : X, y : Y \mid xRy \bullet y \mapsto x\} \widehat{=} R^{\sim} \text{ inverse de } R ; \\
R ; \acute{R} & \widehat{=} & \{x : X, z : Z \mid (\exists y : Y \bullet xRy \wedge y\acute{R}z)\} \text{ avec } \acute{R} : Y \leftrightarrow Z \\
 & & \text{composition directe ;} \\
R \circ \acute{R} & \widehat{=} & \acute{R} ; R \text{ composition ;} \\
R^k & & \text{composition récurrente } R : X \leftrightarrow X\ ,\ R^0 \widehat{=} id\ R\ ,\ R^{k+1} \widehat{=} R^k \circ R ; \\
R^* & \widehat{=} & \bigcup \{n : \mathbb{N} \bullet R^n\} \text{ fermeture réflexive et transitive ;} \\
R^+ & \widehat{=} & \bigcup \{n : \mathbb{N}^+ \bullet R^n\} \text{ avec } R : X \leftrightarrow X \text{ fermeture non-réflexive et transitive ;} \\
R(\!|S|\!) & \widehat{=} & \{y : Y | (\exists x : S \bullet xRy)\} \text{ image relationnelle ;} \\
S \lhd R & \widehat{=} & \{x : X, y : Y | (x \in S \wedge xRy)\} \text{ restriction de domaine ;} \\
R \rhd S & \widehat{=} & \{x : X, y : Y | (y \in S \wedge xRy)\} \text{restriction de codomaine avec } S : \mathbb{P}X ; \\
S \lhd\!\!\!- R & \widehat{=} & (X \setminus S) \lhd R \text{ soustraction de domaine, avec} S : \mathbb{P}X ; \\
R \rhd\!\!\!- S & \widehat{=} & R \rhd (Y \setminus S) \text{ soustraction de codomaine, avec} S : \mathbb{P}Y ; \\
R \oplus \acute{R} & \widehat{=} & (dom\ \acute{R} \lhd\!\!\!- R) \bigcup \acute{R} \text{ surcharge, avec } \acute{R} : X \leftrightarrow Y ;
\end{array}
$$

Les relations (globales) de la spécification peuvent s'écrire sous forme axiomatique. Le caractère "_" désigne l'emplacement des arguments (notation infixée) :

$$
\begin{array}{|l}
\_Divise\_ : \mathbb{N} \leftrightarrow \mathbb{N} \\
\hline
\forall x, y : \mathbb{N} \bullet x\ Divise\ y \Leftrightarrow \exists z : \mathbb{N} \bullet x \times z = y
\end{array}
$$

Des définitions génériques sont possible sous la forme suivante[6] :

$$[X, Y]$$
$$first : X \times Y \to X$$
$$\forall x : X;\ y : Y \bullet first(x, y) = x$$

$$[X]$$
$$\_ \subseteq \_ : \mathbb{P}\, X \leftrightarrow \mathbb{P}\, X$$
$$\forall S, T : \mathbb{P}\, X \bullet S \subseteq T \Leftrightarrow \forall x : X \bullet x \in S \Rightarrow x \in T$$

3.4.2 Les propriétés des relations

Soit $R : X \leftrightarrow X$ une relation homogène (i.e. d'un ensemble dans lui-même). Les propriétés de base des relations sont les suivantes :

réflexivité	$reflexive(R) \Leftrightarrow \forall x : X \bullet xRx$;
non-réflexivité	$irreflexive(R) \Leftrightarrow \neg(\exists x \bullet xRx)$;
symétrie	$symmetric(R) \Leftrightarrow \forall x, y : X \bullet xRy \Rightarrow yRx$;
anti-symétrie	$antisymmetric(R) \Leftrightarrow (\forall x, y : X \bullet xRy \wedge yRx \Rightarrow x = y)$;
asymétrie	$asymmetric(R) \Leftrightarrow \forall x, y : X \bullet xRy \Rightarrow \neg(yRx)$
transitivité	$tansitive(R) \Leftrightarrow \forall x, y, z : X \bullet xRy \wedge yRz \Rightarrow xRz$
équivalence	$equivalence(R) \Leftrightarrow reflexive(R) \wedge symmetric(R) \wedge tarnsitive(R)$;
ordre partiel	$partial\_ordre(R) \Leftrightarrow reflexive(R) \wedge antisymmetric(R) \wedge tarnsitive(R)$;
ordre total	$total\_ordre(R) \Leftrightarrow partial\_ordre(R) \wedge (\forall x, y : X \bullet xRy \vee yRx)$;

6. Le paramètre générique est placé entre []. La définition est applicable quel soit le type X et/ou Y.

3.5 Les fonctions

Les fonctions en Z est une relation déterministe qui correspond à la notion d'application en mathématiques. Cette notion est appréhendée via les relations. plus précisément, une fonction est un cas particulier de relation telle qu'à un antécédent est associée (liée) ou plus une image. L'application d'une fonction f à une variable x s'écrit $f\ x$ ou $f(x)$.

Soit X et Y deux ensembles et $x : X$ et $y : Y$ deux variables, nous illustrons ci-dessous les familles de fonctions en Z :

Fonctions partielles	$X \pfun Y \mathrel{\widehat=} \{f : X \rel Y \mid (\forall x : dom\ f \bullet (\exists! y : Y \bullet x\ f\ y\}$
Fonctions totales	$X \fun Y \mathrel{\widehat=} \{f : X \pfun Y \mid dom\ f = X\}$
Injections	$X \pinj Y \mathrel{\widehat=} \{f : X \pfun Y \mid (\forall x_1, x_2 : dom\ f \bullet f(x_1) = f(x_2) \Rightarrow x_1 = x_2)\}$
Injections totales	$X \inj Y \mathrel{\widehat=} \{f : X \pinj Y \mid dom\ f = X\} \mathrel{\widehat=} (X \pinj Y) \bigcap (X \fun Y)$
Surjections	$X \psur Y \mathrel{\widehat=} \{f : X \pfun Y \mid ran\ f = Y\}$
Surjections totales	$X \surj Y \mathrel{\widehat=} \{f : X \psur Y \mid dom\ f = X\} \mathrel{\widehat=} (X \psur Y) \bigcap (X \fun Y)$
Bijections	$X \bij Y \mathrel{\widehat=} (X \psur Y) \bigcap (X \pinj Y)$

Bijections totales	$X \rightarrowtail\!\!\!\!\!\!\rightarrow Y \mathrel{\widehat{=}} (X \rightarrowtail\!\!\!\!\!\!\rightarrow Y) \bigcap (X \rightarrowtail Y)$

Une fonction interne est une fonction dont le type des arguments et du résultat est le même i.e $f : X \times X \rightarrow X$. En mathématique on parle de loi de composition interne.
Exemple : calcul de la moyenne des étudiants

$$
\begin{array}{|l}
som\_ : \mathbb{P}\,\mathbb{N} \leftrightarrow \mathbb{N} \\
moy\_ : \mathbb{P}\,\mathbb{N} \leftrightarrow \mathbb{N} \\
\hline
som\emptyset = 0 \\
\forall\, s : \mathbb{P}\,\mathbb{N} \mid s \neq \emptyset \bullet (\exists\, x : \mathbb{N} \mid x \in s \bullet soms = x + som(s \setminus \{x\})) \\
moy\emptyset = 0 \\
\forall\, s : \mathbb{P}\,\mathbb{N} \mid s \neq \emptyset \bullet moy\ s = (som\ s)\ div\ \#s
\end{array}
$$

$$
\begin{array}{l}
moyenne : ETUDIANT \nrightarrow 0..20 \\
dom\ moyenne = dom\ notes \\
\forall\, e : ETUDIANT \mid e \in dom\ notes \bullet moyenne\ e = moy(notes(\!|\ e\ |\!))
\end{array}
$$

3.6 Les séquences

La séquence[7] introduit une notation d'ordre des éléments dans un multi-ensemble[8]. Autrement dit, La séquence est un ensemble ordonné. Et formellement :

$$Seq X \mathrel{\widehat{=}} \{f : \mathbb{N} \nrightarrow X \mid \exists n : \mathbb{N} \bullet dom f = 1 \ldots n\}$$

Une Séquence définie en extension est notée $\langle x_1, x_2, \ldots x_n \rangle$, ou d'une manière équivalente $[x_1, x_2, \ldots x_n]$. La notation alternative est donnée par : $\{1 \mapsto x_1, 2 \mapsto x_2, \ldots n \mapsto x_n\}$ avec $x_1, x_2, \ldots, x_n : X$. Une séquence vide est notée $\langle\,\rangle$, [ ] ou { }. La taille (le cardinal) d'une séquence "s" est donnée par la fonction "$\#s$". Le $i^{ème}$ élément d'une séquence "s" est indexé par "$s\ i$", avec $0 < i \leq \#s$.

