# Conception de machines probabilistes dédiées aux inférences bayésiennes

Marvin Faix

Communauté
UNIVERSITÉ Grenoble Alpes

# THÈSE

Pour obtenir le grade de

## DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques & Informatique**

Arrêté ministérial : 7 août 2006

Présentée par

**Marvin Faix**

Thèse dirigée par **Emmanuel Mazer**
et codirigée par **Laurent Fesquet**

préparée au sein des **Laboratoires LIG & TIMA**
et de l'**École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

# Conception de machines probabilistes dédiées aux inférences bayésiennes

Thèse soutenue publiquement le **12 décembre 2016**,
devant le jury composé de :

**Monsieur, Olivier Sentieys**
Directeur de recherche INRIA, Président
**Monsieur, Philippe Leray**
Professeur Polytech'Nantes, Rapporteur
**Monsieur, Bruno Rouzeyre**
Professeur des Universités de Montpellier, Rapporteur
**Monsieur, Werner Krauth**
Directeur de recherche CNRS, Examinateur
**Monsieur, Philippe Galy**
Directeur de recherche STmicroelectronics, Examinateur
**Monsieur, Eric Alix**
Président de ProbaYes & Chief Data Officier du Groupe La Poste, Examinateur
**Monsieur, Emmanuel Mazer**
Directeur de recherche CNRS, Directeur de thèse
**Monsieur, Laurent Fesquet**
Maître de conférence à l'Institut Polytechnique de Grenoble, Co-Directeur de thèse

# Table des matières

# Table des figures

# Liste des tableaux

« L'art de la citation est l'art de ceux
qui ne savent pas réfléchir par eux-même. »
Voltaire, Dictionnaire philosophique.

# Chapitre 1

# Introduction

*Franklin savait compter deux par deux et lacer ses chaussures*[1]...mais AlphaGo est capable de battre un des meilleurs joueurs de Go au monde. Qu'est ce qui diffère entre cette machine à la puissance de calcul phénoménale et une tortue représentant, anthropomorphiquement parlant, un enfant de 5 ans ?

Il ne s'agit pas de se poser la question de l'intelligence artificielle mais de se positionner sur une couche beaucoup plus basse du calcul et se demander comment les opérations sont réalisées. Le calcul déterministe, basé sur la logique booléenne, est très performant lorsque les règles du jeu sont définies et surtout finies, au sens des mathématiques. Le cerveau humain est, quant à lui, capable d'interpréter, de saisir, de comprendre et de prendre des décisions cohérentes en interprétant des quantités astronomiques de variables et en leur attribuant plus ou moins de force dans le raisonnement effectué. Les différences fondamentales entre le calcul réalisé par un ordinateur et celui réalisé par un organisme vivant se situent au niveau de la prise en compte des incertitudes sur les variables, la méthode de calcul, la réalisation de ce calcul et la valeur accordée aux prémisses.

Dans ce chapitre, nous donnons le contexte dans lequel cette thèse s'intègre. Nous présentons les motivations et objectifs du projet BAMBI qui est l'instigateur de ces travaux de recherche. Un plan de lecture est ensuite proposé, suivi d'un résumé de notre contribution.

1. *Franklin* est une série animée destinée aux jeunes enfants dont le personnage principal est une tortue.

## 1.1 Tant qu'il y a la vie...

Le monde qui nous entoure est-il régi par des phénomènes aléatoires ? Aucune réponse ne sera proposée dans ce manuscrit. Cependant, nous pouvons affirmer que, au travers de ce supposé chaos, la vie s'est pérennisée où les êtres vivants ont appris à survivre, prenant des décisions à partir de données incomplètes, parfois de mauvaises qualités voire même contradictoires. Comme tout organisme vivant, chaque jour nous prenons des décisions à partir d'observations via un calcul qui n'a rien de déterministe et que l'on peut considérer comme probabiliste. Cette façon de calculer nous permet d'intégrer des nouveaux événements et de nous adapter aux potentielles variations qui existent dans le monde que nous observons. Nous considérons que notre façon intrinsèque de calculer prend compte des erreurs possibles sur la mesure d'une observation et la variation de la valeur d'un tel événement.

Dans mes travaux de recherche, je ne me suis pas intéressé à la retranscription de la façon dont le vivant réalise ses calculs mais je suis parti du constat que le cerveau humain consomme environ $20W$ pour effectuer une multitude d'actions possibles lorsqu'une machine peut avoir besoin de plusieurs kW pour réaliser un nombre bien plus limité de tâches. Si on prend l'exemple des voitures autonomes, représentées sur la figure 1.1, celles-ci ont besoin de plus de $1kW$ entre le système de capteurs, de calcul et la prise de décision pour déplacer le véhicule lorsqu'un homme seul est capable de réaliser cette action. Le problème est que la voiture possède des capteurs très performants qui donnent une évaluation précise de la distance et utilise des algorithmes complexes qui s'exécutent sur des plates-formes multiprocesseurs lorsque l'homme sait prendre des décisions suffisantes pour réaliser la tâche requise avec des approximations sur les distances, l'angle de rotation du volant et la vitesse à utiliser.

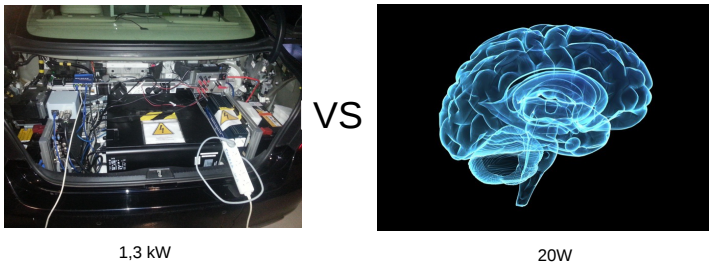



FIGURE 1.1 – Le cerveau face à la machine

On peut ajouter que la manière dont les êtres vivants réalisent leur calcul est très tolérante aux fautes. Il suffit de se rappeler à ses bons souvenirs de soirée arrosée où le taux d'alcool dans le sang important et/ou la prise de produits illicites entravaient sérieusement la mobilité et la réflexion mais n'empêchaient pas (toujours...) de retrouver son chez soi. Cette qualité est absente de la plupart des machines qui nécessitent une attention particulière pour gérer les fautes logicielles et matérielles sous peine d'enrayer irrémissiblement leurs performances. Dans mes travaux, je me suis confronté à ces problèmes classiques de concepteurs de circuits qui cherchent à concevoir des circuits robustes, à faible consommation et à grande efficacité énergétique. Ces trois grandes qualités que l'on retrouve dans le cerveau humain ou, à plus petite échelle, dans la façon de calculer des micro-organismes. Nos travaux se sont orientés vers le calcul stochastique car il nous semble être un candidat crédible dans la recherche de circuits répondant aux critères pré-cités. Il a aussi la bonne idée de bien s'accorder avec la théorie des probabilités.

Dans ce contexte, nous pensons que l'utilisation de la théorie des probabilités, et notamment de la théorie bayésienne, est une voie prometteuse. Tout d'abord parce qu'intrinsèquement une valeur de probabilité permet de donner toute l'information d'incertitude que l'on a sur une variable. Ensuite, parce que l'utilisation du théorème de Bayes nous permet de réviser les prémisses de nos connaissances au fur et à mesure que nous observons, une forme d'apprentissage dont le parallèle avec la façon dont on apprend a déjà été mis en lumière [46].

## 1.2 Il y a des probabilités

La théorie des probabilités permet de formaliser les corrélations qui existent entre des événements. Elle mesure l'incertitude des comportements possibles lors d'une expérience dite probabiliste ou aléatoire. C'est une discipline des mathématiques qui a été intronisée il y a plus de trois siècles notamment pour l'étude des jeux de hasards. L'étude des probabilités a connu un essor croissant au fil des ans, de la physique quantique aux phénomènes biologiques, à la météorologie ou encore l'économie et la finance. Là où il y a des phénomènes aléatoires, on essaie de créer un modèle prédictif qui nous permet de décrire le phénomène observé et d'attribuer une valeur à son caractère probable.

L'approche bayésienne a montré que malgré un manque d'information, des données incomplètes, elle permet de résoudre de nombreux problèmes

inverses. Elle se différencie de l'approche statistique standard car elle permet de déduire des résultats avec peu d'informations, au prix d'une complexité accrue de calculs. Lorsque les données sont en abondance, les deux approches donnent des résultats asymptotiquement équivalents. Alors pourquoi l'approche bayésienne bénéficie-t-elle de tant d'intérêts de la part de la communauté scientifique ?

Cognition [46], perception [16], classification, prédiction comportementale [8], prise de décision [6], robotique [47], les exemples d'utilisation de la théorie bayésienne ne manquent pas. Il y a deux principales explications à cela. Tout d'abord parce que lorsque la collecte d'informations abondantes devient trop coûteuse ou n'est pas possible dans des temps raisonnables, c'est la théorie bayésienne qui est privilégiée pour sa robustesse et ses performances face aux manques d'informations. Ensuite, parce que le modèle décrit est constamment en évolution au fur et à mesure des observations, ce qui rend le système dynamique et permet de le corriger constamment. Enfin, les méthodes bayésiennes permettent de modéliser les attentes au début de l'expérience, attentes qui seront renforcées ou contredites au cours de celle-ci, plus connues sous le nom de connaissances primaires. Cette information est souvent absente des autres méthodes de déduction statistiques.

La force de la théorie bayésienne est donc d'obtenir un maximum de résultats avec un minimum d'informations et de connaissances du monde. De là à dire qu'elle prône la paresse...c'est un pas qu'on ose franchir. C'est d'ailleurs ce que réalise chaque être vivant à tout moment, à partir d'observations et de connaissances préalables ; il réalise un calcul et prend une décision en minimisant les contraintes informatives. Cette façon de raisonner permet non seulement de se souvenir des expériences passées et de mettre à jour le modèle de décision, mais elle est aussi très efficace énergétiquement.

## 1.3 Le projet BAMBI : *Bottom-up Approaches for Machines dedicated to Bayesian Inference*

Les travaux présentés dans cette thèse sont entièrement intégrés dans le projet européen BAMBI FET FP7-ICT-2013-C. Ce projet a pour but de formaliser les principes de théorie du calcul et d'architectures logicielles et matérielles pour la conception de machines dédiées à la résolution de problèmes d'inférence bayésienne, en s'attachant à la compréhension des moyens employés par les êtres vivants pour réaliser leurs opérations. En ce sens, ces machines sont dites bio-inspirées. Ces deux axes de recherche sont complémentés par des chercheurs en physique des matériaux qui innovent dans

la conception de nouveaux nano-composants intrinsèquement aléatoire dans une plage d'utilisation. Le terme *bottom-up* réfère aux principes de conception qui part de la couche la plus basse de l'électronique, la conception de ces nouveaux types de nano-composants, et tente de les assembler pour réaliser la machine souhaitée. La figure 1.2 schématise les trois axes fondamentaux du projet BAMBI.

Ce projet mêle donc les compétences de chercheurs en théorie Bayésienne, biologie moléculaire, nano-physique, informatique et électronique. Les travaux présentés dans cette thèse s'intéressent aux axes de la théorie bayésienne, de l'informatique et de l'électronique.

### 1.3.1 Motivation

Les principes mathématiques théoriques de calcul sur lesquels reposent les machines actuelles ont été formellement exprimés par Alan Turing, il y a plus de 80 ans aujourd'hui. Les briques de base des architectures de ces calculateurs sont tout aussi anciennes, les architectures de Von Neumann portant encore aujourd'hui le nom de leur inventeur. D'un autre coté, le préambule du projet BAMBI est le constat que les êtres vivants possèdent une habilité à manier les variables physiques et à réaliser des calculs à forte efficacité énergétique. Le but n'est pas d'améliorer les façons dont on réalise le calcul aujourd'hui, mais de repenser la manière dont on le fait. En ce sens, les membres du projet BAMBI ont proposé de revoir les principes théoriques de base des calculateurs.

Ceux-ci sont basés sur la manipulation d'ensembles de variables binaires qui forment des composants que l'on peut assembler pour construire des composants plus complexes. Cette agilité a permis l'augmentation des performances des designs, notamment par la facilité de la réutilisation des composants déjà conçus. Un exemple évident est celui des smartphones, qui intègrent des technologies très différentes au sein d'un même appareil.

L'augmentation des performances, aussi bien en puissance de calcul qu'en coût énergétique, a été rendue possible par la miniaturisation de la taille des transistors qui suit la fameuse loi de Moore. C'est à dire que la taille des transistors est divisée par deux tous les 18 mois permettant l'augmentation de la densité d'intégration sur la puce. Cette intégration massive des transistors permet d'avoir des circuits complexes et performants en vitesse de calcul, puisque la réduction de la taille des transistors accélère le temps de passage au sein des portes logiques et donc la rapidité globale du cir-

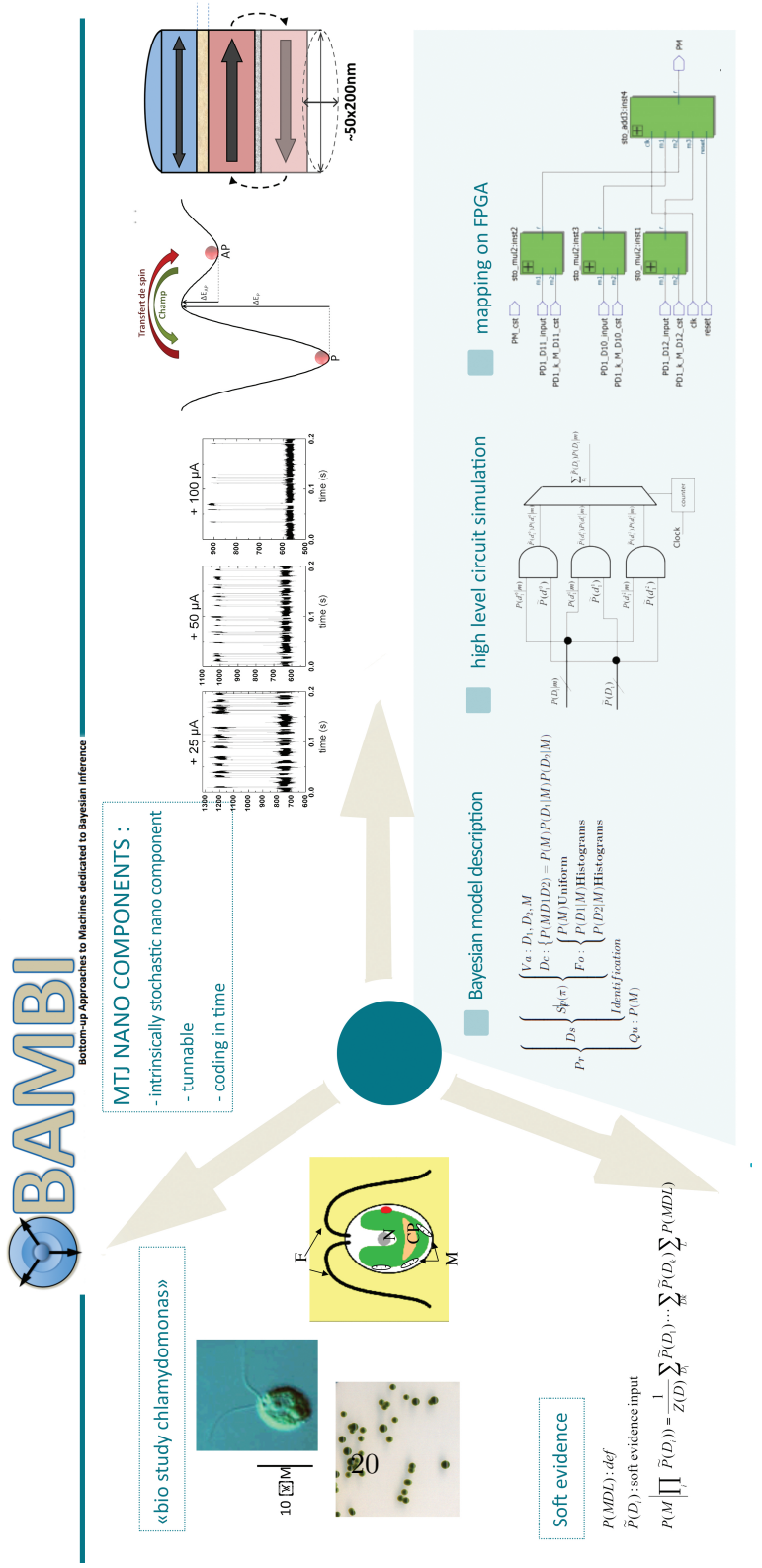



FIGURE 1.2 – Le projet BAMBI en trois axes

cuit. La figure 1.3 [2] schématise cette course à la performance pendant ces 40 dernières années. On constate la croissance exponentielle du nombre de transistors intégrés sur la puce et la stagnation de la fréquence de l'horloge à partir des années 2000.

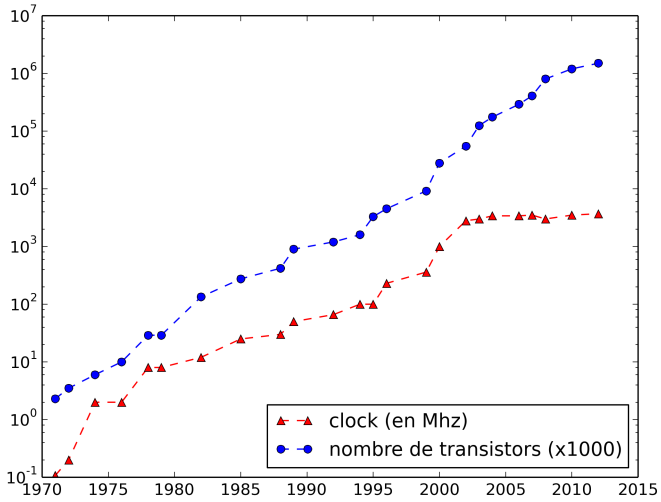



FIGURE 1.3 – Évolution de la densité d'intégration des transistors et de la fréquence d'horloge des processeurs

Toutefois, à la difficulté technologique liée à la réduction de la taille des transistors, s'ajoute le problème de fiabilité des calculs au sein des circuits. La réduction entraîne également des pertes plus importantes en énergie sous forme de fuites de courant (*leakage*). En parallèle, le coût des salles blanches pour rendre cette technologie possible suit une loi exponentielle. De nombreux chercheurs s'accordent à dire que cette loi tend vers sa limite depuis une décennie même si les technologies récentes présentent la possibilité de réduire la taille de $22nm$ à $7nm$.

Pour pallier ce phénomène, la communauté scientifique a stoppé la course à l'augmentation de la vitesse de l'horloge centrale sur un processeur en multipliant le nombre de cœurs au sein du circuit. On a alors vu l'émergence de plate-formes multiprocesseurs qui permettent de réaliser des calculs en parallèle et donc d'accélérer l'exécution des applications. L'horloge centrale a aussi été supprimée pour relaxer la contrainte de synchronisation [28], et

2. Données Intel `http://www.intel.com/pressroom/kits/quickreffam.htm`

pour notamment avoir un gain important en énergie. Cependant, les travaux sur les circuits asynchrones ont montré la nécessité d'ajouter de nombreux composants de contrôle qui entravent ce gain en performance.

Au sein du projet européen FET BAMBI, des chercheurs ont étudié les relations entre le calcul réalisé par les êtres vivants et la théorie bayésienne. Plus précisément, ils ont présenté des travaux montrant comment l'inférence probabiliste peut être réalisée à l'échelle biomoléculaire et comment les cascades de signaux des cellules biochimiques peuvent réaliser certains calculs probabilistes [1]. Ceci nous a conduit à remettre en cause les méthodes de calcul standards, que l'on désignera sous le nom de machines de Von Neumann, utilisant les unités de calculs arithmétiques standards.

Bien sûr, les algorithmes permettant de conduire des inférences probabilistes sont réalisables sur des machines standards. Un florilège de logiciels est d'ailleurs disponible, citons Blog [32], Figaro [35], Church [21] ou encore ProBT [7] que nous utilisons. Mais l'utilisation de ce type d'arithmétique entre en contradiction avec le but d'allégement des calculs à réaliser et de l'efficacité énergétique souhaitée. C'est pourquoi le projet BAMBI s'est fixé comme objectif de concevoir une machine bio-inspirée, non Von Neumann, réalisant les opérations nécessaires au calcul de l'inférence Bayésienne grâce à des composants simples utilisant le calcul stochastique.

### 1.3.2 Objectifs du projet

L'objectif principal du projet est la conception de machines probabilistes bio-inspirées, mêlant la technologie actuelle avec l'intégration de nouveaux nano-composants. Suivant les trois axes présentés précédemment, on peut recentrer les objectifs principaux par domaine.

Dans la théorie du calcul, l'équipe souhaite développer une théorie de calcul probabiliste basée sur des composants élémentaires vu comme une extension de l'algèbre booléenne. A l'instar des langages de programmation haut niveau, la notion de programme probabiliste s'exécutant sur ces nouvelles machines doit être rendue possible pour fournir une couche d'abstraction aux futurs programmeurs. Les données manipulées par une telle machine ne sont plus des données binaires mais des valeurs de probabilités. Le projet BAMBI propose en ce sens d'encoder une valeur de probabilité sur des signaux binaires stochastiques. La connaissance de la valeur d'un bit ne sera donc plus sa valeur $\{0, 1\}$ mais la distribution de probabilité sur cette valeur binaire. L'information portée par la valeur d'un bit s'en retrouve alors grandement augmentée.

Dans le contexte biologique, le lien entre le raisonnement probabiliste et les principes de traitement de l'information au sein de systèmes biologiques plus ou moins complexes est mis en œuvre en analysant les données fournies par leurs travaux, notamment sur les algues *Chlamydomonas reinhardtii*.

Utiliser des principes de calculs déterministes pour des applications probabilistes semble paradoxal tant l'effort, pour créer l'aléa nécessaire aux méthodes MCMCs (*Markov Chain Monte Carlo*) par exemple, contraint les machines actuelles dans leur performance d'exécution et de consommation. L'idée est donc d'adapter le hardware au calcul probabiliste. Inspirer par le vivant, nous imaginons les futures machines probabilistes massivement parallèles. De plus, les études montrent que les événements biologiques se produisent de façon stochastiques. Le sujet des physiciens intégrés au projet est donc d'utiliser des nano-composants dans des plages où leur instabilité, qui est classiquement un problème, est exploitée à des fins stochastiques et notamment comme générateurs aléatoires. Ces composants portent le nom de *super-paramagnetic tunnel junction* (MTJ) [26]. Le passage d'un état parallèle à un état anti-parallèle du composant s'effectue de manière aléatoire, ceci entraînant le passage ou non du courant. Grâce à ce simple composant, on est alors capable de générer des signaux télégraphiques dont la qualité de l'aléa est remarquablement peu coûteuse en taille et énergie. La figure 1.4 [3] montre le principe de changement de spin du composant MTJ.

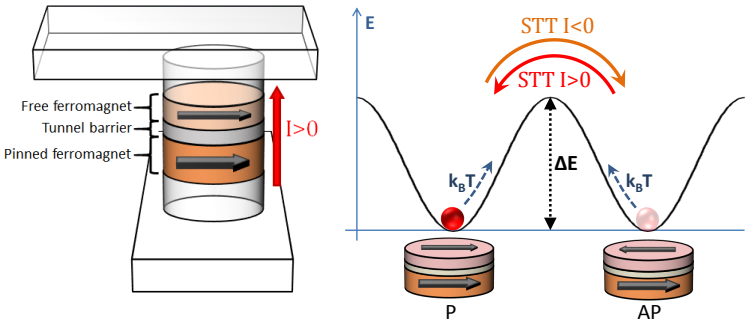



FIGURE 1.4 – Principe du phénomène aléatoire du MTJ

3. Cette figure est tirée de l'article [26]

## 1.4 Les travaux de thèse

Comme nous l'avons mentionné auparavant, les travaux présentés s'intègrent dans les domaines de la théorie bayésienne, de l'informatique et de l'électronique du projet BAMBI. Au vu des objectifs décrits précédemment, nous proposons d'interpréter un langage probabiliste, ProBT, en langage machine permettant la résolution de problèmes d'inférence bayésienne sur des architectures de calcul nouvelles.

### 1.4.1 Conception de deux types d'architecture de calcul

Dans ce manuscrit, nous présentons deux types de machine traitant des problèmes d'inférence bayésienne en utilisant le calcul stochastique.

Le premier type de machine, que nous nommerons *SMTBI*, pour *Stochastic Machine for Tractable Bayesian Inference*, est décrit dans le chapitre 4 et traite des problèmes d'inférences exactes. Le principe est d'utiliser l'expression formelle du calcul de l'inférence, générée par ProBT, et de construire le schéma électronique équivalent réalisant les opérations produits, sommes et division nécessaires pour le calcul stochastique. Le principe d'architecture repose sur la représentation du modèle bayésien en bus stochastiques dont les flux binaires encodent les valeurs de probabilité du modèle. Un intérêt de ces machines est la simplicité des composants électroniques qui réalisent le calcul. Mais l'atout majeur des *SMTBIs* est leur important parallélisme. En contre partie, ce fort parallélisme se paie en termes de surface silicium sur le circuit. Ce problème de taille, dans les deux sens du terme, est lié au fait que les problèmes d'inférence appartiennent à la classe des problèmes *difficiles* ou NP-complet. De ce fait, les machines *SMTBIs* ne sont pas adaptées aux problèmes à trop grandes dimensions.

Dans le cas des problèmes *difficiles*, nous avons conçu les machines nommées *SMABI*, pour *Stochastic Machine for Approximate Bayesian Inference*, présentées dans le chapitre 5. Ces machines diffèrent radicalement des précédentes dans leur conception. Elles reposent sur l'implémentation de l'algorithme de Gibbs comme méthode approchée de résolution de tels problèmes. Cet algorithme est utilisé pour échantillonner des variables discrètes au niveau binaire. Dans le cadre des *SMTBIs*, le calcul stochastique offre un résultat approché de l'expression calculée. Ce résultat est d'autant plus précis que la longueur de la fenêtre temporelle choisie pour reconstruire la valeur est grande. Cette relative lente convergence est un des points faibles de l'utilisation du calcul stochastique. Cependant, dans le cas des machines *SMABIs*, celles-ci se soustraient à cette problématique. En effet, nous n'avons

nul besoin de reconstruire les distributions, nécessaires dans l'exécution de algorithme de Gibbs, à partir des flux binaires pour échantillonner le bit sélectionné. Ce résultat n'a donc rien d'une simple nuance puisqu'il évite le point le plus décrié des méthodes de calcul stochastique, c'est à dire la relative lente convergence des résultats des opérations.

### 1.4.2 Plan de lecture

Ces travaux de recherche se sont intégrés dans un cadre pluridisciplinaire qui possède l'intérêt d'avoir une vision différente et des compétences diverses sur un sujet qui éclot. Une des difficultés est d'arriver à comprendre et se faire comprendre par des personnes aux domaines très variés. La rédaction de cette thèse suit les mêmes principes puisqu'au cours de mes recherches, j'ai navigué des couches basses de l'électronique jusqu'aux modèles théoriques mathématiques, notamment bayésiens, en passant par la programmation haut niveau de compilateurs nécessaires à l'utilisation des machines conçues. Nous espérons ainsi que ces travaux intéresseront aussi bien des électroniciens pour la conception de la machine que des utilisateurs de modèles bayésiens souhaitant un outil de calcul différent des machines standards.

Dans la même veine que le projet BAMBI, la communauté scientifique s'intéresse allègrement à d'autres manières de concevoir le calcul. Le chapitre 2 montre le regain d'intérêt des chercheurs sur la piste du *Rebooting computing*[4] en faisant étalage des projets initiés et des voies possibles pour le renouveau des principes de calcul et d'architecture de machine. La particularité des modèles Bayésiens est qu'ils siéent à de nombreuses applications tant par leur facilité d'adaptation au monde physique que par leurs méthodes de résolution. Naturellement, d'autres unités de recherche se sont donc intéressées à des machines spécialisées dans le calcul d'inférence. Nous faisons une revue des travaux les plus connexes à notre démarche en distinguant les machines traitant de l'inférence exacte grâce à la manipulation de signaux analogiques et celles proposant des méthodes approchées par la conception d'échantillonneurs en manipulant des signaux numériques.

Nous abordons déjà dans ce chapitre 2 quelques termes qui nécessitent une définition précise pour la suite de la lecture de cette thèse. Dans le chapitre 3, nous présentons la théorie des probabilités, les propriétés et théorèmes fondamentaux associés à celle-ci, notamment l'incontournable théorème de Bayes, d'un point de vue formel et terminologique. Nous donnons

4. `http://rebootingcomputing.ieee.org/`

une définition de l'inférence dans le cadre probabiliste et nous explicitons les deux méthodes de résolution : inférence exacte et inférence approchée. Grâce à ProBT, nous avons un langage de programmation probabiliste nécessaire pour modéliser les problèmes bayésiens, tester les résultats préliminaires des applications s'exécutant sur les machines conçues et surtout programmer ces machines. Dans ce chapitre, nous montrons comment programmer avec l'outil ProBT. Toutes ces parties sont largement développées dans l'ouvrage *Bayesian programming* de Bessière et al. [7]. Enfin, nous définissons le termes de calcul stochastique, dernier concept fondamental de notre machine probabiliste. Dans cette section, nous décrivons les différents générateurs de nombres aléatoires utilisés pour créer l'entropie nécessaire et deux méthodes pour comparer des distributions.

Le cœur de ces travaux de recherche se situe dans les chapitres 4, 5 et 6 où les conceptions de machines probabilistes réalisant l'inférence avec du calcul stochastique sont décrites et expérimentées sur des applications concrètes et des problèmes académiques *difficiles*.

Le chapitre 4 présente les *SMTBIs* qui sont des machines probabilistes traitant l'inférence exacte avec des composants logiques simples. Nous décrivons les opérateurs qui effectuent la multiplication, addition et normalisation nécessaire au calcul d'inférence. Nous simulons cette machine sur une application robotique d'évitement d'obstacle comme preuve de concept. La synchronisation de LFSRs dans le cadre des télécommunications montre que cette machine, restreinte dans le nombre de variables qu'elle peut manipuler, offre des perspectives d'applications dépassant le cadre académique. Nous présentons également l'outil de synthèse logique qui permet de spécifier les machines *SMTBIs* pour n'importe quel problème d'inférence. Celui-ci prend en entrée un programme ProBT et génère un code VHDL spécifiant la *SMTBI*.

Le chapitre 5 ouvre la voie du passage à l'échelle que ne permettent pas les *SMTBIs*. Les *SMABIs* peuvent non seulement réaliser le calcul de l'inférence sur des problèmes de tailles raisonnablement solvables en inférence exacte mais elle permet de résoudre également des problèmes *difficiles*. L'idée principale est de disposer des distributions sur les variables binaires, ce qui permet d'implémenter efficacement une méthode MCMC sous la forme de l'algorithme de Gibbs binarisé. Cette méthode nouvelle émane des réflexions de Jacques Droulez [5] sur un moyen efficace d'échantillonner une distribution avec des portes logiques simples. Le développement de cet algorithme au

5. Coordinateur du projet *BAMBI*, directeur de recherche au CNRS

niveau binaire permet d'explorer l'espace de recherche plus rapidement et de converger vers la distribution souhaitée. Les *SMABIs* n'ont besoin que de trois composants pour réaliser l'inférence qui sont décrits et étudiés dans ce chapitre. Ces composants sont les *CDEs*, pour *Conditionnal Disitrunion Element*, qui stockent les distributions connues nécessaires au calcul de l'inférence, les *OP*s, pour *Odd product* qui réalisent l'échantillonnage de Gibbs au niveau binaire et les *RNGs*, pour *Random Number Generator*, utilisés comme source d'entropie. Nous réutilisons l'exemple du robot autonome évitant les obstacles comme preuve de concept de cette machine. Pour montrer l'efficacité de l'algorithme implémenté face à des problèmes intraitables en inférence exacte, nous avons testé la machine sur un problème académique spécifique *difficile*, développé par Jacques Droulez, dont on est capable de connaître une approximation mathématique suffisante pour affirmer et caractériser la réussite de la machine dans la résolution de l'inférence. De la même manière que pour les machines *SMTBIs*, à chaque problème d'inférence une *SMABI* dédiée doit être conçue. Nous avons donc réalisé un nouvel outil de synthèse probabiliste pour ces machines. Celui-ci prend un programme ProBT en entrée et génère le code VHDL correspondant implémentant la machine.

Avoir l'outil ProBT à disposition nous a permis d'intégrer des fonctionnalités nouvelles au code source du langage probabiliste, avec la bienveillance des ingénieurs de Probayes. Leur participation dans ces travaux nous a permis d'avoir un simulateur simple et rapide pour les machines conçues. Probayes a également grandement contribué à l'élaboration des compilateurs (ou synthétiseur logique) à travers les travaux de Raphaël Laurent [6].

La chapitre 6 décrit une *SMABI* générique sous l'acronyme *PSMABI*, pour *Programmable Stochastic Machine for Approximate Bayesian Inference*. En effet, nous avons développé une machine unique qui est capable de résoudre n'importe quel problème d'inférence décrit dans le langage ProBT, sous réserve d'une dimensionnalité raisonnable, y compris des problèmes *difficiles* à très grandes dimensions.

Nous concluons dans le chapitre 7 en proposant des pistes d'améliorations des machines décrites et des voies pour l'intégration des travaux des physiciens sur les composants stochastiques.

6. Docteur en informatique, ingénieur R&D à Probayes

### 1.4.3 Contribution

Ce mémoire constitue une première étape dans la conception de machines probabilistes utilisant du calcul stochastique. Il s'agit d'une idée originale dans l'architecture de nouveaux calculateurs dédiés à la résolution de problèmes d'inférence bayésienne.

Mon apport porte sur l'assemblage des composants stochastiques constituant une machine type *SMTBI* pour la résolution d'applications comme la fusion de capteurs pour le contrôle d'un robot ou la synchronisation de LFSRs. Je propose également des composants permettant de réaliser une somme de probabilités et la normalisation. Ces travaux ont donné lieu à un papier et une communication dans la conférence ICCI*CC (*International Conference on Cognitive Informatics & Cognitive Computing*) [15] et une autre dans un *workshop* de *IROS* dédié aux problématiques bayésiennes [31].

Je me suis approprié la méthode de Gibbs binarisée, ainsi que l'exemple académique *difficile*, proposé par Monsieur Jacques Droulez, pour implémenter les machines type *SMABI* et tester leur efficacité. Une grande partie du travail de développement a consisté en l'élaboration d'outils de synthèse qui permettent de spécifier une machine, type *SMTBI* ou *SMABI*, à partir d'un programme ProBT. Cet outil de synthèse probabiliste réalise une partie du flot de conception de circuits en générant un fichier VHDL synthétisable et implémentable sur carte FPGA. Ce travail a été réalisé en collaboration avec Monsieur Raphaël Laurent. Ces travaux ont donné lieu à un article dans le journal TETC (*Transactions on Emerging Topics in Computing*) [14] et une communication dans la conférence ICCI*CC [37]. Cette communication a été réalisée par Rémi Canillas, alors stagiaire de notre équipe au moment de la publication.

J'ai entièrement conçu, réalisé et testé la machine *PSMABI*. C'est une machine programmable capable de résoudre n'importe quel problème d'inférence par la transformation de n'importe quel programme ProBT en langage interprétable par la machine via un outil de compilation dédié.

# Chapitre 2

# Contexte

*Big data*, loi de Moore, efficacité énergétique, robustesse des circuits, tous ces termes réfèrent aux limites physiques des circuits face à la demande croissante des applications de plus en plus gourmandes en termes de puissance de calcul. Sous l'ère de l'ordinateur déterministe, de l'algèbre de Boole et des machines de Von Neumann, choisir une direction revient à faire un compromis entre vitesse, consommation, coût et efficacité. Les architectures de calcul ont évolué mais pas leur principe de base. On a ajouté les nombres flottants codés sur 64 bits pour obtenir une approximation si bonne qu'on peut la considérer comme exacte dans de nombreuses applications. Doit-on continuer la course à la vitesse d'exécution face à la croissance des volumes des données à traiter ?

Dans ce chapitre, nous faisons une revue des réflexions sur les manières de repenser le calcul sous le terme anglophone *Rebooting Computing*. Nous convergeons vers les travaux des chercheurs qui se sont également intéressés aux machines non-conventionnelles dédiées aux problèmes d'inférences bayésiennes traitant l'inférence exacte et approchée.

## 2.1 *Rebooting Computing*

L'idée de repenser les méthodes de calcul de nos machines est partagée par un pan de la communauté scientifique pour les raisons diverses évoquées en introduction : recherche de circuits à faible consommation, calculateur bio-inspiré, résolution de problèmes n'exigeant pas une réponse exacte, efficacité énergétique ou encore conception de circuits tolérants aux fautes.

Plusieurs voies offrent des possibilités sur les différents aspects de la conception de circuits comme la recherche sur les nouveaux matériaux pour remplacer le classique silicium tel que le graphène ou les nanotubes de carbone. Cette recherche se situe au niveau de la physique des composants et permet un gain en conductivité des transistors ainsi conçus. Dans la même veine d'une amélioration des circuits CMOS standards, les chercheurs pensent à passer des circuits en 2D à des couches électroniques en 3D [2]. Ceci pourrait permettre d'intégrer plus de composants sur une même surface. Cependant, le problème de surchauffe se voit augmenté. Ces pistes intéressantes n'offrent pas de refonte des principes d'architectures et/ou de calculs réalisés par nos machines actuelles mais proposent des améliorations des architectures existantes.

Les machines bio-inspirées proposent quant à elles une voie qui s'oppose plus fortement aux calculateurs actuels. On peut citer le projet européen SpiNNaker [18] appartenant à l'*Human Brain Project* [40] qui conçoit une plate-forme massivement parallèle pour simuler l'activité du cerveau humain. Dans ce projet, les machines sont conçues avec des architectures de Von Neumann classiques. IBM conçoit également des machines bio-inspirées comme dans le projet TrueNorth [3, 13]. Les machines bio-inspirées les plus populaires portent le nom de réseau de neurones. Ces machines tentent de reproduire les interconnexions neuronales qui existent dans notre cerveau et sont donc aussi massivement parallèles. Elles ont montré leurs aptitudes dans l'apprentissage profond. Le plus souvent, les calculs réalisés sur le réseau se font avec des unités arithmétiques et utilisent une architecture de Von Neumann classique.

D'autres travaux sont allés encore plus loin dans la formalisation de principes non-conventionnels pour la conception de circuit. C'est le cas du projet *Probabilistic CMOS* (PCMOS) [9]. Son principal objectif est de construire des machines à faible consommation. L'idée est de faire baisser la tension nominale, de 30%, à laquelle les circuits CMOS standards fonctionnent et de compenser les erreurs avec une conception robuste. Une algèbre probabiliste a été développée pour formaliser la façon dont les erreurs sont propagées

le long d'une chaîne d'opérations logiques. Les chercheurs ont montré, sur une application de traitement d'images, une réduction spectaculaire (facteur 10) de la consommation énergétique. En plus de ce gain en consommation, ce type de circuit semble très tolérant aux fautes puisqu'il est conçu de telle sorte qu'il réalise les bonnes opérations même avec des taux d'erreurs propagées importants.

## 2.2 Travaux connexes

Dans le cadre des inférences bayésiennes, et plus généralement dans le cadre du calcul, on sait en théorie manipuler les événements incertains et extraire l'information qui nous intéresse. Dans [24] Jaynes exprimait l'idée que l'utilisation des probabilités est un candidat naturel pour traiter l'incertitude inhérente à tout raisonnement subjectiviste.

Les problèmes ne demandant pas de réponse exacte comme la perception ou la fusion de capteurs sont des problèmes qui siéent aux modèles probabilistes. Ces exemples montrent qu'une nouvelle approche de calcul pourrait permettre un gain dans les trois directions des problématiques des circuits intégrés : coût, énergie, puissance de calcul. Dans la suite, nous nous intéressons aux machines traitant explicitement de problèmes d'inférences.

### 2.2.1 Circuits analogiques pour l'inférence

En 2003, Vigoda [5] est un des pionniers dans la conception de machines de calcul dédiées aux inférences bayésiennes. L'idée principale est d'utiliser des signaux analogiques pour coder la valeur de probabilité d'une variable. Cependant, contrairement à nos travaux, Vigoda s'est surtout intéressé au calcul d'inférence exacte en utilisant le *message passing algorithm* [23]. Il s'est intéressé à la couche très basse des circuits puisqu'il a conçu les composants nécessaires pour implémenter l'algorithme et définir le circuit réalisant l'inférence. Dans ces travaux, Vigoda montre que ses machines ont un intérêt dans les télécommunications où le nombre de variables reste assez raisonnable pour être traité en inférence exacte. Des résultats probants, montrant notamment un gain en consommation, ont conduit Vigoda à créer l'entreprise Lyrics rachetée par Analog Device en 2010.

Dans les travaux présentés dans cette thèse, nous nous sommes également intéressés aux circuits traitant de l'inférence exacte, chapitre 4, et nous avons utilisé un problème dans les télécommunications comme preuve de concept de ce type de machine. L'application est connue sous le nom de *PN (pseudo*

*noise) sequence acquisition* (acquisition de séquence pseudo bruitée), elle est présentée dans la section 4.5.1.

### 2.2.2 Circuits d'inférences approchées

Toute personne utilisant l'inférence exacte pour résoudre un problème d'inférence bayésienne est confrontée au passage à l'échelle des opérations à réaliser.

Au MIT (Massachusetts Institute of Technology), un département dédié au calcul d'inférence a été créé sous le nom de *Probabilistic Computing Project*. Ce projet est basé sur les travaux de V. Mansinghka [48] qui s'est intéressé aux machines non-conventionnelles résolvant des problèmes bayésiens via le calcul d'inférence approchée. Dans un premier temps, V. Mansinghka a développé un langage probabiliste, *CHURCH* [21], dans lequel des méthodes d'inférences approchées sont implémentées. Cette approche est similaire à celle utilisée dans d'autres langages probabilistes tel que ProBT.

Dans sa thèse, V. Mansinghka va plus loin que le développement logiciel de méthodes probabilistes et propose des composants et des assemblages de ces composants pour le développement hardware de machines probabilistes. C'est E. Jonas, dans sa thèse soutenue en 2014 [12], qui poursuit ces travaux. L'idée principale est la modification des tables de vérité déterministes des composants logiques par des tables de vérité probabilistes. Nous montrons un exemple de porte AND probabiliste sur la table 2.1. Pour réaliser cela, ces chercheurs ajoutent à tout composant probabiliste une entrée booléenne émanant d'un générateur aléatoire qui apporte l'aléa nécessaire au comportement du composant suivant sa table de vérité. Dans ces travaux, V. Mansinghka considère que l'utilisation de ce type de composant peut être vue comme une extension de l'algèbre booléenne dans le sens où la suppression de l'entrée aléatoire entraîne le retour au fonctionnement classique, dans le sens déterministe, du composant. Avec ce principe, la conception de circuits probabilistes possède la même agilité que ceux actuels, c'est à dire que chaque composant stochastique est indépendant et peut être composé avec d'autres de façon totalement abstraite et transparente. Ainsi, il est possible de construire un circuit plus complexe avec une architecture cohérente et se placer à différents niveaux de la conception.

