Conception, modélisation et simulation in silico d'un nanosystème biologique artificiel pour le diagnostic médical

Marc Bouffard

NNT : 2016SACLS302

Thèse de doctorat de l'Université Paris-Saclay préparée à l'Université Paris-Sud

Ecole doctorale n°580
Sciences et Technologies de l'Information et de la Communication
Spécialité de doctorat : Informatique

par

M. Marc Bouffard

Conception, modélisation et simulation *in silico* d'un nanosystème biologique artificiel pour le diagnostic médical

Thèse présentée et soutenue au L.R.I., Bât. 650 "Ada Lovelace", rue Noetzlin, 91190 Gif sur Yvette, le 29 septembre 2016.

Composition du Jury :

M.	Philippe Dague	Professeur Université Paris Sud	(Président)
M.	Gilles Bernot	Professeur Université Nice Sophia Antipolis	(Rapporteur)
M.	Jean-Pierre Mazat	Professeur Université de Bordeaux	(Rapporteur)
M.	Victor Norris	Professeur Université de Rouen	(Examinateur)
M.	Franck Molina	DR CNRS CNRS Sys2Diag, Montpellier	(Co-directeur de thèse)
M.	Patrick Amar	Maître de Conférences HDR Université Paris Sud	(Co-directeur de thèse)

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Conception, modélisation et simulation *in silico* d'un nanosystème biologique artificiel pour le diagnostic médical

Mots clefs : Diagnostic médical, Biologie synthétique, Réseaux métaboliques, Portes logiques

Résumé : Le diagnostic médical, se fait traditionnellement, par l'examen des symptomes cliniques, puis en cherchant sur des prélèvements (sang, urine, biopsies, etc.) la présence (ou l'absence) simultanée des bio-marqueurs des diverses pathologies envisagées par le médecin. La recherche des bio-marqueurs se fait a l'aide d'équipements importants, dans un laboratoire d'analyse; les résultats étant communiqués au médecin, qui va les interpréter en appliquant un algorithme de diagnostic médical.

Nous avons voulu regrouper dans un seul dispositif, pour une pathologie donnée, la détection des bio-marqueurs et une implémentation de l'algorithme de diagnostic approprié. La présence ou l'absence d'un bio-marqueur peut être représentée par une variable booléenne, et l'algorithme de diagnostic par une fonction booléenne complexe dont la valeur indiquera la présence de la pathologie ciblée.

Notre dispositif de diagnostic sera un nano-calculateur biochimique artificiel dans lequel les informations logiques seront représentées par des métabolites et les calculs effectués par un réseau enzymatique synthétique. Pour réaliser ce calculateur, il a été nécessaire d'établir un fondement théorique des réseaux logiques enzymatiques. Nous avons ensuite utilisé cette théorie pour définir ce qu'est un circuit logique enzymatique et comment il calcule correctement la fonction booléenne associée.

Pour des raisons de modularité et de réutilisabilité, nous avons décidé de concevoir des bibliothèques de portes logiques enzymatiques implémentant les opérateurs booléens de base, puis d'assembler ces briques de base pour obtenir le réseau enzymatique complet. J'ai donc conçu et développé deux outils logiciels, NetGate et NetBuild, qui vont réaliser automatiquement ces opérations.

NetGate, qui va créer des bibliothèques contenant des centaines de portes logiques enzymatiques obtenues à partir de réseaux métaboliques d'organismes existants. Auparavant, il était nécessaire d'analyser manuellement ces réseaux métaboliques pour extraire chaque porte.

NetBuild, qui va utiliser une bibliothèque de portes (par exemple créée par NetGate) et les assembler pour construire des circuits qui calculent une fonction booléenne donnée. Ces circuits utilisent comme entrées des métabolites spécifiques (par exemple les bio-marqueurs d'une pathologie) et produisent en sortie une espèce moléculaire facilement détectable (par colorimétrie par exemple).

Title : Design, modelling and *in silico* simulation of a synthetic biological nanosystem for medical diagnosis

Keywords : Medical Diagnosis, Synthetic biology, Metabolic networks, Logic gates

Abstract : The medical diagnosis is traditionally done by examining the clinical symptoms and by searching in samples (blood, urine, biopsies, etc.) for the simultaneous presence (or absence) of biomarkers of the various pathologies considered by the doctor. The search for biomarkers is conducted using large equipments in a specialised laboratory; The results being communicated to the doctor, who will then interpret them by applying a medical diagnosis algorithm.
We wanted to combine in a single device, for a given disease, the detection of its biomarkers and an implementation of the appropriate diagnosis algorithm. The presence or absence of a biomarker can be represented by a boolean variable, and the diagnosis algorithm by a complex boolean function whose value indicates the presence of the targeted disease.
Our diagnosis device is an artificial biochemical nano-computer where logical information is represented by metabolites and the computations are performed by a synthetic enzymatic network. To build this computer, it has been necessary to establish a theoretical basis of enzymatic logical networks. We then used this theory to define what an enzymatic logic network is, and how it computes correctly the associated boolean function.
For modularity and reusability reasons, we decided to design libraries of enzymatic logic gates that implement basic boolean operators, and then to assemble these building blocks to get the complete logic enzymatic network. So, I have designed and developed two software tools, NetGate and NetBuild, which will automatically perform these operations.
NetGate creates libraries containing hundreds of enzymatic logic gates obtained from the metabolic networks of living organisms. Before that, it was necessary to manually analyse these metabolic networks in order to extract each logic gate.
NetBuild uses a library of logic gates (for example created using NetGate) and assembles them to build circuits that compute a given boolean function. These circuits use specific metabolites for its inputs (for example the biomarkers of a pathology) and produce a readily detectable molecular species (using colorimetry for example).

Remerciements

Je tiens à remercier toutes les personnes qui m'ont entourées et soutenues pendant ces trois années de thèse.

Je remercie tout particulièrement mes directeurs de thèse, Patrick Amar pour toutes les choses qu'il m'a apprises, pour nos discussions acharnées et passionnées sur la biologie et l'informatique ainsi que Franck Molina pour ses conseil avisés et son accueil à Sys2Diag Montpellier.

Je suis aussi reconnaissant à tous les membres de l'équipe Bioinfo du LRI pour leur présence qui a contribuée à faire de cette équipe un environnement de travail particulièrement agréable ainsi que l'ensemble du Laboratoire de Recherche en Informatique pour les moyens et l'efficacité des outils mis à notre dispositions.

Je remercie également Adrien Rougny qui a passé sa thèse quelques jours après moi pour les discussions passionnantes sur les façons de refaire le monde.

Je remercie les chercheurs du LRI du 1er étage du bâtiment *Ada Lovelace* qui nous ont aidés Adrien et moi à finir toutes ces grilles de mots croisés, notamment Philippe Dague qui a aussi accepté de faire partie de mon jury.

Je remercie bien sûr ma famille, qui a bien voulu relire mon manuscrit malgré les fautes d'orthographe, et de m'avoir soutenu tout au long de ma thèse (même s'ils ont pensé presque jusqu'à la fin que je devrais finir par commencer à chercher vrai métier...)

Table des matières

Chapitre 1

Introduction

Les sciences informatiques sont une des grandes découvertes du XX^{e} siècle, et nos capacités de traitement de l'information se sont considérablement accrues depuis que nous pouvons construire des systèmes artificiels capables de faire du calcul. Ils nous ont permis de nombreux progrès, dans un grand nombre de disciplines comme la physique, les mathématiques et notamment la biologie et la médecine.

La médecine est un domaine très ancienne, qui a subi de profonds changements au cours du temps. Guérisseurs et chamanes de la préhistoire essayant d'aider leur tribu ; code d'Hammurabi au XVIII^{e} siècle av. J.-C définissant les obligations des soigneurs ; savants de l'antiquité grecque à la fois médecins, physiciens et mathématiciens ; premiers chirurgiens sous la Rome antique ; début de la systématisation de l'expérimentation et de la quantification par les médecins arabes ; institutionnalisation de l'anatomie en Occident au XVII^{e} siècle ; portée par les progrès de la chimie, apparition de la bactériologie et de la virologie au XIX^{e} siècle ; au XX^{e} siècle, plusieurs grands jalons ont été atteints et même dépassés dans l'analyse des systèmes biologiques. Cela a permis d'établir des modèles reflétant de mieux en mieux la réalité infiniment complexe de la biologie. Nous pouvons maintenant simuler des phénomènes sur lesquels nos prédécesseurs étaient limités à faire des hypothèses. De ces recherches, sont nées quantités de nouveaux traitements médicaux, d'angles d'approche novateurs pour l'étude de pathologies, ou encore de méthodes innovantes pour aider les professionnels de santé. La médecine est maintenant largement assistée par les systèmes informatiques. Mais le médecin reste toujours au cœur du sujet, l'informatique n'étant qu'un outil pour acquérir des informations mais pas pour les interpréter.

Ces dernières années, nous avons pu voir plusieurs tentatives pour construire des systèmes d'information capables d'effectuer du traitement intelligent de l'information dans un but médical et des systèmes capables de diagnostiquer des pathologies. Ce mariage entre la biologie et l'informatique reste encore fragile, leur rapprochement est limité par leur non-interaction directe. C'est le médecin qui va faire le lien entre la biologie (du patient) et le système informatique. Ces deux domaines sont complètement hétérogènes et il reste toujours un obstacle important à franchir pour l'utilisation de systèmes informatiques intégrés à des applications biologiques. L'interface entre ces deux types de systèmes en est pour l'instant à ses débuts. Les systèmes informatiques électroniques sont basés sur des jeux complexes entre des impulsions électriques, alors que les systèmes biologiques sont par nature des flux de molécules en interaction. C'est cette différence de nature qui rend difficile la construction d'une interface.

Notre idée principale n'est pas de chercher à interfacer directement les systèmes informatiques et les systèmes biologiques, mais de transformer l'un des systèmes (ici le système informatique) pour qu'il soit plus naturellement adapté à interagir avec l'autre. Les systèmes informatiques sont une implémentation de systèmes logiques booléens par des composants matériels (circuits à semi-conducteurs à base de

silicium) et par des impulsions électriques qui représentent les flux d'informations dans le système. Pour répondre à notre problématique, nous avons décidé d'implémenter des systèmes logiques avec des composants biologiques, plus spécifiquement à partir de réseaux métaboliques et de représenter les informations par des métabolites. Cette solution permet de passer outre les problèmes d'interfaces. Il a néanmoins été nécessaire de d'imaginer un tout nouveau cadre théorique pour créer et comprendre ces nouveaux réseaux logiques enzymatiques.

Notre but est de concevoir un système qui prendra en entrée une fonction logique (*algorithme de diagnostic*), ses arguments (biomarqueurs de la pathologie ciblée) et en sortie, une espèce moléculaire facilement détectable, puis qui proposera en réponse plusieurs réseaux logiques enzymatiques implémentant cette fonction logique (voir fig. 1.1).

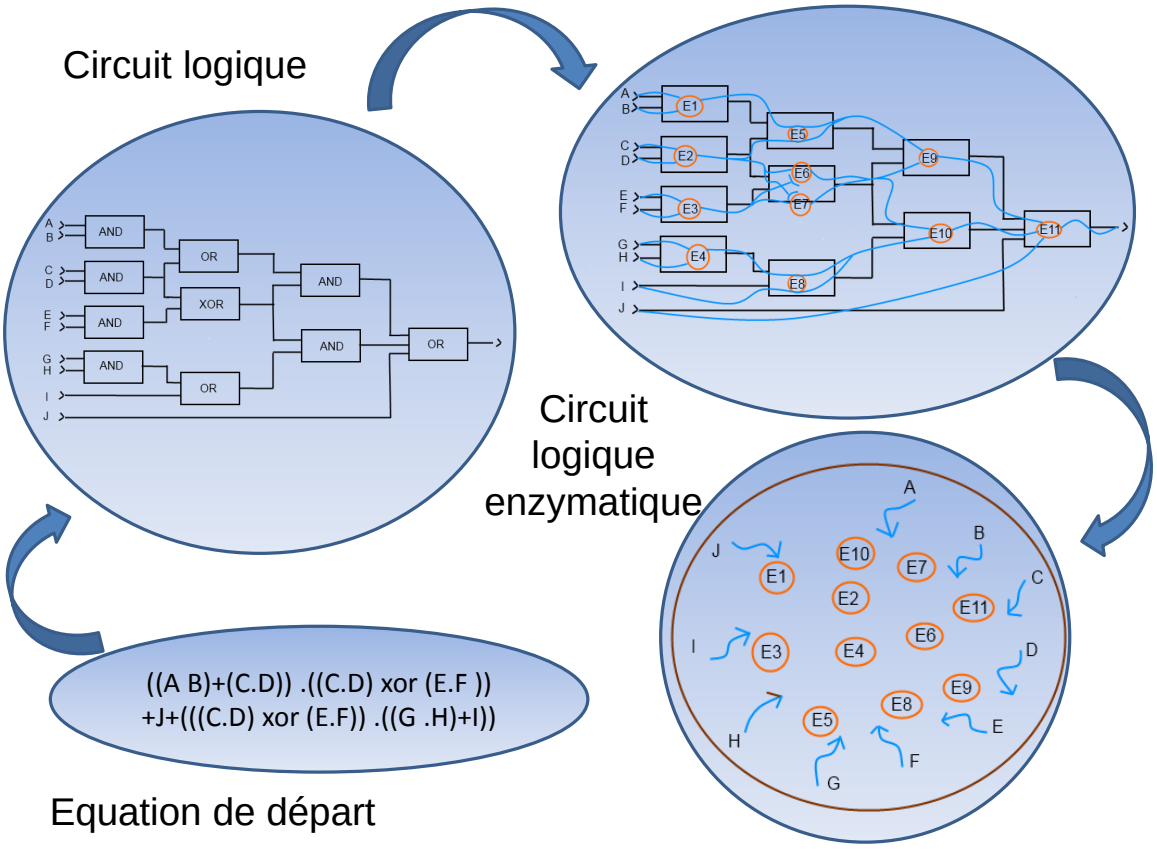


Figure 1.1 – Schéma représentant l'enchaînement des algorithmes développés dans cette thèse. A partir d'une fonction logique, nous allons implémenter un circuit logique enzymatique avec des composants biochimiques.

Pour répondre à cette problématique, nous avons conçu et implémenté deux algorithmes :

— *NetGate* : qui va rechercher à l'intérieur de réseaux métaboliques déjà connus, de petites unités logiques pouvant implémenter des portes logiques enzymatiques.

— *NetBuild* : qui va prendre la fonction logique donnée en paramètre et va utiliser les portes logiques enzymatiques trouvées par *NetGate* pour construire des circuits logiques enzymatiques répondant à cette fonction.

Nous avons aussi développé un troisième algorithme, *NetPlace*, qui place graphiquement des réseaux logiques enzymatiques de façon visible et facilement interprétable par un humain (*pretty printer*).

1.1 Détection de bio-marqueurs pour le diagnostic médical

1.1.1 Le diagnostic médical assisté

Le processus de diagnostic est considéré comme un art [1], une tâche complexe en raison, entre autres, des incertitudes présentes dans ce processus. La relation entre une pathologie et ses symptômes n'est pas toujours une correspondance biunivoque, différentes pathologies peuvent partager un ou plusieurs symptômes, et les observations sont sujettes à des erreurs et peuvent être insuffisantes pour avoir un diagnostic précis. Les professionnels formés à ces actions font appel à leur savoir appris et à leur expérience pour réussir à combiner toutes les informations à leur disposition dans le but d'arriver à un diagnostic fiable. Durant ce processus, assimilable d'un point de vue logique à un algorithme [2](voir fig. 1.3 pour un exemple d'une partie d'algorithme utilisé pour le diagnostic de l'arthrose de la main), le professionnel va comparer toutes les pathologies qu'il connait et leurs symptômes associés, à celle du patient pour chercher des similitudes. Il va sélectionner des pathologies candidates, compatibles avec les symptômes, effectuer des tests complémentaires pour affiner sa sélection, jusqu'à obtenir *a priori* le bon diagnostic.

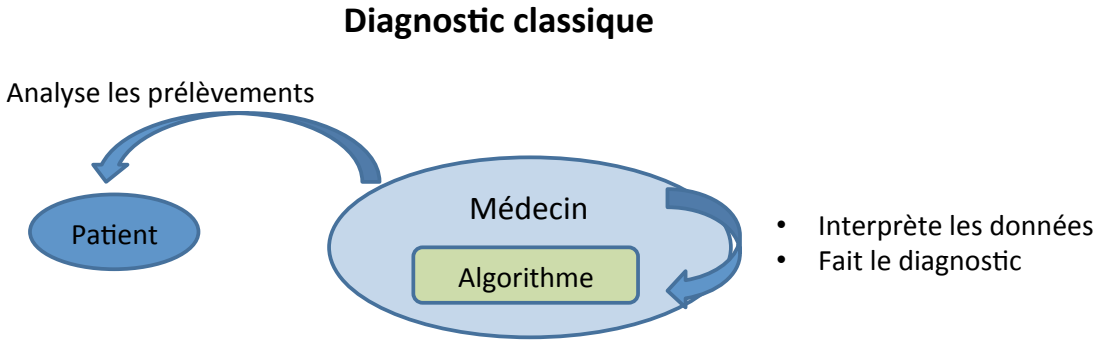


Figure 1.2 – Méthode classique (la plus ancienne) de diagnostic, il est effectué uniquement par le médecin. Il se base sur ses connaissances pour sélectionner ou éliminer des pathologies potentielles. Il effectue toutes les opérations : l'analyse des données et leur traitement.

Ce processus de sélection utilisé par les médecins peut être reproduit artificiellement par des systèmes informatiques. Les connaissances acquises par expérience et par apprentissage peuvent être recensées et utilisées pour créer des systèmes permettant d'obtenir ou d'affiner des diagnostics.

Toha et al. ont démontré que le diagnostic médical automatisé pouvait être généralisé sur plusieurs pathologies [3]. Les auteurs ont conçu un algorithme, qui en fonction des syndromes observés sur les patients, va établir des probabilités de pathologie. Ils sont ainsi capables de reconnaître quinze *profils* différents. Ces diagnostics sont potentiels, ils ont une probabilité quantifiée d'être corrects, tous comme le sont les diagnostics effectués par des médecins humains. Néanmoins le degré de confiance que l'on accorde au système expert et au médecin n'est pas le même, c'est une des limites inhérentes à ces logiciels, mais qui ne peut être réglée que par la banalisation de ces systèmes. De plus, pour obtenir ces résultats, il est nécessaire de construire des tables exhaustives des symptômes possibles avec leurs probabilités d'apparition, et cela pour chaque pathologie, ce qui est fastidieux voire impossible.

Ces systèmes sont encore loin de pouvoir être utilisés directement dans la vie courante, mais plusieurs essais ont été faits pour construire des systèmes directement adaptés à un type d'utilisateur particulier. *Moraes Lopes et al.* ont montré qu'il était possible d'utiliser un processus de logique importé de l'informatique théorique, la logique floue, dans le cadre du diagnostic de l'origine de l'incontinence urinaire [4]. Ici, le développement d'un système automatisé capable d'effectuer un diagnostic permet de pallier le manque de personnels (gynécologues) formés spécifiquement au diagnostic de l'origine

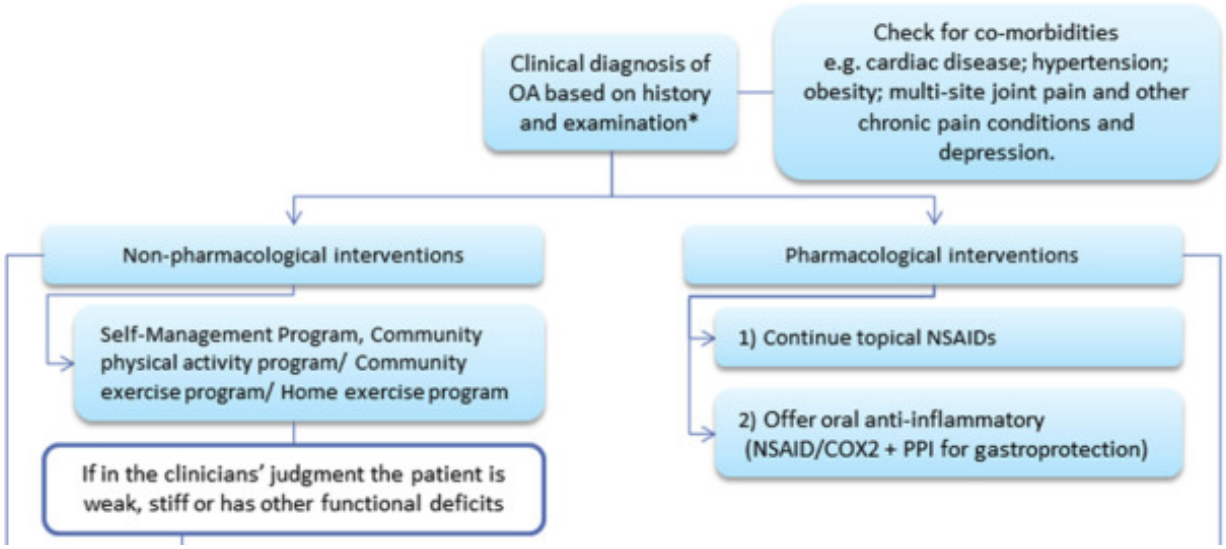


Figure 1.3 – Exemple d'une partie d'algorithme utilisé pour le diagnostic de l'arthrose de la main. On peut voir qu'il a la forme d'un diagramme état-transition, il y a des actions : effectuer des tests, des prélèvements, etc. Puis il y a des choix : y a-t-il eu des traitements pharmacologiques ? N'est présentée ici qu'une petite partie du diagramme qui est bien plus complexe, lui-même faisant partie d'une série de diagrammes qui diffèrent selon les cas.

de l'incontinence urinaire. Les principaux intervenants, médecins non gynécologues et infirmiers, ne peuvent que traiter de façon approximative une pathologie dont ils ne peuvent établir la nature. Ce système est spécifiquement conçu pour être utilisé par du personnel soignant non spécialisé. C'est un aspect important car l'un des défis de l'utilisation de moyens de diagnostic automatisé est leur interface utilisateur.

Les systèmes que nous venons de voir sont basés sur une utilisation *manuelle* de l'algorithme de diagnostic. Il est donc nécessaire qu'un spécialiste utilise correctement le système de diagnostic. Il est aussi nécessaire d'effectuer puis d'analyser les prélèvements effectués sur le patient. Ensuite, il faut convertir ces informations en données adaptées à l'algorithme. Enfin les résultats doivent être interprétés par le spécialiste.

1.1.2 Détection de bio-marqueurs

Quelle que soit la façon d'utiliser les algorithmes pour le diagnostic médical, il faut s'appuyer sur des observations faites sur le patient. Ces observations, peuvent être des symptômes observables, comme un trouble cognitif [5], des mesures faites par des appareils directement sur le patient, comme un électrocardiogramme [6], ou peuvent provenir de prélèvements (sang, urine, biopsie,..). En général c'est un mélange de plusieurs observations et de plusieurs types d'observations qui apporteront une réponse[7]. Ce sont des informations sur le fonctionnement des systèmes biologiques, elles permettent de tracer les processus internes que l'on ne saurait observer sinon, et de nous offrir une fenêtre sur les rouages du corps humain.

1.1.2.1 Bio-marqueurs

Le National Institutes of Health (NIH, qu'on peut traduire par Instituts américains de la santé) a défini en 2007 le terme de biomarqueur : *Une caractéristique qui est mesurée et évaluée de manière*

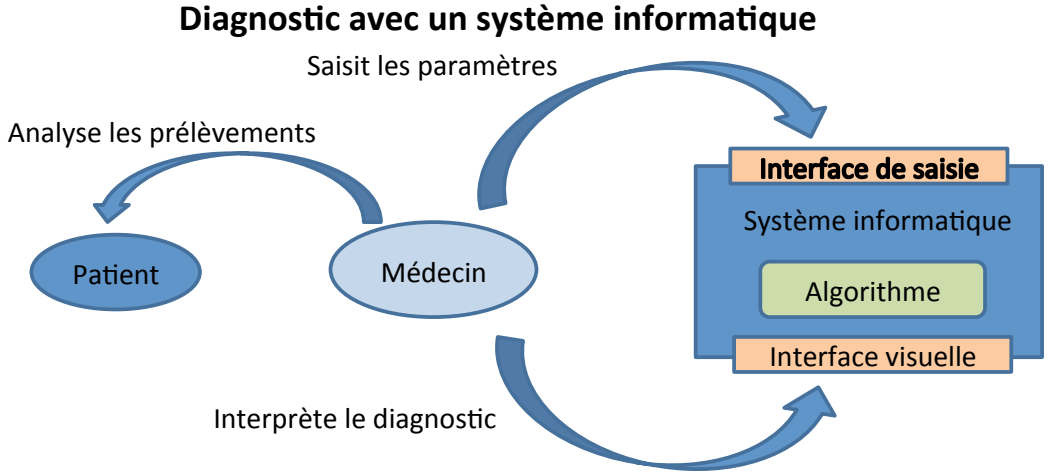


Figure 1.4 – Cette méthode est celle qui se popularise actuellement. Le médecin utilise le système informatique comme un outil qui va lui permettre de faciliter son processus de diagnostic. Il va fournir à l'outil les informations nécessaires relevées sur le patient, puis ensuite, il interprétera les résultats.

objective comme un indicateur de processus biologiques normaux, de processus pathogènes ou de réponses pharmacologiques à une intervention thérapeutique(traduit de l'anglais). De nombreuse études ont montré comment détecter des biomarqueurs, et les utiliser comme indicateurs de la présence ou de l'absence de pathologie. Ces biomarqueurs peuvent être des concentrations moléculaires, mais aussi des mesures externes comme des relevés IRM (Imagerie à résonance magnétique) [8]. Les biomarqueurs moléculaires existent dans un grand nombre de milieux propres au corps humain, par exemple le sang [9] ou les urines [10]. Ils peuvent être de différentes natures, ADN [11], ARN [12], protéine [13] et métabolite [14].

Ces biomarqueurs moléculaires sont en général représentés par des concentrations ($[C]$) de molécules, et ce sont leurs niveaux qui vont apporter des informations. Ils peuvent être utilisés directement, mais souvent, ils le sont sous une forme binaire, il va donc falloir définir des seuils. Ces seuils (S_1) peuvent être propres à chaque biomarqueur. Dans ce cas, si $[C] > S_1$, ces biomarqueurs sont considérés *positifs*, sinon ils sont considérés *négatifs*. Il peut aussi y avoir plusieurs niveaux d'activation pour un biomarqueur, il aura donc autant de seuils que de paliers.

Ces biomarqueurs sont des indicateurs qui vont nous permettre de savoir ce qui se passe précisément dans un système biologique. Ils donnent des indications sur l'état du système qui peuvent être de plusieurs types [15].

— Pronostic : le biomarqueur est un indicateur de l'avancement de la pathologie. Il donne des informations sur l'état de la maladie en mesurant les précurseurs internes qui augmentent ou diminuent la probabilité de développer une maladie.

— Prédictif : le biomarqueur mesure l'effet d'un médicament et permet de dire si le médicament a l'action attendue, mais il ne donne pas d'information directe sur la pathologie.

Pour qu'une espèce moléculaire soit un bon biomarqueur, il ne faut pas seulement qu'elle indique une information utile, il faut aussi qu'elle puisse être mesurée, si possible facilement. Il existe un grand nombre de méthodes pour détecter des biomarqueurs : nano-senseurs [16], spectrométrie de masse [17], 2-D western blot [18], 2-D électrophorèses en gel [19], méthode immuno-enzymatique ELISA [20], biopuces [21], chacune ayant ses avantages et ses inconvénients. Toutes ces méthodes fonctionnent selon le même schéma : il faut faire des prélèvements, les placer dans un appareil, puis interpréter les résultats, le plus souvent avec un ordinateur.

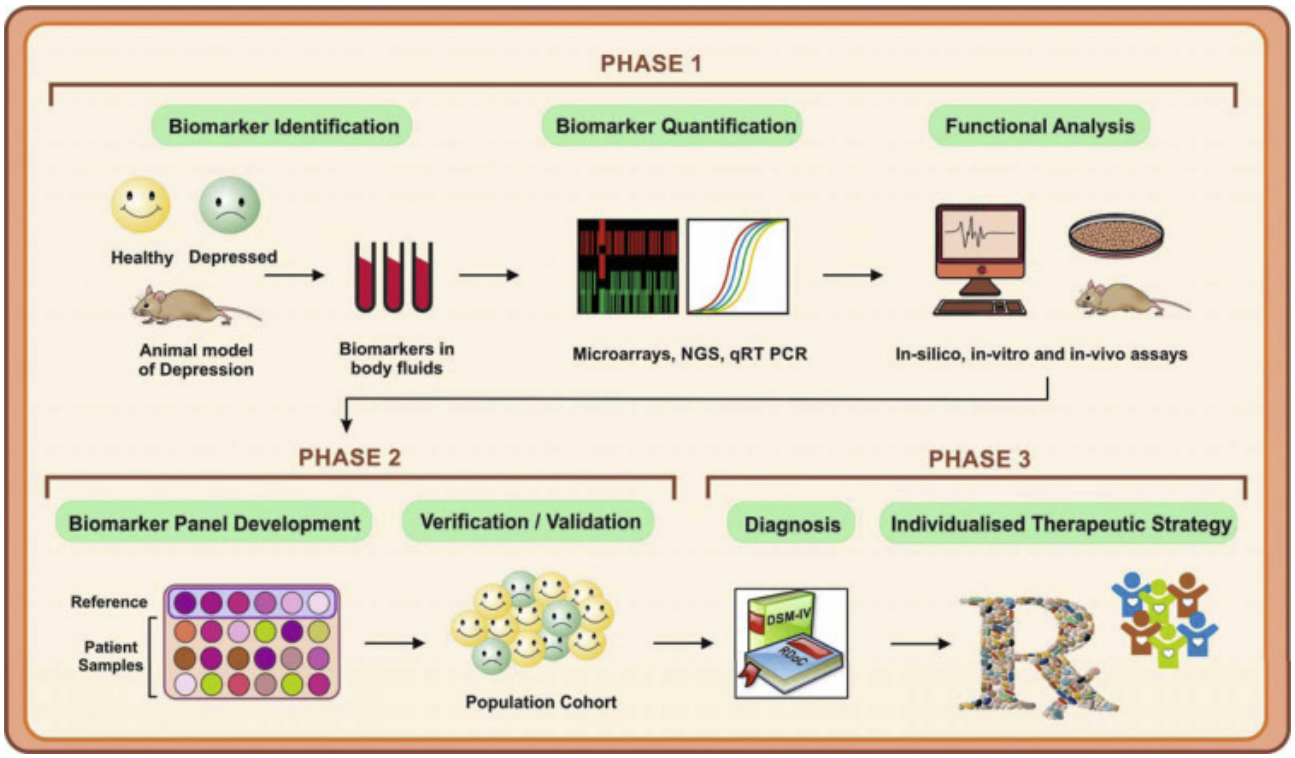


Figure 1.5 – Stratégie proposée pour l'identification et la validation de biomarqueurs. La phase 1 comprend l'identification et la quantification de biomarqueurs potentiels. Lors de la phase 2, une liste de biomarqueurs sélectionnés sera développée puis validée et vérifiée dans différentes populations. Dans la phase 3, l'efficacité de la liste de biomarqueurs sera testée pour déterminer dans quelle mesure il améliore la précision des systèmes de diagnostic actuels. Image de *Gururajan et al.* [22]

1.1.2.2 Une nouvelle interface

Nous savons que les résultats obtenus par les méthodes de détection sont des mesures de niveau de concentration d'espèces moléculaires. Il va falloir ensuite utiliser ces mesures pour poser un diagnostic. Mais il reste toujours le problème du transfert des informations contenues dans les biomarqueurs. En effet, un des principaux défauts de ces systèmes est l'interface entre la partie biologique et la partie informatique. L'information est à l'origine biologique, puisqu'elle provient du patient, mais le traitement de l'information est fait sur un système informatique. C'est cette frontière qui va nécessiter l'intervention de moyens tiers, personnel et/ou machine. Cette interface est source d'erreurs car il va falloir convertir les flux dinformations sous plusieurs formes à chaque étape, jusqu'à ce qu'ils arrivent au système de diagnostic. Il y a donc une forte probabilité de perdre des données. De plus, cela nécessite un investissement conséquent puisqu'il est nécessaire d'utiliser de nombreuses machines ainsi que du personnel spécialisé.

Nous souhaitons passer au-delà de cette frontière, et construire un système qui sera directement capable d'interfacer les informations issues des biomarqueurs avec les algorithmes informatiques qui vont effectuer le traitement de ces informations et donc proposer un diagnostic.

Notre solution est de supprimer la nécessité de cette interface. Les systèmes de diagnostics sont conçus pour fonctionner sur une implémentation matérielle électronique d'un système logique booléen (ordinateur standard). Nous allons concevoir une implémentation différente de systèmes logiques booléens, qui sera naturellement interfacée avec les biomarqueurs puisqu'elle sera de même nature (voir

fig. 1.6). Notre but est de créer un réseau biochimique qui sera capable de répondre à une fonction logique prédéfinie. Il suffira ensuite que ce réseau biochimique artificiel soit au contact des biomarqueurs pour que l'information soit lue et la fonction calculée.

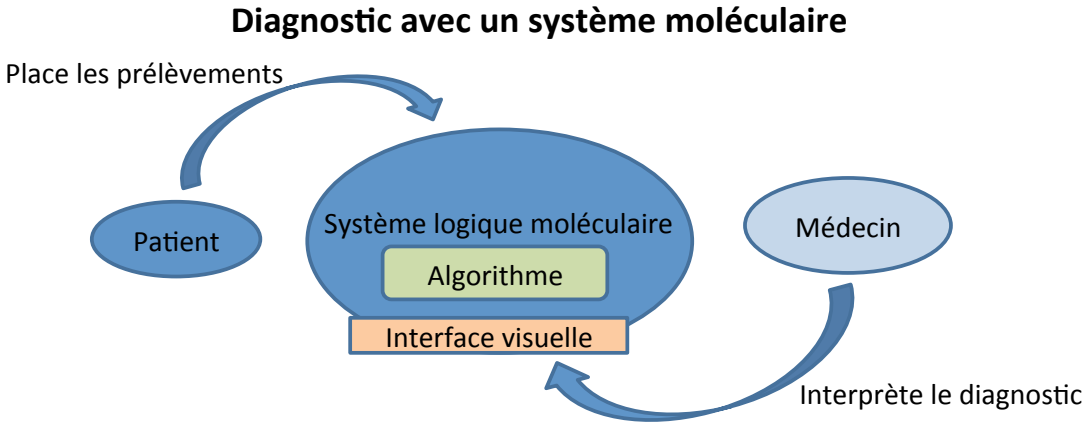


Figure 1.6 – Le système de diagnostic est un système biochimique qui s'interface directement avec les biomarqueurs. le médecin va donc simplement mettre les prélèvements du patient au contact du système, puis il interprétera les résultats pour formuler son diagnostic.

1.1.3 Réseaux logiques

Pour traiter les informations qu'apporteront les biomarqueurs, nous allons utiliser des réseaux logiques booléens. Ces réseaux sont basés sur l'algèbre de Boole, définie au début du XX^{e} siècle par plusieurs mathématiciens, Jevons, Schröder et Huntington [23]. L'algèbre de Boole permet de manipuler des informations binaires qui ont soit la valeur *vrai*, soit la valeur *faux*. C'est la base des circuits logiques, le concept sur lequel sont construits les processus de calcul de tous les systèmes informatiques. Ces circuits logiques sont des concepts abstraits permettant d'effectuer des opérations de calculs complexes. Ils sont composés de portes logiques qui sont les briques élémentaires composables pour construire des circuits logiques répondant aux fonctions logiques voulues.

1.1.3.1 Portes logiques

Une porte logique est un dispositif abstrait mettant en œuvre une fonction booléenne, il effectue une opération logique sur une ou plusieurs entrées logiques et produit une sortie logique unique. Les portes logiques sont principalement implémentées en électronique, avec des diodes et/ou des transistors agissant comme des interrupteurs électroniques. Elles ont été également construites dans le passé en utilisant des tubes à vide, des relais électromagnétiques, ou même des éléments mécaniques. On peut aussi utiliser des dispositifs optiques, ou biochimiques. Les portes logiques peuvent être enchainées de la même manière que les fonctions booléennes peuvent être composées. Cela permet la construction d'une représentation physique d'une équation de la logique booléenne et, par conséquent, l'utilisation de tous les algorithmes et concepts mathématiques pouvant être décrits par la logique booléenne.

1.1.4 Circuits électroniques

Les réseaux logiques sont majoritairement implémentés sous la forme de circuits électroniques. Les circuits électroniques modernes sont des assemblages de transistors, qui sont les éléments fondamentaux

de l'électronique. Un transistor est un dispositif semi-conducteur qui (dans ce cas d'utilisation binaire) va jouer le rôle d'un interrupteur. Ces circuits sont les composants de base utilisés pour construire tous les appareils électroniques qui nous entourent.

1.1.4.1 Transistors

Le transistor est un composant construit à partir d'un arrangement de matériaux semi-conducteurs partageant des frontières physiques communes. Les matériaux les plus couramment utilisés sont le germanium, le silicium et l'arséniure de gallium. Dans ces matériaux ont été introduites des impuretés, par un processus appelé «dopage». Pour les semi-conducteurs de type N, les impuretés, ou dopants, entraînent un excès d'électrons (des charges négatives). Dans les semi-conducteurs de type P, les dopants conduisent à un déficit d'électrons, et donc un excès de porteurs de charges positives qu'on appelle "trous". Les transistors peuvent être utilisés comme des interrupteurs ou des amplificateurs. Nous allons ici plutôt nous concentrer sur leur premier rôle. Les transistors à jonction ont trois broches : la base, le collecteur et l'émetteur. Lorsque la base est alimentée par un faible courant, un courant proportionnellement (jusqu'à un certain point) plus intense peut passer entre le collecteur et l'émetteur (certains types de transistors ont des fonctionnements un peu différents mais nous présentons ici le cas général).

Sur les figures 1.7 et 1.8, nous pouvons voir la représentation simplifiée d'un transistor de type MOSFET (un transistor à effet de champ à grille isolée ou Metal Oxide Semiconductor Field Effect Transistor) qui est le plus couramment utilisé pour la fabrication de microprocesseurs. Ils existent en deux versions, type P ou type N et chacun peut être à enrichissement ou à appauvrissement. Celui sur la figure 1.7 est un transistor MOSFET de type N à enrichissement. Sur la figure 1.8 on peut voir sa structure et son fonctionnement. Les trois broches sont visibles mais ont des noms particuliers pour les MOSFET, la source (collecteur), la grille base) et le drain (émetteur). La source et le drain sont reliés ensemble par une bande isolante faite d'un oxyde et d'un cristal, puis par-dessus se trouve le fil de la grille. Lorsqu'une différence de potentiel est établie entre la grille et la source, la bande de métal et d'oxyde va générer un canal d'électrons qui connectera la source au drain permettant à un courant de passer. La taille de ce canal et donc l'intensité du courant qui peut passer est proportionnelle au potentiel de la grille, ce qui permet un contrôle fin du système.

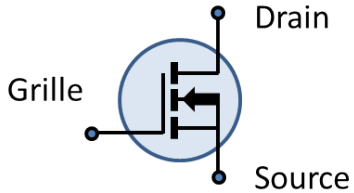


Figure 1.7 – Représentation schématique d'un transistor MOSFET de type N à enrichissement

1.1.4.2 Portes logiques électroniques

Il existe plusieurs familles de portes logiques électroniques, chacune ayant ses avantages et ses inconvénients : résistance-diode ou RDL (Resistor-Diode Logic), résistance-transistor ou RTL (Resistor-Transistor Logic), diode-transistor ou DTL (Diode-Transistor Logic), transistor-transistor ou TTL (Transistor-Transistor logic) et CMOS (Complementary Metal Oxide Semiconductor). Néanmoins depuis les années 1990, la plupart des portes logiques électroniques sont construites avec la technologie CMOS qui utilise à la fois des transistors complémentaires NMOS (type N) et PMOS (type P).

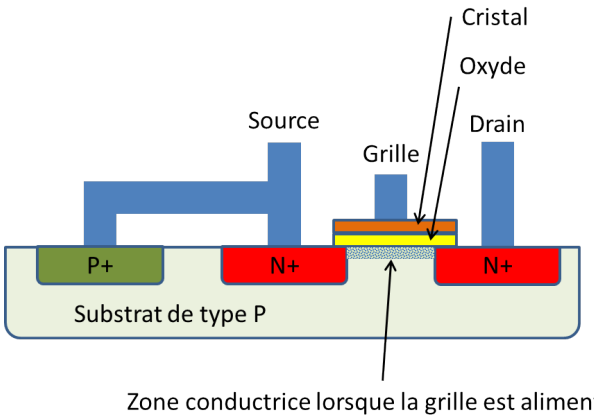


Figure 1.8 – Structure et fonctionnement d'un transistor MOSFET de type N. Lorsque la grille est alimentée, un canal d'électrons se forme entre la source et le drain permettant le passage du courant quasiment proportionnel au potentiel entre la grille et la source.

Ces portes logiques sont implémentées sous forme de circuit intégré modulaire pouvant facilement être réutilisé pour constituer d'autres circuits plus complexes. Dans la figure 1.9 nous pouvons voir un circuit intégré de la série 4000 de *Texas Instrument* implémentant une porte logique NOR [24]. Il a été conçu avec une méthode de design manuel, où des éléments de circuit peuvent être créés, partagés, et connectés à d'autres circuits sans difficulté. Cela permet d'accélérer la conception d'un nouveau matériel en autorisant la réutilisation des approches standards pour la conception de circuits. Le circuit comprend plusieurs parties, l'une est située en amont des entrées, et est principalement composée de diodes qui ont pour effet de protéger le montage contre les décharges électrostatiques (qui sont de petits courants électriques involontaires qui peuvent endommager les composants électriques susceptibles d'être souvent manipulés). Ensuite vient la porte logique elle-même, qui est composée de quatre transistors.

1.1.5 Implémentations existantes de réseaux logiques

De nombreux projets de développement visant à remplacer l'implémentation à base de silicium des systèmes logiques pour obtenir des ordinateurs plus performants ont été menées tout au long des vingt-cinq dernières années. Les alternatives sont généralement basées sur un dispositif qui commute très rapidement, et ils négligent beaucoup d'autres exigences de la logique de l'ordinateur. Robert W. Keyes en 1985 avait déjà établi les facteurs physiques essentiels qui expliquent le succès des transistors dans les applications, de même que les facteurs qui sont absents dans les dispositifs alternatifs proposés [25].

"Enchainabilité" : La sortie d'une porte logique doit être sous une forme adaptée pour être compatible avec l'entrée de la porte logique suivante (voir fig. 1.10).

Puissance de branchement (*fan-out*) : Le signal de sortie d'une porte logique doit être suffisant pour alimenter les entrées d'au moins deux portes suivantes (Puissance de branchement avec un gain de signal d'au moins deux, voir fig. 1.11).

Constance du niveau logique : La qualité du signal logique est établie de sorte que les dégradations éventuelles du signal ne se propagent pas à travers une porte logique. Le signal est *régénéré* en traversant chaque porte (voir fig. 1.12).