7. La séquence Z est aussi appelée chaîne, liste, trace ou suite dans d'autre langages formels.
8. Un multi-ensemble (parfois appelé sac, de l'anglais bag utilisé comme synonyme de multi-set) est une sorte d'ensemble dans lequel chaque élément peut apparaître plusieurs fois.

Nous présentons ci-dessous les principales opérations sur les séquences :

$\frown$: Concaténation
rev :Inversion ou retournement

$$
\begin{array}{|l}
[X] \\
\hline
\_ \frown \_ : \operatorname{seq} X \times \operatorname{seq} X \longrightarrow \operatorname{seq} X \\
rev : \operatorname{seq} X \longrightarrow \operatorname{seq} X \\
\hline
\forall\, s, t : \operatorname{seq} X \bullet \\
\quad s \frown t = s \cup \{\, n : \operatorname{dom} t \bullet n + \#s \mapsto t(n) \,\} \\
\forall\, s : \operatorname{seq} X \bullet \\
\quad rev\ s = (\lambda\, n : \operatorname{dom} s \bullet s(\#s - n + 1)) \\
\hline
\end{array}
$$

Propriétés de $\frown$ et de *rev*

$$
\begin{array}{ll}
(s \frown t) \frown u = s \frown (t \frown u) & rev\ \langle\rangle = \langle\rangle \\
\langle\rangle \frown s = s & rev\ \langle x \rangle = \langle x \rangle \\
s \frown \langle\rangle = s & rev(s \frown t) = (rev\ t) \frown (rev\ s) \\
\#(s \frown t) = \#s + \#t & rev(rev\ s) = s
\end{array}
$$

head : le premier élément, *last* : le dernier élément,
tail : liste sans le premier élément, $front$: liste sans le dernier élément

$$
\begin{array}{|l}
[X] \\
\hline
head, last : \operatorname{seq}_1 X \longrightarrow X \\
tail, front : \operatorname{seq}_1 X \longrightarrow \operatorname{seq} X \\
\hline
\forall\, s : \operatorname{seq}_1 X \bullet \\
\quad head\ s = s(1) \land \\
\quad last\ s = s(\#s) \land \\
\quad tail\ s = (\lambda\, n : 1 \,..\, \#s - 1 \bullet s(n+1)) \land \\
\quad front\ s = (1 \,..\, \#s - 1) \lhd s \\
\hline
\end{array}
$$

Propriétés de *head*, *last*, *tail* et $front$:

$$head\,\langle x\rangle = last\,\langle x\rangle = x \qquad tail\,\langle x\rangle = front\,\langle x\rangle = \langle\rangle$$

$$s \neq \langle\rangle \Rightarrow \quad head(s \frown t) = head\,s \wedge \quad tail(s \frown t) = (tail\,s) \frown t$$

$$t \neq \langle\rangle \Rightarrow \quad last(s \frown t) = last\,t \wedge \quad front(s \frown t) = s \frown (front\,t)$$

$$s \neq \langle\rangle \Rightarrow \langle head\,s\rangle \frown (tail\,s) = s \qquad s \neq \langle\rangle \Rightarrow (front\,s) \frown \langle last\,s\rangle = s$$

$$s \neq \langle\rangle \Rightarrow head(rev\,s) = last\,s \wedge tail(rev\,s) = rev(front\,s)$$
$$s \neq \langle\rangle \Rightarrow last(rev\,s) = head\,s \wedge front(rev\,s) = rev(tail\,s)$$

$\restriction$: extraction, $\upharpoonright$: filtrage et *squash* : compactage

$$[X]$$
$$\_ \restriction \_ : \mathbb{P}\,\mathbb{N}_1 \times \operatorname{seq} X \longrightarrow \operatorname{seq} X$$
$$\_ \upharpoonright \_ : \operatorname{seq} X \times \mathbb{P}\,X \longrightarrow \operatorname{seq} X$$
$$squash : (\mathbb{N}_1 \pfun X) \longrightarrow \operatorname{seq} X$$
$$\forall\, U : \mathbb{P}\,\mathbb{N}_1;\ s : \operatorname{seq} X \bullet \quad U \restriction s = squash\,(U \lhd s)$$
$$\forall\, s : \operatorname{seq} X;\ V : \mathbb{P}\,X \bullet \quad s \upharpoonright V = squash\,(s \rhd V)$$
$$\forall\, f : \mathbb{N}_1 \pfun X \bullet \quad squash\,f = f \circ (\mu\, p : 1 \mathinner{..} \#f \rightarrowtail \operatorname{dom} f \mid p \circ succ \circ p^{\sim} \subseteq (\_ < \_))$$

prefix : préfixe, *suffix* : suffixe et *in* : « dans »

$$[X]$$
$$\_ \text{ prefix } \_, \_ \text{ suffix } \_, \_ \text{ in } \_ : \operatorname{seq} X \leftrightarrow \operatorname{seq} X$$
$$\forall\, s, t : \operatorname{seq} X \bullet$$
$$s \text{ prefix } t \Leftrightarrow (\exists\, v : \operatorname{seq} X \bullet s \frown v = t) \wedge$$
$$s \text{ suffix } t \Leftrightarrow (\exists\, u : \operatorname{seq} X \bullet u \frown s = t) \wedge$$
$$s \text{ in } t \Leftrightarrow (\exists\, u, v : \operatorname{seq} X \bullet u \frown s \frown v = t)$$

disjoint Disjonction d'ensembles et *partition* : Partitionnement d'ensembles

$$[I, X]$$
$$\text{disjoint } \_ : \mathbb{P}(I \pfun \mathbb{P}\,X)$$
$$\_ \text{ partition } \_ : (I \pfun \mathbb{P}\,X) \leftrightarrow \mathbb{P}\,X$$
$$\forall\, S : I \pfun \mathbb{P}\,X;\ T : \mathbb{P}\,X \bullet$$
$$(\text{disjoint } S \Leftrightarrow (\forall\, i, j : \operatorname{dom} S \mid i \neq j \bullet S(i) \cap S(j) = \varnothing)) \wedge$$
$$(S \text{ partition } T \Leftrightarrow \text{disjoint } S \wedge \bigcup \{\, i : \operatorname{dom} S \bullet S(i) \,\} = T)$$

Exemples :

$s = \langle a, b, c\rangle, t = \langle d, e\rangle$
$s = \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}, t = \{1 \mapsto d, 2 \mapsto e\}$
$\#s = 3, \#t = 2$
$s \frown t = \langle a, b, c, d, e\rangle$
$rev\ t = \langle e, d\rangle$
$last\ s = c, tail\ s = \langle b, c\rangle, head\ s = a, front\ s = \langle a, b\rangle$

$squash\{2 \mapsto a, 12 \mapsto c, 5 \mapsto b\} = \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\} = \langle a, b, c\rangle$
$\langle e\rangle prefix \langle e, d\rangle, \langle d\rangle suffix \langle e, d\rangle, \langle b, c\rangle in \langle a, b, c, d, e\rangle$
$\langle b, c\rangle in \langle b, c, d, e\rangle$?
$\langle d, e\rangle in \langle b, c, d, e\rangle$?
$disjoint\langle s, t\rangle = true$
$disjoint\langle\langle a, b\rangle, \langle b, c, d\rangle\rangle = false$
$\langle s, t\rangle partition \{a, b, c, d, e\} = true$

3.7 Exercices

Exercice 1 :
Soit les ensembles suivants (définis en extension) :

$$A \widehat{=} \{1, 3, 5, 6\}; B \widehat{=} \{a, b, c\}; C \widehat{=} \{3, 15\}; D \widehat{=} \mathbb{P}B;$$

Donner la définition en extension des ensembles suivants :

$$\mathbb{P}A; A \times B; A \setminus C; \bigcup D; \bigcap D$$

Donner la valeur de $\#D$.

Exercice 2 :
Définir l'ensemble des entiers positifs, les ensembles des carrés et des cubes d'entiers naturels, et les ensembles des entiers naturels pairs et impairs.

Exercice 3 :
Donner une manière d'introduire une variable de type T qui soit un élément de d'un

ensemble U, avec U un sous-ensemble de T.

Exercice 4 :
Les déclarations suivantes sont-elles correctes?

1. $[NAME, REAL, CLIENT\_ID, BODY]$
2. $x : PERSONNE$
3. $x : \mathbb{N}$
4. $machine : NAME$
5. $m : machine$
6. $machines : \mathbb{P}(NAME \times NAME)$
7. $machine : ID \rightarrow NAME$
8. la définition axiomatique $|x : \mathbb{Z}$

   $$\frac{}{|x \leq 0}$$
9. $prix : machines \times REAL$
10. $ALPHA ::= a|b|c| \ldots |z$

Exercice 5 :
Donner les prédicats en langage Z correspondant aux affirmation suivantes :

1. Les stylos sont des crayons.
2. Les crayons coulent, de même que les stylos à encre.
3. Tous les étudiants sont des adultes.
4. Tous les maisons ont un toit et un mur.
5. Toute personne est née dans une commune donnée, à une date donné et possède une qualification.
6. Tous les étudiants ont cours au moins cinq fois par semaine.
7. Toute commande est facturée.
8. Si un étudiant est surpris à tricher à un examen, il est exclu de l'épreuve.