#### Principe

Les circuits stochastiques présentés dans la thèse de Jonas sont des circuits qui produisent des échantillons à partir d'une distribution de probabi-

| $IN$ | $OUT$ | $P$ |
|---|---|---|
| 00 | 0 | 1 |
| | 1 | 0 |
| 01 | 0 | 1 |
| | 1 | 0 |
| 10 | 0 | 1 |
| | 1 | 0 |
| 11 | 0 | 0 |
| | 1 | 1 |

| $IN$ | $OUT$ | $P$ |
|---|---|---|
| 00 | 0 | 1 |
| | 1 | 0 |
| 01 | 0 | 0.25 |
| | 1 | 0.75 |
| 10 | 0 | 0.75 |
| | 1 | 0.25 |
| 11 | 0 | 0 |
| | 1 | 1 |

TABLE 2.1 – Table de vérité déterministe, à gauche, et probabiliste, à droite, de la porte AND

lité. Ce tirage est effectué grâce à une source entropique extérieure. L'élément de base de tels circuits est donc un composant à deux entrées et une sortie. Cette sortie est conditionnée par une entrée qui est un échantillon et une valeur aléatoire, celle-ci permettant de tirer dans la distribution modélisée, voir figure 2.1. Ce type de circuits est facilement composable puisque le branchement en série de deux tirages suivant une distribution permet d'obtenir l'abstraction d'un tirage suivant une distribution à plusieurs variables. Supposons qu'un élément simple permette de tirer des échantillons B suivant P(B|A) et un autre C suivant P(C|B). Dans ce cas, l'obtention de couples d'échantillons (B,C) suivant P(B, C | A) s'effectue par composition séquentielle des deux composants précédents, voir figure 2.2.

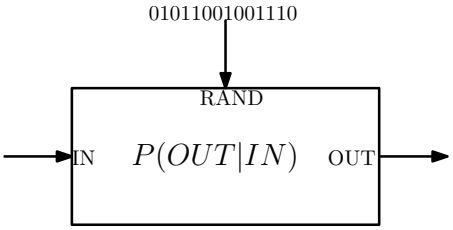



FIGURE 2.1 – Porte de base permettant le tirage conditonnel $P(OUT|IN)$

Un exemple simple de tirage d'échantillons avec ce type d'élément stochastique est le tirage de Bernoulli. Un mot de 4 bits est placé en entré du composant, et, suivant la valeur de ce mot, la sortie possède une probabilité d'être à 0 ou 1. Cette valeur de probabilité est modélisée grâce à un générateur aléatoire de 4 bits couplé à un simple comparateur avec l'entrée (de 4 bits également).

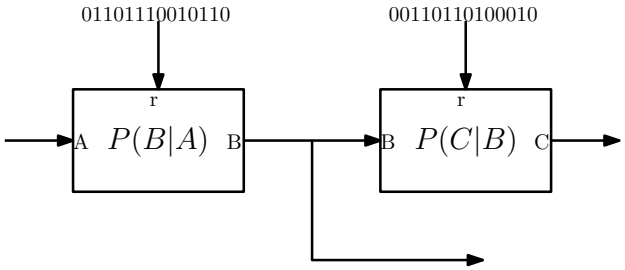



FIGURE 2.2 – Composition en série permettant l'échantillonnage $P(B, C|A)$

Cependant, l'élément de base de la conception de circuits avec des portes probabilistes dans les travaux des chercheurs du MIT est le tirage d'échantillons dans la fonction de répartition représentant la distribution de probabilité. Les valeurs des variables aléatoires sont stockées dans des ROM avec une valeur associée afin de reproduire la fonction de répartition. Puis le tirage est effectué avec un générateur et un comparateur comme précédemment. Ce type de conception est facilement implémentable dans des FPGAs (*Field Programmable Gate Arrays*) via des LUT (*Look Up Table*).

### Exemples académiques

Jonas a testé sa méthode sur trois exemples académiques classiques : modèle de l'arroseur automatique, modèle de Ising dans la physique statistique pour modéliser le phénomène de spin et un réseau bayésien de diagnostique médical ALARM (*A Logical Alarm Reduction Mechanism*).

Ces trois modèles sont décrits dans un langage informatique, *python* dans le cas de Jonas, et un compilateur permet de transformer ce modèle en circuit synthétisable sur FPGA. Cette partie est donc totalement transparente pour l'utilisateur. Chaque distribution et chaque variable vont être stockées via leur fonction de répartition et les générateurs nécessaires vont également être répartis sur le circuit afin de tirer des échantillons dans ces fonctions de répartition.

La description du modèle probabiliste est effectuée sous forme de graphe de facteur (*factor graph* en anglais, qui est un formalisme plus général que celui des réseaux bayésiens pour représenter une distribution conjointe) où les variables sont les nœuds du graphe et les dépendances entre les variables correspondent aux branches émanant de ces nœuds. C'est à partir de ce graphe que le compilateur construit le circuit réalisant l'inférence, c'est à dire répondant à une question posée sur ce graphe.

**Exemple du *sprinkler* : modèle de l'arroseur automatique**

Dans cet exemple on explicite les dépendances entre la météo, le fait qu'il puisse pleuvoir, l'arrosage automatique et le fait que l'herbe soit mouillée ou non, figure 2.3. Chaque nœud du graphe ainsi construit est booléen. La figure 2.2 montre le résultat de questions posées aux modèles. On remarque que les valeurs de probabilités avec tirage sont proches des valeurs théoriques avec une inférence exacte même avec une précision des valeurs sur seulement 5 bits. En effet, la précision des valeurs de probabilité est encodée sur un nombre de bits prédéterminé par l'utilisateur dans le modèle du compilateur. Dans cet exemple simple, 5 bits suffisent puisque les valeurs de probabilités à distinguer sont peu nombreuses et loin des extremums 0 ou 1.

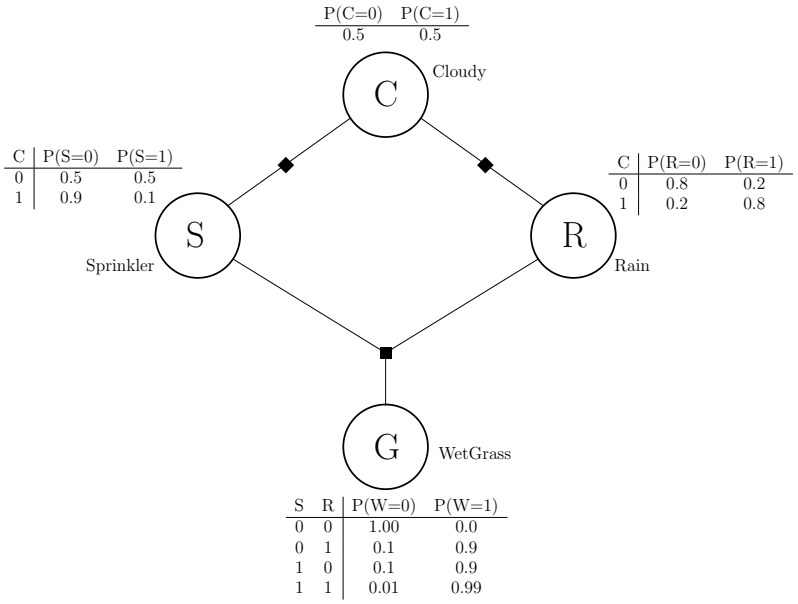



FIGURE 2.3 – Graphe du modèle *sprinkler*

| Question | inférence exacte | 5 bits | 8 bits | 12 bits |
|---|---|---|---|---|
| $P(S\|W)$ | 0.4298 | 0.4535 | 0.4320 | 0.4309 |
| $P(S\|W,R)$ | 0.1945 | 0.2160 | 0.2045 | 0.1935 |

TABLE 2.2 – Comparaison de résultats théoriques et avec tirages pour différentes précisions

## 2.3 Conclusion

La vision à long terme de la communauté "*rebooting computing*" est que les nouvelles applications et les nouveaux usages de systèmes artificiels devront utiliser des informations incomplètes et prendre des décisions en présence d'incertitudes. Ces domaines d'applications comprennent notamment la robotique autonome et le traitement capteurs mais vont bien au-delà et auront pour objectif de construire des outils capables de raisonnement et d'adaptation dans des situations nouvelles et incertaines. Cela implique un changement radical des modèles de calcul actuels. Ces nouveaux moyens seront en mesure de fournir des services inédits comme par exemple la télécommunication large bande (faible énergie, longue distance) ou la construction de capteurs intelligents pour la conduite automobile.

Dans ce contexte, les applications liées à la théorie bayésienne ont connu une forte croissance ces deux dernières décennies, notamment en robotique [47, 6, 16], en modélisation cognitive [46] et en apprentissage [8].

Nous avons vu que les travaux sur ces nouveaux principes de machine utilisaient des architectures de Von Neumann standards pour réaliser leurs opérations ou bien proposaient des méthodes de calculs différentes mais non générales. Nous montrons dans cette thèse des architectures de calcul génériques et utilisant un modèle de calcul non-déterministe.

# Chapitre 3

# Principaux concepts

Dans ce chapitre, nous définissons les termes, le vocabulaire et quelques propriétés fondamentales nécessaires à la bonne compréhension des chapitres suivants. En effet, même si le cas simple du calcul probabiliste est parfaitement compréhensible, les problèmes plus complexes peuvent souffrir d'une lacune de formalisme. Nous nous mettons donc d'accord avec un lecteur aguerri concernant la terminologie et les choix de représentation utilisés tout au long de ce mémoire, et nous espérons permettre à un lecteur plus novice dans le domaine du calcul probabiliste, d'aborder les parties suivantes avec une base théorique suffisante.

Le formalisme de la théorie des probabilités est présenté en définissant les variables, la conjointe, la marginalisation, la normalisation et le théorème de Bayes. Cette partie est tirée du livre *Bayesian Programming* [7], de même que la présentation de la programmation probabiliste. Nous invitons le lecteur à s'y plonger allégrement s'il souhaite étendre ses connaissances sur le sujet.

L'inférence probabiliste est également définie et les méthodes de résolution de ce type de problèmes sont divisées en deux grandes catégories : l'inférence exacte et l'inférence approchée. Les machines Bayésiennes présentées dans les chapitres suivants utilisent le calcul stochastique pour réaliser leurs opérations, nous explicitons donc ces termes et nous présentons différents générateurs aléatoires, outils fondamentaux de création d'entropie nécessaires à nos machines.

## 3.1 Présentation des probabilités

Dans cette section, nous présentons le vocabulaire et quelques propriétés de la théorie des probabilités dans le cadre de l'inférence. Afin d'illustrer ces termes, nous utilisons l'exemple de l'arroseur automatique, *sprinkler* en anglais, qui est un problème d'inférence sur trois variables binaires : la pluie *Rain* ($R$), l'arroseur automatique *Sprinkler* ($S$) et le gazon *Grasswet* ($G$). L'objectif est de savoir s'il a plu ou non ($R = 0$ ou 1) sachant que le gazon est mouillé ($G = 1$). Il s'agit donc de trouver la distribution de probabilité sur la variable *Rain* connaissant les valeurs de *Grasswet*. La variable *Sprinkler* est considérée dans ce problème comme une variable libre.

L'arroseur automatique, même s'il est non présent dans la question, a une importance fondamentale. En effet, s'il arrose toute la journée le gazon sera mouillé. C'est la définition d'une variable libre, elle n'apparaît pas dans la question mais elle est bien présente dans le modèle et impacte directement le résultat cherché.

### 3.1.1 Variable

Dans ces travaux, nous nous sommes intéressés aux problèmes à variables discrètes finies. Ces variables représentent les valeurs possibles des événements pour l'expérience décrite. Nous divisons l'ensemble des variables en trois types distincts :

- Cherchées (*Searched* $S$) : ce sont les variables dont on cherche la distribution.
- Connues ou Données (*Known* $K$, *Data* $D$) : elles représentent la connaissance de notre modèle, elles sont aussi appelées variables observées.
- Libres (*Free* $F$) : ce sont les variables qui apparaissent dans le modèle mais pas directement dans la question posée à celui-ci.

NB : par convention, nous utiliserons les majuscules pour représenter une variable et les minuscules pour représenter la réalisation de la variable, c'est à dire sa valeur.

***Exemple du sprinkler*** :

- *Sprinkler*, $S$ : variable binaire ON/OFF représentant la mise en route de l'arroseur automatique.
- *Rain*, $R$ : variable binaire indiquant s'il a plu ou non.
- *Grasswet*, $G$ : variable binaire représentant le fait que la pelouse soit mouillée ou non.

### 3.1.2 Probabilité

La probabilité quantifie le niveau d'incertitude que l'on donne à la valeur d'une variable. Dans l'exemple du *sprinkler*, la probabilité qu'il ait plu, sans aucune autre information, peut être modélisée par la moyenne des jours de pluie de la région où se situe le gazon étudié.

***Exemple du sprinkler*** :
- $P(R = 1) = 0.3$, pour la région grenobloise.
- $P(R = 1) = 0.9$, pour nos amis anglais.

On peut affiner le modèle dit du *prior* avec de l'apprentissage. Dans l'exemple du *sprinkler*, la variable *Rain* peut être affinée avec la région mais aussi avec la période de l'année par exemple (printemps, été, automne ou hiver).

### 3.1.3 Normalisation

L'ensemble de tous les événements possibles crée ce qu'on appelle l'univers de l'expérience étudiée, noté généralement $\Omega$. Il en résulte que $P(\Omega) = 1$, événement certain.

En discrétisant l'espace des possibles $X$ et en attribuant un poids à chaque événement $x_i$, on peut reconstituer l'univers en sommant tous les événements :

$$\sum_i P(X = x_i) = 1$$

***Exemple du sprinkler*** :
- $P(R = 1) + P(R = 0) = 0.3 + 0.7 = 1$

### 3.1.4 Probabilité conditionnelle

Il est possible d'apporter de l'information à un modèle afin d'obtenir une distribution de probabilité qui est paramétrée en fonction de la condition donnée. C'est ce qu'on appelle une distribution de probabilité conditionnelle : $P(S|R)$, $P(G|R\ S)$, pour l'exemple du *sprinkler*. Par convention, la condition est définie à droite du signe « | », qui se lit « sachant », et la variable cherchée est disposée à gauche de ce signe. Elles correspondent à la connaissance que l'on a des dépendances entre les variables.

***Exemple du sprinkler*** :

- $P(S|R)$ :
  - $R = 0$ : il n'a pas plu
    - $P(S = 1|R = 0) = 0.8$
    - $P(S = 0|R = 0) = 0.2$
  - $R = 1$ : il a plu
    - $P(S = 1|R = 1) = 0.2$
    - $P(S = 0|R = 1) = 0.8$
- $P(G|RS)$ :
  - $R = 0\ S = 0$ :
    - $P(G = 1|R = 0\ S = 0) = 0$
    - $P(G = 0|R = 0\ S = 0) = 1$
  - $R = 0\ S = 1$ :
    - $P(G = 1|R = 0\ S = 1) = 0.7$
    - $P(G = 0|R = 0\ S = 1) = 0.3$
  - $R = 1\ S = 0$ :
    - $P(G = 1|R = 1\ S = 0) = 0.7$
    - $P(G = 0|R = 1\ S = 0) = 0.3$
  - $R = 1\ S = 1$ :
    - $P(G = 1|R = 1\ S = 1) = 1$
    - $P(G = 0|R = 1\ S = 1) = 0$

### 3.1.5 Conjointe

Une distribution de probabilité conjointe correspond aux valeurs de probabilités sur la conjonction logique de toutes les variables, cherchées, connues et libres, du modèle $S \wedge K \wedge F$. Chaque type de variables peut elle-même être une conjonction de $n$ variables, par exemple $K = K_0 \dots K_{n-1}$. Si on appelle $card_S$, $card_K$ et $card_F$, respectivement le cardinal des variables $S$, $K$ et $F$ alors le nombre de valeurs pour la distribution $P(S \wedge K \wedge F)$ vaut $card_S \times card_K \times card_F$.

***Exemple du sprinkler*** :

- $P(R \wedge S \wedge G)$ : $2^3 = 8$ valeurs possibles.

$$\{P(R = 0\ S = 0\ G = 0),\ P(R = 0\ S = 0\ G = 1),$$
$$P(R = 0\ S = 1\ G = 0),\ P(R = 0\ S = 1\ G = 1),$$
$$P(R = 1\ S = 0\ G = 0),\ P(R = 1\ S = 0\ G = 1),$$
$$P(R = 1\ S = 1\ G = 0),\ P(R = 1\ S = 1\ G = 1)\}$$

### 3.1.6 Théorème de Bayes

Le théorème de Bayes est un théorème fondamental de l'inférence probabiliste, il est aussi connu sous le nom d'inversion de Bayes. Soit deux variables $X$, $Y$ et leur conjonction $X \wedge Y$. Le théorème de Bayes s'écrit alors comme suit :

$$\begin{aligned} P(X \wedge Y) &= P(X)P(Y|X) \\ &= P(Y)P(X|Y) \end{aligned} \tag{3.1}$$

On en déduit ce qu'on appelle l'inversion de Bayes :

$$P(X|Y) = \frac{P(X)P(Y|X)}{P(Y)} \tag{3.2}$$

Cette formule montre l'appartenance des modèles Bayésiens aux systèmes dynamiques. Les observations faites sur la variable $Y$ permettent de faire apprendre le modèle et augmentent sa pertinence.

### 3.1.7 Marginalisation

La marginalisation est une propriété qui permet d'obtenir une distribution en sommant sur toutes les valeurs possibles d'une variable de la conjonction d'une autre distribution. En équation, cela se résume comme suit :

$$\begin{aligned} \sum_Y P(X \wedge Y) &= \sum_Y P(X)P(Y|X) \quad \text{par théorème Bayes} \\ &= P(X) \sum_Y P(Y|X) \quad \text{par factorisation} \\ &= P(X) \quad \text{car on somme l'univers, donc } \sum_Y P(Y|X) = 1 \end{aligned} \tag{3.3}$$

### 3.1.8 Questions à un modèle Bayésien

Soit la conjonction suivante avec les variables cherchées, $S$, et connues, $K$ : $P(S \wedge K)$. Avec les définitions et propriétés précédentes, nous sommes en mesure de répondre à toute question sur un modèle probabiliste défini par la conjointe $P(S \wedge K \wedge F)$. La question devient :

$$\begin{aligned} P(S|K) &= \frac{P(S \wedge K)}{P(K)} \\ &= \frac{\sum_F P(S \wedge K \wedge F)}{\sum_{S,F} P(S \wedge K \wedge F)} \end{aligned} \tag{3.4}$$

***Exemple du sprinkler*** : La question posée est $P(R|G)$ sur la conjointe $P(R \wedge S \wedge G)$. En utilisant le théorème de Bayes, il existe de multiples façons de décomposer cette conjointe en produit de termes. Pour modéliser chaque terme, nous choisissons un outil mathématique que l'on nomme forme paramétrique. Nous montrons dans l'équation 3.5, les six façons de décomposer le modèle de *sprinkler* sans indépendance conditionnelle :

$$\begin{aligned} P(R \wedge S \wedge G) &= P(G)P(S|G)P(R|S\ G) \\ &= P(G)P(R|G)P(S|S\ G) \\ &= P(S)P(R|S)P(G|S\ R) \\ &= P(S)P(G|S)P(R|S\ G) \\ &= P(R)P(S|R)P(G|R\ S) \\ &= P(R)P(G|R)P(S|R\ G) \end{aligned} \tag{3.5}$$

La décomposition ainsi définie permet de réaliser les calculs sur les distributions en définissant les dépendances entre les variables. Dans notre exemple du *sprinkler*, on indique justement qu'il n'y a pas d'indépendance entre les variables. Cependant, dans de nombreuses applications, cette indépendance, définie par le modélisateur, implique une réduction drastique de la taille des distributions et du nombre d'opérations à réaliser. Le choix de la décomposition est donc une étape cruciale. Elle se fait par la connaissance du modèle que l'on étudie. Dans *sprinkler*, nous utilisons la décomposition $P(R \wedge S \wedge G) = P(R)P(S|R)P(G|RS)$ ou $P(R \wedge S \wedge G) = P(R)P(G|R)P(S|RG)$ car c'est elle qui permet de définir le *prior* sur la variable pluie ($R$) et, suivant la connaissance que l'on a sur la façon dont varie la variable $S$ suivant $R \wedge G$ ou bien la variable $G$ suivant $R \wedge S$, on choisira l'une ou l'autre des décompositions. Nous avons choisi de modéliser la distribution de $S$ suivant $R$ et $G$ suivant $R \wedge S$. La réponse à la question posée s'obtient donc par l'expression 3.6 suivante :

$$\begin{aligned} P(R|G) &= \frac{P(R \wedge G)}{P(G)} && \text{par le théorème de bayes} \\ &= \frac{\sum_S P(R \wedge S \wedge G)}{\sum_{RS} P(R \wedge S \wedge G))} && \text{par marginalisation} \\ &= \frac{\sum_S P(R)P(S|R)P(G|RS)}{\sum_{RS} P(R)P(S|R)P(G|RS)} && \text{en utilisant la décomposition} \\ &= P(R) \cdot \frac{\sum_S P(S|R)P(G|RS)}{\sum_{RS} P(R)P(S|R)P(G|RS)} && \text{en factorisant} \end{aligned} \tag{3.6}$$

Les distributions $P(R), P(S|R), P(G|RS)$ ont été définies lors de la modélisation. On est donc capable de réaliser le calcul $P(R) \cdot \frac{\sum_S P(S|R)P(G|RS)}{\sum_{RS} P(R)P(S|R)P(G|RS)}$ qui nous permet d'avoir accès à la distribution *a posteriori* $P(R|G)$. Le terme $\sum_{RS} P(R)P(S|R)P(G|RS)$ est régulièrement noté $Z$, il s'agit de la normalisation.

## 3.2 Inférence probabiliste

### 3.2.1 Une définition possible

L'inférence est une méthode de déduction logique. A partir des prémisses considérées comme véritables, on en tire des conclusions considérées alors aussi comme véritables. L'exemple de syllogisme très connu est celui que l'on attribue à Aristote, père de la logique déductive :

*Tous les Hommes sont mortels,*
*Socrate est un homme,*
*alors Socrate est mortel.*

Les ordinateurs sont basés sur cette logique déductive, plus connue sous le nom d'algèbre de Boole, c'est-à-dire une manière de prendre des décisions avec des propositions de type vrai-faux.

L'inférence probabiliste peut permettre d'étendre l'algèbre de Boole dans le sens où elle permet de prendre en compte l'incertitude sur les variables et de réviser ses prémisses. Les prémisses peuvent toujours être considérées comme des valeurs logiques Booléennes mais la conclusion peut prendre toutes les valeurs du segment [0,1]. Il est encore possible d'étendre cette définition en utilisant ce qu'on appelle les *soft évidences*. En effet, celles-ci permettent de définir des prémisses probabilistes. Le terme inférence réfère toujours à tirer des conclusions suivant une série d'hypothèses ou d'observations. Ces conclusions prennent la forme de distributions de probabilité sur les valeurs possibles des variables inférées.

Dans la théorie des probabilités, deux types d'inférence existent. L'inférence dite exacte qui consiste à utiliser les propriétés décrites précédemment pour calculer de façon exacte la distribution cherchée ; l'inférence approchée qui offre une approximation de la distribution cherchée. Ce type d'inférence est utilisée lorsque les variables du problème sont trop nombreuses et/ou de trop grandes dimensions.

### 3.2.2 Inférence exacte

Le calcul de l'inférence exacte revient à réaliser les opérations de sommes et produits, puis effectuer une division à partir de l'expression formelle décrite dans l'équation 3.6 :

$$P(S|K) = \frac{\sum_F P(S \wedge K \wedge F)}{\sum_{S,F} P(S \wedge K \wedge F)}$$

A partir de la description de la conjointe et l'utilisation des différentes propriétés décrites précédemment, nous sommes capable d'obtenir l'expression formelle qui répond à n'importe quelle question d'inférence. Cependant, nous ne sommes pas forcément en mesure de réaliser ces calculs. En effet, les problèmes d'inférences appartiennent à la catégorie des problèmes NP-difficiles [11]. La croissance du nombre des variables fait croître exponentiellement le nombre de sommes à réaliser.

Face à cette problématique, des algorithmes de réduction du nombre d'opérations à réaliser existent, comme le *Successive Restriction Algorithm* [30] par exemple. On est capable de supprimer les termes qui se somment à 1. Les sommes imbriquées sont réduites en factorisant les termes qui sont indépendants des autres. La dernière étape de simplification est l'ordonnancement des sommes restantes. Le problème d'ordonnancement dans un problème d'inférence est aussi NP-complet. Ces algorithmes heuristiques permettent donc de réduire le nombre de sommes à réaliser mais ne proposent pas forcément le nombre minimal de sommes.

### 3.2.3 Inférence approchée

Dans de nombreux cas, le calcul de l'inférence exacte est impossible à cause du trop grand nombre de variables et/ou de leur trop grande cardinalité. On est alors capable d'approximer le calcul de l'inférence en utilisant des algorithmes d'inférence approchée. Certaines de ces méthodes sont connues sous le nom de méthodes de *Sampling* ou d'échantillonnage. Elles consistent à tirer des échantillons dans l'espace des valeurs possibles des variables et approchent la distribution cherchée. Dans la plupart des cas, la difficulté réside dans la façon de générer des échantillons.

Dans les prochaines sous-sections, nous faisons une revue des principales méthodes d'échantillonnage simple puis nous présentons une méthode de Monte-Carlo Markov Chain (*MCMC*) particulière qui est l'algorithme de Gibbs. C'est cet algorithme qui est implémenté dans les machines bayésiennes *SMABI*, chapitre 5 et *PSMABI*, chapitre 6. Ces méthodes sont basées sur l'aléa des tirages effectués. Elles approchent la distribution cherchée

puisque le phénomène stochastique aura tendance à tirer des échantillons là où il y a de la masse de probabilité et ainsi recouvrir la forme de la distribution cherchée. Du fait que l'on tire des échantillons de manière aléatoire dans l'espace des valeurs possibles, plus on tire d'échantillons et plus la précision de notre méthode est grande. Par passage à la limite, on retrouve le résultat de l'inférence exacte. En effet, si on tire une infinité d'échantillons $\{s_i, f_i, k_i\} \in \{S, F, K\}$, on a, pour chacun des points $\{s_j, f_k, k_l\}$ de la distribution conjointe :

$$\lim_{n \to +\infty} \frac{\sum_{i=1}^{n} \delta_{\{s_i, f_i, k_i\} = \{s_j, f_k, k_l\}}}{n} = P(s_j, f_k, k_l)$$

**Échantillonnage direct**

L'échantillonnage direct consiste à tirer des valeurs de variable dans la conjointe afin d'obtenir un vecteur échantillon. Soit le modèle suivant composé de trois variables $\{A, B, C\}$ :

$$P(A \wedge B \wedge C) = P(A)P(B|A)P(C|AB) \tag{3.7}$$

L'algorithme réalisant l'échantillonnage direct sur ce modèle est présenté sur la figure 3.1. Au fur et à mesure des échantillons $\{a, b, c\}$ nous sommes capables de construire la distribution $P(A \wedge B \wedge C)$.

**while** $i <$ *nombre_echantillon_souhaite* : **do**
  Tirer un échantillon $a_i$ de A suivant $P(A)$
  Tirer un échantillon $b_i$ de B suivant $P(B \mid [A = a_i])$
  Tirer un échantillon $c_i$ de C suivant $P(C \mid [A = a_i]\ [B = b_i])$
  $i++$
**end while**

FIGURE 3.1 – Algorithme d'échantillonnage direct pour estimer $P(ABC)$

Dans le cadre de l'inférence, nous posons une question au modèle bayésien. Cette méthode d'échantillonnage n'est donc valable que dans deux cas :

— Soit il n'y a pas de variables «connues» dans notre modèle et on tire toutes les variables de celui-ci selon l'algorithme précédent.
— Soit les variables «connues» n'apparaissent jamais à gauche dans aucun des termes de la décomposition et on tire les variables d'intérêts suivant les termes de la conjointe.

***Exemple du sprinkler*** : On rappelle le modèle $P(R \wedge S \wedge G) = P(R)P(S|R)P(G|R\ S)$. Nous avons vu une question particulière posée à ce modèle dans les sections précédentes : $P(R|G = 1)$. Dans ce cas, nous ne pourrons pas appliquer la méthode d'échantillonnage direct suivant la décomposition proposée puisque l'algorithme montre qu'il faudrait échantillonner la variable $G$ qui est fixée dans notre question. Une question possible, solvable avec cette méthode, pourrait être alors : $P(G|R = r)$, où $r \in \{0, 1\}$ est une valeur possible de $R$. Pour effectuer ces tirages successifs nous utilisons la description des distributions de la section 3.1.4.

**while** $i <$ *nombre_echantillon_souhaite* : **do**
  Tirer un échantillon $s_i$ de S suivant $P(S \mid [R = r])$
  Tirer un échantillon $g_i$ de G suivant $P(G \mid [R = r]\ [S = s_i])$
  $i++$
**end while**

FIGURE 3.2 – Algorithme d'échantillonnage direct pour estimer $P(G|R = r)$

### Échantillonnage par rejet

Soit $P(ABC)$ une distribution sur les variables $A$, $B$ et $C$. L'échantillonnage par rejet consiste à utiliser une autre distribution, $T$, sur laquelle on est capable : (i) d'évaluer sa valeur en tout point $\{a, b, c\}$, (ii) de réaliser un échantillonnage direct dessus.

Chaque échantillon $\{a, b, c\}$, tiré de $T$, est alors accepté avec la proportion :

$$p = \frac{k \times T(\{a, b, c\})}{P(\{a, b, c\})}$$

où $k$ est une constante tel que :

$$\forall \{a, b, c\} \in \{A, B, C\}, \quad k \times T(\{a, b, c\}) < P(\{a, b, c\})$$

A noter que l'on doit aussi être capable d'évaluer $P(\{a, b, c\})$.

Cette méthode nous permet notamment d'échantillonner une distribution conditionnelle $P(B \mid [A = a])$ et donc de donner une solution approchée à un problème d'inférence. La figure 3.3 présente l'algorithme d'échantillonnage par rejet sur l'exemple de la distribution $P(ABC)$.

Avec cette méthode nous sommes capables d'approcher toute distribution conditionnelle. Cependant, le rejet d'échantillons peut faire décroître

**while** $i <$ *nombre_echantillon_souhaite* : **do**
  Tirer un échantillon $\{a_i, b_i, c_i\}$ de $\{A, B, C\}$ suivant $T(ABC)$
  Faire un tirage de Bernoulli avec la proportion $p = \frac{k \times T(\{a_i, b_i, c_i\})}{P(\{a_i, b_i, c_i\})}$
  **if** *resultat_tirage_bernoulli* $= 1$ : **then**
    On stocke $\{a_i, b_i, c_i\}$
  **else**
    On rejette $\{a_i, b_i, c_i\}$
  **end if**
  $i++$
**end while**

FIGURE 3.3 – Algorithme d'échantillonnage par rejet pour estimer $P(ABC)$

drastiquement sa vitesse de convergence. En effet, si la condition est très peu probable, le nombre d'échantillons rejetés est alors très grand et nous avons besoin d'une quantité d'itérations très importante pour obtenir suffisamment d'échantillons.

***Exemple du sprinkler*** : On rappelle le modèle $P(R \wedge S \wedge G) = P(R)P(S|R)P(G|R\ S)$. Nous avons vu une question particulière posé à ce modèle dans les sections précédentes : $P(R|G = 1)$. La figure 3.4 présente un algorithme permettant d'effectuer l'échantillonnage par rejet sur notre exemple.

**while** $i <$ *nombre_echantillon_souhaite* : **do**
  Tirer un échantillon $r_i$ de R suivant $P(R)$
  Tirer un échantillon $s_i$ de S suivant $P(S \mid [R = r_i])$
  Tirer un échantillon $g_i$ de G suivant $P(G \mid [R = r_i]\ [S = s_i])$
  **if** $g_i = 1$ **then**
    On stocke $r_i$
  **else**
    On rejette $r_i$
  **end if**
  $i++$
**end while**

FIGURE 3.4 – Algorithme d'échantillonnage par rejet pour estimer $P(R|G = 1)$

Cette façon d'échantillonner revient à prendre, dans l'algorithme général présenté au dessus, la distribution $T$, la constante multiplicative $k$ et la proportion $p$ suivante :

$$\begin{aligned} T &= P(R)P(S|R)\delta(G=1) \\ k &= 1 \\ p &= \frac{P(R)P(S|R)P(G=1|RS)}{P(R)P(S|R)\delta(G=1)} \end{aligned}$$

Dans ce cas, on tire $G$ suivant le dirac $\delta(G=1)$, on a donc toujours $G=1$, puis on tire $r$ et $s$ suivant $P(R)$ et $P(S|R=r)$. Au vu des formes paramétriques présentées dans la section 3.1.4, soit l'échantillon $\{r,s\}$ est compatible avec $G=1$ et dans ce cas on tire l'acceptation de cet échantillon avec la proportion $p$, soit l'échantillon $\{r,s\}$ est incompatible avec $G=1$ et on rejette alors cet échantillon. Ce qui donne :

$p=0, \quad \Rightarrow$ rejet l'échantillon r
$p>0, \quad \Rightarrow$ acceptation de l'échantillon r avec la proportion p

### Échantillonnage par importance

L'échantillonnage par importance génère des échantillons avec un poids associé. Soit $P(ABC)$ une distribution sur les variables $A$, $B$ et $C$. L'algorithme consiste à utiliser une autre distribution, $T$, sur laquelle on est capable : (i) d'évaluer sa valeur en tout point $\{a,b,c\}$, (ii) de réaliser un échantillonnage direct dessus. A Chaque échantillon $\{a,b,c\}$, tiré de $T$, est alors associé un poids $w = \frac{P(abc)}{T(abc)}$. A noter que l'on doit aussi être capable d'évaluer $P(abc)$.

Cet algorithme est présenté sur la figure 3.5 :

**while** $i <$ *nombre_echantillon_souhaite* : **do**
  Tirer un échantillon $\{a_i,b_i,c_i\}$ de $\{A,B,C\}$ suivant $T(ABC)$
  Calculer le poids $w_i = \frac{P(a_ib_ic_i)}{T(a_ib_ic_i)}$
  On stocke $(\{a_i,b_i,c_i\},w)$
  $i++$
**end while**

FIGURE 3.5 – Algorithme d'échantillonnage par importance pour la distribution $P(ABC)$

Dans le cas où l'on a une décomposition du modèle et que l'on pose une question sous la forme d'une inférence à calculer, alors l'échantillonnage par importance peut être vu comme une méthode d'échantillonnage par rejet mais qui ne s'intéresse qu'aux variables d'intérêt. Cela signifie que l'on ne va générer que des échantillons associés aux conditions/observations. De ce fait, cette méthode ne rejette aucun échantillon. Les échantillons tirés suivent les distributions décrites dans la décomposition. Les distributions où apparaissent une variable observée « à gauche », les vraisemblances donc, permettent de pondérer le résultat de l'échantillon, c'est à dire de calculer «$w$».

***Exemple du sprinkler*** : On rappelle le modèle $P(R \wedge S \wedge G) = P(R)P(S|R)P(G|RS)$. La question posée est $P(R|G = 1)$.

**while** $i <$ *nombre_echantillon_souhaite* : **do**
  Tirer un échantillon $r_i$ de $R$ de suivant $P(R)$
  Tirer un échantillon $s_i$ de $S$ de suivant $P(S \mid [R = r_i])$
  Calculer le poids $w_i = P(G = 1 \mid [R = r_i]\ [S = s_i])$
  On stocke $(r_i, w_i)$
  $i++$
**end while**

FIGURE 3.6 – Algorithme d'échantillonnage par importance pour la distribution pour estimer $P(R|G = 1)$

### Méthode MCMC

Les méthodes de Monte-Carlo par chaînes de Markov sont des méthodes d'échantillonnage qui possèdent la particularité de générer des échantillons à partir des échantillons précédents. L'ensemble des valeurs que peut prendre la distribution peut être assimilé à un ensemble d'états que l'on parcourt. Le passage d'un état à l'autre, suivant la probabilité, est défini par la distribution. La chaîne de Markov ainsi décrite a pour loi stationnaire la distribution à échantillonner.

#### Algorithme de Gibbs

Nous présentons un algorithme utilisant une méthode MCMC, l'algorithme de Gibbs, qui est utilisé dans les chapitres 5 et 6.

Soit $P(X)$ une distribution sur $n$ variables $X_i$ tel que :

$$X = X_1 \wedge \cdots \wedge X_n$$

Cet algorithme tire des nouveaux échantillons $X_i$ à chaque pas suivant la distribution

$$P(X_i \mid X_1 \cdots X_{i-1}\ X_{i+1} \cdots X_k)$$

Une phase d'initialisation du vecteur $X$ est alors nécessaire au début de la boucle algorithmique. La figure 3.7 montre un exemple d'échantillonnage d'une distribution sur trois variables discrètes $\{A, B, C\}$. La principale difficulté réside dans la façon d'effectuer ces tirages successifs avec l'ensemble des variables, non échantillonnées, fixées.

Initialiser le vecteur d'échantillons $\{a_0, b_0, c_0\}$
**while** $i <$ *nombre_echantillon_souhaite* : **do**
  Tirer un échantillon $a_i$ de $A$ de suivant $P(A \mid [B = b_{i-1}]\ [C = c_{i-1}])$
  Tirer un échantillon $b_i$ de $B$ de suivant $P(B \mid [A = a_i]\ [C = c_{i-1}])$
  Tirer un échantillon $c_i$ de $C$ de suivant $P(C \mid [A = a_i]\ [B = b_i])$
  stocker $a_i, b_i, c_i$
  $i++$
**end while**

FIGURE 3.7 – Algorithme de Gibbs

Cet algorithme démarre avec un vecteur d'initialisation $\{a_0, b_0, c_0\}$. Une phase, appelé *burn in*, est souvent utilisée pour se soustraire au biais introduit par le choix du vecteur initial. Cela consiste simplement à rejeter la première partie des échantillons, cette partie peut être aussi grande que l'on souhaite.

***Exemple du sprinkler*** : On rappelle le modèle $P(R \wedge S \wedge G) = P(R)P(S|R)P(G|R\ S)$. La question posée est $P(R|G = 1)$. La figure 3.8 montre l'algorithme utilisé pour estimer la distribution $P(R|G = 1)$. Seule la variable $R$ nous intéresse et la valeur de $G = 1$ est fixée. On ne tirera donc jamais de valeur de $G$. L'échantillonnage de $S$ est nécessaire puisqu'il apparaît dans le terme $P(R \mid S = s\ G = 1)$.

## 3.3 Programmation probabiliste

Il existe de nombreux langages de programmation probabiliste, on peut citer Figaro [35], Blog [32] ou encore Church [21] qui forment une liste non-exhaustive des logiciels existants. Dans nos travaux, nous utilisons ProBT. La programmation probabiliste présentée dans cette section montre les étapes nécessaires à la programmation d'un modèle Bayésien dans ce langage.

Initialiser le vecteur d'échantillons $\{r_0, s_0\}$
**while** $i < nombre\_echantillon\_souhaite$ : **do**
  Tirer un échantillon $r_i$ de $R$ de suivant $P(R \mid S = s_{i-1}\ G = 1)$
  Tirer un échantillon $s_i$ de $S$ de suivant $P(S \mid R = r_i\ G = 1)$
  stocker $\{r_i\}$
  $i++$
**end while**

FIGURE 3.8 – Algorithme d'échantillonnage de Gibbs sur l'exemple du *sprinkler*

La figure 3.9 schématise les étapes de programmation d'un modèle probabiliste. Un programme se décompose en deux grandes parties. La première est la spécification du modèle qui consiste à décrire les variables du modèle, la décomposition de la conjointe en produits de distributions et les formes paramétriques de ces distributions. La seconde est la question posée au modèle qui spécifie l'inférence souhaitée. Ces questions sont de la forme $P(S|K)$, où $S$ est l'espace des variables cherchées et $K$, l'espace de variables connues. Avec cette description, ProBT est capable de résoudre ces inférences par des méthodes exactes ou approchées. La formalisation du modèle probabiliste reste la même pour les machines stochastiques présentées dans ce document. Les programmes ProBT sont alors compilés par les outils dédiés aux machines afin de s'exécuter, non plus sur un ordinateur standard, mais sur celles-ci.

$$Programme \begin{cases} Spécification \begin{cases} Variables \\ Décomposition \\ Formes \end{cases} \\ Question \end{cases}$$

FIGURE 3.9 – Description d'un programme probabiliste avec ProBT

***Exemple du sprinkler*** : nous présentons la description du programme probabiliste, figure 3.10, et sa transcription dans le langage ProbT, figure 3.11 pour modéliser le problème du *sprinkler*.

$$
Prog \begin{cases} Spéc \begin{cases} Var : R(Rain), S(Sprinkler), G(Grasswet) \\ Déc : P(R \wedge S \wedge G) = P(R)\, P(S \mid R)\, P(G \mid RS) \\ Form : \begin{cases} P(R) : \begin{cases} P([R=0]) = 0.7 \\ P([R=1]) = 0.3 \end{cases} \\ P(S \mid R) : \begin{cases} P([S=0] \mid [R=0]) = 0.2 \\ P([S=1] \mid [R=0]) = 0.8 \\ P([S=0] \mid [R=1]) = 0.8 \\ P([S=1] \mid [R=1]) = 0.2 \end{cases} \\ P(G \mid RS) : \begin{cases} P([G=0] \mid [R=0]\,[S=0]) = 1 \\ P([G=1] \mid [R=0]\,[S=0]) = 0 \\ P([G=0] \mid [R=0]\,[S=1]) = 0 \\ P([G=1] \mid [R=0]\,[S=1]) = 1 \\ P([G=0] \mid [R=1]\,[S=0]) = 0 \\ P([G=1] \mid [R=1]\,[S=0]) = 1 \\ P([G=0] \mid [R=1]\,[S=1]) = 0 \\ P([G=1] \mid [R=1]\,[S=1]) = 1 \end{cases} \end{cases} \end{cases} \\ Qu : P(R \mid G=1) \end{cases}
$$

FIGURE 3.10 – Description du programme probabiliste pour l'exemple du *sprinkler* avec ProBT

## 3.4 Calcul stochastique

Après avoir passé en revue les bases de la théorie des probabilités et quelques méthodes de calcul probabiliste, nous allons voir dans cette partie les principes de la conception de circuits électroniques manipulant des données probabilistes.

Dès 1956, Von Neumann [34] pose les bases du concept du calcul stochastique par opposition au modèle déterministe. Mais c'est Gaines [19], en 1967, qui popularise le terme de calcul stochastique. L'idée est d'utiliser le caractère aléatoire des données directement au sein du calculateur. Tout d'abord parce que cela semble plus naturellement adapté aux calculs probabilistes. Ensuite parce que cette approche permet également de répondre aux problèmes classiques rencontrés par les fabricants de circuits, à savoir des circuits à bas coûts, à faible consommation et tolérants aux fautes (dues aux erreurs logicielles, aux attaques extérieures ou encore aux procédés de fabrication dans les salles blanches).

```
        # define the variables type
Booltype = plIntegerType(0,1)

        # define the variable for the submodel
Rain = plVariable('Rain', Booltype)
Sprinkler = plVariable("Sprinkler", Booltype)
GrassWet = plVariable("GrassWet", Booltype)

        # define the distribution for the submodel
PRain = plProbTable(Rain,[0.7,0.3]) # prior on raining probability
PSprinklerkRain = plCndDistribution(Sprinkler, Rain, [0.2,0.8,0.8,0.2])

PGrassWetkRainSprinkler
  = plCndDistribution(GrassWet, Rainˆ Sprinkler, [1, 0, 0, 1, 0, 1, 0, 1])

        # decomposition
model = plJointDistribution(Rainˆ Sprinklerˆ GrassWet,
                            PRain∗
                            PSprinklerkRain∗
                            PGrassWetkRainSprinkler)

        # observation
val = plValues(GrassWet)
val[GrassWet] = 1

        # question
question = model.ask(Rain, Grasswet)
```

FIGURE 3.11 – Spécification de l'exemple *sprinkler* en code ProBT

Les principaux problèmes liés à cette approche sont la faible précision des résultats et/ou le temps de calcul très long. C'est pourquoi les projets de telles machines, dans les années 70, ont été abandonnés au profit de la croissance des processeurs sur lesquels les algorithmes précédemment vus s'exécutaient de manière plus rapide et plus efficace, rendant obsolète ce type de conception.