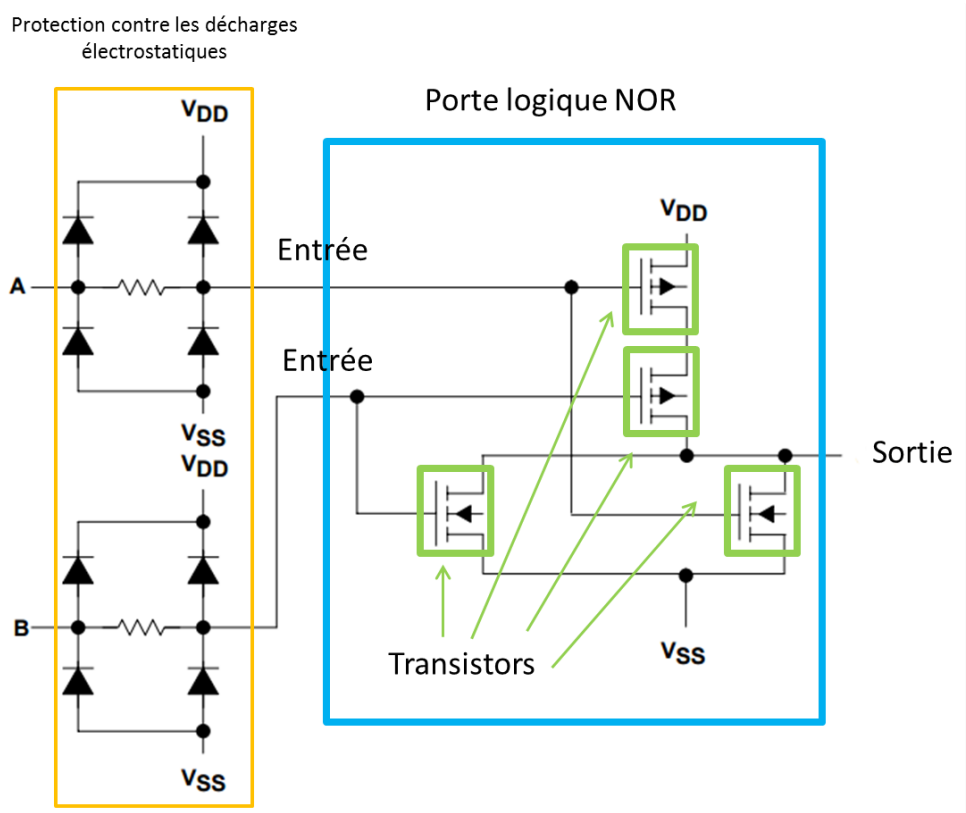


Figure 1.9 – Schéma interne des circuits intégrés de la série CD4xxxB pour l'implémentation sans tampon d'une porte NOR. Elle est composée de deux parties, un module se branchant sur les entrées en amont de la porte pour la protection contre les décharges électrostatiques, et un module implémentant la porte logique elle-même, qui est composée de quatre transistors. Adapté de *Understanding Buffered and Unbuffered CD4xxxB Series Device Characteristics of Texas Instrument* [24]

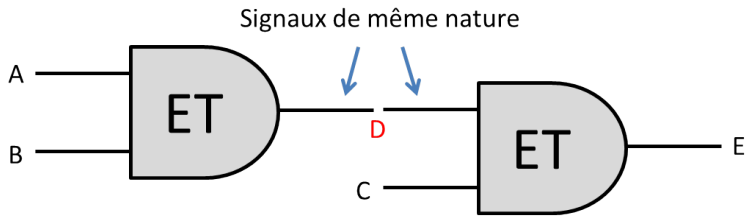


Figure 1.10 – Enchainabilité des portes

Isolation des signaux (diaphonie) : On veut éviter que les signaux de sortie ne se comportent comme s'ils étaient des signaux d'entrée ou, plus généralement, qu'une connexion en influence une autre. Les transistors fournissent cet isolement entre entrée et sortie naturellement, néanmoins un phénomène de diaphonie peut apparaitre si des pistes parallèles sont suffisamment longues à cause du phénomène d'induction électro-magnétique. Pour cela, on doit utiliser un dispositif de connexion avec des fils d'entrée et de sortie isolés (voir fig. 1.13).

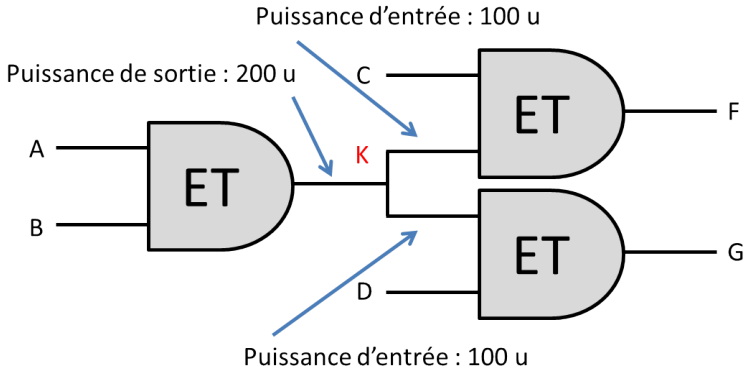


Figure 1.11 – Capacité d'une porte à avoir la puissance suffisante pour etre connectée à plusieurs autres.

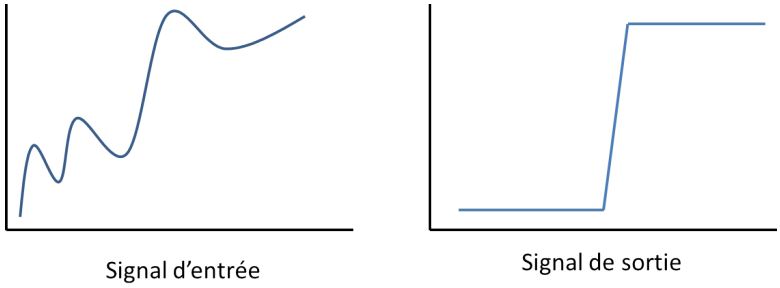


Figure 1.12 – Constance du niveau logique par *nettoyage* du signal à travers une porte.

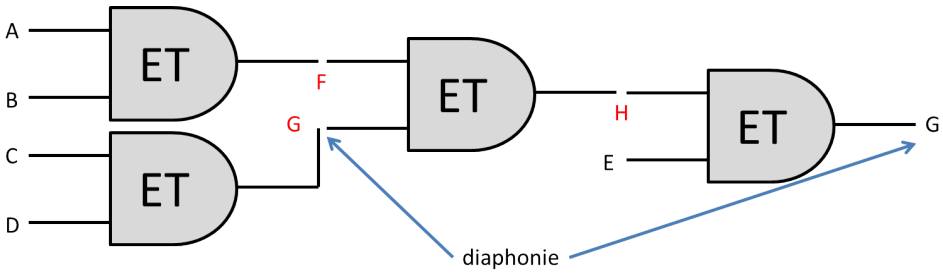


Figure 1.13 – Isolation des connexions entrée/sortie : ici, le support physique de la sortie (G) de la porte ET en bas à gauche est de même nature que celui utilisé pour la sortie de la dernière porte ET du circuit, d'où une diaphonie importante faussant le calcul.

Seuil de séparation des valeurs booléennes : On ne veut pas avoir à définir le seuil de séparation des niveaux logiques de chaque dispositif avec une précision trop élevée (voir fig. 1.14).

Niveau logique indépendant des pertes : Le niveau logique représenté dans un signal ne doit pas dépendre de la perte due à chaque porte intermédiaire. L'amplification du signal à travers les portes est le moyen employé dans les portes électroniques pour éviter ce problème (voir fig. 1.15).

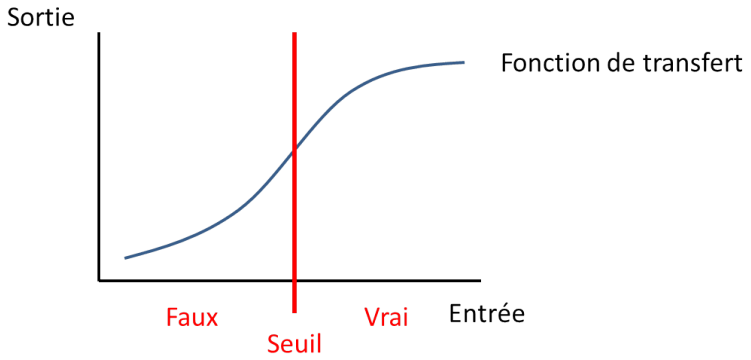


Figure 1.14 – Large intervalle de définition du seuil de séparation entre les niveaux logiques (absence de point critique dans la fonction de transfert d'une porte).

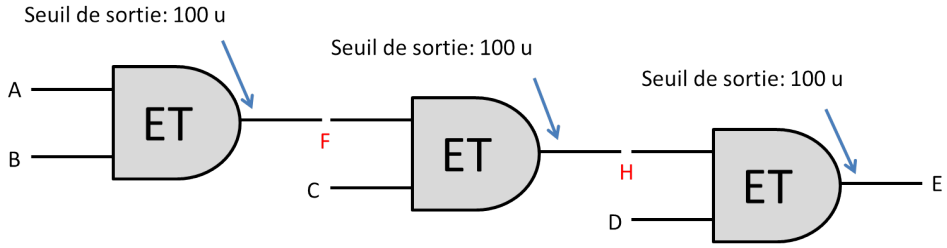


Figure 1.15 – Niveau logique indépendant des pertes : le même seuil de sortie, 100 unités par exemple, assure un fonctionnement correct de la logique de calcul de toutes les portes du circuit.

1.1.5.1 Implémentation optique

L'informatique optique ou photonique utilise les photons produits par des lasers ou des diodes comme support de l'information et pour le calcul. Le développement de ces systèmes est motivé par plusieurs avantages théoriques par rapport aux systèmes électroniques. Les transistors optiques peuvent commuter jusqu'à des fréquences de 4 térahertz, ce qui, comparé au transistor à électrons est 1000 fois plus rapide [26]. Ce gain de rapidité est expliqué par la nature des limitations de vitesse pour les transistors à électrons : le chargement/déchargement de la capacité parasite des fils (en général métalliques) qui connectent les transistors du circuit. L'information dans les systèmes optiques peut aussi prendre plusieurs formes comme, par exemple, la présence ou l'absence de faisceaux [27] ou encore la direction de polarisation [28], ce qui permet de transmettre encore plus de données par flux d'information.
Les facteurs physiques essentiels des circuits logiques ont été appliqués aux composants optiques [29] :

- **"Enchainabilité" :** Les longueurs d'ondes d'entrée et de sortie, les formes de faisceaux et les formes d'impulsions doivent être compatibles.
- **Puissance de branchement :** La puissance des faisceaux et impulsions doit être adaptée.
- **Constance du niveau logique :** Il faut rétablir la qualité du faisceau et/ou la qualité des impulsions ainsi que les niveaux de signal.
- **Isolation entrée/sortie :** La physique microscopique des processus optiques non linéaires n'isole généralement pas bien les signaux, il faut des guides d'ondes spécifiques pour l'optique linéaire.
- **Seuil de séparation des valeurs booléennes :** Les appareils reposant sur l'interférence

cohérente de faisceaux lumineux sont susceptibles de nécessiter des réglages de distance très précis.

— **Niveau logique indépendant des pertes :** La puissance du faisceau est affectée par les pertes. Par conséquent, un unique seuil de puissance ne permet pas distinguer de façon fiable entre *faux* et *vrai* à moins que le seuil soit proche de zéro, ce qui nécessite une très grande plage dynamique dans la modulation du signal et la détection.

Actuellement, la plupart des projets de recherche portent sur le remplacement des composants informatiques actuels par des équivalents optiques. Ces composants optiques pourraient alors être intégrés dans les ordinateurs traditionnels pour produire un hybride optique-électronique. Mais ces interfaces sont couteuses en énergie. Il y a donc aussi des projets pour construire des systèmes entièrement optiques. Ces systèmes sont maintenant suffisamment avancés pour que des modèles de portes logiques optiques plus sophistiqués soient possibles. *Baoa et al.* ont réussi à construire des portes logiques tout-optique à deux dimensions (2D) [27]. Les auteurs ont inventé un système qui permet d'implémenter une porte logique NOR, ou une porte logique NAND au prix de faibles modifications. Le système se compose d'une plaque sur laquelle sont disposés de petits bâtonnets de silicium. Dans cette grille sont aménagés des guides d'ondes qui sont des sections sans bâtonnet. Il y a cinq guides d'ondes dans le système : deux latéraux de taille L, qui réceptionnent les faisceaux d'entrées, un central qui traverse la grille verticalement, et qui réceptionne la source et donne la sortie. Il y a deux guides d'ondes de taille H situés parallèlement au guide d'ondes central. Enfin, il y a deux résonateurs optiques circulaires qui vont renforcer le signal et le purifier (voir fig. 1.16).

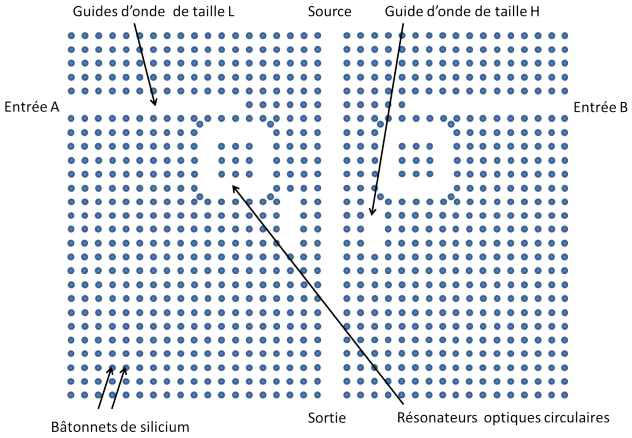


Figure 1.16 – Schéma du système optique. On voit les 5 guides d'ondes du système : les deux latéraux qui réceptionnent les entrées, le central qui traverse la grille verticalement et qui réceptionne la source et produit la sortie, et les résonateurs optiques qui renforcent le signal. Image adaptée de *Baoa et al.* [27]

Sur la figure 1.17 on peut voir le fonctionnement de la porte NOR. Chaque image montre un état de la porte. Lorsque un faisceau arrive dans l'un des guides d'ondes latéraux, il entre dans l'un des deux résonateurs optiques circulaires, puis dans l'un des guides d'onde de taille H. Ce faisceau va ensuite interagir par un phénomène d'interférence avec le faisceau central. Cette interaction dépend de la taille des guides d'ondes parallèles de taille H : ici avec H=8 on a une porte NOR. Sur la figure 1.18 on peut voir le fonctionnement d'une porte NAND quand H=5.

Les composants optiques sont une réalité, même s'ils ne sont pas encore une alternative viable pour les composants électroniques. Des nanotechnologies telles que les nanorésonateurs, les plasmoniques,

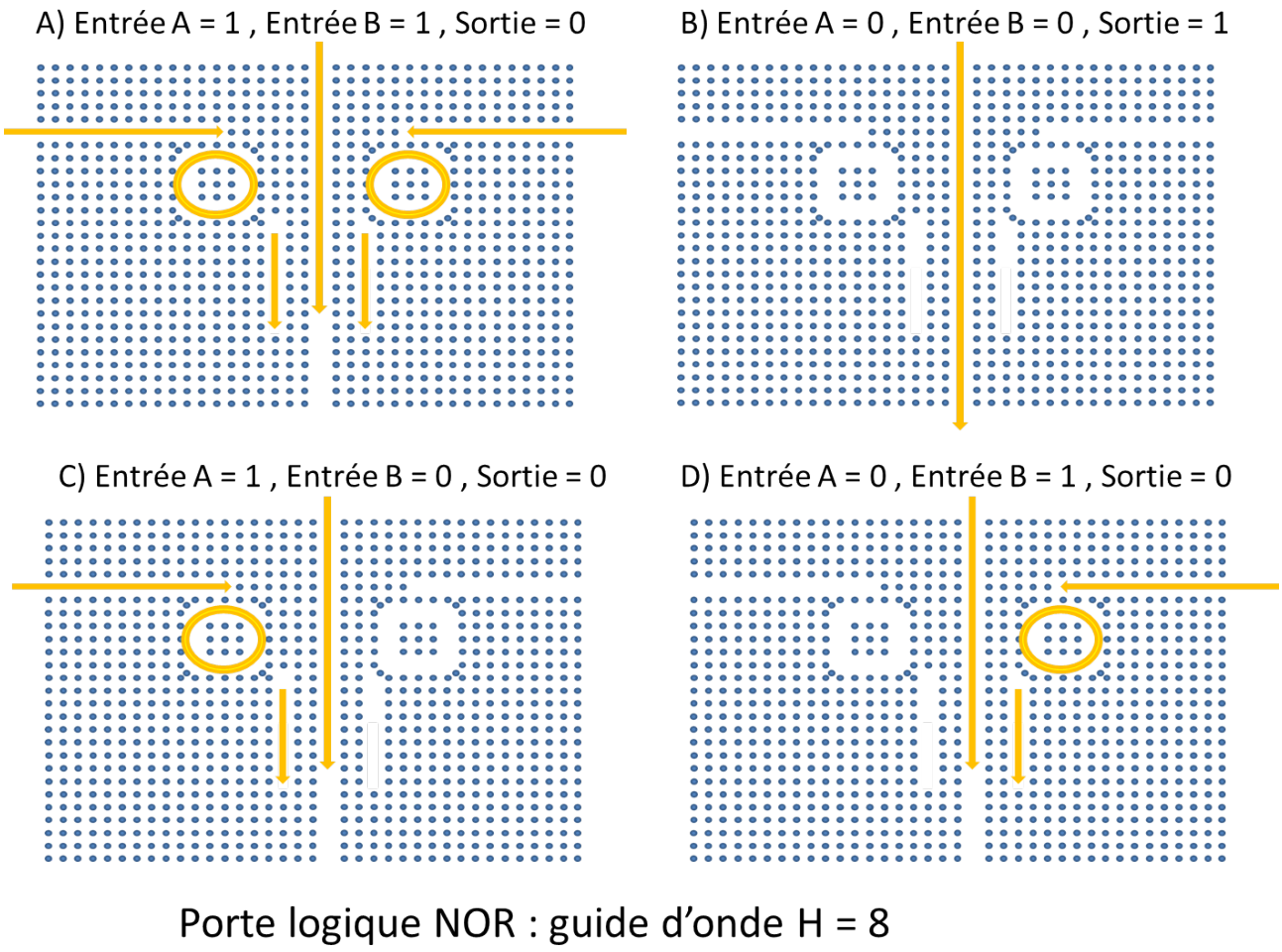


Figure 1.17 – Porte logique optique NOR avec un guide d'ondes parallèle de taille 8. On peut voir les quatre états de la porte. Image adaptée de *Baoa et al.* [27]

nanometallics, les points quantiques et même des molécules simples offrent de nouvelles possibilités pour l'informatique optique [29].

1.1.5.2 Biologie synthétique

Pour appliquer les principes de la logique booléenne sur un matériel biologique, il est nécessaire de pouvoir ingéniérer des systèmes biologiques. C'est le domaine de la "biologie synthétique". Le terme date du début du XX^{e} siècle [30] mais la discipline a vraiment commencé à prendre de l'ampleur avec le génie génétique et l'invention des techniques d'ADN recombinant en 1973. Il a en effet fallu attendre que la connaissance des procédés biochimiques et du fonctionnement des systèmes biologiques en général soit suffisamment avancée. Ces connaissances constituent l'un des piliers de la biologie synthétique, l'autre étant le savoir-faire en ingénierie acquis dans un grand nombre de disciplines qui peuvent être assez éloignées de la biologie, comme la chimie, les mathématiques ou encore l'informatique. La biologie synthétique est l'ingénierie du vivant, qui cherche comment construire des systèmes biologiques ayant la capacité d'effectuer des fonctions non-naturelles et des systèmes artificiels utilisant des composants d'origine biologique. C'est un champ de recherche vaste dont les d'applications

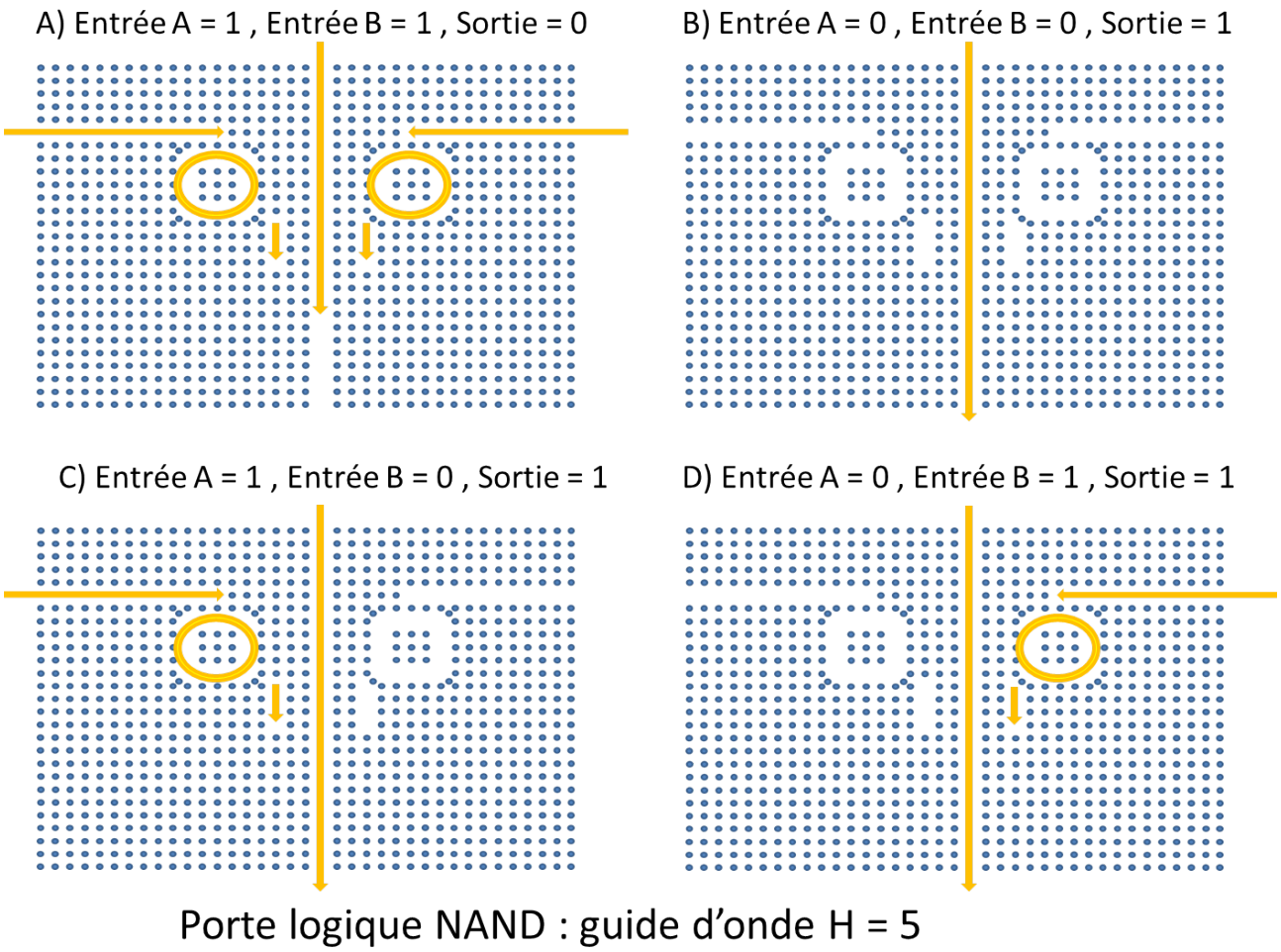


Figure 1.18 – Porte logique optique NAND avec un guide d'ondes parallèle de taille 5. On peut voir les quatre états de la porte. Image adaptée de *Baoa et al.* [27]

sont nombreuses. Nous distinguerons ici deux fonctions : la production et le traitement de l'information ainsi que deux principaux médias pour les réaliser (voir fig. 1.19). La création d'organismes génétiquement modifiés est un point fondamental et très probablement l'aspect le plus connu de la biologie synthétique, c'est la modification ou la "reprogrammation" d'organismes déjà existant pour accomplir de nouvelles fonctions, par exemple la production de biocarburants [31], la production de médicaments [32], ou encore pour effectuer du traitement de l'information [33] [34] [35] [36]. Il existe aussi d'autre fonctions plus spécifique telles que la création d'organismes génétiquement modifiés à des fins environnementales [37]. Un autre médium est la biochimie de synthèse, il est étroitement lié à la modification d'organismes vivants mais nous le considérerons ici comme séparé. En effet, il est possible de considérer le vivant, uniquement sous un aspect biochimique, le but sera donc de pouvoir créer de nouveaux chemins biochimiques ainsi que de nouvelles molécules. Il est aussi possible d'utiliser ces réseaux biochimiques pour d'autres applications que le vivant comme le traitement d'informations [38] [39] [40]. Nous nous sommes concentrés sur ces aspects *calcul* qui sont les plus pertinents pour notre travail de recherche.

Il existe actuellement des méthodes et des systèmes permettant la création et la construction de systèmes biologiques artificiels à partir de briques standardisées. Pour l'ADN, le projet des *Biobricks*

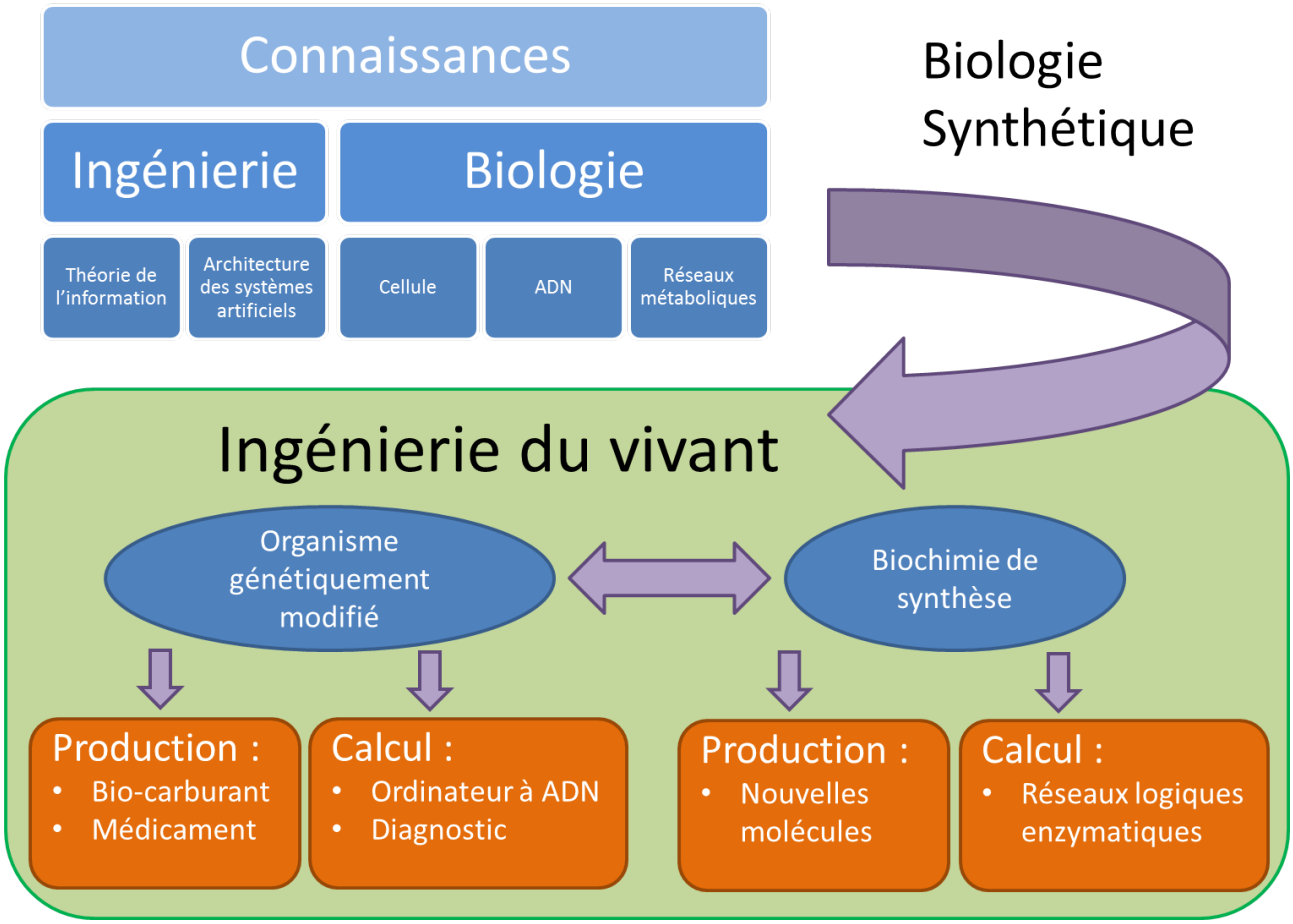


Figure 1.19 – Schéma représentant une vison de la biologie synthétique. La biologie synthétique a pour objectif l'ingénierie du vivant. La discipline a été créée dans la volonté d'appliquer nos connaissances en ingénierie à la biologie. Deux grands aspects de la biologie synthétique sont représentés, les organismes génétiquement modifiés et la biologie de synthèse. Chacun peut être utilisé pour différentes applications dont nous montrons uniquement deux exemples : la production d'un composé biologique et l'utilisation de composants biologiques pour effectuer du traitement de l'information.

créé par le MIT (Massachusetts Institute of Technology , *http* : *//partsregistry.org/Main*$_{PAGE}$) par exemple, donne accès à une importante bibliothèque de séquence d'ADN codant pour un grand nombre de molécules différentes. Sur la figure 1.20, on peut voir un exemple d'une liste de biobricks, ainsi que de la construction d'une biobrick plus complexe à partir d'éléments plus simples [41]. Pour les réseaux biochimiques, on peut citer *BioNetCAD* qui permet la conception de réseaux abstraits pouvant être mis en œuvre grâce à *CompuBioTicDB*, une base de données de pièces pour la biologie synthétique. Le logiciel permet également d'effectuer des simulations avec le logiciel GISS pour les équations différentielles ordinaires (ODE) et HSIM pour les simulations stochastiques. [42] (voir fig. 1.21).

Nous allons nous concentrer sur deux types d'implémentations des réseaux logiques réalisés grâce à la biologie synthétique : les circuits logiques génétiques et les circuits logiques enzymatiques.

Part Number	Function	Notation
BBa_G00000	BioBrick cloning site prefix	E X
BBa_G00001	BioBrick cloning site suffix	S P
BBa_P1016	*ccdB* positive selection marker	
BBa_I50022	pUC19-derived high copy replication origin	1
BBa_B0042	translational stop sequence	*
BBa_B0053 & BBa_B0054	forward transcriptional terminator	
BBa_B0055 & BBa_B0062	reverse transcriptional terminator	
BBa_G00100	forward verification primer annealing site (VF2)	
BBa_G00102	reverse verification primer annealing site (VR)	
BBa_B0045	NheI restriction site	N
BBa_P1006	ampicillin resistance marker (reverse orientation)	A

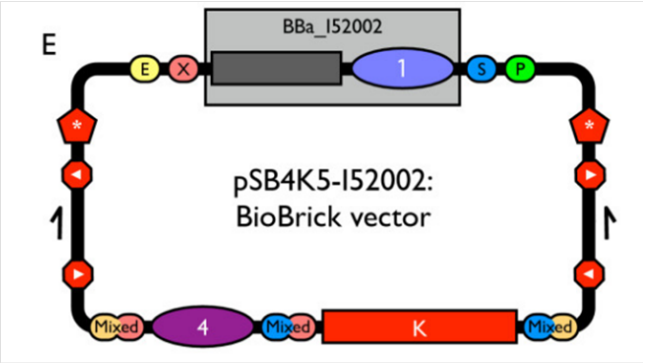


Figure 1.20 – Sur la gauche, on peut voir une liste de biobricks nouvellement créées qui seront ajoutées à la bibliothèque centrale. Ces biobricks sont définies par leur *Port Number* qui va être leur identifiant, leur fonction et une notation qui sera utilisée pour les représenter (même si cette représentation est propre à l'article et non universelle). Une information importante qui n'est pas mentionnée ici est leur séquence de nucléotides. Sur la droite on peut voir une biobick créée pour permettre de tester de nouvelles biobricks plus facilement. Cette biobrick complexe est composée d'un ensemble de biobricks plus simples. Tirée de [41]

1.1.5.3 Implémentation génétique

L'idée d'utiliser des molécules biologiques pour faire du calcul n'est pas nouvelle, en 1994 Leonard Adleman de l'université de Californie du Sud montra qu'il était théoriquement possible d'utiliser de l'ADN pour résoudre le problème du chemin hamiltonien à sept points [33]. Pour ce problème, il faut trouver un cycle commençant au sommet v_{in} et terminant au sommet v_{out} et passant par tous les sommets une seule et unique fois, l'algorithme qu'il a utilisé est le suivant :

- Étape 1 : Générer des chemins aléatoires à travers le graphe.
- Étape 2 : Garder uniquement les chemins qui commencent par v_{in} et terminant par v_{out}
- Étape 3 : Si le graphe a n sommets, garder seulement les chemins qui passent exactement par n sommets.
- Étape 4 : Garder uniquement les chemins qui passent par tous les sommets du graphe au moins une fois.

L'auteur a conçu une méthode de calcul à partir d'ADN pour effectuer l'algorithme. Il prend un graphe avec 7 sommets, puis pour tous les sommets i, il génère une séquence O_i aléatoire de 20 nucléotides. Ensuite pour chaque arc $i - j$, il construit une séquence O_{i-j} composé des 10 nucléotides en 3' de O_i

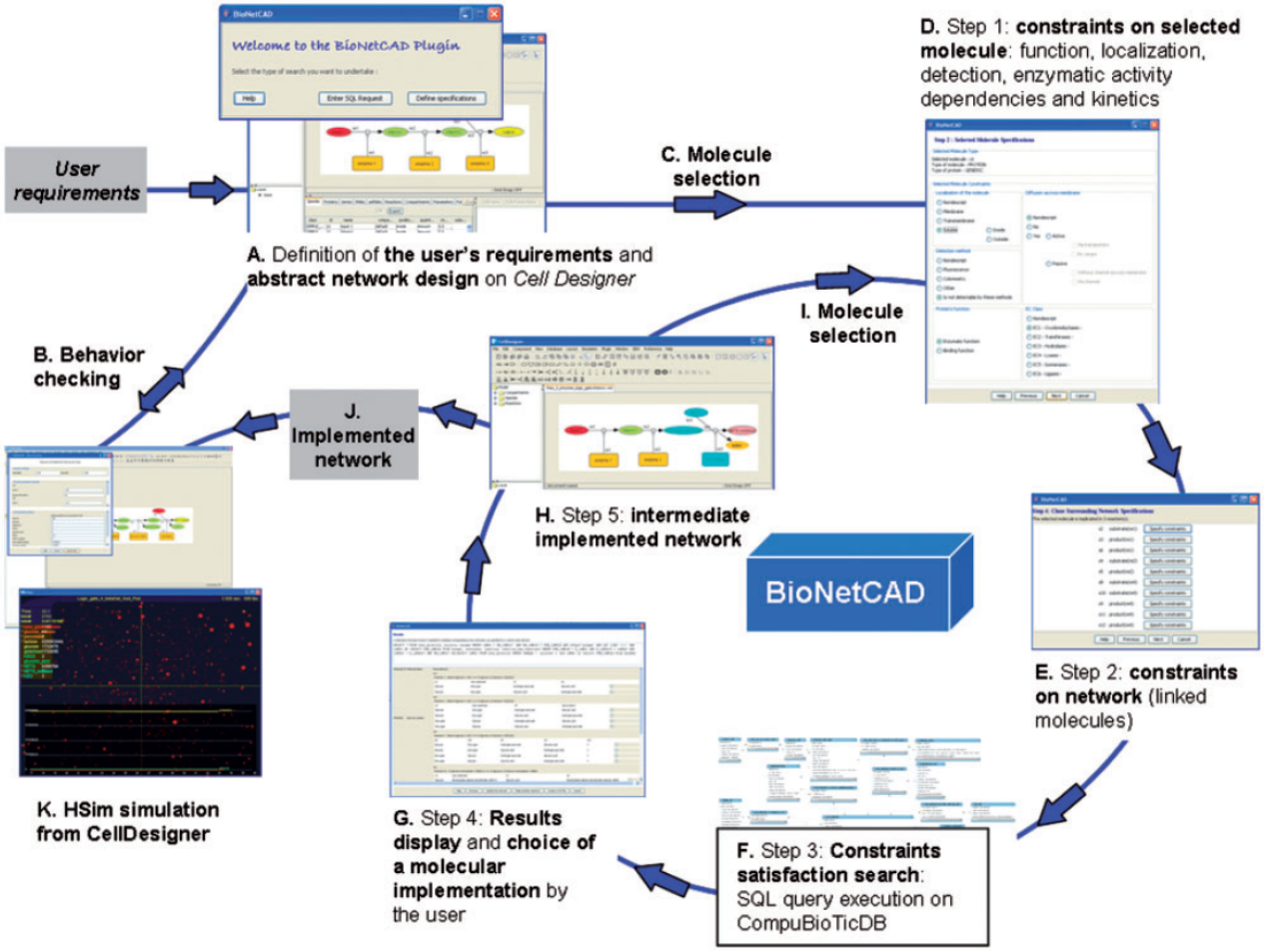


Figure 1.21 – Conception, mise en uvre et simulation avec BioNetCAD. La stratégie de conception commence par concevoir *à la main*, dans CellDesigner, un réseau abstrait qui permettrait de répondre à la question posée. Ensuite le plug-in est lancé et l'utilisateur est invité à sélectionner une molécule au sein du réseau, puis plusieurs étapes conduisent l'utilisateur à spécifier des contraintes sur la molécule sélectionnée et les molécules voisines dans le réseau. Par exemple, si l'utilisateur sélectionne une protéine qui catalyse une réaction, il sera en mesure d'indiquer la classe enzymatique (numéro EC), le substrat et produit de cette réaction. Ensuite, BioNetCAD intègre les spécifications et élabore une requête SQL sur CompuBioTicDB pour trouver des molécules qui répondent à ces exigences. Les résultats sont affichés et l'utilisateur peut choisir une molécule parmi celles proposées. Le réseau est ensuite mis à jour avec la molécule choisie et éventuellement avec des molécules liées, telles que les substrats ou les produits. Lorsque le réseau est assemblé, il est alors possible de générer automatiquement les fichiers d'entrée HSIM à partir du modèle CellDesigner pour lancer une simulation stochastique ou bien d'utiliser le logiciel GISS. Tirée de [42]

suivis des 10 nucléotides en 5' de O_j. Les séquences complémentaires des sommet sont notées O'_i (voir fig. 1.22).

L'auteur a ensuite appliqué cet algorithme aux séquences construites :

— **Étape 1 :** Pour chaque sommet i et pour chaque arc $i - j$ du graphe, 50 pMol de O'_i et 50 pMol de O_{i-j}, ont été mélangées ensemble.

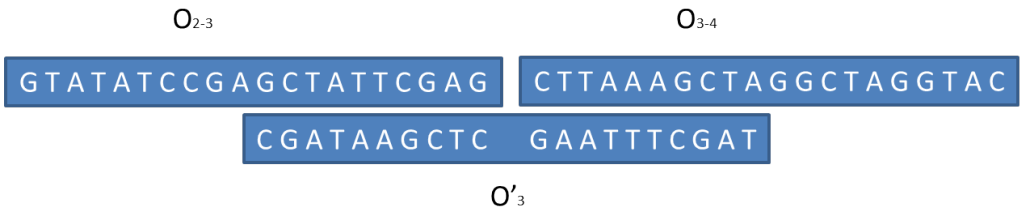


Figure 1.22 – Pour chaque sommet i, est générée une séquence O_i aléatoire de 20 nucléotides. Pour chaque arc $i-j$ est construite une séquence O_{i-j} composée des 10 nucléotides 3' de O_i suivis des 10 nucléotides 5' de O_j. Les séquences complémentaires des sommets sont notées O'_i. Image de *Adleman* [33]

— **Étape 2 :** Le produit de l'étape 1 a été amplifié par la méthode des réactions en chaîne par polymérase (PCR) en utilisant des amorces O_0 et O'_6. Ainsi, seules les molécules codant des chemins qui commencent par le sommet 0 et qui finissent par le sommet 6 sont amplifiées.

— **Étape 3 :** Les molécules restantes ont été placées sur un gel d'agarose pour mesurer leur taille et uniquement celles dont la taille est exactement de 140 paires de bases ont été sélectionnées.

— **Étape 4 :** Enfin, chaque séquence résultante a été séparée en ADN simples brins qui ont été multipliés. Il a fallut tester chacune avec une séquence O'_i et cela pour tous les O'_i pour vérifier que tous les sommets étaient bien présents.

L'idée présentée ici est le concept fondateur des calculs à base d'ADN : réussir à faire effectuer les opérations de calcul directement par l'ADN en utilisant ses propriétés de reconnaissance entre séquences. Ces calculs sont plus couteux en temps pris individuellement, mais on peut en faire un très grand nombre simultanément. On peut dire que les "ordinateurs" moléculaires sont par nature multiprocesseurs.

Par la suite de nombreux progrès ont été effectués, *Boneh et al.* ont généralisés les méthodes de simulation des calculs classiques utilisant l'ADN [43]. Ils ont montré comment calculer les affectations de manière efficace pour satisfaire les circuits booléens généraux avec des portes arbitraires binaires. Ils ont aussi montrés qu'il était possible d'utiliser de l'ADN pour faire le comptage approximatif des affectations satisfaisantes, ce qui signifie que l'ADN peut être utilisé pour faire des calculs qui vont au-delà de la classe de complexité NP. Ils ont néanmoins apporté une mise en garde en conclusion sur les problèmes de bruit inhérent aux systèmes à ADN. En effet, les processeurs à ADN se trouvent au sein d'une machinerie complexe (cellule, bactérie, etc.) qui est nécessaire à leur fonctionnement. Cela provoque une dégradation des molécules utilisées pour transmettre le signal, ainsi qu'un bruit non négligeable.

En 2004, *Benenson et al.* ont réussi à construire un ordinateur biomoléculaire in vitro [34]. Ce système analyse de façon logique les niveaux d'ARN messager de son environnement dans le but de faire varier le taux d'expression d'un gène précis. Cet "ordinateur" de près d'un milliard de plus petits ordinateurs par microlitre se compose de trois modules programmables :

— Un module de calcul, qui est un automate stochastique moléculaire.

— Un module d'entrée, dans lequel les niveaux d'ARNm spécifiques régulent les concentrations des molécules internes à l'ordinateur, et donc des probabilités de transition de l'automate.

— Un module de sortie, capable de libérer de façon contrôlée une courte molécule d'ADN simple brin.

Cette approche a pour but d'être appliquée *in vivo* pour la détection biochimique, le génie génétique

et le diagnostic médical. Ils ont adapté leur système pour identifier et analyser l'ARNm de gènes associés à des pathologies, avec des modèles de cancer du poumon à petites cellules et de cancer de la prostate, ainsi que pour produire une molécule d'ADN simple brin utilisée comme un médicament anticancéreux (voir fig. 1.23). Par exemple, pour obtenir un diagnostic du cancer de la prostate, il faut respecter quatre conditions :

— Un faible niveau de l'expression du gène PPAP2B.

— Un faible niveau de l'expression du gène GSTP1.

— Un fort niveau de l'expression du gène PIM1.

— Un fort niveau de l'expression du gène HPN.

Si ces conditions sont respectées, alors il y aura production d'une molécule contre le cancer de la prostate. En revanche si elles sont fausses alors c'est une autre molécule qui sera délivrée, qui inhibera la molécule précédente. Ce double contrôle permet d'affiner au mieux le diagnostic et le traitement. Il assurera que le bruit du système ne nuira pas au patient. En effet, il y a un très grand nombre de nano-ordinateurs qui feront le calcul, et même si les conditions sont toutes remplies ou toutes non remplies, il y a de fortes chances, à cause de divers facteurs non-contrôlables, que certains de ces nano-ordinateurs donnent une mauvaise réponse, même si globalement, la moyenne de ces nano-ordinateurs donne la bonne réponse. Mais il faut éviter la présence de la molécule anti-cancer si elle n'est pas nécessaire (l'inhibiteur de cette molécule n'étant lui pas problématique par lui-même), (voir table 1.1).

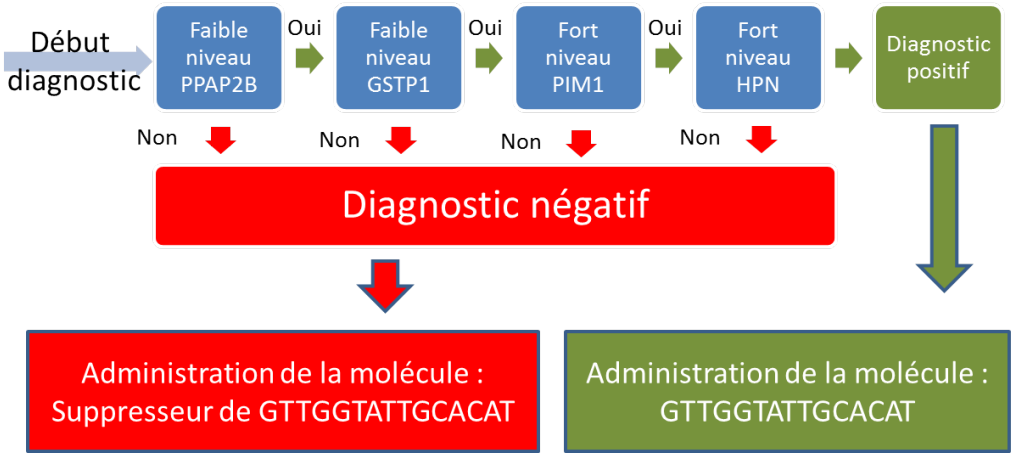


Figure 1.23 – Schéma expliquant le fonctionnement des ordinateurs à ADN pour le diagnostic du cancer de la prostate. Il y a quatre conditions à remplir : un faible niveau de l'expression du gène PPAP2B, un faible niveau de l'expression du gène GSTP1, un fort niveau de l'expression du gène PIM1, et un fort niveau de l'expression du gène HPN. Si le diagnostic est positif alors une molécule contre le cancer de la prostate est libérée. S'il est négatif alors c'est une autre molécule inhibitrice de la première qui est libérée. Schéma créé a partir de *Benenson et al.* [34]

Ces approches ont déjà conduit à la création de plusieurs composants à base d'ADN : des portes logiques [35], de la mémoire, des horloges et des oscillateurs. C'est un domaine maintenant bien maitrisé, *Auslander et al.* ont réussi à construire des portes logiques non triviales (autre que ET et OU). Ces deux portes logiques, des N-IMPLY (qui sont des ET avec une entrée inversée), utilisent

Diagnostic	Réponse positive (%)	Réponse négative (%)	Molécules finales disponibles (%)
POSITIF	90%	10%	80%
POSITIF	70%	30%	40%
POSITIF	60%	40%	20%
NÉGATIF	40%	60%	0%
NÉGATIF	20%	80%	0%

Table 1.1 – Réponse finale du système avec un double contrôle. La réponse négative produit une molécule qui inhibe la molécule produite par la réponse positive. Cela permet d'obtenir un résultat plus tranché.

les mêmes entrées et la même sortie mais leur fonction logique est différente : Erythromycin ETNON Phloretin donne d2EYFP pour l'une et Erythromycin NONET Phloretin donne d2EYFP pour l'autre. C'est un cas intéressant, car la majorité de leur composants sont communs, et on peut voir qu'il suffit de changer quelques composants pour changer radicalement la fonction logique (voir fig. 1.24). De même les résultats sont particulièrement bons puisque il n'y a aucune trace de bruit.