Chapitre 4

Langage Z : La Spécification

Sommaire

4.1 Introduction

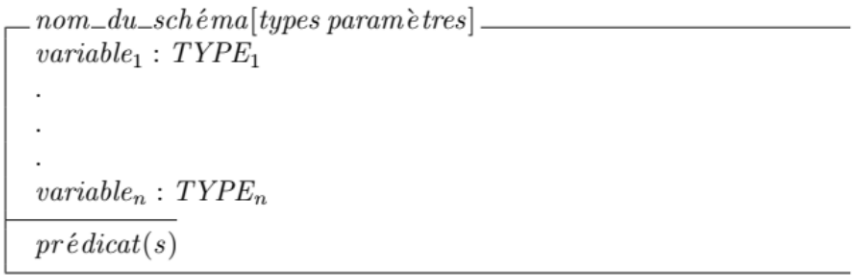
Après avoir présenter les notations de base du langage Z, ce chapitre présente la structure fondamentale, l'organisation et la démarche de la spécification en langage Z.

4.2 Structuration de la spécification

La structure fondamentale de modélisation de systèmes séquentiels en Z c'est le schéma. Un schéma est une déclaration de variables locales et de prédicats portant sur ces variables. L'intérêt des schémas vient du calcul associé et de l'interprétation qui en est faite pour la modélisation des systèmes.

4.2.1 schéma

Un schéma est utiliser pour donner un nom à un groupe de déclaration et de prédicats. Il est présenté sous forme de boite (on parle aussi de format vertical) :



Où : Nom_du_schéma : portée globale,
et Variable 1 ... n : portée locale au schéma.

Exemple : Modélisation d'une classe d'étudiants.
Les étudiants sont spécifiés par un type de base.

$[ETUDIANT]$

$$\begin{array}{|l}
\textit{Classe} \\
\hline
effectif_max : \mathbb{N} \\
eleves : \mathbb{P}\, ETUDIANT \\
\hline
\#eleves \leq effectif_max \\
\hline
\end{array}$$

Une définition d'un schéma générique d'une classe est donnée par :

$$\begin{array}{|l}
\textit{Classe}[X] \\
\hline
effectif_max : \mathbb{N} \\
eleves : \mathbb{P}\, X \\
\hline
\#eleves \leq effectif_max \\
\hline
\end{array}$$

Un schéma peut être utilisé à la place d'une déclaration, à titre d'exemple :

$$nat\_pair \mathrel{\widehat=} \{x : \mathbb{Z} \mid x \geq 0 \bullet 2 * x\}$$

$$\begin{array}{|l}
nat_pair \\
\hline
x : \mathbb{Z} \\
\hline
x \geq 0 \land x mod 2 = 0 \\
\hline
\end{array}$$

Ici, le type (ou l'ensemble) des entiers naturels pairs est donné en compréhension et par un schéma.

Deux types schéma sont équivalents s'ils ont les mêmes variables, la partie prédicative n'est pas considérée.

$[JOUEUR]$

$$\begin{array}{|l}
\textit{Titulaire} \\
\hline
effectif_max : \mathbb{N} \\
joueurs : \mathbb{P}\, JOUEUR \\
\hline
\#joueurs \leq (effectif_max \; div \; 2) \\
\hline
\end{array}$$

$[JOUEUR]$

$$\begin{array}{|l}
\textit{Equipe} \\
\hline
effectif_max : \mathbb{N} \\
joueurs : \mathbb{P}\, JOUEUR \\
\hline
\#joueurs \leq effectif_max \\
\hline
\end{array}$$

Le schéma s'ajoute aux autres formes de déclarations (simples, axiomatiques ou

génériques) de types, constantes, ensembles ou fonctions. Ainsi, on peut définir une instance d'un équipe (du schéma précédent) par $eq : Equipe$. La notation pointée $eq.joueurs$ désigne les $joueurs$ de la classe eq, on écrite aussi $joueurs(eq)$.
Les instances (valeurs) d'un type schéma sont appelées liens ou liaisons. Une liaison est une affectation de valeurs aux variables du schéma, qui vérifie la partie prédicative du schéma. C'est une sorte de n-uplet dont les valeurs sont ordonnées non pas par des positions mais par des noms. Une liaison quelconque d'un schéma S et notée θS.

$[JOUEUR]$

$$\begin{array}{|l} Equipe \\ effectif\_max : \mathbb{N} \\ joueurs : \mathbb{P}\, JOUEUR \\ \hline \#joueurs \leq effectif\_max \end{array}$$

$$\begin{array}{|l} effectif : Equipe \to \mathbb{Z} \\ \hline \forall\, Equipe \bullet effectif(\theta Equipe) = \#joueurs \end{array}$$

Un schéma peut être inclus dans un autre schéma (écriture modulaire) . L'inclusion signifie la fusion des déclarations et la conjonction des prédicats. Par exemple :

$CHAR ::= A \mid B \mid C \mid D$

$$\begin{array}{|l} Joueur \\ nom : seqCHAR \\ num : \mathbb{N}_1 \\ \hline \#nom \leq 15 \end{array}$$

$$\begin{array}{|l} JoueurTitulaire \\ Joueur \\ adresse : seqCHAR \\ age : \mathbb{N}_1 \\ \hline age \leq 45 \land age \geq 17 \end{array}$$

$\Rightarrow$

$$\begin{array}{|l} JoueurTitulaire \\ nom : seqCHAR \\ num : \mathbb{N}_1 \\ adresse : seqCHAR \\ age : \mathbb{N}_1 \\ \hline \#nom \leq 15 \land age \leq 45 \land age \geq 17 \end{array}$$

Une décoration est un suffixage de variables par une marque de ponctuation (‘ ,? ,!). Un schéma peut être « décoré ». La décoration d'un schéma est celle de toutes ses variables. Le sens de la décoration dépend du contexte de l'utilisation. Si S est un schéma, S' est le schéma dans lequel les variables sont décoré par } dans la déclaration et le prédicat.

$$
\begin{array}{|l}
\hline
JoueurTitulaire \\
nom : seqCHAR \\
num : \mathbb{N}_1 \\
adresse : seqCHAR \\
age : \mathbb{N}_1 \\
\hline
\#nom \leq 15 \land age \leq 45 \land age \geq 17 \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
JoueurTitulaire' \\
nom' : seqCHAR \\
num' : \mathbb{N}_1 \\
adresse' : seqCHAR \\
age' : \mathbb{N}_1 \\
\hline
\#nom' \leq 15 \land age' \leq 45 \land age' \geq 17 \\
\hline
\end{array}
$$

La notation $\delta SCHEMA$ est un raccourci pour définir l'inclusion d'un schéma $SCHEMA$ et du schéma décoré $SCHEMA'$. On utilise la notation $\delta SCHEMA$ pour définir une opération qui modifie l'état du système. Les variables de $SCHEMA$ (resp. $SCHEMA'$) représentent l'état avant (resp. après) l'opération.

$$
\begin{array}{|l}
\hline
\Delta JoueurTitulaire \\
JoueurTitulaire \\
JoueurTitulaire' \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\Delta JoueurTitulaire \\
nom : seqCHAR \\
nom' : seqCHAR \\
num : \mathbb{N}_1 \\
num' : \mathbb{N}_1 \\
adresse : seqCHAR \\
adresse' : seqCHAR \\
age : \mathbb{N}_1 age' : \mathbb{N}_1 \\
\hline
\#nom \leq 15 \land age \leq 45 \land age \geq 17 \land \\
\#nom' \leq 15 \land age' \leq 45 \land age' \geq 17 \\
\hline
\end{array}
$$

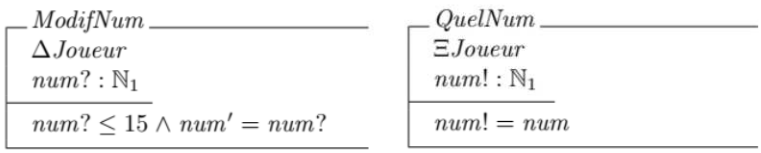
Le raccourci $\Xi SCHEMA$ est le schéma invariant : les nouvelles valeurs des variables sont égales aux anciennes valeurs.

$$
\begin{array}{|l}
\hline
Joueur \\
nom : seqCHAR \\
num : \mathbb{N}_1 \\
\hline
\#nom \leq 15 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\Xi Joueur \\
\Delta Joueur \\
\hline
\theta Joueur = \theta Joueur' \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\Xi Joueur \\
nom : seqCHAR \\
num : \mathbb{N}_1 \\
nom' : seqCHAR \\
num' : \mathbb{N}_1 \\
\hline
\#nom \leq 15 \land \#nom' \leq 15 \\
\hline
\end{array}
$$

La décoration « ? » est utilisée pour les variables en paramètre d'entrées d'opérations, et la décoration « ! » pour la sorties (résultats de l'opération).