Aujourd'hui, le calcul stochastique connaît un regain d'intérêt notamment grâce aux développements d'applications faisant de plus en plus appel à la théorie des probabilités comme l'apprentissage [33], les réseaux de neu-

rones [42, 38] et les problèmes d'inférences bayésiennes en général. Mais aussi parce que la miniaturisation des circuits (des transistors en particulier) entraîne de nombreux phénomènes aléatoires au sein des architectures [39].

### 3.4.1 Codage de l'information

Une valeur de probabilité, $p$, est un nombre tel que $p \in [0, 1]$. Il en existe donc une infinité. Avec les doubles précisions utilisées aujourd'hui pour représenter les nombres, on peut discrétiser l'ensemble $[0, 1]$ avec plusieurs millions de valeurs, ce qui laisse largement assez de valeurs possibles pour la plupart des problèmes probabilistes. Mais il faut alors s'assurer d'avoir assez de mémoire pour pouvoir stocker toutes les variables et avoir des composants de calcul qui opèrent sur des longues chaînes de bits (64 dans le cadre de la double précision).

Dans le modèle de Gaines, une donnée, c'est à dire une valeur de probabilité, est représentée par une chaîne de bits où le ratio du nombre de bits à 1 sur le cardinal de la chaîne code le nombre entre 0 et 1, comme montré dans l'équation 3.8.

$$\begin{aligned} 000000 &= 0 \\ 111111 &= 1 \\ 010110 &= \frac{3}{6} = \frac{1}{2} \end{aligned} \tag{3.8}$$

L'intérêt d'un tel codage est la simplicité de réalisation, en terme de portes logiques, des opérations arithmétiques de base comme la multiplication et l'addition (se reporter à la section 4.3.2).

On discrétise ainsi le nombre de valeurs possibles qui dépend de la taille des chaînes de bits utilisées. Le concept de mémoire est alors remplacé par des générateurs aléatoires de bits possédant le bon ratio. Si l'on souhaite représenter une valeur de probabilité de $\frac{4}{9}$ alors nous concevons un générateur de telle sorte que la probabilité qu'il émette un bit à 1 soit de $\frac{4}{9}$ à chaque pas d'horloge.

### 3.4.2 Précision, vitesse d'exécution

La précision d'une valeur de probabilité avec ce type de codage est directement liée à la taille de la chaîne utilisée, en considérant que l'on a un bon générateur de Bernoulli. En effet, le générateur aléatoire revient à faire un tirage de Bernoulli avec la probabilité p. C'est donc une loi binomiale de paramètre n et p qui génère la chaîne de n bits représentant la valeur

de probabilité. L'espérance d'une variable aléatoire X, sa variance et son écart-type suivant une telle loi sont donnés dans l'équation 3.9 :

$$\begin{aligned} E(X) &= np \\ V(X) &= np(1-p) \\ \sigma(X) &= \sqrt{V(X)} = \sqrt{np(1-p)} \end{aligned} \tag{3.9}$$

La précision vaut donc

$$Pr = \frac{\sqrt{np(1-p)}}{n} = \frac{\sqrt{p(1-p)}}{\sqrt{n}}$$

Cette précision est dépendante du ratio souhaité (valeur de p) et surtout de la taille de la chaîne de bits qui est en $O(\sqrt{n})$. Cette convergence en $O(\sqrt{n})$ quantifie la « lente convergence » décriée pour cette méthode de représentation d'une valeur de probabilité.

### 3.4.3 Générateurs de nombres aléatoires

#### Principe

La qualité des générateurs de nombres aléatoires est un des points clés de l'utilisation du codage en chaîne de bits d'une donnée. Chaque variable doit être indépendante et décorrélée dans le temps pour assurer un résultat cohérent sur les opérations arithmétiques.

Il existe deux grands types de générateurs de nombres aléatoires, les pseudo aléatoires *PRNGs* (*pseudo random number generator*) et les *TRNGs* (*true random number generator*). Ces derniers émanent de sources physiques possédant intrinsèquement de l'entropie comme le bruit thermique par exemple. Les générateurs pseudo-aléatoires sont créés de manière algorithmique, et sont, en ce sens, déterministes.

La plupart des *PRNGs* uniformes sont définis de telle sorte que l'on tire $u \in [0, 1]$ comme suit :

$$u = \frac{X_n}{T}$$

où les éléments $X_n \in [0, T-1]$ sont générés.Pour cela, on peut par exemple utiliser la formule de récurrence suivante :

$$X_{n+1} = (aX_n + b)\ [T] \tag{3.10}$$

En choisissant des valeurs spécifiques de $X_0$, a, b et T, on est capable de générer des séquences de période de plus de $2^{30}$. Les *PRNGs*, de par leur construction algorithmique, possèdent toujours une périodicité puisqu'ils lient l'élément $X_{n+1}$ aux précédents $X_k$ générés. P.L'Écuyer dans [25] propose une formalisation mathématique de la définition des *PRNGs*.

Dans un premier temps, nous avons choisi d'utiliser des *Linear Feedback Shift Registers LFSRs* [20] pour leur faciliter d'implémentation matérielle. En effet, ceux-ci ne requièrent que des portes logiques simples XOR et des registres. Les études de Marsaglia [27] ont montré le lien entre les polynômes caractéristiques des matrices mis en jeux dans l'élaboration du *LFSR* et la période maximale de celui-ci.

Les *LFSRs* 32 bits implémentés ont une période maximale de $T = 2^{32}-1$. En plus de leur qualité d'aléa médiocre, ils possèdent donc une périodicité qui est contraignante pour certaines de nos applications présentées dans ce manuscrit. Nous avons utilisé des générateurs à la périodicité bien plus grande comme Mersenne-Twister [29] où $T = 2^{19937} - 1$. On peut aussi citer d'autre algorithme comme celui de Tausworthe [44] qui possède une période de $T = 2^{88} - 1$.

Contrairement aux *PRNGs*, les *TRNGs* tirent leur aléa de sources physiques et seraient donc non-déterministes. En ce sens, ils semblent être de meilleurs candidats pour la génération de nombres aléatoires. Dans nos machines, un *TRNG* à forte vitesse, comme dans [10], pourra être utilisé comme générateur aléatoire distribué sur toutes les entrées en nécessitant. Mais étonnamment, il a été montré que les générateurs pseudo-aléatoires étaient non seulement suffisants mais posséderaient même un comportement qui sied mieux à la logique et aux calculs stochastiques [4].

La qualité de ces générateurs est déterminé par la dépendance qu'il existe entre deux bits consécutifs ainsi que la longueur du cycle sans redondance, c'est à dire la période du générateur. En plus de leur performance stochastique, il faut ajouter leur performance en vitesse et/ou leur coût. En effet, dans un problème d'inférence contenant beaucoup de variables il faudra une quantité importante de nombres aléatoires. Des travaux ont déjà été réalisés pour tirer profit d'une source aléatoire afin d'alimenter plusieurs valeurs de probabilités [36]. L'utilisation de générateurs aléatoires en parallèles nécessite cependant une attention particulière. Les difficultés liées à cette méthode ont déjà été mis en évidence dans [43].

### Les générateurs utilisés

Pour générer les flux binaires sur les machines, nous utilisons les travaux de Gupta & Kumaresan [22, 4], qui ont proposé un circuit capable de générer une chaîne de bits stochastiques $B(p)$ encodant une probabilité $p$ stockée dans un format à virgule fixe $p = \sum_{i=1}^{i=n} x_i 2^{-i}$ dans un buffer $X$ (voir la partie rouge, en haut, de la figure 3.12).

Nous proposons de compléter ce circuit par une mémoire adressable (voir partie bleue, en bas, de la figure 3.12). Soit une adresse $Y$, le contenu de la mémoire à cette adresse, qui code une valeur probabilité $p_Y$, est écrit dans $X$. La chaîne de bits générée $B(p_Y)$ est alors conditionnée par la valeur $Y$.

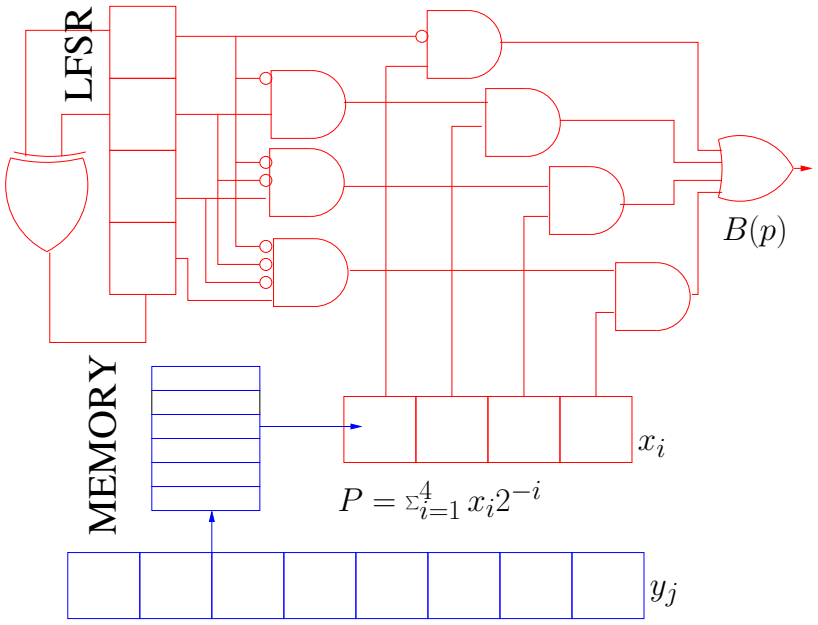



FIGURE 3.12 – Implémentation matérielle du générateur du flux binaire codant une valeur de probabilité

Sur la figure 3.12, nous montrons que nous avons utilisé un LFSR pour générer les bits aléatoires nécessaires. Comme dit dans la section 3.4.3, dans certaines applications la qualité de l'aléa d'un LFSR n'est plus suffisante, nous avons alors implémenté un Mersene-Twister [29] qui s'est montré suffisamment efficace pour nos expériences sur FPGA. Le domaine des générateurs aléatoires étant à lui seul un pan foisonnant de la recherche, notre expertise dans ces travaux s'est limitée à trouver des générateurs aléatoires suffisamment performants. Une des perspectives de recherche est le dévelop-

pement de nos connaissances sur le sujet ainsi que l'étude de l'intégration des MTJs comme source entropique pour de telles machines. Dans le cadre de nos travaux, nous nous sommes également intéressés aux *TRNGs* (*True Random Number Generator*) tel que [10], qui possède des qualités d'aléa extrêmement performantes mais aussi une vitesse de génération de bits très élevée. Ceci pourrait nous permettre de n'utiliser qu'un seul *BSG*, pour *Bit Stream Generator*, et de disperser les bits aléatoires sur toutes les entrées nécessitantes.

## 3.5 Comparaison de distributions

Les deux types de machine probabiliste présentés dans les chapitres suivants diffèrent dans leur méthode de calcul de l'inférence. Les *SMTBIs* utilisent l'inférence exacte comme méthode de résolution. Cependant, le résultat n'est pas exact puisque les *SMTBIs* utilisent le calcul stochastique comme méthode de calcul. En cela, la distribution calculée est une approximation de la distribution visée. Le constat est le même pour les machines *SMABIs* mais pas pour les mêmes raisons. En effet, celles-ci implémentent une méthode d'échantillonnage et ne peuvent donc offrir que des approximations des distributions cherchées de par leur méthode de résolution.

Dans les deux cas, nous avons besoin d'analyser quantitativement l'approximation qui est faite par ces machines. Pour cela, nous utilisons la *RMSE*, pour *Root Mean Square Error*, qui est l'erreur quadratique moyenne entre deux distributions ainsi que la divergence de Kullback-Leibler (*DKL*). La différence entre ces deux types de comparaison est la prise en compte des erreurs sur les faibles valeurs de probabilités dans le cas de la *DKL* grâce à la fonction logarithme. En effet, de par sa formule, la *RMSE* sera peu impactée par des différences sur des faibles valeurs de probabilités.

### 3.5.1 *RMSE : Root Mean Square Error*

Soit $P(X_{ex})$ le résultat expérimental calculé par une machine et $P(X_{th})$ le résultat théorique souhaité. Nous calculons la *RMSE* en considérant plusieurs expériences conduisant à plusieurs $P(X_{ex})$ et moyennant la somme des carrés des différences entre les valeurs théoriques et expérimentales, comme montré sur l'équation 3.11. La moyenne est indiquée par les symboles "< >".

$$RMSE\ (P(X_{th}), P(X_{ex})) =< \sqrt{\sum_{x^i \in X} (P(x^i_{th}) - P(x^i_{ex}))^2} > \qquad (3.11)$$

### 3.5.2 DKL : divergence de Kullback-Leibler

Soit $P(X_{ex})$ le résultat expérimental calculé par une machine et $P(X_{th})$ le résultat théorique souhaité. Nous calculons la *DKL* comme montré sur l'équation 3.12.

$$DKL\ (P(X_{th}), P(X_{ex})) = \sum_{x^i \in X} P(x^i_{th}) \log \left( \frac{P(x^i_{th})}{P(x^i_{ex})} \right) \tag{3.12}$$

## 3.6 Conclusion

Nous avons donné les définitions des principaux concepts utilisés dans ces travaux de thèse et énoncé les propriétés et théorèmes liés au domaine des probabilités et de l'inférence probabiliste. La présentation de l'inférence exacte et de ses limites nous permet de comprendre pourquoi les machines *SMTBIs* du chapitre 4 sont conçues de cette façon et pourquoi elles ne peuvent adresser qu'une classe de problèmes restreinte. Les machines *SMABIs*, présentées dans le chapitre 5, implémentent une méthode de résolution approchée de l'inférence. Il nous a alors paru opportun de revoir certaines méthodes d'échantillonnage et d'expliciter l'algorithme de Gibbs qui est utilisé dans nos machines. Le fonctionnement des machines étant basé sur l'interprétation de programmes ProBT, il est nécessaire de savoir programmer avec ce langage probabiliste. Celui-ci nous permet également d'avoir les résultats des inférences des applications présentées et ainsi étudier l'efficacité de nos machines. Enfin, nous avons présenté le calcul stochastique qui est le modèle de calcul choisi pour les deux types de machine. Cette partie comprend la présentation des *BSGs* utilisés comme source d'entropie nécessaire à nos machines.

# Chapitre 4

# Machine bayésienne stochastique pour la résolution de problèmes d'inférence exacte

Dans ce chapitre est présenté la conception d'une machine bayésienne traitant l'inférence exacte, appelée *SMTBI* pour *Stochastic Machine for Tractable Bayesian Inference*. Pour cela, nous utilisons la formule de Bayes et nous réalisons tous les calculs nécessaires : sommes, produits et division avec une arithmétique stochastique.

Nous présentons dans les sections suivantes les principes mathématiques, les principes du codage de l'information et les principes d'architecture matérielle sur lesquels sont fondées de telles machines.

Celles-ci réalisent le calcul d'une inférence pour un problème bayésien spécifique. Afin de rendre générique l'implémentation de tels circuits, un synthétiseur logique transforme n'importe quel programme probabiliste ProBT en un fichier VHDL qui spécifie le circuit réalisant l'inférence.

Nous développons trois exemples d'application comme illustration de cette chaîne de synthèse. Un exemple académique simple sur trois variables est utilisé comme preuve de concept. Les filtres bayésiens sur le problème de la synchronisation de *LFSRs* pour l'acquisition de séquences pseudo bruitées (*Pseudo noise (PN) sequence acquisition*) en télécommunications et la fusion de capteurs, pour permettre l'évitement d'obstacle d'un robot autonome, sont deux applications où l'on montre l'efficacité des machines *SMTBI*.

## 4.1 Problématique

A ce jour, le calcul d'inférence peut être réalisé par n'importe quel ordinateur en utilisant les outils logiciels de programmation probabiliste cités dans le chapitre précédent. Ces calculs sont effectués sur des processeurs standards qui utilisent une arithmétique flottante et une architecture matérielle de Von Neumann. Nous avons mentionné précédemment que l'augmentation de la puissance de calcul nécessite des progrès constants de la part des fondeurs pour réduire la taille des composants. La difficulté de maintenir ce qu'on appelle la loi de Moore entrave également les performances en efficacité énergétique des circuits. Dans le cadre spécifique des inférences bayésiennes, il semble naturel de penser que des calculateurs déterministes standardisés ne siéent pas au mieux aux problèmes où l'aléa est une donnée fondamentale du problème.

La complexité des calculs nécessaires pour l'inférence exacte est liée au temps que ceux-ci prennent pour être réalisés. En effet, les problèmes que l'on peut traiter en inférence exacte sont des problèmes où le temps de calcul est assez réduit pour que le résultat soit disponible dans un temps raisonnable. Dans le cas d'un traitement robotique par exemple, le temps de calcul raisonnable correspond au temps maximal entre deux commandes.

Nous pensons que le calcul stochastique, qui intègre l'aléatoire dans son concept, est plus approprié pour résoudre les problèmes probabilistes. La machine présentée dans ce chapitre est une première initiative qui va dans ce sens.

## 4.2 Principe

Les arbres de calcul et les algorithmes tels que le *Successive Reductions Algorithm* (SRA) [30] sont utilisés pour réduire et simplifier les expressions d'inférence à calculer. Grâce à ces algorithmes, ProBT génère donc une expression symbolique qui correspond à l'arbre de calcul simplifié pour réaliser le calcul d'inférence. A partir de cette expression, nous avons développé un synthétiseur logique qui génère un code VHDL (*VHSIC Hardware Description Language, VHSIC : Very High Speed Integrated Circuit*) spécifiant la machine. Cette spécification correspond à associer à chaque produit et somme, présents dans l'expression, le composant correspondant. La description matérielle de la machine qui réalise le calcul d'inférence est donc générée automatiquement.

Nous avons choisi la discrétisation du codage temporel par des flux binaires pour représenter les valeurs de probabilités. L'intérêt d'un tel codage est la simplicité des opérateurs logiques pour les opérations. Ceci permet non seulement de se soustraire à une représentation en arithmétique flottante, mais aussi nous verrons que ce type de circuit n'a besoin que d'un nombre réduit de composants électroniques. Le langage de programmation ProBT est utilisé pour décrire le problème d'inférence bayésienne puis le synthétiseur probabiliste de haut niveau spécifie le circuit correspondant. Cette architecture de circuit est décrite dans le langage matériel VHDL. Ce fichier VHDL est utilisé pour les simulations et l'implémentation sur FPGA (Field-Programmable Gate Array), il peut également servir de base pour la conception de circuits ASIC (*Application-Specific Integrated Circuit*) si l'on souhaite le fabriquer.

## 4.3 Architecture de calcul

Dans cette section, nous présentons le codage de l'information et les opérateurs de base permettant le calcul de l'inférence.

### 4.3.1 Bus stochastique

Tout nombre rationnel $r \in [0, 1]$ peut être encodé par une chaîne de bits. Soit $N_1$ le nombre de bits à 1 dans une chaîne de bits de taille $N$, le nombre rationnel $r$ correspondant est la valeur $\frac{N_1}{N}$.

***Explication*** : Soit une chaîne de bits de cardinal $N$ ayant $N_1$ bits à 1. Alors la fraction $\frac{N_1}{N}$ est un nombre rationnel $r \in [0, 1]$. La réciproque est vraie également, soit un nombre rationnel $r \in [0, 1]$, alors on peut trouver une chaîne de bits tel que $\frac{N_1}{N} \to r$ quand $N \to \infty$.

De la même manière qu'un bus de données dans une architecture standard de circuit, on définit une variable probabiliste discrète X comme son ensemble de valeurs $x_i$ de sa distribution. Chaque valeur $x_i$ est codée par sa chaîne de bits, $c_i$. On obtient donc en normalisant : $P(X = x_i) = \frac{c_i}{\sum_{j=1}^{n} c_j}$. Alors, une distribution sur une variable X peut être encodée avec un ensemble de $n = \text{card}(X)$ chaînes de bits stochastiques qui forment ce qui est appelé un « bus probabiliste » ou « bus stochastique ».

### 4.3.2 Opérateurs de bases

Le calcul d'une inférence requiert trois opérations : le produit de probabilité, la marginalisation (addition) et la normalisation (division). Avec la représentation des données probabilistes choisie, de simples composants électroniques suffisent à réaliser ces opérations.

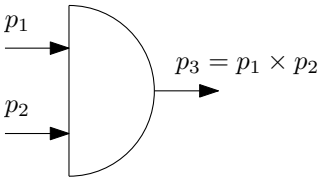
#### Produit : Porte AND

La plus simple des opérations à réaliser avec des chaînes de bits est le produit. Multiplier deux probabilités $p_1$ et $p_2$ encodées par deux chaînes indépendantes $c_1$ et $c_2$ n'utilise qu'une simple porte logique AND, figure 4.1.

En effet, soit $p_1$ la probabilité d'avoir un 1 dans la chaîne $c_1$ et $p_2$ la probabilité d'avoir un 1 dans la chaîne $c_2$. Alors, à la traversée d'une porte AND, l'événement « avoir 1 en sortie de la porte AND », qui correspond à la probabilité $p_3$, est réalisé si et seulement si les deux entrées $e_1$ et $e_2$ de la porte valent 1. Si les deux chaînes sont indépendantes, alors la probabilité conjointe est le produit des probabilités comme suit :

$$\begin{aligned} &P(\text{"avoir 1 sur l'entrée } e_1\text{"} \cap \text{"avoir 1 sur l'entrée } e_2\text{"}) \\ &\qquad = P(\text{"avoir 1 sur l'entrée } e_1\text{"}) \times P(\text{"avoir 1 sur l'entrée } e_2\text{"}) \\ &\qquad = p_1 \times p_2 \end{aligned}$$

D'où le fait que la chaîne $c_3$, en sortie de la porte AND, encode la probabilité $p_3 = p_1 \times p_2$.




| AND | | |
|---|---|---|
| $A$ | $B$ | $S$ |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

FIGURE 4.1 – Produit de probabilités avec porte AND

L'indépendance entre les chaînes $c_1$ et $c_2$ est fondamentale. En effet, plus les chaînes sont corrélées et plus le biais sur le résultat sera important.

**Calcul de la corrélation**

Dans le cas général, la corrélation entre deux variables X et Y discrètes est donné par l'équation 4.1 :

$$\begin{aligned}\rho(XY) &= \frac{Cov(XY)}{\sigma_X \sigma_Y} \\ &= \frac{E[(X - E(X))(Y - E(Y))]}{\sigma_X \sigma_Y} \\ &= \frac{E(XY) - E(X)E(Y)}{\sigma_X \sigma_Y}\end{aligned} \tag{4.1}$$

Avec :

— $\rho(XY)$ est la corrélation entre les variables X et Y.
— $Cov(XY)$ signifie covariance entre les variables X et Y.
— $E(X)$ représente l'espérance de la variable X.
— $\sigma_X$ représente l'écart type pour la variable X.

Dans le cas d'une corrélation maximale, on peut imaginer deux chaînes $X = c_1$ et $Y = c_2$ qui encodent la même probabilité $p_1 = p_2 = p < 1$. Si $p$ est représentée par la même chaîne, c'est à dire $c_1 = c_2$, dans ce cas on a $\rho(XY) = 1$, alors la chaîne résultante en sortie de la porte AND encode la probabilité $p$ et non pas le résultat souhaité $p^2$.

Pour pallier ce problème, il y a deux types d'indépendance à vérifier. L'indépendance entre les chaînes qui encodent les probabilités mais aussi l'indépendance bit à bit sur une même chaîne. La première indépendance a été présentée juste au dessus. Pour la seconde, l'exemple suivant montre son importance également. Soit deux chaînes :

$$\begin{aligned}c_1 = 0000011111 \ \text{ encodant } \ p_1 = \frac{1}{2} \\ c_2 = 1111000000 \ \text{ encodant } \ p_2 = \frac{2}{5}\end{aligned}$$

On note $p_{th}$ la probabilité théorique du produit des probabilités, $c_s$ la chaîne résultant du passage des chaînes $c_1$ et $c_2$ à travers une porte AND et $p_s$ la probabilité associée à $c_s$. Sur ce simple exemple, les résultats suivants

montrent le problème lié à la non indépendance bit à bit :

$$p_{th} = \frac{1}{5}$$
$$c_s = 0000000000$$
$$p_s = 0$$

Il faut donc prêter une attention particulière à la qualité des générateurs aléatoires qui sont des composants fondamentaux dans la conception des machines présentées. Nous verrons dans le chapitre 6 que des générateurs à faible qualité d'aléa, type *LFSR*, se montre insuffisant pour certaines applications.

### Addition pondérée : multiplexeur

Soit deux chaînes de bits $c_1$ et $c_2$, indépendantes, encodant les probabilités respectives $p_1$ et $p_2$. Si ces chaînes traversent un multiplexeur, ayant un sélecteur stochastique non biaisé ($p = 0.5$), alors la chaîne résultante $c_s$ encode la probabilité $p_s = \frac{p_1+p_2}{2}$. On peut étendre ce résultat à $n$ chaînes indépendantes $c_1 \dots c_n$ encodant les probabilités $p_1 \dots p_n$, alors la chaîne résultante encode la probabilité $p_s = \frac{\sum_{i=1}^{n} p_i}{n}$, comme schématisé sur la figure 4.2. En effet, l'événement « obtenir 1 en sortie du multiplexeur » est l'intersection des événements « avoir un 1 sur l'entrée $e_i$ » et « avoir été choisie par le sélecteur » d'où $p_s = \frac{p_1}{n} + \frac{p_2}{n} + ... + \frac{p_2}{n} = \frac{\sum_{i=1}^{n} p_i}{n}$.

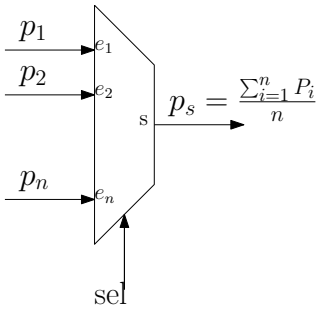



FIGURE 4.2 – Somme moyennée avec multiplexeur

Cependant, avec cette méthode de sommation de probabilités, nous devons faire face au problème de dilution temporelle. Effectivement, avec les notations précédentes, pour $n$ signaux en entrée d'un multiplexeur, $p_s = \frac{p_1+p_2+...+p_n}{n}$. Le diviseur $n$ réduit inéluctablement le nombre de 1 dans le

flux binaire résultant. Ce problème devient de plus en plus important lorsque l'inférence nécessite le calcul de sommes imbriquées. Par exemple, si l'inférence nécessite la cascade de quatre sommes avec $n_1 = 5$, $n_2 = 10$, $n_3 = 8$ et $n_4 = 10$, alors le résultat sera divisé par 4000 ! Dans ce cas, le flux binaire résultant devra être observé pendant un laps de temps très long afin d'obtenir une approximation suffisante de la valeur de probabilité encodée par cette chaîne de bits. Pour résoudre ce problème, nous avons conçu un autre additionneur utilisant une simple porte OR combinée avec un compteur.

### Addition : *OR+*, Porte OR avec mémoire

La porte logique OR permet l'addition de probabilités encodées par des chaînes de bits, avec un biais du fait qu'elle « oublie » de temps en temps de compter un 1. Au vu de sa table de vérité figure 4.1 (gauche), le problème apparaît lorsque les deux bits en entrées valent 1. En effet, on devrait compter deux 1 alors que la sortie de la porte ne donne qu'un seul bit à 1. On peut calculer l'erreur moyenne sur la valeur théorique puisqu'il s'agit d'obtenir 1 à chaque entrée soit la probabilité $p_1 \times p_2$. Pour remédier à ce biais, on utilise une mémoire, sous la forme d'un compteur qui va s'incrémenter lorsque la condition « avoir deux 1 en entrée » est réalisée et qui va décrémenter lorsque la condition « avoir deux 0 en entrée et un compteur non nul » est réalisée. Dans ce cas, la sortie du composant ne sera pas 0, comme l'est la sortie de la porte OR, mais 1 qui correspond à l'ajout d'un 1 « oublié » au préalable. Nous nommons ce composant *OR+*.

| *inputs* | output | $R = 0$ |
|---|---|---|
| 00 | 0 | 0 |
| 01 | 1 | 0 |
| 10 | 1 | 0 |
| 11 | 1 | 1 |

| *inputs* | output | $R \neq 0$ |
|---|---|---|
| 00 | 1 | $R - 1$ |
| 01 | 1 | $R$ |
| 10 | 1 | $R$ |
| 11 | 1 | $R + 1$ |

TABLE 4.1 – Tables de vérité des portes OR et OR+. La table de gauche est la table de vérité de la porte OR standard et la table de droite est la table de vérité de la porte OR avec mémoire (OR+). Dans la table OR+, la colonne de droite contient la valeur du compteur mise à jour.

### Division : $JK$ flip flop

La figure 4.3 présente un schéma de la bascule $JK$ et sa table de vérité. Elle se comporte comme une porte XOR lorsque les entrées sont distinctes.

Lorsque les deux entrées sont à 0, le bit de sortie vaut le bit précédent, et lorsque les deux entrées sont à 1, le bit de sortie vaut le conjugué du bit précédent. C'est ce composant qui permet de réaliser la division.

***Explication*** : Soit $P_A$ et $P_B$, les probabilités encodées par des chaînes de bits placées aux entrées $J$ et $K$ de la bascule $JK$. Soit $P_C$ la probabilité qu'un bit en sortie de la bascule $JK$ vaille 1. La probabilité qu'un bit de sortie passe de l'état 1 à 0 au temps $t$ vaut :

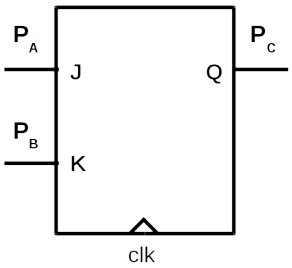
$$P_C((1-P_A)P_B + P_A P_B) = P_C P_B$$

De la même manière, la probabilité d'avoir la transition 0 à 1 en sortie de la bascule vaut :

$$(1-P_C)(P_A(1-P_B) + P_A P_B) = (1-P_C)P_A$$

A l'équilibre, les transitions 1 vers 0 et 0 vers 1 sont égales (cf *Detailled balance equation*). On obtient donc le résultat suivant : $P_C P_B = (1-P_C)P_A$, c'est à dire :

$$P_C = \frac{P_A}{P_A + P_B}$$




| Table de vérité de la JK flip-flop | | |
|---|---|---|
| $J$ | $K$ | $Q$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 0 | hold |
| 1 | 1 | reverse |

FIGURE 4.3 – Table de vérité et schéma de la bascule $JK$ utilisée comme opérateur de division

### 4.3.3 Normalisation

A partir de ce composant, nous n'avons pas explicitement accès à la fraction $\frac{P_A}{P_B}$ mais nous avons une manière de normaliser les résultats des autres opérations.

En effet, soit $Z$ la somme des résultats d'une distribution de probabilité. Si on fait entrer une chaîne $c_A$, encodant la probabilité $p_A$, dans l'entrée $J$

et une chaîne $c_B$, encodant la probabilité $p_B = Z - p_A$, dans l'entrée $K$, alors la chaîne résultante en sortie de la bascule $JK$ encode la probabilité $p_C = \frac{p_A}{Z}$.

Le calcul de $Z$ est réalisé en sommant toutes les valeurs de la distribution grâce au composant $OR+$ décrit dans la section 4.3.2. La soustraction est réalisée en utilisant une porte logique XOR : $p_B = Z \oplus p_A$. Ce composant permet de «retirer» tous les "1" de Z, apportés par $p_A$ via $c_A$. Dans ce cas, les chaînes $c_A$ et $c_B$ sont très fortement corrélées. Alors, pour conserver l'indépendance entre ces chaînes, nous utilisons un décalage (*shift*) d'un bit sur la chaîne $c_B$. Sous réserve d'avoir un générateur aléatoire assez performant pour avoir une corrélation bit à bit très faible, voire nulle, les chaînes $c_A$ et $c_B$ sont alors décorrélées.

## 4.4 Synthèse probabiliste de haut niveau

### 4.4.1 Spécification de la machine à partir d'un programme bayésien

Une machine bayésienne peut être rigoureusement définie en utilisant le formalisme de programmation bayésienne décrit dans [7]. Une telle machine réalise un calcul d'inférence sur un modèle bayésien. Elle est homogène dans le sens où ses entrées sont des distributions de probabilité, appelés inférences molles (*soft inferences*), et sa sortie est également une distribution de probabilité sur les variables cherchées. Un synthétiseur logique a été développé permettant de transformer n'importe quelle inférence spécifiée dans le langage de programmation ProBT en une description du circuit électronique qui réalise le calcul de l'inférence. L'architecture de tels circuits est présentée dans la section 4.4.2.

Dans le cas général, nous avons vu que l'on peut considérer une distribution de probabilité conjointe sur un ensemble de variables discrètes comme ci : $P(S \wedge D \wedge F)$. Les variables S (pour *searched*, variables cherchées), D (pour *Data*, variables observées, connues) et F (pour *free*, variables libres) peuvent elles-mêmes être des conjonctions de variables, par exemple : $D = D_1 \wedge \ldots \wedge D_k$. Nous définissons les *soft evidences*, $\tilde{P}(D_k)$, sur les variables $D_k$ comme une distribution sur cette variable. Ce sont ces *soft evidences* qui seront les entrées de la machine ainsi définie. A partir de la définition de ces *soft evidences* et de la conjointe $P(S \wedge D \wedge F)$, on réalise l'inférence qui répond à cette question : quelle est la distribution de probabilité sur les

variables cherchées $S$ ?

$$P(S) = \frac{1}{Z} \sum_{D_1} \tilde{P}(D_1) \dots \sum_{D_k} \tilde{P}(D_k) \sum_{F} P(S \wedge D \wedge F) \ , \tag{4.2}$$

avec le terme Z qui correspond à la normalisation :

$$Z = \sum_{S} \left( \sum_{D_1} \tilde{P}(D_1) \dots \sum_{D_k} \tilde{P}(D_k) \sum_{F} P(S \wedge D \wedge F) \right) \tag{4.3}$$

La machine est donc un circuit électronique qui réalise le calcul de l'inférence sur n'importe quel modèle bayésien spécifié par sa conjointe $P(S \wedge D \wedge F)$.

Cette machine est décrite dans un langage de description matérielle grâce aux spécifications de la distribution de probabilité conjointe, c'est-à-dire sa décomposition, et la spécification de ses entrées et sorties dans le langage ProBT.

Le chaîne de compilation proposée accepte tout programme ProBT avec des variables discrètes comme entrées. Cette synthèse génère un fichier VHDL qui est la description du circuit stochastique et peut être implémenté sur un FPGA (Field-Programmable Gate Array). Une carte Cyclone IV FPGA Altera a été ciblée en tant que dispositif de support électronique. Un exemple simple de machine a été synthétisé pour démontrer l'applicabilité et l'évolutivité de la chaîne de synthèse proposée. Par ailleurs, ProBT est également utilisé pour calculer le résultat en inférence exacte en utilisant l'arithmétique flottante sur des processeurs standards. Cela permet d'évaluer la qualité de l'approximation réalisée avec le circuit synthétisé sur la carte FPGA. La figure 4.4 [1] résume le fonctionnement de la chaîne de synthèse et des scripts disponibles pour l'automatisation de la génération des circuits *SMTBIs* et des analyses de résultats.

### 4.4.2 Un exemple simple

Nous utilisons un exemple simple à trois variables pour présenter l'outil de synthèse développé. Soit le programme suivant, figure 4.5, qui spécifie un modèle bayésien à deux variables connues $D_1, D_2$ et une variable cherchée que l'on note $M$ pour *Model*.

1. Cette figure est tirée du délivrable ®Probayes pour le projet BAMBI réalisé par Raphaël Laurent

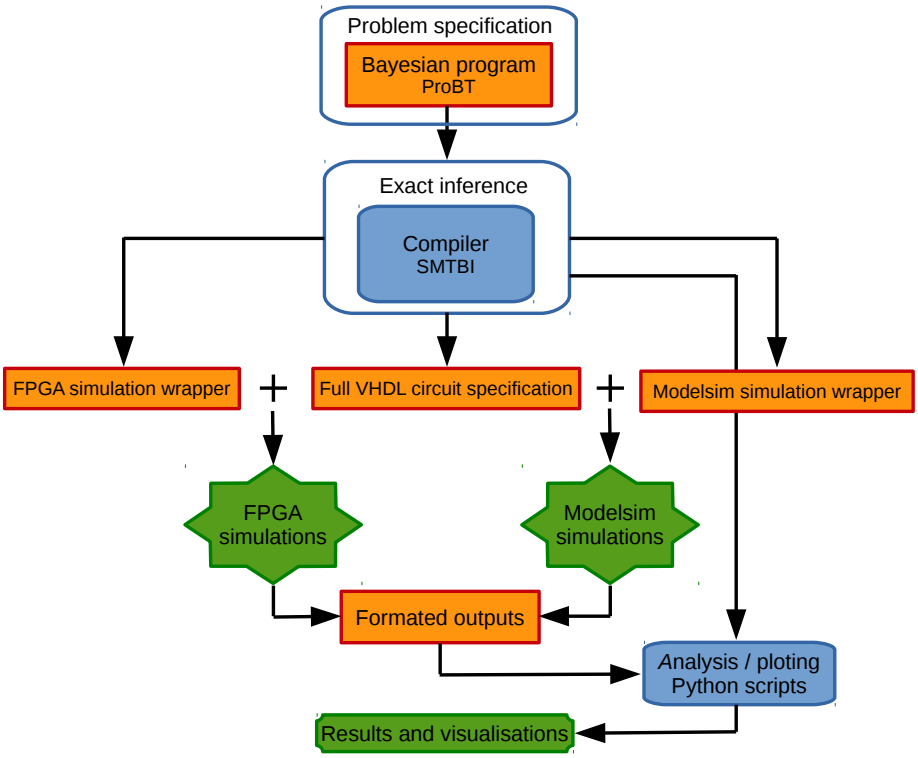



FIGURE 4.4 – Schéma de la chaîne de synthèse et des outils d'analyse de performance

Ce programme exprime explicitement les hypothèses d'indépendances conditionnelles permettant de simplifier le calcul de la distribution de probabilité conjointe (voir figure 4.5, ligne 25) :

$$P(M \wedge D_1 \wedge D_2) = P(M)P(D_1|M)P(D_2|M)$$

Il précise également la sortie $S$ et les entrées $D_1$ et $D_2$ de la machine. Aucune variable libre $F$ n'est spécifiée dans cet exemple, mais elles peuvent être décrites également, nous en verrons un exemple sur l'application décrivant un filtre bayésien dans la section 4.5.1.

La dernière instruction de ce programme *question = model.ask(M)* produira une simplification interne de l'expression (4.2) qui minimise la charge de calcul en réduisant le nombre de sommes en fonction de la structure de la décomposition de la distribution. L'algorithme utilisé permettant cette simplification est le *Successive Reductions Algorithm* (SRA) [30], qui est un

algorithme similaire à l'algorithme *sum-product* [23]. Dans le programme présenté figure 4.5, étant donné que la distribution de probabilité conjointe est

$$P(M \wedge D_1 \wedge D_2) = P(M)P(D_1|M)P(D_2|M) \ ,$$

le SRA va transformer l'équation (4.2) en :

$$P'(M) = \frac{1}{Z}P(M)\sum_{D_1}\tilde{P}(D_1)P(D_1|M)\sum_{D_2}\tilde{P}(D_2)P(D_2|M) \tag{4.4}$$

La figure 4.6 présente l'architecture haut niveau de la machine bayésienne. Elle comprend l'appareil stochastique principal qui génère les flux binaires aléatoires nécessaires qui encodent les valeurs de probabilité des constantes considérées dans le problème. Les générateurs de nombres aléatoires (*RNGs*) produisent des flux binaires uniformes. Ces générateurs sont alors biaisés de façon à délivrer en sortie les valeurs de probabilité souhaitées. Ces constantes représentent les connaissances du modèle.

### Synthèse logique de l'exemple simple

Dans cette section nous décrivons plus en détail comment l'équation (4.4) est évaluée en termes de composants logiques et la mise en œuvre du circuit.

Chaque distribution de probabilité de l'expression (4.4) est codée avec un bus stochastique. Nous distinguons les *soft* évidences, qui sont les entrées de la machine probabiliste, des autres distributions de probabilité de la décomposition de la conjointe. Les entrées peuvent changer au cours du temps alors que les autres distributions sont fixées par le programmeur en tant que paramètres internes lors de la spécification du problème d'inférence, c'est-à-dire la spécification de la distribution conjointe, dans ProBT. Les *RNGs* biaisés sont utilisés pour produire les bus probabilistes associés à ces distributions, voir la figure 4.6. Par exemple, dans l'expression (4.4), $P(D_1|M = m)$ est représenté par un bus stochastique avec trois signaux stochastiques, comme montré dans l'équation 4.5 suivante :

$$\begin{aligned} c_0 &\propto P(d_1^0|m) \\ c_1 &\propto P(d_1^1|m) \\ c_2 &\propto P(d_1^2|m) \end{aligned} \tag{4.5}$$

où $d_1^i \Leftrightarrow D_1 = i$. Ainsi, trois *RNGs* biaisés sont utilisés pour produire les flux binaires stochastiques représentant $c_0$, $c_1$ et $c_2$.

```
1:  import the ProBT bindings
2:  from pypl import *
3:  ## define the variables
4:  dim3 = plIntegerType(0, 2)
5:  D_1 = plSymbol('D'_1, dim3)
6:  D_2 = plSymbol('D'_2, dim3)
7:  M = plSymbol('M', dim3)
8:
9:  ## define the distribution on M
10: P_M = plProbTable(M, [0.8, 0.1, 0.1])
11:
12: ## define a conditional distribution on D_1
13: P_D1_k_M = plDistributionTable(D_1, M)
14: P_D1_k_M.push(plProbTable(D_1, [0.5, 0.2, 0.3]), 0)
15: P_D1_k_M.push(plProbTable(D_1, [0.5, 0.3, 0.2]), 1)
16: P_D1_k_M.push(plProbTable(D_1, [0.4, 0.3, 0.3]), 2)
17:
18: ## define a conditional distribution on D_2
19: P_D2_k_M = plDistributionTable(D_2, M)
20: P_D2_k_M.push(plProbTable(D_2, [0.2, 0.6, 0.2]), 0)
21: P_D2_k_M.push(plProbTable(D_2, [0.6, 0.3, 0.1]), 1)
22: P_D2_k_M.push(plProbTable(D_2, [0.3, 0.6, 0.1]), 2)
23:
24: ## define the joint distribution
25: model = plJointDistribution(P_M * P_D1_k_M * P_D2_k_M)
26:
27: ## define the soft evidence variables
28: model.set_soft_evidence_variables(D_1 ∧ D_2)
29:
30: ## define the output
31: question = model.ask(M)
```

FIGURE 4.5 – Spécification d'un programme bayésien sur un exemple simple en utilisant les *bindings* Python pour ProBT. Il comprend la définition des variables, la définition de la distribution de probabilité conjointe sur toutes les variables considérées comme un produit de distributions simples (voir la ligne 25), la spécification des valeurs pour les paramètres de ces distributions, et une question posée au modèle (voir la ligne 31), qui sera résolue par inférence bayésienne

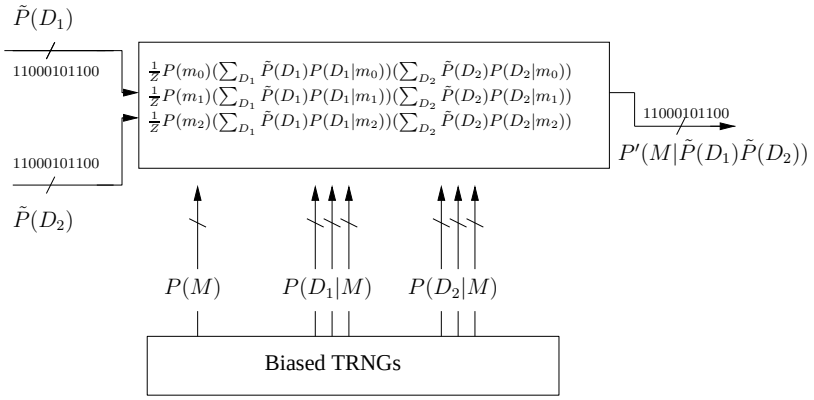



FIGURE 4.6 – Schéma de l'architecture d'une machine *SMTBI* : ses entrées sont les *soft* évidences (flèches entrant à gauche) et sa sortie est la distribution de probabilité sur la variable d'intérêt $M$. Les valeurs de probabilité des paramètres internes des termes de la distribution conjointe sont fixées et lese chaînes de bits émanent des *RNGs*.