Nous avons vus que les cellules pouvaient être programmées en utilisant des circuits génétiques synthétiques pour effectuer une opération logique spécifique voulue. Cependant, même la conception des circuits simples est très chronophage et peu fiable. En effet, tous les travaux précédents sont basés sur l'étude d'un jeu de molécules particulier, et des possibilités de construction existantes à partir de ces pièces. La prochaine étape est de réaliser des éléments de logique standard tels que des portes logiques pour réussir à construire des circuits, si possible de façon automatique. L'automatisation de la conception électronique (EDA) a été développée pour aider les ingénieurs dans la conception de l'électronique à semi-conducteurs. Pour accélérer la conception de circuits génétiques, *Nielsen et al.* ont appliqué les principes de l'EDA pour permettre une augmentation de la complexité des circuits [36]. Ils ont utilisé le langage de description bas-niveau *Verilog* pour donner à l'utilisateur un moyen de décrire la fonction qu'il voudrait que le circuit implémente. L'utilisateur définit également les capteurs, les actionneurs et les "contraintes". Ces contraintes peuvent comprendre l'organisme dans lequel le circuit évoluera, le type de portes logiques, la technologie et les conditions de validation.

Ils ont créé le logiciel Cello (www.cellocad.org) qui utilise ces informations pour concevoir automatiquement une séquence d'ADN implémentant le circuit souhaité. Ce traitement est effectué par un ensemble d'algorithmes qui vont analyser le texte Verilog, créer le schéma de circuit, assigner des portes, vérifier les contraintes d'équilibre pour construire l'ADN, et enfin simuler les performances (voir fig. 1.25).

Cello conçoit des circuits en utilisant une bibliothèque de portes logiques booléennes. Il a été appliqué à la conception de 60 circuits pour *Escherichia coli*, dans laquelle la fonction du circuit a été spécifiée à l'aide de code Verilog et transformée avec succès en une séquence d'ADN. Les séquences d'ADN ont été ensuite construites comme indiqué, sans réglage supplémentaire, exigeant 880.000 paires de bases d'assemblage d'ADN. Parmi ceux-ci, 45 circuits fonctionnaient parfaitement pour tous les états logiques possibles.

Pour conclure nous pouvons appliquer les facteurs physiques essentiels des circuits logiques aux composants génétiques :

— **"Enchainabilité" :** Les portes logiques génétiques sont connectées par des espèces moléculaires. Donc l'espèce moléculaire représentant la sortie d'une porte doit être la même que l'espèce moléculaire représentant l'entrée de la porte suivante.

— **Puissance de branchement :** Nécessite des amplificateurs.

— **Constance du niveau logique :** Le signal de sortie est généralement nettoyé mais le bruit

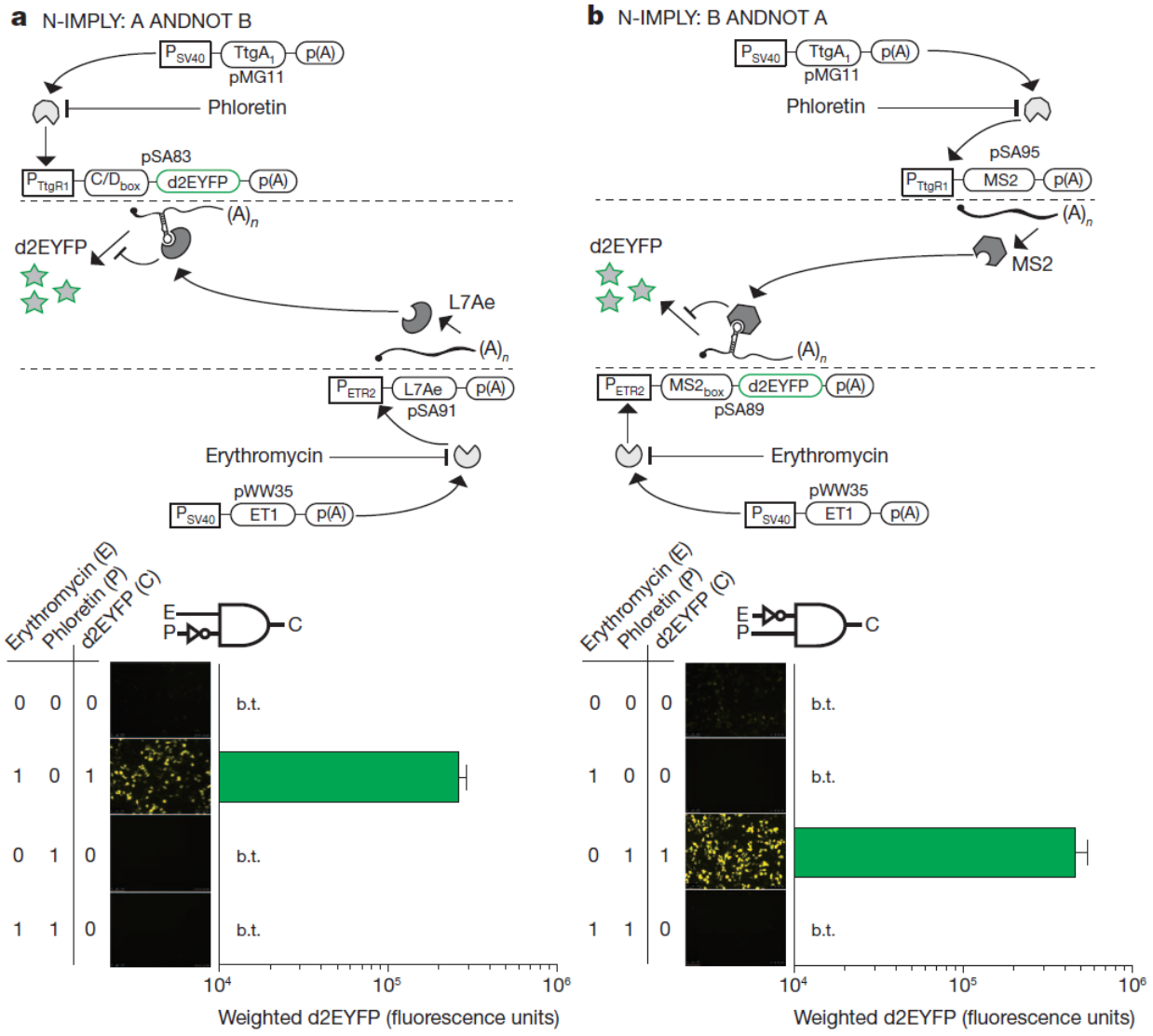


Figure 1.24 – Conception et performances de portes N-IMPLIQUE de synthèse dans des cellules humaines. **a :** la porte logique A ANDNOT B combine les deux signaux d'entrée érythromycine et phlorétine, les résultats sont conforme à la table de vérité. Les cellules HEK-293 sont programmées pour produire du d2EYFP exclusivement en présence de l'érythromycine et en l'absence de phlorétine comme montré par l'analyse au microscope par fluorescence. **b :** la porte logique B ANDNOTA combine les deux signaux d'entrée érythromycine et phlorétine, les résultats sont conformes à la table de vérité. Les cellules HEK-293 sont programmées pour produire du d2EYFP exclusivement en absence de l'érythromycine et en présence de phlorétine comme montré par l'analyse au microscope par fluorescence. Image de *Auslander et al.* [35]

ambiant reste un problème.

— **Isolation entrée/sortie :** Il faut des espèces moléculaires différentes pour chaque entrée et chaque sortie de chaque porte.

— **Seuil de séparation des valeurs booléennes :** Le seuil dépend de chaque porte, on doit

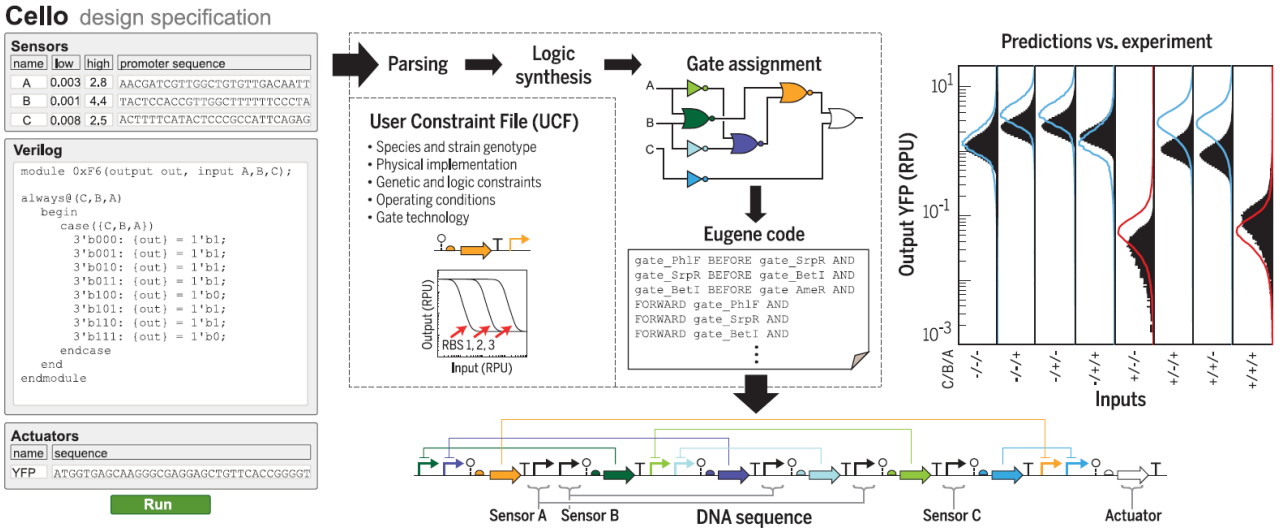


Figure 1.25 – L'utilisateur spécifie la fonction souhaitée dans le code Verilog, ce qui va donner une séquence d'ADN. Un exemple de circuit est représenté (0xF6). Les courbes rouges et bleues sont les états de sortie prévus pour les populations de cellules, et les distributions noires solides sont les données expérimentales de cytométrie. Les sorties sont présentées pour toutes les combinaisons d'états des capteurs, des signes moins (-) et plus (+) indiquent la présence ou l'absence de signal d'entrée. Image de *Nielsen et al.* [36]

regarder la forme de leur réponse pour savoir si elles sont enchainables.

— **Niveau logique indépendant des pertes :** L'accumulation des portes logiques successives nécessite des amplificateurs.

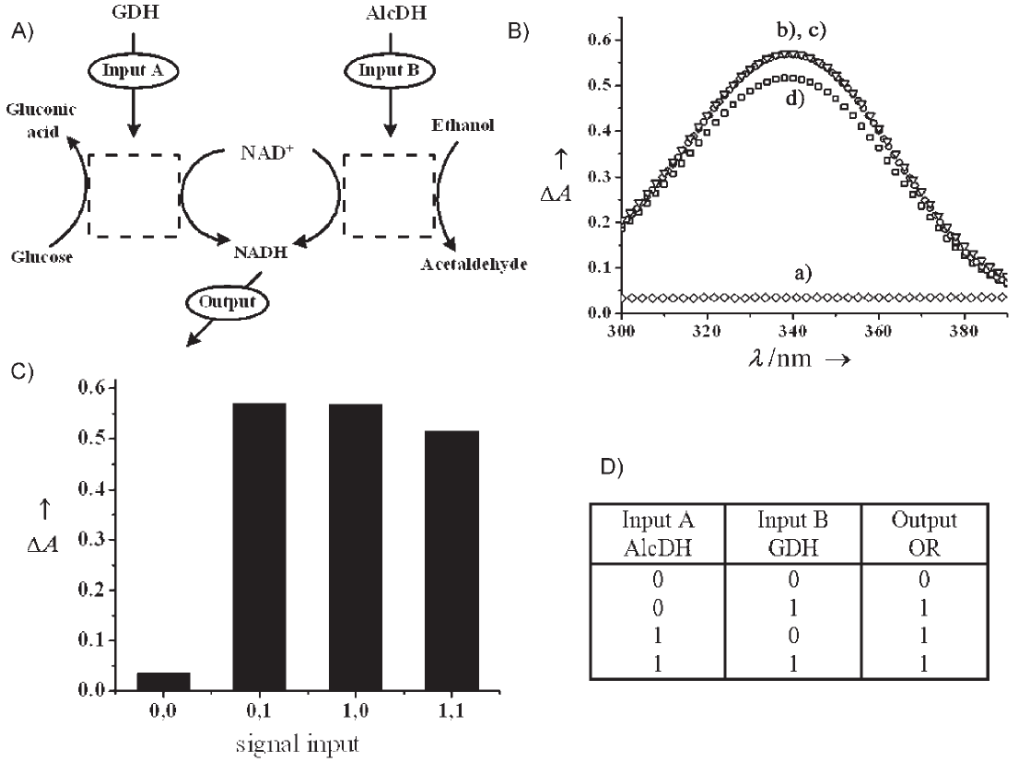
1.1.5.4 Implémentation avec des réactions enzymatiques

Il est également possible de construire des composants logiques avec des molécules organiques mais sans ADN. Cela permet de ne pas avoir besoin de toute la machinerie de transcription et de s'affranchir de tous ce qui n'est pas un élément interne du réseau logique. Des portes logiques moléculaires ont été construites avec des peptides [38] et des circuits de plusieurs portes logiques ont été réalisés avec des équations d'oxydo-réduction [44]. Il existe également des portes logiques à base d'enzymes. Elles utilisent la spécificité des réactions enzymatiques pour permettre à plusieurs réactions se déroulant simultanément dans une solution de ne pas interférer, donc limiter le *cross-talk* entre elles. Cela a permis la création de composants capable d'effectuer des calculs (théoriquement un demi-additionneur et un demi-soustracteur [39]) et de circuits logiques composés de plusieurs portes logiques concaténées qui fonctionnent simultanément dans une solutions [40]. Ces portes logiques sont composées d'enzymes solubles qui fonctionnent comme des dispositifs moléculaires qui acceptent des substrats (par exemple, le glucose et l'eau oxygénée) en tant que signaux d'entrée. Par exemple, *Strack et al.* ont montré qu'il était parfaitement possible de construire une porte fonctionnelle avec des réactions enzymatiques [45] (voir fig. 1.26). Pour construire une porte OU, ils utilisent deux réactions enzymatiques :

— $\text{Glucose} + \text{NAD}^+ \xrightarrow{\text{GHD}} \text{GluconicAcid} + \text{NADH}$

— $\text{Ethanol} + \text{NAD}^+ \xrightarrow{\text{AlcDH}} \text{Acetaldehyde} + \text{NADH}$

C'est une porte logique enzymatique assez simple, elle ne comprend que deux réactions enzyma-



Input A AlcDH	Input B GDH	Output OR
0	0	0
0	1	1
1	0	1
1	1	1

Figure 1.26 – Porte logique OU opérant avec des enzymes solubles comme signaux d'entrée. **A)** Représentation schématique du système. **B)** Spectre obtenu à 5 min après que les signaux d'entrée : a) "0,0" : sans addition de GDH ou AlcDH ; b) «0,1» : après addition de AlcDH (3,14 unités) ; c) «1,0» : après addition de GDH (1,5 unités) ; d) "1,1" : après l'ajout de AlcDH (3,14 unités) et de GDH (1,5 unités). **C)** Barres représentant l'absorbance des sorties de la porte OU à l = 340 nm. **D)** La table de vérité correspondant à la porte OU. Image de *Strack et al.* [36]

tiques mais elle donne bien du NADH si on a en entrée du glucose ou de l'éthanol (voir fig. 1.26). *Niazov et al.* ont réussi à construire expérimentalement un circuit logique enzymatique combinant trois portes logiques enzymatiques : un OU, un ET et un XOR [40]. Leur système se compose de quatre biocatalyseurs, l'acétylcholinestérase (AChE), la choline oxydase, la microperoxydase-11 (MP-11) et la glucose-déshydrogénase (GDH). L'Acétylcholine (entrée A) et la butyrylcholine (entrée B) sont hydrolysées par l'acétylcholinestérase pour former de la choline, qui agit en tant que sortie de la première porte OU. Ensuite la choline générée et l'O_2 (entrée C) activent la porte ET qui donne de la bétaïne aldéhyde et de l'H_2O_2 en tant que produits. La porte suivante, le XOR utilise deux enzymes, MP-11 et GDH. Les entrées de la porte sont l'H_2O_2 qui est le produit de la porte ET, et du glucose (entrée D). La sortie finale de la porte OU EXCLUSIF est du NADH qui est mesuré par absorbance (voir fig. 1.27).

Néanmoins les auteurs ont précisé que le fonctionnement de la concaténation des portes logiques enzymatiques a nécessité de nombreuses expériences d'optimisation pour obtenir des concentrations des entrées et des biocatalyseurs valides.

Pour conclure nous pouvons appliquer les facteurs physiques essentiels des circuits logiques aux composants enzymatiques :

— **Enchainabilité :** Les portes logiques enzymatiques sont connectées par des espèces molé-

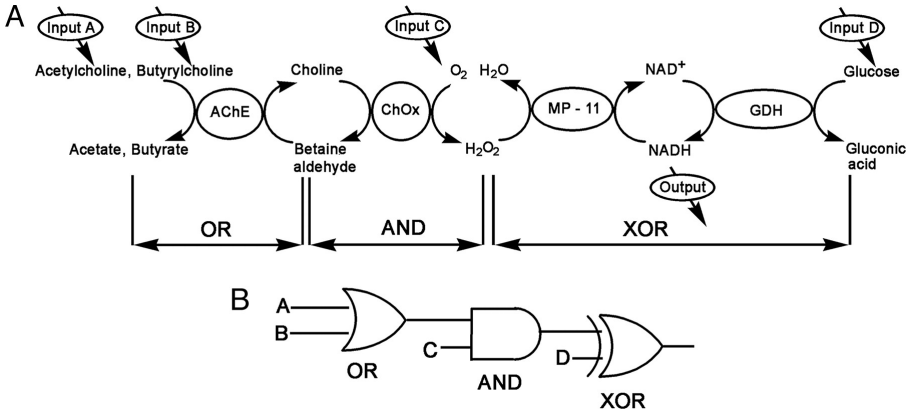


Figure 1.27 – Circuit logique enzymatique constitué de trois portes logiques enzymatiques : OU, ET et XOR. Les entrées sont l'acétylcholine, le butyrylcholine et l'O_2. L'acétylcholine et le butyrylcholine peuvent être consommés pour produire de la chlorine, c'est la porte OU. La chlorine et l'O_2 doivent tous les deux être présents pour être consommés et produire de l'H_2O_2, c'est la porte ET. Ensuite il faut, soit du H_2O_2, soit du glucose, mais pas les deux à la fois pour obtenir du NADH, c'est la porte OU EXCLUSIF. Illustration de *Niazov et al.* [40]

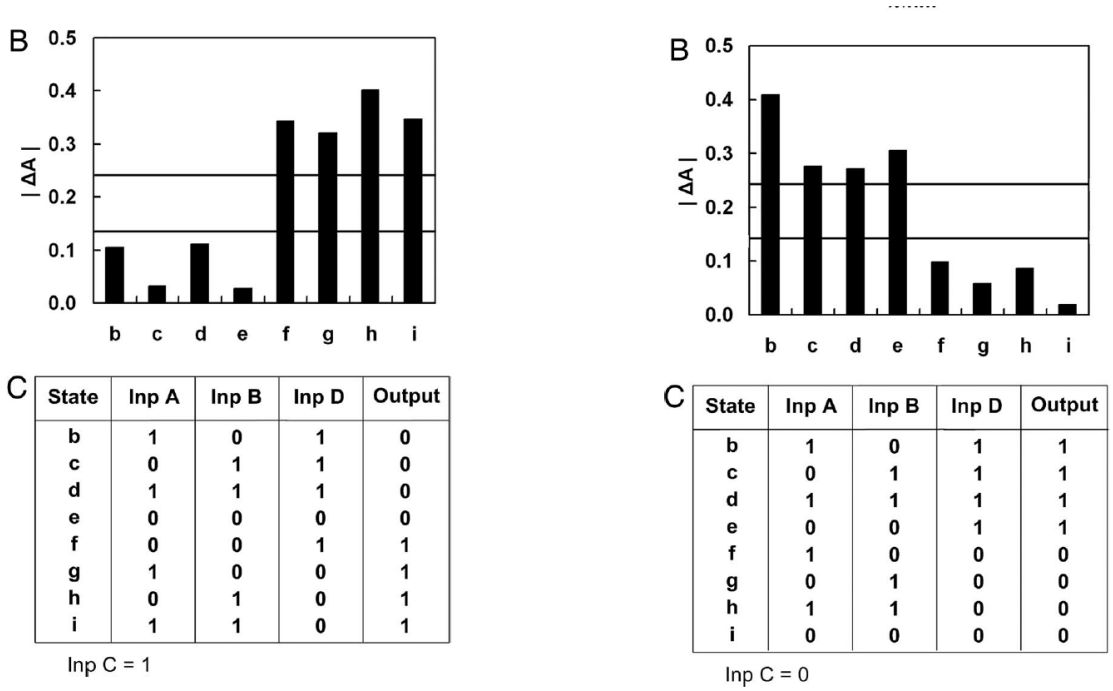


Figure 1.28 – Analyse de l'implémentation réelle des portes biocatalytiques concaténées pour les quatre entrées. **B :** Barres représentant la sortie des portes concaténées, dérivées à partir de l'absorbance du NADH. Les graphiques de gauche représentent les expériences ou l'entrée C (l'O_2) est à 1. Le seuil d'absorbance peut prendre deux valeurs, 0.14 et 0.24, qui sont indiquées pour toutes les barres. **C :** Table de vérité du circuit avec les portes concaténées en présence de l'O_2. Illustration de *Niazov et al.* [40]

culaires comme leurs homologues génétiques. Donc l'espèce moléculaire représentant la sortie d'une porte doit être la même que l'espèce moléculaire représentant l'entrée de la porte suivante.

— **Puissance de branchement :** Nécessite des amplificateurs.

— **Constance du niveau logique :** Dépend très fortement du type de portes.

— **Isolation entrée/sortie :** Il faut des molécules différentes pour chaque entrée et chaque sortie de chaque porte.

— **Seuil de séparation des valeurs booléennes :** Le seuil dépend de chaque porte, on doit regarder la forme de leur réponse pour savoir si elles sont enchainables.

— **Niveau logique indépendant des pertes :** Trop peu de tests.

Les réponses sont similaires à celles pour les portes logiques génétiques sur certains points. Sur d'autres, il est difficile de répondre à cause du manque de données sur le sujet. Dans le chapitre 2, nous allons définir de façon beaucoup approfondie toutes les règles des réseaux logiques enzymatiques, ce qui nous permettra de répondre sans ambiguïté à toutes ces questions.

1.2 Construire des réseaux enzymatiques implémentant une fonction booléenne

Nous voulons pouvoir construire des circuits logiques enzymatiques répondant une fonction logique précise. Nous avons vu qu'actuellement il est tout à fait possible de construire des portes logiques enzymatiques et de les utiliser ensuite dans de petits circuits. Mais la tâche de conception reste encore complètement manuelle. Pour chaque porte, il est nécessaire d'étudier le sous-réseau enzymatique la constituant, qu'il faut analyser à la main. De même, la création de circuit est aussi complètement manuelle, et donc il va falloir tester indépendamment chaque liaison entre portes et vérifier à chaque ajout de porte, que le circuit reste valide. Si l'on veut permettre à ce type d'implémentation enzymatique de se populariser et d'être utilisé dans plus de projets de recherche et industriels, il est nécessaire de pouvoir automatiser ces tâches. Il faut donc trouver un moyen d'obtenir un grand nombre portes logiques enzymatiques de façon fiable et automatisée. Ces portes logiques pourront ensuite constituer une bibliothèque qui pourra servir lors que la création de circuits logiques. Enfin, il faudra que l'on puisse créer des circuits logiques enzymatiques implémentant une fonction logique déterminée par l'utilisateur.

Pour cela nous allons avoir besoin de plusieurs éléments :

— **Une théorie générale des portes logiques enzymatique :** La construction d'une méthode automatisée de recherche de portes va nécessiter que l'on puisse analyser et comprendre les réseaux métaboliques d'un point de vue des portes logiques booléennes. C'est un cadre qui n'est pour l'instant pas encore suffisamment développé. Il nous faut donc approfondir considérablement notre connaissance du fonctionnement et de la dynamique des circuits logiques enzymatiques. Cette nouvelle théorie doit s'intéresser à la structure, à la dynamique et aux propriétés des portes logiques et des circuits logiques enzymatiques. Cela constituera la deuxième partie de ma thèse.

— **Un logiciel, *NetGate*, qui cherche des portes logiques enzymatiques :** Une fois que nous aurons les bases théoriques nécessaires, nous allons pouvoir concevoir et implémenter un algorithme capable d'analyser des réseaux métaboliques et d'en extraire des portes logiques enzymatiques. Ce logiciel va commencer par décomposer des réseaux métaboliques déjà existants pour construire tous les sous-réseaux possibles, ensuite chaque sous-réseaux sera testé et évalué pour déterminer s'il implémente une porte logique particulière. Les portes logiques enzyma-

tiques validées seront stockées et triées pour ne conserver que les modèles les plus optimisés. Cela sera explicité au début de la troisième partie de cette thèse.

— **Un logiciel, *NetBuild*, qui assemble des portes logiques enzymatiques :** maintenant que nous disposons d'un répertoire de porte logiques enzymatiques, nous allons devoir les assembler et les enchaîner pour construire des circuit logiques enzymatiques. Nous allons commencer par transformer l'équation logique donnée en paramètre pour obtenir toutes les formes intéressantes de cette équation. Ensuite on tentera d'implémenter chacune de ces formes grâce aux portes logiques enzymatiques à notre disposition. Nous prendrons pour chaque forme sa représentation arborescente que nous parcourrons depuis la racine jusqu'aux feuilles. Chaque opérateur logique sera testé dans le but d'être identifié à une porte logique enzymatique connue. Cette partie sera explicitée à la fin de la troisième partie de cette thèse.

1.2.1 Chercher des portes logiques enzymatiques dans des réseaux métaboliques

Maintenant que nous avons établi pourquoi nous cherchons des portes logiques enzymatiques, nous allons essayer de regarder comment trouver ces portes. Les portes logiques enzymatiques sont composée d'un ensemble de réactions enzymatiques, chacune de ces réactions possède un jeu de constantes cinétiques qui vont en déterminer leur vitesses. La nature des réactions ainsi que la valeur des constantes cinétiques sont en partie liées au milieu dans laquelle elles se produisent. *Laider et Peterman* [46] définissent le lien unissant les constantes cinétiques à la température comme étant :

$k = Ae^{-E/RT}$

Avec A le facteur de fréquence, R la constante des gaz parfaits, E l'énergie d'activation et T la température absolue.

De même, ils ont montré que le pH joue aussi un rôle important. Un réseau métabolique fonctionne donc avant tout dans un environnement déterminé. Cela ne veut pas dire qu'en dehors de cet environnement il n'est plus valable, mais seulement qu'il est possible que le réseau fonctionne différemment dans des conditions différentes. Pour un réseau métabolique particulier, nous avons des connaissances sur les relations qui unissent les différents composants moléculaires de ce réseau. Si nous mélangeons deux réseaux de provenance différente, rien ne nous permet de supposer que ces deux réseaux ne vont pas se perturber l'un l'autre, à part bien sur de le tester expérimentalement.

Pour maximiser les chances de trouver des portes logiques enzymatiques fonctionnelles, nous allons les chercher dans les réseaux métaboliques deja existant dans des organismes vivants. Les portes seront issues du même réseau métabolique, et donc, tous leurs constituants, réactions et composés, seront extraits du même réseau métabolique. Une porte logique "à cheval" sur deux réseaux aurait en effet beaucoup moins de chances de fonctionner car les environnements de chaque réseau sont probablement différents. Pour les mêmes raisons d'environnement commun, nous allons essayer de n'utiliser pour un même circuit que des portes logiques enzymatiques provenant du même réseau métabolique (voir fig. 1.29).

1.2.2 Spécificités de l'assemblage de portes logiques enzymatiques

Nous voulons pouvoir créer automatiquement des circuits logiques biologiques répondant à une fonction logique prédéfinie. Pour accomplir cela, nous devons être capables de concevoir et de construire des circuits logiques enzymatiques à partir d'une bibliothèque de portes logiques enzymatiques. Nous avons vu plusieurs éléments et méthodes nécessaires à la construction de cette bibliothèque, maintenant nous allons voir comment assembler ces portes logiques pour qu'elles forment des circuits logiques enzymatiques. La construction automatisée de circuits logiques à partir d'une liste de portes logiques disponibles est un champ de recherche qui a déjà été étudié. En 1980, Carver Mead et Lynn Conway

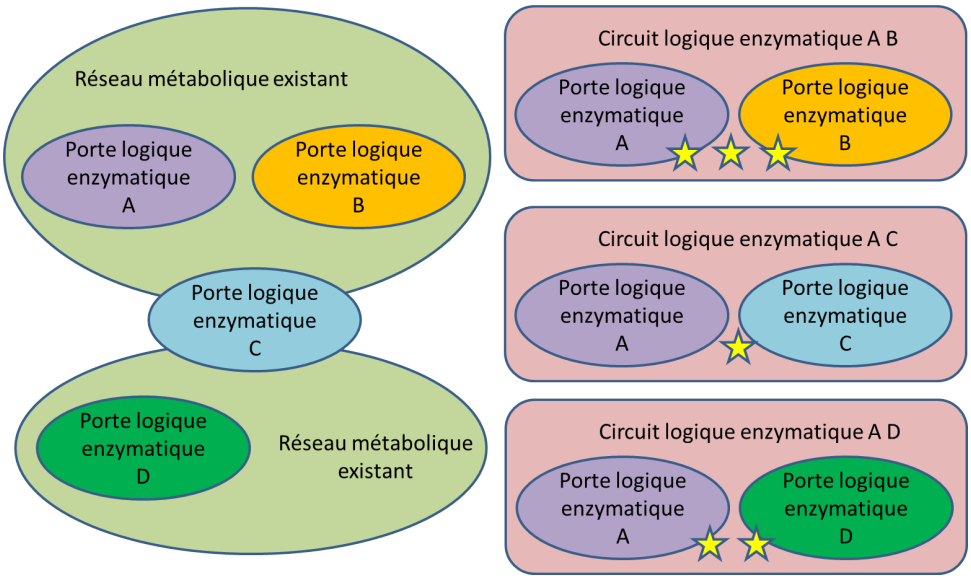


Figure 1.29 – Schéma expliquant plusieurs situations possibles pour la recherche de portes logiques et leur associations. Il y a plusieurs portes logiques enzymatiques utilisées et plusieurs circuits logiques enzymatiques possibles. Leur chance de fonctionnement sont représentées par des étoiles : plus il y en a, mieux c'est. Nous avons quatre portes logiques enzymatiques : les portes A et B sont issues d'une même et unique réseau, la porte D vient d'un autre unique réseau et la porte C vient de deux réseaux différents. Nous avons ensuite pris trois exemples de circuits logiques enzymatiques : le circuit A B est celui qui a le plus de chance de fonctionner car les deux portes viennent chacune d'un unique réseau et ce réseau est le même pour les deux portes. Ensuite pour le circuit A D, les deux portes viennent chacune d'un unique réseau et ce réseau est différent, il a une chance moyenne de fonctionner selon la compatibilité des réseaux en question. Finalement vient le circuit A C qui à le moins de chance de fonctionner car la porte C est la moins sûre.

dans *Introduction to VLSI Systems* ont développé le concept de langages de programmation spécifiques servant à construire et à modéliser des circuits logiques. Ces langages comme *VHDL* et *Verilog* sont des langages informatiques qui vont permettre de décrire des systèmes électroniques comme des circuits logiques. Ils vont codifier le fonctionnement des composants en prenant en compte des comportements et des attributs spécifiques des matériaux avec lesquels les circuits logiques seront construits. Ces langages sont communément appelés des langages RTL (*Register Transfer Level*) car ils vont simuler le fonctionnement des circuits logiqes par des transferts de flux d'informations entre des registres, et des opérations binaires entres ces flux d'informations.

Ces langages ont permis la création d'un workflow standardisé pour la conception automatisée de circuits électroniques. Ce workflow décrivant le processus est appelé flot de conception ou EDA (pour *Electronic Design Automation*). Nous allons distinguer plusieurs étapes (voir fig. 1.30) :

— **Spécification des fonctionnalités :** La première étape est d'établir toutes les fonctionnalités que nous voudrions que le futur circuit réalise. Il faut construire le "cahier des charges", qui comprendra la table de vérité du circuit, le matériel autorisé, les portes logiques électroniques disponibles ainsi que d'autres informations de ce type.

— **Représentation dans un langage RTL :** Cette étape va servir à formaliser toutes les spécifications qui ont été établies à l'étape précédente. Le circuit sous forme de langage RTL est purement abstrait, mais tous ses composants ont déjà été décris tel qu'on voudrait qu'ils

soient, et quelles fonctions nous voudrions qu'ils exécutent.

— **Design du circuit :** Cette étape représente la construction du circuit physique, comment les composants seront placés, comment communiqueront-ils ensemble. Nous verrons plus en détails ce que cela comprend ensuite.

— **Simulation et vérification :** Enfin la dernière étape sert à valider le circuit construit, nous allons donc le simuler et le modéliser pour s'assurer que son comportement correspond bien a nos attentes.

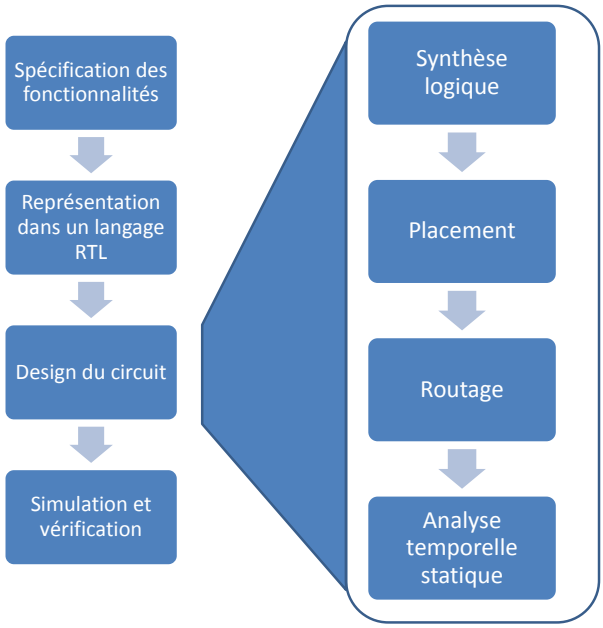


Figure 1.30 – Cycle de conception d'un circuit logique électronique.

Nous allons voir une partie plus spécifiquement : le design de circuit. En effet c'est dans cette étape que sera décidé de la conception réelle et physique du circuit électronique (voir fig. 1.31) :

— **Synthèse logique :** La première étape du design physique du circuit est la transcription du programme RTL spécifiant tous les éléments nécessaires au circuit, en un circuit logique abstrait constitué de portes logiques. C'est notamment la conversion de la table de vérité du circuit en équations logiques qui devront être implémentées sur la carte imprimée.

— **Placement :** La deuxième étape consiste à placer les éléments physiques correspondant aux éléments logiques présents dans les équations trouvées à l'étape précédente. Pour trouver les composants physiques équivalents, nous allons utiliser le répertoire des éléments physiques disponibles et construits dans l'étape de spécification des fonctionnalités. Ensuite ces éléments vont être placés sur la carte imprimée. Ce placement peut dépendre de plusieurs facteurs, par exemple la longueur totale des pistes sur le circuit, la taille de la piste la plus longue, la répartition de la température. Plusieurs de ces caractéristiques nécessitent d'effectuer l'étape suivante, le routage, pour être calculées. Cette étape et la suivante ne sont pas toujours indépendantes.

— **Routage :** Une fois que les composants physiques implémentant les opérations logiques ont été placés sur la carte imprimée, il faut les relier entre eux. Ce n'est pas un problème simple, les circuits imprimés ont en général des dimensions très contraintes, ils sont donc très denses et de

plus, il y a plusieurs propriétés des ondes électromagnétiques (effet d'antenne, diaphonie, etc.) qu'il faut prendre en compte afin de pouvoir dessiner des pistes de métal qui fonctionneront correctement ensemble et seront capables de transmettre l'information de façon fiable.

— **Analyse temporelle statique :** Cette étape constitue une première vérification qui peut s'effectuer de façon statique, il s'agit de regarder les effets temporels dû à la traversée des composants du circuit (composants actifs et pistes) par le signal pour vérifier par exemple que le temps de traversée total du circuit sera toujours inférieur à un tic d'horloge.

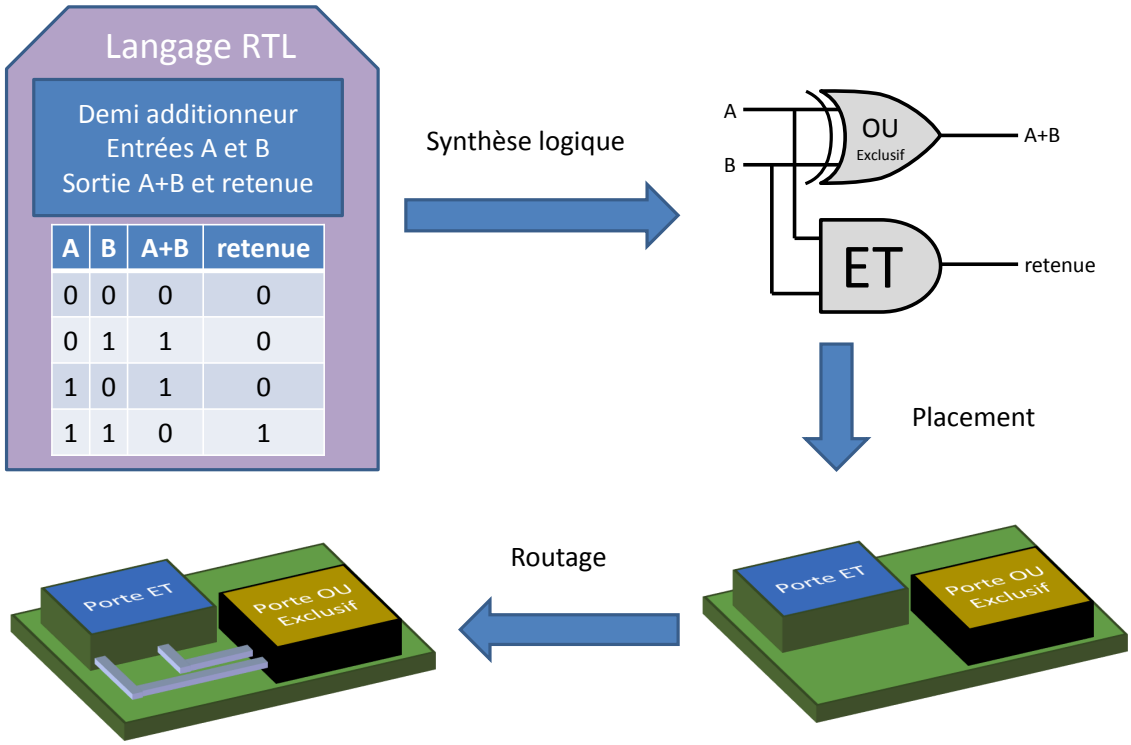


Figure 1.31 – Design d'un circuit logique électronique

Ce workflow permet de concevoir et de construire à priori n'importe quel circuit logique de façon automatisée. De nombreux logiciels existent implémentant différents algorithmes utilisant ce workflow. Nous avons vu précédemment, pour les réseaux logiques génétiques, des chercheurs [36] utiliser les principes de l'EDA pour la conception automatisée de circuits logiques génétiques, ils ont bien sur largement adapté le workflow pour qu'il fonctionne pour des composants génétiques et non électroniques. Nous même, pour construire nos circuits logiques enzymatiques nous n'allons en reprendre que certaines grandes lignes car contrairement aux circuits génétiques, il y a de grandes différences dans la conception de ces différents types de circuits.

Notre processus de conception commence une fois que nous est donnée l'équation logique pour laquelle nous devons produire un réseau de portes enzymatiques la calculant. À la différence des circuits électroniques, dans notre approche, il n'y a ni placement, ni routage puisque les composants, enzymes et métabolites, vont être mélangés dans la même vésicule. On pourrait donc penser que concevoir des circuits logiques enzymatiques est plus simple et que cela va nécessiter moins d'étapes, mais en réalité,

les étapes restantes sont beaucoup plus complexes que leur équivalentes en électronique. C'est le choix des composants physiques servant à implémenter les composants logiques abstraits qui va déterminer la façon dont ces éléments existeront dans l'espace, et la façon dont ils vont communiquer entre eux. Cette particularité est due à la structure de nos circuits logiques enzymatiques : ils sont composés d'espèces moléculaires réparties de façon homogène dans l'espace, la communication s'effectuant par le fait que certaines de ces espèces réagissent ensembles. Tous le processus de conception, à part les parties de vérification post-conception est donc regroupé dans une seule étape où toutes les contraintes qui existent sur les circuits logiques doivent être vérifiées.

En plus de cette différence déjà importante, il existe certaines étapes non présentes au sein de la conception des circuits logiques électroniques qui sont nécessaires pour la conception de circuits logiques enzymatiques. En effet, le choix de la forme de l'équation logique utilisée pour la construction du circuit électronique va dépendre de facteurs tels que le nombre d'éléments logiques qui seront nécessaires pour l'implémenter ou encore les éléments standardisés disponibles. Tous cela est dû au fait que les éléments physiques implémentant les opérateurs logiques en électronique sont interchangeables, deux exemplaires de la porte ET sont identiques et utilisables à différents endroits du même circuit. Alors que dans les circuits logiques enzymatiques chaque porte logique enzymatique est unique, si on veut utiliser deux portes logiques ET à deux endroits du même circuit il faudra en prendre deux, constituées d'espèces moléculaires différentes. De plus, seules certaines versions de la porte ET pourront être utilisées dans un circuit particulier.

Les méthodes de conception automatisées de circuits logiques électroniques ne sont donc pas du tout réutilisables pour la conception de circuits logiques enzymatiques. Il est donc nécessaire de concevoir de nouvelles méthodes et de nouveaux outils plus adaptés à la conception de ce type d'implementation ; c'est ce que nous avons fait avec *NetBuild*.

1.3 Rappels Théoriques

Mes travaux sont à l'intersection de la biologie et de l'informatique. Ils ont pour but d'implémenter certains concepts fondamentaux et abstraits de l'informatique avec des composants biologiques, il est donc nécessaire de détailler plus profondément tous ces aspects.

1.3.1 Les systèmes biologiques

1.3.1.1 Les réactions enzymatiques

Nous avons utilisé pour représenter les molécules, les réactions enzymatiques et donc les réseaux métaboliques, les standards établis par SBGN (Systems Biology Graphical Notation) qui est la norme établie par la communauté (http ://www.sbgn.org/).

Les réactions enzymatiques sont l'un des processus de bases des systèmes biologiques. Elles permettent de produire beaucoup, sinon la plupart des molécules organiques. Ce sont des réactions biochimiques entre plusieurs composants organiques. On peut les décrire par une enzyme, une liste de métabolites et des paramètres cinétiques. Les métabolites sont des molécules qui seront consommées et produites par les réactions enzymatiques. Au sein dune réaction enzymatique, on distingue plusieurs types de métabolites, les éléments qui seront consommés, les substrats, et les éléments qui seront fabriqués, les produits. C'est une différence uniquement fonctionnelle et un même métabolite pourra être le produit d'une réaction et le substrat d'une autre. Il est aussi possible qu'un métabolite puisse être un inhibiteur de réaction, dans ce cas il va donc avoir la capacité de ralentir, voire d'arrêter la réaction enzymatique.