$\Delta joueur$ introduit la notion de l'état : état avant opération (schéma $Joueur$) et après opération (schéma $Joueur'$). Le prédicat $num \leq 15$ du schéma $Joueur$ est gardée pour respecter les contraintes. La notation $\Xi joueur$ est utilisée quand aucune variable ne change.

4.3 Calcul des schémas

Cette section présente les opérations utilisées pour le calcul des schémas :

$tuple\ S$	la partie déclarative de S ;
$pred\ S$	la partie prédictive de S ;
$S \mid P$	ajout de prédicat P à S $[tuple\ S \mid pred\ S \wedge P]$;
$S; D$	déclarations jointes $[tuple\ S; D \mid pred\ S]$;
$\neg\ S$	négation du schéma S : $[tuple\ S \mid \neg(pred\ S)]$;
$S \wedge T$	conjonction de schémas $[tuple\ S; tuple\ T \mid pred\ S\ \wedge\ pred\ T]$;
$S \vee T$	disjonction de schémas $[tuple\ S; tuple\ T \mid pred\ S\ \vee\ pred\ T]$;
$S \Rightarrow T$	implication de schémas $[tuple\ S; tuple\ T \mid pred\ S\ \Rightarrow\ pred\ T]$;
$S \Leftrightarrow T$	équivalence de schémas $[tuple\ S; tuple\ T \mid pred\ S\ \Leftrightarrow\ pred\ T]$;
$\forall S$	quantificateur universel de schéma ;
$\exists S$	quantificateur existentiel de schéma ;
$\exists_1 S$	quantificateur existentiel et l'unicité de schéma ;
$\setminus$	masquage : $S \setminus (v_1, v_2, \ldots, v_n)$: déclaration de S sans celles des variables concernées $(v_1, v_2, \ldots, v_n)$, liées par $\exists$ dans $pred\ S$;
$\upharpoonright$	projection : inverse du masquage, i.e. uniquement les variables nommées. pré-condition, composition, surcharge et tubage sont données par :

$$\text{pre } S \mathrel{\widehat{=}} (\exists\, state';\ y! : Y \bullet S)\ (pr\acute{e}condition)$$
$$S \fatsemi T (composition)\ Exemple : ModifNom \fatsemi QuelNom$$
$$S \oplus T \mathrel{\widehat{=}} (S \wedge \neg \text{ pre } T) \vee T (surcharge)$$
$$S >> T\ (\ tubage\ :\ les\ sorties\ de\ S\ deviennent\ les\ entr\acute{e}es\ de\ T)$$

4.4 La démarche de la spécification

A l'instar des méthodes formelles, la démarche de la spécification en Z suit le cycle linéaire : Analyse préliminaire, Spécification formelle, Conception abstraite et Réalisation.

On décrit un système par un état et des opérations qui permettent de lire ou de modifier cet état. Z décrit alors des systèmes séquentiels i.e. des systèmes qui évoluent par des séquences d'opérations. Les éléments d'une spécification Z sont :

- Définition des types (de base, libres...) ;
- Déclarations globales (constantes, variables, opérateurs ou relation par définition axiomatique..) ;
- Déclaration de schémas. On distingues plusieurs types de schémas : Schéma particulier donnant l'état initial du système ;
- Opérations (ou partie d'opérations) accédant ou modifiant l'état du système ;
- Schémas particuliers décrivant les préconditions des opérations ;
- Prédicats : apparaissent dans les schémas ou dans les déclarations ;
- Preuves : qui consiste à démonter les théorèmes et les propriétés essentiels du système.

La spécification Z est organisée de la façon suivante :

1. Les déclarations globales débutent la spécification. Elles contiennent : des déclarations de types de base et de types libres, des déclarations de constantes, variables et fonctions globales.
2. Les types structurés sont définis par l'utilisateur de la manière suivante :
   - un schéma d'état regroupe les variables et un prédicat sur ces variables ;
   - un schéma initial est donné pour chaque schéma d'état ;
   - des schémas d'opérations décrivant chaque opération du type structuré. Ils se décomposent en *i*) un schéma du nom de l'opération décrit les variables d'état, les paramètres et un prédicat et *ii*) un schéma de pré-condition comprenant uniquement des paramètres en entrée ;
3. L'état du système est défini par un schéma d'état comprenant *a*)Un schéma donnant la structure du système (variables et invariant), et *b*) Un schéma initial décrivant l'état initial du système ;
4. Des schémas d'opérations décrivant chaque opération du système. Ils comprennent *a*) un schéma décrivant les variables d'état, les paramètres et un prédicat, et *b*) un schéma de pré-condition ;

5. Des théorèmes et des preuves enrichissent la spécification.

La dernière étape est la validation, qui comprend trois parties :

1. La preuve de propriétés générales de la spécification : la cohérence et la complétude.
2. La preuve des propriétés attendues du système : formaliser les propriétés des analystes (textuelles à l'origine), et les prouver dans des théorèmes.
3. Les propriétés liées à la pratique de Z : l'existante d'un état initial, préservation des invariants dans les opérations, preuve de raffinage de données ou d'opérations

4.5 Exercices

Exercice 1 :
L'administration d'un hôpital veut gérer automatiquement ses salles d'attente. Chaque salle a une capacité d'accueil limitée. chaque salle porte un nom, un emplacement. Les patients sont admis normalement ou en urgence. Les patients admis en urgence sont appelés prioritairement vis-à-vis les autres. L'ordre d'appel est l'ordre d'arrivée, compte-tenu des éventuelles urgences, un patient appelé est reçu et soigner par un médecin. Les soins ne sont pas interrompus par l'arrivée d'un patient en urgence. Sous l'hypothèse de l'existence de plusieurs salles d'attente, donnez le modèle conceptuel des données et le modèle conceptuel des traitements correspondants à cet énoncé.
Exercice 2 :
On a enregistré des utilisateurs d'un système informatique. A un moment donné, certains utilisateurs sont connectés à l'ordinateur et d'autres non. Spécifier en Z le système 'ordinateur' avec les opérations AjouterUtilsateur, SupprimerUtilisateur, Connexion, Déconnexion, et les erreurs correspondantes.
Exercice 3 :

— Définir le type libre Bin pour les chiffres binaires ;
— Définir, par un schéma, le type Octet (une suite de huit chiffres binaires) ;
— Définir, par un schéma, une opération d'addition binaire AddBin ayant en entrée deux chiffres binaires et en sortie un chiffre binaire resultat et un chiffre binaire retenue ;

— Modifier le schéma de la question 3 pour prendre en considération un autre chiffre binaire (retenue) en entrée ;
— Définir un schéma « Decal » de décalage à gauche des bits d'un octet (le décalage à gauche consiste à introduire un 0 à droite de l'octet et éliminer le chiffre le plus à gauche) ;
— En utilisant, le schéma de question 5, définir un schéma pour la multiplication binaire par deux (2) avec la retenue ;

Chapitre 5

Introduction à la Méthode B : Les Bases du Langage

Sommaire

5.1 Introduction

La méthode B est une méthode formelle permettant le développement de logiciels sûrs. Elle a été conçue par Jean-Raymond ABRIAL, qui avait déjà participé dans les années 1980 à la conception de la notation Z. La méthode B permet de formaliser des spécifications et des programmes. Pour cela, elle utilise son propre langage formel. Au niveau des spécifications, il s'agit d'un langage logique reposant sur une version simplifiée ad-hoc de la théorie des ensembles (cf. **??**).

5.2 Démarche de la Méthode B

Le développement d'un projet selon la méthode B comporte deux activités fortement liées : l'écriture de spécifications formelles et la preuve de ces mêmes spécifications.

L'activité d'écriture consiste à rédiger les spécifications formelles de machines abstraites à l'aide d'un formalisme mathématique de haut niveau. Ainsi, une spécification B comporte des données[1], des propriétés invariantes portant sur ces données[2], et enfin des services permettant d'initialiser puis de faire évoluer ces données[3]. L'activité de preuve d'une spécification B consiste alors à réaliser un certain nombre de démonstrations afin de prouver l'établissement et la conservation des propriétés invariantes en question[4]. La génération des assertions à démontrer est complètement systématique. Elle s'appuie notamment sur la transformation de prédicats par des substitutions.

Le développement d'une machine abstraite se poursuit par une extension de l'activité d'écriture lors d'étapes successives de raffinement. Raffiner une spécification consiste à la reformuler et enrichir une expression de manière plus concrète. L'activité de preuve concernant les raffinements et consiste également à réaliser un certain nombre de vérifications statiques et à prouver que le raffinement constitue bien une reformulation valide de la spécification. Le dernier niveau de raffinement d'une machine abstraite se nomme l'implantation (qui est assujetti à quelques contraintes supplémentaires. Par exemple, il ne peut plus manipuler que des données ou des substitutions ayant un équivalent informatique). Les données et les substitutions de l'implantation constituent un langage informatique similaire à un langage impératif. À ce titre, il peut

1. Des entiers, des booléens, des ensembles, des relations, des fonctions ou des suites
2. La logique des prédicats du premier ordre
3. les transformations des données sont exprimées à l'aide de substitutions
4. par exemple il faut prouver que l'appel d'un service conserve bien les propriétés invariantes

donc s'exécuter sur un système informatique après fabrication d'un exécutable, soit à l'aide d'un compilateur dédié soit en passant par une étape intermédiaire de traduction automatique vers Ada, C++ ou C.