La compilation produit deux résultats fondamentaux. Premièrement, en utilisant ProBT, elle produit l'expression à calculer qui correspond à l'inférence à réaliser (voir l'expression (4.4) générée par la ligne 31 du programme figure 4.5). Puis, par analyse syntaxique de la formule obtenue, elle génère une description matérielle du circuit effectuant les sommes et les produits nécessaires en utilisant les opérateurs logiques décrits précédemment. Le langage VHDL est utilisé pour décrire l'architecture d'un tel circuit. De ce fait, par la nature de l'inférence bayésienne, l'architecture du circuit résultant est le produit de plusieurs sous-blocs identiques. Pour des raisons de lisibilité, nous montrons sur la figure 4.7 la sous-partie du circuit généré qui calcule la sous-expression $\sum_{D_1}\tilde{P}(D_1)P(D_1|M=m)$ de l'expression (4.4). Cette figure est une représentation schématique du code VHDL généré par la chaîne de compilation. En bas nous distinguons la description RTL (Register Transfer Level) générée par l'outil de synthèse QUARTUS, où il est possible d'identifier les connexions entre les composants, qui correspond au circuit logique schématisé dans la figure au dessus.

Puisque la structure de l'expression (4.4) est indépendante d'une valeur particulière $m$ de $M$, le circuit généré pour calculer chaque sortie $m$ de $M$ a exactement la même structure. En conséquence, les poids introduits par les multiplexeurs, pour réaliser les sommes, sont annulés lors de la normalisation et $P(M)$ peut être correctement calculée à partir des sorties, sans utiliser le

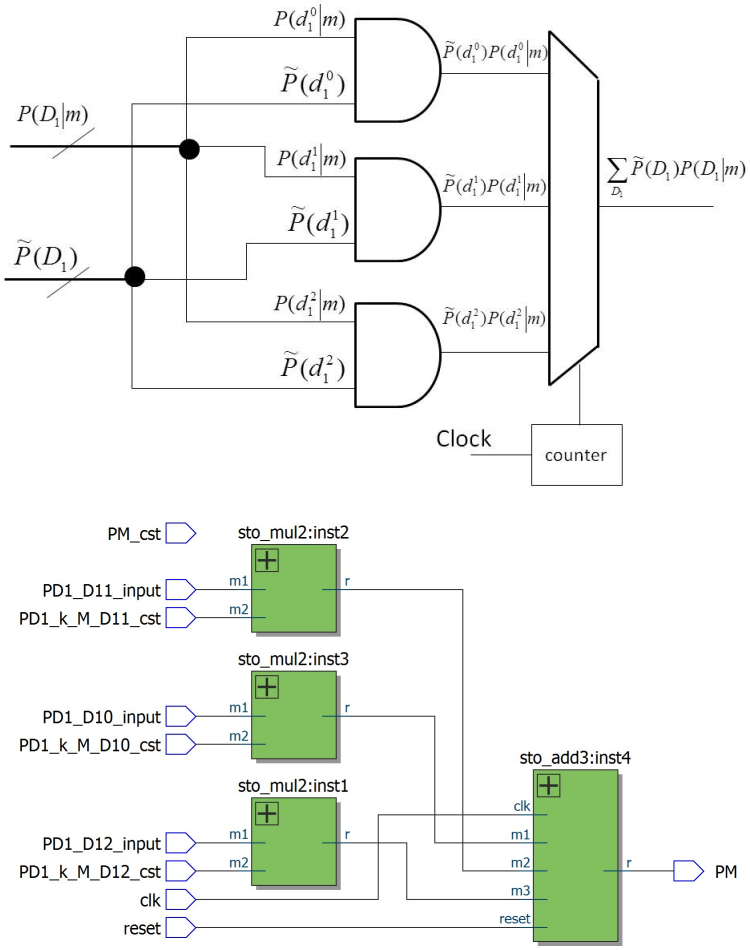



FIGURE 4.7 – Sous bloc de la machine calculant $\frac{1}{3}\sum_{D1}\tilde{P}(D_1)P(D_1|m)$ et le modèle RTL (Register Transfer Level) correspondant.

composant *OR+* pour réaliser les additions.

### Simulation haut niveau

Pour aller plus loin dans l'étude du comportement des circuits conçus, nous avons développé un simulateur haut-niveau qui permet de simuler rapidement le circuit résolvant l'inférence. Il nous permet également d'avoir un environnement de travail simplifié pour la récupération et l'étude des

résultats en sortie. Nous simulons donc les calculs effectués par le circuit électronique résultant de la compilation du programme bayésien décrit dans la figure 4.5. Comme les valeurs de probabilité sont représentées par des chaînes de bits, la longueur de ces flux, c'est-à-dire le nombre de bits utilisés pour coder une valeur de probabilité, a évidemment un impact direct sur la précision des valeurs codées et par conséquent sur la précision des calculs réalisés par un circuit donné. L'intérêt de ces simulations est donc double :

— valider le fait que les circuits générés par la chaîne de compilation réalisent correctement l'inférence demandée
— quantifier l'impact de l'utilisation du calcul stochastique sur la précision des calculs.

Pour simuler les calculs réalisés par les circuits générés, nous avons besoin de spécifier deux ensembles de valeurs de probabilité. Tout d'abord, les paramètres internes du modèle, c'est-à-dire les paramètres des termes de la distribution conjointe qui encodent la connaissance du modèle. Ces paramètres, calculés avec ProBT, sont codés en dur dans le circuit et ne varieront pas. Deuxièmement, les entrées du circuit qui sont des distributions de probabilité codant les évidences molles. Ces distributions quantifient la confiance relative sur les observations. Ces entrées sont donc des variables qui peuvent changer au fil du temps. Dans l'exemple simple, les valeurs de ces paramètres sont définies dans le tableau 4.2.

| | Internal parameters | | | | |
|---|---|---|---|---|---|
| name | $P(d_1^0\|m^0)$ | $P(d_1^1\|m^0)$ | $P(d_1^2\|m^0)$ | $P(d_1^0\|m^1)$ | $P(d_1^1\|m^1)$ |
| value | 0.2 | 0.3 | 0.5 | 0.3 | 0.2 |

| | Internal parameters | | | |
|---|---|---|---|---|
| name | $P(d_1^2\|m^1)$ | $P(d_1^0\|m^2)$ | $P(d_1^1\|m^2)$ | $P(d_1^2\|m^2)$ |
| value | 0.5 | 0.4 | 0.3 | 0.3 |

| | Inputs | | | | | |
|---|---|---|---|---|---|---|
| name | $\tilde{P}(d_1^0)$ | $\tilde{P}(d_1^1)$ | $\tilde{P}(d_1^2)$ | $\tilde{P}(d_2^0)$ | $\tilde{P}(d_2^1)$ | $\tilde{P}(d_2^2)$ |
| value | 0.8 | 0.1 | 0.1 | 0.8 | 0.1 | 0.1 |

TABLE 4.2 – Paramètres internes du modèle et entrées utilisées pour les simulations

Avec cette paramétrisation, nous simulons le comportement du circuit qui calcule

$$P'(M) = P(M \mid \tilde{P}(D_1)\ \tilde{P}(D_2))$$

Nous comparons ces résultats aux valeurs théoriques obtenues avec ProBT. L'erreur quadratique moyenne standard (RMSE) est utilisée pour mesurer la différence, sur la distribution cherchée, entre les calculs réalisés par le circuit et la valeur théorique calculée avec ProBT. La figure 4.8 montre comment cette erreur, calculée sur dix simulations avec différentes graines aléatoires, évolue lorsque la longueur des chaînes de bits augmente.

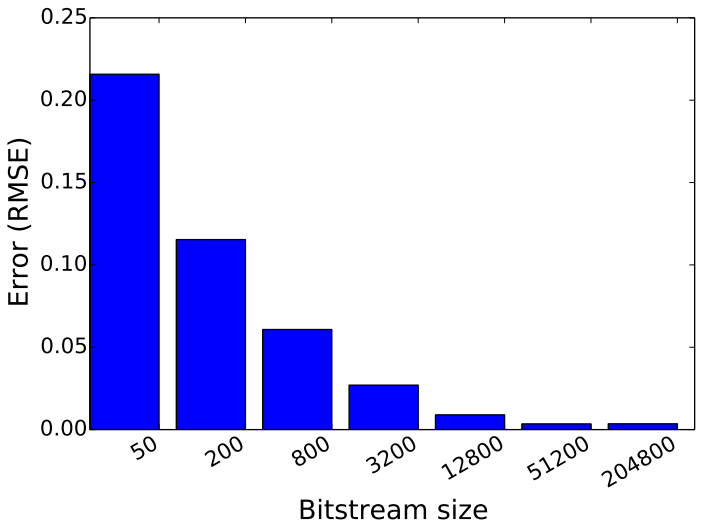



FIGURE 4.8 – Évolution du RMSE en fonction de la taille de la chaîne de bits

On observe ce qui était attendu, c'est-à-dire que l'erreur diminue lorsque la taille de la chaîne de bits augmente. De plus, même avec une taille de chaîne réduite ($< 50$ bits), le circuit est capable de fournir la valeur de la variable de la distribution cherchée ayant la probabilité maximum. Cependant, cette augmentation progressive de la précision du résultat calculé est assez lente. Il est nécessaire d'avoir des flux de plusieurs milliers de bits pour avoir des erreurs de moins de 1%. Mathématiquement, comme nous l'avons vu précédemment, la variance de l'encodage d'une probabilité $p$ par une séquence de Bernoulli de taille $N$ vaut $\sigma^2 = \frac{p(1-p)}{N}$. La précision d'un tel encodage est en $O(\sqrt{N})$. Cette propriété est en adéquation empirique avec

les données de la figure 4.8. Le tableau 4.3 montre plus en détail les valeurs de la simulation qui ont permis de tracer le graphique 4.8.

| bitstream size | 50 | 200 | 800 | 3200 | 12800 | 51200 | 204800 | **reference** |
|---|---|---|---|---|---|---|---|---|
| P(Best) | 0.801 | 0.723 | 0.724 | 0.715 | 0.714 | 0.710 | 0.711 | 0.711 |
| RMSE | 0.21 | 0.11 | 0.06 | 0.02 | 0.008 | 0.0034 | 0.0035 | 0 |

TABLE 4.3 – Comparaison sur le *best* et la RMSE de la distribution entre la valeur théorique et les simulations pour différentes tailles de chaînes de bits

Il est intéressant de noter trois choses. Tout d'abord, notre mesure d'erreur est telle que les différences dans les directions opposées ne se compensent pas. En conséquence, la valeur moyenne de la meilleure estimation de la probabilité semble converger plus rapidement que le RMSE. Deuxièmement, pour les flux de taille 204800, l'erreur est plus grande que pour les flux de taille 51200. Cela peut être due à la façon de calculer la moyenne sur 10 itérations seulement pour les simulations, ce qui introduit une sorte de bruit autour des valeurs théoriques. Pour les grandes tailles de chaînes de bit, l'erreur est faible et seul le bruit est observé. Et troisièmement, pour les petites tailles de chaîne, l'erreur est assez grande mais le biais correspondant est en faveur de la réponse la plus probable. En effet, avec un flux de petite taille, les probabilités faibles sont sous-estimées (par exemple, pour coder une valeur de probabilité de $0,1\%$, il est nécessaire d'avoir une chaîne de taille 1000 à observer pour espérer un bit non nul en moyenne). Favorisant les fortes valeurs de probabilités lors du calcul de la distribution cherchée, un système opérant avec des chaînes de tailles faibles prendra des décisions sur le *max* de façon plus rapide, et les probabilités des événements rares seront sous-estimées.

## 4.5 Applications

Dans l'exemple simple précédent, la définition d'un bus probabiliste autorisait de calculer la distribution de sortie sans faire le calcul de la constante de normalisation Z (voir l'équation (4.3)). De plus, l'utilisation d'un multiplexeur pour réaliser les sommes ne nous causait pas de soucis puisque, si nous voulions les valeurs normalisées *a posteriori* de la distribution en sortie, les facteurs de moyennement disparaissaient. Cependant, lorsque l'on cascade cet opérateur de sommation, on perd de plus en plus d'information

puisque les chaînes de bits accroissent inéluctablement leurs nombres de 0 en proportion de leurs nombres de 1. C'est ce qui a été mentionné sous le nom de dilution temporelle. On doit alors élargir la fenêtre temporelle sur laquelle on regarde la chaîne de bits résultante afin d'obtenir l'information suffisante pour extraire la valeur de la probabilité associée à cette chaîne. Nous avons montré comment nous évitons ce problème grâce au composant *OR+* que nous utilisons dans ces applications. Une illustration est donnée avec l'exemple de la conception d'un filtre bayésien pour l'acquisition de séquences pseudo-bruitées.

### 4.5.1 Filtre Bayésien

Tout d'abord, nous rappelons comment les PN (*pseudo-noise*) séquences sont produites avec des Linear Feedback Shift Registers (LFSRs) et comment deux LFSRs peuvent être synchronisés en utilisant un filtre bayésien. Nous proposons l'implémentation du circuit d'un tel filtre, y compris une partie permettant de calculer le terme de normalisation, $Z$. Nous simulons ce circuit pour quantifier la vitesse à laquelle il peut résoudre la tâche d'acquisition de séquence pseudo-bruitée pour différents taux d'erreurs de transmission.

#### Filtre LFSR - PN acquisition

La norme 3G couramment utilisée dans nos téléphones mobiles utilise les PN séquences pour propager le signal transmis. Les LFSRs (linéaires de registres à décalage) sont utilisés pour produire ces séquences pseudo-bruitées. Avant de commencer une communication, il faut synchroniser les LFSRs entre émetteur et transmetteur alors que la transmission de la séquence est corrompue par des erreurs de transmissions. Cette phase est appelée acquisition d'une séquence pseudo-bruitée [41]. Pour la partie réception, l'objectif est d'inférer l'état du LFSR transmetteur à partir d'une séquence de bits corrompue.
La figure 4.9 montre deux LFSRs de Fibonacci identiques utilisant une porte XOR et quatre bascules. Les deux LFSRs implémentent le même automate déterministe. A partir d'un état initial, c'est à dire des valeurs initiales dans les registres $Z_i$ (différent de 0000, qui est un état absorbant), les LFSRs vont parcourir en boucle une séquence finie d'états $S_T = z_4 z_3 z_2 z_1$ qui correspondent aux valeurs dans les registres et produisent un bit pseudo aléatoire $z_1$. Cette séquence d'états est appelée *Pseudo-Noise sequence* (séquence pseudo-bruitée). Le nombre maximal d'états qu'un LFSR peut parcourir avant de boucler est $P = 2^n - 1$, avec $n$ le nombre de bascules. Pour le cas

$n = 4$, il y a donc 15 états possibles distincts. Pour l'émetteur, le signal à émettre est « mélangé » par une porte XOR via $z_1 \oplus z_4$. Pour le récepteur, le signal reçu subit la même opération $z_1 oplus z_4$. Si les deux LFSRs sont synchronisés, autrement dit s'ils ont, à chaque instant, le même état $S_T$, alors, en l'absence de bruit sur la ligne, le récepteur retrouve le message émis alors qu'il apparaît sur la ligne comme un signal pseudo aléatoire. Notons ici que le but n'est pas de coder le signal mais d'équilibrer le nombre de un et de zéro transmis pour des raisons électroniques de performance énergétique.

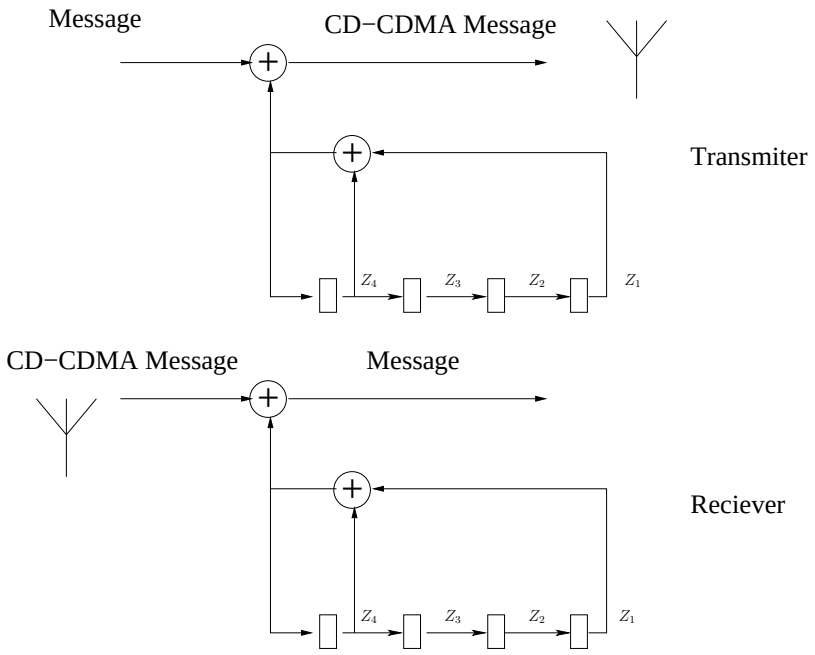



FIGURE 4.9 – Émission et réception de messages codés par LFSRs. Le même LFSR est utilisé pour coder le message transmis et décoder le message reçu.

Une condition nécessaire à l'utilisation de ce système de transmission est de passer par une phase de synchronisation où le LFSR récepteur va tout d'abord retrouver l'état $S_T$ du LFSR transmetteur. Pour cela, le transmetteur va d'abord émettre un message composé uniquement de zéro, il envoie donc les valeurs successives de son bit « $z_1 \oplus z_4$ ». La difficulté provient du fait que les bits reçus sont corrompus par le canal de transmission. Le récepteur va utiliser une chaîne de Markov pour inférer l'état du LFSR de l'émetteur malgré le bruit. On suppose bien entendu que la structure de ce

LFSR est connu du récepteur. Ainsi, si les deux LFSRs sont synchronisés, le message envoyé peut alors être reçu en fin de chaîne de transmission. Le schéma de la figure 4.9 montre le message correctement reçu à travers l'implémentation présentée. En effet, la double utilisation de la porte XOR en série, avec le même deuxième bit comme entrée, correspond à la porte *identité*. Ceci peut être noté :

$$(a \oplus b) \oplus b = a$$

Cette formule s'obtient en utilisant les règles et propriétés de l'algèbre de Boole comme montré dans l'équation 4.6.

$$\begin{aligned}
(a \oplus b) \oplus b &= (\overline{a}.b + a.\overline{b}) \oplus b \\
&= (\overline{\overline{a}.b + a.\overline{a}}).b + (\overline{a}.b + a.\overline{a}).\overline{b} \\
&= (\overline{\overline{a}b}.\overline{a\overline{b}}).b + \overline{a}b\overline{b} + a\overline{b}\overline{b} \\
&= (a + \overline{b}).(\overline{a} + b) + a\overline{b} \\
&= a\overline{a}b + ab + \overline{a}\overline{b}b + a\overline{b}b + \overline{b}b + a\overline{b} \\
&= a\overline{a}b + ab + \overline{a}\overline{b}b + a\overline{b}b + b + a\overline{b} \\
&= ab + a\overline{b} \\
&= a
\end{aligned} \tag{4.6}$$

**Fonctionnement du filtre**

Le LFSR émetteur peut être modélisé comme une chaîne de Markov récursive. Notons $S_t$ et $O_t$ l'état du LFSR et l'observation du bit reçu au temps $t$. Le LFSR est défini par la distribution selon la distribution sur l'état initial $P(S_0)$, qui est choisie uniforme, et par une expression récursive de la distribution conjointe comme présenté sur l'équation 4.7. Cette expression est l'expression générale des filtres bayésiens.

$$P(S_{t-1} \wedge S_t \wedge O_t) = P(S_{t-1})P(S_t|S_{t-1})P(O_t|S_t) \tag{4.7}$$

L'automate caractérisant le LFSR étant déterministe, en appelant $f$ la fonction qui à tout état $s$ fait correspondre un unique successeur $s'$, on pose :

$$P(S_t = s|S_{t-1} = s') = \delta_{s=f(s')}$$

La distribution $P(O_t|S_t)$ est la fonction capteur, elle code le bruit sur la ligne.

Si à l'instant $T$, la valeur du bit reçu est $o_t$ et si on l'on connaît $P(S_{t-1})$, alors on peut calculer la distribution de probabilité sur l'état courant, $S_T$,

du LFSR transmetteur à instant $T$, en réalisant l'inférence de l'équation 4.8 suivante :

$$P(S_T|O_T = o) = \frac{1}{Z}P(O_T|S_T)\sum_{S_{T-1}} P(S_{T-1})P(S_T|S_{T-1}) \tag{4.8}$$

Dans ce cas particulier l'équation 4.8 devient :

$$P(S_T = f(s')|O_T = o) = \frac{1}{Z}P(O_T|S_T = f(s'))P(S_{T-1} = s') \tag{4.9}$$

Une machine de type *SMTBI* peut alors facilement calculer le terme

$$P(O_T|S_T = f(s'))P(S_{T-1} = s')$$

Une opération de normalisation doit cependant être effectuée sur le bus stochastique pour pouvoir utiliser le résultat de l'inférence au temps $T + 1$ en posant

$$P(S_{T+1}) = P(S_T|O_T = O_t)$$

### Architecture

La figure 4.10 montre une vue simplifiée d'un circuit résolvant l'inférence sur un problème de filtre bayésien dans le cadre du problème de synchronisation de LFSRs. L'architecture proposée est composée de deux modules : la machine bayésienne qui calcule la probabilité pour l'émetteur d'être dans un état donné et le module de modélisation d'erreurs qui produit le signal d'erreur $P(O_T|S_t)$ selon les valeurs reçues du LFSR transmetteur. Pour chaque bit reçu de la PN séquence, la machine bayésienne calcule l'inférence sous forme d'un bus stochastique qui encode la distribution de probabilité sur les états du transmetteur. Chaque valeur de probabilité d'un état $P(S_T)$, c'est-à-dire une sortie du bus probabiliste, est normalisée et stockée dans une mémoire FIFO (First In First Out). Dans un premier temps, ces FIFOs sont remplies avec une séquence aléatoire de manière à encoder une distribution de probabilité uniforme correspondant à $P(S_0)$. Par la suite, chaque fois qu'elles reçoivent un nouveau bit, elles rejettent le bit qui a été stocké le plus longtemps et stocke le nouvel arrivé. Cette architecture compacte permet de mettre en œuvre un filtre avec un nombre très limité de composants CMOS.

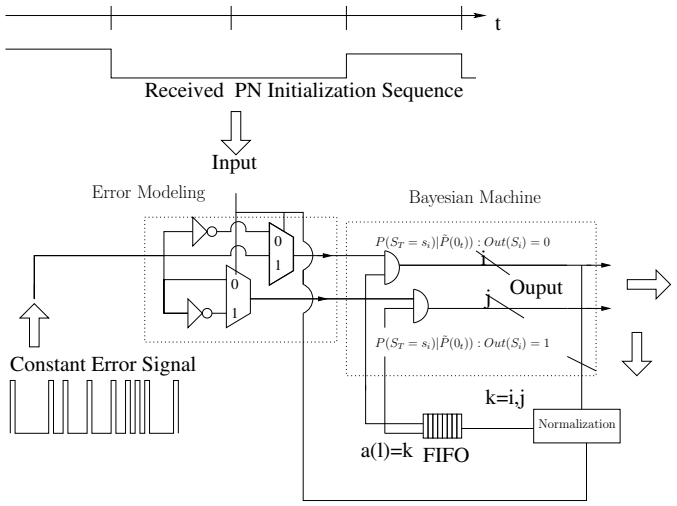



FIGURE 4.10 – Architecture du circuit implémentant un filtre bayésien en utilisant une arithmétique stochastique pour la synchronisation de LFRSs

### Résultats

Le circuit d'inférence résolvant le problème de la PN sequence acquisition (figure 4.10) a été testé avec les outils de simulation haut niveau intégrés au compilateur. Cette simple application a permis de faire émerger une limite fondamentale dans l'utilisation de la bascule JK utilisée pour réaliser la division et la normalisation. En effet, le calcul de chaque bit en sortie étant local, et impliquant au plus un état précédent, ceci entraîne des erreurs causées par les corrélations possibles entre les entrées, ou par auto-corrélation temporelle de chaque entrée. Pour éviter de telles erreurs, nous proposons d'étendre la mémoire de la bascule JK. Le composant ainsi construit s'utilise de la même manière qu'une bascule JK classique, mais ne retient pas seulement le bit précédent mais un nombre plus important dans un buffer. Lorsque la bascule doit renvoyer une valeur en fonction de l'état précédent (entrées 00 ou 11 sur la figure 4.3), elle tire au hasard dans son buffer, ce qui est une façon de ré-échantillonner une partie de la distribution calculée jusqu'à présent.

Le circuit de filtrage LFSR pour la synchronisation a été simulé avec une séquence de 100 observations en entrée et une longueur de chaîne de bits de 10000 (valeur basée sur la figure 4.8). Le circuit calcule 1500 divisions (pour normaliser chacun des 15 états possibles pour chaque entrée). Ces données sont comparées aux valeurs théoriques dans le tableau 4.4 qui montre une

mesure globale des erreurs, l'erreur quadratique moyenne, ainsi que l'erreur maximale que l'on observe.

| Taille de la mémoire | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| Division RMSE | 0.062 | 0.058 | 0.052 | 0.044 | 0.034 | 0.024 | 0.016 | 0.012 | 0.014 | 0.024 |
| Pire cas d'erreur | 0.298 | 0.287 | 0.273 | 0.240 | 0.201 | 0.148 | 0.097 | 0.066 | 0.044 | 0.089 |

TABLE 4.4 – RMSE et pire cas d'erreur pour le composant diviseur, JK *flip-flop* en fonction de la taille de la mémoire du buffer

Le cas où la taille de la mémoire du buffer est égale à 1 correspond à la bascule JK standard, qui stocke seulement un état antérieur. Les erreurs sont globalement acceptables mais dans le pire des cas elles atteignent des taux de 30%, qui entraînent une désynchronisation locale. L'augmentation de la taille de la mémoire a pour effet de diminuer à la fois la RMSE et le pire cas d'erreur, jusqu'à un point où diminue à nouveau la précision des calculs. Cela se produit parce que la mémoire utilisée est initialisée avec des valeurs aléatoires uniformément choisies, 0 ou 1, ce qui introduit un peu de bruit au début du calcul, c'est-à-dire avant que le contenu de la mémoire soit significatif. Même si le tableau 4.4 nous montre des limites évidentes sur la précision des opérations divisions en utilisant la bascule JK, l'approximation peut être assez bonne pour certaines applications spécifiques telles que les codes correcteurs d'erreurs dans les *factor graph* [45] (à noter que dans ce papier, les auteurs semblent ignorer que la bascule JK peut être peu fiable pour réaliser les divisions).

Enfin, nous avons simulé le comportement du circuit LFSR sur des séquences de 200 observations, avec différents taux d'erreur de transmission. Chaque observation a été biaisée avec une probabilité variant entre 0 et 30%. Nous avons utilisé des mémoires FIFOs de taille 10000 pour stocker les représentations des chaînes de bits de $P(S_{t-1})$. Selon les données présentées sur la figure 4.8, cela fournit une précision raisonnable. La taille des FIFOs n'est pas rédhibitoire puisque lors d'une implémentation matérielle ces FIFOs seront remplacées par des compteurs combinés avec un générateur aléatoire afin de produire une chaîne de bits codant pour la valeur de probabilité, ce qui évite d'avoir des FIFOs de longueur trop importante. A chaque étape, notre circuit reçoit un bit envoyé par l'émetteur qui est biaisé avec une probabilité variant entre 0 et 30%, et calcule une distribution de probabilité sur les états possibles de l'émetteur (voir la figure 4.10). Nous définissons l'état reconnu par notre circuit comme l'état ayant la plus forte probabilité selon cette distribution. Le tableau 4.5 montre la comparaison

entre la séquence d'états reconnus et la séquence d'origine (*i.e.* sans les erreurs de transmission) pour différents taux d'erreur.

| Taux d'erreur de transmission | 0% | 10% | 20% | 30% |
|---|---|---|---|---|
| États reconnus correctement | 198 | 188 | 184 | 177 |
| Itérations avant synchronisation | 2 | 57 | 59 | 64 |

TABLE 4.5 – Performance du circuit en fonction du taux d'erreur.

Ces résultats montrent que, lorsqu'il n'y a pas d'erreur de transmission, la synchronisation est très rapide. Sans surprise, il faut plus d'itérations pour reconnaître correctement l'état de l'émetteur lorsque l'erreur de transmission augmente. Pour les erreurs de transmission supérieures à 30%, le circuit reconnaît correctement certains états, mais ne semble pas converger vers la boucle correcte de ceux-ci. Ce n'est pas un problème inhérent au filtre LFSR (des simulations numériques exactes ont été réalisées et montrent une convergence correcte même avec des taux d'erreurs de transmission élevées), mais cela est plutôt expliqué par le fait que la division calculée par la bascule JK n'est pas stable. Elle introduit parfois des erreurs sur le calcul de la normalisation et donc sur la distribution de probabilité, ce qui entraîne la désynchronisation de l'ensemble du système lorsque la probabilité d'erreur de transmission est trop élevée. Cependant, cela peut être compensé par l'introduction d'un moyen de prendre en compte une fenêtre temporelle plus longue au lieu de se limiter à l'ordre 1 du modèle de Markov actuel. Ainsi, chaque état sera calculé suivant l'observation et la confiance dans l'état précédent, comme ce qui a été présenté, mais aussi en fonction des états encore antérieurs.

### 4.5.2 Fusion de capteurs pour robot autonome

#### Problème et Objectifs

Le but est de permettre à un robot autonome d'ajuster sa trajectoire pour éviter les obstacles en donnant une estimation de la distance fournie par deux types de capteur. Les capteurs infrarouges et les capteurs à ultrasons sont fusionnés pour augmenter la précision sur l'estimation de la distance. Le robot est présenté sur la figure 4.11. Le résultat de cette inférence est la distribution de probabilité sur la vitesse de rotation permettant au robot d'éviter les obstacles lors d'un déplacement vers l'avant.

Un schéma du robot est donné figure 4.12. Nous avons accès à trois capteurs ultrasons et trois capteurs infrarouges pour distinguer trois directions.

FIGURE 4.11 – Robot autonome évitant les obstacles grâce à la machine bayésienne.

Le modèle a donc 10 variables :

— 3 variables pour les trois capteurs infrarouges : $IR_0, IR_1, IR_2$
— 3 variables pour les trois capteurs ultrasoniques : $US_0, US_1, US_2$
— 3 variables pour les trois *priors* sur les distances : $D_0, D_1, D_2$
— 1 variable pour la vitesse de rotation cherchée : $V_{ROT}$

### Modèle capteurs

Chaque capteur a son propre modèle en fonction de la distance. Trois niveaux de distance sont définis : proche, moyen et lointain. Le robot tourne en augmentant la vitesse de la roue correspondant au côté opposé de la direction désirée. Nous définissons cinq vitesses de rotation : forte vitesse sur la gauche, faible vitesse sur la gauche, vitesse nulle, faible vitesse sur la droite, forte vitesse sur la droite. Une distribution uniforme est définie sur les *priors* des variables de distance. Les deux capteurs infrarouges et ultrasoniques ont 80% de confiance sur la lecture connaissant la distance. Nous modélisons aussi la vitesse de rotation sachant la distance. Une partie de code ProBT qui décrit ce système est donnée figure 4.13.

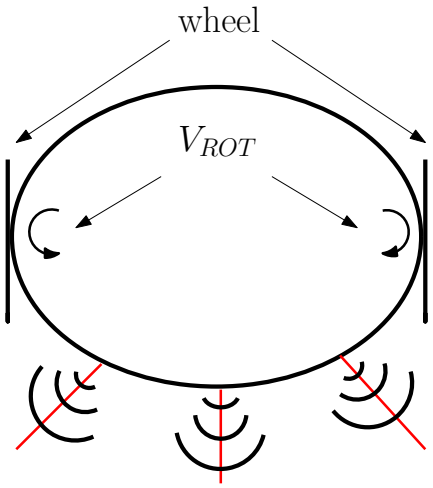



FIGURE 4.12 – Schéma du robot autonome utilisé dans l'expérience de l'évitement d'obsacle

*priors*

$P_{D_0} = plUniform(D_0)$
$P_{D_1} = plUniform(D_1)$
$P_{D_2} = plUniform(D_2)$

*modèle de capteur IR*

$P_{IR} = [0.8, 0.1, 0.1, 0.1, 0.8, 0.1, 0.1, 0.1, 0.8]$
$P_{IR_0}\_k\_D_0 = plDistributionTable(IR_0, D_0, P_{IR})$
$P_{IR_1}\_k\_D_1 = plDistributionTable(IR_1, D_1, P_{IR})$
$P_{IR_2}\_k\_D_2 = plDistributionTable(IR_2, D_2, P_{IR})$

*modèle de capteur US*

$P_{US} = [0.8, 0.1, 0.1, 0.1, 0.8, 0.1, 0.1, 0.1, 0.8]$
$P_{US_0}\_k\_D_0 = plDistributionTable(US_0, D_0, P_{US})$
$P_{US_1}\_k\_D_1 = plDistributionTable(US_1, D_1, P_{US})$
$P_{US_1}\_k\_D_2 = plDistributionTable(US_2, D_2, P_{US})$

*modèle de la vitesse de rotation*

$P_{VROT}\_k\_D_0D_1D_2 = plDistributionTable(V_{ROT}, D_0D_1D_2, P_{VROT})$

FIGURE 4.13 – Spécification du modèle capteur en code ProBT

**Fusion**

La fusion de capteurs peut être modélisée par la distribution conjointe suivante :

$$\begin{aligned} P(D_0 \wedge D_1 \wedge D_2 \wedge IR_0 \wedge IR_1 \wedge IR_2 \wedge US_0 \wedge US_1 \wedge US_2 \wedge V_{ROT}) \\ = \; P(V_{ROT}|D_0D_1D_2) \times P(D_0) \times P(D_1) \times P(D_2) \\ \times P(IR_0|D_0) \times P(IR_1|D_1) \times P(IR_2|D_2) \\ \times P(US_0|D_0) \times P(US_1|D_1) \times P(US_2|D_2) \end{aligned} \tag{4.10}$$

C'est la distribution conjointe portant sur les variables définies dans le modèle. La décomposition est le produit des trois *priors* sur les distances, des trois distributions conditionnelles sur la variable IR (infrarouge), des trois distributions conditionnelles sur la variable US (ultrason) et de la distribution conditionnelle, $P(V_{ROT}|D_0D_1D_2)$, sur la vitesse de rotation connaissant les distances qui spécifie une forme de la loi de contrôle. Inférer sur ce modèle revient à poser la question $P(V_{ROT}|IR_0IR_1IR_2US_0US_1US_2)$ et donc calculer l'expression 4.11 suivante :

$$\begin{aligned} P(V_{ROT} \mid IR_0IR_1IR_2US_0US_1US_2) \\ = \sum_{D_2} \Big[ P(D_2)P(US_2 \mid D_2)P(IR_2 \mid D_2) \\ \sum_{D_1} \Big( P(D_1)P(US_1 \mid D_1)P(IR_1 \mid D_1) \\ \sum_{D_0} \left[ P(D_0)P(US_0 \mid D_0)P(IR_0 \mid D_0)P(V_{ROT} \mid D_0D_1D_2) \right] \Big) \Big] \end{aligned} \tag{4.11}$$

**Architecture**

Le circuit généré comporte environ 2500 composants et plusieurs milliers de signaux. Il est dupliqué cinq fois, puisqu'il y a cinq valeurs possibles sur la variable d'intérêt cherchée qui sont $P(v_{ROT_0})...P(v_{ROT_4})$. Ainsi, le circuit est entièrement parallèle en termes de la cardinalité de la variable cherchée $P(V_{ROT})$. Une chose importante à observer est le nombre d'étapes dans le circuit. La dilution temporelle des valeurs de probabilités, en raison des nombreux produits nécessaires, diminue considérablement ces valeurs. Ainsi, la chaîne de bits représentant la valeur de probabilité est remplie avec beaucoup de 0. La longueur de la chaîne de bits doit alors être très

longue pour évaluer cette valeur en raison du choix du codage temporel des probabilités.

Pour utiliser la plate-forme du robot, une simulation avec du code $C++$ est générée à partir du code VHDL. Elle correspond exactement au code VHDL en termes de composants et signaux.

### Simulation et résultats

Pour simuler notre circuit, nous avons utilisé le code $C++$ qui prend comme entrées les capteurs de lecture, calcule les distributions de probabilité et envoie l'ordre de la vitesse de rotation. La vitesse linéaire donnée est de $0,2ms^{-1}$ en raison du temps de calcul de notre simulateur. En effet, pour une taille de chaîne de bits de 5000, le simulateur a besoin de $86ms$ pour envoyer une commande. Ce temps de calcul est linéaire avec la longueur de la chaîne de bits utilisée pour le calcul dans le simulateur. Le cahier des charges pour ce robot stipule un temps de $100ms$ entre deux ordres avec cette vitesse linéaire. Donc, si nous voulons augmenter la vitesse linéaire, nous devons faire des simulations plus rapide ou diminuer la taille du flux binaire.

| Taille de la chaîne | outil | $v_{ROT_0}$ | $v_{ROT_1}$ | $v_{ROT_2}$ | $v_{ROT_3}$ | $v_{ROT_4}$ |
|---|---|---|---|---|---|---|
| calcul exact | ProBT | 0.005 | 0.012 | 0.068 | 0.293 | 0.622 |
| 5000 | simulateur | 0 | 0 | 0.052 | 0.21 | 0.74 |
| | erreur | n.a | n.a | 24% | 28% | 19% |
| 10000 | simulateur | 0 | 0 | 0.040 | 0.320 | 0.640 |
| | erreur | n.a | n.a | 41% | 9% | 2.9% |
| 100000 | simulateur | 0.004 | 0.007 | 0.064 | 0.331 | 0.594 |
| | erreur | 25% | 4% | 6% | 13% | 4.5% |
| 1000000 | simulateur | 0.003 | 0.013 | 0.068 | 0.289 | 0.628 |
| | erreur | 41% | 8% | 0% | 1.5% | 1% |

TABLE 4.6 – Comparaison entre le résultat théorique et la simulation pour différentes tailles de chaînes de bits pour le calcul de la vitesse de rotation

Le tableau 4.6 montre, sans surprise, que plus la longueur de la chaîne est grande, plus l'erreur, entre le calcul stochastique réalisé par la machine et le calcul exact, est réduite. Les valeurs de $v_{ROT_0}$ et $v_{ROT_1}$ ne sont pas pris en compte dans l'analyse qualitative au vu de leur faible valeur de probabilité par rapport aux autres variables.

## 4.6 Discussion

Nous avons décrit une chaîne de compilation qui prend en entrée un programme bayésien et produit un circuit qui réalise le calcul d'inférence. La spécificité d'un tel calculateur réside dans son architecture non conventionnelle car se soustrayant aux principes de Von Neumann et à l'arithmétique flottante.

L'implémentation de telles machines, utilisant une arithmétique stochastique pour traiter des problèmes d'inférence exacte, entre parfaitement dans le cadre de la recherche de solutions pour la diminution des contraintes électroniques telles que la conception de circuit tolérant aux fautes, l'efficacité énergétique ou encore la robustesse. En effet, le codage intrinsèque des probabilités répond directement à ces contraintes. Une application non triviale de cette architecture, modélisant les filtres bayésiens, a montré l'intérêt de la conception de ces machines. Elle résout le problème de synchronisation de LFSRs avec un nombre de composants très réduit.

Les outils développés compilent tout programme bayésien, modélisé avec des variables discrètes, et produisent un code VHDL qui décrit un circuit uniquement basé sur une arithmétique stochastique. Une des forces de l'utilisation de signaux binaires, par rapport aux signaux analogiques par exemple, est qu'ils sont capables de transmettre des informations de façon fiable sur le circuit, sans perte en ligne notamment. Nous avons présenté deux nouveaux composants pour l'arithmétique stochastique. Tout d'abord, le *OR+* qui permet d'éviter le biais sur le résultat des valeurs de probabilité lors de la sommation. Deuxièmement, nous avons conçu un moyen pratique et compact pour normaliser toute distribution de probabilité grâce à une bascule JK étendue avec une mémoire tampon. Cette extension étant nécessaire pour résoudre le problème de la possible forte auto-corrélation temporelle des entrées. C'est une façon de stocker une partie de la distribution calculée jusqu'alors.

En l'état, les outils mis en place compilent tout programme Bayésien modélisé avec ProBT. Cependant, bien que la nature intrinsèquement parallèle des circuits que nous générons permet de gagner en temps d'exécution par rapport à une implémentation séquentielle réalisant l'inférence exacte, le coût de ce gain en performance n'est pas toujours négligeable. En effet, la complexité temporelle de l'exécution séquentielle d'un programme sur du logiciel résulte directement, pour la machine, d'une complexité spatiale (en termes de nombre de composants nécessaires) sur le matériel. De plus, l'utilisation de chaînes de bits comme représentation d'une donnée probabiliste

est un frein à l'accélération du calcul de l'inférence. La comparaison, en vitesse d'exécution, pour chaque application devra être étudiée pour s'assurer du gain effectif par rapport à un développement logiciel.

Enfin, comme nous l'avions mentionné en introduction, les *SMTBIs* sont restreintes à des problèmes de faibles dimensions de par la méthode de résolution d'inférence exacte implémentée. Ce qui est aussi le cas avec les architectures standards du fait de la complexité NP-*difficile* des problèmes d'inférences. C'est pourquoi, nous proposons une nouvelle architecture de machine aux principes totalement différents de celles-ci qui permet de s'intéresser à des problèmes de grandes dimensions.

# Chapitre 5

# Machine bayésienne stochastique implémentant une méthode d'inférence approchée

Dans ce chapitre, nous présentons une nouvelle architecture de machine bayésienne qui traite l'inférence approchée, appelée *SMABI* pour *Stochastic Machines for Approximate Bayesian Inference*.

L'intérêt d'une telle machine est de résoudre n'importe quel problème d'inférence, même de grandes dimensions. Elle se distingue donc de la *SMTBI* par sa capacité à passer à l'échelle. La méthode de résolution des *SMABIs* est radicalement différente puisque celles-ci implémentent une méthode approchée afin d'obtenir une approximation de l'inférence cherchée. Nous décrivons également la chaîne de synthèse logique qui, comme celle de la machine traitant l'inférence exacte, décrite dans le chapitre 4, prend en entrée un programme bayésien sur des variables discrètes et le compile en un code VHDL qui spécifie le circuit réalisant l'inférence approchée.