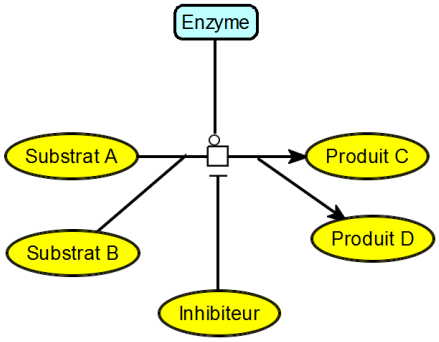


Figure 1.32 – Exemple de réaction métabolique avec une enzyme, un inhibiteur, deux substrats et deux produits

1.3.1.2 Cinétique des réactions enzymatiques

Les enzymes sont des protéines qui ont la capacité à catalyser des réactions. Sans les enzymes, les réactions seraient tellement lentes qu'à l'échelle temporelle d'un système biologique, on pourrait les considérer comme inexistantes (voir table : 1.2). De façon générale les enzymes sont des macro-molécules : elles sont plus grandes que leurs substrats et produits ; c'est leur structure qui détermine les réactions qu'elles seront capables de catalyser.

	Vitesse sans enzyme (K_n in s-1)	Vitesse avec enzyme (K_cat in s-1)	Taux d'accélération (K_catK_n)
Carbonic anhydrase	10^{-1}	10^{6}	8 x 10^{6}
Chymotrypsin	4 x 10^{-9}	4 x 10^{-2}	10^{7}
Lysozyme	3 x 10^{-9}	5 x 10^{-1}	2 x 10^{8}
Triose phosphate isomerase	4 x 10^{-6}	4 x 10^{3}	10^{9}
Fumarase	2 x 10^{-8}	2 x 10^{6}	10^{11}
b-Amylase	3 x 10^{-9}	10^{3}	3 x 10^{11}
Adenosine deaminase	2 x 10^{-10}	4 x 102	2 x 10^{12}
Urease	3 x 10^{-10}	3 x 10^{4}	10^{14}
Alkaline phosphatase	10^{-15}	10^{2}	10^{17}
Orotidine 5'-phosphate decarboxylase	3 x 10^{-16}	4 x 10	10^{17}

Table 1.2 – Exemple de taux d'accélération de réactions catalysées par des enzymes. L'orotidine 5'-phosphate decarboxylase par exemple gagne 17 ordres de grandeur, et passe d'une occurrence tous les 100 millions d'années environ, à 40 occurrences par seconde. Tableau de (http ://xray.bmc.uu.se/Courses/Tables/Tables.html)

Les deux propriétés cinétiques les plus importantes d'une enzyme sont la rapidité avec laquelle l'enzyme est saturée avec un substrat particulier et le débit maximal de fabrication du produit qu'elle peut atteindre. Il existe plusieurs mécanismes de catalyse selon la nature des enzymes et le nombre de substrats et produits impliqués dans la réaction. Toutes les réactions d'un réseau métabolique ont des cinétiques différentes et vont donc à des vitesses différentes. Nous allons voir dans le chapitre 2 que ces vitesses sont importantes pour le fonctionnement des réseaux logiques enzymatiques. Notre

but ici est d'obtenir la vitesse de réaction en fonction des paramètres cinétiques et des concentrations des substrats pour trouver comment ajuster la vitesse (en jouant sur la concentration de l'enzyme par exemple) pour obtenir des portes logiques fiables.

Réactions mono-substrats : Nous allons utiliser le modèle de cinétique enzymatique de Henri-Michaelis-Menten [47, 48, 49, 50, 51] qui est basé sur les réactions mono-substrats mais peut être étendu à plusieurs substrats :

$$\mathrm{E} + \mathrm{S} \underset{\mathrm{k_{-1}}}{\overset{\mathrm{k_1}}{\rightleftharpoons}} \mathrm{ES} \xrightarrow{\mathrm{k_{cat}}} \mathrm{E} + \mathrm{P}$$

Avec

— $[E]$ la concentration de l'enzyme E

— $[S]$ la concentration du substrat S

— $[P]$ la concentration du produit P

— k_1 la constante cinétique associé à la réaction $\mathrm{E} + \mathrm{S} \longrightarrow \mathrm{ES}$

— k_{-1} la constante cinétique associé à la réaction $\mathrm{ES} \longrightarrow \mathrm{E} + \mathrm{S}$

— k_{cat} la constante cinétique associé à la réaction $\mathrm{ES} \longrightarrow \mathrm{E} + \mathrm{P}$

la vitesse de la réaction v est exprimée par :

$$v = \frac{d[P]}{dt} = k_1[ES]$$

On suppose que nous sommes dans un état quasi-stationnaire, donc la concentration du complexe ES est stable :

$$\frac{d[ES]}{dt} = 0 = k_1[E][S] - k_{-1}[ES] - k_{cat}[ES]$$

La quantité d'enzyme reste fixe dans le temps, donc :

$$[E] + [ES] = [E]_0$$

$$v = \frac{k_{cat}[E]_0}{1 + \frac{k_{-1}+k_{cat}}{k_1[S]}}$$

On définit $V_{max} = k_{cat}[E]_0$ et $K_m = \frac{k_{-1}+k_{cat}}{k_1}$. La constante de Michaelis-Menten K_m va représenter l'affinité d'un substrat pour son enzyme, c'est la concentration de substrat qui donne la moitié de la vitesse de la réaction. Plus K_m est petit, plus l'enzyme et le substrat sont *compatibles*. Et finalement

$$v = V_{max}\frac{[S]}{[S]+K_m}$$

A partir de maintenant nous pouvons faire des hypothèses sur le comportement de v.

— Si on a $[S]$ petit devant K_m, $[S] + K_m \approx K_m$ donc $v \approx V_{max}\frac{[S]}{K_m}$ est linéaire en $[S]$. La vitesse augmente rapidement avec $[S]$.

— Si on a $[S]$ grand devant K_m, $[S] + K_m \approx [S]$ donc $v \approx V_{max}$ est maximale et n'évolue plus même quand la quantité de substrat augmente.

Si on veut optimiser la vitesse d'une réaction mono-substrat, il faut donc commencer par saturer les enzymes pour obienir $v \approx V_{max}$. Comme $V_{max} = k_{cat} \cdot [E]_0$ et que k_{cat} dépend de la nature des espèces moléculaires impliquées dans la réaction, c'est la quantité d'enzymes qui va nous permettre de moduler la vitesse des réactions enzymatiques mono-substrat.

Réactions multi-substrats : Pour des réactions multi-substrats le principe est le même mais le modèle devient plus compliqué. En effet contrairement au mono-substrat ou il n'y a qu'un seul

mécanisme pour le processus, pour les multi-substrats il en existe trois : ordonné, aléatoire et ping-pong.

Quand la formation d'un complexe enzyme-substrats ternaire est possible (modes ordonné et aléatoire), il peut y avoir deux possibilités :

(1) $\mathrm{E} + \mathrm{S}_1 + \mathrm{S}_2 \underset{\mathrm{k}_{-1}}{\overset{\mathrm{k}_1}{\rightleftharpoons}} \mathrm{ES}_1 + \mathrm{S}_2 \underset{\mathrm{k}_{-2}}{\overset{\mathrm{k}_2}{\rightleftharpoons}} \mathrm{ES}_1\mathrm{S}_2$

(2) $\mathrm{E} + \mathrm{S}_1 + \mathrm{S}_2 \underset{\mathrm{k}_{-2'}}{\overset{\mathrm{k}_{2'}}{\rightleftharpoons}} \mathrm{ES}_2 + \mathrm{S}_1 \underset{\mathrm{k}_{-1'}}{\overset{\mathrm{k}_{1'}}{\rightleftharpoons}} \mathrm{ES}_1\mathrm{S}_2$

Le mécanisme est dit ordonné si l'ordre de capture des substrats est fixé, et aléatoire sinon. Pour la séparation du complexe et la libération des produits, il peut aussi y avoir deux possibilités :

(3) $\mathrm{ES}_1\mathrm{S}_2 \xrightarrow{\mathrm{k}_{\mathrm{cat}2}} \mathrm{EP}_1 + \mathrm{P}_2 \xrightarrow{\mathrm{k}_{\mathrm{cat}1}} \mathrm{E} + \mathrm{P}_1$

(4) $\mathrm{ES}_1\mathrm{S}_2 \xrightarrow{\mathrm{k}_{\mathrm{cat}1'}} \mathrm{EP}_2 + \mathrm{P}_1 \xrightarrow{\mathrm{k}_{\mathrm{cat}2'}} \mathrm{E} + \mathrm{P}_2$

Mais en général, la libération des produits se fait de façon simultanée et un seul k_{cat} est pris en compte.

Si la formation d'un complexe ternaire n'est pas possible il y a deux possibilités :

1. $\mathrm{E} + \mathrm{S}_1 + \mathrm{S}_2 \underset{\mathrm{k}_{-1}}{\overset{\mathrm{k}_1}{\rightleftharpoons}} \mathrm{ES}_1 + \mathrm{S}_2$

   $\mathrm{ES}_1 + \mathrm{S}_2 \underset{\mathrm{k}_{-2}}{\overset{\mathrm{k}_2}{\rightleftharpoons}} \mathrm{E'P}_1 + \mathrm{S}_2$

   $\mathrm{E'P}_1 + \mathrm{S}_2 \xrightarrow{\mathrm{k}_{\mathrm{cat}1}} \mathrm{E'} + \mathrm{P}_1 + \mathrm{S}_2$

   $\mathrm{E'} + \mathrm{P}_1 + \mathrm{S}_2 \underset{\mathrm{k}_{-2}}{\overset{\mathrm{k}_2}{\rightleftharpoons}} \mathrm{E'S}_2 + \mathrm{P}_1$

   $\mathrm{E'S}_2 + \mathrm{P}_1 \xrightarrow{\mathrm{k}_{\mathrm{cat}2}} \mathrm{E} + \mathrm{P}_2 + \mathrm{P}_1$

2. $\mathrm{E} + \mathrm{S}_1 + \mathrm{S}_2 \underset{\mathrm{k}_{-1}}{\overset{\mathrm{k}_1}{\rightleftharpoons}} \mathrm{ES}_2 + \mathrm{S}_1$

   $\mathrm{ES}_2 + \mathrm{S}_1 \underset{\mathrm{k}_{-2}}{\overset{\mathrm{k}_2}{\rightleftharpoons}} \mathrm{E'P}_2 + \mathrm{S}_1$

   $\mathrm{E'P}_2 + \mathrm{S}_1 \xrightarrow{\mathrm{k}_{\mathrm{cat}1}} \mathrm{E'} + \mathrm{P}_2 + \mathrm{S}_1$

   $\mathrm{E'} + \mathrm{P}_2 + \mathrm{S}_1 \underset{\mathrm{k}_{-2}}{\overset{\mathrm{k}_2}{\rightleftharpoons}} \mathrm{E'S}_1 + \mathrm{P}_2$

   $\mathrm{E'S}_1 + \mathrm{P}_2 \xrightarrow{\mathrm{k}_{\mathrm{cat}2}} \mathrm{E} + \mathrm{P}_1 + \mathrm{P}_2$

La particularité de ces cas est d'avoir l'enzyme qui change de configuration au milieu de la réaction. L'enzyme E va commencer par fixer l'un des substrats, S_1 par exemple, le transformer en produit P1, ce qui modifiera E en E'. Ensuite E' va fixer le deuxième substrat S_2 pour le transformer en P2 et retrouver sa configuration initiale E. Dans ces deux cas, le mécanisme de catalyse est dit *ping-pong.*
La vitesse de la réaction pour ces systèmes, v est :

$v = \frac{V_{max}[S_1][S_2]}{k_1k_2+k_2[S_1]+k_1[S_2]+[S_1][S_2]}$ pour les mécanismes ordonnés et aléatoires

$v = \frac{V_{max}[S_1][S_2]}{k_2[S_1]+k_1[S_2]+[S_1][S_2]}$ pour le mécanisme ping-pong

Ces formules sont relativement similaires à celles que nous avions obtenues pour les réactions mono-substrats. Elles ont un comportement comparable quand $[S_1]$ et $[S_2]$ sont grands :

— Pour les mécanismes ordonnés et aléatoires : $k_1k_2 + k_2[S_1] + k_1[S_2] + [S_1][S_2] \approx [S_1][S_2]$ donc $v \approx V_{max}$

— Pour le mécanisme ping-pong : $k_1k_2 + k_2[S_1] + k_1[S_2] + [S_1][S_2] \approx [S_1][S_2]$ donc $v \approx V_{max}$

De même que dans les réactions mono-substrat $V_{max} = f(k_{cat1}, k_{cat2}) \cdot [E]_0$ et ici encore, si nous voulons moduler la vitesse de la réaction, il va falloir jouer sur la quantité initiale d'enzymes.

Il est également possible d'obtenir des K_m pour des réactions multi-substrats. Pour calculer K_{m1}, on considère que le milieu est saturé en S_2, on obtiendra alors une reaction similaire à une réaction mono-substrat S_1. De même, en saturant en S_1, on peut obtenir K_{m2}. Dans les bases de données de cinétiques enzymatiques (par exemple *BRENDA*) il est courant d'avoir un K_m par substrat et un K_{cat} global pour la réaction.

Réactions inhibitrices : Une réaction inhibitrice peut être obtenue par "désactivation" d'une enzyme, diminuant partiellement ou totalement certaines des réactions qu'elle catalyse. Ces inhibitions sont toujours à considérer conjointement avec les réactions qu'elles inhibent. Il y en a deux types possibles :

— **Réaction d'inhibition concurrente :** le substrat inhibiteur I, se lie à l'enzyme E pour former un complexe qui l'empêchera de fixer le substrat S.

$$\mathrm{E + S} \underset{\mathrm{k_{-1}}}{\overset{\mathrm{k_1}}{\rightleftharpoons}} \mathrm{ES} \xrightarrow{\mathrm{k_{cat}}} \mathrm{E + P}$$

$$\mathrm{E + I} \underset{\mathrm{k_{-i}}}{\overset{\mathrm{k_i}}{\rightleftharpoons}} \mathrm{EI}$$

— **Réaction d'inhibition non-concurrente :** le substrat inhibiteur I, se lie au complexe ES pour former un complexe ternaire qui l'empêchera de produire P.

$$\mathrm{E + S} \underset{\mathrm{k_{-1}}}{\overset{\mathrm{k_1}}{\rightleftharpoons}} \mathrm{ES} \xrightarrow{\mathrm{k_{cat}}} \mathrm{E + P}$$

$$\mathrm{I + ES} \underset{\mathrm{k_{-i}}}{\overset{\mathrm{k_i}}{\rightleftharpoons}} \mathrm{IES}$$

Nous voudrions pouvoir contrôler la force de l'inhibition, sa rapidité à désactiver toutes les enzymes présentes. En effet comme nous le verrons plus tard les réactions d'inhibition sont très utiles quand elles se font plus rapidement que la réaction qu'elles inhibent. La vitesse de ces réactions d'inhibition dépend de constantes cinétiques et de la concentration du substrat inhibiteur relativement à la concentration de l'enzyme.

Si $[C_{inhibiteur}] >> [C_{enzyme}]$ alors l'inhibition sera forte, et la réaction $S \longrightarrow P$ sera quasiment stoppée. Si $[C_{inhibiteur}] << [C_{enzyme}]$ alors l'inhibition sera tres faible, voire inexistante. Dans les cas intermédiaires la réaction ne sera au mieux que ralentie.

Simulation *fine* des réseaux métaboliques : Pour obtenir des profils de réactions réalistes, nous allons utiliser l'automate stochastique HSIM developpé au laboratoire [52, 53], pour simuler le comportement des réseaux enzymatiques qui implémenteront nos portes et circuits logiques. Pour les modèles *concrets*, constitués d'espèces moléculaires réelles, nous utiliserons des constantes cinétiques tirées de la littérature ou de bases de données spécifiques. Pour les réseaux métaboliques *abstraits*, nous allons utiliser trois réactions réelles qui nous permettrons de construire trois modèles cinétiques différents. Ces trois réactions comprendrons un réaction dite lente (voir fig. 1.35), une dite rapide (voir fig. 1.34) et une qui sera dans la moyenne (voir fig. 1.33). Nous allons les adapter pour qu'elles puissent toutes être a l'origine d'une réaction avec deux substrats et deux produits. Ensuite ces modeles seront convertis en modèle HSIM utilisable pour les simulations.

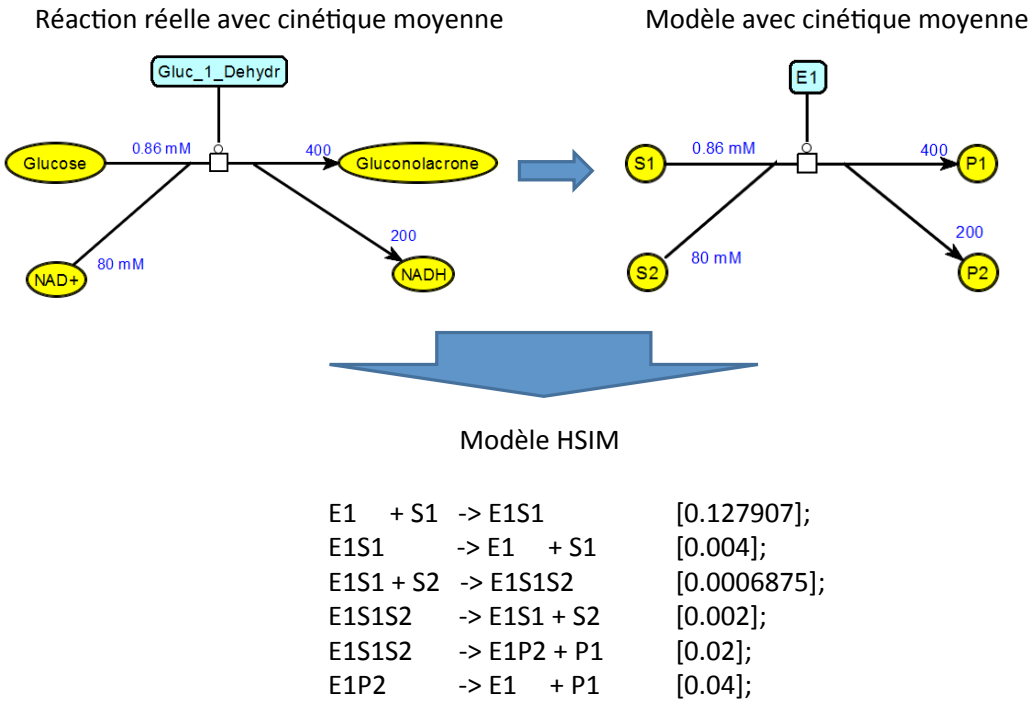


Figure 1.33 – Construction d'une modèle HSIM de cinétique moyenne pour une réaction réelle

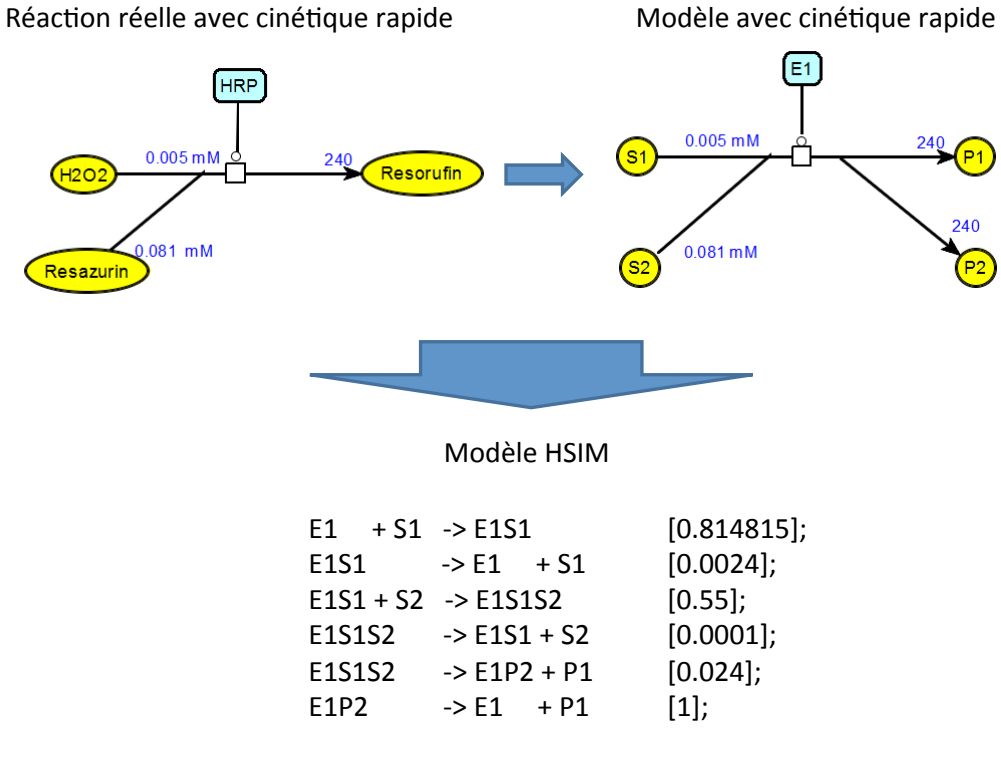


Figure 1.34 – Construction d'un modèle HSIM de cinétique rapide pour une réaction réelle

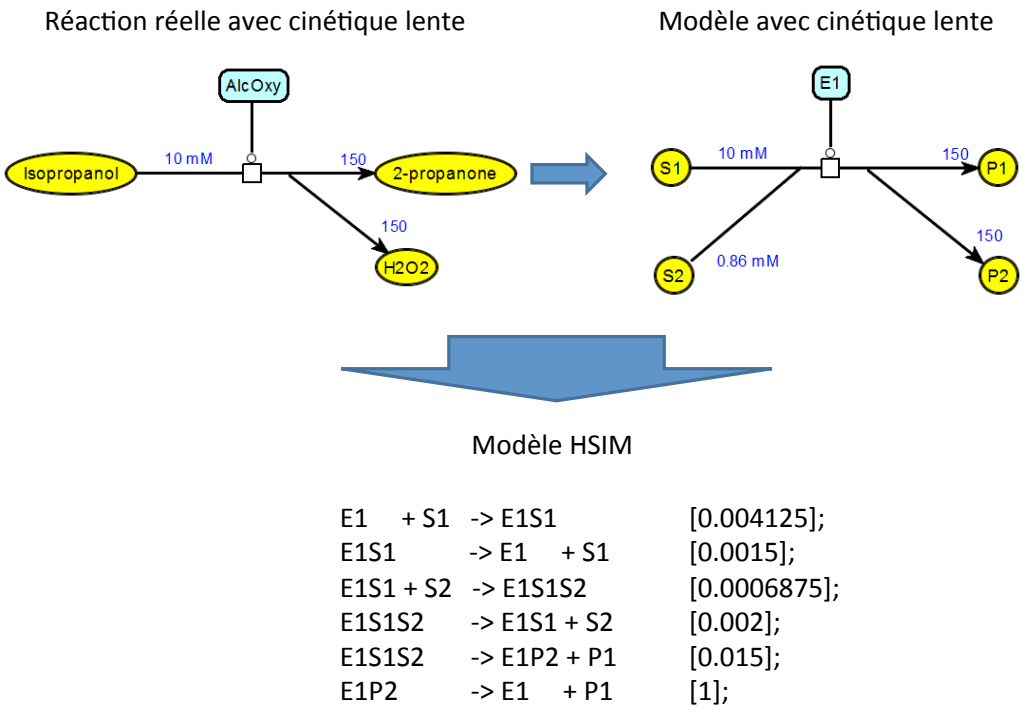


Figure 1.35 – Construction d'un modèle HSIM de cinétique lente pour une réaction réelle

1.3.1.3 Les réseaux métaboliques

Les réactions enzymatiques sont organisées au sein des systèmes biologiques en réseaux appelés réseaux métaboliques. Ces réactions enzymatiques fonctionnent ensemble car elles sont liées les unes aux autres par des composants communs. Elles peuvent l'être de plusieurs façons, chacune ayant des conséquences différentes :

— elles partagent la même enzyme
— elles partagent les mêmes produits
— elles partagent les mêmes substrats
— des substrats de l'une sont des produits de l'autre
— des substrats de l'une sont des inhibiteurs de l'autre.

1.3.2 Logique booléenne

La logique booléenne est un concept tirée de l'algèbre permettant d'effectuer du calcul sur des informations binaires. Elle est communément appelée Algèbre de Boole, et existe depuis le XIX^{e} siècle, elle a été utilisée dans de nombreux domaines. C'est notamment la base de nos ordinateurs, tous les systèmes informatiques actuels fonctionnent sur ce principe. Cette algèbre est basée sur une représentation binaire de l'information. Chaque donnée ne peut exister que dans deux états VRAI et FAUX, représentés respectivement par 1 et 0.

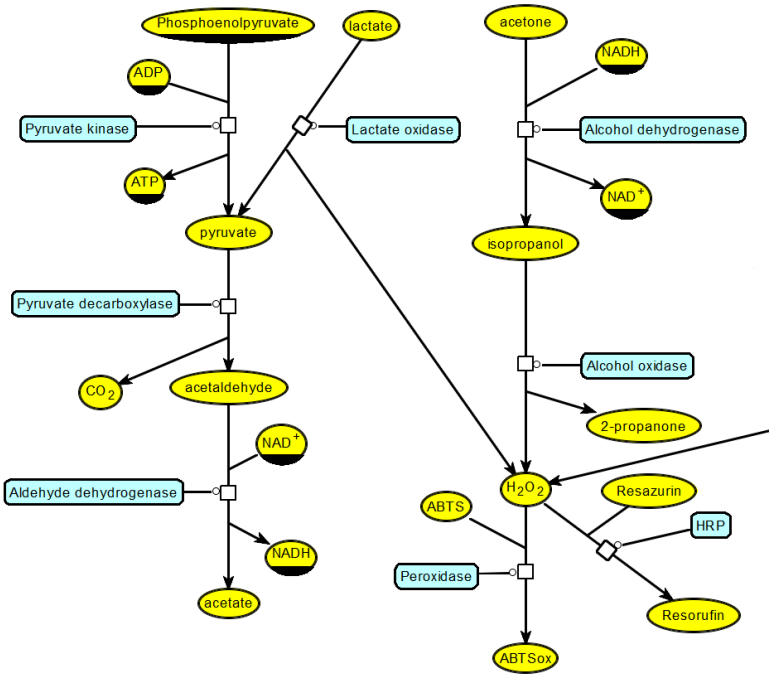


Figure 1.36 – Exemple de réseau métabolique composé de 8 réactions enzymatiques. Ce réseau est lui-même un sous-réseau d'un plus grand réseau métabolique

1.3.2.1 Opérateurs logiques booléens

Les opérateurs logiques sont des constructions abstraites manipulant des informations booléennes et effectuant des opérations sur ces informations. Un opérateur possède des entrées qui représentent les données qui sont utilisés par l'opérateur et des sorties qui représentent les données qui sont produites par l'opérateur. Un opérateur est défini par une fonction qui représente le type de traitement qu'il effectue sur ses entrées, ainsi que son nombre d'entrées. Ces informations seront présentées sous forme de table de vérité qui décrira la valeur de sortie de l'opérateur pour toutes les combinaisons de valeurs des entrées.

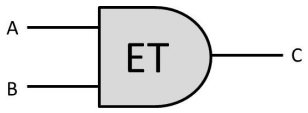


Figure 1.37 – Exemple d'un opérateur logique à 2 entrées

Entrée A	Entrée B	Sortie C
false	false	false
false	true	false
true	false	false
true	true	true

Table 1.3 – Exemple de table de vérité de l'opérateur ET à 2 entrées

Type d'opérateurs logiques booléens Il peut y avoir autant d'opérateurs différents que de tables de vérité différentes. Comme le nombre d'entrées d'un opérateur logique n'est pas limité, il y a un nombre illimité d'opérateurs possibles. Par exemple, pour les opérateurs logiques à deux entrées, la table de vérité comporte quatre lignes (2 variables, chacune pouvant avoir 2 valeurs), les quatre valeurs binaires de sortie possibles donnent donc une combinatoire de 2^4 tables de vérité différentes. Nous allons nous restreindre ici à décrire les opérateurs qui nous intéresseront dans la suite de nos travaux.

Opérateurs unaires Les opérateurs unaires que nous utiliserons sont le NON et le OUI, qui inversent ou pas leur entree. Ce sont les deux seuls opérateurs unaires possibles.

Opérateurs binaires : Les opérateurs binaires que nous allons utiliser sont les ET, OU, NON-ET, ET-NON, NON-OU, OU-NON, NAND et NOR.

Opérateurs ternaires et plus : Nous n'utiliserons ici que peu d'opérateurs ternaires, seulement le ET et le OU.

Équation logique Une équation logique est simplement la combinaison d'opérateurs logiques et de variables. Elle peut aussi être représentée par une table de vérité. Chaque équation logique n'a qu'une seule table de vérité, mais une table de vérité peut correspondre à plusieurs équations logiques. Une équation logique peut subir des transformations qui altéreront sa forme mais pas son sens.

1.3.2.2 Portes logiques

Les opérateurs sont des concepts abstraits, d'un certain point de vue ils n'existent que théoriquement. Mais des systèmes physiques peuvent être utilisés pour implémenter les fonctions décrites par ces opérateurs. Ce sont les *portes* logiques, des dispositifs physiques qui implémentent des opérateurs logiques abstraits. De même que l'opérateur qui lui correspond, une porte logique est caractérisée par sa sortie, ses entrées, et sa fonction logique.

1.3.2.3 Circuits

Pour construire des fonctions logiques sophistiquées, on peut connecter entre elles plusieurs portes logiques simples. Ce chaînage des portes permet la formation de circuits logiques, qui pourront avoir un nombre important d'entrées et de sorties et qui surtout seront capables de traiter des fonctions logiques complexes.

Bien sûr, comme nous l'avons vu précédemment, comme il possible de concevoir des opérateurs logiques de n'importe quelle taille, il serait *a priori* tout aussi possible de construire des *super* portes implémentant directement la fonction voulue. Mais la conception *efficace* de circuits logiques obéit à des lois différentes de celles de la conception des portes logiques simples. De la même façon que pour construire une maison, on fabrique d'abord des briques, que l'on assemblera ensuite pour faire la maison, nous ne construisons pas une unique brique en forme de maison ! Il est préférable (*i.e.* plus efficace, plus versatile, moins coûteux et plus reproductible) pour concevoir un système logique complexe de le faire par assemblage raisonné de portes logiques simples.

Chapitre 2

Théorie des réseaux logiques enzymatiques

Il convient d'expliciter ici les concepts de "modèle" et de "système réel". Les modèles sont des concepts abstraits qui vont nous permettre de raisonner. Ils sont basés sur un cadre théorique qui peut être l'algèbre ou tout autre formalisme. Ils ont leur propres règles et contraintes qui régissent leur fonctionnement. L'information numérique au sein des modèles peut être représentée sous différentes formes : continue, discrète, multi-valuée ou booléenne.

Il est commun de penser qu'on ne peut comprendre un système réel qu'à travers le prisme d'un modèle. Toutes les informations que nous avons sur un système réel proviennent de relevés que nous avons effectués à l'aide d'appareils de mesure. Ce sont ces informations qui vont nous permettre d'élaborer un modèle, puis de le confronter à la réalité. Un système réel donné peut être représenté avec plusieurs modèles, chacun par exemple modélisant un aspect particulier du système réel.

Réciproquement, un modèle peut être aussi construit de façon purement abstraite, n'ayant pas vocation à représenter un aspect de la réalité, mais à établir un cadre purement théorique (par exemple la *machine de Turing* en informatique [REF article d'Alan Turing de 1936]). Ce type de modèle abstrait peut avoir plusieurs implémentations possibles. Une implémentation sera un système physique dont les propriétés satisferont celles du modèle.

Les systèmes biologiques, et plus spécifiquement les réseaux métaboliques, sont par nature des systèmes où l'information peux être interprétée sous plusieurs formes différentes. En effet, dans un réseau métabolique, l'information peut être représentée par les flux de molécules évoluant dans le système. On pourrait donc penser que l'information est continue : le rapport du nombre de copies de chaque espèce moléculaire au volume du compartiment, la *concentration* moléculaire, est l'information pertinente. C'est par exemple la représentation qui est choisie dans les modélisations par équations différentielles ordinaires. Cependant, on peut aussi regarder ces flux comme de la matière qui diffuse et évolue, et considérer directement le nombre de copies de chaque espèce moléculaire ; dans ce cas, l'information est discrète et dénombrable. C'est la représentation choisie dans les systèmes de modélisation stochastique globaux (méthodes de *Monte Carlo*) ou dans les systèmes *entité-centrées*, qui ajoutent la dimension spatiale à l'information (voir figure : 2.1).

Pour notre approche, nous allons utiliser deux niveaux d'information. Nous considérerons les quantités moléculaires sous forme discrète. Les nombres effectifs de molécules ne seront pas gigantesques car dans les cas qui nous intéressent le volume du compartiment est très faible (vésicules lipidiques de quelques centaines de nanomètres de diamètre) et les concentrations seront donc malgré tout assez fortes. Ensuite, à ce premier modèle par quantité moléculaire, nous allons superposer une abstraction de portes logiques. C'est l'interaction entre ces deux niveaux d'information qui va nous permettre de

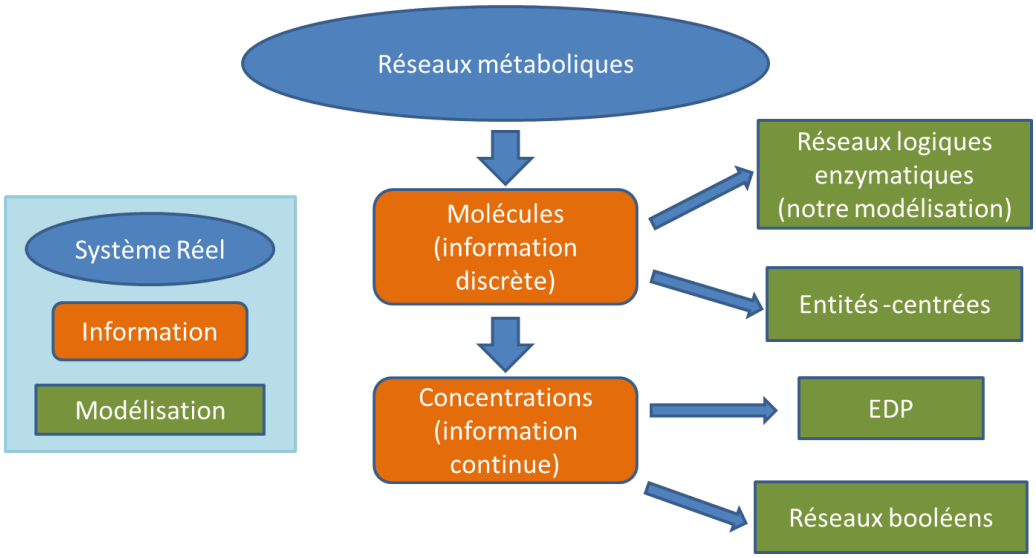


Figure 2.1 – Organisation de plusieurs modèles et types d'informations relatives aux réseaux métaboliques

réaliser des portes logiques à partir de réseaux métaboliques.

Nous voulons donc construire une implémentation d'un système booléen (abstrait) à l'aide de réactions enzymatiques (concrètes). On va utiliser comme modèle intermédiaire un modèle de réactions biochimiques par quantité moléculaire pour implémenter un modèle de calcul booléen.

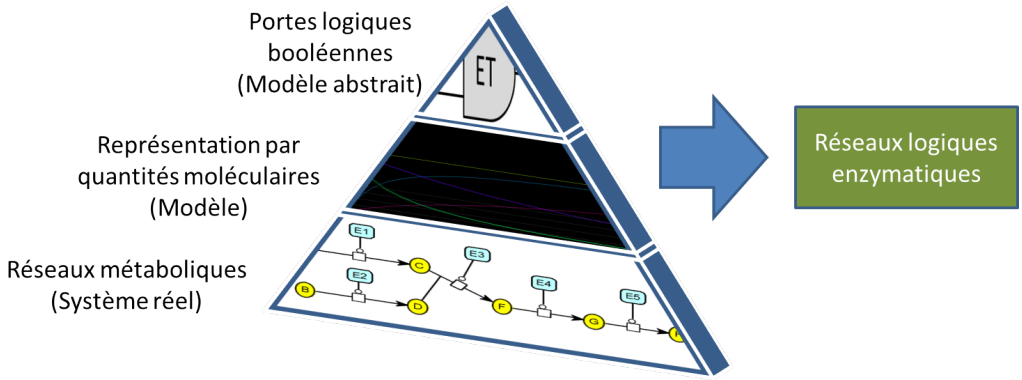


Figure 2.2 – Hiérarchie de modèles pour la création de réseaux logiques enzymatiques

Nous allons avoir deux problèmes à résoudre : i) le modèle booléen requiert plusieurs flux d'informations, chacun étant binaire et ii) chacun devant être *isolé* des autres. Le modèle de notre système réel dispose de flux d'information discrets, correspondant à la quantité de molécules d'une espèce moléculaire, et tous ces flux d'information sont mélangés dans l'espace (tout est dans le même compartiment).

2.1 Codage *Monophase*

La première façon de coder des informations booléennes avec des flux moléculaires sera de représenter une connexion par une espèce moléculaire particulière, et de représenter l'information booléenne par la présence (*vrai*) ou l'absence (*faux*) de cette espèce moléculaire. Nous avons appelé ce codage *monophase*

	Réactions enzymatiques	Algèbre de Boole
Représentation de l'information	Discrète (nombre de molécules)	Discrète (binaire)
Distribution de l'information	Répartition homogène des molécules, flux d'information mélangés	Enchaînement explicite
Opérateurs	Transformation d'espèces moléculaires	ET, OU, NOT, etc.

Table 2.1 – Comparaison des modèles

en ce sens qu'une seule espèce moléculaire est nécessaire pour coder une valeur booléenne. Nous verrons par la suite qu'on peut utiliser un autre codage que nous appellerons *Diphase différentiel*.

2.1.1 Représentation des valeurs booléennes

Nous allons donc représenter les constantes booléennes par la quantité de molécules du flux correspondant. L'idéal serait respectivement de prendre *zéro* molécule pour *faux* et *quantité maximum* pour *vrai*. En réalité, il n'y a jamais zéro copie d'une espèce moléculaire mais plutôt une *faible* quantité, et jamais non plus un nombre gigantesque de copies, mais plutôt une *forte* quantité, qui peut d'ailleurs être très variable selon les espèces moléculaires et les réactions.

Pour obtenir un signal binaire, nous avons utilisé un seuil de quantité de molécules pour définir les intervalles de valeurs associés aux états *faux* et *vrai* de l'information moléculaire. Si le nombre de molécules de l'espèce correspondant au flux d'information est inférieur au seuil, alors ce flux d'information sera interprété comme la constante booléenne *faux*. Dans le cas contraire, le flux sera interprété comme représentant *vrai*.

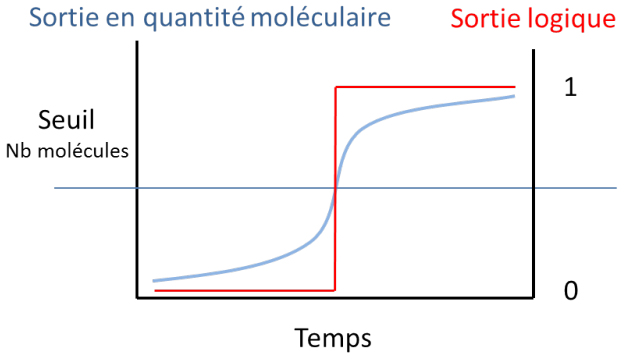


Figure 2.3 – État logique en fonction de la quantité de molécules et du seuil de séparation d'états.

2.1.2 Portes logiques enzymatiques

2.1.3 Portes simples

Une porte logique enzymatique est une porte logique, c'est à dire une implémentation par un système physique d'un opérateur logique, le système physique étant constitué de réactions enzymatiques. Nous avons défini précédemment un codage pour l'information booléenne (niveau de quantité moléculaires), nous allons voir maintenant comment implémenter des opérateurs booléens simples.

A l'aide des lois de distributivité des opérateurs ET et OU et des lois de *De Morgan* les liant à l'opérateur NON, il a été montré qu'une expression booléenne quelconque peut être transformée en une expression de même valeur n'utilisant que les opérateurs ET et NON, ou bien les opérateurs OU et NON. Les formes normales conjonctives et disjonctives (uniques pour une expression donnée) n'utilisent que les operateurs ET, OU et NON. Ce sont donc les portes enzymatiques correspondant à ces trois opérateurs que nous avons essayé d'implémenter en premier.

Chaque flux d'information (fil de connexion) dans un réseau logique enzymatique est représenté par une espèce moléculaire unique ; toutes les molécules du compartiment sont réparties de façon quasi homogène dans l'espace. Chaque porte utilisera en interne plusieurs espèces moléculaires pour coder ses flux d'information (ses connexions internes), donc la réutilisation de la même implémentation d'une porte logique dans un réseau complexe est quasiment impossible. L'usage de plusieurs portes qui utilisent une même espèce moléculaires pour transmettre des flux d'information différents créerait un *court-circuit*. De même, pour faire un réseau logique enzymatique complexe, il va falloir raccorder ensemble différentes portes logiques enzymatiques donc cette construction est fortement contrainte par la possibilité unique de chaque *fil* de câblage.

Comme chaque porte est à usage unique (dans un même circuit), on aura tout intérêt à avoir en stock le plus grand nombre possible de portes logiques enzymatiques implémentant les mêmes opérateurs booléens, mais avec des espèces moléculaires différentes. De plus, dans un souci de robustesse, nous allons analyser l'influence des constantes cinétiques des réactions qui composent chacune de ces portes dans la réalisation de la fonction booléenne associée.

Les modèles d'implémentations que nous allons présenter sont parmi les plus simples que nous puissions construire. Ce sont des *modèles* d'implémentation et non pas des implémentations *réelles*. Donc tous les sous-réseaux métaboliques qui respecteront les contraintes topologiques, ainsi que les éventuelles contraintes cinétiques d'un des modèles d'implémentation, seront éligibles pour construire une implémentation réelle. De plus ce sont des modèles simples, destinés à présenter des exemples et les grandes catégories de problématiques associées aux réseaux logiques enzymatiques.

Enfin, nous prendrons les mêmes seuils de séparation pour les entrées et pour les sorties. C'est-à-dire que la quantité de molécules d'une entrée déterminant un 1 logique sera aussi la quantité utilisée pour déterminer l'état de la sortie de la porte logique enzymatique.

2.1.3.1 Contraintes cinétiques

L'influence de la cinétique des réactions est un point important pour l'implémentation d'une porte logique enzymatique. En effet, plus une porte est dépendante de la cinétique d'une réaction particulière, moins elle sera considérée comme robuste. Cette recherche de robustesse a deux raisons : la première est que les modèles d'implémentation que nous allons présenter sont composés de réactions types avec des composés moléculaires purement abstraits, ils sont conçus pour être implémentables en pratique par un maximum d'enzymes et de métabolites réels. La deuxième raison tient au fait que même pour une implémentation d'une porte logique enzymatique réelle (avec des molécules existantes), les constantes cinétiques peuvent varier fortement (dépendance au milieu, provenance des enzymes, etc.). Nous allons donc pour chaque modèle d'implémentation tester trois cinétiques différentes correspondants à une réaction qui se fera à une vitesse que l'on qualifiera de *lente*, *moyenne*, ou *rapide* :

	Cinétique lente	Cinétique moyenne	Cinétique rapide
Km_1	80 mM	10 mM	5 μM
Km_2	0.86 mM	0.86 mM	81 μM
$Kcat_1$	150 s^-1	200 s^-1	240 s^-1
$Kcat_2$		400 s^-1	

Définir des modèles d'implémentations robustes est donc un point très important : il permet d'étendre leur utilisation à de nombreux réseaux métaboliques différents, et donc d'augmenter le nombre d'implémentations potentiellement valides que l'on peut trouver.

2.1.3.2 Porte logique enzymatique ET

Nous allons commencer par détailler l'implémentation d'une porte ET à deux d'entrées A et B, et une sortie C. La seule réaction utilisée est :

$$A + B \xrightarrow{E} C$$

Cette réaction où l'enzyme E catalyse la réaction qui produit C à partir de A et de B répond correctement aux besoins de la porte logique, C ne pourra être obtenu que si l'on a bien A et B présents. Construire cette porte ne va nécessiter que 3 types de métabolites et une seule enzyme.

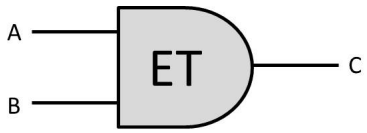


Figure 2.4 – Représentation symbolique standard d'une porte ET à deux entrées.

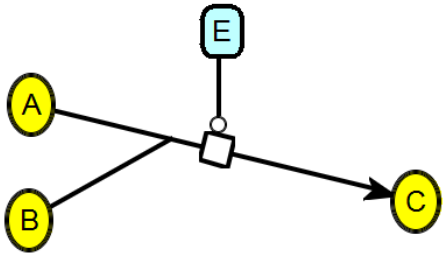


Figure 2.5 – Modèle d'implementation d'une porte ET avec une réaction enzymatique : $C = A \wedge B$.