Un projet B est constitué d'un certain nombre de composants.L'analyse d'un composant se scinde en trois parties successives : l'analyse lexicale, l'analyse syntaxique et l'analyse sémantique.

L'analyse lexicale consiste à vérifier qu'un composant est constitué d'une suite de lexèmes valides et à effectuer l'analyse et le remplacement des définitions textuelles. Ainsi,les éléments du vocabulaire terminal du Langage B (comme les identificateurs) sont définis.

Expression régulière	Chaîne de caractères	Exemple
x	le caractère x	a : la lettre a
$[x]$	le caractère x	$[+]$: le caractère $+$
$[xy]$	le caractère x ou y	$[aA]$: le caractère a ou A
$[x-y]$	un caractère de l'intervalle $x..y$	$[a-z]$: une lettre miniscule $[A-Z]$: une lettre majuscule
$[\hat{}x]$	tout caractère sauf x	$[\hat{}1]$: pas de caractère 1

5.3 Notations

Dans cette section, nous récapitulons la notation mathématique de base du langage B.

5.3.1 Prédicats

Les expressions logiques dénotent des prédicats qui sont interprétés par *true* ou $false$. Les opérateurs qui permettent de combiner des prédicats sont les connecteurs usuels de la logique : $\vee$, $\wedge$, $\neg$, $\Rightarrow$, et $\Leftrightarrow$. On peut quantifier les prédicats par les quantificateurs universel ($\forall$) et existentiel ($\exists$).
Les prédicats (et les expressions), contiennent des occurrences de variables. Ces occurrences sont liées si elles apparaissent sous la portée d'un quantificateur, sinon elles sont libres.
Notation de la "substitution". En B, la notion de substitution est à la base de la spécification des opérations. Une substitution est notée : $[x := E]P$.

Elle indique le remplacement uniforme des occurrences libres de x par E dans le prédicat P. Il est possible, aussi, d'utiliser la substitution multiple de la forme :

$$[x1, x2, \ldots := E1, E2, \ldots]P$$

Dans cette substitution, chaque variable x_i est substituée par E_i.
Les identificateurs des spécifications B doivent avoir au minimum deux caractères (deux lettres ou une lettre et un chiffre). Les parenthèses peuvent être utilisées pour délimiter les sous-expressions d'une expression logique.
La logique des prédicats implémentée par la méthode B est égalitaire, ainsi, toutes les valeurs (expressions) d'ensemble ou les valeurs (expressions) de types prédéfinis possèdent un prédicat égalité.

Une expression est une formule qui possède une valeur qui appartient à un type du langage B. Parmi les expressions, nous trouvons les constantes, les variables, les expressions arithmétiques formées par application des opérateurs usuels $(+, -, \times, mod, \ldots)$, des expression booléen formées par les opérateurs (connecteurs) logiques usuels, les expressions de fonctions, les expressions de relation, les expressions et les construction d'ensembles ... .
Une expression booléenne est de type "BOOL", et elle peut prendre les deux valeurs *"TRUE"* et *"FALSE"*. La conversion d'un prédicat *Pred* en une expression booléenne est réalisée par l'expression $bool(Pred)$. A titre d'exemple, l'expression : $bool(\exists x \cdot (x \in \mathbb{N} \wedge x > 4)$ a pour valeur *TRUE*.

5.3.2 La théorie des ensembles

Les ensembles peuvent être des ensembles d'éléments sans structure, ou bien des produits cartésiens d'ensembles, ou encore des parties d'un ensemble. Les notations de construction d'ensembles sont :

$\emptyset$	ensemble vide	
$\times$	produit cartésien	$Ens \times Ens$
$\mathbb{P}$	ensemble des parties d'un ensemble	$\mathbb{P}(Ens)$
$\mathbb{P}_1$	ensemble des sous-ensembles non vides	$\mathbb{P}_1(Ens)$
$\mathbb{F}$	ensemble des sous-ensembles finis	$\mathbb{F}(Ens)$
$\mathbb{F}_1$	ensemble des sous-ensembles finis et non vides	$\mathbb{F}_1(Ens)$
$\{e_1, e_2, \ldots, e_n\}$ $\{Expres \mid preds\}$	ensembles définis en extension ensembles définis en compréhension	
$\mathbb{N}$ $\mathbb{Z}$ INT	L'ensemble des entiers naturels L'ensemble des entiers relatifs L'ensemble des entiers relatifs compris entre MININT et MAXINT	

B offre des prédicats relatifs aux ensembles : $\in$ l'appartenance et sa négation $\notin$, l'inclusion $\subseteq$ et l'inclusion strict $\subset$, et leurs négations respectives $\not\subseteq$ et $\not\subset$. B implémente également les opérateurs ensemblistes d'intersection $\cap$, d'union $\cup$ et différence d'ensembles "$-$" ($S_1 - S_2 \stackrel{\text{def}}{=} \{x \mid x \in S_1 \wedge x \notin S_2\}$).

5.3.3 Relations

Les relations sont un cas particulier de construction d'ensembles. Il s'agit d'ensembles de couples d'éléments. La relation entre deux ensembles E_1 et E_2 est notée $E_1 \leftrightarrow E_2$, formellement cette relation est définie par $E_1 \leftrightarrow E_2 \stackrel{\text{def}}{=} \mathbb{P}(E_1 \times E_2)$.

Le domaine d'une relation R, noté $dom(R)$, est définir comme les éléments de E_1 qui sont effectivement en relation avec des éléments de E_2. Le codomaine, , noté $ran(R)$, est l'ensemble des éléments de E_2 qui sont en relation avec des éléments de E_1. L'image d'un ensemble par une relation, noté $r[F]$ est l'ensemble des éléments de E_2 qui sont en relation avec les éléments de F par la relation R.

En plus des opérations sur les ensembles qui s'appliquent aux relations. B offre plusieurs autres opérations : la relation identité id sur un ensemble qui associe à chaque élément le même élément, la relation inverse "$^{-1}$", la composition séquentielle ";", le produit direct "$\otimes$", la composition parallèle "$\|$", et les opérations de projection qui construisent les relations entre les ensembles paramètres et les éléments projetés droite ou gauche. Les définitions formelles de ces opérations sont :

opération	pré-condition	définition
$Id(E)$		$\{x, y \mid x \mapsto y \in E \times E \wedge x = y\}$
R^{-1}	$R \in S \times T$	$\{y, x \mid (y, x) \in T \times S \wedge (x, y) \in R\}$
$R_1; R_2$	$R_1 \in T \times U$ $R_2 \in U \times S$	$\{x, z \mid x, z \in T \times S \wedge$ $\exists y \cdot (y \in U \wedge (x \mapsto y) \in R_1 \wedge (y \mapsto z) \in R_2)\}$
$R_1 \otimes R_2$	$R_1 \in T \times U$ $R_2 \in U \times S$	$\{x, (y, z) \mid x, (y, z) \in T \times (U \times S) \wedge$ $(x \mapsto y) \in R_1 \wedge (x \mapsto z) \in R_2\}$
$R_1 \| R_2$	$R_1 \in T \times U$ $R_2 \in S \times W$	$\{(t, s), (u, w) \mid (t, s), (u, w) \in (T \times S) \times (U \times W) \wedge$ $(t \mapsto u) \in R_1 \wedge (s \mapsto w) \in R_2\}$
$prj_1(E, F)$		$\{x, y, z \mid x, y, z \in E \times F \times E \wedge z = x\}$
$prj_2(E, F)$		$\{x, y, z \mid x, y, z \in E \times F \times E \wedge z = y\}$

A l'instar du langage Z, B définit aussi les opérations de restriction du domaine et du codomaine, de soustraction du domaine et codomaine (en utilisant la même notation du langage Z).

5.3.4 Fonctions

Les fonctions sont des relations dont chaque élément du domaine n'est associé qu'à un seul élément du codomaine. La notation des familles de fonction en langage B est la même définie en langage Z (cf. Section 3.5 Page 40 pour plus de détails).

5.3.5 Types de données

En B, les ensembles sont introduit par leur nom au moment de leur première déclaration. Leurs éléments ne sont pas précisé, mais ils sont supposés non vides. Ils

sont appelés des "ensembles abstraits". Leur véritable contenu ne sera connu qu'à la phase d'implémentation. D'autres ensembles peuvent être définis avec leurs éléments sous forme d'énumération. Ils sont appelés "ensembles définis". D'autre part, on peut construire des expressions d'ensembles avec des types effectifs.