L'exemple du *sprinkler* est réutilisé pour montrer le fonctionnement de cette nouvelle architecture. La *SMABI* a été testée sur l'application du robot autonome déjà utilisée pour la *SMTBI*. Mais c'est sur un problème *difficile* en inférence exacte que la machine démontre toute son efficacité. Ce problème est décrit dans la section 5.6.2. Une étude analytique permet d'approcher la solution avec la précision souhaitée, ce qui nous permet de montrer la qualité d'approximation de la *SMABI*. Ces applications ont été exécutées sur le simulateur haut niveau mis en place dans le cadre de ces travaux.

## 5.1 Problématique

La machine *SMTBI*, présentée dans le chapitre 4, traite les problèmes d'inférence exacte avec de l'arithmétique stochastique. Elle nous a permis de montrer, non seulement la plausibilité de concevoir des architectures de calcul stochastique, mais aussi son efficacité sur des applications où le nombre de variables et leur cardinalité restent restreints. Cependant, ce type de machine, traitant l'inférence exacte, ne permet pas de passer à l'échelle, ce qui est aussi le cas sur les architectures standards. En effet, les problèmes d'inférence font partie de la catégorie des problèmes NP-*difficiles* où l'explosion combinatoire entraîne une explosion exponentielle (non-polynomiale) en temps de calcul. Pour faire face à cette contrainte, les méthodes MCMCs ont été développées avec succès. L'échantillonnage de Gibbs fait partie de ces méthodes d'inférences approchées. Cet algorithme a été choisi comme base du calcul pour l'architecture de cette nouvelle machine qui montre son efficacité sur les problèmes d'inférence approchée à grandes dimensions dans le cadre d'une implémentation matérielle.

## 5.2 Principe

### 5.2.1 Principe général

L'algorithme d'échantillonnage de Gibbs est une méthode MCMC particulière pour réaliser l'inférence approchée. L'idée des machines *SMABIs* est d'implémenter de manière efficace cette méthode d'échantillonnage. Seuls trois blocs sont nécessaires à l'implémentation d'un tel algorithme dans la machine stochastique : les *Conditionnal Distributions Elements* (*CDE*), les générateurs aléatoires de chaînes de bits (*BSG* pour *BitStream Generator*) et les *OPs*, pour *Odd Products*, qui permettent de réaliser les produits de cotes nécessaires à l'échantillonnage de Gibbs.

Les *CDEs* permettent de stocker toutes les valeurs de probabilité des distributions connues. Les *BSGs*, décrits dans le chapitre 3, sont utilisés pour générer des flux binaires stochastiques nécessaires à la représentation des valeurs de probabilité. Nous avons développé deux types de composant *OPs*. Le premier est une extension du composant logique *porte de Muller* ou *C-element*, extension dans le sens où la mémoire n'est plus une simple bascule mais un buffer qui stocke une partie de la distribution calculée jusqu'à lors. Ce composant se rapproche de la bascule $JK$ étendue présentée dans la section 4.3.2. Le second utilise la porte XNOR, avec mémoire, et des signaux de validation du bit échantillonné. Dans la suite, nous rappelons l'algorithme

de Gibbs dans le cas des variables binaires et nous montrons comment il est implémenté grâce aux trois composants cités préalablement.

**Retour sur les chaînes de bit**

Un flux de bits permet d'encoder une valeur de probabilité d'une variable binaire : $p = n_1/(n_1 + n_0)$, où $n_1$ est le nombre de 1 et $n_0$ le nombre de 0 dans le flux. Une façon isomorphe de voir cet encodage est d'utiliser les cotes (aussi appelées cotes, quotients, rapports ou ratios) qui sont le rapport du nombre de 1 sur le nombre de 0 dans une chaîne binaire. Ce qui donne :

$$o = \frac{n_1}{n_0} \qquad p = \frac{n_1}{n_1 + n_0}$$
$$o = \frac{p}{1-p} \qquad p = \frac{o}{1+o}$$

Étant donné une probabilité $p$, ou une cote $o$, nous utilisons des processus stochastiques logiciels ou matériels pour générer un flux de bits $B(p)$ (respectivement $B(o)$) qui encodent approximativement la probabilité $p$ (respectivement la cote $o$). Les opérateurs sur les flux binaires peuvent être ensuite utilisés pour effectuer les calculs nécessaires à la résolution de l'inférence sur les distributions binaires. Un des intérêts d'une telle architecture est que ces opérateurs sont simples et peu nombreux, il s'agit des composants *BSGs* et *OPs*.

### 5.2.2 Algorithme de Gibbs : cas binaire

Posons un ensemble $X$ de $\lfloor X \rfloor$ variables binaires, $X = \{B_1, \cdots, B_{\lfloor X \rfloor}\}$. Soit un sous-ensemble $K = \{K_1, \cdots, K_{\lfloor K \rfloor}\}$ de $X$ qui représente les variables connues $\{k_1, \cdots, k_{\lfloor K \rfloor}\}$. Soit un autre sous-ensemble $I = \{I_1, \cdots, I_{\lfloor I \rfloor}\}$ de $X$ qui représente les variables d'intérêt ou cherchées. On peut aussi poser $F$ le sous-ensemble complémentaire qui représente les variables libres. On a alors $X = K \cup I \cup F$ et forcément $\lfloor X \rfloor = \lfloor K \rfloor + \lfloor I \rfloor + \lfloor F \rfloor$. Le but de l'algorithme de Gibbs est de tirer des échantillons suivant la distribution $P(I_1 \cdots I_{\lfloor I \rfloor} F_1 \cdots F_{\lfloor F \rfloor} \; k_1, \cdots, k_{\lfloor K \rfloor})$. Nous utilisons l'algorithme de Gibbs dans une variante binaire décrite sur la figure 5.1 :

Cet algorithme nous montre que l'on a seulement besoin de savoir tirer suivant les distributions $P(B_i \mid b_1 \dots b_{i-1}, b_{i+1} \dots, b_{\lfloor X \rfloor})$, $\forall i \in [0; \lfloor I \rfloor + \lfloor F \rfloor]$. Dans la suite nous montrons une méthode efficace pour échantillonner ces distributions.

**for** $B_i \in K$ **do**
  Mettre dans $B_i$ la valeur correspondante $k_j$
**end for**
**for** $B_i \in I \cup F$ **do**
  Tirer une valeur $b_i$ aléatoire.
**end for**{Initialisation}
**while** TRUE **do**
  **for** $B_i \in I \cup F$ **do**
    Tirer $b_i$ selon $P(B_i \mid b_1 \dots b_{i-1}, b_{i+1}, \dots, b_{\lfloor X \rfloor})$
    Stocker la valeur binaire des variables dans $I$
  **end for**
**end while**

FIGURE 5.1 – Algorithme générique de l'échantillonnage de Gibbs sur des variables binaires

**Comment tirer suivant** $P(B_i \mid b_1 \dots b_{i-1}\ b_{i+1} \dots b_{\lfloor X \rfloor})$

Toute distribution de probabilité conjointe sur $\lfloor X \rfloor$ variables binaires peut être décrite de la manière suivante en appliquant la propriété sur les probabilités conditionnelles, il s'agit de la décomposition la plus naïve :

$$P\left(B_1 B_2 \cdots B_{\lfloor X \rfloor}\right) = \prod_{n=1}^{\lfloor X \rfloor} \left[P\left(B_n \mid B_1 \cdots B_{n-1}\right)\right]$$

Par le théorème de Bayes, nous obtenons l'expression 5.16 suivante :

$$P(B_i \mid b_1 \cdots b_{i-1}\ b_{i+1} \cdots b_{\lfloor X \rfloor}) = \frac{P(B_i\ b_1 \cdots b_{i-1}\ b_{i+1} \cdots b_{\lfloor X \rfloor})}{P(b_1 \cdots b_{i-1}\ b_{i+1} \cdots b_{\lfloor X \rfloor})} \tag{5.1}$$

En supprimant la constante de normalisation et en utilisant la décomposition précédente, on obtient l'équation 5.2 :

$$\begin{aligned} P(B_i \mid b_1 \cdots b_{i-1}\ b_{i+1} \cdots b_{\lfloor X \rfloor}) \propto\ & P\left(B_i | b_1 \cdots b_{i-1}\right) \\ & \times \prod_{n=1}^{i-1} \left[P\left(b_n | b_1 \cdots b_{n-1}\right)\right] \\ & \times \prod_{n=i+1}^{\lfloor X \rfloor} \left[P\left(b_n | b_1 \cdots B_i \cdots b_{n-1}\right)\right] \end{aligned} \tag{5.2}$$

La cote $B_i$ est quant à elle calculée comme suit :

$$\begin{aligned} & O(B_i \mid b_1 \cdots b_{i-1}\ b_{i+1} \cdots b_{\lfloor X \rfloor}) \\ & \quad = \frac{P(B_i = 1 \mid b_1 \cdots b_{i-1}\ b_{i+1} \cdots b_{\lfloor X \rfloor})}{P(B_i = 0 \mid b_1 \cdots b_{i-1} b_{i+1} \cdots b_{\lfloor X \rfloor})} \end{aligned} \tag{5.3}$$

En utilisant l'équation 5.2, on en déduit l'expression 5.4 de la cote $B_i$ :

$$\begin{aligned} & O(B_i \mid b_1 \cdots b_{i-1}\ b_{i+1} \cdots b_{\lfloor X \rfloor}) \\ & \quad = \frac{P\left(B_i = 1 | b_1 \cdots b_{i-1}\right)}{P\left(B_i = 0 | b_1 \cdots b_{i-1}\right)} \\ & \qquad \times \prod_{n=1}^{i-1} \left[ \frac{P\left(b_n | b_1 \cdots b_{n-1}\right)}{P\left(b_n | b_1 \cdots b_{n-1}\right)} \right] \\ & \qquad \times \prod_{n=i+1}^{\lfloor X \rfloor} \left[ \frac{P\left(b_n | b_1 \cdots B_i = 1 \cdots b_{n-1}\right)}{P\left(b_n | b_1 \cdots B_i = 0 \cdots b_{n-1}\right)} \right] \end{aligned} \tag{5.4}$$

On constate ici que l'on a bien une égalité car le terme normatif disparaît lors du rapport. Il est remarquable de noter que se soustraire de la normalisation est l'un des points forts de cette méthode.

Comme $B_i$ n'est pas présent dans les termes $\prod_{n=1}^{i-1} \left[ \frac{P(b_n | b_1, \cdots, b_{n-1})}{P(b_n | b_1, \cdots, b_{n-1})} \right]$, alors chaque quotient de ce produit vaut 1 et ces termes disparaissent donc de l'expression, ce qui donne :

$$\begin{aligned} & O(B_i \mid b_1 \cdots, b_{i-1} b_{i+1} \cdots b_{\lfloor X \rfloor}) \\ & \quad = \frac{P\left(B_i = 1 | b_1 \cdots b_{i-1}\right)}{P\left(B_i = 0 | b_1 \cdots b_{i-1}\right)} \\ & \qquad \times \prod_{n=i+1}^{\lfloor X \rfloor} \left[ \frac{P\left(b_n | b_1 \cdots B_i = 1 \cdots b_{n-1}\right)}{P\left(b_n | b_1 \cdots B_i = 0 \cdots b_{n-1}\right)} \right] \end{aligned} \tag{5.5}$$

Le résultat de ce produit de cotes donne la proportion qu'il existe entre les valeurs $\{0\ ou\ 1\}$ de la variable binaire $B_i$. En tirant aléatoirement suivant cette proportion, $p = \frac{o}{1+o}$, on obtient un échantillon de $B_i$.

Ici, nous avons utilisé la décomposition la plus naïve pour un modèle bayésien dont les variables sont toutes binaires. En général, on se sait pas tirer suivant les distributions $P\left(b_n | b_1, \cdots, b_{n-1}\right)$. Dans la pratique, nous

utilisons la décomposition de la conjointe et les formes paramétriques associées pour calculer la valeur des termes qui apparaissent dans le produit de cotes. Dans ce cas, les termes du produit où n'apparaît pas la variable échantillonnée valent 1 et sont donc simplifiés comme précédemment.

Chaque terme du produit de cotes a donc a une probabilité $p$, stockée dans le *CDE* correspondant. Ce composant est présenté dans la section 5.3.1. Cette valeur de probabilité est isomorphe à une cote $o$, avec $p = o/(1+o)$, qui peut être échantillonnée par des flux binaires générés par les *BSGs* décrits dans le chapitre 3 section 3.4.3. Leurs produits sont calculés par les *OPs* présentés dans la section 5.3.2.

### Passage aux variables discrètes

Les exemples décrits dans ces travaux utilisent des variables binaires et discrètes. Cependant l'algorithme de Gibbs présenté précédemment fonctionne de la même manière avec ce type de variable. Il s'agit simplement de changer la représentation de la variable, qui sera donc binarisée pour être exploitée.

**Exemple**

Soit A, B et C trois variables discrètes représentées respectivement par $A_2A_1A_0$, $B_1B_0$ et $C_3C_2C_1C_0$. Supposons que l'on a la décomposition suivante :

$$P(ABC) = P(A) \times P(B|A) \times P(C|B)$$

Alors on peut la réécrire suivant l'équation 5.6.

$$\begin{aligned} &P(A_2A_1A_0\ B_1B_0\ C_3C_2C_1C_0) \\ &\quad = P(A_2A_1A_0) \times P(B_1B_0 \mid A_2A_1A_0) \times P(C_3C_2C_1C_0 \mid B_1B_0) \end{aligned} \tag{5.6}$$

Chacune des variables discrètes du modèle sera échantillonnée bit à bit. Supposons que l'on échantillonne le bit $B_1$ de la variable $B$. Le quotient à calculer est donc le suivant :

$$\begin{aligned} O(B_1 \mid a_2\ a_1\ a_0\ b_0\ c_3\ c_2\ c_1\ c_0) &= \frac{\dfrac{P(a_2\ a_1\ a_0\ B_1 = 1\ b_0\ c_3\ c_2\ c_1\ c_0)}{P(a_2\ a_1\ a_0\ b_0\ c_3\ c_2\ c_1\ c_0)}}{\dfrac{P(a_2\ a_1\ a_0\ B_1 = 0\ b_0\ c_3\ c_2\ c_1\ c_0)}{P(a_2\ a_1\ a_0\ b_0\ c_3\ c_2\ c_1\ c_0)}} \\ &= \frac{P(a_2\ a_1\ a_0\ B_1 = 1\ b_0\ c_3\ c_2\ c_1\ c_0)}{P(a_2\ a_1\ a_0\ B_1 = 0\ b_0\ c_3\ c_2\ c_1\ c_0)} \end{aligned} \tag{5.7}$$

En utilisant la décomposition de la conjointe, on obtient l'équation 5.8 suivante :

$$\begin{aligned} & O(B_1 \mid a_2\ a_1\ a_0\ b_0\ c_3\ c_2\ c_1\ c_0) \\ &= \frac{P(a_2\ a_1\ a_0)\ P(B_1 = 1\ b_0 \mid a_2\ a_1\ a_0)\ P(c_3\ c_2\ c_1\ c_0 \mid B_1 = 1\ b_0)}{P(a_2\ a_1\ a_0)\ P(B_1 = 0\ b_0 \mid a_2\ a_1\ a_0\ )\ P(c_3\ c_2\ c_1\ c_0 \mid B_1 = 0\ b_0)} \\ &= \frac{P(a_2\ a_1\ a_0)}{P(a_2\ a_1\ a_0)} \times \frac{P(B_1 = 1\ b_0 \mid a_2\ a_1\ a_0)}{P(B_1 = 0\ b_0 \mid a_2\ a_1\ a_0)} \times \frac{P(c_3\ c_2\ c_1\ c_0 \mid B_1 = 1\ b_0)}{P(c_3\ c_2\ c_1\ c_0 \mid B_1 = 0\ b_0)} \end{aligned} \tag{5.8}$$

Comme $B_i$ n'apparaît pas dans le terme $P(A_2A_1A_0)$, son ratio vaut donc 1. Ce qui se simplifie, comme le montre l'équation 5.9, de la manière suivante :

$$\begin{aligned} & O(B_1 \mid a_2\ a_1\ a_0\ b_0\ c_3\ c_2\ c_1\ c_0) \\ &= \frac{P(B_1 = 1\ b_0 \mid a_2\ a_1\ a_0)}{P(B_1 = 0\ b_0 \mid a_2\ a_1\ a_0)} \times \frac{P(c_3\ c_2\ c_1\ c_0 \mid B_1 = 1\ b_0)}{P(c_3\ c_2\ c_1\ c_0 \mid B_1 = 0\ b_0)} \end{aligned} \tag{5.9}$$

Comme l'on connaît les distributions :

$$P(B_1\ B_0 \mid A_2\ A_1\ A_0) \text{ et } P(C_3\ C_2\ C_1\ C_0 \mid B_1\ B_0)$$

nous sommes capables de calculer chaque quotient et d'en faire le produit. Ainsi nous avons accès à la probabilité

$$p = \frac{O(B_1 \mid a_2\ a_1\ a_0\ b_0\ c_3\ c_2\ c_1\ c_0)}{1 + O(B_1 \mid a_2\ a_1\ a_0\ b_0\ c_3\ c_2\ c_1\ c_0)}$$

selon laquelle il faut tirer pour échantillonner $B_1$.

Nous verrons dans les prochaines sections que l'utilisation du calcul stochastique couplé à des composants spécialement conçus pour réaliser le produit de cotes nécessaire génère directement l'échantillon $B_1$ avec la bonne proportion.

## 5.3 Architecture de calcul

Dans cette section, nous présentons les éléments *CDEs* qui stockent l'ensemble des informations des distributions et *OPs* qui permettent l'échantillonnage d'un bit de la conjointe en réalisant le produit de cotes. En plus des *BSGs*, ces deux composants sont les seuls nécessaires à l'implémentation de l'algorithme de Gibbs pour résoudre l'inférence approchée sur la machine.

### 5.3.1 CDE : *Conditional Distribution Element*

Les *CDEs* sont des composants mémoires qui stockent toutes les valeurs des quotients possibles pour chaque terme de la décomposition de la conjointe. La valeur de ces quotients est pré-calculée lors de la synthèse logique présentée dans la section 5.5. La mémoire des machines *SMABI* est donc décomposée en sous-blocs. Chaque bloc est un *CDE* associé à un terme de la conjointe. C'est une manière de stocker la distribution de ce terme.

Le nombre de valeurs à stocker dans un *CDE* dépend du nombre de variables binaires au sein du terme associé. La formule 5.10 donne la taille du *CDE* en fonction du nombre de variables binaires $n$ :

$$taille\_CDE = n \times 2^{n-1} \tag{5.10}$$

### 5.3.2 OP : *Odd Product*

De la même façon que Gaines [19, 4], nous pouvons utiliser des opérateurs logiques qui combinent des signaux stochastiques et réalisent des opérations arithmétiques. Par exemple, en prenant deux flux binaires stochastiques indépendants $B(p_1)$ et $B(p_2)$, nous avons montré, dans le chapitre 4, qu'une simple porte AND nous permettait d'obtenir $B(p_1.p_2)$. Comme nous l'avons expliqué dans la section 5.2.2, ce dont nous avons besoin ici est le produit de deux cotes, c'est-à-dire un moyen de générer $B(o_1.o_2)$ à partir de deux flux binaires indépendants $B(o_1)$ et $B(o_2)$. Pour réaliser cela, nous proposons deux solutions : (i) la première consiste à utiliser la porte de Muller (ou C-éléments) en étendant la ressource mémoire (ii) la seconde est l'utilisation de la porte XNOR basée sur un principe proche de celui de la porte C. Dans une implémentation, afin de minimiser la profondeur de ces opérateurs, et donc le chemin critique, ces *OPs* seront agencés sous forme d'arbre.

**ECE : *Extended C-Element***

La table de vérité de la porte de Muller est donnée dans le tableau 5.1. Ce composant utilise une mémoire d'un bit pour fournir la valeur précédente $Z_{prev}$ en sortie, lorsque les entrées sont distinctes, c'est à dire soit $\{0, 1\}$ soit $\{1, 0\}$. Quand deux chaînes de bits indépendantes et non-corrélées, $B(o_1)$ et $B(o_2)$, sont données en entrée d'un C-élément, la sortie $B(o_1.o_2)$ est un estimateur sans biais du produit de cotes de $o_1 \times o_2$. Ce résultat peut être obtenu en appliquant le principe d'équilibre (*detailed balance principle*) à la chaîne de Markov $Z_n \to Z_{n+1}$.

| $X$ | $Y$ | $Z$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | $Z_{prev}$ |
| 1 | 0 | $Z_{prev}$ |
| 1 | 1 | 1 |

TABLE 5.1 – Table de vérité de la porte de Muller avec deux entrées $X$ and $Y$ et une sortie $Z$.

Cependant, il y a une certaine auto-corrélation dans le flux de sortie car un bit au temps $t$ n'est pas indépendant du bit au temps $t-1$ en raison de l'effet de mémoire. Cette auto-corrélation pose des problèmes lorsque les C-éléments sont cascadés [17]. Pour réduire l'auto-corrélation en sortie, nous étendons la capacité de la mémoire en remplaçant la bascule sur un bit par un buffer $C$ de taille $D$ bits. Ce buffer toroïdal fonctionne comme suit : lorsque les entrées ont la même valeur ($X$=$Y$=$v$), le contenu du buffer est décalé vers la gauche, son bit de droite vaut dorénavant $v$ et nous fournissons $v$ en sortie. Lorsque les entrées ont des valeurs différentes, nous fournissons le bit de droite de $C$ en sortie et nous faisons pivoter le contenu du buffer vers la droite.

Pour chaque produit de cotes à réaliser nous avons donc une période de remplissage des buffers afin d'obtenir une représentation de la distribution de chaque terme du produit. Si $D$=1, nous avons une porte de Muller classique. La table de vérité de l'*ECE* est présentée dans le tableau 5.2.

La figure 5.2 présente le RMSE expérimental pour deux cascades d'*ECEs* lorsque la taille du buffer est augmentée. On remarque que l'utilisation d'une porte de Muller classique (non-étendue), c'est à dire $D = 1$, entraîne les pires performances. Le RMSE décroit drastiquement lorsque l'on fait varier la taille du buffer de 1 à 4, puis stagne pour des tailles supérieures à 5.

Ce type d'*OPs*, ayant une mémoire étendue stockant une partie de la

| $X$ | $Y$ | $Z$ | Action |
|---|---|---|---|
| 0 | 0 | 0 | décalage à gauche de C<br>mettre le dernier bit de C à 0 |
| 0 | 1 | Bit le plus à droite de C | Rotation à droite de C |
| 1 | 0 | Bit le plus à droite de C | Rotation à droite de C |
| 1 | 1 | 1 | décalage à gauche de C<br>mettre le dernier bit de C à 1 |

TABLE 5.2 – Table de vérité de l'*ECE* avec $X$ et $Y$ en entrée et $Z$ en sortie.

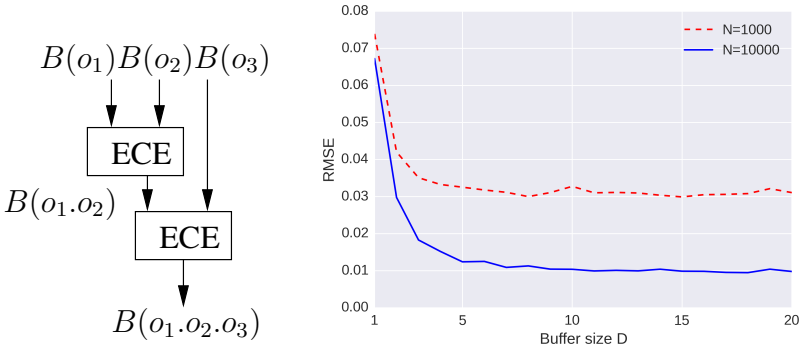



FIGURE 5.2 – Cascades de deux *ECEs*. Deux *ECEs* sont cascadés pour calculer le produit de cotes : $B(o_1.o_2.o_3)$. Les cotes $o_1$, $o_2$ et $o_3$ correspondent à des valeurs de probabilités tirées aléatoirement entre 0 et 1 et le RMSE est calculé expérimentalement pour des tailles de buffer variant de 1 à 20, moyenné sur plus de 10 000 exécutions. La précision dépend aussi de la taille $N$ de la chaîne de bits considérée (dans ce cas 1000 ou 10 000) et varie en $\frac{1}{\sqrt[2]{N}}$.

distribution, ont besoin d'une période d'initialisation pour que le contenu de la mémoire soit fiable. C'est à dire qu'il représente la distribution qui est calculée. Ceci est réalisé en utilisant un compteur $l$ qui valide la sortie après une séquence « d'amorçage » (*Burn in*) de longueur $k$.

### XNOR étendue

Le produit de cote réalisé avec la porte XNOR est basé sur la validation des échantillons binaires de chaque terme du produit. La figure 5.3 montre le fonctionnement global du composant. Les signaux $S_1$ et $S_2$ sont les entrées

du composant, ils encodent donc les valeurs des termes dont on veut faire le produit. Ils correspondent aux bits générés avec la proportion souhaitée pour le premier étage des *OPs*. Les signaux $V_1$ et $V_2$ sont les signaux de validation de $S_1$ et $S_2$. Ils sont mis à "1" pour le premier étage des *OPs*. Les signaux $I_1$ et $I_2$ sont les signaux dits «informatifs». Ils nous permettent d'ignorer (*bypass*) le circuit lorsque l'on a un nombre impair de produits de cotes par exemple et ainsi laisser passer le signal contenant de l'information sans réaliser aucune opération. Le tableau 5.3 donne la table de vérité de ce composant. S'il n'y a pas de produits de cotes à réaliser, le composant retourne alors un signal *is_informative* nul et le signal $S_{out}$ ne sera pas considéré dans la suite de l'arbre d'*OPs*. Si un des deux signaux n'est pas informatif, le composant ne laisse alors passer que le signal d'intérêt. Si les deux signaux sont informatifs, le composant réalise le produit de cotes et envoie le résultat correspondant en sortie.

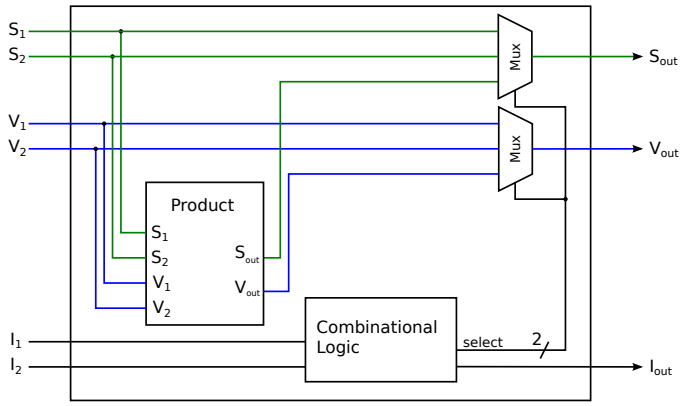



FIGURE 5.3 – Vue d'ensemble du composant de produit de cotes stochastiques. Le circuit de *bypass* correspond aux signaux de validation $V$, en bleu, tandis que les flux binaires correspondent aux signaux $S$ indiqués en vert. Si un seul des bits est informatif il est alors directement connecté à la sortie.

La porte XNOR permet de détecter l'égalité de toutes les entrées et donc de réaliser le produit de cotes entre deux signaux. En effet, l'*odd* en sortie, $o(S_{out})$, vaut :

$$o(S_{out}) = \frac{P(S_{out} = 1)}{P(S_{out} = 0)}$$

Or, lorsque la sortie du XNOR est à "1", alors l'*odd* vaut :

$$o(S_{out}) = \frac{P(S_1 = 1, S_2 = 1)}{P(S_1 = 0, S_2 = 0)}$$

| *is_informative_1* | *is_informative_2* | output | *is_informative_output* |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | $bit_1$ | 1 |
| 1 | 0 | $bit_2$ | 1 |
| 1 | 1 | $1 \times 2$ | 1 |

TABLE 5.3 – table de vérité du composant indiquant si le signal de sortie est informatif.

Si $S_1$ et $S_2$ sont indépendants, on peut écrire :

$$o = \frac{P(S_1 = 1)P(S_2 = 1)}{P(S_1 = 0)P(S_2 = 0)} = o(S_1)o(S_2)$$

Le produit est donc réalisé comme sur la figure 5.4.

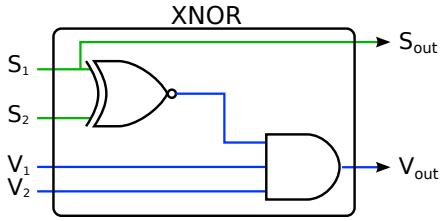



FIGURE 5.4 – Vue d'ensemble de la porte XNOR qui permet le calcul du produit de cotes de deux flux binaires stochastiques. Un arbre binaire de ces blocs peuvent être assemblés afin de calculer le produit de cotes d'un nombre quelconque de chaînes de bits.

Le problème de la dilution est toujours présent avec cette méthode. Il se peut alors que l'on attende un temps très long pour avoir un signal valide en sortie de l'arbre d'*OPs*. C'est notamment le cas lorsque un terme proche de 0 est multiplié par un terme proche de 1 (l'*odd* de ce terme tend alors vers l'infini). Pour pallier ce problème, nous proposons d'étendre la porte XNOR en lui ajoutant une mémoire qui permet de stocker les bits valides au fur et à mesure du traitement. Cette mémoire est un buffer accessible en lecture (*read mode*) et écriture (*write mode*). La figure 5.5 montre les deux modes d'utilisation du composant. Lorsque le bit produit est valide, il est placé en sortie et stocker dans le buffer avec un signal de validation qui vaut "1". Lorsque le bit produit est non-valide, deux cas sont possibles : (i) le buffer possède des bits valides, le dernier est alors mis en sortie du composant et le pointeur de validité du buffer est décalé. Le signal de validité en sortie

du composant est mis à ”1”. (ii) Le pointeur de validité du buffer est en position nulle, aucun bit valide n'a été stocké, la sortie du composant est donc la valeur du bit non-valide et le signal de validité est mis vaut donc ”0”.

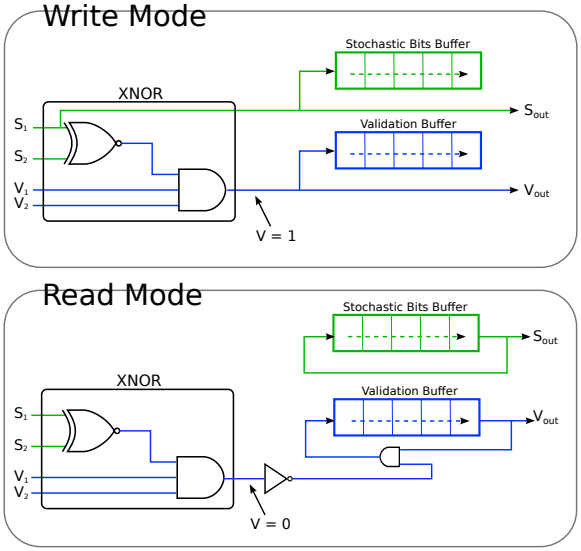



FIGURE 5.5 – Vue d'ensemble de la version étendue de la porte XNOR avec de la mémoire. Les deux modes de fonctionnement sont réglés en fonction de la validité de la sortie de la porte XNOR.

## 5.4 Architecture physique

L'architecture de la *SMABI* correspond à la mise en relation des trois éléments qui composent la machine, c'est à dire les *CDEs*, les *BSGs* et les *OPs*. Les *CDEs* stockent tous les quotients nécessaires aux produits de cotes. Les *BSGs* permettent de générer les flux binaires dont la valeur de probabilité a été adressée dans les *CDEs*. Le calcul qui réalise l'échantillonnage du bit visé est effectué avec l'arbre d'*OPs*. Afin de stocker et mettre à jour la valeur des bits échantillonnés, nous avons implémenté un registre de Gibbs qui est un simple banc de registres de taille égale au nombre de bits présents dans la conjointe décrite dans notre modèle avec ProBT.

### 5.4.1 L'exemple du *Sprinkler*

Reprenons l'exemple du *sprinkler* pour présenter l'architecture de la *SMABI*. On rappelle que cet exemple contient trois variables binaires $R$, $S$, $G$ correspondant aux prédicats il a plu, l'arroseur automatique a été activée et l'herbe est humide. La distribution conjointe peut être décomposée comme suit : $P(R,S,G) = P(R)P(S|R)P(G|R,S)$. La question que l'on se pose est : quelle est la probabilité qu'il ait plu sachant que l'herbe est humide ou sèche : $P(R|G=g)$.

Selon l'algorithme de Gibbs, nous avons besoin d'échantillonner les variables $R$ et $S$, ce qui signifie, en utilisant l'équation 5.5, réaliser le calcul de deux cotes selon les formules de l'équation 5.11 :

$$\begin{aligned} O(R\ G=g,S=s) &= \tfrac{P(R=1)}{P(R=0)} \cdot \tfrac{P(s|R=1)}{P(s|R=0)} \cdot \tfrac{P(g|R=1,s)}{P(g|R=0,s)}\ , \\ O(S\ G=g,R=r) &= \tfrac{P(S=1|r)}{P(S=0|r)} \cdot \tfrac{P(g|r,S=1)}{P(g|r,S=0)}\ . \end{aligned} \tag{5.11}$$

Ces formules montrent qu'il existe 13 ratios possibles qui sont les connaissances préliminaires spécifiées sur le modèle bayésien étudié. A chaque terme de la décomposition est associé un *CDE* qui va stocker tous les quotients possibles de cette distribution. On a donc un *CDE* pour $P(R)$ afin de produire la chaîne de bits $B(o_R)$ avec $o_R = \frac{P(R=1)}{P(R=0)}$, un autre *CDE* pour $P(S|R)$ avec 4 valeurs en mémoire, qui correspondent aux 4 quotients possibles, afin de produire les chaînes de bits $B(o_S^1)$, encodant $o_S^1 = \frac{P(S=1|R=0)}{P(S=0|R=0)}$, $B(o_S^2)$, encodant $o_S^2 = \frac{P(S=1|R=1)}{P(S=0|R=1)}$, $B(o_S^3)$, encodant $o_S^3 = \frac{P(S=0|R=1)}{P(S=0|R=0)}$ et $B(o_S^4)$, encodant $o_S^4 = \frac{P(S=1|R=1)}{P(S=1|R=0)}$. Pour le troisième terme de la décomposition, $P(G|R,S)$ on a besoin de stocker 8 valeurs en mémoire.

Pour réaliser les produits de cotes, deux *OPs* sont implémentés pour tirer $O(R \mid S = sG = g)$, qui dépend de trois termes et nécessitent donc deux produits, et pour tirer $O(S \mid R = rG = g)$ qui ne dépend que de deux termes et nécessite donc un seul produit de cotes. La figure 5.6 schématise le circuit qui implémente l'inférence $P(R|G=g)$. L'entrée est réglée sur la valeur désirée de $G=g$ et la sortie est une chaîne binaire $B(o)$ encodant $O(R|G=g,S=s)$ ou $O(S|G=g,R=r)$ suivant le bit que l'on échantillonne. L'une des forces de cette méthode est que le premier bit émanant de l'arbre d'*OPs* contient déjà toute l'information nécessaire sur la distribution de la variable binaire échantillonnée. Le gain en temps de calcul est alors très important puisqu'une telle architecture ne nécessite pas d'attendre une longue chaîne de bits pour avoir un résultat avec une précision suffisante, comme il est

souvent reproché, à juste titre, aux architectures stochastiques utilisant des flux binaires.

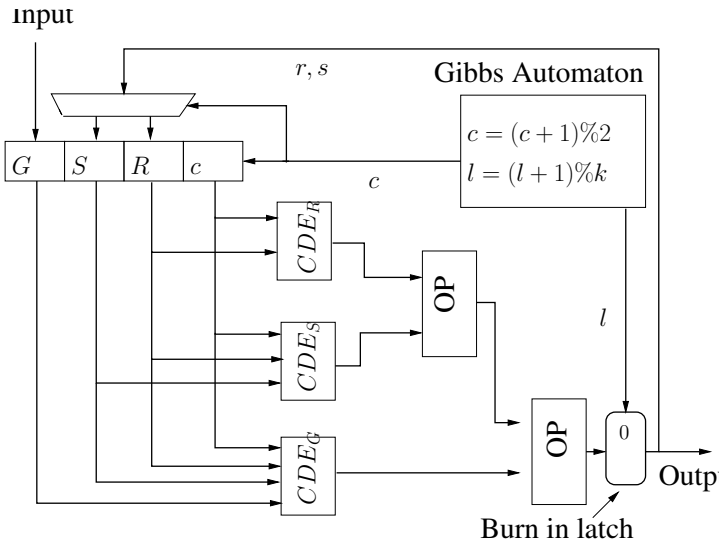



FIGURE 5.6 – Circuit réalisant l'inférence approchée sur l'exemple du *sprinkler*.

### 5.4.2 Contrôle et subtilités

L'algorithme de Gibbs nécessite de scanner en séquence toutes les variables binaires des variables cherchées et libres. Ceci est obtenu en utilisant un compteur $c$. Ce pointeur est utilisé par le multiplexeur en haut à gauche de la figure 5.6 pour sélectionner la variable à mettre à jour. Il est également utilisé comme une entrée de chaque $CDE$ pour sélectionner le rapport approprié pour le produit de cotes. En effet, ceux-ci dépendent de la variable échantillonnée, il s'agit de $R$ ou $S$ dans notre exemple. Dans certains cas, la variable échantillonnée n'est pas présente dans le terme de la décomposition, ce qui est équivalent à avoir un rapport 1, qui est l'élément neutre pour le produit de cotes. Dans notre exemple, si nous échantillonnons $S$ ($c = 0$), nous n'avons pas besoin du rapport $\frac{P(R=1)}{P(R=0)}$ et la sortie du $CDE_R$ devrait être la chaîne de bits $B(o = 1)$ avec un nombre égal de 0 et 1. Cette chaîne est donc l'élément neutre pour l'opérateur $OP$. Un simple signal de validation nous permet d'éviter d'envoyer les chaînes neutres, inutiles au calcul, sur les $OPs$.

Notons que les valeurs des distributions conditionnelles stockées par les *CDEs* ne sont pas nécessairement les valeurs des distributions conditionnelles du modèle. Par exemple, pour les $CDE_S$, la valeur que le compilateur met en mémoire à l'adresse $\{S = 1, R = 1, C = 1\}$ est la valeur $p = \frac{o_S^4}{1+o_S^4}$ avec $o_S^4 = \frac{P(S=1|R=1)}{P(S=1|R=0)}$.

## 5.5 Synthèse probabiliste de haut niveau

Une chaîne de synthèse probabiliste de haut niveau a été développée. Elle fonctionne de la même manière que celle présentée dans le chapitre 4 pour la *SMTBI*. L'entrée est un programme bayésien, utilisant des variables discrètes, écrit en ProBT (voir la figure 5.12 pour un exemple). La sortie est un programme VHDL spécifiant le circuit dédié à l'inférence correspondante.

Comme nous l'avons déjà vu, tout programme probabiliste utilisant des variables discrètes, peut être défini par la conjointe : $P(S \wedge K \wedge F)$, où $S$ représente les variables cherchées, $K$, les variables connues et $F$, les variables libres du modèle. Chacune des ces variables peut être elle-même une conjonction de variables. Les variables libres, $F$, sont échantillonnées de la même manière que les variables cherchées, $S$, nous regroupons donc ces deux types de variables et les représentons par la lettre $U$ pour *Unknown variables*. L'algorithme de Gibbs utilisé opérant au niveau binaire, la première étape de la chaîne de synthèse est donc de binariser la description du modèle. En effet, on va parcourir la vecteur des variables inconnues bit à bit. Le principe reste inchangé quant à la méthode d'échantillonnage comme expliqué à la fin de la section 5.2.2. La décomposition reste la même mais les variables ont une représentation binaire au sein des différents termes. C'est lors du post-traitement que l'on extrait les informations d'intérêt, c'est à dire les valeurs des échantillons des variables cherchées.

Le compilateur prend en entrée le programme bayésien binarisé et génère un fichier VHDL décrivant le circuit. Il spécifie la mise en œuvre du circuit de l'équation 5.5 dont l'architecture est similaire à celle présentée dans la figure 5.6.

La figure 5.7 [1] schématise l'outil de synthèse mis en place. Les options permettent de définir le type de machine souhaité pour réaliser le calcul de l'inférence. Dans le cas d'une machine *SMABI*, une étape de binarisation

1. Cette figure est tirée du délivrable ®Probayes pour le projet BAMBI réalisé par Raphaël Laurent

est nécessaire. Dans les deux cas l'outil de synthèse génère un fichier VHDL spécifiant la machine visée.

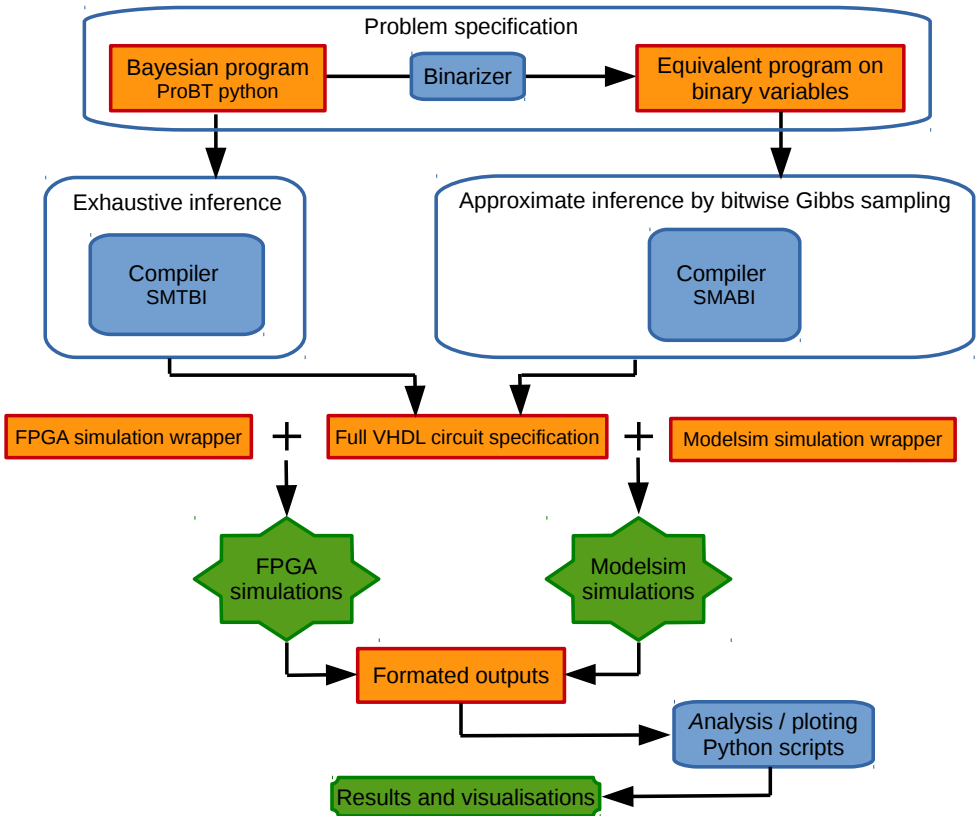



FIGURE 5.7 – Schéma des outils de synthèse probabiliste pour les *SMABIs*

### Simulation

Le code VHDL généré peut être simulé avec les outils de CAO classiques. Cependant, afin de gagner du temps de simulation au vu de la simplicité des opérations, nous avons amélioré le moteur de simulation de la *SMTBI* pour qu'il simule également des machines de type *SMABI*.

## 5.6 Applications

### 5.6.1 Retour sur le robot autonome

L'application du robot autonome présentée dans le chapitre 4 est réutilisée pour montrer l'efficacité dans la *SMABI*. La problématique reste l'évitement d'obstacle pour le robot en inférant sur trois distances $D_0, D_1$ et $D_2$ dont les données des capteurs ultrasons $US$ et infrarouges $IR$ sont incertaines. Nous redonnons la figure 5.8 qui montre les possibilités de directions du robot.