Analyse cinétique :

Ce modèle d'implémentation de porte ET est très robuste par rapport à la cinétique. En effet quelle que soit la cinétique de la réaction, la fonction booléenne ET sera respectée. Il sera toujours nécessaire de consommer une molécule de A et une molécule de B pour obtenir une molécule de C. La cinétique de la réaction n'affectera que la rapidité du *calcul* effectué par la porte logique (c'est un facteur qu'on considérera sans importance ici). Nous avons utilisé trois différents jeux de constantes cinétiques (lente, moyenne et rapide) pour simuler avec HSIM le calcul de la sortie *vrai* de la porte (état logique A = *vrai* et B = *vrai*). Ces simulations ont été effectuées en mettant pour chaque signal d'entrée 10000 copies de l'espèce moléculaire associée ; il y a 300 copies de l'enzyme. La réponse tend asymptotiquement vers 10000 copies du métabolite de sortie et donc le signal est de très bonne qualité. On peut voir que la cinétique change uniquement la vitesse de traitement de la porte logique et non sa réponse booléenne. Les courbes pour les trois cinétiques ont des formes similaires, plus la cinétique est rapide, plus les courbes sont compactes (voir fig. 2.6).

Nous avons ensuite effectué des simulations HSIM pour toutes les valuations booléennes des entrées de la porte logique enzymatique ET, avec les trois cinétiques (voir table 2.2). Pour chaque cinétique de réaction, nous avons attribué un critère de robustesse qui représente la confiance dans les capacités

de la porte à assurer sa fonction de façon correcte. Pour la portes ET, nous pouvons voir que le niveau de la réponse est toujours excellent et donc quelle que soit la cinétique, la porte ET est considérée comme très robuste.

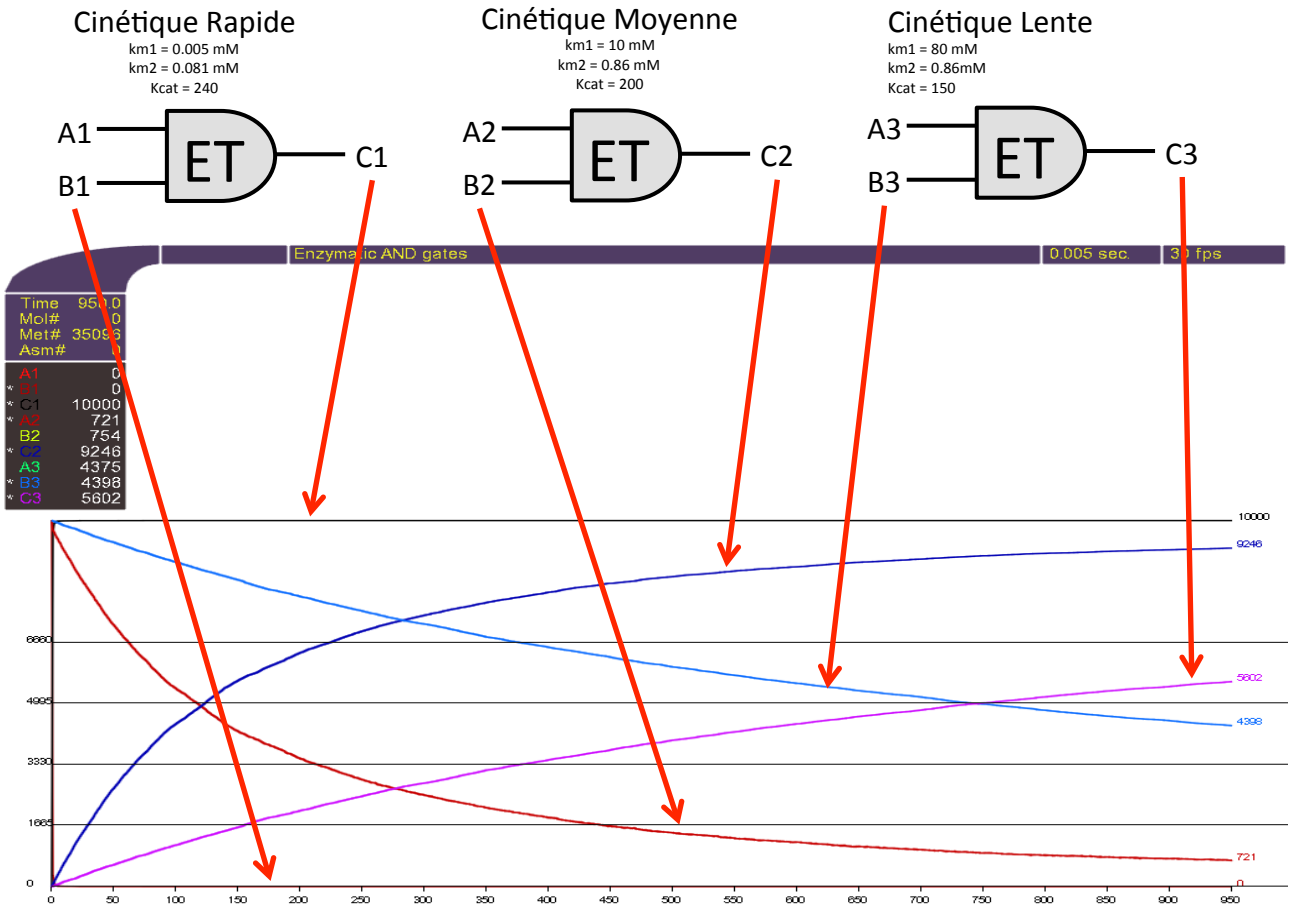


Figure 2.6 – Influence de la variation de cinétique enzymatique du modèle d'implémentation de porte logique ET. Trois cinétiques différentes ont été testées avec HSIM : lente, moyenne et rapide.

On peut noter que dans les cas où l'entrée B est fausse et l'entrée A est vraie, cette dernière se retrouve avec 9700 copies à la fin de la simulation et non 10000, parce que les 300 enzymes ont fixé 300 copies de A. Alors que dans le cas contraire, A faux et B vrai, les enzymes n'ayant pas pu fixer le premier substrat A, n'ont pas non plus fixé le deuxième substat B, qui est resté à 10000 copies (mécanisme de réaction *ordered*).

Cinétique de réaction	Quantités initiales		Quantités moléculaires finales			Niveau réponse	Robustesse
	Entrée A	Entrée B	Valeur A	Valeur B	Valeur C		
Moyenne	0	0	0	0	0	100%	très bonne
	0	10000	0	10000	0	100%	
	10000	0	9700	0	0	100%	
	10000	10000	0	0	10000	100%	
Rapide	0	0	0	0	0	100%	très bonne
	0	10000	0	10000	0	100%	
	10000	0	9700	0	0	100%	
	10000	10000	0	0	10000	100%	
Lente	0	0	0	0	0	100%	très bonne
	0	10000	0	10000	0	100%	
	10000	0	9700	0	0	100%	
	10000	10000	0	0	10000	100%	

Table 2.2 – Résultat des simulations HSIM de la porte ET pour les trois cinétiques.

2.1.3.3 Porte logique enzymatique OU

Nous allons maintenant détailler l'implémentation d'une porte OU à deux d'entrées A et B, et une sortie C. Les réactions utilisées sont :

$$\begin{array}{ccc} A & \xrightarrow{E_1} & C \\ B & \xrightarrow{E_2} & C \end{array}$$

Il va falloir soit deux enzymes, soit une enzyme capable de catalyser les deux réactions. Si les quantités de A ou de B sont supérieures au seuil prédéfini qui sépare la valeur booléenne *faux* de *vrai*, alors C sera aussi supérieur à ce seuil et représentera bien la valeur booléenne *vrai*.

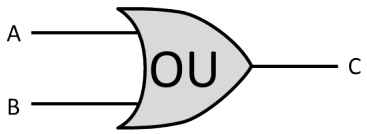


Figure 2.7 – Représentation d'une porte OU avec un symbole logique.

Analyse cinétique :
De même que pour la porte ET, ce modèle d'implémentation de porte OU est très robuste vis-à-vis de la cinétique. Quelle que soit la cinétique des réactions, la fonction booléenne sera correctement calculée. On obtient une molécule de sortie C à chaque fois que sera consommée une molécule de A ou bien une molécule de B.

Les cinétiques des réactions n'influeront que sur la vitesse de traitement de la porte logique. Nous avons fait les mêmes types de simulation que pour la porte ET, dans les mêmes conditions, et les résultats se sont avérés aussi bons (voir table 2.3).

2.1.4 Portes inverseuses

Ce type de porte doit donner un 1 logique lorsque son entrée est à 0 et un 0 logique quand son entrée est à 1. Nous devons donc transformer la présence de métabolites en une absence de métabolite, et surtout, une absence de métabolite en une présence de métabolites. Nous avons contourné ce problème

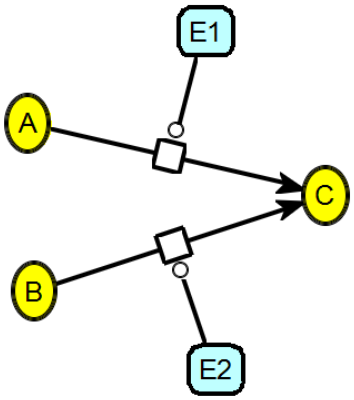


Figure 2.8 – Implémentation d'une porte OU avec deux réactions enzymatiques : $C = A \vee B$.

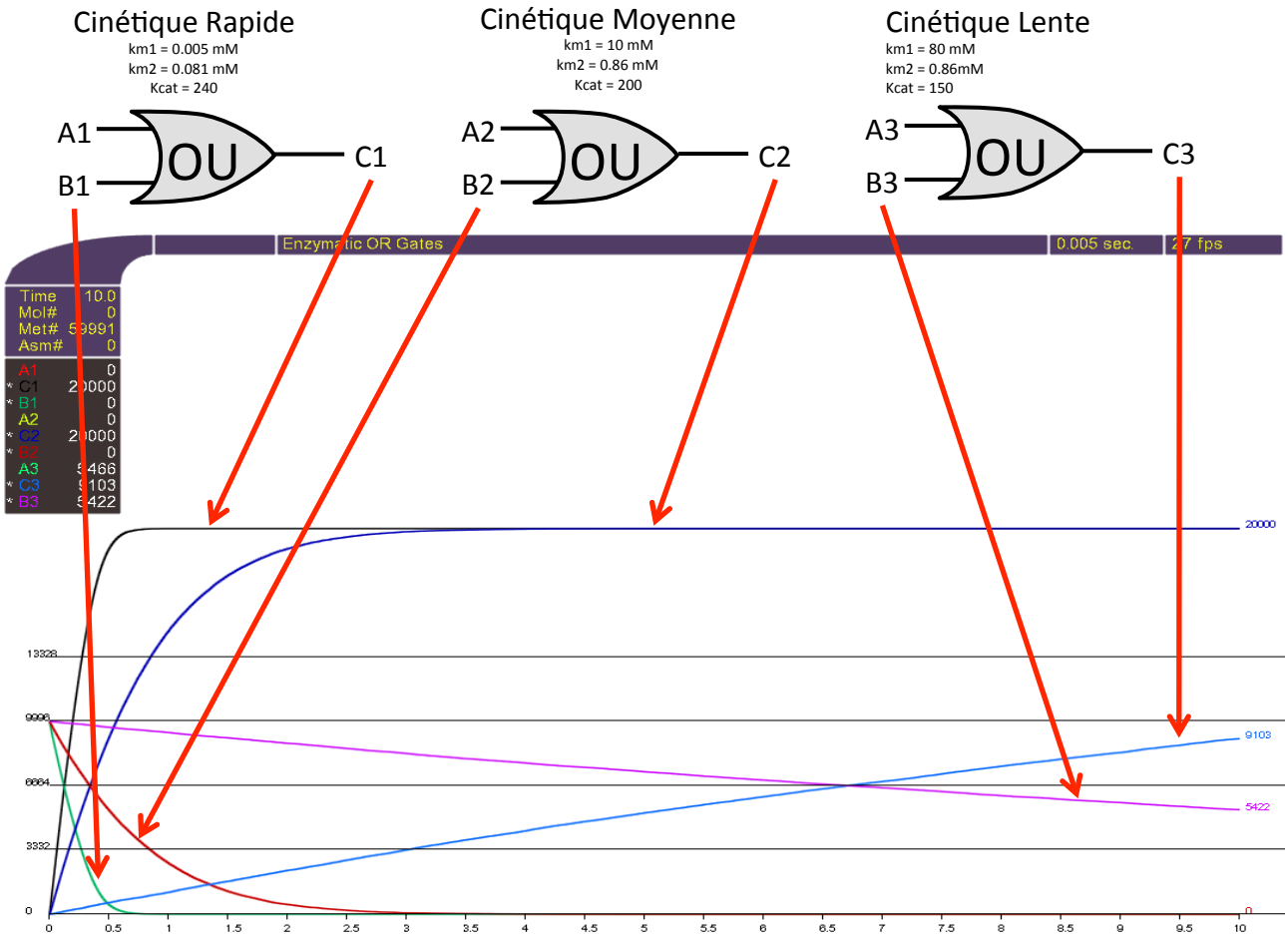


Figure 2.9 – Influence des variations de constante cinétique sur le modèle d'implémentation de porte logique OU. Trois cinétiques différentes ont été testées avec HSIM : lente, moyenne et rapide.

de création de métabolites en utilisant une autre espèce moléculaire comme ressource (qui devra donc

Cinétique de réaction	Quantités initiales		Quantités moléculaires finales			Niveau réponse	Robustesse
	Entrée A	Entrée B	Valeur A	Valeur B	Valeur C		
Moyenne	0	0	0	0	0	100%	très bonne
	0	10000	0	0	10000	100%	
	10000	0	0	0	10000	100%	
	10000	10000	0	0	20000	100%	
Rapide	0	0	0	0	0	100%	très bonne
	0	10000	0	0	10000	100%	
	10000	0	0	0	10000	100%	
	10000	10000	0	0	20000	100%	
Lente	0	0	0	0	0	100%	très bonne
	0	10000	0	0	10000	100%	
	10000	0	0	0	10000	100%	
	10000	10000	0	0	20000	100%	

Table 2.3 – Résultat des simulations HSIM de la porte OU pour les trois cinétiques.

être en abondance initialement) et selon la valeur de l'entrée déclencher ou pas une réaction qui transforme cette ressource dans le métabolite de sortie. On va définir les portes à *action négative* (ou *action inhibitrice*) comme étant des portes nécessitant qu'une de leurs entrées soit à 0 logique pour produire une sortie à 1 logique. L'inverseur (porte NON) est la plus simple des portes à action négative, mais le XOR, le NAND et le NOR sont aussi de ce type.

Prenons la porte NON-ET comme exemple. Nous avons une entrée (B) qui alimente la sortie, et l'autre entrée (A) qui peut la bloquer. La seule combinaison qui donne une sortie au niveau 1 logique est A = 0 et B = 1.

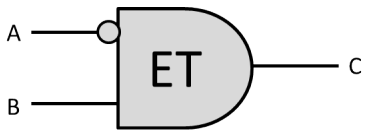


Figure 2.10 – Représentation dune porte NON-ET avec un symbole logique

Entrée A	Entrée B	Sortie C
faux	faux	faux
faux	vrai	vrai
vrai	faux	faux
vrai	vrai	faux

Table 2.4 – Table de vérité de l'opérateur NON-ET

2.1.4.1 Inversion par inhibition

Plusieurs modèles d'implémentations simples sont possibles et nous allons tous les présenter. Nous avons commencé par le plus simple, qui utilise une réaction pouvant être inhibée par un métabolite. Nous avons donc une unique réaction où B est consommé pour produire C, cette réaction pouvant être inhibée par A.

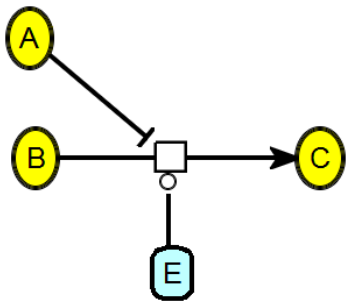


Figure 2.11 – Modèle d'implementation possible d'une porte NON-ET avec une réaction enzymatique inhibée.

Cette réaction inhibée cache en fait une autre réaction qui réalise l'inhibition par séquestration de l'enzyme sous une forme non active :

$$\mathrm{B} \xrightarrow{\mathrm{E}} \mathrm{C}$$
$$\mathrm{A} + \mathrm{E} \rightleftharpoons \mathrm{AE}$$

Analyse cinétique :
Ce modèle d'implémentation d'une porte NON-ET est moyennement robuste vis-à-vis de la cinétique des réactions. En effet, pour que l'inhibition soit efficace, il est nécessaire que la réaction d'inhibition se fasse bien plus rapidement que la réaction de production de C. C'est une contrainte forte, mais pas rédhibitoire, sur les vitesses relatives des deux réactions.

2.1.4.2 Inversion par concurrence

La deuxième façon d'implémenter une porte inverseuse fait intervenir un phénomène de concurrence entre deux réactions. La première réaction va consommer B pour produire C, et la deuxième va consommer A et aussi B pour produire P. Le métabolite P est un produit résiduel non utilisé.

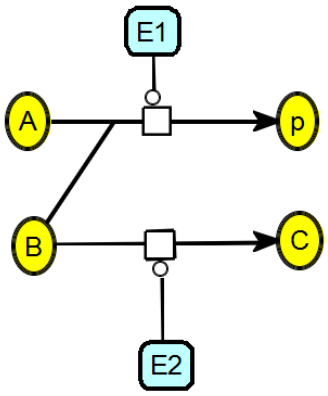


Figure 2.12 – Modèle d'implementation d'une porte NON-ET avec deux réactions enzymatiques concurrentes

L'action négative est réalisée par la consommation concurrente du produit B par la réaction qui utilise

l'entrée inhibitrice A :

$$\begin{aligned} \text{R1} : \quad & \text{B} \xrightarrow{\text{E}_2} \text{C} \\ \text{R2} : \text{A} + \text{B} & \xrightarrow{\text{E}_1} \text{P} \end{aligned}$$

Analyse cinétique :
Cette autre implémentation de la porte NON-ET est peu robuste vis-à-vis de la cinétique des réactions. En effet, il est critique que la réaction R2 se fasse bien plus rapidement que la réaction R1. Nous avons effectué des simulations de cette implémentation concurrente de la porte NON-ET en utilisant toutes les combinaisons des trois gammes de cinétiques pour chacune des deux réactions (voir table 2.5).

On peut constater que dans de nombreux cas la porte ne fonctionne pas correctement, et qu'il est nécessaire d'avoir des cinétiques bien particulières pour avoir la réponse correcte. La meilleure possibilité est la version de la porte où la réaction R1 est lente et la réaction R2 rapide, ce qui confirme nos intuitions. Il est possible d'améliorer ces résultats en ajustant les quantités d'enzymes pour différencier encore plus les vitesses des deux réactions. Néanmoins, cela ne transformera pas les versions ayant une robustesse *nulle* en porte à robustesse *très bonne*. Par contre cela peut améliorer les portes à robustesse *bonne* ou *moyenne* en les rendant *très bonne*.

Nous avons pu voir que les modèles d'implémentation des portes inverseuses sont systématiquement dépendants de la cinétique des réactions. Le modèle par inhibition est plus robuste mais nécessite que le métabolite utilisé pour l'entrée inversée ait la capacité d'inhiber la réaction principale ; la probabilité de trouver un jeu de molécules réelles ayant ces caractéristiques est relativement restreinte. Le modèle par concurrence peut être implémenté par un grand nombre de sous-réseaux différents car il est topologiquement assez commun, mais il est plus fortement contraint par la cinétique des réactions et donc nécessitera très probablement un ajustement des quantités d'enzymes.

2.1.5 Portes complexes

Nous avons présenté des modèles simples pour des implémentations possibles de portes logiques enzymatiques. Il n'y a pas vraiment de limite au nombre et à la taille des modèles possibles pour une implémentation. Néanmoins les implémentations plus complexes sont en général des combinaisons des modèles montrés précédemment (voir figure 2.13).

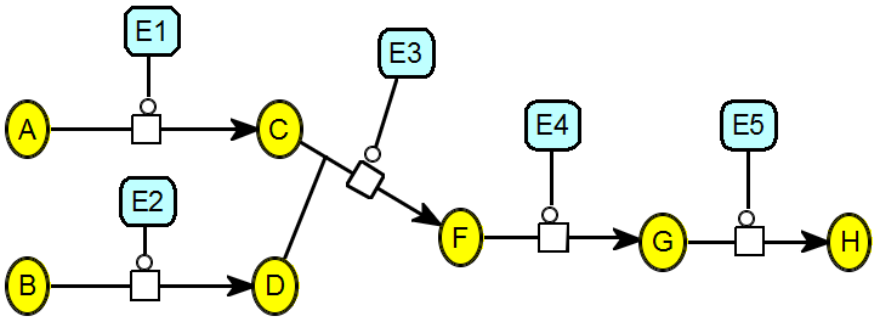


Figure 2.13 – Modèle d'implémentation d'une porte ET composé d'un modèle de porte ET simple et de portes OUI changeant les types de métabolites d'entrée et de sortie

Cinétique réaction		Quantités initiales		Quantités moléculaires finales			Niveau	Robustesse
Réaction 1	Réaction 2	A	B	Valeur A	Valeur B	Valeur C	réponse	
Moyenne	Moyenne	0	0	0	0	0	100%	nulle
		0	10000	0	0	10000	100%	
		10000	0	10000	0	0	100%	
		10000	10000	9936	0	9936	0,6%	
Moyenne	Rapide	0	0	0	0	0	100%	bonne
		0	10000	0	0	10000	100%	
		10000	0	10000	0	0	100%	
		10000	10000	2499	0	2499	75,0%	
Moyenne	Lente	0	0	0	0	0	100%	nulle
		0	10000	0	0	10000	100%	
		10000	0	10000	0	0	100%	
		10000	10000	9994	0	9994	0,1%	
Rapide	Moyenne	0	0	0	0	0	100%	nulle
		0	10000	0	0	10000	100%	
		10000	0	10000	0	0	100%	
		10000	10000	9988	0	9988	0,1%	
Rapide	Rapide	0	0	0	0	0	100%	moyenne
		0	10000	0	0	10000	100%	
		10000	0	10000	0	0	100%	
		10000	10000	5807	0	5807	41,9%	
Rapide	Lente	0	0	0	0	0	100%	nulle
		0	10000	0	0	10000	100%	
		10000	0	10000	0	0	100%	
		10000	10000	9999	0	9999	0,0%	
Lente	Moyenne	0	0	0	0	0	100%	nulle
		0	10000	0	0	10000	100%	
		10000	0	10000	0	0	100%	
		10000	10000	8632	0	8632	13,7%	
Lente	Rapide	0	0	0	0	0	100%	très bonne
		0	10000	0	0	10000	100%	
		10000	0	10000	0	0	100%	
		10000	10000	145	0	145	98,6%	
Lente	Lente	0	0	0	0	0	100%	nulle
		0	10000	0	0	10000	100%	
		10000	0	10000	0	0	100%	
		10000	10000	9762	0	9762	2,4%	

Table 2.5 – Résultat des simulations HSIM de la porte NON-ET par concurrence pour les trois cinétiques. La porte comprend 2 réactions, avec 3 cinétiques différentes, soit 9 cas à tester.

2.1.6 Plusieurs fonctions pour une même porte

Lors des tests précédents, nous avons supposé que les seuils de séparation *faux/vrai* utilisés pour les entées étaient les mêmes que ceux utilisés pour les sorties. Ce n'est pas nécessairement le cas ; nous

allons nous intéresser à l'influence que peuvent avoir ces seuils sur la fonction booléenne calculée par une porte logique enzymatique. Pour cela, nous allons utiliser l'algorithme d'évaluation des sous-réseaux utilisé par Netgate, décrit dans le chapitre 3. Cet algorithme va faire correspondre une fonction booléenne à un sous-réseau en évaluant ce sous-réseau. Chaque combinaison des valeurs d'entrées est évaluée indépendamment. Cette évaluation nécessite de transformer l'information de forme booléenne en une information sous forme numérique pour effectuer la simulation du sous-réseau, puis de faire la transformation inverse sur le résultat. Ces transformations sont effectuées en utilisant des seuils pour déterminer le nombre de jetons représentant la valeur booléenne *vrai*, (*faux* étant représenté par aucun jeton). Les valeurs de ces seuils ont une importance considérable, car selon la valeur choisie, une même porte enzymatique pourra calculer une fonction booléenne différente.

Nous allons utiliser comme exemple le sous-réseau simple constitué des deux réactions :

$$\begin{array}{ccc} A & \xrightarrow{E_1} & C \\ B & \xrightarrow{E_2} & C \end{array}$$

Ce sous-réseau est en général associé à la fonction OU car C peut être produit par A ou par B (voir fig. 2.8). En réalité sa fonction dépend des seuils que l'on a choisis pour la tester. Nous allons commencer par fixer le seuil des entrées et prendre plusieurs valeurs pour le seuil de sortie :

- Entrées (valeur booléenne vers valeur numérique) : $vrai = 10$ et $faux = 0$
- Sortie (valeur numérique vers valeur booléenne) : nous testons deux cas différents :
  - si (valeur multivaluée ≥ 10) alors *vrai* sinon $faux$
  - si (valeur multivaluée ≥ 20) alors *vrai* sinon $faux$

Valeur entrée A		Valeur entrée B		Valeur sortie C (≥ 10)	
Booléenne	Numérique	Booléenne	Numérique	Booléenne	Numérique
faux	0	faux	0	0	faux
faux	0	vrai	10	10	vrai
vrai	10	faux	0	10	vrai
vrai	10	vrai	10	20	vrai

Table 2.6 – Seuil de sortie à 10 : fonction booléenne **OU** pour le sous-réseau de la figure 2.8.

Valeur entrée A		Valeur entrée B		Valeur sortie C (≥ 20)	
Booléenne	Numérique	Booléenne	Numérique	Booléenne	Numérique
faux	0	faux	0	0	faux
faux	0	vrai	10	10	faux
vrai	10	faux	0	10	faux
vrai	10	vrai	10	20	vrai

Table 2.7 – Seuil de sortie à 20 : fonction booléenne **ET** pour le sous-réseau de la figure 2.8.

On peut voir sur le tableau 2.6 que pour un seuil à 10 nous obtenons bien une porte OU alors qu'avec un seuil à 20, le même réseau calculera un ET (tableau 2.7). Nous avons donc deux fonctions logiques différentes pour un même sous-réseau, prenant les mêmes entrées et les mêmes sorties, selon le seuil utilisé pour discriminer la valeur booléenne de la sortie.

Une porte logique enzymatique associée à un sous-réseau peut donc avoir plusieurs fonctions booléennes potentielles. Une fonction booléenne sera correctement calculée lorsque les valeurs des seuils d'entrées et de sorties seront à l'intérieur d'intervalles spécifiques (domaines) associés à cette fonction.

Les domaines de ces fonctions booléennes ne peuvent pas se superposer pour un même sous-réseau. Pour un jeu défini de seuils, une seule fonction logique est calculée. Une fois les seuils fixés, une porte logique enzymatique ne peut donc calculer qu'une seule fonction booléenne. Le sous-réseau associé, lui, n'a pas réellement de *fonction logique*, on ne peut faire que différentes interprétations de ce que calcule ce sous-réseau.

En regardant les tableaux 2.6 et 2.7, on peut constater que ce n'est pas tant la valeur des seuils qui est importante, que les rapports entre le seuil de sortie et les seuils d'entrée. Pour la première valeur (10), nous avons un ratio de 1, alors que pour la deuxième valeur (20), le ratio est de 2, ce qui fait que pour interpréter la sortie à *vrai*, il faut que les deux entrées contribuent complètement, d'où la fonction booléenne ET.

2.1.6.1 Trouver les fonctions booléennes implémentées par une porte logique enzymatique

Pour utiliser correctement une porte logique enzymatique, il est crucial de connaitre tous les fonctions booléennes qu'elle peut calculer. On va avoir besoin d'une méthode qui permette d'obtenir pour n'importe quelle porte la liste exhaustives des fonctions booléennes qu'elle réalise, ainsi que pour chacune, son domaine de validité. Pour réaliser cette tâche nous avons conçu à partir de l'algorithme d'évaluation de sous-réseau de Netgate, un nouvel algorithme qui va construire le diagramme fonctionnel d'une porte logique enzymatique. Pour cela nous allons fixer le seuil de sortie et nous allons faire varier les seuils des entrées. Nous utilisons une méthode par percolation pour tester toutes les valeurs possibles de seuils d'entrée.

Nous allons utiliser comme exemple les portes logiques enzymatiques à deux entrées A et B et une sortie C. Le diagramme fonctionnel permettra de connaître la fonction booléenne (parmi les $(2^2)^2 = 16$ possibles) calculée en fonction du seuil choisi pour chacune des deux entrées. Chaque axe représentera les valeurs possibles de seuil pour une entrée (abcisse pour A et ordonnées pour B) et chaque point de coordonnées (s_A, s_B) du diagramme représentera la fonction booléenne implémentée. Ces points seront obtenus en *reconnaissant* la fonction booléenne grâce à sa table de vérité (comparée à la liste des tables de vérité associées aux 16 fonctions booléennes à deux entrées). Celle-ci sera obtenue en évaluant la sortie de la porte logique enzymatique sur chacune des 4 combinaisons possibles des entrées, en utilisant les seuils (s_A et s_B) pour la discrétisation.

Nous avons appliqué l'algorithme sur le sous-réseau présenté dans la figure 2.8 (voir fig. 2.14). Nous avons utilisé un seuil de sortie s_C fixé à 10, avec des seuil d'entrée variant de 0 à 19 par pas de 1. Nous avons trouvé plusieurs fonctions logiques possibles pour cette porte logique enzymatique : ET, OU, FAUX, OUI(A) et OUI(B). Nous savions déjà grâce aux tableaux 2.6 et 2.7 que nous allions avoir au moins les fonctions ET et OU. FAUX est la fonction où la valeur de sortie vaut *faux* pour toutes les combinaisons des valeurs d'entrées (C=*faux*), OUI(X) transmet la valeur de l'entrée X à la sortie, indépendamment de la valeur de l'autre entrée (C=X).

On peut constater que les domaines de seuils qui permettent le calcul d'une fonction booléenne donnée sont connexes et déterminés par des relations simples entre les seuils d'entrée s_A et s_B et le seuil de sortie s_C :

- quand $s_A \geq s_C$ et $s_B \geq s_C$ la fonction calculée est $C = A \vee B$
- quand $s_A < s_C$ et $s_B < s_C$ et $s_A + s_B \geq s_C$ la fonction calculée est $C = A \wedge B$
- quand $s_A + s_B < s_C$ la fonction calculée est $C = faux$
- quand $s_A < s_C$ et $s_B \geq s_C$ la fonction calculée est $C = A$
- quand $s_A \geq s_C$ et $s_B < s_C$ la fonction calculée est $C = B$

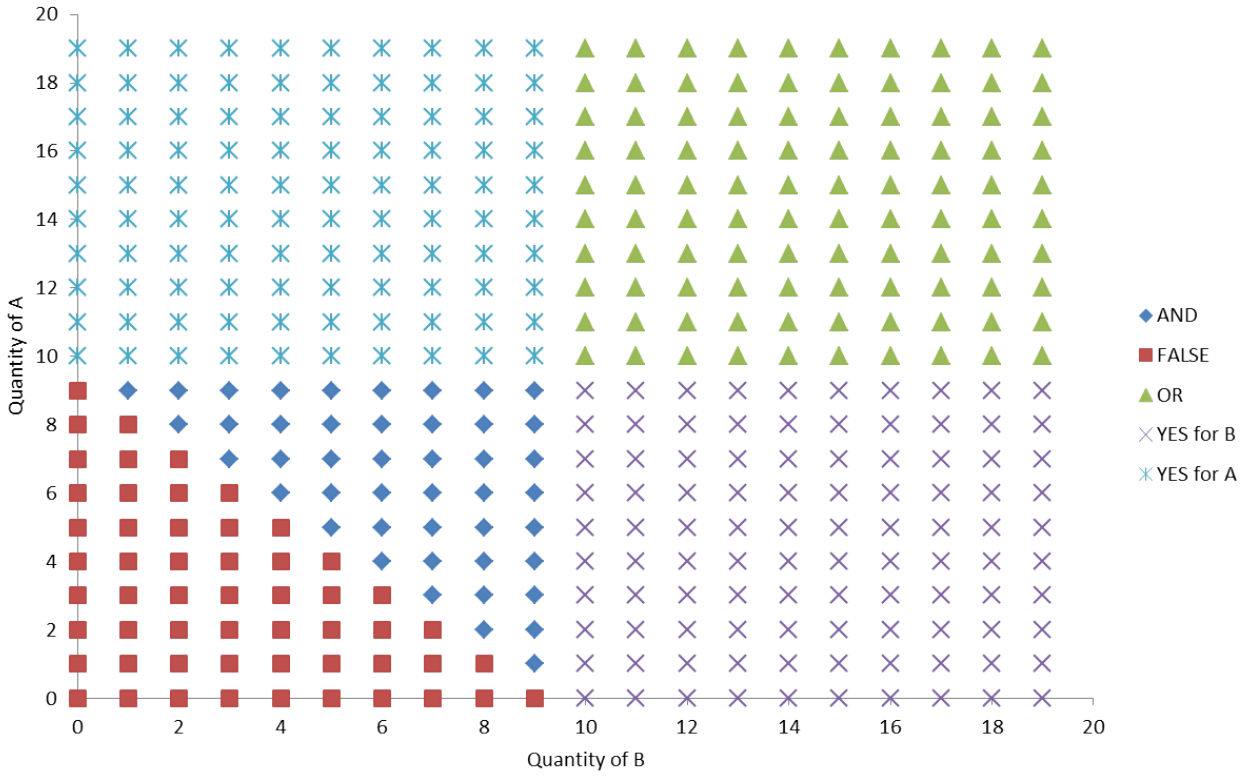


Figure 2.14 – Diagramme fonctionnel de la porte logique enzymatique associée au sous-réseau de la figure 2.8

2.2 Structures des circuits logiques enzymatiques

Les circuits logiques métaboliques sont des circuit logiques construits à partir de portes logiques enzymatiques. Ils ont leurs propres règles de construction, qui sont différentes des circuit logiques classiques. Les circuits logiques peuvent par exemple contenir des boucles. C'est une notion importante qui a de nombreuses applications utiles, mais que nous n'utiliseront pas. Nous allons nous intéresser uniquement à des circuits logiques non récursifs, avec une seule sortie et un nombre quelconque d'entrées. Nos circuits logiques auront donc toujours une forme arborescente avec la sortie à la racine et les entrées aux feuilles. Une seule sortie nous intéresse explicitement, mais il peut bien sûr y avoir des espèces moléculaires produites et non consommées.

Si un circuit logique ne contient pas de boucle, ce n'est pas forcément le cas pour le réseau métabolique qui l'implémente. Il est en effet tout à fait possible qu'une porte logique enzymatique possède une boucle dans son réseau métabolique (voir figure 2.15).

2.2.1 Contraintes

2.2.1.1 Contrainte de connexion

La première contrainte est la contrainte de connexion : pour enchainer deux portes logiques enzymatiques, il faut vérifier qu'elles puissent être connectées. Chaque flux d'information, chaque fil de connexion, est une espèce moléculaire spécifique. Ce métabolite doit être celui de sortie de la première porte et celui de l'entrée de la deuxième porte à laquelle on veut la connecter (voir figure 2.16).

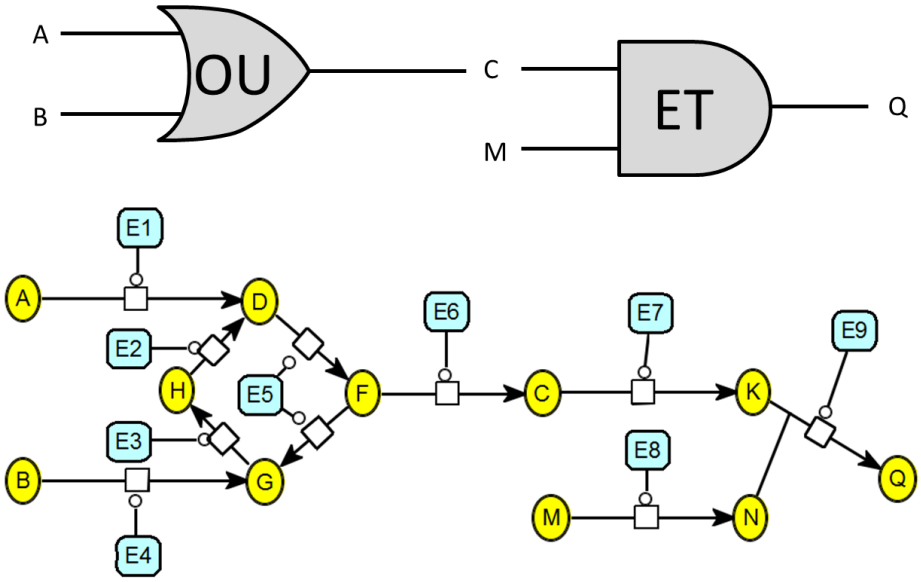


Figure 2.15 – Circuit calculant la formule booléenne $Q = M \wedge (A \vee B)$. Le circuit logique est arborescent (racine Q, feuilles A, B et M) nœuds internes OU et ET. L'implémentation de ce circuit présente une boucle dans la porte OU (pour des raisons de lisibilté, tous les substrats des réactions de la boucle ne sont pas montrés).

C'est la principale contrainte des réseaux logiques enzymatiques, et la raison pour laquelle il est nécessaire d'avoir à sa disposition le plus d'implémentations possible de chaque type de porte logique. On peut évidemment utiliser des *adaptateurs*, tels que des portes OUI, pour connecter deux portes qui ne partagent pas de métabolite commun. Ces composants vont permettre de convertir une espèce moléculaire en une autre pour adapter la sortie d'une porte à l'entrée de la suivante sans modifier la fonction booléenne calculée (voir figure 2.17).

2.2.1.2 Contrainte d'unicité

La deuxième contrainte est la contrainte d'unicité : si un métabolite est utilisé pour transporter une information alors il ne peut être utilisé que par la porte qui le produit et celles qui le consomment. Sinon il se produira l'équivalent d'un *court circuit* dans un circuit électrique. On peut voir à la figure 2.18) un circuit composé de cinq portes dont la fonction logique est $K = ((A \vee B) \wedge (C \vee D)) \wedge J$. On remarque que la molécule G a été utilisée à plusieurs endroits : la sortie de la porte OU, l'entrée de la porte ET suivante, la sortie de la porte OUI et l'entree de la porte ET finale. Cette molécule va donc connecter tous ces éléments les uns avec les autres. Ce court-circuit va modifier la fonction logique du circuit pour donner $K = (C \vee D) \wedge J$.

2.2.2 Portes à accumulation

Nos portes logiques enzymatiques sont des portes à accumulation : la forme arborescente des circuits fait que les flux convergent vers la sortie qui va donc s'accumuler. Nous mesurons la sortie après un temps suffisant pour en connaitre sa valeur booléenne. Nous considérerons que pour être une implémentation valide de porte enzymatique, le réseau métabolique doit avoir les propriétés suivantes :

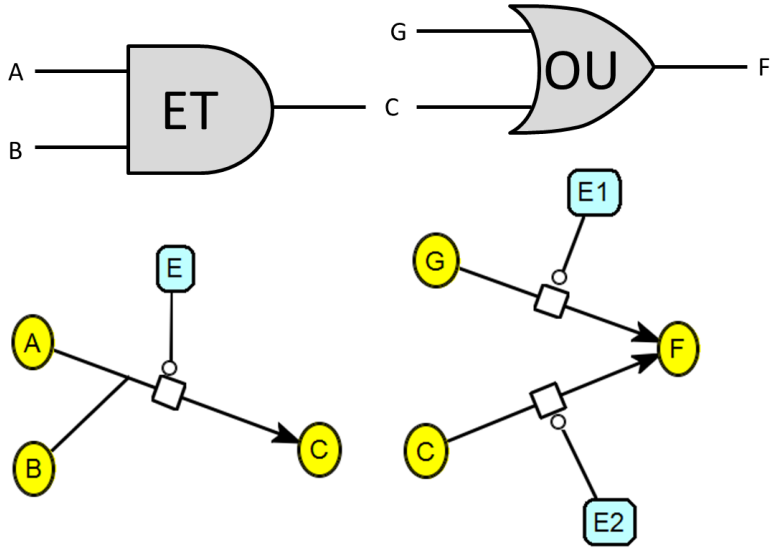


Figure 2.16 – Propriété de connexion des réseaux logiques enzymatiques. Il faut que la sortie d'une porte soit de même nature biochimique que l'entrée de la porte suivante.

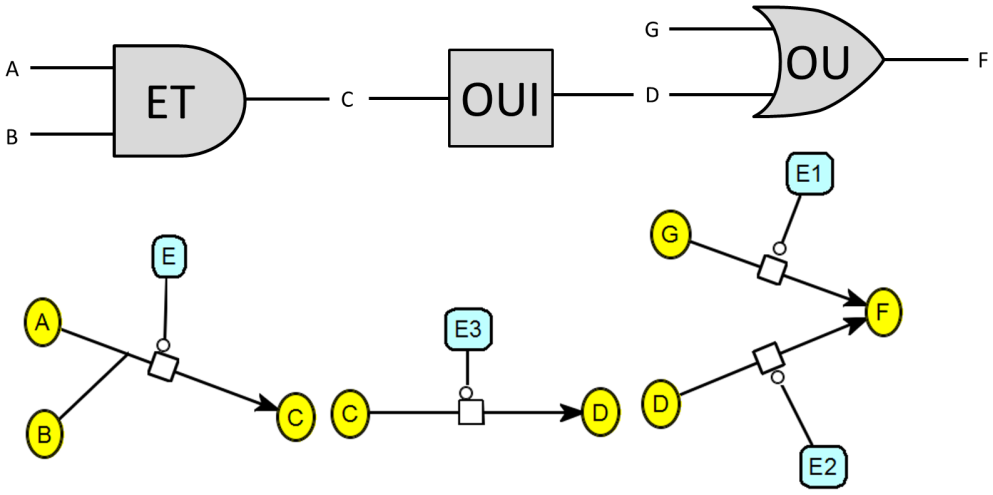


Figure 2.17 – Utilisation d'un adaptateur d'espèce moléculaire (porte OUI). Cette porte supplémentaire va permettre de transformer une espèce moléculaire en une autre pour s'adapter à la porte suivante.

les entrées de la porte ne peuvent qu'être consommées et jamais produites, et la sortie de la porte ne doit être que produite et jamais être consommée.

Ces propriétés permettent d'éviter des problèmes de concurrence à la jonction des portes logiques enzymatiques. Chaque implémentation d'une porte logique doit pouvoir fonctionner indépendamment de la composition du reste du circuit (sous réserve de respect de la règle d'unicité). Plus spécifiquement, le fonctionnement d'une porte ne doit pas être perturbé, quelle que soit la topologie de la porte qui la suit.

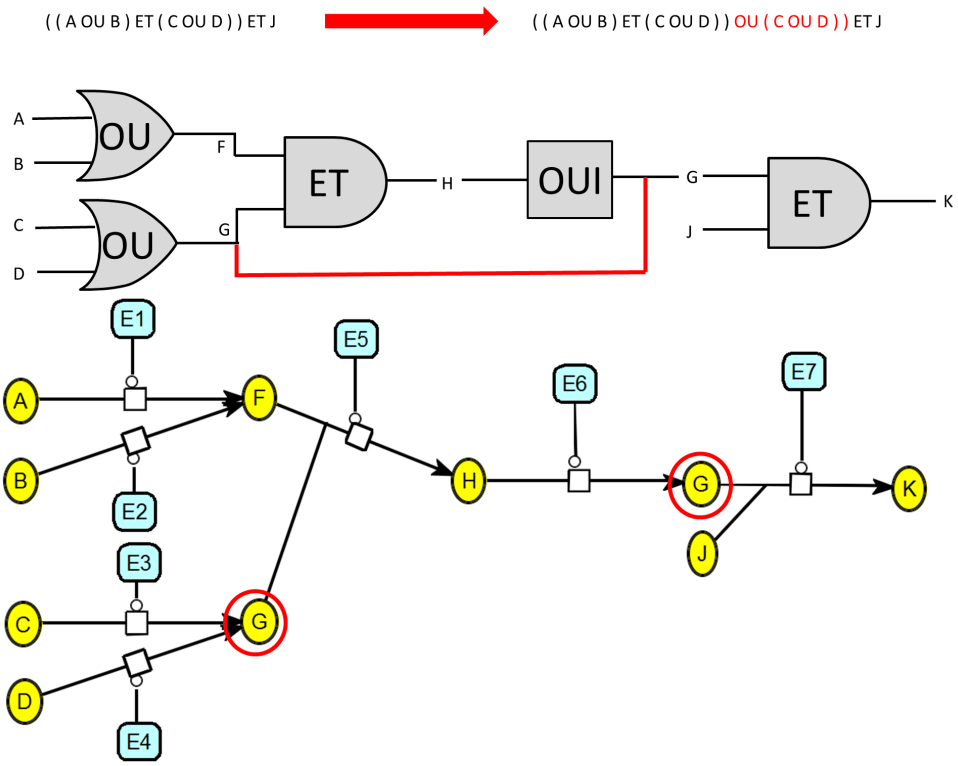


Figure 2.18 – Propriété d'unicité dans les réseaux logiques enzymatiques. On peut voir que le métabolite G, utilisé à deux endroits du circuit, va créer un court-circuit qui va changer la fonction logique calculée par le circuit.