5.3.6 Les séquences

Les séquences sont des suites ordonnées d'objets d'un ensemble E. L'ensemble des séquences sur E est noté $seq(E)$. Les principales constructions sur les séquences sont :

[] la séquence vide ;	
$[s_1, s_2, \dots, s_n]$	la séquence formée des n éléments $e_1, \dots, e_n$;
$s_1 \hat{} s_2$	la concaténation des séquences s_1 et s_2 ;
$e \rightarrow s$	ajout de e au début de la séquence s ;
$s \rightarrow e$	ajout de e à la fin de la séquence s ;
$first(s)$	le premier élément de la séquence non vide s ;
$tail(s)$	retourne la séquence s sans son premier élément ;
$size(s)$	la taille de la séquence s ;
$front(s)$	tous les éléments sauf le dernier.
$rev(s)$	retourne la séquence inversée de s ;
$last(s)$	le dernier élément de s.

5.4 Machine Abstraite

La machine abstraite[5] est l'unité de décomposition de la méthode B. Elle contient deux parties, l'une statique et l'autre dynamique :

— La partie statique qui spécifie l'état du système. Elle détermine les variables qui décrivent l'état des différents composants du système et les invariants qui sont des formules logiques permettant de décrire les règles statiques du système. Ces variables sont définies au moyen de prédicats.

— La partie dynamique qui exprime l'initialisation et l'évolution de l'état du système à travers un ensemble d'opérations. Ces dernières sont définies par des

5. C'est un concept proche des notions de modules, de classes et de types abstraits de données.

substitutions généralisées et chacune d'elles contient une précondition [6] et une action [7].

La machine de base est formée d'un *nom*, un ensemble de *variables*, un *invariant* portant sur les variables de la machine, une *initialisation* de ces variables ainsi que des *opérations*.

MACHINE point3D
VARIABLES x, y, z
INVARIANT $x \in 0..100 \wedge y \in 0..200 \wedge z \in 0..150$
INITIALISATION $x, y, z := 0, 0, 0$
OPERATIONS
$v \longleftarrow abscisse = v := x;$
$v \longleftarrow ordonnee = v := y;$
$v \longleftarrow altitude = v := z;$
$Deplacer(dx, dy, dz) =$
$PRE\ x + dx \leq 100 \wedge y + dy \leq 200 \wedge z + dz \leq 150$
$THEN\ (x, y, z) := (x + dx, y + dy, z + dz)$
END
END

Clause MACHINE : permet de définir l'identificateur de la machine qui doit être unique. Cet identificateur peut être paramétré par des valeurs scalaires ou des ensembles.
Clause VARIABLES : contient un ensemble d'identificateurs représentant les variables de la machine. Elles correspondent à l'état de cette machine.
Clause INVARIANT : c'est un prédicat exprimant le typage des variables et les contraintes qu'elles doivent satisfaire.
Clause INITIALISATION : contient une substitution généralisée permettant de spécifier les valeurs initiales possibles des variables de la machine.
Clause OPERATIONS : contient les opérations de la machine qui doivent satisfaire l'invariant de celle-ci.

6. Une précondition représentée par un prédicat exprimant les conditions indispensables pour appeler l'opération.
7. Une action représentée par une substitution exprimant comment les variables de la machine évoluent.

5.5 Exercices

Exercice 1 :
Étant donnée les relations R_1, R_2, et R_3 ci-dessous,
$R_1 = \{(0,2),(1,5),(2,5),(3,5),(2,7)\}$
$R_2 = \{(0,1),(1,3),(2,1)\}$
$R_3 = \{(1,3),(0,1),(2,-1),(6,8),(3,5)\}$
calculer les ensembles de relations donnés par :
$R_1 \bigotimes R_2$, $R_1 || R_3$, R_2^0, R_2^1, R_2^2, R_2^3, R_2^n, R_2^*
Exercice 1 :
Étant donnée la relation $R = \{(1,3),(0,1),(2,-1),(6,8),(3,5)\}$:
Écrire une machine B qui contient la relation R,
Écrire une opération qui transforme R en incrémentant de 2 l'image (les images) de chaque élément.
Écrire une opération qui extrait la sous-relation de R dont le domaine ne contient que des nombres impairs.

Chapitre 6

Introduction aux Réseaux de Petri

Sommaire

6.1 Introduction

Ce chapitre introduit Les réseaux de Petri qui est un outil de modélisation utilisé généralement en phase préliminaire de conception de système (informatique, industriel, ...) pour leur spécification fonctionnelle, modélisation et évaluation.

6.2 Définitions

Le réseaux de Petri (RdP) est :

— Est un modèle mathématique servant à représenter divers systèmes (informatiques, industriels,...) travaillant sur des variables discrètes.

— Est un moyen de modélisation du comportement des systèmes dynamiques à événements discrets.

— Est un moyen de description des relations existantes entre des conditions et des évènements.

Un RdP est composé de places, transitions et arcs

a) Une place est représentée par un cercle ;

b) Une transition est représentée par un trait ;

c) Un arc relie soit une place à une transition, ou soit une transition à une place.

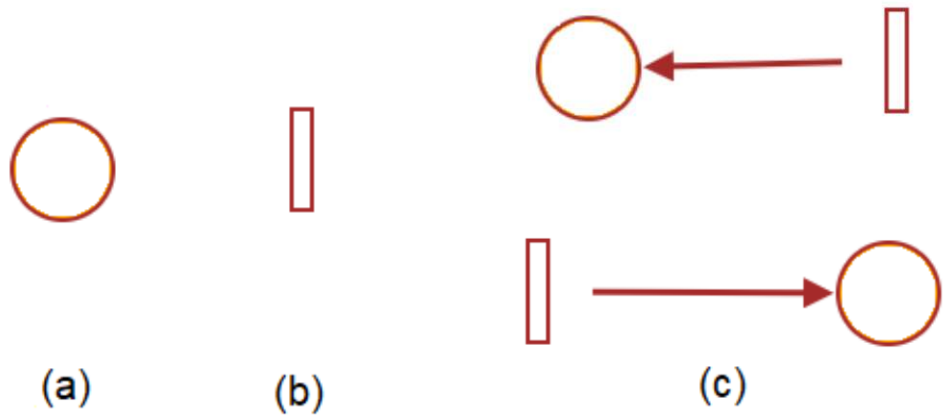


FIGURE 6.1 – Place, Transition et Arc

Chaque place contient un nombre entier positif ou nul de marques ou jetons. Le marquage M définit l'état du système décrit par le réseau à un instant donné. C'est

un vecteur colonne de dimension égale au nombre de places dans le réseau. Le i-ème élément du vecteur correspond au nombre de jetons contenus dans la place P_i.

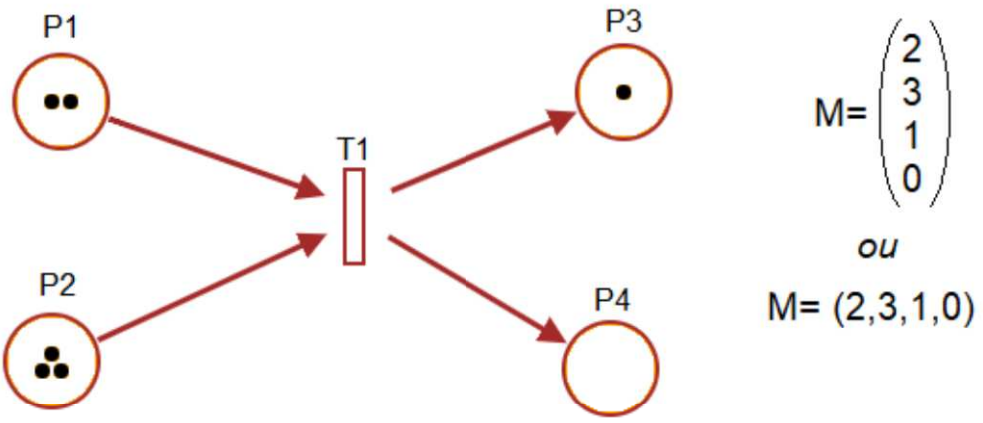


FIGURE 6.2 – Marquage

6.3 Franchissement d'une transition

— Une transition est franchissable lorsque toutes les places qui lui sont en amont (ou toutes les places d'entrée de la transition) contiennent au moins un jeton.
— Le franchissement consiste à retirer un jeton de chacune des places d'entrée et à rajouter un jeton à chacune des places de sortie de la même transition.
— Une transition franchissable n'est pas forcément immédiatement franchie.
— Une transition sans place d'entrée est toujours franchissable : c'est une transition source. Le franchissement d'une transition source consiste à rajouter un jeton à chacune de ces places de sortie.
— Une transition sans place de sortie est une transition puits. Le franchissement d'une transition puits consiste à retirer un jeton de chacune de ses places d'entrée.