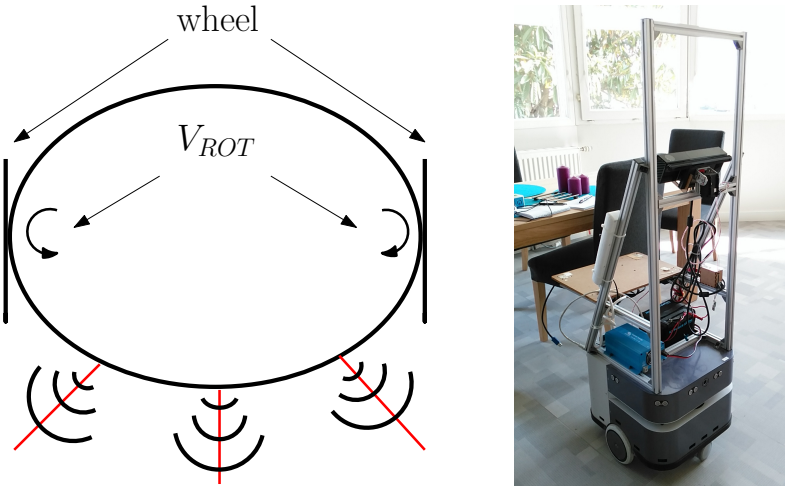



FIGURE 5.8 – Vue schématique du robot autonome

La distribution conjointe, les formes paramétriques du modèle capteur et la question posée sont les mêmes que celles présentées dans 4.5.2. On rappelle la décomposition de la conjointe dans l'équation 5.12 :

$$\begin{aligned} & P(D_0\ D_1\ D_2\ IR_0\ IR_1\ IR_2\ US_0\ US_1\ US_2\ V_{ROT}) \\ &= P(D_0) \times P(D_1) \times P(D_2) \times \\ & \quad P(IR_0|D_0) \times P(IR_1|D_1) \times P(IR_2|D_2) \times \\ & \quad P(US_0|D_0) \times P(US_1|D_1) \times P(US_2|D_2) \times \\ & \quad P(V_{ROT}|D_0 D_1 D_2) \end{aligned} \tag{5.12}$$

L'inférence à réaliser est donnée dans l'équation 5.13 :

$$\begin{aligned} & P(V_{ROT}|IR_0IR_1IR_2US_0US_1US_2) \\ =& \sum_{D_2}\Big[(P(D_2)P(US_2|D_2)P(IR_2|D_2) \\ & \sum_{D_1}\Big[P(D_1)P(US_1|D_1)P(IR_1|D_1) \\ & \sum_{D_0}\left[P(D_0)P(US_0|D_0)P(IR_0|D_0)P(V_{ROT}|D_0D_1D_2)\right]\Big]\Big] \end{aligned} \tag{5.13}$$

En utilisant l'algorithme de Gibbs, on cherche à échantillonner la variable $V_{ROT}$.

**Algorithme de Gibbs pour le robot autonome**

Le système est composé d'un ensemble de 10 variables :

$$V = \{V_{ROT}, D_0, D_1, D_2, IR_0, IR_1, IR_2, US_0, US_1, US_2\}$$

On partitionne V en trois sous-ensembles. Le premier sous-ensemble contient les données des capteurs, c'est à dire les variables connues :

$$K = \{IR_0, IR_1, IR_2, US_0, US_1, US_2\}$$

Le second sous-ensemble contient la variable recherchée :

$$S = \{V_{ROT}\}$$

Un dernier sous-ensemble contient les variables libres :

$$F = \{D_0, D_1, D_2\}$$

Puisqu'on réalise un échantillonnage de Gibbs, le sous-ensemble intéressant est celui qui contient les variables inconnues, notées $U$, c'est-a-dire qui vont être échantillonnées :

$$U = F \cup S = \{D_0, D_1, D_2, V_{ROT}\}$$

Afin d'obtenir la distribution de probabilité sur $V_{ROT}$, connaissant les valeurs des six lectures de capteurs $IR_0$, $IR_1$, $IR_2$, $US_0$, $US_1$, $US_2$, nous utilisons la méthode d'échantillonnage présentée dans la section 5.2.2.

Nous proposons de montrer une étape du calcul pour illustrer le fonctionnement de l'algorithme pour le contrôle du robot autonome. Supposons qu'à une étape du calcul, l'état du système soit le suivant :

$$\begin{array}{llll} D_0 = d_0 & US_0 = us_0 & IR_0 = ir_0 & V_{ROT} = v_{rot} \\ D_1 = d_1 & US_1 = us_1 & IR_1 = ir_1 & \\ D_2 = d_2 & IR_2 = ir_2 & IR_2 = ir_2 & \end{array}$$

Pour obtenir un nouvel échantillon de $D_0$, nous avons besoin de tirer suivant la distribution $P(D_0\ d_1, d_2, ir_0, ir_1, ir_2, us_0, us_1, us_2)$. Ces tirages s'effectuent sur les trois bits représentant $D_0 = B^2_{D_0} B^1_{D_0} B^0_{D_0}$. Supposons que l'état actuel soit $D_0 = 1$ et que nous tirons suivant le troisième bit de $D_0$, $B^2_{D_0}$, le principe étant le même pour les autres bits. La cote $B^2_{D_0}$ est définie par le rapport des probabilités de l'équation 5.14 :

$$\begin{aligned} &O(B^2_{D_0} |\ \ b^1_{D_0}, b^0_{D_0}, d_1, d_2, ir_0, ir_1, ir_2, us_0, us_1, us_2) \\ &\qquad = \frac{P(B^2_{D_0} = 1 \mid b^1_{D_0}, b^0_{D_0}, d_1, d_2, ir_0, ir_1, ir_2, us_0, us_1, us_2)}{P(B^2_{D_0} = 0 \mid b^1_{D_0}, b^0_{D_0}, d_1, d_2, ir_0, ir_1, ir_2, us_0, us_1, us_2)} \end{aligned} \tag{5.14}$$

Nous pouvons noter que cette formule, au niveau binaire, est équivalente à celle au niveau discret de l'équation 5.15 suivante :

$$\begin{aligned} &O(D_0 \mid d_1, d_2, ir_0, ir_1, ir_2, us_0, us_1, us_2) \\ &\qquad = \frac{P(D_0 = 5 \mid d_1\ d_2\ ir_0\ ir_1\ ir_2\ us_0\ us_1\ us_2)}{P(D_0 = 1 \mid d_1\ d_2\ ir_0\ ir_1\ ir_2\ us_0\ us_1\ us_2)} \end{aligned} \tag{5.15}$$

D'après le théorème de Bayes, chaque terme du quotient peut être calculé avec la formule de l'équation 5.16 suivante :

$$\begin{aligned} &P(D_0 \mid d_1\ d_2\ ir_0\ ir_1\ ir_2\ us_0\ us_1\ us_2) \\ &\qquad = \frac{P(D_0\ d_1\ d_2\ ir_0\ ir_1\ ir_2\ us_0\ us_1\ us_2)}{P(d_1\ d_2\ ir_0\ ir_1\ ir_2\ us_0\ us_1\ us_2)} \end{aligned} \tag{5.16}$$

En combinant les équations 5.14 et 5.16, la constante de normalisation disparaît dans le rapport, ce que nous avons déjà mentionné comme étant l'un des atouts principaux de cette méthode. En utilisant ensuite la décom-

position explicitée dans l'équation 5.12, nous obtenons l'équation 5.17 :

$$\begin{aligned} & O(D_0 \mid d_1\ d_2\ ir_0\ ir_1\ ir_2\ us_0\ us_1\ us_2) \\ & \quad = \frac{P([D_0=5])\ P(d_1)\ P(d_2)}{P([D_0=1])\ P(d_1)\ P(d_2)} \\ & \qquad \times \frac{P(ir_0 \mid\ [D_0=5])P(ir_1 \mid d_1)\ P(ir_2 \mid d_2)}{P(ir_0 \mid\ [D_0=1])P(ir_1 \mid d_1)\ P(ir_2 \mid d_2)} \\ & \qquad \times \frac{P(us_0 \mid\ [D_0=5])P(us_1 \mid d_1)\ P(us_2 \mid d_2)}{P(us_0 \mid\ [D_0=1])P(us_1 \mid d_1)P(us_2 \mid d_2)} \\ & \qquad \times \frac{P(v_{rot} \mid\ [D_0=5]\ d_1d_2)}{P(v_{rot} \mid\ [D_0=1]\ d_1d_2)} \end{aligned} \tag{5.17}$$

Les termes où $D_0$ n'est pas présent valent 1 et correspondent à l'élément neutre pour le produit de cotes, ils disparaissent donc de l'expression qui se simplifie comme le montre l'équation 5.18 :

$$\begin{aligned} & O(D_0 \mid d_1, d_2, ir_0, ir_1, ir_2, us_0, us_1, us_2) \\ & \quad = \frac{P([D_0=5])}{P([D_0=1])} \times \frac{P(ir_0 \mid\ [D_0=5])}{P(ir_0 \mid\ [D_0=1])} \\ & \qquad \times \frac{P(us_0 \mid\ [D_0=5])}{P(us_0 \mid\ [D_0=1])} \times \frac{P(v_{rot} \mid\ [D_0=5]\ d_1d_2)}{P(v_{rot} \mid\ [D_0=1]\ d_1d_2)} \end{aligned} \tag{5.18}$$

Tous les bits de toutes les variables inconnues sont échantillonnés avec cette méthode, ce qui revient à réaliser un produit de cotes entre différents termes de la conjointe dont toutes les valeurs possibles sont stockées dans les *CDEs* correspondants.

La figure 5.9 montre l'architecture utilisée pour l'échantillon d'un bit de $D_0$. Les produits sont réalisés par l'arbre d'*OPs* (ici les *ECEs*), qui prend en entrée un flux binaire codant pour la valeur du ratio correspondant. Le nombre de termes de la décomposition faisant intervenir $D_0$ étant égal à 4, on a donc 4 *CDEs*. L'automate de Gibbs permet de mettre à jour le registre de Gibbs à la fin du calcul et de signifier quelle est la variable binaire échantillonnée.

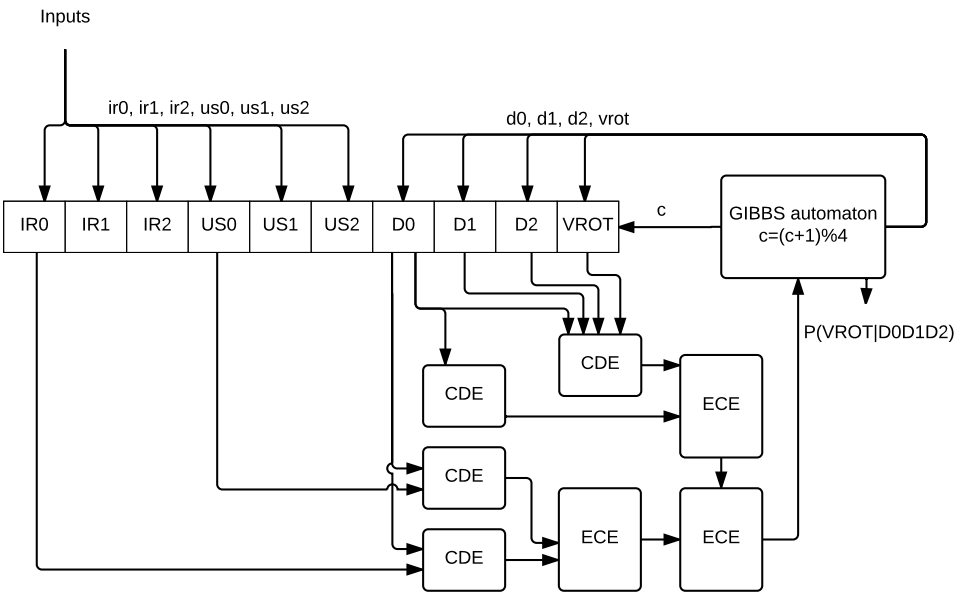



FIGURE 5.9 – Architecture d'une *SMABI* pour l'application du robot autonome : les valeurs actuelles des variables sont stockées dans un banc de registres. L'automate de Gibbs choisit le pointeur de bits à échantillonner, c, pour mettre à jour à chaque itération le registre de Gibbs. Les composants *CDEs* et *OPs* permettent d'effectuer les opérations probabilistes nécessaires. Pour plus de clarté, nous montrons uniquement les composants nécessaires à l'élaboration d'un nouveau bit pour $D_0$. La binarisation des variables étant implicite.

### Code de Gray

Le codage binaire standard entraîne ce qu'on appelle un "désert de probabilité" ou "creux de probabilité". Comme le montre la première ligne du tableau 5.4, certains nombres entiers pourtant proches ont un codage binaire très éloigné. C'est le cas de 4 = 100 et 3 = 011 par exemple. De ce fait, comme nous échantillonnons les bits de ces valeurs entières, il peut être difficile de passer de la valeur 4 à la valeur 3 car cela nécessite de changer tous les bits de leur représentation. Hors, si certains états ont une très faible chance d'être atteint, le passage de l'état 4 à 3, ou inversement, risque de ne pas être réalisé dans un temps raisonnable. C'est un problème classique dû à notre façon de cherchée la masse de probabilité dans l'espace des possibles. Pour résoudre ce problème, nous utilisons un code de Gray [13] en interprétant nos variables discrètes et leur équivalent binaire, comme le montre le tableau 5.4. Ce code garantit que les valeurs codées de tout entier et son successeur diffèrent exactement d'un seul bit.

| Entier | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Binaire | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| Code de Gray | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |

TABLE 5.4 – Code de Gray pour 3 bits

### Synthèse et simulation

Le circuit généré est un code VHDL spécifiant le circuit dont une partie est présenté sur la figure 5.9. Nous avons utilisé notre simulateur c++ pour simuler ce circuit et réaliser l'expérience en temps réel de l'évitement d'obstacle, visible sur la vidéo [2]. Dans nos expériences, nous comparons plusieurs architectures qui diffèrent sur les valeurs de discrétisation des distances et de la vitesse de rotation. Tout d'abord, l'expérience A, utilise un facteur de discrétisation similaire à celle de l'expérience en inférence exacte, à savoir 3 valeurs pour les distances et 5 pour la vitesse de rotation. La seconde expérience, noté B, dispose de 8 valeurs pour discrétiser les distances et 7 pour la vitesse de rotation. La troisième expérience, noté C, a 16 valeurs pour les distances et 15 pour la vitesse de rotation. L'objectif étant d'étudier l'effet de la discrétisation sur la vitesse de calcul et le comportement du robot. Nous nous restreignons à ces ensembles de valeurs de façon à pouvoir faire fonctionner le robot en temps réel avec notre simulateur.

2. `https://www.youtube.com/watch?v=LcXwHg45jPM`

### L'expérience

La machine stochastique pour le robot a été testée dans un environnement réel. Le robot a été placé dans un appartement avec des obstacles tels que des chaises, des tables et des canapés. Nous avons fait avancer le robot à une vitesse constante de $0.15ms^{-1}$. Le dispositif de commande fonctionne comme suit : tout d'abord on lit les données du capteur, ensuite on exécute le calcul de l'inférence pendant 100ms, C'est à dire que l'on échantillonne les variables du modèle durant le temps imparti. Puis on envoie l'ordre de la vitesse de rotation de telle sorte que le robot évite les obstacles potentiels. La position de départ des différentes expériences pour les trois types d'architecture est la même. La figure 5.10 montre l'appartement utilisé lors de nos expérimentations.

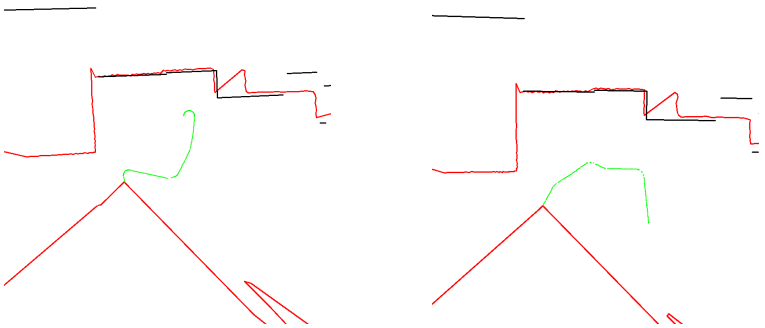
FIGURE 5.10 – Vue de l'appartement utilisé pour l'expérimentation

### Résultats

La figure 5.11 montre une carte de l'appartement et les trajectoires prises par le robot pour les différentes architectures.

On voit que le robot est capable de modifier sa trajectoire afin d'éviter les obstacles. Nous constatons que sur l'architecture A, les angles de trajectoire du robot sont très abruptes, proche 90° pour chaque changement de direction, et que, pour des valeurs de discrétisations supérieures (architecture B et C), la trajectoire est remarquablement plus lisse.

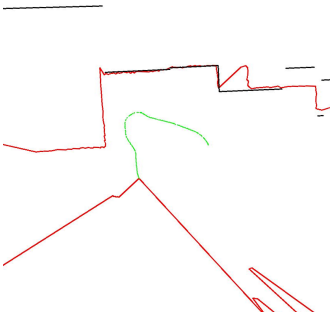
Le tableau 5.5 montre le nombre de composants nécessaires pour les trois architectures A, B et C produites par la chaîne de synthèse du chapitre 4 pour l'inférence exacte, et pour celles de l'inférence approchée. Nous voyons



(a) Architecture $A$



(b) Architecture $B$



(c) Architecture $C$

FIGURE 5.11 – Trajectoires du robot autonome calculées par le programme embarqué dans sa plate forme. La délimitation de la pièce est représentée par des lignes noires. En rouge, on peut voir la vue des obstacles détectés par le capteur du robot. Les lignes vertes correspondent aux trajectoires suivies par le robot pour les trois architectures différentes.

| Architecture | A | B | C |
|---|---|---|---|
| Inférence exacte | 1825 | 78190 | 2297310 |
| Échantillonnage de Gibbs | 92 | 119 | 149 |

TABLE 5.5 – Comparaison du nombre de composant pour les architectures d'inférence exactes et approchées.

que le nombre de composants augmente de façon exponentielle lorsque la discrétisation est plus précise pour les architectures réalisant l'inférence exacte. Nous notons que le nombre de composants utilisés par les *SMABIs* augmente de manière logarithmique. Cette augmentation est due aux composants de contrôle plus nombreux.

### 5.6.2 Le *Beta* Problème *difficile*

#### Présentation du problème

Nous avons conçu un problème *difficile* qui possède des dépendances circulaires entre ses variables (comme cela se passe dans certaines applications de traitement d'images par exemple). Nous avons choisi des formes paramétriques de telles sortes que nous puissions calculer une solution analytique à utiliser comme référence.

Nous définissons la distribution de probabilité conjointe sur $n$ variables binaires $B_i \in \{0,1\}$ et $n$ variables discrètes $V_j \in \{0,\ldots,2^m-1\}$ comme présenté sur l'équation 5.19 :

$$P(V_1 \wedge \cdots \wedge V_n \wedge B_1 \wedge \cdots \wedge B_n) = \left(\prod_{i=1}^{n} P(V_i)\right) \prod_{i=1}^{n} P(B_i | V_i V_{1+i\%n}) \quad (5.19)$$

#### Description des distributions

Le but de ce problème est de présenter un calcul d'inférence *difficile* mais dont la solution peut être approchée avec une très bonne précision de manière analytique. Pour cela nous utiliserons les fonctions de densité *Beta* sur lesquelles il est possible de réaliser les calculs d'intégrales, et notamment de normalisation. L'étude réalisée sur les distributions *Beta* nous amène à considérer des probabilités continues. C'est pourquoi les priors $P(V_i)$ sont décrits de tels sortes qu'ils deviennent une discrétisation des lois de probabilités continues $P(U_i)$, $U_i \in [0,1]$, définies par l'équation 5.20 :

$$V_i = k \in \{0,\ldots,2^m-1\} \Rightarrow U_i = u = (k+0.5)2^{-m} \in [0,1] \quad (5.20)$$

On choisit une loi Bêta pour représenter les $P(U_i)$ avec comme paramètres $\alpha = 2, \beta = 2$. On obtient alors, avec le terme $\beta(2,2)$ qui est la normalisation, l'équation 5.21 qui nous donne la probabilité $P(U_i = u)$ :

$$P(U_i = u) = \frac{u(1-u)}{\beta(2,2)} = \frac{u(1-u)}{\int_0^1 u(1-u)\mathrm{d}u} = 6u(1-u) \quad (5.21)$$

On a alors l'approximation pour le cas discret par la formule 5.22 :

$$P(V_i = k) \propto 6u(1-u)2^{-m} \tag{5.22}$$

Nous décrivons les fonctions de vraisemblances, $P(B_i = b_i|V_iV_{i+1\%n})$, en utilisant une loi de Bernoulli, dans l'équation 5.23 :

$$\begin{aligned} P(B_i = 1|V_i = k_1\ V_j = k_2) &= \frac{u_i^{a_i}(1-u_i)^{b_i}u_j^{c_j}(1-u_j)^{d_j}}{G(a_i,b_i,c_j,d_j)} \\ j &= 1+i \ \ [n] \\ u_i &= (k_1+0.5)2^{-m} \\ u_j &= (k_2+0.5)2^{-m} \end{aligned} \tag{5.23}$$

Le terme $G(a_i, b_i, c_j, d_j)$ est le max de la fonction :

$$g(u_i, u_j) = u_i^{a_i}(1-u_i)^{b_i}u_j^{c_j}(1-u_j)^{d_j}$$

qui permet d'avoir un paramètre

$$p_i = \frac{u_i^{a_i}(1-u_i)^{b_i}u_j^{c_j}(1-u_j)^{d_j}}{G(a_i,b_i,c_j,d_j)} \leq 1$$

homogène à une probabilité.

**Recherche du** $\max_{u_i,u_j\in[0,1]} g(u_i, u_j)$ **:**

$$\begin{aligned} &\frac{\partial g(u_i,u_j)}{\partial u_i} = 0 \\ \Leftrightarrow\ & a_i u_i^{a_i-1}(1-u_i)^{b_i}u_j^{c_j}(1-u_j)^{d_j} - b_i u_i^{a_i}(1-u_i)^{b_i-1}u_j^{c_j}(1-u_j)^{d_j} = 0 \\ \Leftrightarrow\ & a_i(1-u_i) - b_i u_i = 0 \\ \Leftrightarrow\ & u_i = \frac{a_i}{a_i+b_i} \end{aligned} \tag{5.24}$$

De la même manière :

$$\frac{\partial g(u_i,u_j)}{\partial u_j} = 0 \Leftrightarrow u_j = \frac{c_j}{c_j+d_j} \tag{5.25}$$

On en conclue que :

$$\begin{aligned} G(a_i, b_i, c_j, d_j) &= (\frac{a_i}{a_i + b_i})^{a_i}(1 - \frac{a_i}{a_i + b_i})^{b_i}(\frac{c_j}{c_j + d_j})^{c_j}(1 - \frac{c_j}{c_j + d_j})^{d_j} \\ &= \frac{a_i^{a_i}}{(a_i + b_i)^{a_i}} \frac{b_i^{b_i}}{(a_i + b_i)^{b_i}} \frac{c_j^{c_j}}{(c_j + d_j)^{c_j}} \frac{d_j^{d_j}}{(c_j + d_j)^{d_j}} \\ &= \frac{a_i^{a_i} b_i^{b_i} c_j^{c_j} d_j^{d_j}}{(a_i + b_i)^{a_i + b_i}(c_j + d_j)^{c_j + d_j}} \end{aligned} \tag{5.26}$$

### Le problème de l'explosion combinatoire

Nous considérons l'inférence suivante basée sur la spécification de notre modèle décrit précédemment : $P(V_i|b_1 = 1, \ldots, b_n = 1)$.

Dans cet exemple, $2^{m*n}$ additions sont nécessaires pour calculer l'inférence exacte, avec $n$ le nombre de variable et $m$ le $log_2$(dimension des variables $V_i$). Ce calcul devient vite impossible, même pour des valeurs de $m$ et $n$ qui paraissent raisonnables. Supposons que l'on a 20 variables binaires $B_i$ et 20 variables discrètes $V_i$ sur 256 valeurs chacune. On a donc $n = 20$ et $m = 8$. Dans ce cas, le nombre de sommes à réaliser vaut $2^{160} \simeq 10^{48}$. Si on utilise un processeur actuel, ayant une fréquence d'horloge de $2GHz$, alors il lui faudrait plus de $10^{30}$ années pour réaliser le calcul de l'inférence. Cependant, il est possible d'obtenir une approximation de la solution exacte par une méthode analytique.

### Résolution analytique

La question posée au modèle est $P(V_i|B_1 = 1 \ldots B_n = 1)$. En utilisant la règle de Bayes, on en déduit l'équation 5.27 :

$$\begin{aligned} P(V_i|B_1 = 1 \ldots B_n = 1) &= \frac{\sum\limits_{\substack{V_k \\ k \neq i}} P(V_1 \ldots V_n B_1 = 1 \ldots B_n = 1)}{P(B_1 = 1 \ldots B_n = 1)} \\ &= \frac{\sum\limits_{V_{i-1}} \sum\limits_{V_{i+1}} \sum\limits_{\substack{V_k \\ k \neq i, i-1, i+1}} \left( \prod\limits_{i=1}^{n} P(V_i) \right) \prod\limits_{i=1}^{n} P(B_i = 1 | V_i V_{1+i\%n})}{P(B_1 = 1 \ldots B_n = 1)} \end{aligned} \tag{5.27}$$

On extrait les termes qui dépendent de $V_i$, la factorisation donne l'équation 5.28 :

$$\begin{aligned}
& P(V_i|B_1 = 1 \dots B_n = 1) \\
= \; & \frac{1}{P(B_1 = 1 \dots B_n = 1)} \times P(V_i) \times \sum_{V_{i-1}} P(V_{i-1}) \\
& \times \sum_{V_{i+1}} P(V_{i+1}) P(B_{i-1} = 1|V_{i-1}V_i) P(B_i = 1|V_i V_{1+i}) \\
& \times \sum_{\substack{V_k \\ k \neq i, i-1, i+1}} \left( \prod_{l=1}^{n} P(V_l) \right) \prod_{l=1}^{n} P(B_l = 1|V_l V_{1+l}) P(B_1 = 1 \dots B_n = 1)
\end{aligned} \tag{5.28}$$

Les termes $\sum_{\substack{V_k \\ k \neq i, i-1, i+1}} \left( \prod_{l=1}^{n} P(V_l) \right) \prod_{l=1}^{n} P(B_l=1|V_l V_{1+l})$ et $P(B_1=1 \dots B_n=1)$ sont des constantes pour toutes les valeurs de $V_i$.

De plus, le passage aux probabilités continues nous permet d'écrire l'équation 5.29 :

$$\begin{aligned}
& P(U_i|B_1 = 1 \dots B_n = 1) \\
= \; & \frac{1}{Z} \times u_i(1 - u_i) \\
& \quad \times \sum_{U_{i-1}} P(U_{i-1}) u_{i-1}^{a_{i-1}} (1 - u_{i-1})^{b_{i-1}} u_i^{a_i} (1 - u_i)^{b_i} \\
& \quad \times \sum_{U_{i+1}} P(U_{i+1}) u_i^{c_i} (1 - u_i)^{d_i} u_{i+1}^{a_{i+1}} (1 - u_{i+1})^{b_{i+1}} \\
= \; & \frac{1}{Z} \times u_i(1 - u_i) u_i^{a_i} (1 - u_i)^{b_i} u_i^{c_i} (1 - u_i)^{d_i} \\
& \quad \times \sum_{U_{i-1}} P(U_{i-1}) u_{i-1}^{a_{i-1}} (1 - u_{i-1})^{b_{i-1}} \\
& \quad \times \sum_{U_{i+1}} P(U_{i+1}) u_{i+1}^{a_{i+1}} (1 - u_{i+1})^{b_{i+1}} \\
= \; & \frac{1}{Z} \times u_i^{1+a_i+c_i} (1 - u_i)^{1+b_i+d_i}
\end{aligned} \tag{5.29}$$

Avec $Z$, la constante de normalisation.

On remarque donc que $P(U_i|B_1 = 1 \dots B_n = 1)$ est une distribution *Beta* avec $\alpha = 2 + a_i + c_i$ et $\beta = 2 + b_i + d_i$. Il devient alors beaucoup plus simple de calculer la constante, qui est la constante de normalisation, et qui vaut $\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\mathrm{d}u$. Ce calcul est réalisé avec les outils logiciels d'approximation d'intégrale. La distribution cherchée, dans sa version continue, est donc la formule 5.30 suivante :

$$P(U_i|B_1 = 1 \dots B_n = 1) = \frac{u^{\alpha-1}(1-u)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\mathrm{d}u} \tag{5.30}$$

Étant donné que la question porte sur les variables discrètes $V_i$, nous sommes en mesure de passer de la distribution *a posteriori* en continue à la discrétisation souhaitée par la formule 5.31 :

$$\begin{aligned} P(V_i = k|B_1 = 1 \dots B_n = 1) &\approx \frac{\int_{\frac{k}{2^m}}^{\frac{k+1}{2^m}} u^{\alpha-1}(1-u)^{\beta-1}\mathrm{d}u}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\mathrm{d}u} \\ &\approx 2^{-m} Beta((k+0.5)2^{-m}, \alpha, \beta) \end{aligned} \tag{5.31}$$

Où $Beta(u, \alpha, \beta)$ est la fonction de densité de probabilité de la distribution *Beta* avec deux paramètres connus $\alpha$, $\beta \in \mathbb{N}$.

Notons que les $V_i$ sont une discrétisation des variables $U_i$. De ce fait, plus l'espace des valeurs possibles de $V_i$ est grand et plus l'approximation du résultat de la distribution *a posteriori*, par notre approche, est bonne.

### Résultats

Nous avons utilisé une instance calculable de l'inférence définie précédemment (avec $m = n = 5$) pour évaluer la qualité de notre approximation analytique. ProBT a été utilisé pour calculer $P(V_2|b_1 = 1 \dots b_5 = 1)$ en inférence exacte en marginalisant sur toutes les variables libres. Le programme est donné figure 5.12.

La différence absolue avec la solution analytique est toujours inférieure $10^{-17}$, ce qui valide la qualité de l'approximation analytique. De plus, l'augmentation de la dimension des variables, $m$, améliore la précision du calcul des intégrales et entraîne donc une augmentation de la précision de l'approximation analytique.

L'expérimentation du problème *difficile* a été réalisée avec un problème de taille $m = 8,\ n = 20$. Nous utilisons l'approximation analytique pour évaluer les résultats obtenus avec la machine *SMABI*. A noter que ce sont les *ECEs* qui ont été utilisés pour réaliser les produits de cotes.

```
from probt import *
V = plArray("V", plIntegerType(0, 1), 5)
B = plArray("B", plIntegerType(0, 31), 5)
model = plComputableObjectList()
b = plValues(B)
for i < 5 : do
   j = (i + 1) mod 32
   model = model * plDistribution(V[i], FV(i))
   model = model * plCndDistribution(B[i], V[i]V[j], FB(i))
   b[i] = 1 {tous les B[i] sont mis à 1}
end for
joint = plJointDistribution(model)
joint.ask(V[9],b).instantiate(b).VHDL('Ex.vhd')
```

FIGURE 5.12 – Spécification du *Beta* problème *difficile* décrit dans le langage ProBT : FV et FB sont des fonctions Python qui implémentent l'équation 5.23. Ce programme est compilé et un code VHDL est généré. Il décrit le fonctionnement du circuit réalisant l'inférence, il est simulé grâce à notre simulateur haut niveau. Une implémentation FPGA de ce circuit a été réalisée.

La précision dépend donc de :

— La taille $D$ des buffers utilisés dans les *ECEs* pour diminuer l'autocorrélation entre les bits de sortie.

— La durée de la période de *burn-in*, $BI$, permettant la stabilisation des *ECEs* pour la chaîne de bits encodant le produit $B(o_1 o_2)$.

— Le nombre $N$ d'échantillons choisis pour approximer la distribution *a posteriori*.

La figure 5.13 montre les résultats pour l'inférence $P(V_3|b_1 = 1, \ldots, b_{20} = 1)$ avec les paramètres suivants : $D = 16$, $BI = 200$, $N = 100000$. La taille $D$ des buffers des *ECEs* dépend de la profondeur de l'arbre d'*OPs*, c'est à dire du nombre d'*ECEs* cascadés. Dans l'exemple de la figure 5.2, la précision a été évaluée sur une cascade d'*ECEs*. Ici, nous utilisons les *ECEs* avec une taille de mémoire pour les buffers de $D$=16, ce qui constitue un bon compromis entre la taille et la précision pour l'exemple *difficile* dont la profondeur de la cascade est de deux. En effet, un bit de la conjointe est présent trois fois au maximum dans les termes de la décomposition.

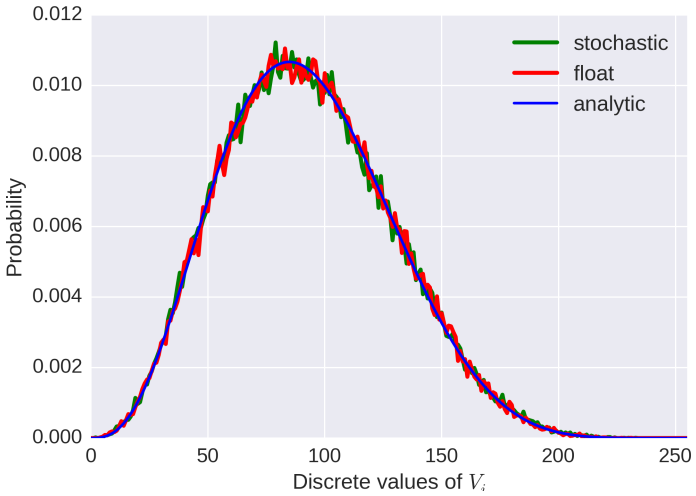



FIGURE 5.13 – Comparaison entre l'approximation calculée par la machine stochastique (en vert), avec l'utilisation de l'arithmétique standard flottante en utilisant le même algorithme (en rouge) et la solution analytique (en bleu)

La table 5.6 montre l'impact du *burn-in* $BI$ et du nombre $N$ d'échantillons sur la DKL entre l'approximation analytique et les résultats obtenus avec notre architecture stochastique. Nous observons que la précision augmente avec $N$, mais lentement. Ceci n'est pas surprenant puisque, comme mentionné précédemment, la précision des séquences de Bernoulli est en $O(\sqrt{N})$. L'augmentation de la longueur de la séquence de combustion, après quoi les échantillons calculés par l'arbre d'*ECEs* sont considérés comme valides, a un coût en termes de temps de calcul, mais elle permet d'augmenter significativement la précision des résultats jusqu'à ce qu'ils soient aussi bons que les calculs réalisés en virgule flottante. Ceci suggère une façon d'utiliser les mêmes circuits avec différents compromis entre la latence, puissance et précision souhaitée.

## 5.7 Discussion

Nous avons présenté une chaîne de synthèse logique concevant automatiquement des machines stochastiques résolvant une inférence bayésienne par une méthode approchée. L'algorithme implémenté dans ces machines est l'algorithme d'échantillonnage de Gibbs fonctionnant sur des variables binaires. Cette chaîne de synthèse est générique : elle traduit un problème d'inférence exprimé dans le langage ProBT et génère une description matérielle d'un

| Nombre d'échantillons | 1000 | 10000 | 100000 |
|---|---|---|---|
| burn in = 10 | 1.5293 | 1.2222 | 1.15294 |
| burn in = 20 | 1.3145 | 1.1482 | 1.13608 |
| burn in = 50 | 0.2447 | 0.1000 | 0.07979 |
| burn in = 100 | 0.1414 | 0.0178 | 0.00218 |
| burn in = 200 | 0.1333 | 0.0155 | 0.00182 |
| burn in = 400 | 0.1529 | 0.0151 | 0.00188 |
| calcul en flottant | 0.1458 | 0.0184 | 0.00181 |

TABLE 5.6 – Kullblack divergence à partir de la solution analytique (en bit).

circuit qui résout l'inférence demandée. Ce circuit est constitué uniquement de trois types de composants stochastiques : *CDE*, *BSGs* et *OP*. Nous avons conçu un test quantitatif afin d'évaluer notre architecture sur un problème insoluble en inférence exacte. Les résultats sont prometteurs. La distribution en sortie de notre machine stochastique suit précisément la courbe de la solution analytique et elle est aussi performante qu'une machine travaillant avec une représentation en virgule flottante des données.

En l'état, nous sommes donc capable de générer un circuit spécifique pour chaque modèle bayésien. Dans la suite, nous allons étendre le terme de machine puisque nous présentons une machine générique et programmable, qui accepte tout nouveau programme bayésien sans recourir à une nouvelle synthèse, ouvrant la voie à des dispositifs informatiques bayésiens effectivement programmables.

# Chapitre 6

# Machine bayésienne stochastique programmable implémentant une méthode d'inférence approchée

Ce dernier chapitre est l'aboutissement de la réflexion de ces trois années de recherche sur la conception d'un ordinateur probabiliste dédié aux inférences bayésiennes. Cette *PSMABI*, pour *Programmable Stochastic Machine for Approximate Bayesian Inference*, permet non seulement de résoudre les problèmes d'inférences de faibles dimensions et *difficiles* mais intègre une programmabilité qui permet d'avoir un unique circuit pour traiter une infinité de problèmes d'inférence. Cette machine générique peut donc être utilisée de la même manière qu'un ordinateur standard, où les couches de compilation et de calcul sur le circuit sont rendues invisibles pour le programmeur.

Nous présentons le dimensionnement choisi pour ce premier prototype de *PSMABI*, qui permet de résoudre l'inférence du problème *difficile* présenté dans le chapitre 5. Ensuite, nous explicitons l'architecture qui est composée d'un registre de Gibbs, d'une mémoire d'instructions ou de programme, d'une mémoire de données, du module de calcul qui est l'arbre d'*OPs*, d'une méthode d'adressage des données pour les *CDEs* et d'une machine à états qui régit l'ensemble du système. Nous expliquons le fonctionnement de la compilation pour programmer la machine et nous montrons les résultats sur les tests effectués sur le problème *difficile* du chapitre 5.

## 6.1 Problématique

La machine bayésienne *SMABI* présentée dans le chapitre 5 permet de résoudre des problèmes d'inférence en implémentant l'algorithme de Gibbs. A partir de n'importe quel problème, modélisé avec l'outil de programmation ProBT, nous sommes capable de générer une description VHDL du circuit qui résout l'inférence. A chaque problème est donc associée une machine bayésienne spécifique.

Dans ce chapitre, nous étendons la définition de *machine* par la possibilité de programmer, au sens commun, la *PSMABI*. Cette machine universelle est basée sur les mêmes principes que la machine *SMABI*. Elle est dite programmable car son implémentation reste la même et ce sont les programmes bayésiens qui viennent se "charger" dans la mémoire, après compilation, à l'instar des modèles de calculs des processeurs standards.

## 6.2 Principe

Cette machine fonctionne sur les principes proches des architectures de calculateurs standards. C'est à dire qu'elle est composée d'une unité de calcul, d'une mémoire de programme et d'une mémoire de données. Les instructions sont chargées dans la mémoire de programme, elles décrivent les étapes nécessaires à l'échantillonnage des bits par la méthode de Gibbs. Le principe de calcul est le même que celui présenté dans le chapitre 5, c'est à dire l'implémentation de l'algorithme de Gibbs en utilisant des produits de cotes. La programmation du modèle bayésien est toujours réalisée avec l'outil ProBT et le compilateur, dédié à cette machine, extrait les instructions et tous les quotients possibles du modèle, à stocker en mémoire, à partir des informations de la base de données ProBT. Les paramètres de cette machine étant fixes, c'est à dire le nombre de produits de cotes qui puissent être réalisées, la taille de la mémoire de données, la taille de la mémoire de programme et la taille du registre de Gibbs, les problèmes qui pourront être traités par cette machine devront répondre à certains critères de dimensionnalité.

## 6.3 Architecture de calcul

L'unité de calcul est la même que celle de la machine présentée dans le chapitre 5. C'est à dire qu'elle utilise des *CDEs* pour stocker tous les quotients et des *OPs* pour réaliser les produits de cotes.

## 6.4 Architecture physique

Dans cette section est présentée l'architecture de la machine basée sur les principes d'architecture des calculateurs standards, c'est à dire la mémoire de programme découplée de la mémoire de données et une unité de calcul. Le point difficile de l'architecture concerne la manière dont les données sont adressées, ce choix d'adressage est présenté dans le paragraphe 6.4.6. Enfin, la machine à états qui régit la machine est présentée en fin de section. Afin d'améliorer la lecture du fonctionnement des modules, nous utilisons l'exemple du *Sprinkler* comme modèle.

### 6.4.1 Vue d'ensemble de la machine

La figure 6.1 présente la vue d'ensemble de la machine programmable. Elle est composée de 5 éléments principaux qui sont :

— la mémoire de programme
— la mémoire des données
— l'unité de calcul
— le registre de Gibbs
— la machine à états

Le registre de Gibbs est composé d'un double banc de registres. Le premier stocke tous les bits de la conjointe qui sont mis à jour au fur et à mesure de l'exécution de l'algorithme d'inférence approchée. Le second permet de savoir à tout moment si la variable du registre de Gibbs est une variable observée ou non. Cette information est nécessaire pour connaître la position du bit échantillonné et réaliser l'adressage souhaité.

La mémoire de données est construite de la même manière que la mémoire dans les *SMABIs* présentées dans la section 5.3.1. Elle est divisée en sous-blocs nommés *CDE*. Chacun de ces sous-blocs stocke tous les quotients possibles pour un terme de la décomposition de la conjointe.

L'exécution de l'algorithme de Gibbs entraîne l'échantillonnage de chaque bit de la décomposition un nombre de fois nécessaire à la convergence vers la distribution visée. Pour chacun de ces bits, il est donc nécessaire de connaître les termes de la décomposition où ce bit intervient. En effet, tous les autres termes de la décomposition auront comme quotient l'élément neutre pour le produit de cotes, c'est à dire la valeur 1.