Nous pouvons voir sur les figures 2.19 et 2.20, l'explication de la règle sur les sorties. Nous avons créé un modèle d'implémentation d'une porte NON-ET avec concurrence. L'inhibition est produite grâce une concurrence sur le métabolite F, qui peut être consommé de deux manières différentes. On supposera que le modèle respecte les contraintes cinétiques liées à ce type de concurrence (réaction $A + F \xrightarrow{E_3} p$ plus rapide que $F \xrightarrow{E_2} D$). Ensuite, une porte OUI est branchée sur D. Cette jonction ne pose aucun problème car nous avons respecté la règle : D n'est pas consommé dans la porte NON-ET (voir fig. 2.19).

Nous avons aussi construit un autre modèle d'implémentation pour une porte NON-ET avec concurrence, similaire à celui de la figure 2.19, mais cette fois-ci sans respecter la règle, ensuite nous avons branché la même porte OUI sur D (voir fig. 2.20).

Cette porte NON-ET est bien fonctionnelle : testée indépendamment elle sera correcte et correspondra fidèlement à la table de vérité attendue. Mais quand on la chaîne, on crée une concurrence sur D puisqu'il y aura désormais deux réactions qui vont le consommer. Cette concurrence ne sera pas testée cinétiquement puisqu'elle ne fait partie d'aucune des deux portes, le fonctionnement du circuit ne sera pas garanti. Son comportement sera incertain et dépendra de la cinétique de la première réaction qui va suivre la porte NON-ET. C'est un cas de figure que nous voulons absolument éviter, car c'est incompatible avec le concept de bibliothèque de portes utilisables pour la construction automatisée de circuits.

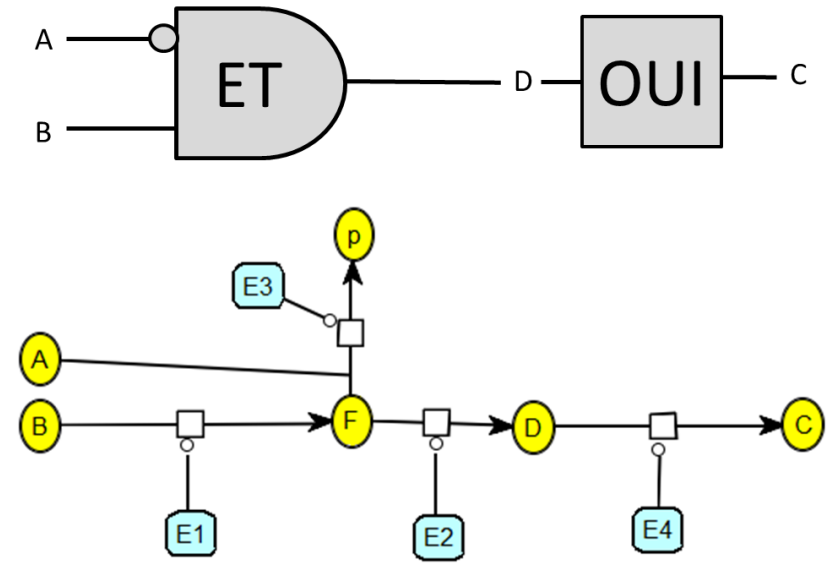


Figure 2.19 – Réseau logique enzymatique composé d'une porte NON-ET correctement implémentée, suivie d'une porte OUI. Il n'y a aucun problème de concurrence à la jonction des portes.

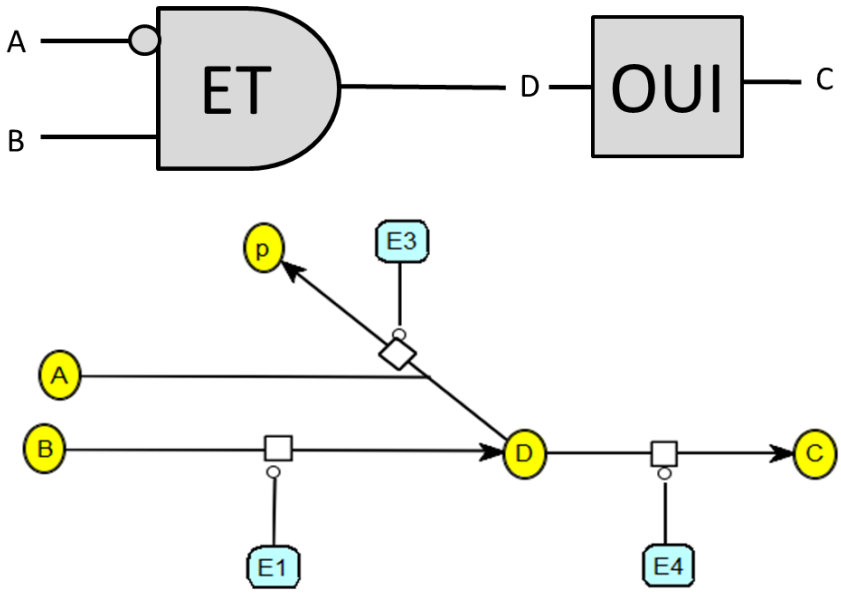


Figure 2.20 – Réseau logique enzymatique composé d'une porte NON-ET mal implémentée (sa sortie D est consommée en interne) suivie d'une porte OUI. Une concurrence involontaire a été créée à la jonction des portes.

2.2.3 Structures spatiale et temporelle

Un des aspects qui différencie fortement les circuits logiques enzymatiques des autres types d'implémentations est la notion de localisation spatiale des composants. Les circuits électroniques sont

implantés sur des plaques de silicium, les fils de connexion entre les portes sont des pistes de métal et les portes logiques elles-mêmes sont implémentées par des transistors, diodes et résistances avec une localisation spatiale fixe. Les circuits logiques optiques sont similaires, à ceci près que les fils sont des guides d'onde, mais n'en restent pas moins des composants matériels à implantation spatiale fixe elle aussi. Les circuits logiques génétiques sont déjà plus proches de nos circuits enzymatiques, par contre, étant implémentés sur de l'ADN, il y a encore un support physique unique pour chaque porte, support qui est relativement peu mobile. Ces trois types d'implémentation ont en commun le fait que même si le circuit n'est pas en train de fonctionner, si les entrées ne sont pas présentes, les portes logiques sont implémentées préalablement et présentes physiquement.

Du point de vue de *l'implantation physique*, les portes logiques enzymatiques peuvent être considérées comme presque entièrement abstraites. Lorsque la porte (ou le circuit complet) n'est pas utilisée, il n'y a de présent que des enzymes et des métabolites nécessaires (voir la définition des *extra-entrées* chapitre 3) répartis dans l'espace, sans rien d'actif. Rien n'est physiquement construit et placé à l'avance, d'ailleurs, toutes ces molécules diffusent du fait de l'agitation thermique. Ce ne sera que lorsque les entrées (à *vrai*) seront présentes que les métabolites associés commenceront à réagir avec les enzymes et les autres métabolites. C'est pendant que ces réactions seront actives que les portes logiques auront une existence physique. On peut considérer que c'est une forme de système qui s'auto-assemble (du point de vue fonctionnel, non pas physique) pour *accomplir sa fonction* et se déassemble lorsque les métabolites d'entrée ont été épuisés et que les réactions s'arrêtent [54]. D'ailleurs, selon les espèces moléculaires présentes à un moment donné, il serait tout à fait concevable d'avoir des circuits logiques enzymatiques complètement différents, calculant des fonctions booléennes différentes.

Les portes logiques enzymatiques sont donc temporaires, elles n'existent que lorsqu'elles fonctionnent. Elles ne sont pas vraiment localisées dans l'espace et leur arrangement est dynamique : au fur et à mesure que des portes entrent en fonctionnement (s'assemblent) d'autres s'arrêtent de fonctionner (se désassemblent).

2.3 Synchronisation

Dans un circuit logique enzymatique, toutes les réactions s'effectuent en parallèle. Les entrées d'une porte logique enzymatique peuvent être le résultat d'une longue suite de réactions, représentant les mécanismes des portes qui ont précédé la porte actuelle. Il est tout à fait possible que cette suite de réactions soit différente selon la valeur des entrées d'une même porte. Or c'est cette suite de réactions qui va déterminer le rythme d'arrivée des molécules représentant le signal d'entrée. Jusqu'à présent, nous avons testé les portes indépendamment du circuit dans lequel elles vont évoluer, nous avons supposé que les métabolites des entrées (à *vrai*) de ces portes étaient toujours présents en totalité au début de la simulation. Nous dirons que ces entrées sont présentes *en bloc*.

Nous allons maintenant nous intéresser au comportement qu'adoptent les portes logiques enzymatiques lorsque leurs entrées n'arrivent plus en bloc mais de façon continue, et éventuellement à des rythmes différents selon les entrées. Il est possible que l'espèce moléculaire correspondant à une entrée particulière soit entièrement présente alors que d'autres sont toujours en cours de calcul et donc encore à concentration nulle.

Nous allons séparer cette étude en deux parties : l'une traitera des différences de synchronisation au sein d'un circuit logique enzymatique (plus spécifiquement des différences de vitesse des flux moléculaires sur les entrées) et l'autre des différences de synchronisation dans la connexion du circuit à son environnement, de la forme des signaux, en bloc ou continus.

2.3.1 Synchronisation interne

Alors que certaines portes logiques sont plus sensibles que d'autres à une désynchronisation des signaux d'entrée, il existe des portes logiques qui, au lieu de subir les effets de la désynchronisation, vont changer la vitesse des flux moléculaires les traversant. Pour tester les effets de la désynchronisation, nous allons construire un circuit logique enzymatique de test. Ce circuit permettra de tester une porte logique enzymatique à deux entrées. L'une d'elles sera alimentée par le signal de sortie d'une porte, et l'autre alimentée à travers une longue chaîne de portes ET. Cela va créer explicitement une grande différence de temps entre les deux entrées. La longueur de la chaine de portes ET est variable mais supérieure à 1. La différence de temps sera donc au minimum de une unité et les signaux décalés auront donc une *forme continue* au niveau des entrées de la porte à tester.

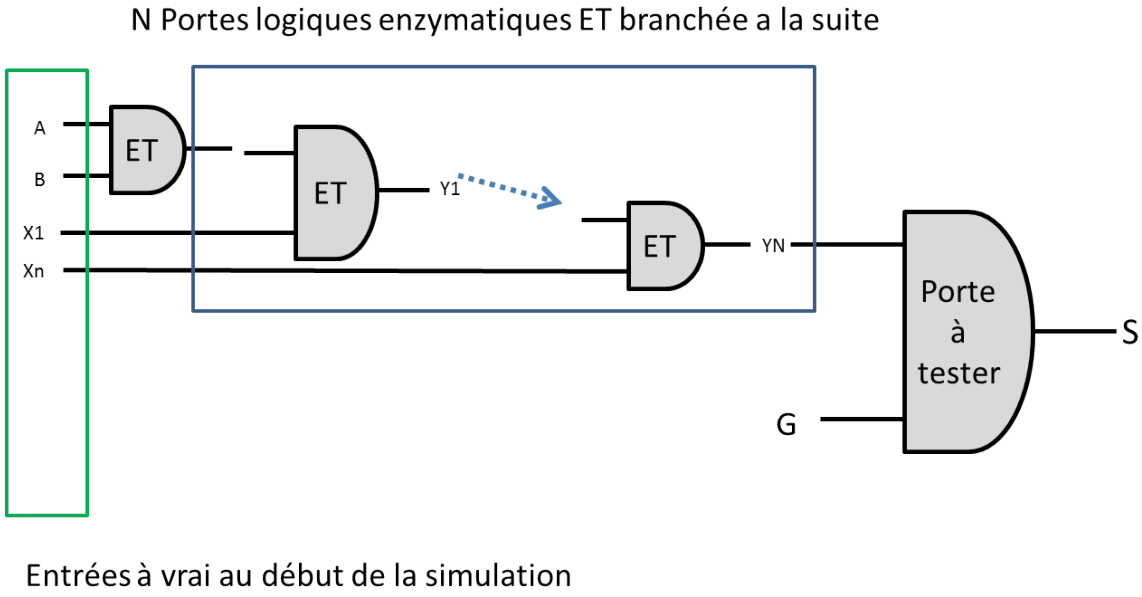


Figure 2.21 – Circuit logique enzymatique pour les tests de synchronisation interne. Ce circuit permettra de tester une porte logique enzymatique à deux entrées. L'une des entrées est alimentée par le signal de sortie G d'une porte (non représentée) alors que l'autre est alimentée à travers une longue chaîne de portes ET.

Une porte logique enzymatique ET classique, implémentée par exemple avec le sous-réseau de la figure 2.5, va non seulement être insensible à la désynchronisation des signaux d'entrées, mais aussi faire que le flux de sortie aura le même rythme que le plus lent des flux d'entrées. Il est en effet nécessaire pour produire une molécule de C, d'avoir la présence d'une molécule de A et d'une molécule de B. On peut voir sur la figures 2.22 des simulations où nous avons respectivement 2 et 11 portes ET en amont de la porte à tester. Le nombre de portes intermédiaires ne modifie pas la nature de la réponse mais seulement sa vitesse. Avec une forte désynchronisation des entrées, la courbe est moins compacte et la valeur finale met plus de temps à être atteinte.

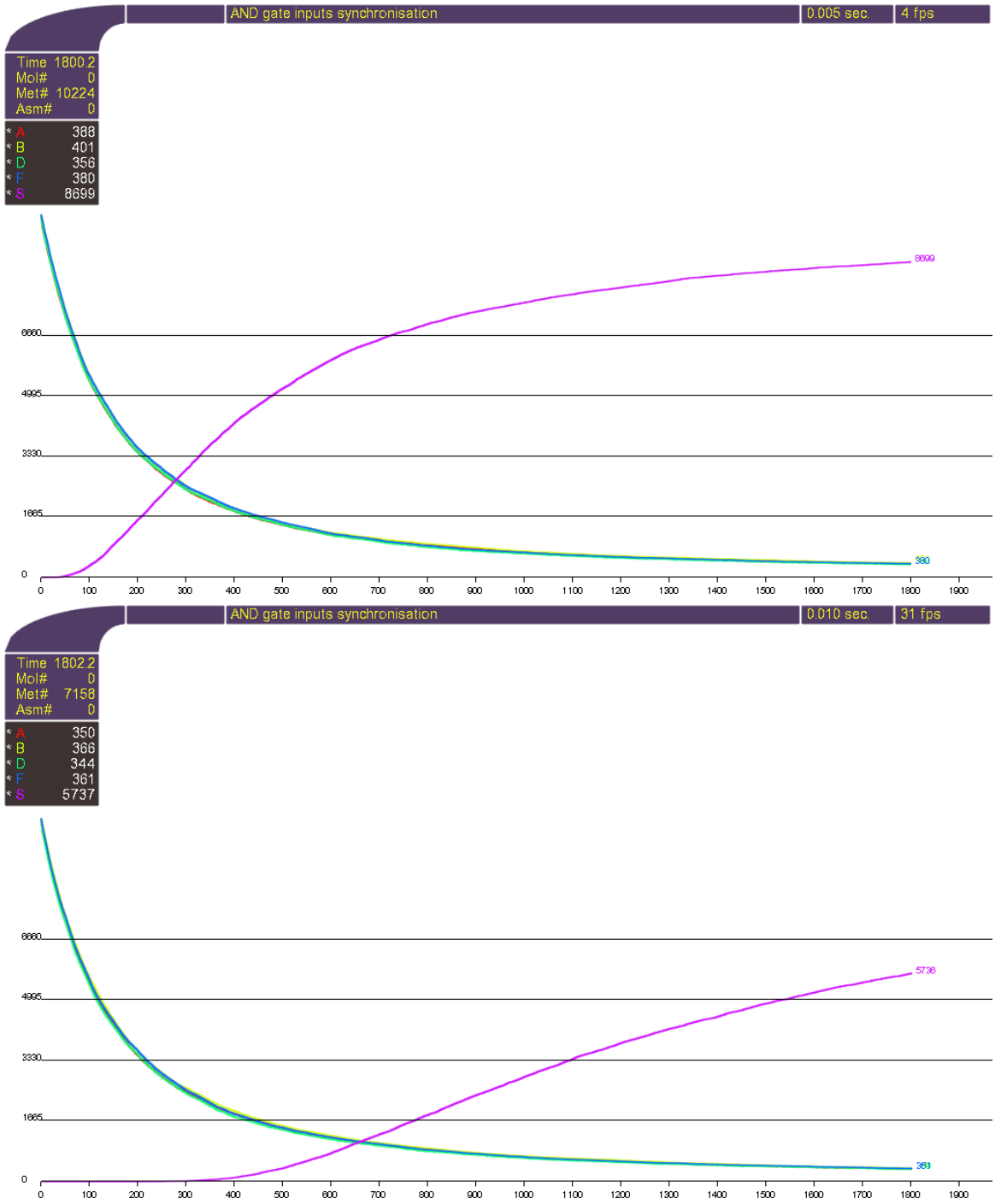


Figure 2.22 – Tests de synchronisation d'une porte ET. Avec 2 portes intermédiaires (en haut), et 11 portes intermédiaires (en bas).

Une porte logique enzymatique OU classique, comme celle implémentée par le sous-réseau de la figure 2.8 sera insensible à la désynchronisation des signaux d'entrée et ne changera que très peu le flux moléculaire qui la traverse. En effet une porte OU est d'un certain point de vue, deux portes OUI sortant le même métabolite, chacune prenant en entrée un métabolite différent. On peut voir sur la figure 2.23 des simulations avec respectivement 2 et 11 portes ET en amont de la porte OU à tester. La longueur de la chaîne de portes ET intermédiaires ne modifie pas la nature de la réponse. La courbe

est un peu moins compacte et la valeur finale est atteinte un peu plus lentement.

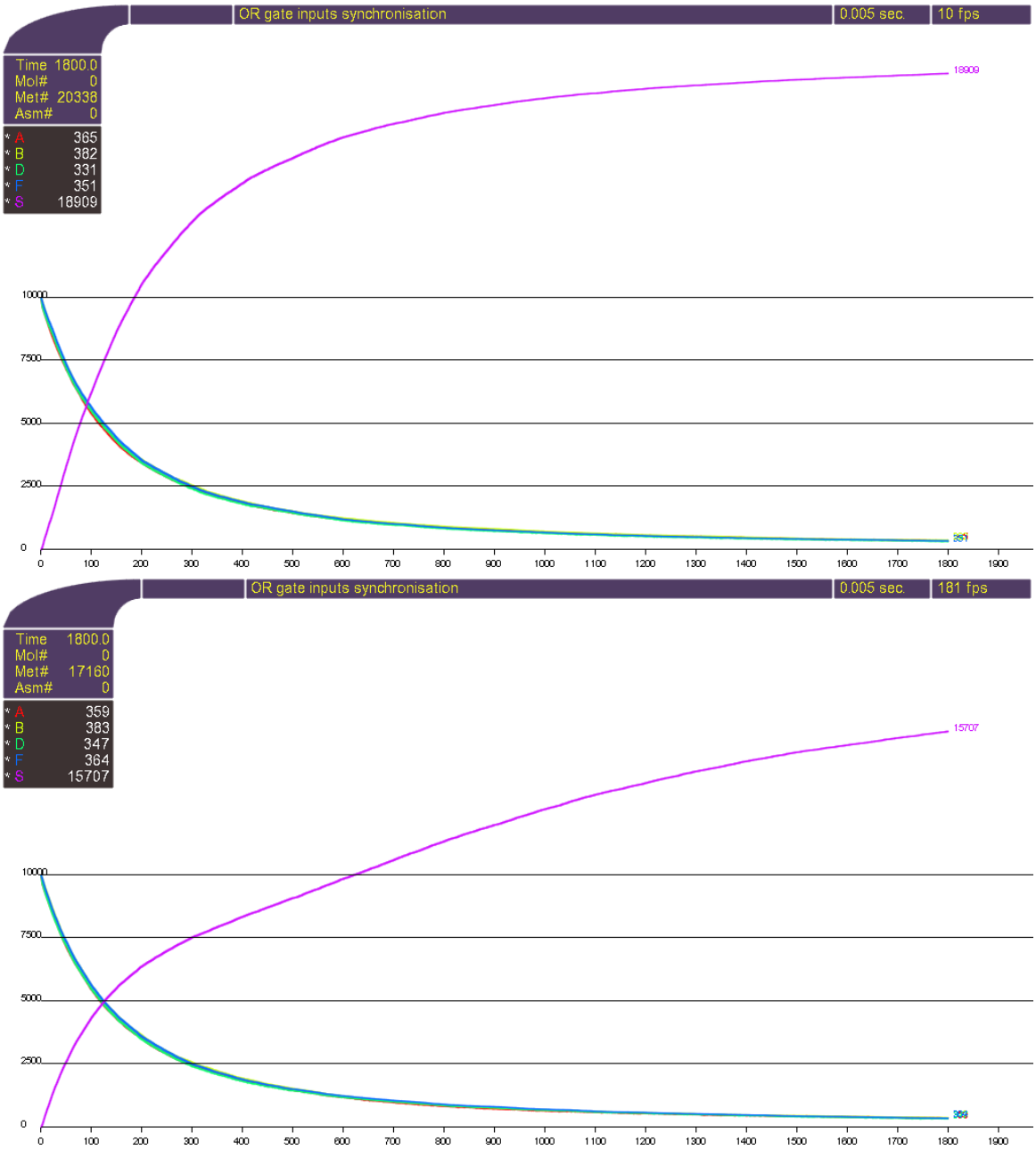


Figure 2.23 – Tests de synchronisation d'une porte OU. Avec 2 portes intermédiaires (en haut) et 11 portes intermédiaires (en bas).

Une porte logique enzymatique NON-ET par concurrence, comme celle implémentée par le sous-réseau de la figure 2.12, sera par contre très sensible à la désynchronisation des signaux d'entrée, et ne changera que très peu les flux moléculaires qui la traversent. Comme elle est basée sur une réaction de concurrence, c'est la différence de vitesse des réactions qui rend cette porte valide et selon l'entrée qui sera retardée, on va obtenir des résultats très différents.

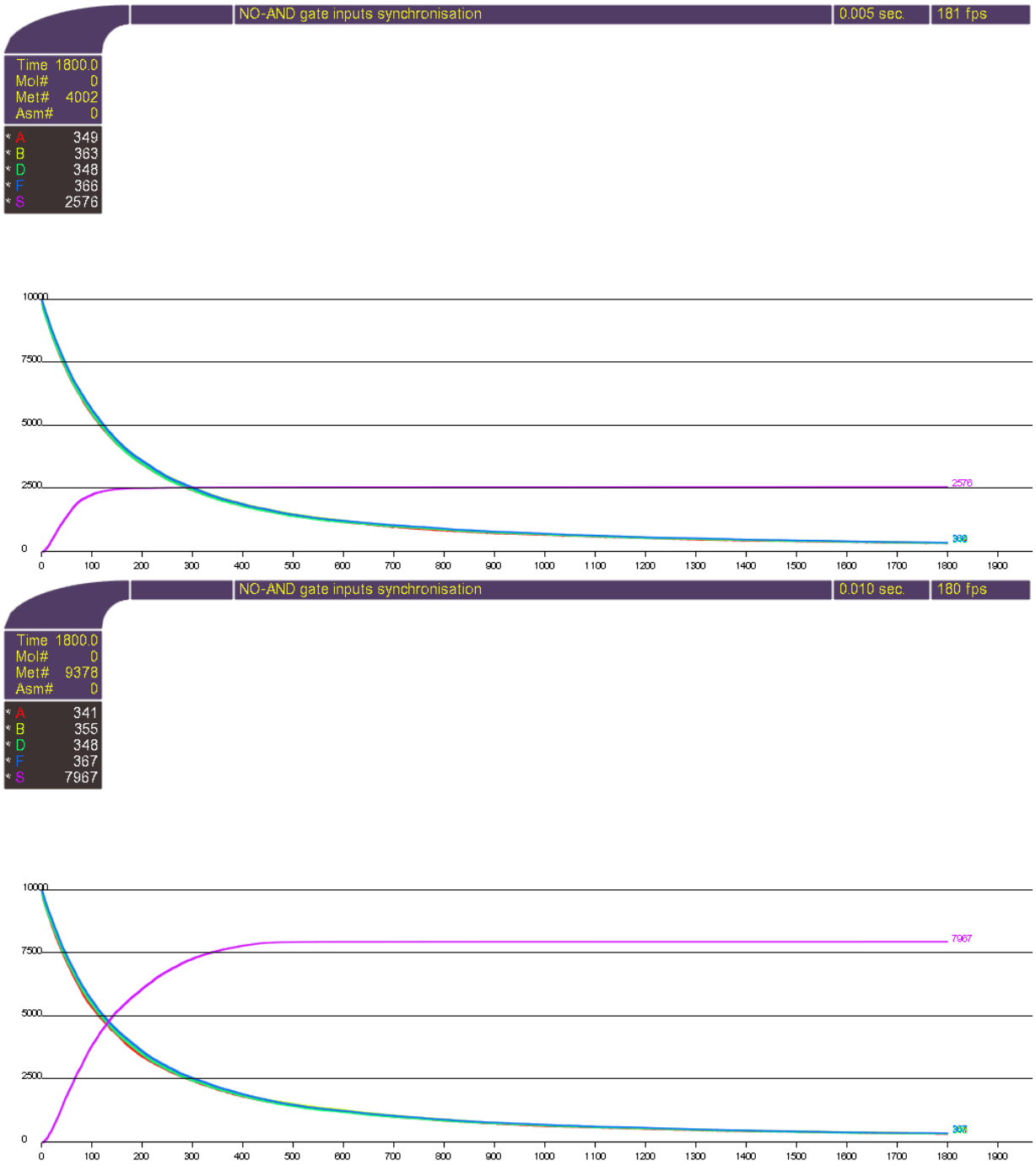


Figure 2.24 – Tests de synchronisation d'une porte NON-ET à concurrence. Avec 2 portes intermédiaires (en haut) et 11 portes intermédiaires (en bas). La chaîne de portes ET étant connectée à l'entrée inhibitrice.

Sur les simulations de la figure 2.24 la chaîne de portes ET est branchée sur l'entrée inhibitrice, avec respectivement 2 et 11 portes ET avant la porte à tester. Nous voyons que la désynchronisation a un effet important : même avec seulement une porte de différence entre les entrées, la réponse est déjà très forte pour un 0 logique. Avec 10 portes de différence, la signal de sortie devient complètement faux avec une réponse correspondant à un 1 logique.

Sur les simulations de la figures 2.25 la chaîne de portes ET est branchée sur l'entrée non-inhibitrice, avec respectivement 2 et 11 portes ET avant la porte NON-ET. On constate que la désynchronisation a l'effet inverse que précédemment, et consolide l'inhibition en retardant la réaction directe.

On peut donc voir que l'effet de désynchronisation des entrées d'une porte NON-ET peut être avantageux ou bien rédhibitoire selon la façon dont elle est connectée.

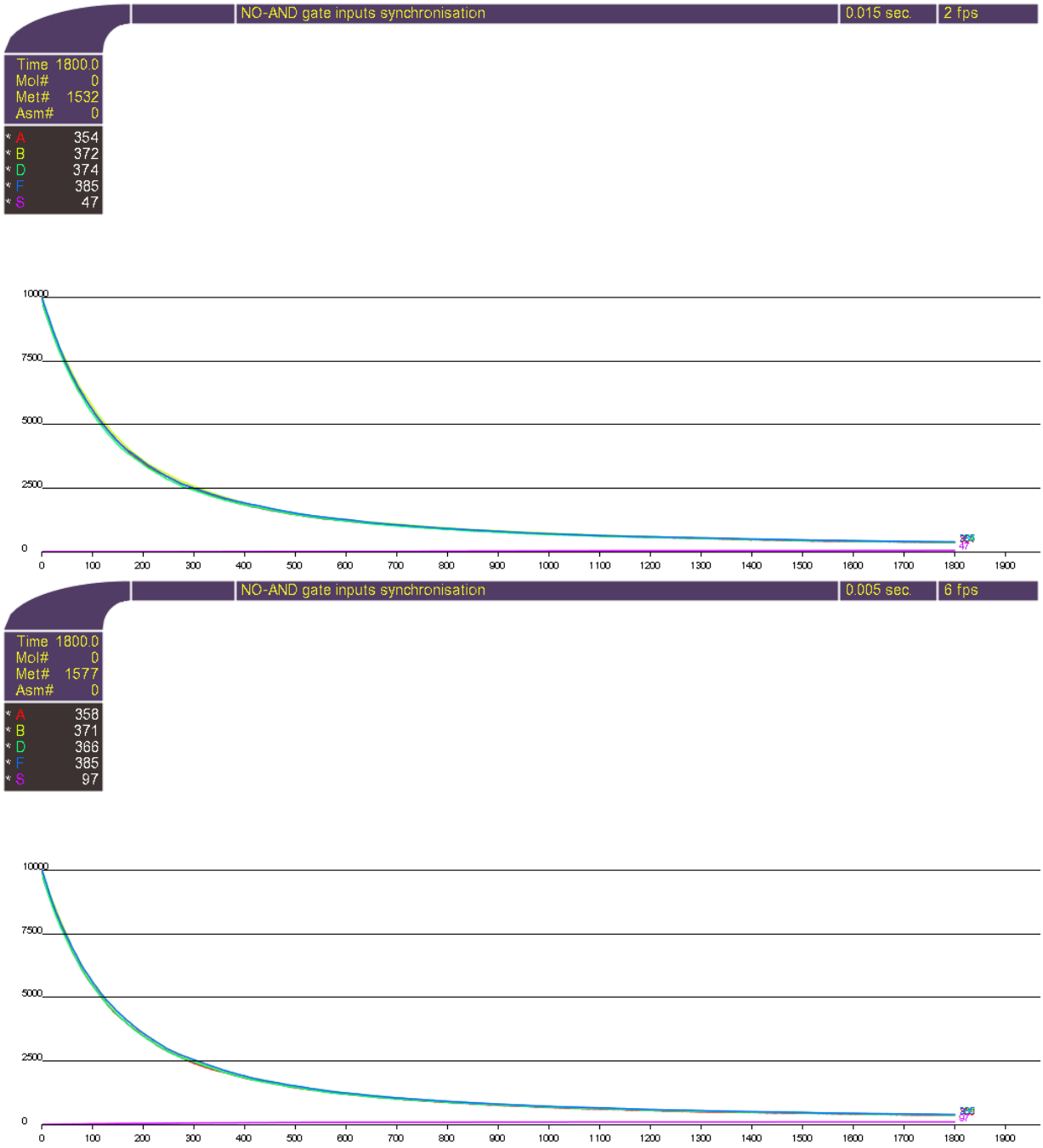


Figure 2.25 – Tests de synchronisation d'une porte NON-ET à concurrence. Avec 2 portes intermédiaires (en haut) et 11 portes intermédiaires (en bas). La chaine de porte ET étant branchée sur l'entrée non inhibitrice.

2.3.2 Synchronisation externe

Avec les portes ET et OU il n'y a aucune différence si les entrées sont *continues* ou *en bloc*. Nous allons maintenant voir ce qu'il en est pour les portes NON-ET à concurrence. Pour cela, nous utiliserons un nouveau circuit de test assez simple où l'une des entrées de la porte à tester sera directement en bloc et l'autre sera la sortie d'une unique porte ET.

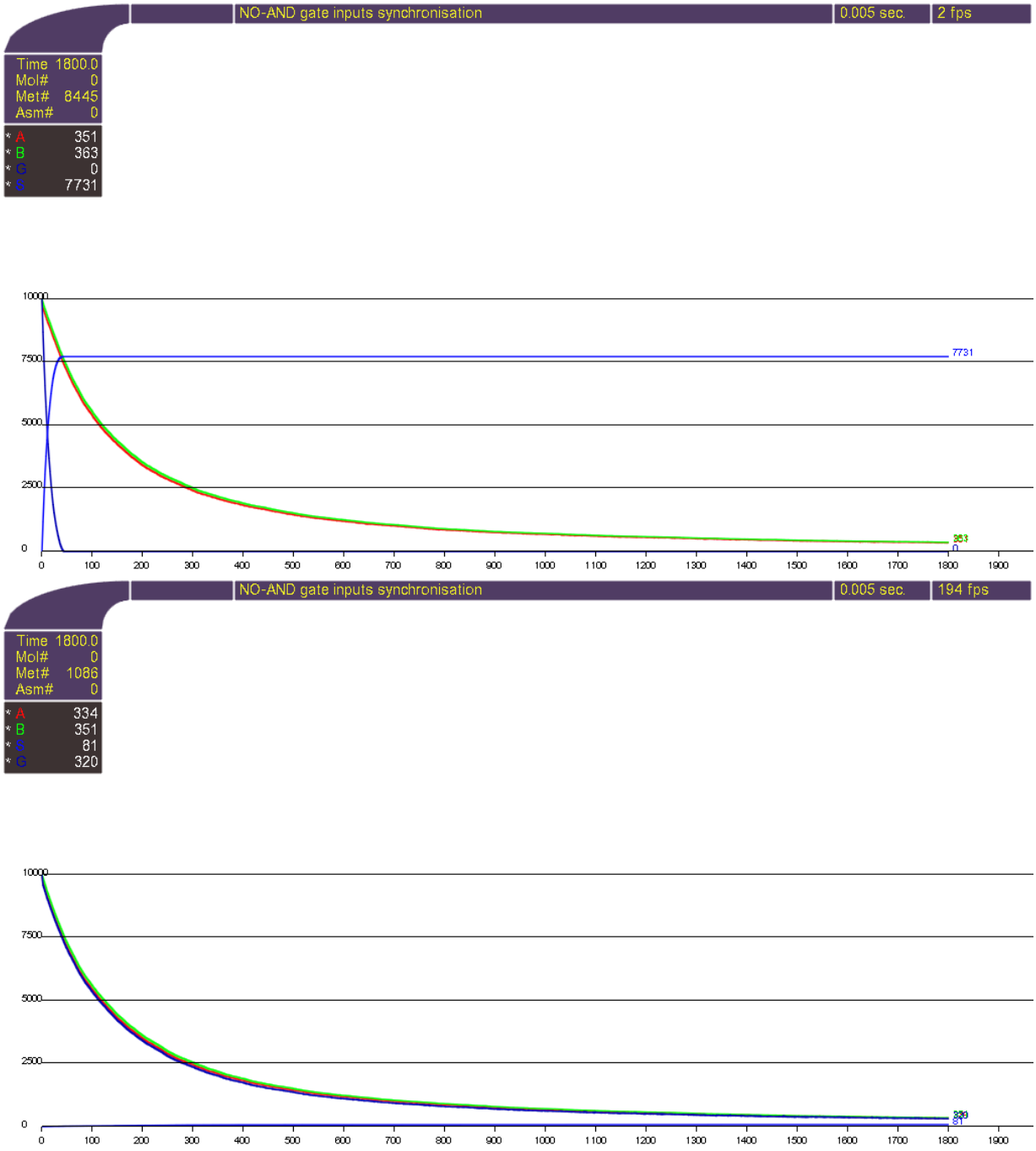


Figure 2.26 – Tests de synchronisation externe d'une porte NON-ET à concurrence. L'entrée en bloc est connectée à l'entrée non-inhibitrice (en haut) ou à l'entrée inhibitrice (en bas).

Nous pouvons voir sur la figure 2.26 le test de la porte NON-ET selon que l'entrée en bloc est connectée

à l'entrée inhibitrice ou non. Le signal est de très bonne qualité (97%) dans le premier cas, mais est très mauvais (22%) dans le deuxième. Donc une différence de synchronisation externe est encore pire qu'une différence due à un retard de 10 portes logiques ET en synchronisation interne.

Il est donc nécessaire de faire particulièrement attention aux portes logiques directement en contact avec une entrée *en bloc*. Il est conseillé de mettre une pré-couche constituée de portes OUI devant le circuit logique enzymatique, qui va transformer la forme du signal pour lui faire perdre sa forme *en bloc*.

2.4 Composants supplémentaires

Au delà des portes logiques, nous avons essayé de construire d'autres composants utiles, tels que ceux existant en électronique.

2.4.1 Amplificateur

Un amplificateur va être utile par exemple pour restaurer un signal affaibli par le passage à travers certains types de portes. Un bon amplificateur doit sortir un signal amplifié tout en gardant un lien de proportionnalité avec le signal original. Une perte partielle de ce lien de proportionnalité introduit une *distorsion* du signal, mais ce n'est pas forcément très grave quand il représente un signal logique binaire.

Un moyen simple d'obtenir un amplificateur ayant un gain de 2 (par exemple) est de trouver une réaction qui *coupe* son substrat (A) en deux sous-parties identiques (B), et une réaction (ou une chaîne de réactions) qui prend comme substrat le métabolite à amplifier sig et qui produit A (entre autres) :

$$sig + X \xrightarrow{\mathrm{E_1}} \mathrm{met} + \mathrm{A}$$
$$\mathrm{A} \xrightarrow{\mathrm{E_2}} \mathrm{B} + \mathrm{B}$$

Il suffit d'avoir initialement une certaine quantité de l'espèce moléculaire X, pour que la première réaction fabrique autant de copies de A qu'il y aura de copies de sig (et d'autant plus vite que la concentraton de sig est élevée). La deuxième réaction transformera chaque copie de A en deux copies de B, réalisant ainsi l'amplification d'un facteur 2 du signal représenté par sig.

Nous avons fait une simulation de test avec une cinétique moyenne, il y a bien doublement des quantités moléculaires. Nous somme partis de 600 molécules de sig et nous avons 1190 molécules de B après 90 secondes de temps simulé (voir fig. 2.27).

2.4.2 Capteur

Si on veut détecter un métabolite que l'on sait être en très faible quantité quand il est présent, et transformer cette présence ou absence en un signal logique, on va amplifier ce signal de façon non linéaire de façon à ce que son absence (ou quasi absence) fournisse une faible quantité de produit (niveau logique 0) et à ce que sa présence, même à faible concentration, fournisse une quantité de produit dépassant le seuil défini pour avoir le niveau logique 1.

Pour réaliser cela, il suffit de trouver une enzyme qui nécessite au préalable de fixer un métabolite pour être active et pouvoir catalyser une réaction :

$$\mathrm{E_{inac}} + \mathrm{m} \rightleftharpoons \mathrm{E_{act}}$$
$$\mathrm{A} \xrightarrow{\mathrm{E}_{act}} \mathrm{B}$$
$$\mathrm{B} \xrightarrow{Deg} \mathrm{J}$$

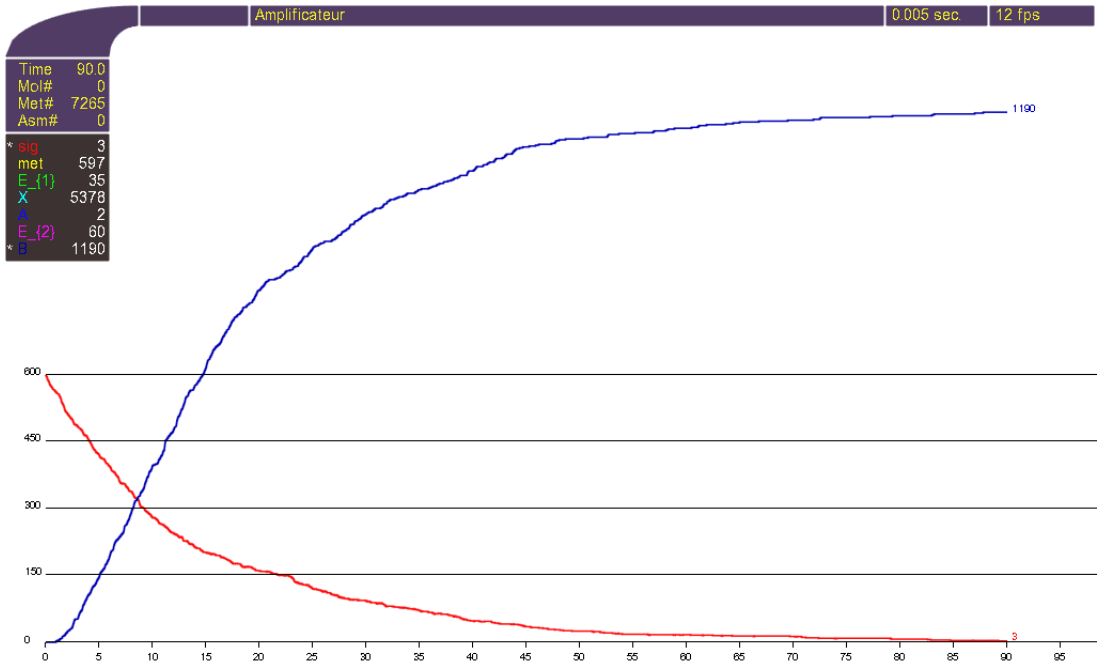


Figure 2.27 – Amplificateur enzymatique : on peut voir qu'après 90 secondes, la quantité de métabolite B (en bleu) est quasiment le double de la quantité initiale de métabolite *sig* (en rouge).

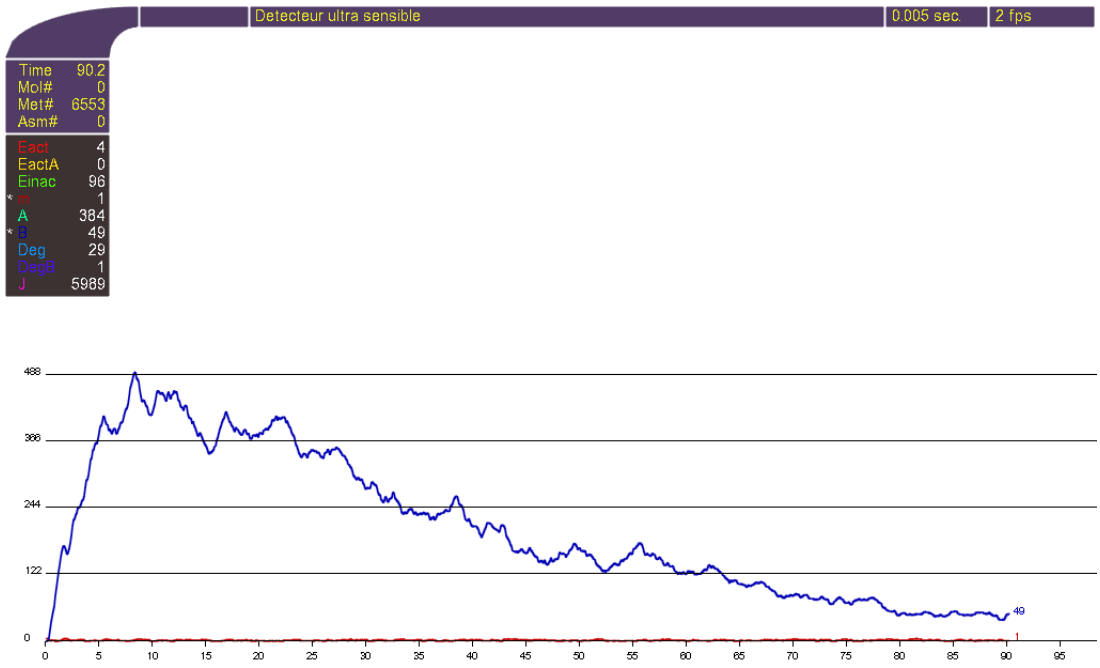


Figure 2.28 – Détecteur ultra sensible : on peut voir qu'au début de la simulation, la quantité de métabolite B (en bleu) atteint presque 500 copies pour seulement 5 copies de métabolite *sig* (en rouge) initialement.

Pour détecter la présence en faible quantité du métabolite *sig*, il suffira alors de trouver une (ou une

série) de réactions (portes OUI) qui transforme *sig* en m et d'avoir initalement une forte concentration du métabolite A. Si une faible concentration du métabolite *sig* est présente, aprés un certain temps on va obtenir quelques copies du métabolite m, qui vont activer un certain nombre d'enzymes. Ces enzymes E_{act} vont catalyser la réaction qui va produire B à une vitesse d'autant plus grande que la concentration de *sig* est élevée, et entrer en concurrence avec la réaction de *dégradation* catalysée par l'enzyme *Deg* qui va consommer B pour produire le métabolite inutilisé J.

Si le nombre de copies de *sig*, bien que très faible, est au dessus d'un certain seuil, la quantité résiduelle de B sera au dessus du seuil de définition du niveau logique 1. Si le nombre de copies de *sig* est trop faible, la réaction de dégradation prendra le pas sur celle de production de B, et sa quantité ne dépassera jamais le seuil et représentera le niveau logique 0. On peut donc théoriquement, modulo des *réglages* fins (cinétiques relatives des diverses réactions, quantité initiale de A), transformer un signal très faible en signal logique *propre* (fig. 2.28).