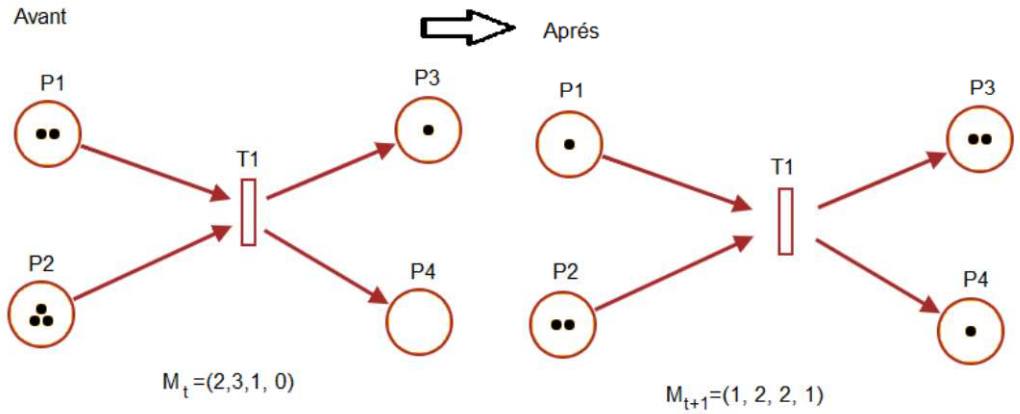


FIGURE 6.3 – Exemple de Franchissement d'une Transition.

6.4 Graphe de marquage

L'évolution temporelle d'un RdP peut être décrite par un graphe de marquage représentant l'ensemble des marquages accessibles, à partir d'un marquage initial M_0, et d'arcs correspondant aux franchissements des transitions faisant passer d'un marquage à l'autre. Le graphe de marquages est utilisé quand le nombre de marquages accessibles est fini. La figure 6.4 illustre un exemple de graphe de marquage associé à un RdP.

6.5 RdP autonome et non autonome

— Un RdP autonome décrit le fonctionnement d'un système dont les instants de franchissement ne sont pas connus ou indiqués.
— Un RdP non autonome décrit le fonctionnement d'un système dont l'évolution est conditionnée par des événements externes ou par le temps. Un RdP non autonome est synchronisé et/ou temporisé.

Il est possible d'adjoindre une temporisation à une place, une transition ou un arc du RdP[1]. Ainsi les jetons sont maintenus dans une place pendant un certain temps t avant tout franchissement de transition amont. On parle de RdP P-temporisé, T-temporisé, Arc-temporisé. Un exemple de RdP T-temporisé est illustré dans la Figure 6.5.

1. Ces réseaux de Petri nécessitent la définition d'une unité d'horloge.

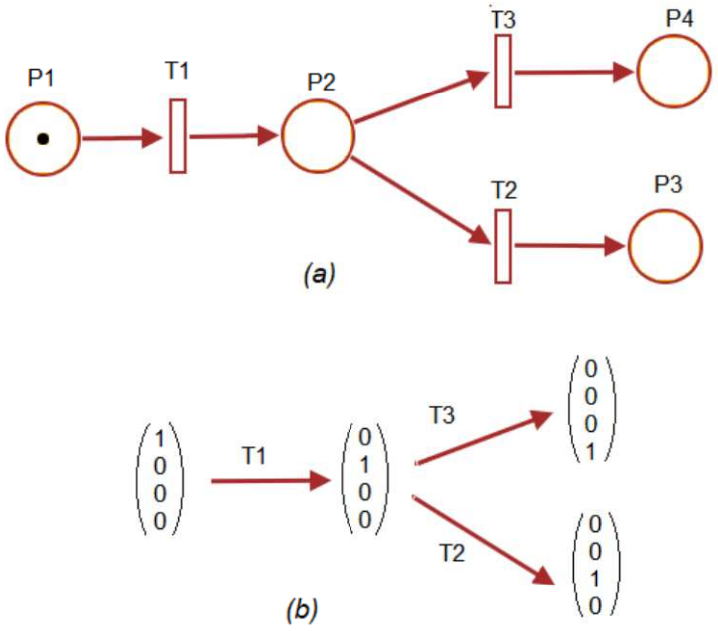


FIGURE 6.4 – (b) Graphe de Marquage associé au RdP de la figure (a)

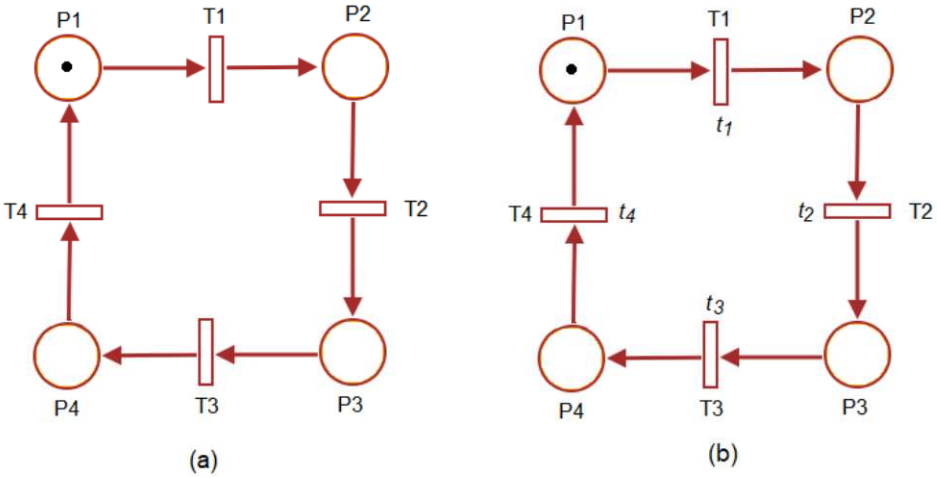


FIGURE 6.5 – (a) RdP autonome, (b) RdP T-temporisé

6.6 RdP Particuliers

6.6.1 Graphe d'état

Un réseau de Pétri non marqué est un graphe d'état si et seulement si toute transition a exactement une seule place d'entrée et une seule place de sortie. Un exemple de graphe d'état est donné dans la figure 6.6.

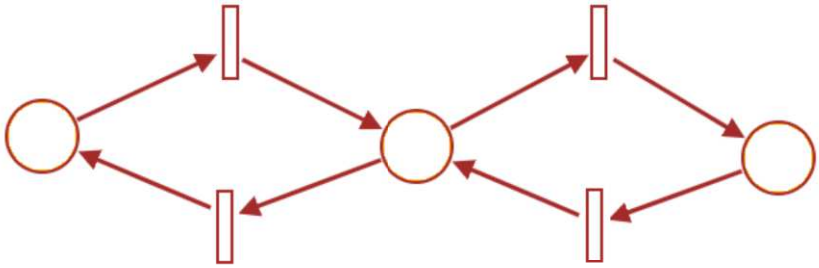


FIGURE 6.6 – Graphe d'état

6.6.2 Graphe d'événement

Un RdP est un graphe d'événement si et seulement si chaque place possède exactement une seule transition d'entrée et une seule transition de sortie. Dans la figure 6.7, le RdP (a) est un graphe d'événement contrairement aux RdP (b) et (c) où la place P_b possède deux transitions de sortie et la place P_c possède deux transitions d'entrée.

6.6.3 RdP Avec ou Sans Conflit

Un RdP sans conflit est un réseau dans lequel chaque place a au plus une transition de sortie.

Un RdP avec conflit est un réseau qui possède donc une place avec au moins deux transitions de sorties. Un conflit est noté : $[P_i, \{T_1, T_2, \dots, T_n\}]$; avec T_1, T_2, $\dots$, T_n sont les transitions de sorties de la place P_i.

Des exemples de RdP avec et sans conflits sont donnés dans la Figure 6.8.