La mémoire de programme est donc remplie de manière à traverser le registre de Gibbs bit à bit. Pour chaque bit à échantillonner, nous avons besoin de connaître quels sont les *CDEs* mis en jeux. La première donnée stockée

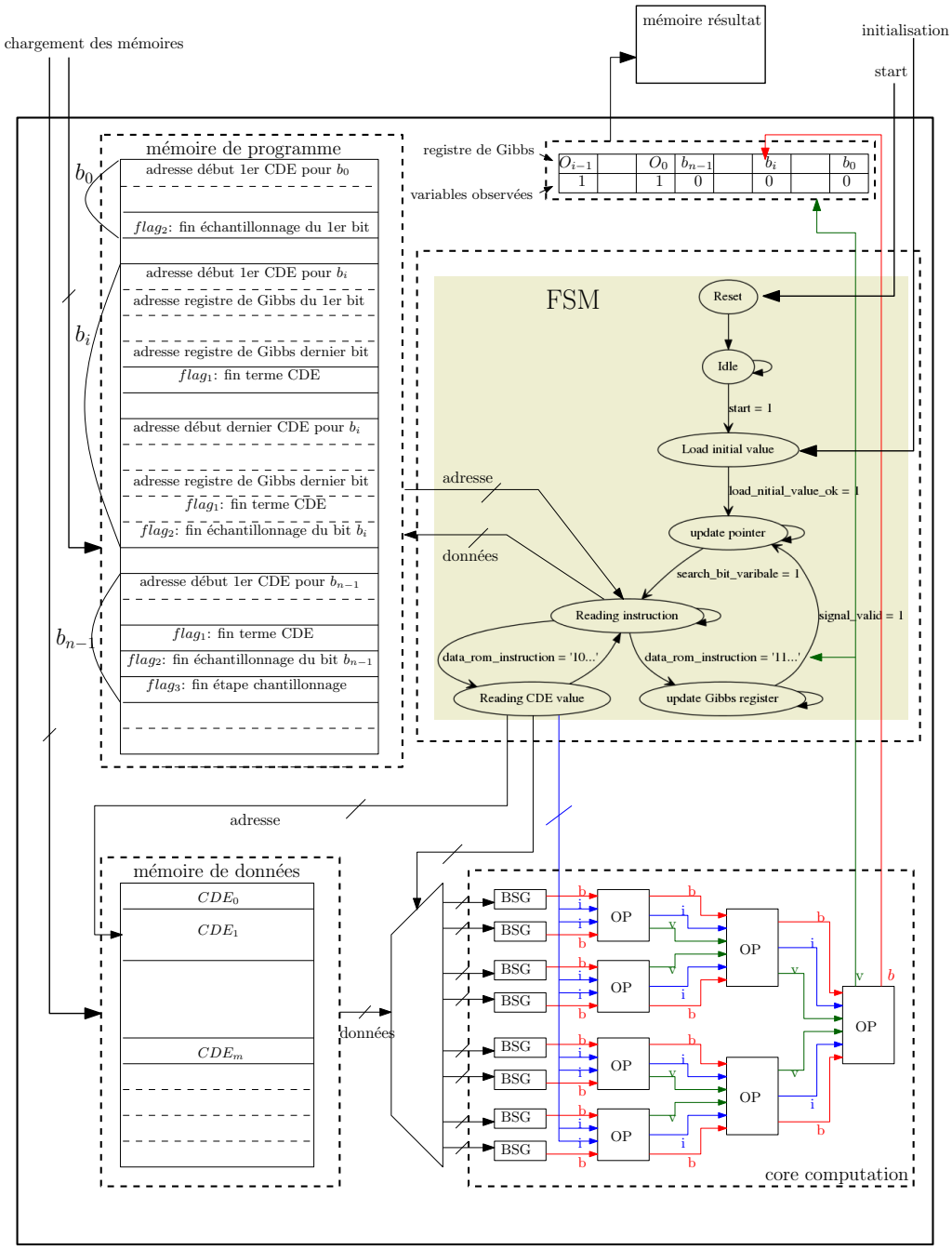



FIGURE 6.1 – Schéma de la machine bayésienne programmable pour l'inférence approchée

dans la mémoire d'instructions est donc l'adresse du début du premier *CDE* où apparaît la variable à échantillonner. Nous devons ensuite avoir accès aux valeurs des autres bits dont dépend ce *CDE* pour savoir où se situe la donnée cherchée dans le bloc du *CDE* visée. Nous utilisons alors autant de cases mémoires de la mémoire programme que de bits présents dans le terme de la décomposition. Ces cases stockent la position, dans le registre de Gibbs, du bit. Ainsi, nous avons accès à sa valeur. Le premier terme de la conjointe étant entièrement connu, nous sommes alors capable de récupérer la valeur du quotient dans la mémoire de données correspondant à ce terme. Nous développons cette méthode pour chaque terme de la conjointe où apparaît le bit à échantillonné. Des *flags* sont utilisés comme condition d'arrêt de la lecture des instructions.

Lorsque l'on a lu toutes les instructions, c'est à dire que l'on connaît tous les *CDEs* mis en jeux pour le bit que l'on souhaite échantillonner, on exécute alors le calcul des produits de cotes. Ce calcul est réalisé avec le même arbre d'*OPs* présenté dans la section 5.3.2. A la fin de ce calcul, on obtient la valeur d'un bit, 0 ou 1, qui contient le poids de la probabilité du bit échantillonné et l'on met à jour le registre de Gibbs avec la nouvelle valeur du bit.

La machine à états régit le fonctionnement de l'ensemble de ces blocs.

### 6.4.2 Registre de Gibbs

Le registre de Gibbs est un module simple composé d'un double banc de registres de taille 256. Ce dernier stocke tous les bits nécessaires à la description de la conjointe. Si dans celle-ci, les variables sont discrètes, alors une binarisation préalable est effectuée par la chaîne de compilation. A l'état initial, les registres correspondant aux bits à échantillonner sont remplis de façon aléatoire. Les bits de la conjointe correspondant aux variables observées sont remplis avec leurs valeurs définies dans le modèle. Le deuxième banc de registre permet de définir les variables à échantillonner.

**Exemple du *sprinkler* :**

Dans l'exemple du *sprinkler*, la question posée au modèle est $P(R|G = 1)$. Dans le tableau 6.1, le bit de la variable *Grasswet* est donc mis à 1 dans le registre de Gibbs. On remplit les deux autres bits avec des valeurs initiales choisies aléatoirement, ici $R = 1$ et $S = 0$. Le registre indiquant quels sont les bits observés, et donc lesquels sont à échantillonner, est rempli avec les

| Variables | G | S | R |
|---|---|---|---|
| Registre de Gibbs | 1 | 0 | 1 |
| Bit observé | 1 | 0 | 0 |

TABLE 6.1 – Registre de Gibbs pour l'exemple du *sprinkler*

valeurs $G = 1$, puisqu'il s'agit d'une variable observée, et $R = S = 0$, puisqu'il s'agit de variables à échantillonner.

### 6.4.3 Mémoire de programme

La mémoire de programme est une mémoire RAM où sont stockées les instructions, c'est à dire les bits à échantillonner. C'est la chaîne de compilation qui va récupérer les informations issues du modèle pour remplir cette mémoire. Pour chaque bit à échantillonner, nous stockons toutes les informations nécessaires à la connaissance des *CDEs* mis en jeux, afin de récupérer la valeur de tous les quotients dont la machine a besoin pour réaliser leur produit. Cette mémoire est lue séquentiellement par la machine. La première valeur lue correspond à l'adresse du début du *CDE* d'intérêt. Les valeurs suivantes sont les positions, dans le registre de Gibbs, des bits présents dans le terme de la conjointe correspondant. Un premier *flag*, $flag_1$, indique la fin de la lecture des instructions pour un terme de la conjointe mis en jeux. Un deuxième *flag*, $flag_2$, indique la fin de la lecture des instructions pour le bit échantillonné et donc le début du calcul des produits de cotes. Un troisième *flag*, $flag_3$, indique la fin de la lecture des instructions de tous les bits à échantillonner et le retour aux instructions du premier bit échantillonné.

**Exemple du *sprinkler* :** Le tableau 6.2 montre le remplissage de la mémoire programme pour l'exemple du *sprinkler*. Seules les variables *Rain (R)* et *Sprinkler (S)* sont à échantillonner. Pour chacune de ces variables nous devons identifier les blocs *CDEs* et les valeurs des bits présents dans ces blocs pour récupérer la valeur du quotient nécessaire pour réaliser les produits de cotes. Pour cet exemple, les valeurs stockées sont sur 7 bits, 5 bits pour adresser la mémoire de données et 2 bits de contrôle pour les *flags*. La variable *Rain* étant présente dans les trois termes de la décomposition, nous distinguons trois sous-blocs. De même la variable *Sprinkler* étant présente dans deux termes, nous distinguons deux sous-blocs pour l'échantillonnage de cette variable.

| variables à échantillonner | informations | adresse | valeur |
|---|---|---|---|
| R | adresse début du terme P(R) | 0 | 00 00000 |
| | position du bit R | 1 | 00 00000 |
| | $flag_1$ | 2 | 01 00000 |
| | adresse début du terme P(S\|R) | 3 | 00 00001 |
| | position du bit R | 4 | 00 00000 |
| | position du bit S | 5 | 00 00001 |
| | $flag_1$ | 6 | 01 00000 |
| | adresse début du terme P(G\|SR) | 7 | 00 00101 |
| | position du bit R | 8 | 00 00000 |
| | position du bit S | 9 | 00 00001 |
| | position du bit G | 10 | 00 00010 |
| | $flag_1$ | 11 | 01 00000 |
| | $flag_2$ | 12 | 10 00000 |
| S | adresse début du terme P(S\|R) | 13 | 00 00001 |
| | position du bit R | 14 | 00 00000 |
| | position du bit S | 15 | 00 00001 |
| | $flag_1$ | 16 | 01 00000 |
| | adresse début du terme P(G\|SR) | 17 | 00 00101 |
| | position du bit R | 18 | 00 00000 |
| | position du bit S | 19 | 00 00001 |
| | position du bit G | 20 | 00 00010 |
| | $flag_1$ | 21 | 01 00000 |
| | $flag_3$ | 22 | 11 00000 |

TABLE 6.2 – Organisation de la mémoire programme sur l'exemple du *sprinkler*

### 6.4.4 Mémoire de données - CDE

Les *CDEs* sont des modules mémoires RAMs qui stockent tous les quotients possibles des termes de la conjointe.

**Exemple du *sprinkler* :**

Le tableau 6.3 présente l'organisation de la mémoire de données pour l'exemple du *sprinkler*. On rappelle que le nombre de quotients à stocker par terme suit la formule $n2^{n-1}$, avec $n$ le nombre de bits présents dans celui-ci. Pour le terme $P(R)$, une seule adresse est nécessaire puisqu'une

| Termes de la conjointe | Bit échantillonné | valeurs des autres bits | adresse | valeur |
|---|---|---|---|---|
| P(R) | R | $\phi$ | 0 | 0100 1100 |
| P(S\|R) | R | S=0 | 1 | 1110 0110 |
| | | S=1 | 2 | 0001 1001 |
| | S | R=0 | 3 | 1110 0110 |
| | | R=1 | 4 | 0001 1001 |
| P(G\|RS) | R | G=0 S=0 | 5 | 0000 0000 |
| | | G=0 S=1 | 6 | 1000 0000 |
| | | G=1 S=0 | 7 | 1111 1111 |
| | | G=1 S=1 | 8 | 1000 0000 |
| | S | G=0 R=0 | 9 | 0000 0000 |
| | | G=0 R=1 | 10 | 1000 0000 |
| | | G=1 R=0 | 11 | 1111 1111 |
| | | G=1 R=1 | 12 | 1000 0000 |
| | G | S=0 R=0 | 13 | 0000 0000 |
| | | S=0 R=1 | 14 | 1111 1111 |
| | | S=1 R=0 | 15 | 1111 1111 |
| | | S=1 R=1 | 16 | 1111 1111 |

TABLE 6.3 – Organisation de la mémoire de données pour l'exemple du *sprinkler*

seule variable est présente. On a donc besoin de la valeur de ce quotient seulement lorsque c'est la variable *Rain (R)* qui est échantillonnée. Pour le terme $P(S|R)$, soit c'est la variable *Rain (R)* qui est échantillonnée et on a alors deux quotients possibles suivant la valeur de *Sprinkler (S)*, soit c'est la variable *Sprinkler (S)* qui est échantillonnée, alors, suivant la valeur de *Rain (R)*, il y a deux quotients possibles. On réalise la même chose pour le terme $P(G|RS)$.

On remarque que tous les quotients possibles sont bien présents, même ceux où l'on considère la variable G comme une variable échantillonnable alors que dans notre exemple elle est considérée comme une variable observée. Cette subtilité nous permet notamment de poser plusieurs questions au même modèle sans charger une nouvelle mémoire de données.

### 6.4.5 Module de calcul

Le module de calcul réalise le produit de cotes. C'est le même que celui présenté dans le chapitre 5.

### 6.4.6 Adressage des données

L'exécution de l'algorithme de Gibbs à travers la *PSMABI* est réalisée par le produit de cotes des termes de la conjointe qui représente le tirage d'un bit d'une variable cherchée. L'entrée de ces produits de cotes sont des *BSGs* qui génèrent des flux binaires avec la proportion isomorphe au quotient stocké dans le *CDE*. Il faut donc remplir les *BSGs* avec les bonnes valeurs pour commencer le calcul des produits de cotes. Le but de cette section est de présenter l'adressage des données stockées dans la mémoire de données nécessaire aux *BSGs* pour générer les flux binaires qui sont les entrées du module de calcul des produits de cotes.

Nous rappelons que pour chaque bit échantillonné, on lit les instructions de la mémoire de programme. Ces valeurs sont les adresses des débuts des blocs des *CDEs* mis en jeux dans l'échantillonnage de ce bit et la position, dans le registre de Gibbs, des bits présents dans ce terme. De même, la mémoire de données est divisée en sous-blocs qui sont les *CDEs* des différents termes de la conjointe. Chaque *CDE* est lui même divisé en sous-blocs suivant le bit échantillonné.

Pour réaliser l'adressage de la mémoire de données, nous avons besoin de connaître le *CDE* mis en jeux, le sous-bloc de ce *CDE*, c'est à dire la variable échantillonnée, et une façon d'adresser ce sous-bloc. Nous avons choisi d'adresser le sous-bloc avec les valeurs des bits présents dans le terme de la conjointe, sauf le bit échantillonné bien sûr. Le calcul de l'adresse du *CDE* est alors réalisé par la formule 6.1 suivante :

$$\begin{aligned} \text{Addr} &= \text{addr\_debut\_cde} + \text{addr\_sous\_bloc} + \text{addr\_value} \\ &= \text{addr\_debut\_cde} + p_{os}2^{k-1} + \text{"}b_{k-2}...b_0\text{"} \end{aligned} \tag{6.1}$$

Avec :

- $k$ : le nombre de bits présents dans le terme de la décomposition.
- $p_{os} \in [0..k-1]$ : la position du bit échantillonné par rapport aux $k$ autres bits.
- addr_debut_cde : l'adresse du début du *CDE* mis en jeux.
- addr_sous_bloc= $p_{os}2^{k-1}$ : l'adresse du sous-bloc du bloc *CDE*
- addr_value = "$b_{k-1}...b_0$" : la concaténation des valeurs des bits présents dans le terme de la décomposition, excepté le bit échantillonné.

Une fois que l'on a identifié le bloc *CDE* et le sous-bloc du *CDE* où est stockée la donnée à adresser, il faut connaître quelle est valeur à choisir dans ce sous-bloc. Il y a $2^{k-1}$ valeurs possibles pour un sous-bloc. La lecture de la mémoire d'instructions nous permet de récupérer la valeur de tous les bits présents dans le *CDE* correspondant. Chaque valeur de bit est alors un bit que l'on concatène afin d'obtenir l'adresse cherchée que l'on nomme *addr_value*. Ce dernier terme est donc un vecteur de bits qui est la concaténation des valeurs des bits présents dans le terme de la décomposition.

### 6.4.7 Machine à États

Cette section présente le fonctionnement de la machine à états qui régit l'ensemble du système {mémoire - calcul - registre de Gibbs}. Une description du passage d'un état à un autre est décrit figure 6.2.

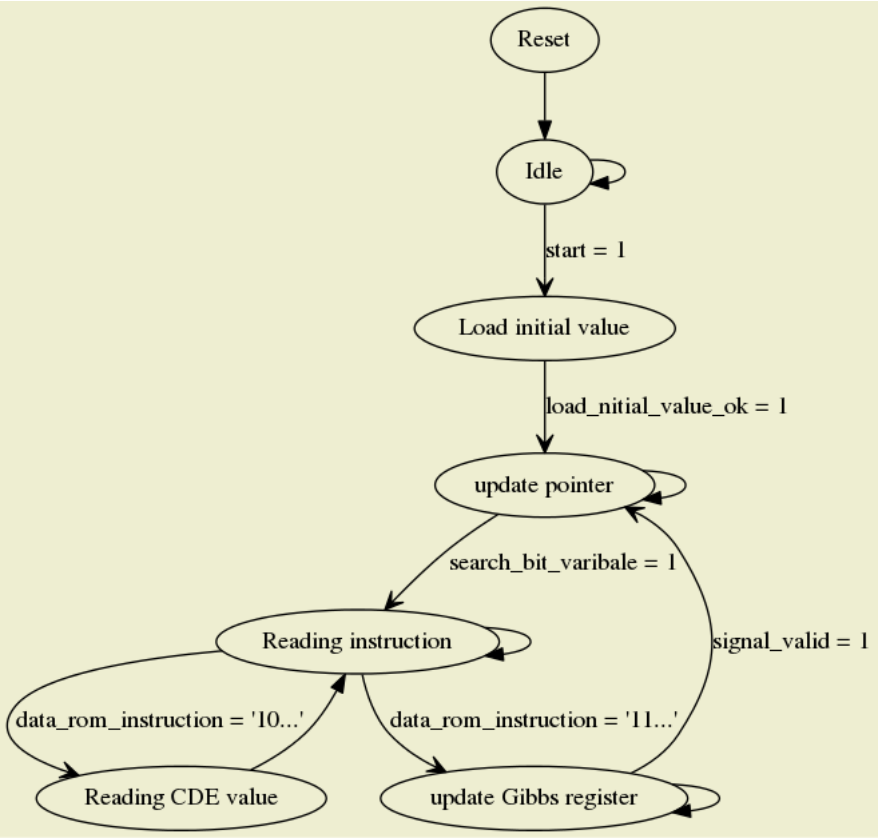



FIGURE 6.2 – Machine à états de la machine bayésienne programmable

L'état *Reset* permet de réinitialiser les paramètres de la machine, notamment le registre de Gibbs et le registre "variables cherchées" mis à 0, et les variables internes. Après le *Reset*, la machine passe directement dans l'état d'attente *Idle*. C'est son état stable où les variables et mémoires sont simplement rafraîchies à chaque front d'horloge. Pour démarrer l'exécution de l'algorithme de Gibbs, un signal de début d'activité, *start*, est mis à 1. La machine passe alors à l'état de chargement des valeurs initiales du registre de Gibbs et du registre «variables observées», c'est l'état *Load initial value*.

Lorsque le signal de validation *load_initial_value_ok* vaut 1, la machine passe dans l'état de mise à jour de la position du bit échantillonné, *Update pointer*. Celui-ci s'incrémente pour parcourir tout le registre de Gibbs et s'arrête lorsque pour une position d'un bit dans le registre de Gibbs correspond une valeur '0' dans le registre "variables observées", ce qui signifie qu'il s'agit bien d'un bit à échantillonner.

La machine passe ensuite dans l'état de lecture des instructions, *Reading instruction*. Elle lit et décode la mémoire d'instructions. C'est dans cet état qu'est réalisé le calcul de l'adresse de la valeur du terme du produit de cote correspondant à l'équation 6.1 dans la section 6.4.6. Le *flag* "01" sur les deux bits de poids forts d'une valeur dans la mémoire de programme indique la fin de la lecture des bits présents dans le *CDE* concerné et le passage à l'état *Reading CDE value*. Cet état correspond à la lecture et au stockage de la valeur du quotient du terme considéré pour générer le flux binaire nécessaire au calcul du produit de cotes. Au front d'horloge suivant, la machine retourne dans l'état *Reading instruction* et calcule la nouvelle adresse du nouveau terme du *CDE*. Si tous les termes ont été identifiés, la machine exécute le calcul du produit de cotes et passe dans l'état *Update Gibbs register*.

Elle reste dans cet état jusqu'à la fin du calcul. Afin de boucler l'algorithme de Gibbs, lorsque le signal de validation *signal_valid* est à 1, la machine retourne dans l'état de mise à jour de la position du nouveau bit à échantillonner, c'est à dire l'état *Update pointer*.

La variable *nombre_de_samples* permet de définir le nombre d'échantillons souhaité. Lorsque ce nombre est atteint, la machine retourne dans l'état *Idle* à la fin de l'échantillonnage.

Un résumé des états est donné dans le tableau 6.4.

| **Etat** | **Fonction** |
|---|---|
| Reset | Reset du système<br>met tous les paramètres et variables dans leur état initial |
| Idle | Etat d'attente du système |
| Load initial value | Chargement des valeurs initiales<br>du registre de Gibbs |
| Update pointer | Mise à jour de la position du bit échantillonné |
| Reading instruction | Lecture des instructions |
| Reading CDE value | Lecture et stockage de la valeur du quotient<br>pour le calcul des produits de cotes |
| Update Gibbs register | Mise à jour du registre de Gibbs<br>avec la nouvelle valeur du bit samplé |

TABLE 6.4 – Description des états de la machine à états

### 6.4.8 Une itération de la machine pour l'exemple du *sprinkler*

Dans cette sous-partie nous déroulons le fonctionnement de la machine pour un pas de l'algorithme de Gibbs, c'est à dire pour l'échantillonnage d'un bit.

Supposons que nous échantillonnons le bit S et que l'état du registre de Gibbs est le suivant, figure 6.5.

| Variables | G | S | R |
|---|---|---|---|
| Registre de Gibbs | 1 | 0 | 0 |
| Bit observé | 1 | 0 | 0 |

TABLE 6.5 – Registre de Gibbs pour l'exemple du *sprinkler*

La machine à états se trouvent alors dans l'état *Update pointer*. La valeur du pointeur de Gibbs, qui vise le bit à échantillonner, est donc mise à jour et vaut $p_G = 1$. La machine passe dans l'état *Reading instruction*. Il s'agit maintenant de récupérer toutes les informations nécessaires au calcul de l'adresse :

$$\begin{aligned} \text{Addr} &= \text{addr\_debut\_cde} + \text{addr\_sous\_bloc} + \text{addr\_value} \\ &= \text{addr\_debut\_cde} + p_{os}2^{k-1} + \text{"}b_{k-2}...b_0\text{"} \end{aligned}$$

L'algorithme permettant le calcul de *Addr* est décrit tout au long des paragraphes suivants. On note $p_G$, le pointeur de Gibbs qui donne la position

du bit échantillonné dans le registre de Gibbs. Dans notre exemple $p_G = 1$ tout au long de l'échantillonnage de $S$. La valeur $k$ est incrémentée à chaque fois qu'une instruction sur la position d'un bit est lue. A la fin de la lecture des instructions pour un *CDE* donné, nous avons alors la valeur $k$ qui vaut le nombre de bits dans un terme de la conjointe, comme souhaité.

### Algorithme de calcul de *Addr*

On lit la première instruction, en gras, qui se trouve à l'adresse 13 du tableau 6.6.

| variables à échantillonner | informations | adresse | valeur |
|---|---|---|---|
| R | adresse début du terme P(R) | 0 | 00 00000 |
| | position du bit R | 1 | 00 00000 |
| | $flag_1$ | 2 | 01 00000 |
| | adresse début du terme P(S\|R) | 3 | 00 00001 |
| | position du bit R | 4 | 00 00000 |
| | position du bit S | 5 | 00 00001 |
| | $flag_1$ | 6 | 01 00000 |
| | adresse début du terme P(G\|SR) | 7 | 00 00101 |
| | position du bit R | 8 | 00 00000 |
| | position du bit S | 9 | 00 00001 |
| | position du bit G | 10 | 00 00010 |
| | $flag_1$ | 11 | 01 00000 |
| | $flag_2$ | 12 | 10 00000 |
| S | **adresse début du terme P(S\|R)** | **13** | **00 00001** |
| | position du bit R | 14 | 00 00000 |
| | position du bit S | 15 | 00 00001 |
| | $flag_1$ | 16 | 01 00000 |
| | adresse début du terme P(G\|SR) | 17 | 00 00101 |
| | position du bit R | 18 | 00 00000 |
| | position du bit S | 19 | 00 00001 |
| | position du bit G | 20 | 00 00010 |
| | $flag_1$ | 21 | 01 00000 |
| | $flag_3$ | 22 | 11 00000 |

TABLE 6.6 – Organisation de la mémoire programme sur l'exemple du *sprinkler*

Cette première instruction correspond à la valeur de l'adresse du début du bloc qui stocke le *CDE* $P(S|R)$ :

$$adress\_debut\_CDE = 00001$$

On continue de lire les instructions tant qu'aucun *flag* indiquant l'arrêt n'apparaît. Les prochaines instructions sont toutes des positions de bit dans le registre de Gibbs. Elles permettent de calculer les valeurs de « addr_sous_bloc » et « addr_value » suivant l'algorithme présenté dans la figure 6.3. Un compteur $c$ nous permet de compter le nombre de bits avant le bit échantillonné.

**while** bit_de_controle = "00" **do**
  **if** data_memoire_programme = $p_g$ **then**
    $p_{os} = c$
  **else**
    $addr\_value[k] = Registre\_Gibbs[data\_memoire\_programme]$
    c +=1
    k +=1
  **end if**
**end while**
$addr\_sous\_bloc = p_{os} \times 2^{k-1}$

FIGURE 6.3 – Calcul de addr_sous_bloc et addr_value

L'instruction suivante donne alors la position du bit $R$ dans le registre de Gibbs. On a donc :

addr_value = "0"

On incrémente $k$ et $c$ :

$$k = c = 1$$

Ensuite, la nouvelle instruction donne la position du bit $S$ dans le registre de Gibbs. Or cette valeur de position est égale à la valeur de pointeur de Gibbs $p_G$ (en effet c'est bien $S$ qui est échantillonné). On connaît alors la valeur de $p_{os}$ :

$$p_{os} = c = 1$$

On incrémente seulement $k$ :

$$k = 2$$

La dernière instruction est le $flag_1$ qui signifie l'arrêt des lectures des instructions pour ce *CDE*.

Nous sommes donc en mesure de calculer l'adresse du quotient pour le terme $P(S|R)$ avec :

$$\begin{aligned} \text{addr\_debut\_cde} &= 00001 \\ \text{addr\_sous\_bloc} &= 1 \times 2^1 = 00010 \\ \text{addr\_value} &= 0 = 00000 \end{aligned}$$

Ce qui donne :

$$\begin{aligned} \text{Addr} &= \text{addr\_debut\_cde} + \text{addr\_sous\_bloc} + \text{addr\_value} \\ &= 00001 + 00010 + 00000 \\ &= 00011 \end{aligned}$$

La valeur à lire dans la mémoire de données se situe donc à l'adresse 3, ligne en gras dans le tableau 6.7.

| Termes de la conjointe | Bit échantillonné | valeurs des autres bits | adresse | valeur |
|---|---|---|---|---|
| P(R) | R | $\phi$ | 0 | 0100 1100 |
| P(S\|R) | R | S=0 | 1 | 1110 0110 |
| | | S=1 | 2 | 0001 1001 |
| | S | **R=0** | **3** | **1110 0110** |
| | | R=1 | 4 | 0001 1001 |
| P(G\|RS) | R | G=0 S=0 | 5 | 0000 0000 |
| | | G=0 S=1 | 6 | 1000 0000 |
| | | G=1 S=0 | 7 | 1111 1111 |
| | | G=1 S=1 | 8 | 1000 0000 |
| | S | G=0 R=0 | 9 | 0000 0000 |
| | | G=0 R=1 | 10 | 1000 0000 |
| | | **G=1 R=0** | **11** | **1111 1111** |
| | | G=1 R=1 | 12 | 1000 0000 |
| | G | S=0 R=0 | 13 | 0000 0000 |
| | | S=0 R=1 | 14 | 1111 1111 |
| | | S=1 R=0 | 15 | 1111 1111 |
| | | S=1 R=1 | 16 | 1111 1111 |

TABLE 6.7 – Organisation de la mémoire de données pour l'exemple du *sprinkler*

A ce moment la machine à états passe dans l'état *Reading CDE value* et stocke la valeur lue pour le *BSG* pour le futur produit de cotes à réaliser. La machine retourne dans l'état *Reading instructions*.

Il s'agit maintenant de récupérer la valeur du quotient pour le terme $P(G|SR)$. La première instruction nous donne :

$$\text{addr\_debut\_cde} = 00101$$

Les autres instructions nous permettent de calculer les valeurs de « addr_sous_bloc » et « addr_value ». Puisque $p_{os} = 1$ et qu'il y a 3 variables binaires, on a :

$$\text{addr\_sous\_bloc} = p_{os} \times 2^{3-1} = 4 = 00100$$

De plus, comme les valeurs de *Rain* et *Grasswet* dans le registre de Gibbs valent $R = 0; G = 1$, on a :

$$\text{addr\_value} = 00010$$

A la fin des lectures des instructions sur les positions des bits, la machine lit le $flag_1$ qui annonce la fin des instructions pour ce terme de la décomposition. On calcule alors l'adresse de la valeur du quotient :

$$\begin{aligned} \text{Addr} &= \text{addr\_debut\_cde} + \text{addr\_sous\_bloc} + \text{addr\_value} \\ &= 00101 + 00100 + 00010 \\ &= 01011 \text{ (en binaire)} \\ &= 11 \text{ (en décimale)} \end{aligned}$$

La machine passe dans l'état *Reading CDE value*, stocke la valeur à l'adresse 11, en gras dans le tableau 6.7, dans le *BSG* et retourne à l'état *Reading instructions*. La dernière instruction lue indique le $flag_3$, le pointeur de la mémoire d'instructions est donc remis à 0 afin de réitérer le processus d'échantillonnage des bits. La machine passe alors dans l'état *Update register* et attend la fin du calcul des produits de cotes. Lors de la réception du signal de validation, le registre de Gibbs est mis à jour avec la valeur du bit échantillonné et la machine retourne à l'état *Update pointer*.

## 6.5 Compilation

Les sections précédentes ont présenté l'architecture de la machine programmable et son fonctionnement. Avec le dimensionnement choisi, cette machine peut résoudre une grande quantité de problèmes d'inférence même de grandes dimensions, c'est à dire des problèmes traitables en inférence exacte mais aussi ceux intraitables avec cette méthode. C'est d'ailleurs l'un

des atouts principaux de l'utilisation d'algorithme d'échantillonnage. Toute la partie décrite précédemment est générique et auto-suffisante. Le programmeur n'a besoin que de modéliser son problème d'inférence avec ProBT et la chaîne de compilation organise le chargement des mémoires, paramètres et variables nécessaires comme entrées de la *PSMABI*.

De la même manière que les synthèses logiques présentées dans les chapitres 4 et 5, une partie de la compilation repose sur la base de données générée par ProBT lors de la modélisation d'un problème d'inférence. En effet, le compilateur pré-calcule tous les quotients possibles pour chaque terme de la décomposition et crée ainsi la mémoire de données où sont stockés toutes les valeurs des *CDEs*. De la même manière, il crée le jeu d'instructions et le tableau de correspondance pour la mémoire d'instructions. Le compilateur génère également la séquence d'initialisation du registre de Gibbs. Une fois toutes les mémoires chargées, on peut lancer l'exécution de la machine afin d'obtenir la distribution cherchée.

## 6.6 Expérimentations

La machine *PSMABI* a été dimensionnée pour résoudre une instance du problème *difficile* à 3 bits. Le nombre de variables binaires, $B_i$, et discrètes, $V_i$ est 20. La décomposition du modèle reste toujours la suivante :

$$P(V_1 \wedge \cdots \wedge V_20 \wedge B_1 \wedge \cdots \wedge B_20) = \left(\prod_{i=1}^{20} P(V_i)\right) \prod_{i=1}^{20} P(B_i | V_i V_{1+i\%20})$$

Elle a été synthétisée et simulée avec les outils Quartus II. Cette partie montre les résultats en simulation, avec ModelSim, post-synthèse. Nous avons testé la machine programmable sur deux exemples : *Sprinkler* et le *Beta* problème *difficile* à 3 bits.

### 6.6.1 L'architecture

Dans cette section nous décrivons l'architecture de la *PSMABI* qui a servi à la résolution des deux problèmes précités.

#### Dimensionnement

De la même manière qu'un ordinateur standard, qui peut stocker des données de taille 16, 32 ou 64 bits suivant les architectures, cette machine bayésienne générique possède des limites en capacité dues à sa structure.

Ces limites correspondent aux limites des composants de la machine :

— La taille du registre de Gibbs dépend du nombre de données binarisées du problème, c'est à dire de la conjointe.

— La taille de la mémoire programme dépend du nombre de variables dans le problème et surtout de leurs dépendances dans la décomposition.

— La taille de la mémoire pour stocker tous les *CDEs* est aussi liée aux nombres de variables et à leurs dépendances

— Le nombre de produits de cotes maximal que l'on peut faire pour un pas de l'algorithme de Gibbs, c'est à dire pour un bit échantillonné, qui dépend essentiellement de la dépendance entre les variables.

Nous dimensionnons la machine pour qu'elle puise résoudre la problème *difficile* à trois bits. La taille du registre de Gibbs dépend du nombre de bits des variables $V_i$ (3 dans notre cas) et des variables $B_i$ (1 car variable binaire). On en déduit :

$$\text{Taille registre de Gibbs} = 20 \times (3 + 1) = 80$$

La taille d'un *CDE* varie en fonction du nombre de bits $n$, présents dans le terme de la décomposition correspondant suivant la formule :

$$taille\_CDE = n2^{n-1}$$

On a donc besoin de $20 \times 3 \times 2^2 = 240$ données pour l'ensemble des $P(V_i)$ et $20 \times 7 \times 2^6 = 8960$ données pour l'ensemble des $P(B_i|V_iV_{i+1\%20})$. Chaque valeur de quotient est codée sur 32 bits On implémente donc une mémoire de 9200 mots de 32 bits.

Au regard de la décomposition de la distribution conjointe, on constate que, pour toute variable $V_i$, chaque bit à échantillonner est présent dans exactement 3 termes. Le nombre de données minimum dans la mémoire de programme dépend du nombre de bits dans les termes $P(V_i)$ (3 dans notre cas) et $P(B_i|V_iV_{i+1})$ (7 dans notre cas). On en déduit donc la taille de la mémoire de programme :

$$\text{Taille mémoire programme} = 20 \times 3 \times (3 + 7 \times 2) = 1020$$

Sachant que des *flags* seront nécessaires pour dissocier les pointeurs des bits échantillonnées et afin d'obtenir un nombre de cases mémoires puissance de deux, nous dimensionnons la mémoire d'instructions avec 2048 mots de 16

bits. Les deux bits de poids forts servent de bits de contrôle, ils correspondent aux trois types de *flags* nécessaires. Les 14 autres bits servent à adresser la mémoire de données.

Pour ce problème difficile, le nombre de produits de cotes à réaliser à chaque nouveau bit échantillonné est 3. Ce nombre montre une relative faible dépendance entre les variables dans ce cas particulier. C'est pourquoi, dans l'optique d'une machine générique, nous augmentons le nombre de produit de cotes possibles à réaliser à 8. Ce paramètre indique alors qu'une variable ne peut être présente dans plus de 8 termes de la décomposition.

**NB** : on peut facilement implémenter une autre machine générique qui aurait un nombre de produits de cote maximal supérieur. A noter qu'il en est de même pour tous les autres paramètres.

Voici le résumé des dimensions des paramètres du premier prototype de machine programmable :

— Taille du registre de Gibbs : $80 \times 2$ registres
— Taille de la mémoire d'instructions : $2048 \times 16$ bits
— Taille de la mémoire données : $320ko$
— Nombre de produits de cotes maximal : 8

Les figures 6.4 et 6.5 montre le *top level* et le module de calcul de l'implémentation de la *PSMABI*.

### Ressources

La mémoire de données a été dimensionnée pour s'adapter à la taille nécessaire pour stocker tous les quotients dont on a besoin. Le tableau 6.8 résume les différents éléments utilisés par la machine. La mémoire RAM qui stocke les *CDEs* utilisent $317K$ bits. Le nombre de registres utilisé correspond aux deux registres nécessaires pour le registre de Gibbs et aux différentes valeurs à stocker dans la machine à états. Le nombre de composants combinatoires correspond aux éléments nécessaires pour réaliser les produits de cotes et générer les chaînes de bits.

| **Entity** | **LC Combinational** | **LC Registers** | **Memory Bits** |
|---|---|---|---|
| **Total** | 2301 | 873 | 317 472 |

TABLE 6.8 – Ressources nécessaires pour l'implémentation de la *PSMABI* dimensionnée pour le problème *difficile* à 3 variables

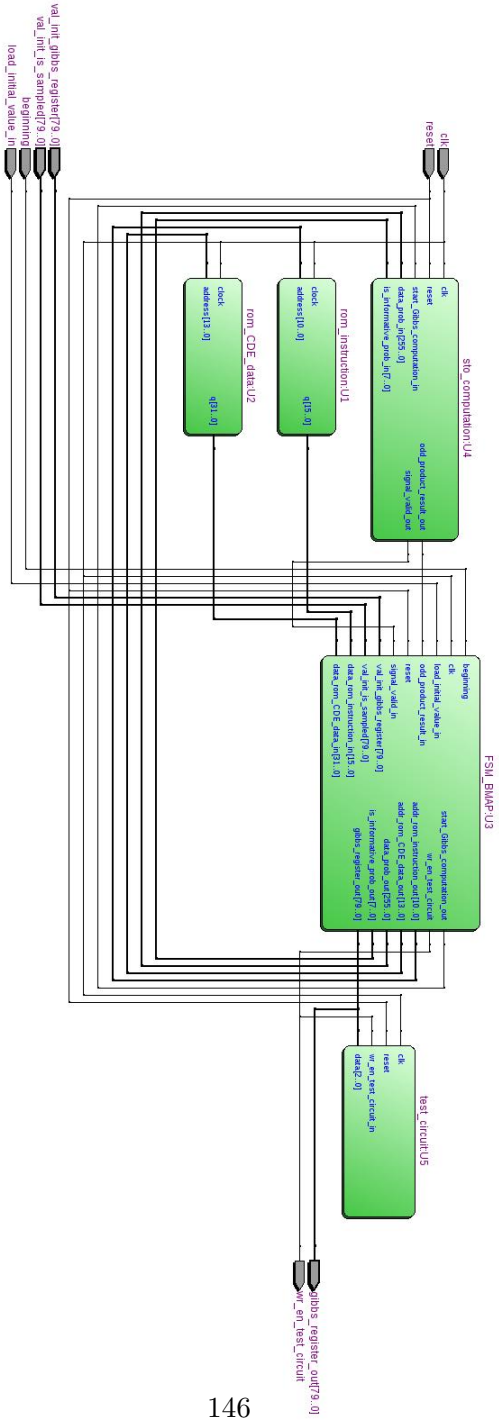





FIGURE 6.4 – Schéma rtl du *top level* pour la *PSMABI*

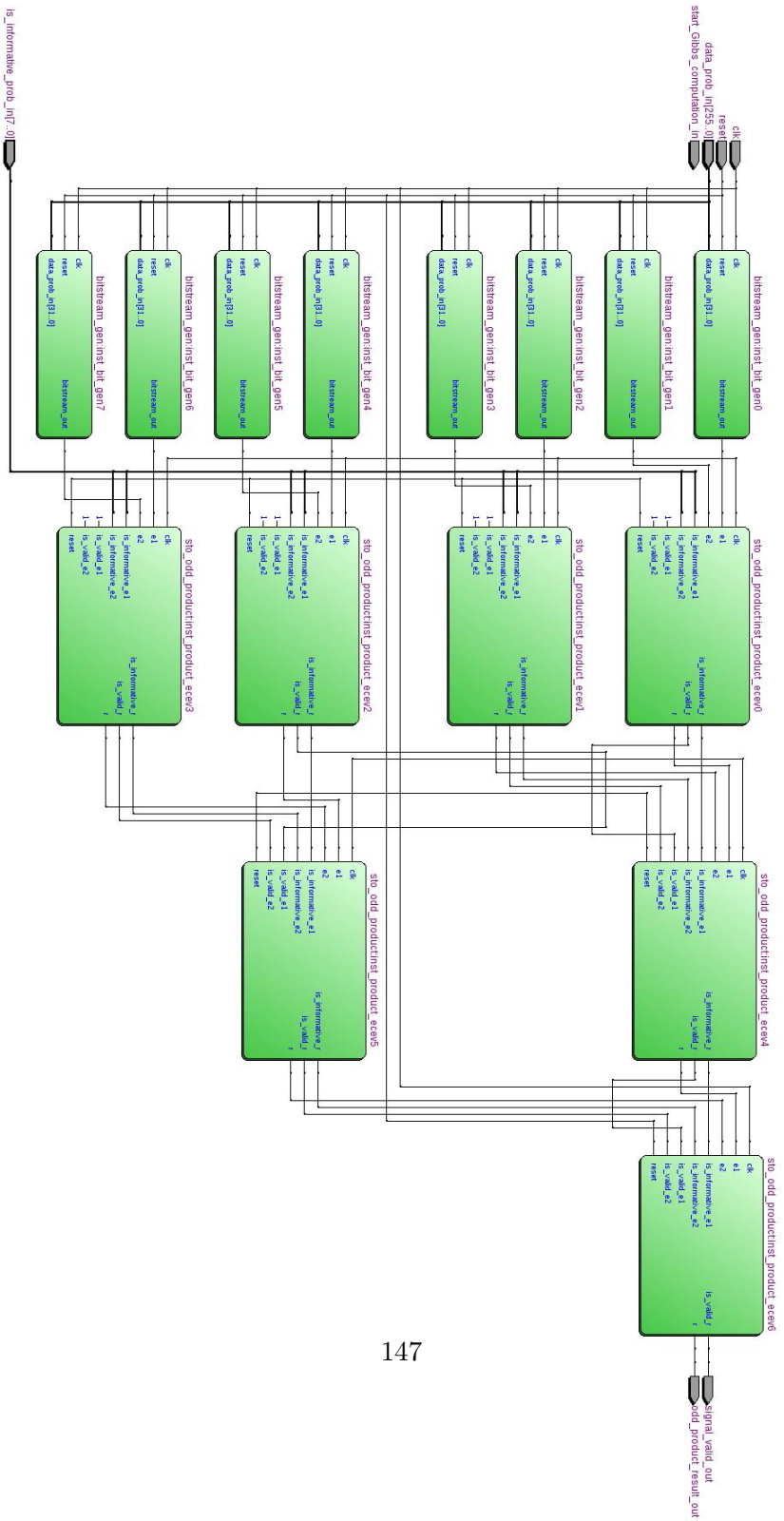





FIGURE 6.5 – Schéma rtl du module de calcul pour la *PSMABI*

### 6.6.2 *Sprinkler*

Nous avons dans un premier temps simulé l'exemple du *Sprinkler* sur la machine. La question posée au modèle $P(RSG)$ est $P(R|G = 1)$. La figure 6.6 montre la divergence de Kullback-Leibler du résultat sur la variable cherchée *Rain* pour des expériences allant de 100 à 50000 tirages. Le résultat exact a été calculé avec ProBT. Il était attendu que la *DKL* décroisse drastiquement avec le nombre d'échantillons tirés.

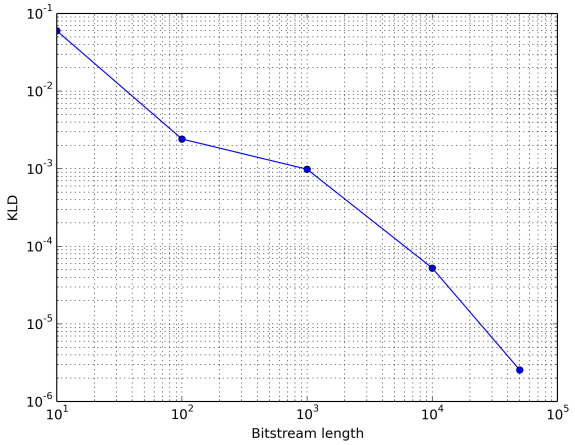



FIGURE 6.6 – Divergence de Kullback-Leibler sur la variable *Rain* pour le problème du *sprinkler*

### 6.6.3 *Beta* problème *difficile* à 3 bits

#### Résultats

La figure 6.7 compare les résultats de la machine sur la variable $V_2$ avec le calcul de l'inférence exacte réalisée par ProBT pour des expérimentations allant de 100 à 50 000 échantillons.

Dans cette simulation, nous avons utilisé le *LFSR* comme générateur aléatoire. On remarque alors une stagnation de la forme de la distribution pour un nombre d'échantillons supérieurs à 10000. Cette analyse qualitative se retrouve sur la figure 6.8 qui montre la valeur de la divergence de Kullback-Leibler sur la variable $V_2$ entre la référence et la distribution approchée par la machine. On note une stagnation de la *DKL* autour de $2.10^{-3}$ à partir de 10 000 échantillons.