2.5 Codage *Diphase différentiel*

La seconde méthode de codage de l'information par des flux moléculaires que nous avons conçue consiste à représenter un fil de connexion, non plus par une, mais par deux espèces moléculaires. L'une des espèces moléculaires transportera un signal correspondant au 1 logique et l'autre un signal correspondant au 0 logique. Ce type de codage n'utilise donc pas l'absence de molécules pour coder le 0 logique. Il permet en outre d'avoir un état *éteint* qui va correspondre une absence de signal :

— $[a_0]$ forte et $[a_1]$ faible : valeur booléenne *faux* (1 logique)
— $[a_0$ faible et $[a_1]$ forte : valeur booléenne *vrai* (0 logique)
— $[a_0]$ forte et $[a_1]$ forte : état *indéterminé* (interdit)
— $[a_0]$ faible et $[a_1]$ faible : état *éteint*

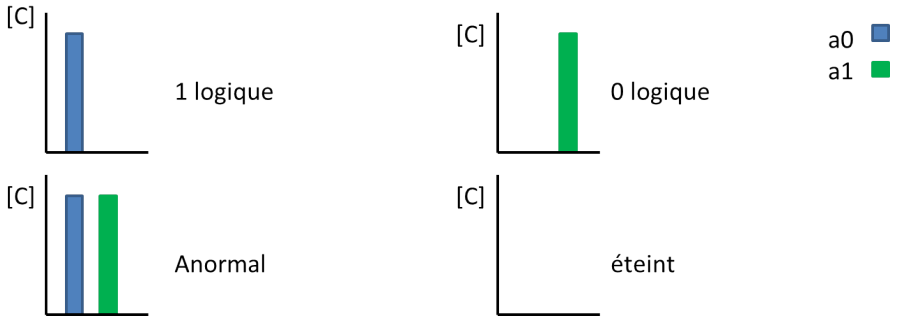


Figure 2.29 – Les quatre états d'un flux d'information en représentation Diphase : 1 logique, 0 logique, anormal, et éteint.

Ce système est avantageux, car il ne nécessite pas vraiment de seuil, les valeurs relatives des quantités de molécules de a_0 et de a_1 déterminent la valeur logique, c'est une sorte de codage différentiel.

2.5.1 Porte ET

Nous avons adapté la table de vérité de l'opérateur booléen ET pour refléter les états internes de la porte ET Diphase avec A et B comme entrées et C comme sortie (voir table : 2.8).

On va utiliser quatre réactions bi-substrats. Elles vont représenter toutes les lignes de la table de vérité. Chaque réaction utilise une composante du flux d'information A et une composante du flux

d'information B pour donner une composante du flux d'information C (voir fig. 2.30).

Entrée A	Entrée B	Sortie C
présence a_0 et absence a_1	présence b_0 et absence b_1	présence c_0 et absence c_1
présence a_0 et absence a_1	absence b_0 et présence b_1	présence c_0 et absence c_1
absence a_0 et présence a_1	présence b_0 et absence b_1	présence c_0 et absence c_1
absence a_0 et présence a_1	absence b_0 et présence b_1	absence c_0 et présence c_1

Table 2.8 – Table de vérité adaptée au codage Diphase de l'opérateur ET.

$$\begin{aligned} \mathrm{a_0} + \mathrm{b_0} &\xrightarrow{\mathrm{E_1}} \mathrm{c_0} \\ \mathrm{a_1} + \mathrm{b_0} &\xrightarrow{\mathrm{E_2}} \mathrm{c_0} \\ \mathrm{a_0} + \mathrm{b_1} &\xrightarrow{\mathrm{E_4}} \mathrm{c_0} \\ \mathrm{a_1} + \mathrm{b_1} &\xrightarrow{\mathrm{E_3}} \mathrm{c_1} \end{aligned}$$

Nous pouvons voir que cela correspond à quatre portes ET Monophase imbriquées. Les portes Diphase sont plus complexes à construire, elles nécessitent plus d'enzymes, et des enzymes très spécifiques. Connecter ces portes entre elles sera plus compliqué : puisqu'il y a deux espèces moléculaires par connexion, on aura beaucoup moins de possibilités pour trouver des portes connectables (sans court-circuit). La différentiation entre les valeurs logiques 0 et 1 sera par contre plus sûre, puisque chaque valeur sera *portée* soit par une espèce moléculaire, soit par l'autre, et cela, quasiment quelle que soit la valeur absolue des concentrations.

Ce codage autorise aussi un état éteint, qui donne la possibilité d'avoir des connexions non actives. Cela correspond à l'état *haute impédance* des portes logiques électroniques *tristate*. Lorsque les entrées d'un circuit ne sont pas présentes, les connexions de toutes les portes sont dans cet état éteint.

On verra par la suite que les portes Diphase ne sont constituées que de modèles d'implémentation de portes ET Monophase, et donc sont très robustes vis-à-vis de la cinétique des réactions. De plus, elles provoquent la synchronisation systématique des flux d'entrée : si l'un des deux flux n'est pas présent, la porte ne s'activera pas. La contrainte majeure de cet type de codage, et donc des implémentations, est le type très spécifique des enzymes qu'on aura beaucoup de mal à trouver dans le réseau métabolique d'un organisme vivant.

A partir de la porte ET Diphase, il est facile de dériver toutes les portes à deux entrées et une sortie, même les plus compliquées, comme par exemple le XOR (table 2.9).

Entrée A	Entrée B	Sortie C
présence a_0 et absence a_1	présence b_0 et absence b_1	présence c_0 et absence c_1
présence a_0 et absence a_1	absence b_0 et présence b_1	absence c_0 et présence c_1
absence a_0 et présence a_1	présence b_0 et absence b_1	absence c_0 et présence c_1
absence a_0 et présence a_1	absence b_0 et présence b_1	présence c_0 et absence c_1

Table 2.9 – Table de vérité adaptée au codage Diphase de l'opérateur XOR.

On constate que la porte XOR, comme la porte ET, est aussi constituée de quatre réactions de type porte ET en codage Monophase (fig. 2.31). La seule différence vient du fait qu'il va falloir des enzymes avec des spécificités légèrement différentes, mais ni leur type, ni le nombre d'éléments incorporés ne

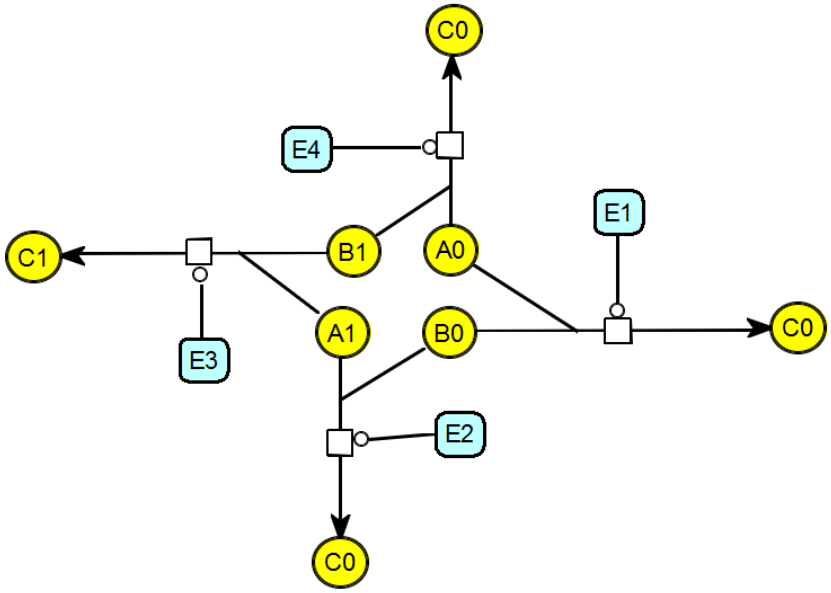


Figure 2.30 – Modèle d'implémentation d'une porte logique ET en codage Diphase.

changent. Cette caractéristique reste vraie pour toutes les portes à deux entrées et une sortie que l'on construira sur ce modèle. Il suffit pour chaque version d'ajuster les produits des réactions pour obtenir la table de vérité, et donc la fonction voulue.

$$\begin{aligned} a_0 + b_0 &\xrightarrow{E_1} c_0 \\ a_1 + b_0 &\xrightarrow{E_2} c_1 \\ a_0 + b_1 &\xrightarrow{E_4} c_1 \\ a_1 + b_1 &\xrightarrow{E_3} c_0 \end{aligned}$$

Il est également très simple d'implémenter un inverseur (porte NON) sans *réserve* de molécules ou de réaction inhibée, puisque dans le codage Diphase, la valeur logique 0 n'est pas équivalente à l'absence de signal, mais au flux inverse de celui représentant le 1 logique (table 2.10).

Entrée A	Sortie C
présence a_0 et absence a_1	absence c_0 et présence c_1
absence a_0 et présence a_1	présence c_0 et absence c_1

Table 2.10 – Table de vérité adaptée au codage Diphase de l'opérateur NON.

La table de vérité ayant deux ligne, nous n'aurons besoins que de deux réactions (correspondant à deux portes OUI en codage Monophase, voir fig. 2.32) :

$$\begin{aligned} a_0 &\xrightarrow{E} c_1 \\ a_1 &\xrightarrow{E} c_0 \end{aligned}$$

Bien qu'ayant beaucoup d'avantages sur le codage Monophase, le codage Diphase différentiel n'est pas bien adapté à notre problématique. Nous cherchons à construire de grandes bibliothèques de portes logiques enzymatiques à partir des réseaux métaboliques d'organismes vivants. Les réseaux

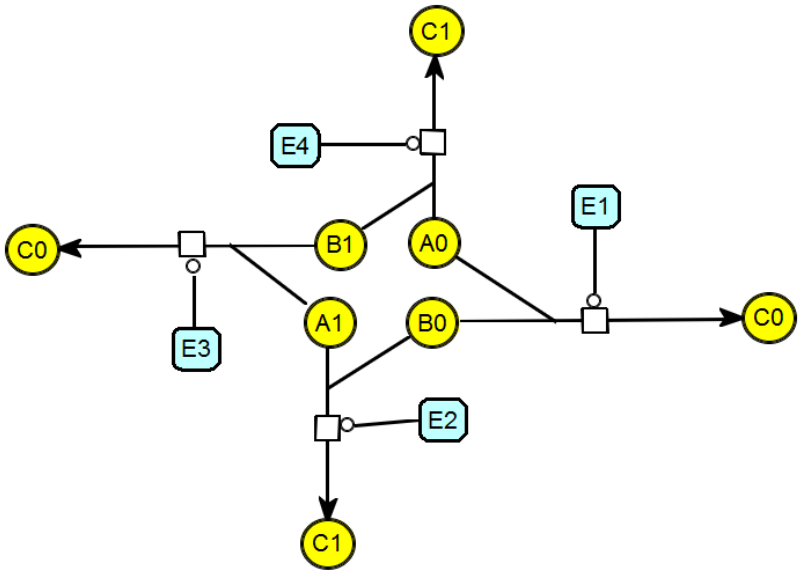


Figure 2.31 – Modèle d'implémentation d'une porte logique XOR en codage Diphase.

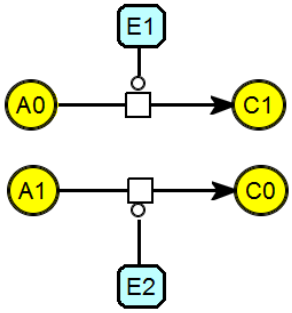


Figure 2.32 – Modèle d'implémentation d'une porte NON en codage Diphase.

métaboliques naturels sont très divers, néanmoins il est très peu probable d'y trouver des enzymes aussi spécialisées que celles nécessaires au codage Diphase. C'est pour cela qu'on utilisera le codage Monophase, qui est beaucoup moins contraint sur les enzymes, pour implémenter nos portes logiques.

Chapitre 3

Des réseaux métaboliques aux circuits logiques moléculaires

Nous avons établi un cadre théorique pour les portes et les circuits logiques enzymatiques. Nous allons l'utiliser pour concevoir deux logiciels qui vont nous permettre d'atteindre notre but : la création automatisée de réseaux métaboliques capables de traiter de l'information sous forme booléenne (voir fig. 3.1).

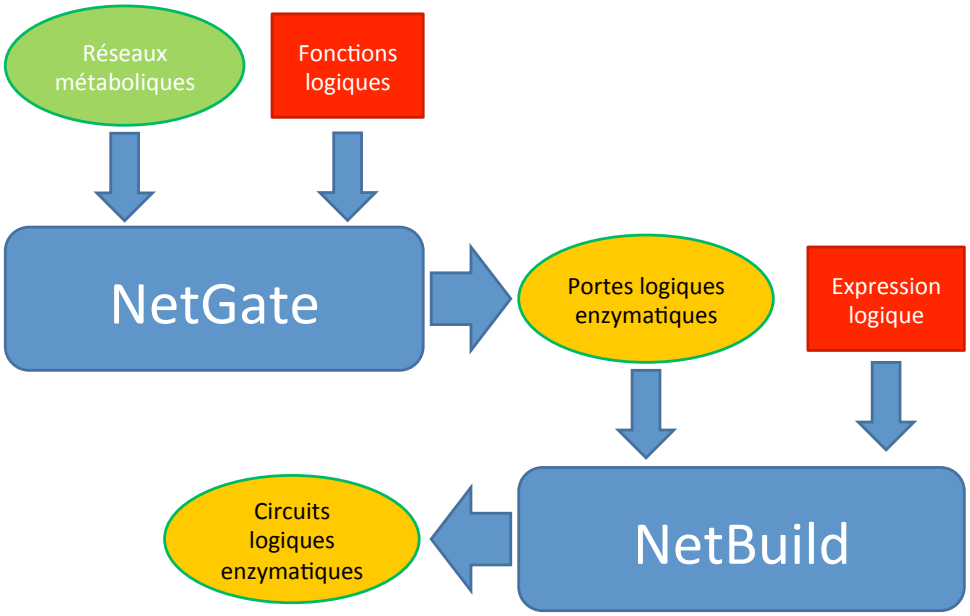


Figure 3.1 – Enchainement des logiciels permettant de créer une bibliothèque de portes logiques enzymatiques, et de l'utiliser pour fournir des circuits implémentant une expression booléenne donnée.

Le premier programme, *NetGate* [55], va chercher dans des réseaux métaboliques existants, à partir d'une liste de tables de vérité d'opérateurs booléens, des portes logiques enzymatiques (des sous-réseaux métaboliques artificiels) calculant les fonctions logiques correspondant à ces tables de vérité. Ce programme va générer une bibliothèque de portes qui sera ensuite utilisable par d'autres applications. Le deuxième programme, *NetBuild*, va construire des circuits logiques enzymatiques implémentant une expression booléenne fournie par l'utilisateur à partir des portes logiques enzymatiques présentes dans une bibliothèque, telle que celles créées par NetGate.

Nous allons dans un premier temps détailler ces programmes, puis nous verrons plus succincte-

ment un troisième programme, *NetPlace*, qui met en uvre un algorithme de placement, qui fournit des représentations visuelles lisibles de réseaux métaboliques, plus particulièrement ceux trouvés par NetBuild.

3.1 Recherche des portes logiques enzymatiques : NetGate

La recherche de portes logiques moléculaires au sein de réseaux métaboliques existants est un problème complexe. En effet, un réseau métabolique est un ensemble de réactions connectées entre elles par leurs substrats, leurs produits et leurs catalyseurs. Ces réactions forment un ensemble dense et généralement fortement connecté. Les portes que nous devons trouver sont des sous-réseaux possédant certaines propriétés dynamiques. Or, deux sous-réseaux quasiment identiques (une interaction de différence) peuvent posséder des propriétés complètement différentes. Cela oblige à tester tous les réseaux indépendamment les uns des autres, quelle que soit leur similitude ou leur parenté. Pour trouver ces propriétés, nous allons spécifier des motifs qui poseront des contraintes sur les entrées et les sorties, ce seront les implémentations. Ces implémentations représentent des possibilités de portes logiques enzymatiques.

La première étape consiste à construire des sous-réseaux à partir du réseau métabolique original, puis pour chacun de ces sous-réseaux, construire toutes les implémentations potentielles de portes logiques (voir fig. 3.2).

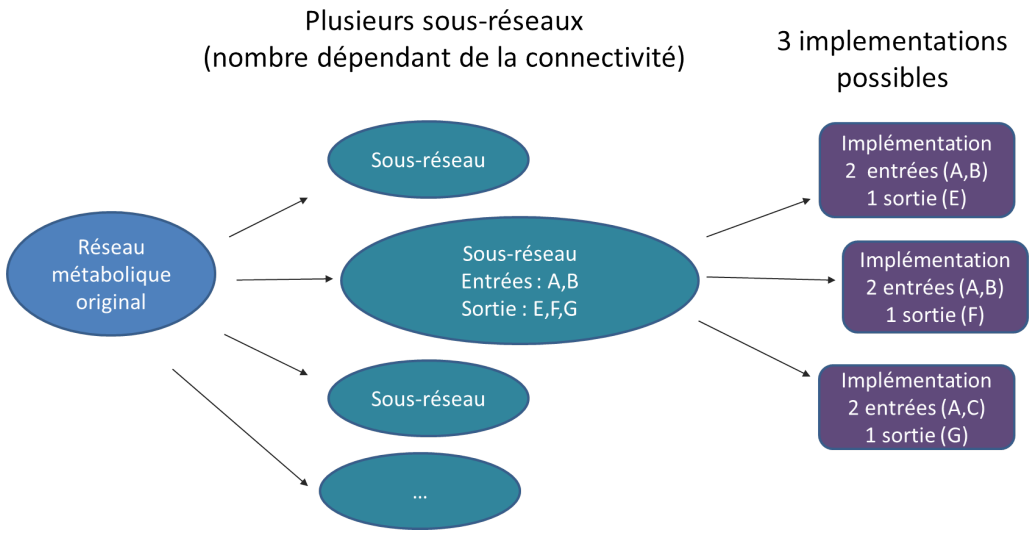


Figure 3.2 – Algorithme synthétique de la partie de NetGate qui liste des implémentations potentielles de portes logiques.

Dans une seconde étape, pour chaque implémentation, on effectue plusieurs évaluations qui vont nous donner des informations sur les propriétés de l'implémentation. A partir de ces informations, nous allons essayer d'identifier la fonction logique que l'on peut associer à l'implémentation pour obtenir une porte logique enzymatique valide (voir fig. 3.3).

L'encadré 1 présente l'algorithme général de NetGate sous une forme séquentielle : d'abord on énumère les sous-réseaux, puis toutes les implémentations, et enfin celles-ci sont testées. L'algorithme réel mixe ces opérations dans le temps : à chaque fois qu'un sous-réseau est construit, ses implémentations sont calculées, et chaque implémentation est aussitôt évaluée. La version que nous présentons ici est

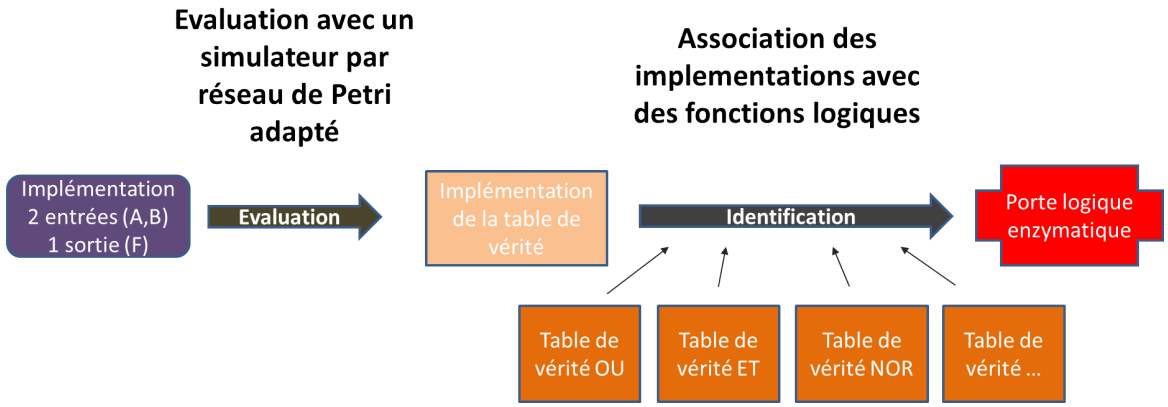


Figure 3.3 – Algorithme synthétique montrant le fonctionnement de NetGate pour l'évaluation des implémentations.

plus simple à exposer, sachant que celle réellement implémentée est meilleure en terme d'encombrement mémoire.

```
1  Fonction NetGate( Fichier listeFonctionLogique , Fichier fichierReseauMetabolique)
2      ReseauMetabolique reseauOriginal = lectureReseau( fichierReseauMetabolique ) ;
3      ListeFonctionLogique listeFoncLo = lectureFonctionsLogiques( listeFonctionLogique ) ;
4      ListePortesLogiquesEnzymatiques listePortesLogiquesEnzymatiques ;
5      ListeReseauMetabolique listeSousReseauxCandidat ;
6      creationSousReseauxCandidats( reseauOriginal , listeSousReseauxCandidat ) ;
7      pour tous les ReseauMetabolique sousReseau de listeSousReseauxCandidat faire
8          ListeImplementation listeImpl = creationImpl( sousReseau ) ;
9          pour tous les Implementation impl de listeImpl faire
10             FonctionLogique fl = evaluationImpl( impl , sousReseau ) ;
11             si fl appartient a listeFoncLo alors
12                 PorteLogiqueEnzymatique ple = creerPLE( impl , sousReseau , fl ) ;
13                 listePortesLogiquesEnzymatiques.ajouter( ple ) ;
14             fin
15         fin
16     fin
17 fin fonction
```

Algorithm 1: Algorithme implémenté dans NetGate (version séquentielle simplifiée)

NetGate va prendre en entrée les données suivantes :

— Un réseau métabolique au format SBML.

— Une liste de fonctions logiques à chercher. La syntaxe utilisée est très simple et le format est du type CSV : le nom de la fonction logique, son nombre d'entrées, son nombre de sorties, et sa

table de vérité ligne par ligne. Ce même format permet évidemment de spécifier des fonctions logiques à 3 entrées ou plus.
Par exemple, on spécifie une porte OU à deux entrées de la façon suivante :

GATE ;OR ;2 ;1 ;
;0 ;0 ;0 ;
;0 ;1 ;1 ;
;1 ;0 ;1 ;
;1 ;1 ;1 ;
; ; ; ;

NetGate va produire les différents résultats suivants :

- La liste des sous-réseaux trouvés ainsi qu'une liste succincte des portes trouvées pour chaque sous-réseau au format texte. Plus précisément, pour chaque porte sont indiqués :
  - Son type : AND, OR, etc.
  - la relation logique entrée/sorties (+ signifiant OU et . signifiant ET) : A+B=C, C.D=E, etc.
  - Les réactions impliquées (incluant les enzymes nécessaires à leur catalyse).
  - Les espèces moléculaires (métabolites et/ou enzymes) qui doivent avoir une concentration initiale non nulle pour que la porte puisse fonctionner correctement.

  Ces informations sont suffisantes pour être capable de construire *biochimiquement* la porte ainsi que ce qui est utile pour tester la compatibilité inter-portes si nous voulons l'utiliser dans un circuit logique enzymatique.
- Des informations générales sur le déroulement du programme :
  - Temps de calcul total
  - Répartition du temps de calcul entre les différentes phases
  - Nombre de sous-réseaux examinés
  - Nombre d'implémentations proposées
  - Paramètres du réseau d'entrée
  - Statistiques sur la répartition des portes logiques par enzymes et par taille.
- La liste des portes logiques enzymatiques trouvées au format CSV.

Cette liste de portes au format CSV est l'information la plus importante puisque ce sont ces données qui seront transmises ensuite à *NetBuild* pour la construction des circuits logiques complexes.

La figure 3.4 montre un exemple de porte logique (porte OUI) trouvée par NetGate. Cette porte transmet la valeur d'entrée booléenne portée par le métabolite B2 vers sa sortie, portée par le métabolite B5. L'apparente complexité de la porte vient du fait que la réaction principale, celle qui transforme B2 en B5 est catalysée par l'enzyme E6, qui catalyse une autre réaction, qui elle aussi a B2 comme substrat ! Cet exemple permet de montrer comment NetGate réussit à invalider cette réaction annexe qui sinon perturberait la réaction principale.

Le fichier de sortie, au format CSV, utilise une ligne de texte par porte trouvée, avec une syntaxe spécifique pour décrire les informations. Voici par exemple la ligne produite par NetGate qui décrit la porte OUI de la figure 3.4. Pour plus de clarté, nous avons découpé cette ligne de texte en morceaux, avec pour chacun sa signification.

- `E/S;1;B2;1;B5;YES;ENZ;1;E6;`
  La porte a une entree : B2 et une sortie : B5. La fonction logique est la copie (porte YES). Il

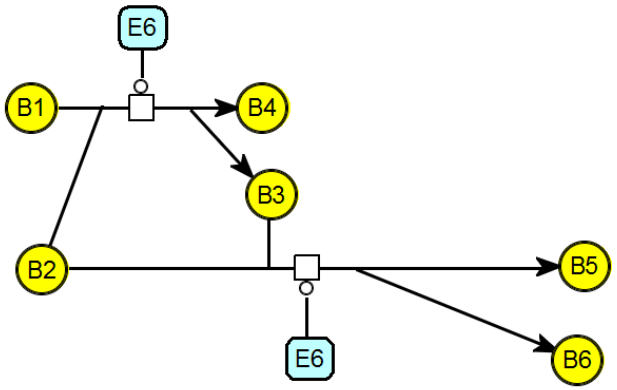


Figure 3.4 – Exemple de porte logique enzymatique découverte par NetGate à partir d'un réseau métabolique complexe.

n'y a qu'une enzyme pour cette porte : E6.

— `ELNEC;1;B3;ELABS;1;B1;`
Il y a un métabolite nécessairement en concentration initiale **non nulle** : B3. Il y a un métabolite nécessairement en concentration initiale **nulle** : B1

— `REACTIONS;2;`
Cette porte utilise 2 réactions.

— `REACBEGIN;re40;0;;SUBS;2;B1;;B2;;PROD;2;B4;;B3;;ENZY;1;E6;INHI;0;REACEND;`
La première, notée *re40* dans le réseau initial, a 2 substrats : B1 et B2, 2 produits B4 et B3. Elle est catalysée par 1 enzyme, E6, et cette réaction n'a pas d'inhibiteur.

— `REACBEGIN;re41;0;;SUBS;2;B2;;B3;;PROD;2;B6;;B5;;ENZY;1;E6;INHI;0;REACEND;`
La deuxième, notée *re41* dans le réseau initial, a 2 substrats : B2 et B3, 2 produits B6 et B5. Elle est catalysée par 1 enzyme, E6, et cette réaction n'a pas d'inhibiteur.

On peut remarquer qu'on aurait pu obtenir la même porte, composée des deux mêmes réactions, mais avec des conditions initiales différentes : B1 en concentration initiale **non nulle** et pas de contrainte sur B3. La porte aurait bien sur fonctionné, mais pas vraiment avec les mêmes caractéristiques : elle aurait consommé plus de B2 et elle aurait produit un métabolite supplémentaire : B4. Cette implémentation est moins performante que celle trouvée par NetGate car consommant plus de *signal* d'entree et produisant un métabolite *parasite*, elle imposerait beaucoup plus de contraintes lors de son utilisation avec d'autres portes dans un circuit complexe.

3.1.1 Création des sous-réseaux métaboliques candidats

Le réseau métabolique d'origine est la base de départ de notre algorithme. La première étape va consister à l'analyser et en extraire suffisamment d'informations pour construire une représentation qui facilitera ensuite l'application de l'algorithme proprement dit.

Un réseau métabolique peut être representé par un graphe décrivant les interactions entre des composants biologiques au sein d'un organisme vivant. Ce graphe a comme sommets des espèces biochimiques (métabolites) et comme arcs les réactions entre ces différents métabolites. Ces arcs sont aussi *qualifiés* par des catalyseurs (enzymes) et/ou des inhibiteurs.

La première étape consiste à créer tous les sous-réseaux connexes (réactions liées ensemble) existants dans le réseau métabolique d'origine. L'algorithme va commencer par analyser le réseau d'origine pour *comprendre* sa topologie et être ensuite capable de le décomposer en unités élémentaires de dif-

férents types. Ensuite ces unités seront recomposées pour former l'ensemble des sous-réseaux métaboliques candidats. Nous distinguons deux niveaux de décompositions du réseau métabolique d'origine : le niveau le plus large est la décomposition en interactions élémentaires, le plus fin est la décomposition en réactions élémentaires. Chaque décomposition représente un aspect du réseau métabolique.

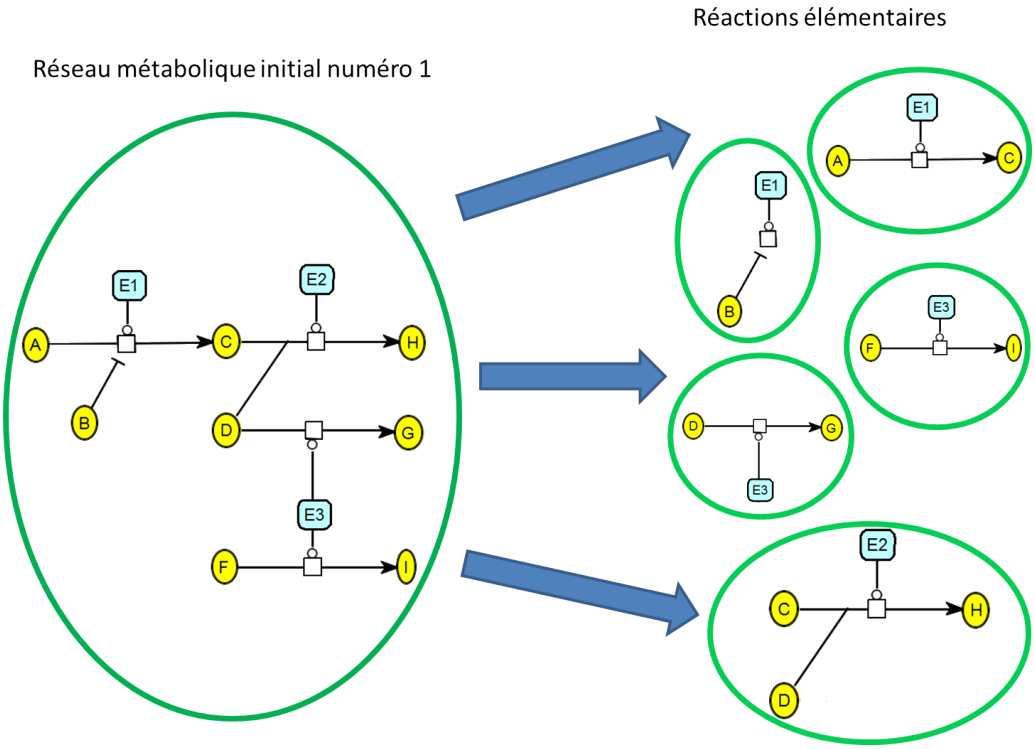


Figure 3.5 – Exemple de décomposition du réseau métabolique n°1 en réactions élémentaires.

Nous allons considérer un réseau métabolique comme un ensemble de flux de matière, associé à un système de règles. Les flux de matière représentent les différentes quantités de chaque espèce moléculaire présente dans le réseau et les règles représentent les réactions du réseau. Une réaction sera donc vue comme une transformation de flux de matières d'entrée en flux de matières de sortie. Cette transformation peut être effectuée lorsque des conditions de déclenchement sont présentes. Pour des réactions enzymatiques (cas majoritaire des réseaux métaboliques) alors une des conditions sera la présence de l'enzyme catalysant la réaction. Les quantités d'enzymes au sein de ces réseaux sont souvent fixes : homéostasie dans les organismes vivants, ou conditions initiales données dans des systemes artificiels. Les enzymes étant juste des catalyseurs, les quantités vont en général peu varier au cours du temps. On peut donc associer à un type d'enzyme, une liste de règles qui vont représenter toutes les transformations catalysées par cette enzyme. Nous appellerons *réaction élémentaire* une transformation catalysée, et *interaction élémentaire* la liste des règles associées à une enzyme.
Pour représenter les inhibitions, nous allons choisir le mécanisme par concurrence où c'est l'enzyme qui catalyse la réaction qui sera rendue inactive par formation d'un complexe avec l'inhibiteur. Ce complexe séquestrant tout ou partie du pool d'enzymes a comme effet de réduire la vitesse de la réaction, allant éventuellement jusqu'à l'inhiber complètement si toutes les enzymes sont séquestrées.

Sur la figure 3.5 on peut voir la décomposition du réseau métabolique d'origine (réseau n°1) en cinq réactions élémentaires, où tous les processus de transformation de flux de matière ont été isolés. Sur la figure 3.6, on peut voir que l'interaction élémentaire A) est composée de deux réactions élémentaires : l'inhibition et la réaction inhibée. Ces deux réactions représentent deux transformations de

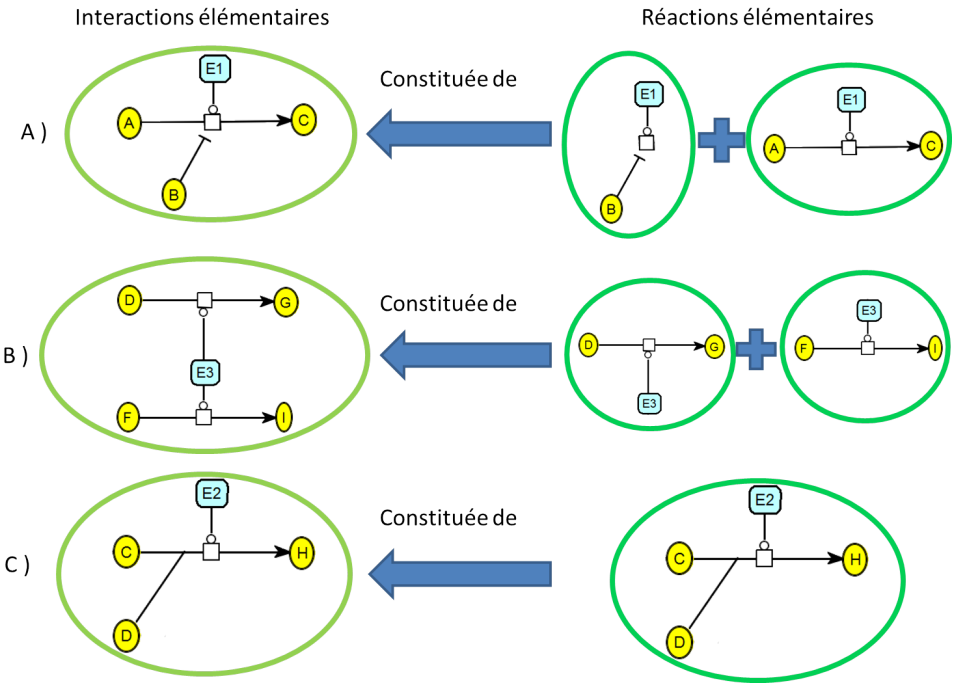


Figure 3.6 – Interactions élémentaires et réactions élémentaires

flux de matière : $E_1 + B \longrightarrow E_1B$ et $A \xrightarrow{E_1} C$. L'interaction élémentaire B) est elle aussi composée de deux réactions élémentaires : $D \xrightarrow{E_3} G$ et $F \xrightarrow{E_3} I$. L'interaction élémentaire C) n'est composée que d'une seule réaction élémentaire.

Le but de ces unités élémentaires est de pouvoir effectuer un inventaire plus pertinent qu'une simple liste de réactions. Les réactions élémentaires sont des transformations indivisibles, elles répertorient toutes les actions qui peuvent être effectuées dans le réseau métabolique. Les interactions élémentaires représentent des sous-réseaux indivisibles, elles serviront d'unité de base pour la construction des sous-réseaux candidats. Le but est d'éviter des séparations de réactions élémentaires qui pourraient induire des erreurs par la suite. Par exemple, si on veut utiliser la réaction élémentaire $F \xrightarrow{E_3} I$ et qu'il y a présence de E3, alors il sera alors nécessaire de considérer aussi la réaction $D \xrightarrow{E_3} G$ comme faisant partie du sous-réseau, car s'il y avait une concentration non nulle de D alors cette réaction serait effective.

3.1.1.1 Analyse du réseau métabolique d'origine

Le réseau initial nous est donné sous la forme dun document de type XML, plus précisément au format SBML. Ce format est spécialement dédié à la représentation de réseaux métaboliques. Il est composé de plusieurs listes : la liste des composants du réseau, métabolites et enzymes ainsi que la liste des réactions du réseau. Pour chaque composant et chaque réaction, il associe des identifiants uniques. Pour les réactions spécifiquement, il décrit tous les éléments impliqués ainsi que leur rôle : substrats, produits, catalyseur, inhibiteur.

Nous allons construire plusieurs représentations du réseau métabolique, chacune étant plus adaptée à un aspect particulier de la récupération d'information. La principale représentation du réseau métabolique va être matricielle, c'est celle que nous allons détailler. A chaque réaction élémentaire correspondra une ligne dans une matrice d'équations, les colonnes correspondant aux composants bio-

chimiques du réseau (enzymes et métabolites). Les substrats de la réaction seront notés -1 (ils sont consommés), les produits notés 1 (ils sont produits), les catalyseurs notés 2, les inhibiteurs notés -2 et les autres molécules présentent à la fois en entrée et en sortie sans être des enzymes seront notées 3 (voir table 3.1). Pour l'inhibition, on ne crée pas une espèce spécifique pour le complexe composé de l'inhibiteur et de l'enzyme, l'enzyme est considérée comme inactive.

Id Interaction élém.	Id Réaction élém.	E1	E2	E3	A	B	C	D	G	H	F	I
Inter1	Reac1	2	0	0	- 1	0	1	0	0	0	0	0
Inter1	Reac2	-2	0	0	0	-2	0	0	0	0	0	0
Inter2	Reac3	0	0	2	0	0	0	-1	1	0	0	0
Inter2	Reac4	0	0	2	0	0	0	0	0	0	-1	1
Inter3	Reac5	0	2	0	0	0	-1	-1	0	1	0	0

Table 3.1 – Matrice d'équations du réseau métabolique n°1. Elle est composée de cinq lignes correspondant aux cinq réactions élémentaires

Dans le cas des réactions réversibles, nous sommes obligés de considérer les deux sens car nous n'avons ni information sur la cinétique ni sur les concentrations des réactifs pour en déterminer *a priori* le sens. Nous les décomposons donc en deux réactions de sens opposé. Nous allons aussi dédoubler aussi les interactions auxquelles elles appartiennent (voir fig. 3.7 et fig. 3.8). C'est un moyen efficace de compenser le manque d'information, mais qui induit une augmentation de la complexité : chaque réaction réversible multipliant le nombre de sous-réseaux métaboliques candidats. Les interactions élémentaires résultantes sont dites *incompatibles* : elles ne pourront jamais se retrouver ensembles dans le même sous-réseau.

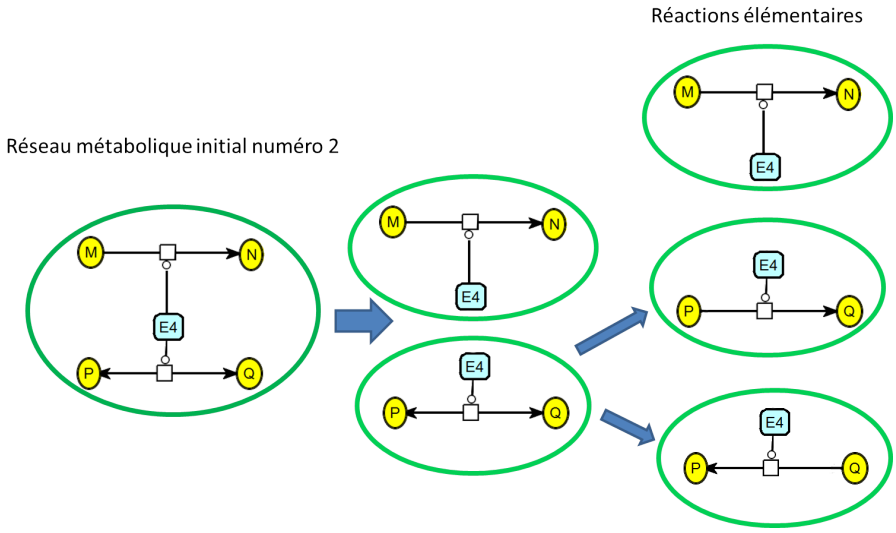


Figure 3.7 – Décomposition du réseau métabolique n°2 en réactions élémentaires. La réaction réversible va donner deux réactions élémentaires différentes, chacune allant dans un sens opposé.

3.1.1.2 Construction des sous-réseaux métaboliques candidats

Le but de cette étape de l'algorithme est de construire des sous-réseaux métaboliques qui seront de bons candidats pour la recherche de portes logiques enzymatiques au sein du réseau métabolique d'origine.

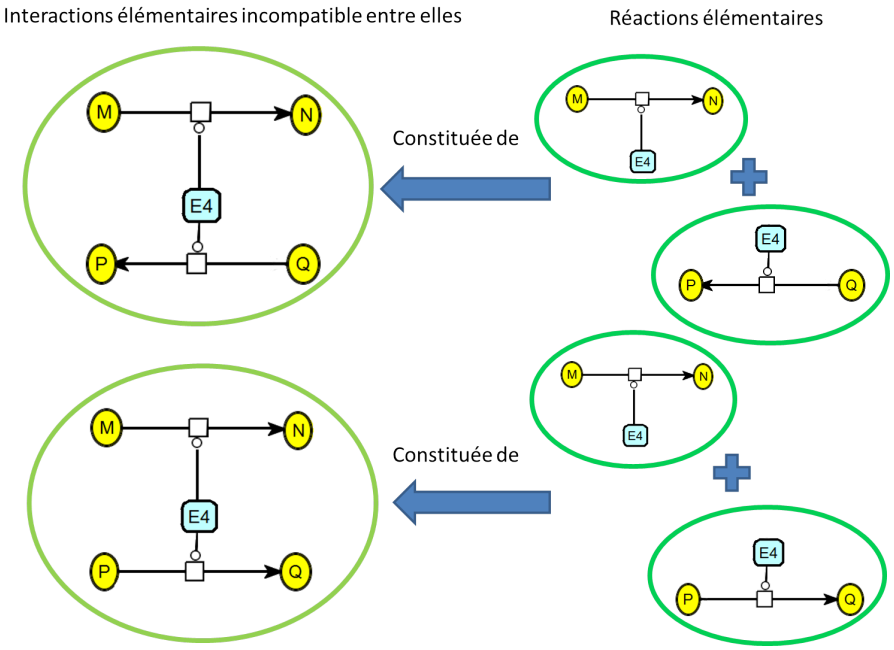


Figure 3.8 – Construction des interactions élémentaires du réseau métaboliques n°2. On peut constater qu'il y a formation de deux interactions élémentaires incompatibles, chacune représentant un sens de la réaction réversible.

Nous allons donc extraire de façon exhaustive les sous-réseaux à partir du réseau initial. Ces sous-réseaux devront être connexes, c'est à dire composés d'un ensemble d'interactions élémentaires liées entre elles. Deux interactions élémentaires seront dites liées si elles partagent un métabolite consommé ou produit.
Pour construire ces sous-réseaux, on va utiliser un algorithme glouton arborescent sur toutes les interactions élémentaires répertoriées à la phase précédente. On démarre le processus à partir d'une interaction quelconque, on va obtenir le premier sous-réseau qui sera de taille 1. Ensuite nous allons tenter d'agréger à ce sous-réseau d'autres interactions. Ces interactions devrons être *joignables*, c'est-à-dire qu'elles devront consommer ou produire un métabolite déjà présent dans le sous-réseau. De plus, ces réactions ne doivent pas comprendre d'interactions incompatibles (construites par la décomposition de réactions réversibles). Pour chacune de ces nouvelles interactions, on va créer un nouveau sous-réseau qui aura comme interactions toutes celles du sous-réseau considéré, ainsi que celle sélectionnée. Ensuite, on va ré-appliquer cet algorithme sur chacun de ces nouveaux sous-réseaux. Lorsqu'il n'y a plus de nouvelle interaction à ajouter, cela veut dire que le réseau métabolique d'origine a été atteint (modulo les réactions réversibles) toutes les interactions élémentaires y ont été ajoutées. Ce processus va être lancé à partir de chaque interaction élémentaire, pour former à chaque application un nouveau sous-réseau de taille 1 qui sera développé, et ainsi de suite.

Dans NetGate, le paramètre *limiteTailleSousReseau* permet de limiter la construction des sous-réseaux candidats à une taille choisie. Cette taille limite a été introduite pour deux raisons, d'une part les portes logiques enzymatiques implémentées par un réseau de grande taille ne seront pas très utiles en pratique (nous y reviendrons plus tard) et d'autre part la complexité de NetGate croît de façon exponentielle en fonction de la taille de ces sous-réseaux candidats.