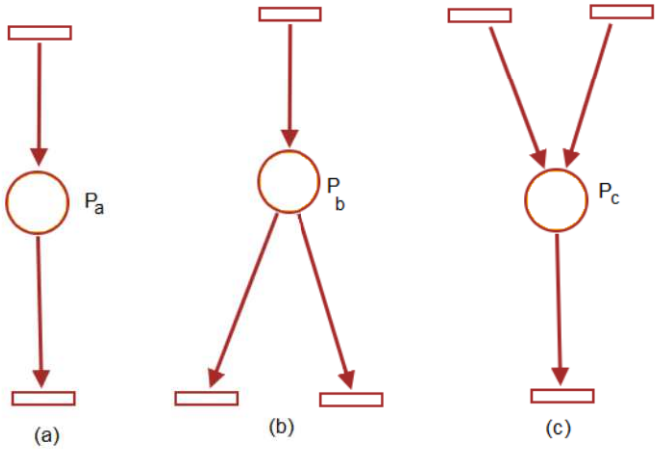


FIGURE 6.7 – (a) Graphe d'événement, (b) et (c) ne sont pas des Graphes d'événement

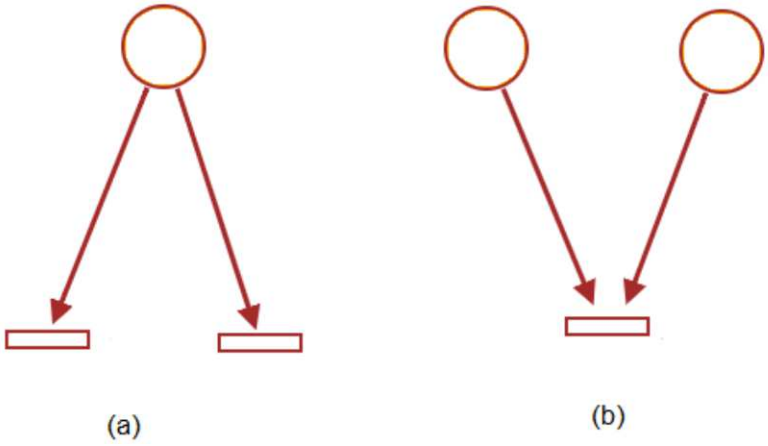


FIGURE 6.8 – (a) RdP Avec conflit, (b) RdP Sans conflit

6.6.4 RdP à choix libre

Un RdP est à choix libre est un réseau dans lequel pour tout conflit $[P_i, \{T_1, T_2, \dots, T_n\}]$ aucune des transitions $T_1, T_2, \dots, T_n$ ne possède aucune autre place d'entrée que P_i.

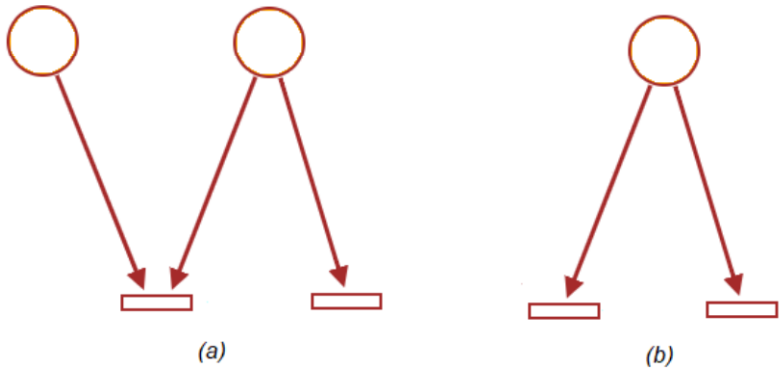


FIGURE 6.9 – RdP Avec conflit : (a) Sans Choix Libre (b) à Choix Libre

6.6.5 RdP généralisés

Un RdP généralisé est un RdP dans lequel des poids (nombres entiers strictement positifs) sont associés aux arcs. Si un arc (P_i, T_j) a un poids k : la transition T_j n'est franchie que si la place P_i possède au moins k jetons. Le franchissement consiste à retirer k jetons de la place P_i. Si un arc (T_j, P_i) a un poids k, le franchissement de la transition rajoute k jetons à la place P_i. Lorsque le poids n'est pas indiqué, il est égal à 1 par défaut. Les Figures 6.10 et 6.11 illustrent des exemples de franchissement des transitions dans des RdP généralisés.

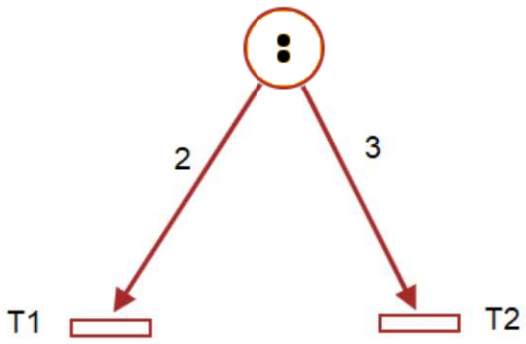


FIGURE 6.10 – RdP généralisé : T_1 est franchissable et T_2 ne l'est pas

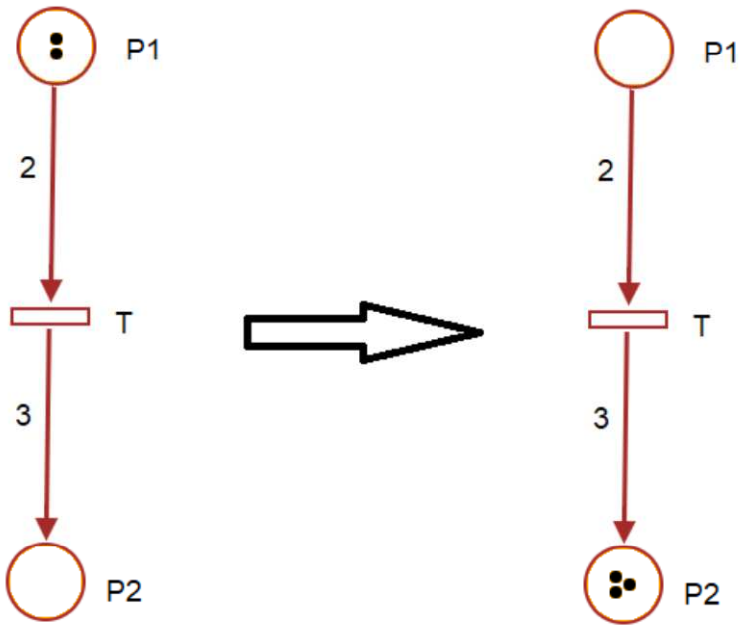


FIGURE 6.11 – RdP généralisé : le Franchissement de T consiste à retirer 2 jetons de P_1 et d'injecter 3 jetons dans P_2.

6.6.6 RdP à capacités

Un RdP à capacités est un RdP dans lequel des capacités (nombres entiers strictement positifs) sont associées aux places. Le franchissement d'une transition d'entrée d'une place P_i dont la capacité est $cap(P_i)$ n'est possible que si le franchissement ne conduit pas à un nombre de jetons dans P_i plus grand que $cap(P_i)$.

6.6.7 RdP à priorités

Dans un tel réseau si on atteint un marquage tel que plusieurs transitions sont franchissables, on doit franchir la transition qui a la plus grande priorité. Un tel réseau est utilisé lorsqu'on veut imposer un choix entre plusieurs transitions.

6.7 Exercices

Exercice 1 :
La machine 1 (resp. la machine 2) reçoit une pièce, la traite puis la dépose dans le stock 1 (respectivement stock 2), ces machines ne peuvent traiter qu'une seule pièce à la fois. La machine 3 fait l'assemblage : elle prend une pièce dans le stock 1 et une autre dans le stock 2, elle les assemble avant de les déposer dans un stock aval non considéré.

a) Construire le réseau de Pétri modélisant ce système ;

b) On considère que les stocks 1 et 2 ont des capacités limitées de 2 et de 3 pièces respectivement. Modifier le réseau de pétri pour prendre en compte cela.

Exercice 2 :
L'exécution à tour de rôle de deux tâches par une unité centrale consiste à exécuter une partie des instructions de la première tâche puis une partie de la seconde tâche, ainsi de suite.

a) Modéliser par un RdP l'exécution à tour de rôle de deux tâches, avec exécution d'une instruction d'une tâche à chaque fois.

a) Modéliser par un RdP généralisé l'exécution à tour de rôle de deux tâches, avec exécution de trois instructions pour la première tâche et de six instructions pour la deuxième tâche à chaque fois.

Exercice 3 :
Modéliser par un RdP une file d'attente (on suppose que la file a une capacité de 3).
Exercice 3 :
Etant donné deux files d'attente F1 et F2 (FIFO) de taille respective de 3 et de 2 éléments :

a) Modéliser par un RdP (standard) le fonctionnement en parallèle de ces deux files avec l'ajout simultané des éléments.

b) Modéliser par un RdP (standard) le fonctionnement en parallèle de ces deux files avec le retrait simultané des éléments.

Bibliographie

[1] J. M. Spivey, *The Z Notation, A reference Manual,* Prentice Hall, 1992.

[2] ClearSy, *Manuel de Référence du Langage B*, 2002.

[3] P. André & A. Vailly, *Génie logiciel : Spécification des logiciels deux exemples de pratiques récentes Z et UML*, Paris : Ellipses, 2001.

[4] M. Rached, *Spécification et vérification des systèmes temps réel réactifs en B*, Thèse de Doctorat, Université Paul Sabatier -Toulouse III, 2007

[5] J-P. Courtiat, C.A.S. Santos, C. Lohr, and B. Outtaj. *Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique*. Computer Communications, pages 1104–1123, 2000.

[6] J. Sinclair, *Action Systems: A Method Combining State-Based and Event-Based Specification.* In: Frappier, M., Habrias, H. (eds) Software Specification Methods. Formal Approaches to Computing and Information Technology FACIT. Springer, London. 2001.

[7] R. Milner: *A Calculus of Communicating Systems*, Springer Verlag, 1980.

[8] A. Choquet-Geniet, *Les réseaux de Petri : Un outil de modélisation*, Paris, Dunod, coll. « Sciences Sup », 2006,