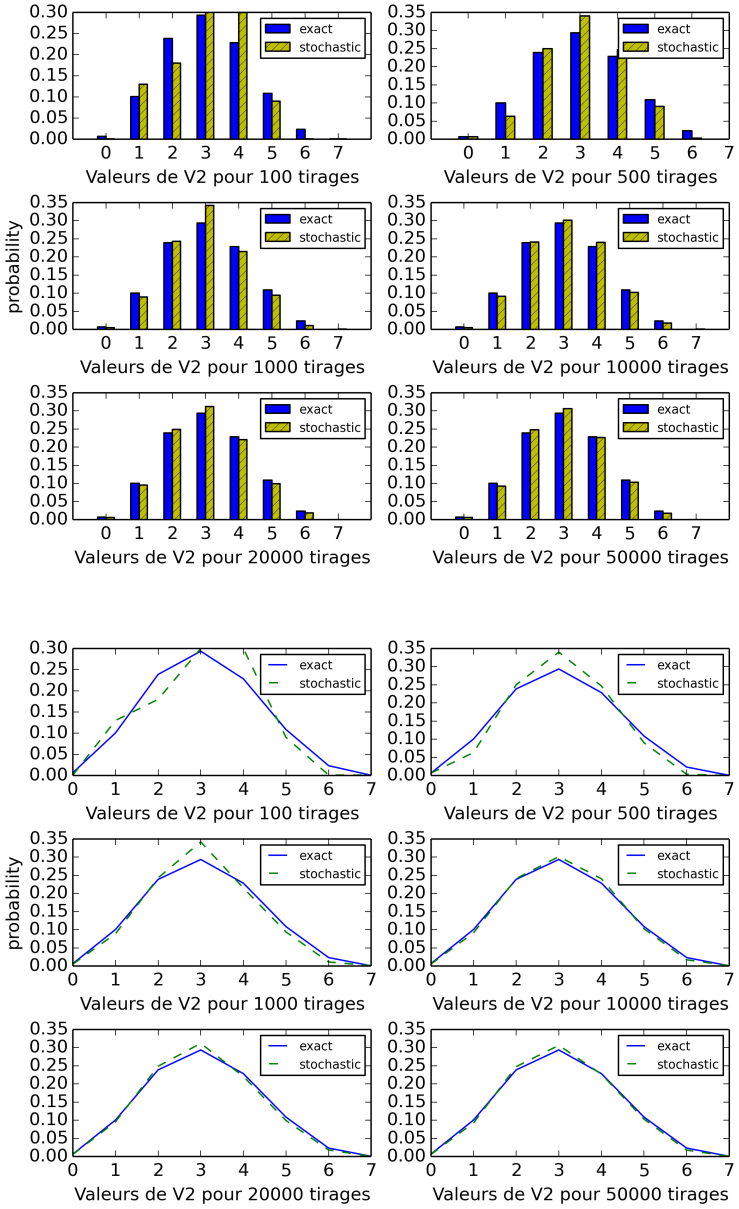



FIGURE 6.7 – Comparaison de la distribution calculée en inférence exacte et approchée avec la *PSMABI* en fonction du nombre d'échantillons

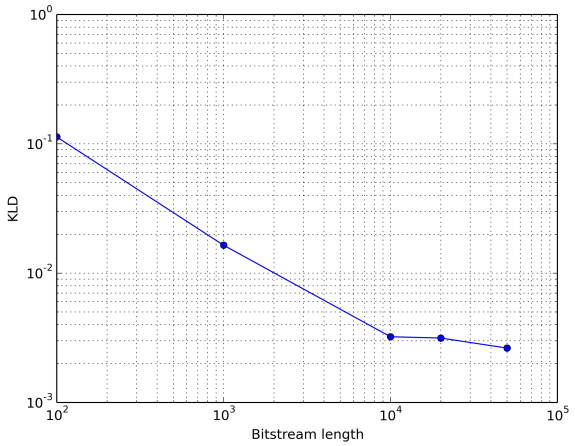



FIGURE 6.8 – Divergence de Kullback-Leibler sur la variable $V_2$ entre la référence et la distribution approchée par la machine avec un LFSR

Ceci s'explique simplement par le générateur utilisé pour cette expérience. Il s'agit du LFSR dont la qualité d'aléa est insuffisante pour aller en deçà du seuil présenté. Nos partenaires du laboratoire ISR (*Institute of Systems and Robotics*), au Portugal, ont approfondi l'étude en implémentant un générateur aléatoire Mersenne-Twister, qui linéarise la courbe de la figure 6.8. La courbe 6.9 montre ce phénomène.

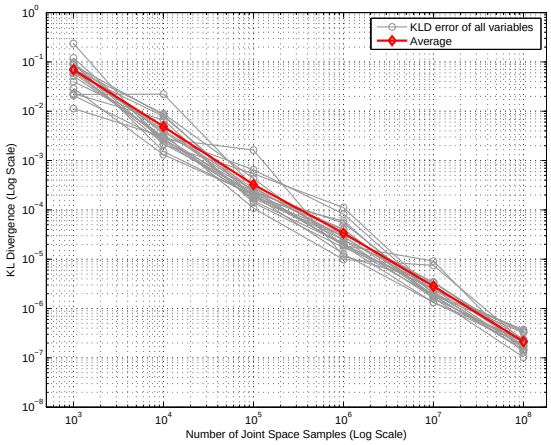



FIGURE 6.9 – Divergence de Kullback-Leibler sur la variable $V_2$ en fonction du nombre d'échantillons avec Mersenne-Twister

### Étude énergétique

L'outil Quartus nous permet également d'avoir accès à une simulation de la consommation moyenne du FPGA visé lors de la synthèse, tableau 6.9.

| | **3-bits** |
|---|---|
| Core Dynamic Thermal Power Dissipation [mW] | 30.7 |
| Core Static Thermal Power Dissipation [mW] | 118.3 |
| I/O Thermal Power Dissipation [mW] | 84.6 |
| Total Thermal Power Dissipation [mW] | 233.6 |

TABLE 6.9 – Consommation estimée du circuit FPGA pour la résolution du problème *difficile* sur 3 bits

Ce tableau nous permet d'estimer l'énergie nécessaire pour générer les échantillons. En moyenne, la machine a besoin de 800 cycles pour générer un échantillon, soit 6400 cycles pour mettre à jour l'ensemble du vecteur de Gibbs. La fréquence d'horloge du FPGA visé est de $50MHz$. On estime alors l'énergie nécessaire, en joule, par la formule 6.2 suivante :

$$\begin{aligned} \text{energie} &= power \times \frac{\text{nb\_cycle\_par\_sample} \times \text{nb\_samples}}{\text{fréquence\_horloge}} \\ &= 200 \cdot 10^{-3} \times \frac{6400 \times \text{nb\_samples}}{50 \cdot 10^{6}} \\ &= 0.25\mu J \text{ par échantillon} \end{aligned} \tag{6.2}$$

Nous avons mis en place un simulateur logiciel avec du code C++ généré à partir du code VHDL de la *SMABI* qui réalise les produits de cotes avec du calcul flottant. La librairie powerAPI permet, à chaque milliseconde, d'estimer la consommation du CPU pour un processus particulier à partir du pourcentage d'utilisation du CPU par le process et des caractéristiques du CPU. Avec ces données nous calculons la puissance moyenne et la consommation énergétique sur la durée du calcul du processeur. Il en résulte que pour atteindre une $DKL = 2.10^{-3}$, un ordinateur standard utilise une énergie de $1.25J$ lorsque la *PSMABI* s'exécutant sur le FPGA n'a besoin que de $0.256J$. Ce facteur 10 est à mettre en relation avec l'architecture *SMABI* dont l'étude énergétique comparée à un CPU a été réalisée par les chercheurs de l'ISR.

La figure 6.10 montre l'évolution de la DKL en fonction de l'énergie consommée pour une *SMABI* et un CPU. Les chercheurs de l'ISR ont pu

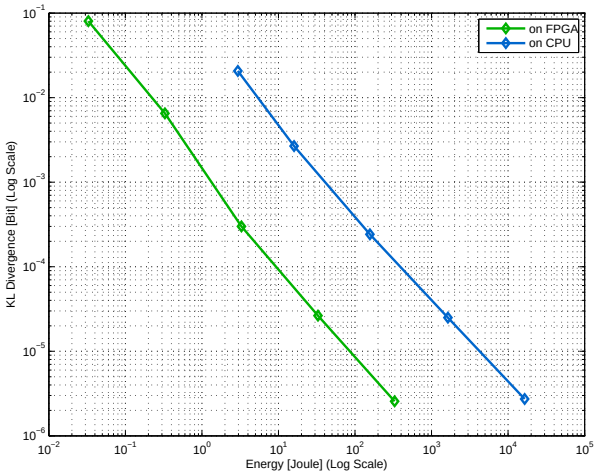



FIGURE 6.10 – Comparaison de l'énergie consommé par le FPGA et celui d'un CPU en fonction de la DKL. Le générateur utilisé ici est Mersenne-Twister

aller plus loin dans la DKL grâce à l'utilisation du générateur Mersenne-Twister. Le gain ici est de deux ordres de grandeurs. Ce qui s'explique très bien par le fait que les *CDEs* étant «câblés» sur le module de calcul, le nombre de cycle d'horloge nécessaire à l'échantillonnage du vecteur de Gibbs s'en retrouve considérablement réduit et par conséquent l'énergie utilisée également.

Cette simple comparaison énergétique nous pousse à poursuivre les investigations sur l'efficacité énergétique des machines présentées. En effet, nous avons mentionné à plusieurs reprises l'intérêt du calcul stochastique dans la production de circuits à faible consommation. Nos résultats tendent à aller en ce sens, il est alors nécessaire d'améliorer qualitativement et quantitativement l'étude énergétique si l'on souhaite tirer des conclusions assurées.

## 6.7 Discussion

Nous avons présenté une machine probabiliste programmable capable d'approcher la solution de n'importe quelle inférence bayésienne. Cette machine générique utilise un compilateur, basé sur l'outil ProBT, qui transforme tout programme bayésien en une série d'instructions qui permettent à la machine d'échantillonner la distribution cherchée. Le principe de son

cœur de calcul est l'algorithme de Gibbs dans le cas binaire comme pour les machines *SMABIs*. Le *Beta* problème a été utilisé comme preuve de concept de ce prototype. La difficulté réside dans le remplissage et l'adressage des données du modèle.

Ce gain en flexibilité se paie en vitesse d'exécution, par rapport aux machines *SMABI*, notamment lorsque les termes de la conjointe sont de grandes dimensions. En effet, nous avons dans ce cas de longues chaînes d'instructions donnant la position des bits présents dans ces termes. Le temps nécessaire à la connaissance de tous les bits est proportionnel au nombre de bits présents dans le terme. Dans la conclusion nous proposons des voies pour contourner cette contrainte.

# Chapitre 7

# Conclusions, perspectives et philosophie bayésienne

On ne saurait dire si l'avenir des machines probabilistes est proche, lointain ou utopique. Cependant, au cours des ces trois années de recherche, nous nous sommes efforcés de démontrer la pertinence d'une approche autre que celles érigées pour nos machines actuelles. Nous espérons que les travaux présentés forment un plaidoyer convaincant et qui fait sens dans tous les domaines touchés par le calcul. Dans le cadre de la conception de circuits, nous sommes capables de proposer des machines dédiées à une application et donc moins énergivores, mais nous avons également pensé à des machines génériques permettant la programmation d'une machine bayésienne.

Dans ce chapitre, nous concluons sur les travaux présentés dans ce mémoire et nous ouvrons la voie à des changements, améliorations et futures contributions possibles. A court et moyen termes nous pouvons imaginer des structures de machines où la mémoire et le cœur du calcul se rapprochent physiquement au sein du circuit, permettant un gain en vitesse remarquable. A plus long terme, nous rêvons du couplage entre une technologie de nouveaux nano-composants matures avec la logique CMOS utilisée dans nos machines conçues.

## 7.1 La fin du préambule

Nous avons réalisés deux types de machine qui permettent de résoudre des problèmes spécifiques. La *SMTBI* a été conçue dans le but de montrer la faisabilité de l'utilisation de signaux stochastiques traversant des portes logiques standards afin de réaliser les opérations nécessaires, addition, multiplication et normalisation, au calcul de l'inférence exacte. On a montré que, même si ce type de machine ne pouvait prétendre à résoudre des problèmes de grandes dimensions, elle permet de s'attaquer à des applications où le nombre de variables n'entraîne pas d'explosion combinatoire. Les télécommunications, avec notamment le problème d'acquisition de séquences pseudo bruitées, ont été un candidat crédible à la démonstration de la pertinence de cette machine.

Les problèmes d'inférence étant NP-complet, nous nous sommes orientés vers les méthodes de résolution approchée afin de répondre aux applications à grandes dimensions. L'algorithme de Gibbs a été choisi comme méthode MCMC pour être le cœur du calcul des *SMABIs*. Nous avons montré qu'avec l'implémentation de seulement trois types de composants, nous sommes capables d'approcher la solution de l'inférence pour des problèmes *difficiles*.

Pour ces deux machines, nous avons conçu une chaîne de synthèse logique qui prend en entrée un programme ProBT et spécifie, automatiquement, le circuit, sous forme d'un code VHDL, réalisant le calcul d'inférence souhaitée.

Devant les résultats prometteurs de cette *SMABI*, nous avons conçu la *PSMABI* qui peut être considérée comme la première machine bayésienne dans le sens où elle abstrait son fonctionnement à l'utilisateur qui peut réaliser autant de calculs d'inférence sur autant de problèmes bayésiens avec cette même machine. Dans ces travaux, nous avons bâti les fondements d'une machine, au sens générique du terme, et de son environnement de programmation.

Cependant, ces premiers résultats encourageants ne satisfont pas tous les critères que nous nous sommes fixés. En effet, certaines difficultés persistent et entravent les performances pour d'autres applications. Pour l'instant, tous les circuits présentés utilisent une horloge centrale. Cette idée va à l'encontre de la théorie sur la façon dont les êtres vivants calculent. Un objectif est de passer à des flux binaires stochastiques asynchrones, appelés signaux télégraphiques, où aucune horloge centrale ne serait nécessaire. Les MTJs sont de bons candidats pour relever ce défi. Ils impliquent de revoir les architectures présentées pour les faire fonctionner de manière asynchrone.

L'utilisation des mémoires, l'interface entre signaux analogiques et circuits digitaux, les principes globaux d'une architecture stochastique asynchrone pour la génération automatique de telles machines sont autant de champs de recherche à explorer.

## 7.2 Le début de l'histoire

La *PSMABI* conçue possède de nombreuses voies d'améliorations mimées sur ce qui se fait sur les architectures de calculateurs standards. Dans un premier temps, nous pouvons ajouter une mémoire cache, proche du cœur du calcul, qui aura pour rôle d'accélérer les données transmises à l'arbre d'*OPs*. Dans la même approche, nous pouvons pipeliner les adresses transmises afin de minimiser la latence et ainsi améliorer la vitesse d'exécution de l'algorithme de Gibbs. Une autre voie possible est la multiplication des machines à états afin d'échantillonner plusieurs bits. Enfin, nous présentons une façon simple d'ajouter les MTJs comme générateurs aléatoires nécessaires à la machine.

### 7.2.1 Cache

De la même façon que les architectures de processeurs, le temps d'accès à la mémoire est un point fondamental de la vitesse d'exécution de la machine. Dans un premier temps, la *PSMABI* a été conçue avec une mémoire RAM où est stockée l'ensemble des valeurs des distributions connues. A chaque nouveau bit échantillonné, à chaque *CDE* visé, la valeur en mémoire à chercher diffère. Dans ce cas, la requête prend plusieurs cycles ce qui contraint le cœur de calcul à attendre la donnée la plupart du temps durant son fonctionnement. L'idée est donc d'ajouter une mémoire cache proche de ce cœur de calcul et de copier une partie de la mémoire RAM dans celui-ci. Au vu de l'organisation en tableau de *CDE* de la mémoire, le cache n'aurait que peu d'intérêt puisque les données ne sont, par définition, pas contiguës au sein des *CDEs*. On pourrait donc remplacer l'organisation par terme de la conjointe par une organisation par bit à échantillonner, ce qui rendrait efficace l'agencement architecturale de la machine avec une mémoire cache, voir la figure 7.1. En effet, cette représentation permet d'avoir les éléments nécessaires à l'échantillonnage d'un bit concentré sur des blocs contigus de la mémoire.

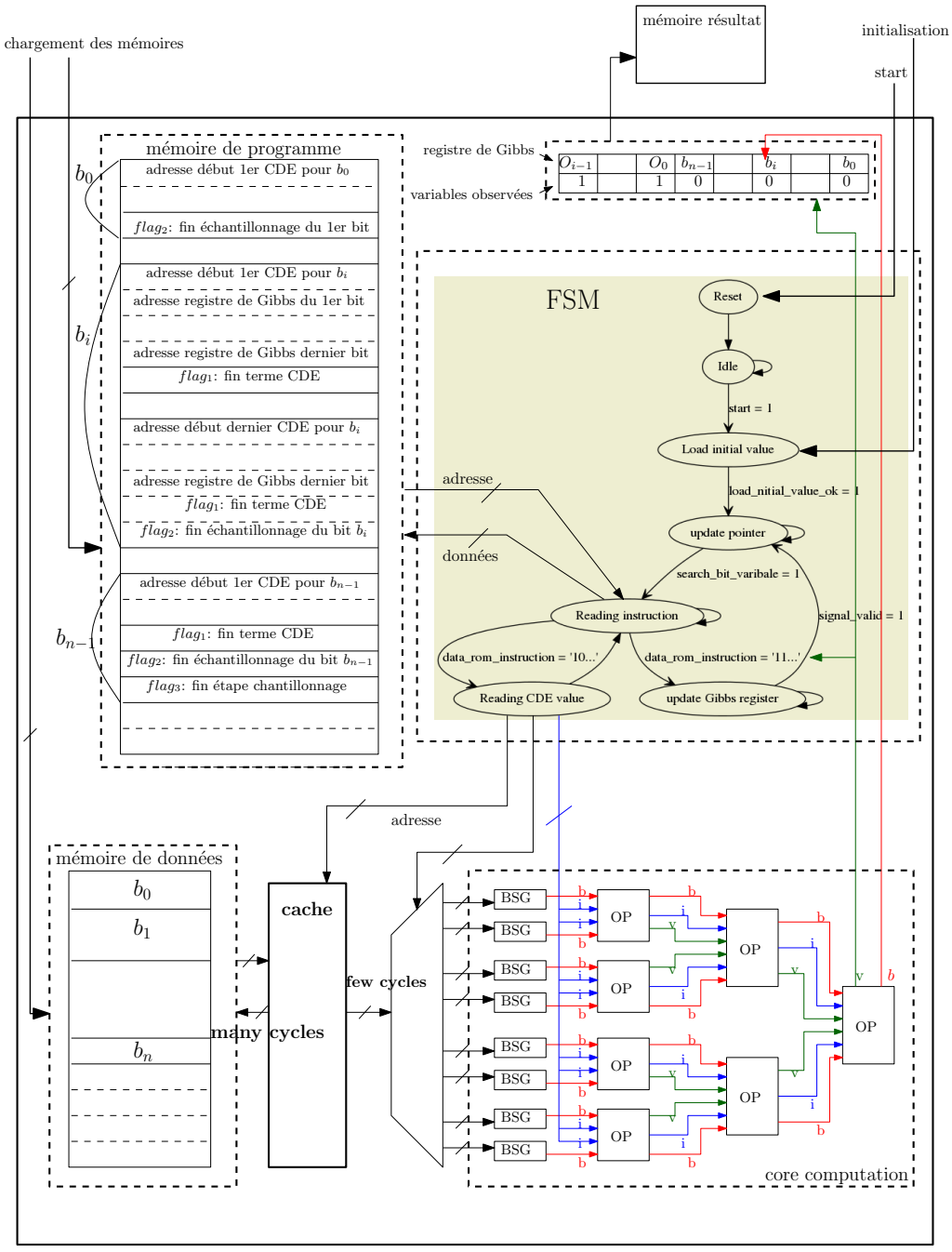



FIGURE 7.1 – Ajout de mémoire cache dans la *PSMABI*

### 7.2.2 Pipeline

A l'instar du pipeline réalisé dans les processeurs modernes, on peut imaginer un étage de pipeline au niveau du fonctionnement de la machine à états qui permettrait de générer un nouveau calcul d'adresse, pour un autre échantillon, pendant que l'arbre d'*OPs* réalise sa fonction.

### 7.2.3 Multi-FSM

Comme les plate-formes multicœurs, nous proposons d'étudier les performances d'une *PSMABI* à plusieurs machines à états 7.2. Celles-ci permettraient d'échantillonner plusieurs bits du registre de Gibbs et ainsi accélérer possiblement la convergence. Une étude préalable de la convergence de l'algorithme est nécessaire puisque dans cette hypothèse entre en jeu la façon dont les bits influent les uns sur les autres.

### 7.2.4 MTJ

La mémoire est un module imposant, donc coûteux et qui réduit la vitesse d'exécution car son accès est limité. Dans le projet BAMBI, l'idée directrice a toujours été de construire une machine dont la base serait le couplage de nouveaux nano-composants avec la logique CMOS. L'architecture de notre machine laisse la voie libre au passage de l'utilisation de nano-composants comme les MTJs. Comme le montre la figure 7.3, nous pensons remplacer le stockage en virgule fixe de tous les quotients de la conjointe et les générateurs aléatoires associés par des tables de MTJs qui posséderaient toute l'information des distributions de vraisemblances et généreraient les flux stochastiques nécessaires au module de calcul. Il s'agirait alors de savoir adresser les bons MTJs, adressage déjà réalisé dans la *PSMABI*.

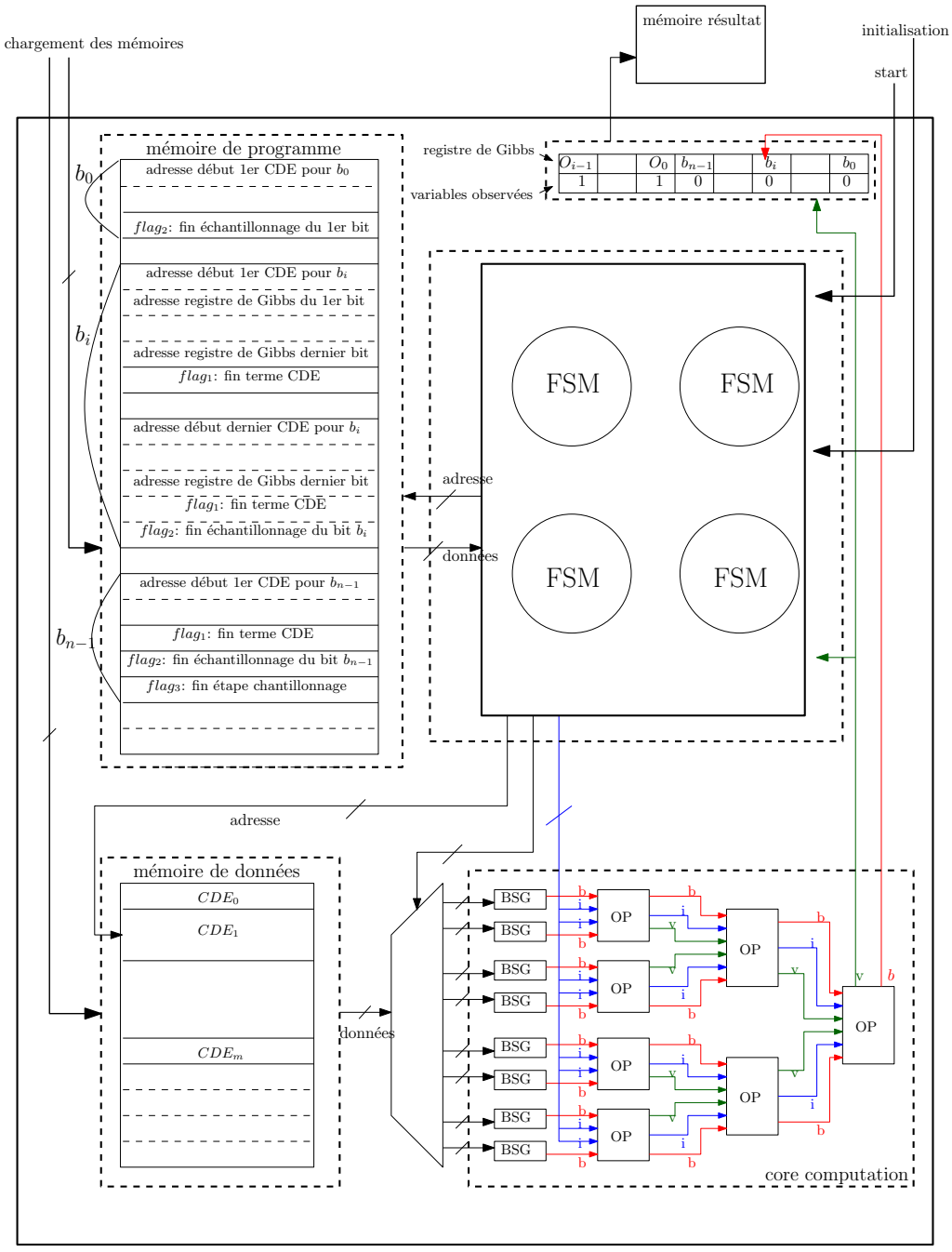



FIGURE 7.2 – La *PSMABI* à plusieurs machines à états

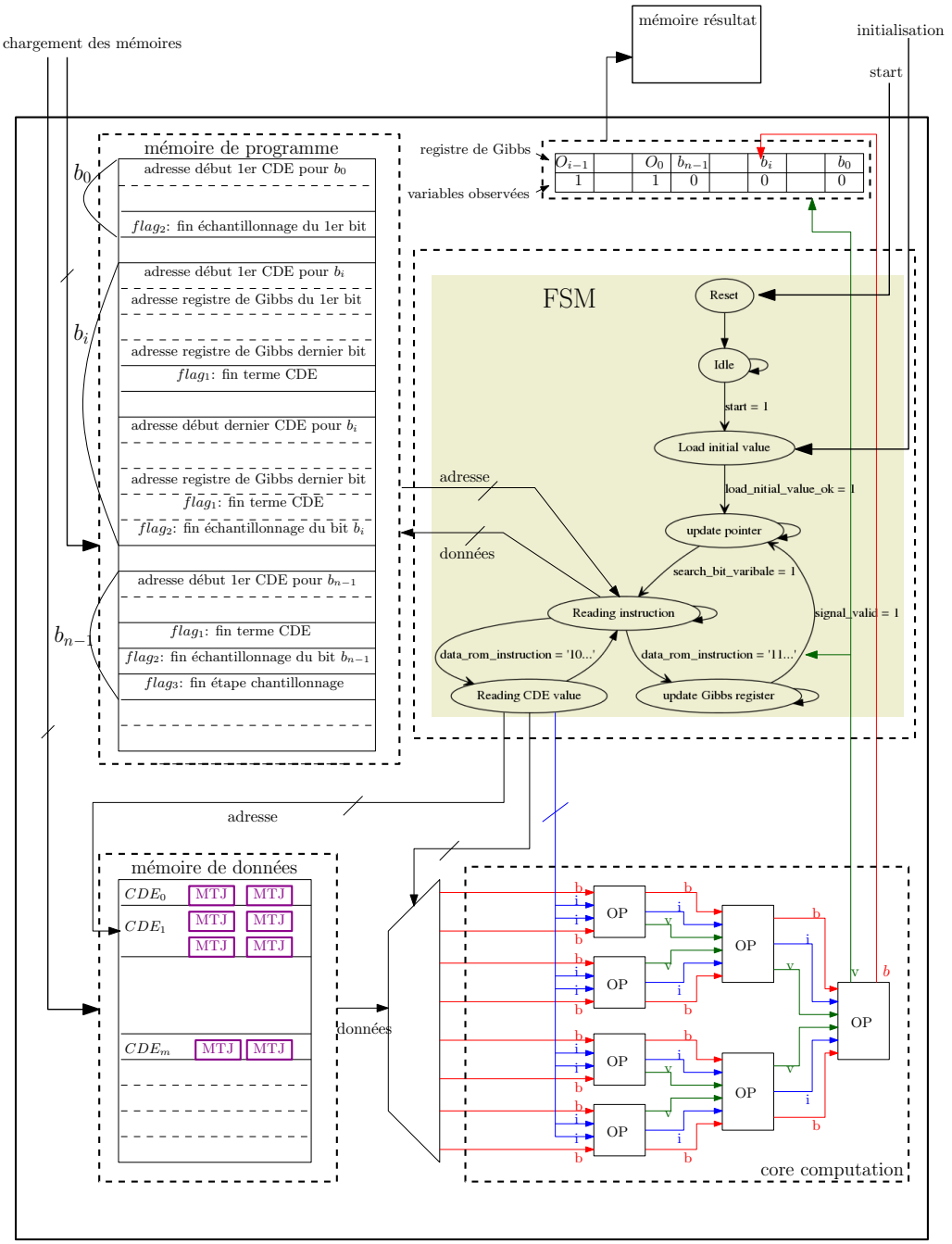



FIGURE 7.3 – L'aléa par les MTJs

### 7.2.5 Les projets futurs

Les voies améliorations proposées dans les sections précédentes sont directement liées à l'architecture des machines de type *SMABI*. Cependant, nous pensons que la méthode choisie pour se déplacer dans l'espace des possibles, c'est à dire l'échantillonnage des variables binarisées, est également une piste intéressante à approfondir. En effet, des études mathématiques sur l'exploration des espaces à grandes dimensions pourraient permettre à la machine d'avoir une façon d'explorer l'espace de la distribution de façon « plus intelligente ». En ce sens, le projet *MicroBayes*, à travers le LabEx PERSYVAL [1], propose de poursuivre les investigations sur les machines probabilistes développées dans cette thèse. Dans ce projet, nous nous intéressons à deux applications qui sont la localisation de sources sonores, voir figure 7.4, et la séparation de sources sonores, voir figure 7.5. Le premier problème peut se résoudre avec une *SMTBI* car c'est un problème de fusion capteur. Le deuxième est plus difficile et une machine de type *SMABI* doit être adaptée pour parcourir intelligemment un espace de recherche très grand.

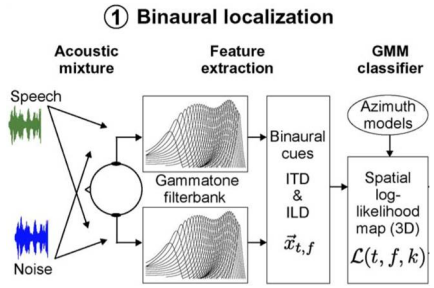



FIGURE 7.4 – Localisation de sources sonores

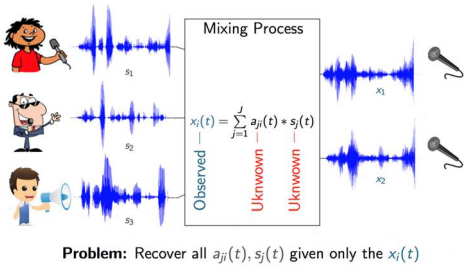



FIGURE 7.5 – Séparation de sources sonores

1. `https://persyval-lab.org/`

## 7.3 Mais que se cache-t-il sous le tapis ?

Il est temps de prendre du recul sur les travaux présentés dans ce manuscrit. Nous avons déjà évoqué les difficultés des machines *SMTBIs* à résoudre des problèmes d'inférence de grandes dimensions. Cette difficulté est due au fait que ce type de problème appartient à la catégorie des problèmes NP-complets. De la même manière, les machines *SMABIs* sont contraintes par la méthode de résolution qu'elles utilisent. L'algorithme de Gibbs n'offre pas l'assurance absolue d'avoir une distribution approchée qui nous siée. De plus, l'utilisation du cas binaire, qui est une méthode de recherche possible dans l'espace des valeurs de la conjointe, peut s'avérer également contraignante. En effet, les sauts effectués dans l'espace par l'échantillonnage d'un bit, peuvent gêner, voire empêcher la convergence vers la distribution souhaitée. On peut imaginer l'implémentation d'autres méthodes d'échantillonnage et ainsi concevoir d'autres types de machine utilisant le calcul stochastique.

Il est important de noter que pour le moment, les machines conçues ne s'interfacent pas aisément avec d'autres types de modules de calcul, en particulier, avec d'autres machines du même type. Cependant, cette généricité est une étape facilement envisageable par la réutilisation des échantillons tirés d'autres machines.

Enfin, vous aurez remarqué que l'idée originelle de se soustraire aux architectures de calculateurs conventionnelles a été égratignée au fur et à mesure de ce manuscrit. En effet, les machines *SMTBI* sont très différentes des calculateurs existants mais la *PSMABI* ressemble à un petit processeur probabiliste dédié. L'utilisation du calcul stochastique et des flux binaires pour représenter les données nous permet d'imaginer le passage à des signaux télégraphiques, émanant de sources comme les MTJs par exemple, que l'on voit comme une représentation bio-inspirée de l'information. Nous devons alors nous demander s'il faut coupler ces signaux analogiques et les intégrer dans une technologie CMOS maîtrisée ou bien continuer à pousser notre imagination pour aller vers des calculateurs totalement différents.

Et alors, peut être, un jour, une machine aura la même agilité de calcul que *Franklin*.

# Bibliographie


[1] Houillon A, Bessière P, and Droulez J. The probabilistic cell : Implementation of a probabilistic inference by the biochemical mechanisms of phototransduction. *Acta Biotheoretica 58*, pages 103–120, 2010.

[2] Topol A. W., La Tulipe J. D. C., Shi L., Frank D. J., Bernstein K., Steen S. E., Kumar A., Singco G. U., Young A. M., Guarini K. W., and Ieong M. Three-dimensional integrated circuits. *IBM J. Res. Develop.*, Vol. 50 :491–506, 2006.

[3] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha. Truenorth : Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10) :1537–1557, 2015.

[4] Armin Alaghi and John P Hayes. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12(2s) :92, 2013.

[5] Vigoda B. *Analog Logic : Continuous-Time Analog Circuits for Statistical Signal Processing.* PhD Dissertation, 2003.

[6] Pierre Bessière, Christian Laugier, and Roland Siegwart. Probabilistic reasoning and decision making in sensory-motor systems. *Springer Science & Business Media*, 46, 2008.

[7] Pierre Bessière, Emmanuel Mazer, Kamel Mekhnacha, and Juan Manuel Ahuactzin. In *Bayesian Programming.* Chapman & Hall/CRC Press, 2013.

[8] Christopher M Bishop. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[9] Lakshmi N. Chakrapani, Bilge E. S. Akgul, Suresh Cheemalavagu, Pinar Korkmaz, Krishna V. Palem, and Balasubramanian Seshasayee.

Ultra efficient embedded soc architectures based on probabilistic cmos (pcmos) technology. *Design Automation and Test in Europe Conference*, 2006.

[10] Karim Cherkaoui, Viktor Fischer, Alain Aubert, and Laurent Fesquet. A very high speed true random number generator with entropy assessment. *Workshop on Cryptographic Hardware and Embedded Systems-CHES*, pages 179–196, 2013.

[11] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2) :393 – 405, 1990.

[12] Jonas E. Stochastic architectures for probabilistic computation. *MIT, PhD Dissertation*, 2014.

[13] Steven K Esser, Alexander Andreopoulos, Rathinakumar Appuswamy, Pallab Datta, Davis Barch, Arnon Amir, John Arthur, Andrew Cassidy, Myron Flickner, Paul Merolla, et al. Cognitive computing systems : Algorithms and applications for networks of neurosynaptic cores. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–10. IEEE, 2013.

[14] M. Faix, R. Laurent, P. Bessière, E. Mazer, and J. Droulez. Design of stochastic machines dedicated to approximate bayesian inferences. *IEEE Transactions on Emerging Topics in Computing*, PP(99), 2016.

[15] M. Faix, E. Mazer, R. Laurent, M. Othman Abdallah, R. Le Hy, and J. Lobo. Cognitive computation : A bayesian machine case study. In *Cognitive Informatics Cognitive Computing (ICCI*CC), 2015 IEEE 14th International Conference*, pages 67–75, July 2015.

[16] Joào Filipe Ferreira and Jorge Dias. Probabilistic approaches to robotic perception. *Springer*, 2005.

[17] Joseph Friedman, Lucie Calvet, Pierre Bessière, Jacques Droulez, and Damien Querlioz. Bayesian inference with Muller C-Elements. *IEEE Transactions on Circuits and Systems I*, 2016.

[18] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 62(12) :2454–2467, 2013.

[19] BR Gaines. Stochastic computing systems. In *Advances in information systems science*, volume 2, pages 37–172. Springer, 1969.

[20] Solomon W. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, CA, USA, 1981.

[21] Noah Goodman, Vikash Mansinghka, Daniel Roy, Keith Bonawitz, and Daniel Tarlow. Church : a language for generative models. *arXiv preprint arXiv :1206.3255*, 2012.

[22] P. K. Gupta and R. Kumaresan. Binary multiplication with pn sequences. *IEEE Trans. Acoustics Speech Signal Process*, 36 :603–606, 1988.

[23] Pearl J. Morgan Kaufmann., 1988.

[24] Jaynes. *Probability Theory : the Logic of Science*. Cambridge University Press., 2003.

[25] P. L'Ecuyer. Uniform random number generation. *Annals of Operations Research*, 53 :77–120, 1994.

[26] N. Locatelli, A. F. Vincent, A. Mizrahi, J. S. Friedman, D. Vodenicarevic, J. V. Kim, J. O. Klein, W. Zhao, J. Grollier, and D. Querlioz. Spintronic devices as key elements for energy-efficient neuroinspired architectures. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 994–999, 2015.

[27] George Marsaglia and Liang-Huei Tsay. Matrices and the structure of random number sequences. *Linear Algebra and its Applications*, 67 :147 – 156, 1985.

[28] Alain J. Martin. 25 years ago : The first asynchronous microprocessor. *Caltech Technical Report*, 2014.

[29] Makoto Matsumoto and Takuji Nishimura. Mersenne twister : a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1) :3–30, 1998.

[30] Kamel Mekhnacha, Juan-Manuel Ahuactzin, Pierre Bessière, Emanuel Mazer, and Linda Smail. Exact and approximate inference in probt. *Revue d'Intelligence Artificielle*, pages Volume 31/3, 295–3342, 2007.

[31] M.Faix, J.Lobo, R.Laurent, D.Vaufreydaz, and E.Mazer. Stochastic bayesian computation for autonomous robot sensorimotor system. *IROS workshop*, 2015.

[32] B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov. *BLOG : Probabilistic models with unknown objects*. IJCAI., 2005.

[33] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. Cambridge, MA, 2012.

[34] Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies*, pages 46–98, Princeton Press, 1956.

[35] Avi Pfeffer. Practical probabilistic programming. In *Inductive Logic Programming*. Springer, 2011.

[36] Weikang Qian, Marc D. Riedel, Hongchao Zhou, and Jehoshua Bruck. Transforming probabilities with combinational logic. *IEEE transactions on computer aidded design of integrated circuits and systems, vol. 30, NO. 9*, 2011.

[37] R.Canillas, R.Laurent, Marvin Faix, D.Vaufreydaz, and E.Mazer. Autonomous robot controller using bitwise gibbs sampling. In *Cognitive Informatics Cognitive Computing (ICCI*CC), 2016 IEEE 15th International Conference*, 2016.

[38] Michael D. Richard and Richard P. Lippmann. Neural network classifiers estimate bayesian a posteriori probabilities. *Neural Computation,*, Vol. 3 :461–483, 1991.

[39] Kenneth L Shepard and Vinod Narayanan. Conquering noise in deep-submicron digital ics. *IEEE Design & Test of Computers*, 15(1) :51–62, 1998.

[40] Gordon M. Shepherd, Jason S. Mirsky, Matthew D. Healy, Michael S. Singer, Emmanouil Skoufos, Michael S. Hines, Prakash M. Nadkarni, and Perry L. Miller. The human brain project : neuroinformatics tools for integrating, searching and modeling multidisciplinary neuroscience data. *Trends in Neurosciences*, 21(11) :460 – 468, 1998.

[41] Marvin K. Simon, Jim K Omura, Robert A. Scholtz, and Barry K. Levitk. *Spread Spectrum Communications Handbook*. McGraw-Hill, 1994.

[42] Donald F. Specht. Probabilistic neural networks. *Neural Networks*, Vol. 3, :109–118, 1990.

[43] Ashok Srinivasan, Michael Mascagni, and David Ceperley. Testing parallel random number generators. *Parallel Comput.*, 29(1) :69–94, 2003.

[44] Robert C. Tausworthe. Random numbers recurrence generated by linear modulo two. *Mathematics and Computation 19*, pages 201–209, 1965.

[45] Saeed Sharifi Tehrani, Shie Mannor, and Warren J Gross. Survey of stochastic computation on factor graphs. In *Multiple-Valued Logic, 2007. ISMVL 2007. 37th International Symposium on*, pages 54–54. IEEE, 2007.

[46] Joshua B. Tenenbaum, Kemp C., T. L. Griffiths, and N. D. Goodman. How to grow a mind : Statistics, structure, and abstraction. *Science 331*, 2011.

[47] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[48] Mansinghka V. Natively probabilistic computation. *MIT, PhD Dissertation*, 2009.

## Conception de machines probabilistes dédiées aux inférences bayésiennes

**Résumé -** Ces travaux de recherche ont pour but de concevoir des ordinateurs basés sur une organisation du calcul mieux adaptée au raisonnement probabiliste. Pour cela, nous proposons des architectures de machines se soustrayant au modèle Von Neumann, supprimant notamment l'utilisation de l'arithmétique en virgule fixe ou flottante.
Plus spécifiquement, ces travaux décrivent deux types de machines probabilistes, radicalement différentes dans leur conception, dédiées aux problèmes d'inférences bayésiennes et utilisant le calcul stochastique. La première traite les problèmes d'inférence de faibles dimensions. Cette machine est basée sur le concept de bus probabiliste et possède un très fort parallélisme. La deuxième machine permet de traiter les problèmes d'inférence en grandes dimensions. Elle implémente une méthode MCMC sous la forme d'un algorithme de Gibbs au niveau binaire. Une importante caractéristique de cette machine est de contourner les problèmes de convergence généralement attribués au calcul stochastique. Nous présentons en fin de manuscrit une machine générique et programmable permettant de trouver une solution approchée à n'importe quel problème d'inférence.

**Mots Clés :** inférence, calcul stochastique, Monte-Carlo, probabilité, machine bayésienne

## Design of stochastic machines dedicated to bayesian inferences

**Abstract -** The aim of this research is to design computers best suited to do probabilistic reasoning. For this, new machine architectures are presented. The concept they are built on is different to the one proposed by Von Neumann, without any fixed or floating point arithmetic.
In this thesis, two types of probabilistic machines are presented. Their designs are radically different, but both are dedicated to Bayesian inferences and use stochastic computing. The first deals with small-dimension inference problems and uses stochastic computing to perform the necessary operations to calculate the inference. The second machine can deal with intractable inference problems. It implements a particular MCMC method : the Gibbs algorithm at the binary level. An important feature of this machine is the ability to circumvent the convergence problems generally attributed to stochastic computing. Finally, we present a generic and programmable machine designed to approximate solution to any inference problem.

**Keywords :** inference, stochastic computing, Monte-Carlo, probability, bayesian machine

Thèse préparée aux laboratoires LIG (Laboratoire informatique de Grenoble), 655 avenue de l'Europe, 38330 Montbonnot et TIMA (Techniques de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs), 46 Avenue Félix Viallet, 38031, Grenoble Cedex, France
**ISBN : 978-2-11-129223-9**