Cette méthode va évidemment produire un grand nombre de sous-réseaux identiques, présents en plusieurs exemplaires, car on discrimine les sous-réseaux par l'ordre d'ajout des interactions élémentaires qui le composent. C'est ce qui pose problème car un réseau composé des interactions A, B, et

C par exemple, est en fait identique au réseau composé des interactions B, C, et A. Il va donc falloir modifier l'algorithme pour qu'il ne fournisse chaque sous réseau candidat qu'en un seul exemplaire. Pour cela on va exclure les interactions au fur et à mesure qu'on construit les sous-réseaux. Une fois que l'on a ajouté l'interaction I au sous-réseau X de taille n, pour créer le sous-réseau Y de taille $n+1$, cette interaction i ne pourra plus être ajoutée à d'autres sous-réseaux descendant de X (voir table 3.2).

Liste sous-réseaux avec doublons	Liste sous-réseaux sans doublons
Inter1	Inter1
Inter1,Inter2	Inter1,Inter2
Inter1,Inter2,Inter3	Inter1,Inter2,Inter3
Inter1,Inter3	Inter1,Inter3
Inter2	Inter2
Inter2,Inter1	
Inter2,Inter1,Inter3	
Inter2,Inter3	Inter2,Inter3
Inter3	Inter3
Inter3,Inter2	
Inter3,Inter2,Inter1	
Inter3,Inter1	

Table 3.2 – Tableau représentant la construction de sous-réseaux métaboliques candidats appliquée au réseau métabolique n°1

Deux fonctions sont utilisée pour implémenter cet algorithme : la première va créer les sous-réseaux de taille 1 à partir d'une interaction élémentaire (encadré 2), la deuxième, va développer chaque sous-réseaux jusqu'à la taille maximale autorisée (encadré 3).

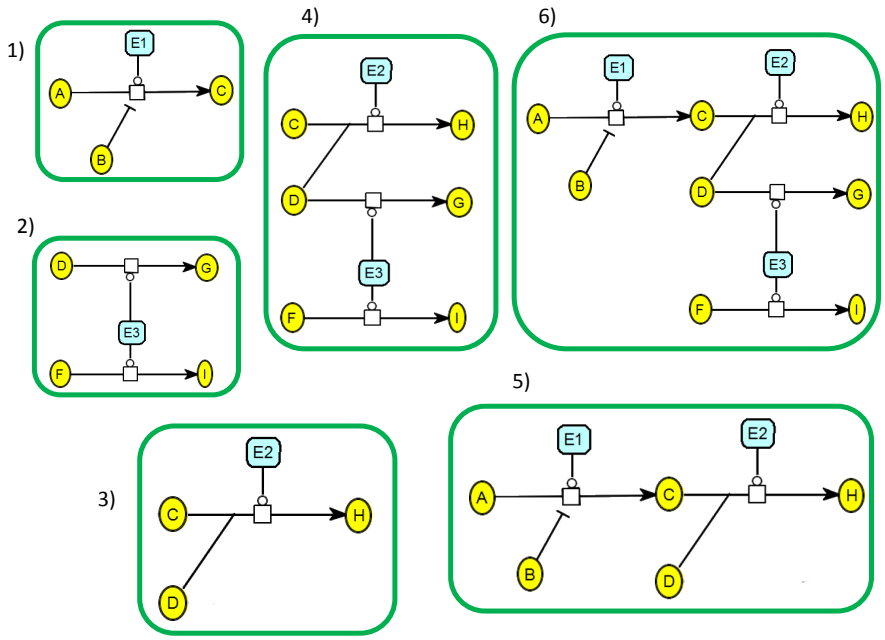


Figure 3.9 – Création des sous-réseaux à partir du réseau métabolique n°1.

Entrées: listeTotaleInteractions est la liste de toutes les interactions construites à partir du réseau métabolique original. listeSousRéseauxTotale est la liste de tous les sous-réseaux candidats, elle est vide au début de l'algorithme et remplie à la fin

```
1  Fonction creationSousResauxCandidats(ListeInteractions listeTotaleInteractions ,
     ListeReseauMetabolique listeSousRéseauxTotale)
2      pour tous les Interaction interaction de listeTotaleInteractions faire
3          ReseauMetabolique sr;
4          sr.ajouterInteraction( interaction );
5          /* Nous allons stocker le sous-réseaux candidat pour la phase de
              construction des implémentations                                  */
6          listeSousRéseauxTotale.ajouter( sr );
7          ListeInteractions listeInteractionDisponnible;
8          listeInteractionDisponnible.ajouter( listeTotaleInteractions );
9          listeInteractionDisponnible.retirer( interaction );
10         developperSousReseauCandidat( sr , listeInteractionDisponnible ,
             listeSousRéseauxTotale );
11     fin
12 fin fonction
```

Algorithm 2: Fonction de création des sous-réseaux initiaux de taille 1

Pour donner un exemple parlant, nous avons appliqué l'algorithme de création de sous-réseaux au réseau métabolique n°1 (voir fig. 3.9). On a obtenu 6 sous-réseaux : les sous-réseaux 1, 2, et 3 sont des interactions élémentaires, ils sont donc également les premiers sous-réseaux candidats à être trouvés. Le réseau 4 est une combinaison de 2 et 3, le réseau 5 est une combinaison de 1 et 3. Enfin, le réseau 6 est une combinaison de 4 et 1. Le réseau 6 aurait pu être trouvé de différentes façons (c'est en fait le réseau d'origine).

3.1.2 Recherche des fonctions logiques associées aux sous-réseaux candidats

L'étape suivante de l'algorithme est la recherche des fonctions logiques associées aux sous-réseaux candidats. À l'étape précédente nous avons créé les sous-réseaux candidats, ils constituent des implémentations potentielles de portes logiques enzymatiques. Comme les portes logiques enzymatiques possèdent un nombre fixé d'entrées et une seule sortie, un même sous-réseau va pouvoir implémenter un grand nombre de portes possibles. On va commencer par énumérer et répertorier ces portes, ensuite chacune sera évaluée pour en déterminer sa table de vérité, qui sera comparée à celles qui ont été fournies en argument au programme. Enfin, la description complète des portes logiques enzymatiques validées sera stockée. Quand l'algorithme trouve plusieurs réseaux implémentant la même porte (mêmes métabolites d'entrée, même métabolite de sortie et même fonction logique), leurs descriptions sont triées pour ne conserver que la plus optimisée.

```
1  Fonction developperSousReseauCandidat(ReseauMetabolique sr, ListeInteractions
     listeInteractionDisponnible , ListeSousReseau listeSousRéseauxTotale)
2    /* Nous cherchons les interactions élémentaires qui peuvent se lier au
        sous-réseau actuel                                                  */
3    ListeInteractions listeInteractionPossible;
4    /* Cette liste va récupérer les interactions utilisées pour
        l'optimisation                                                      */
5    ListeInteractions listeInteractionUtilisées;
6    pour tous les Interaction interactionSr de sr.interactionInternes faire
7      pour tous les Interaction interactionDisp de listeInteractionDisponnible faire
8        si interactionSr est liée à interactionDisp alors
9          listeInteractionPossible.ajouter( interactionDisp );
10       fin
11     fin
12   fin
13   /* Pour chaque interaction qui peut être liée, nous allons construire un
        nouveau sous-réseau candidat                                        */
14   pour tous les Interaction interactionPos de listeInteractionPossible faire
15     listeInteractionDisponnible.retirer( interactionPos );
16     listeInteractionUtilisées.ajouter( interactionPos );
17     ReseauMetabolique sr;
18     sr.ajouterInteractions( sr.interactionInternes );
19     sr.ajouterInteraction( interactionPos );
20     /* Nous allons stocker le sous-réseau candidat pour la phase de
          construction des implémentations                                  */
21     listeSousRéseauxTotale.ajouter( sr );
22     /* Si la taille du nouveau sous-reseau est en-dessous de la taille
          limite, nous continuons a le faire grossir                        */
23     si sr.interactionInternes.taille < limiteTailleSousReseau alors
24       developperSousReseauCandidat( sr , listeInteractionDisponnible );
25     fin
26   fin
27   pour tous les Interaction interactionUti de listeInteractionUtilisées faire
28     listeInteractionDisponnible.ajouter( interactionUti );
29   fin
30 fin fonction
```

Algorithm 3: Fonction de développement des sous-réseaux candidats

3.1.2.1 Énumération des implémentations

Chaque sous-réseau candidat possède un nombre $nb_{entrée}$ d'entrées ainsi que un nombre nb_{sortie} de sorties. Pour un sous-réseau métabolique candidat, les entrées d'une porte potentielle seront sélectionnées parmi les métabolites qui sont uniquement consommés (et donc pas produits en interne). De façon analogue, les sorties seront sélectionnées parmi les métabolites qui sont uniquement produits (et pas consommés en interne). Il va falloir choisir parmi les $nb_{entrée}$ entrées du sous-réseau les $nb_{entréePorte}$ qui correspondront à celles de la porte cherchée ($nb_{entréePorte} \leq nb_{entrée}$) ; de même, une seule sortie parmi les nb_{sortie} du réseau sera la sortie de la porte potentielle. Les entrées du réseau qui ne sont pas choisies comme entrées de la porte sont des *extra-entrées* auxquelles seront affectées des valeurs booléenes constantes, assurant un fonctionnement correct (voir fig. 3.10).

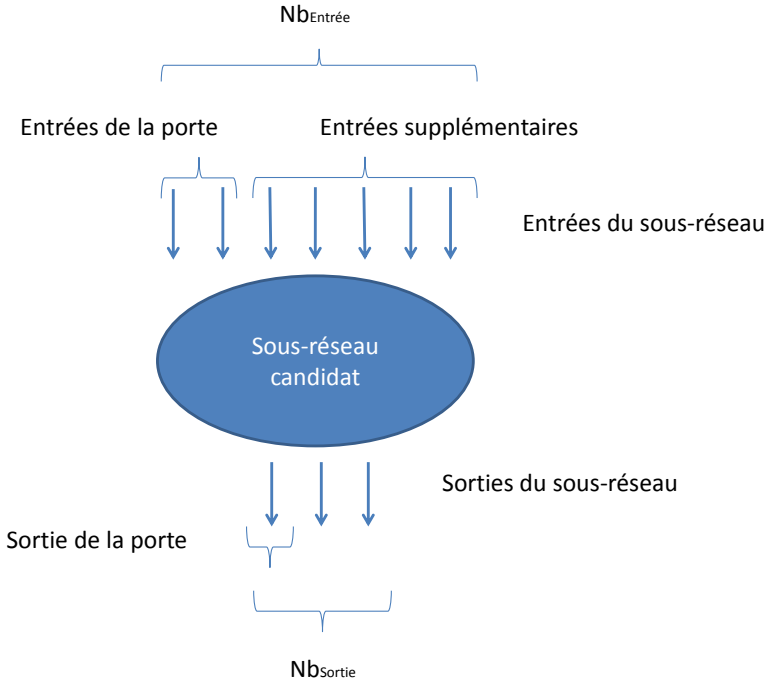


Figure 3.10 – Structure d'une implémentation dans un sous réseau candidat. L'implémentation d'une porte à deux entrées laisse cinq autres entrées supplémentaires libres : les *extra-entrées*.

Le nombre d'implémentations total est le produit du nombre de sorties différentes par le nombre de combinaison des entrées de la porte parmi celles du réseau (indépendamment de l'ordre pour les fonctions logiques commutatives).

Pour les portes logiques non commutatives (par exemple NON-ET où l'une des entrées est inversée) on va ajouter dans la liste des portes à chercher (argument d'entrée du programme) plusieurs exemplaires de ces portes avec toutes les combinaisons d'ordre possible des entrées. Par exemple, pour la porte NON-ET à deux entrées, on cherchera à la fois la porte qui inverse la première entrée et celle qui inverse la seconde entrée.

Le nombre d'implémentations possibles par sous-réseau sera donc de : $\binom{nb_{entrée}}{nb_{entréePorte}} * nb_{sortie}$

Une fois que l'on a choisi les entrées et la sortie pour l'implémentation à tester, il va falloir décider que faire des entrées restantes du sous-réseau (les *extra-entrées*). Leur valeur, *vrai* ou *faux* peut avoir une

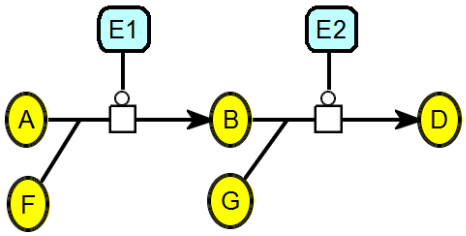


Figure 3.11 – Schéma représentant un sous-réseau candidat nécessitant la prise en compte des *extra-entrées*. On peut voir que la porte logique ET, qui prend comme entrées A et F et comme sortie D, nécessite qu'il y ait présence de G pour fonctionner.

influence cruciale sur la fonction que calcule l'implémentation. Par exemple sur la figure 3.11, si on considère la porte ET qui prend comme entrées A et F et comme sortie D, nous pouvons voir qu'il est nécessaire qu'il y ait présence de G pour que la porte fonctionne. Ces *extra-entrées* ne peuvent donc pas être ignorées, mais il est impossible de savoir à l'avance quelle valeur leur donner pour que des portes correspondant à celles cherchées soient trouvées. On va donc considérer toutes les combinaisons de valuations possibles pour les *extra-entrées*.

Comme ll y a : $2^{(nb_{entrée} - nb_{entréePorte})}$ combinaisons de valeurs pour les extra-entrées, le nombre d'implémentations potentielles par sous-réseau devient donc :

$$\binom{nb_{entrée}}{nb_{entréePorte}} * nb_{sortie} * 2^{(nb_{entrée} - nb_{entréePorte})}$$

Ce nombre pouvant rapidement devenir très grand, nous avons cherché à optimiser l'énumération des implémentations. Jusqu'à présent, nous avons ignoré la topologie interne des sous-réseaux ; nous allons maintenant en tirer des informations utiles pour restreindre le champ de recherche, sans pour autant perdre des implémentations possibles.

Pour chaque sortie d'un sous-réseau, nous allons considérer la liste des composants qui peuvent influer sur sa valeur. Cette liste, que nous appellerons *liste des prédécesseurs*, sera ensuite croisée avec la liste des entrées du sous-réseau. Le résultat fournira une nouvelle liste d'entrées (de taille plus petite ou égale) qui agissent spécifiquement sur chaque sortie. On la substituera à la liste normale des entrées du sous-réseau lors de l'énumération des implémentations possibles, réduisant de ce fait le temps de recherche.

Pour déterminer quels sont les éléments qui peuvent influer sur une sortie, on va partir d'un métabolite de sortie MT_{sortie} dont on veut trouver les prédécesseurs. Puis on va chercher toutes les réactions élémentaires RE_i qui produisent MT_{sortie}. Tous les métabolites ($MT_{predecesseurDirect}$) qui sont consommés par ces réactions seront ajoutés à la liste des prédécesseurs ($listePredecesseurs$). Ensuite on va chercher toutes les réactions élémentaires qui consomment $MT_{predecesseurDirect}$ et ajouter les autres métabolites ($MT_{predecesseurIndirect}$) consommés par ces réactions à la liste des prédécesseurs $listePredecesseurs$.

Pour obtenir la liste complète des prédécesseurs d'une sortie S, on applique donc cette procédure sur S puis on la ré-applique récursivement à tous les éléments de la liste des prédécesseurs jusqu'à ce que cette liste de prédécesseurs ne change plus. Les prédécesseurs directs sont les substrats des réactions élémentaires qui vont produire le métabolite représentant la sortie S. Les prédécesseurs indirects sont

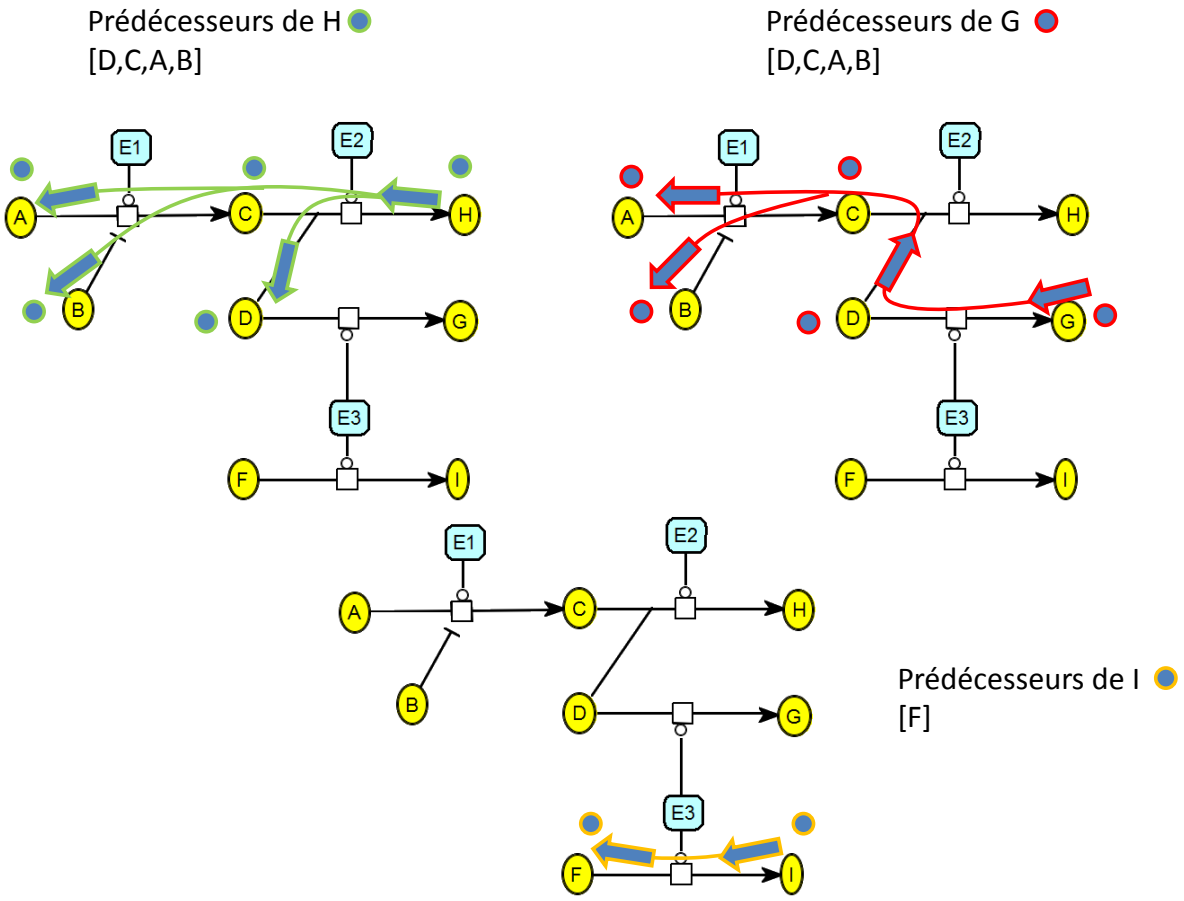


Figure 3.12 – Recherche des prédécesseurs des 3 sorties du sous-réseau candidat 6).

des substrats dont la présence peut induire des phénomènes de concurrence pouvant influencer la production du métabolite de sortie S.

Nous avons recherché, par exemple, les prédécesseurs sur le sous-réseau candidat 6) de la figure 3.9. On peut voir trois diagrammes, un pour chacune des trois sorties H, G et I de ce sous-réseau. Pour la sortie H, on obtient C et D comme prédécesseurs directs, puis en repartant de C, on ajoute A et B comme autres prédécesseurs directs. Pour la sortie G, on obtient D comme prédécesseur direct et C comme prédécesseur indirect (puisque C influe indirectement sur la concentration de D car ils sont substrats de la même réaction catalysée par E2), puis à partir de C, on ajoute A et B comme prédécesseurs indirects. La sortie I n'a qu'un prédécesseur (direct) F (voir fig : 3.12).

Nous allons maintenant calculer sur cet exemple le gain apporté par l'utilisation des listes de prédécesseurs pour l'obtention des implémentations potentielles de portes logiques.

Cas standard : $\binom{2}{4} * 2^{(4-2)} = 24$ implémentations possibles pour chacune des 3 sorties.

Utilisation des listes de prédécesseurs par sortie :

— Pour H : $\binom{2}{3} * 2^{(3-2)} = 6$

— Pour G : $\binom{2}{3} * 2^{(3-2)} = 6$

— Pour I : $\binom{2}{1} * 2^{(1-2)} = 0$

On peut constater que la version optimisée (12 implémentations potentielles), est beaucoup plus efficace que la version non optimisée (72 implémentations potentielles). L'utilisation des listes de prédécesseurs permet meme, sur cet exemple, de supprimer l'examen de toutes les implémentations qui auraient I comme sortie. On peut voir sur la table 3.3 les 12 implémentations trouvées par l'algorithme appliqué au sous-réseau candidat 6).

Sortie	Entrées disponibles	Combinaisons possibles	Extra -entrées	Combinaisons possibles extra -entrées	Implémentations complètes
H	A,B,D	A,B	D	D = 1	Entrée : A, B. Sortie : H et D = 1
H	A,B,D	A,B	D	D = 0	Entrée : A, B. Sortie : H et D = 0
H	A,B,D	A,D	B	B = 1	Entrée : A, D. Sortie : H et B = 1
H	A,B,D	A,D	B	B = 0	Entrée : A, D. Sortie : H et B = 0
H	A,B,D	B,D	A	A = 1	Entrée : B, D. Sortie : H et A = 1
H	A,B,D	B,D	A	A = 0	Entrée : B, D. Sortie : H et A = 0
G	A,B,D	A,B	D	D = 1	Entrée : A, B. Sortie : G et D = 1
G	A,B,D	A,B	D	D = 0	Entrée : A, B. Sortie : G et D = 0
G	A,B,D	A,D	B	B = 1	Entrée : A, D. Sortie : G et B = 1
G	A,B,D	A,D	B	B = 0	Entrée : A, D. Sortie : G et B = 0
G	A,B,D	B,D	A	A = 1	Entrée : B, D. Sortie : G et A = 1
G	A,B,D	B,D	A	A = 0	Entrée : B, D. Sortie : G et A = 0

Table 3.3 – Tableau énumérant les implémentations de portes potentielles du sous-réseau candidat 6).

3.1.2.2 Calcul des rapports de concentrations

Selon les concentrations des métabolites d'entrée, une porte logique enzymatique peut théoriquement implémenter plusieurs fonctions booléenes différentes, nous l'avons montré dans le chapitre 2. Lorsque la porte est utilisée dans un circuit, elle ne doit avoir qu'une seule fonction booléene déterminée à l'avance et stable dans tous les cas d'utilisation.

La fonction booléene que calcule une porte logique enzymatique dépend, entre autres, des intervalles de concentration des métabolites d'entrée. Plus ces intervalles sont larges, plus la fonction logique sera considérée comme étant *robuste*. Notre but est donc de trouver des portes logiques enzymatiques calculant des fonctions booléenes de la façon la plus robuste possible. Un bon moyen de connaitre la robustesse d'une porte enzymatique est de construire son diagramme fonctionnel comme indiqué dans le chapitre 2.

Cela nécessite donc pour chaque implémentation possible un très grand nombre de calculs. En effet, si nous voulons un degré de précision intéressant, il faut beaucoup de points dans le diagramme, et donc un grand nombre d'évaluations. En pratique, cela signifie qu'il est irréaliste de calculer un diagramme fonctionnel pour chaque implémentation. On va donc devoir trouver un autre moyen de s'assurer de la robustesse de la fonction booléene que calcule une porte logique enzymatique.

Lors de son fonctionnement, NetGate essaye d'identifier la table de vérité qui correspond à chaque implémentation potentielle. Pour cela, on doit évaluer cette implémentation, c'est-à-dire effectuer des simulations qui vont permettre de trouver l'évolution dans le temps de la concentration du produit de sortie en fonction des concentrations des produits d'entrée. La méthode utilisée pour effectuer ces

simulations nécessite de convertir les valeurs booléennes en valeurs discrètes décimales qui représenterons les concentrations de métabolites. A la fin de la simulation, le résultat sera aussi une valeur discrète décimale qui devra être reconvertie en valeur booléenne.

Pour effectuer ces diverses conversions, nous allons avoir besoin de définir des seuils séparant les valeurs booléennes *faux* et *vrai*. Ce sont ces seuils qui vont déterminer la fonction booléenne que calcule l'implémentation testée. Ces seuils sont propres à chaque implémentation et donc à chaque porte logique enzymatique. Nous pouvons considérer, comme pour la construction des diagrammes de fonction, que le seuil sur les sorties est fixe et que nous avons uniquement à déterminer des seuils pour les entrées.

Pour résoudre ce problème, nous allons utiliser une heuristique qui va fournir dans une large majorité des cas des seuils de conversion des entrées permettant d'obtenir la fonction logique la plus *robuste*. Pour cela, nous allons utiliser une liste de modèles de sous-réseaux spécifiques (*patterns* de porte), chacun associé à une fonction booléenne déterminée (voir fig. 3.13). Ces *patterns* sont artificiels en ce sens que nous avons décidé qu'ils soient reconnus comme implémentant les fonctions booléennes choisies (en analysant leur diagramme fonctionnel pour s'assurer qu'elles sont effectivement les plus robustes). Nous allons programmer l'heuristique de façon à ce qu'elle reconnaisse toutes ces associations correctement, et qu'elle extrapole pour les autres sous-réseaux.

Voici comment l'heuristique fonctionne : pour chaque implémentation, on va parcourir le sous-réseau, depuis la sortie en le remontant à travers les réactions jusqu'aux entrées, en appliquant des règles qui permettront d'inférer les seuils des entrées. Le seuil de sortie, S_{sortie}, a été fixé arbitrairement à 10. Nous voulons calculer le seuil de tous ses prédécesseurs. Pour calculer le seuil S_i d'un métabolite MT_i présent dans la liste des prédécesseurs de la sortie du sous-réseau, nous allons appliquer les règles suivantes : (l'ordre de parcours est similaire à un parcours en profondeur en prenant la sortie de l'implémentation comme racine)

- **Règle 1 :** Si tous les successeurs (les métabolites qui sont produits par la consommation de MT_i) ont leur seuils S_j calculés, on prend pour chaque réaction où MT_i est consommé, le maximum entre tous les métabolites produits par la réaction. La somme de ces maximums donne S_i.
- **Règle 2 :** Si un seuil n'est pas calculé (quand c'est un prédécesseur indirect, et donc certains de ses successeurs ne sont pas dans les prédécesseurs de la sortie), alors S_i est le maximum des S_k où les MT_k sont les métabolites consommés dans la même réaction que MT_i et dont les seuils sont calculés.
- **Règle 3 :** Si un seuil n'est pas calculé (quand il y a une boucle dans le sous-réseau), alors le métabolite est mis à la fin de la liste de ceux dont il faut calculer le seuil. Si tous les métabolites qui attendent d'avoir leur seuil calculé sont eux aussi incalculables, alors le système est bloqué. Pour résoudre le problème, le métabolite ayant le plus de prédécesseurs calculés est choisi et son seuil est calculé avec la règle 1, en supposant que tous ses successeurs non calculés ont un seuil à 0.

3.1.2.3 Évaluation des implémentations potentielles

Chaque implémentation potentielle va être évaluée (simulation à gros grain), pour identifier si elle implémente effectivement l'une de fonctions booléennes dont la table de vérité a été passée en argument à NetGate.

Pour cela, on va tester le comportement de l'implémentation (valeur booléenne de sa sortie) pour toutes les combinaisons possibles des valeurs booléennes de ses entrées. Les entrées du sous-réseau candidat seront initialisées avec successivement chacune de ces combinaisons, puis on simulera le sous-

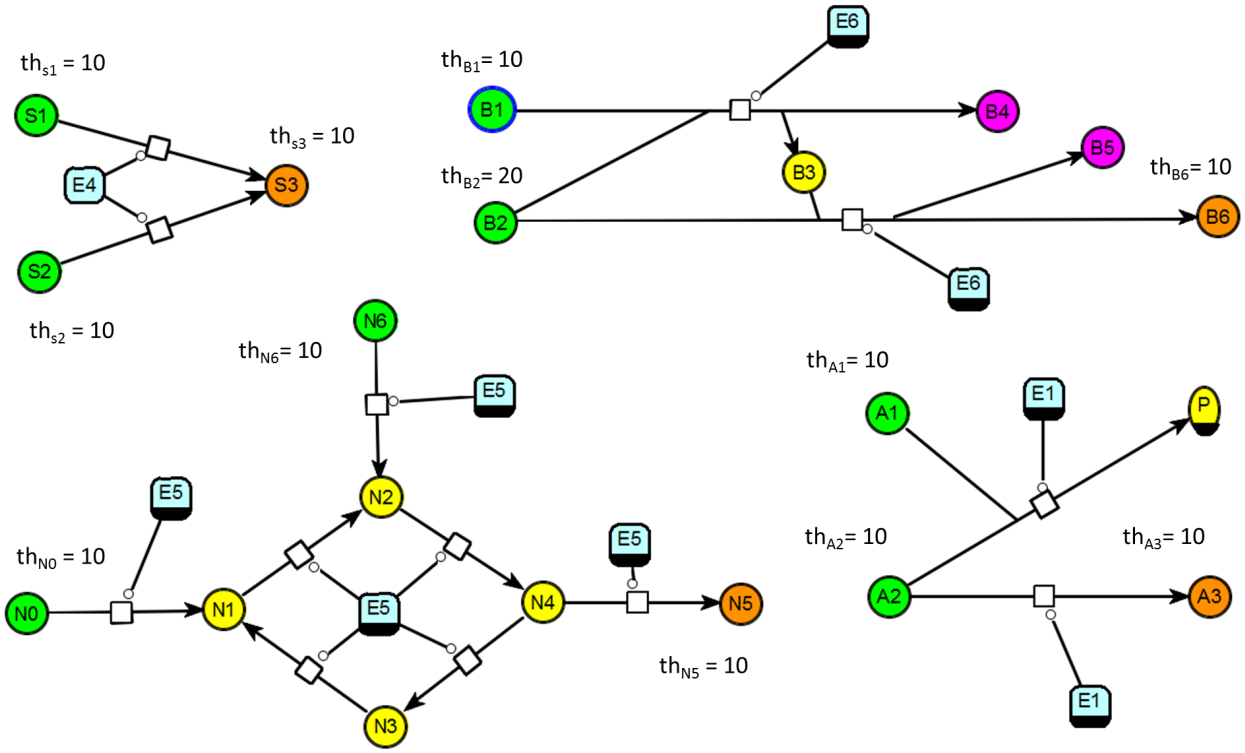


Figure 3.13 – Exemple de modèles particuliers utilisés pour la construction de l'heuristique de calcul des seuils.

réseau pour obtenir la valeur de sortie correspondante (chaque simulation est appelée *évaluation*).

On obtiendra finalement la table de vérité de l'implémentation qu'on comparera ensuite aux tables de vérité des fonctions logiques cherchées. Par exemple, sur la figure 3.14, on a pris le sous-réseau candidat 6) et une de ses implémentations : entrées A et B, sortie H, et *extra-entrée* D = 1. L'implémentation ayant deux entrées on a quatre combinaisons et donc quatre évaluations à effectuer. Ces évaluations vont nous donner chacune une valeur pour la sortie H et donc la table de vérité de l'implémentation.

Nous avons choisi d'utiliser un système de simulation à gros grain, de type *réseau de Petri*, appliqué aux réseaux métaboliques [56]. Les réseaux de Petri sont des graphes orientés bipartis composés de deux types de nœuds : les *places*, qui vont contenir des jetons, et les *transitions*, qui vont contrôler le passage des jetons des places d'entrée vers les places de sortie (voir fig. 3.15). Pour qu'une transition *s'enflamme* (se déclenche) il faut que toutes ses places d'entrées possèdent au moins un jeton. Quand elle est activée, la transition va consommer un jeton dans chacune de ses places d'entrée et produire un jeton dans chacune de ses places de sorties.

La transformation d'un réseau de réactions enzymatiques en réseau de Petri est assez directe : chaque substrat et produit sera représenté par une place et chaque réaction représentée par une transition. Les jetons représenteront les quantités de chaque métabolite dans le système. Les quantités de métabolites seront considérées sans dimension (exprimées dans une unité virtuelle arbitraire).

Nous allons ajouter quelques propriétés pour que ce formalisme puisse répondre à nos besoins :

— Les réactions enzymatiques nécessitent la présence d'enzymes pour fonctionner, la quantité d'enzymes pouvant varier à cause des inhibitions. Nous allons définir un arc d'un nouveau type

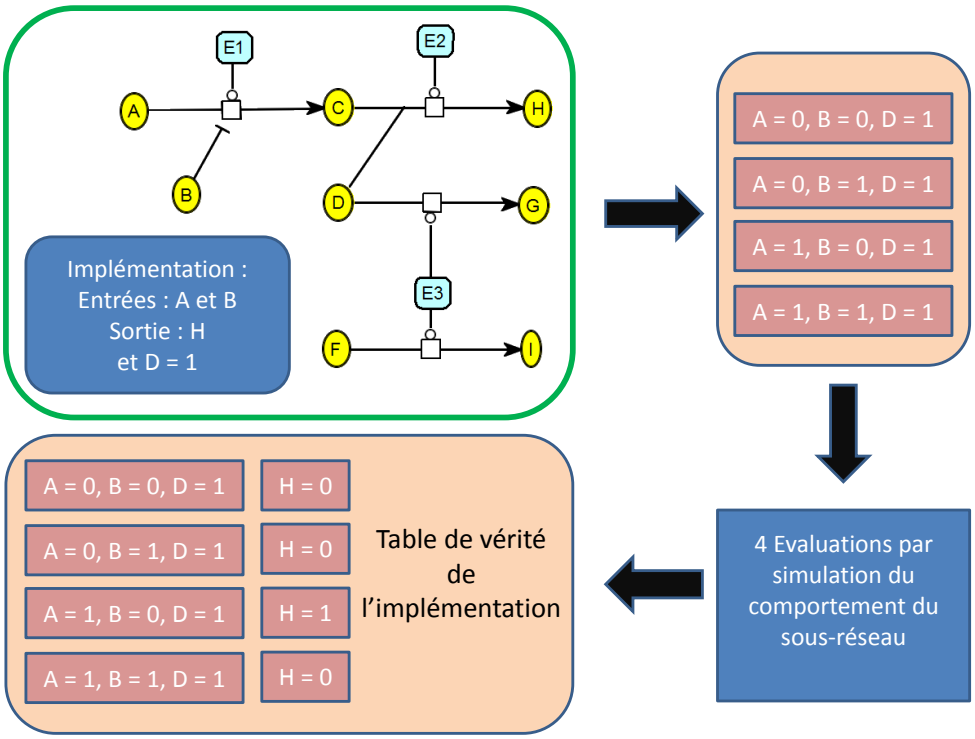


Figure 3.14 – Illustration du processus d'évaluation d'une implémentation du sous-réseau candidat 6)

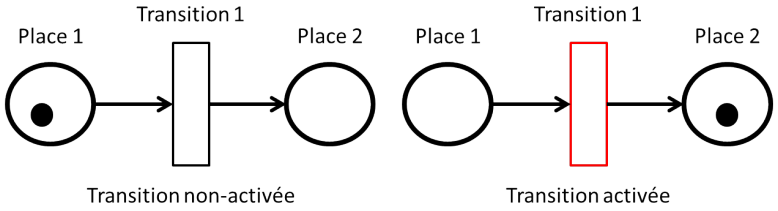


Figure 3.15 – Fonctionnement d'un réseau de Petri. On distingue les deux types de nœuds : les transitions et les places. La transition est activable car sa place d'entrée a (au moins) un jeton. Quand elle est activée le jeton est consommé dans *Place 1* et un jeton est produit dans *Place 2*.

pour représenter ce fait. Ces arcs connecteront une place représentant l'enzyme à la transition représentant la réaction catalysée par cette enzyme. Il faut donc qu'au minimum un jeton soit présent dans la place de l'enzyme pour que la transition puisse être déclenchée.

— Nous utiliserons une simulation de type déterministe, simple à implémenter et ne changeant pas fondamentalement le résultat. Pour pallier le non-déterminisme inhérent aux réseaux de Petri, nous considérerons que si plusieurs transitions peuvent être déclenchées, nous les déclencherons toutes en même temps (même si le nombre de jetons présents dans la place d'entrée des transitions concurrentes n'est pas suffisant pour alimenter toutes ces transitions). Il est donc possible que le nombre de jetons dans une place devienne négatif et dans ce cas les transitions correspondantes ne pourront plus être déclenchées tant qu'il ne sera pas redevenu positif.

Sur la figure 3.16 on peut voir les deux sous-réseaux candidats 1) et 3) convertis en réseau de Perti. Le sous-réseau candidat 3) est implémenté par deux transitions, une pour l'inhibition et l'autre pour la

transformation de A en C. Nous pouvons voir que la transformation en réseau de Petri est identique à la création des réactions élémentaires, à chaque réaction élémentaire correspond une transition. Nous avons appliqué la même conversion avec le plus gros des sous-réseaux candidats, le 6) (voir fig.3.17).

Pour effectuer une simulation on va devoir convertir chaque combinaison booléenne d'entrée de l'implémentation en un nombre de jetons correspondant, mis dans les places d'entrée. Cette conversion, pour l'espèce moléculaire A, s'effectue de la façon suivante :

— Si A est une entrée à *vrai* on mettra X jetons dans la place A.

— Si A est une *extra-entrée* à *vrai* on mettra 100000 jetons dans la place A.

— Si A est une enzyme présente on mettra 1 jeton dans la place A.

— Quel que soit le type de A, quand sa valeur booléenne est *faux*, la place A sera vide.

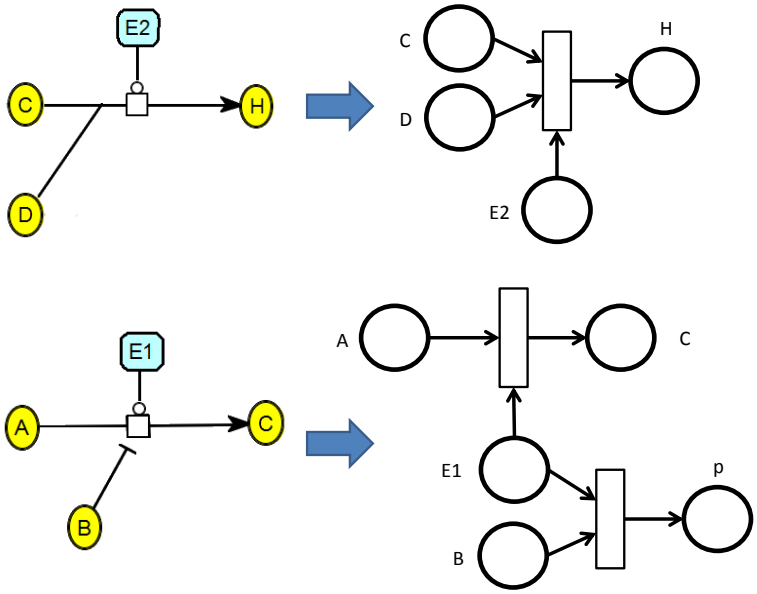


Figure 3.16 – Conversion des sous-réseaux candidats 1) et 3) en réseaux de Petri.

La valeur X peut différer selon l'entrée, l'implémentation et le sous-réseau candidat. Elle est fixée à l'aide des seuils calculés par l'heuristique présentée dans la partie précédente. Sur la table 3.4, on peut voir le déroulement complet d'une simulation pour les valeurs booléennes A = *vrai*, B = *faux* de l'implémentation montrée sur la figure 3.17, pour le sous-réseau candidat 6 (sortie : H, *extra-entrée* D = *vrai*). Les valeurs booléennes des paramètres sont converties en valeurs entières : ici il y a 10 jetons dans la place A, 1 jeton dans les place E_1, E_2, E_3, et un grand nombre de jetons, 1000000, dans la place D.

La simulation est divisée en pas (*rounds*) qui seront des unités de temps, durant chacune desquelles on tente de déclencher toutes les transitions possibles, une fois. Plusieurs transitions sont déclenchées au fur et à mesure du déroulement de la simulation. La place C se remplit alors que la place A se vide, ensuite les places C et D se vident et la place H se remplit. La simulation s'arrête lorsqu'il n'y a plus aucune transition déclenchée pendant un round entier.

Pour éviter que la simulation ne boucle indéfiniment quand le sous-réseau candidat contient un cycle, le nombre maximum de rounds pour une simulation est borné par la somme de la taille du sous-réseau candidat et du nombre de jetons de la plus fournie des entrées. Ce nombre est suffisamment

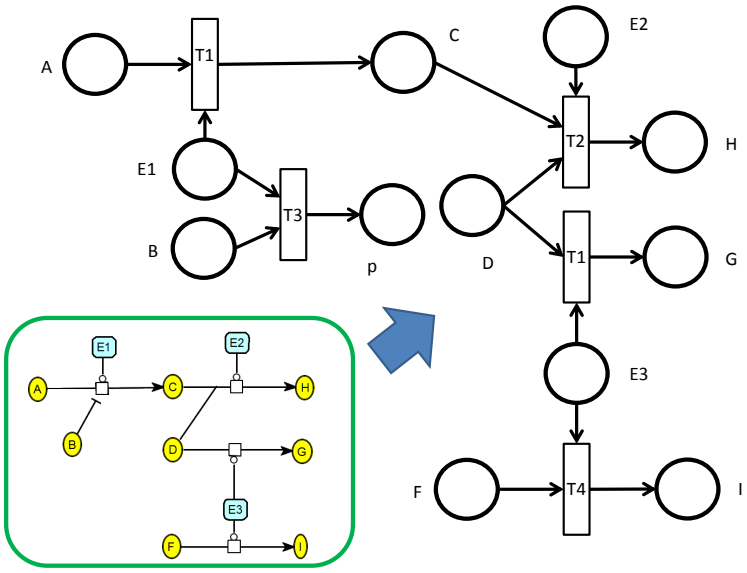


Figure 3.17 – Conversion du sous-réseau candidat 6) en réseau de Petri.

grand pour permettre d'obtenir le résultat final du sous-réseau candidat puisque tous les jetons auront eu le temps de parcourir l'intégralité du réseau.

Round	E1	E2	E3	A	B	C	D	H	G	F	I	Réaction Élémentaire Appliquée
1	1	1	1	10	0	0	1000000	0	0	0	0	
2	1	1	1	9	0	1	1000000	0	0	0	0	Reac1
3	1	1	1	8	0	1	999999	1	0	0	0	Reac1,Reac5
4	1	1	1	7	0	1	999998	2	0	0	0	Reac1,Reac5
5	1	1	1	6	0	1	999997	3	0	0	0	Reac1,Reac5
6	1	1	1	5	0	1	999996	4	0	0	0	Reac1,Reac5
7	1	1	1	4	0	1	999995	5	0	0	0	Reac1,Reac5
8	1	1	1	3	0	1	999994	6	0	0	0	Reac1,Reac5
9	1	1	1	2	0	1	999993	7	0	0	0	Reac1,Reac5
10	1	1	1	1	0	1	999992	8	0	0	0	Reac1,Reac5
11	1	1	1	0	0	1	999991	9	0	0	0	Reac1,Reac5
12	1	1	1	0	0	0	999990	10	0	0	0	Reac1
13	1	1	1	0	0	0	999990	10	0	0	0	

Table 3.4 – Simulation complète avec les valeurs d'entrée : A = *vrai*, B = *faux*, de l'implémentation montrée sur la figure 3.15, entrées A et B, sortie H, *extra-entrée* D = *vrai*, pour le sous-réseau candidat 6)

Une fois la simulation terminée, on convertit le nombre de jetons présents dans la place représentant la sortie de l'implémentation (métabolite A) de la façon suivante :

— Si $A \geq 10$ alors A est vrai.

— Sinon A est faux.

Une fois que toutes les simulations pour une implémentation donnée sont effectuées, on obtient la table de vérité de cette implémentation. Cette table sera comparée à toutes les tables données en paramètre

à NetGate, pour trouver la fonction logique de l'implémentation. Si une correspondance est trouvée, alors l'implémentation, son sous-réseau candidat, ainsi que sa fonction booléenne sont combinés pour construire une porte logique enzymatique qui sera stockée dans une bibliothèque temporaire.

3.1.2.4 Tri des implémentations

Il est tout à fait possible que NetGate trouve plusieurs versions différentes d'une même porte logique enzymatique. Elles auront les mêmes entrées, la même sortie et la même fonction mais leurs sous-réseaux candidats pourront être différents. Sur la figure 3.18, nous pouvons voir un exemple, il y a trois portes enzymatiques différentes, toutes prenant en entrée A et F et en sortie D. Néanmoins ces portes sont basées sur trois sous-réseaux candidats différents. Nous voulons avoir un maximum de portes logiques enzymatiques dans notre bibliothèque, et donc disposer de plusieurs exemplaires de la même porte basés sur des réseaux différents est une bonne chose. On peut en effet imaginer une situation où une porte ayant entrées, sortie et fonction logique déterminées est requise ; dans ce cas, avoir plusieurs possibilités d'implémentation est un avantage car on pourra choisir celle dont les espèces moléculaires sont les plus compatibles avec le reste du circuit. C'est le cas par exemple des portes 1 et 2, l'une utilise l'espèce moléculaire B alors que l'autre utilise l'espèce M. Mais nous avons aussi le cas de la porte 3, cette porte comparée à la porte 1 ne présente que des désavantages. En effet, le sous-réseau de la porte 1 est entièrement inclut dans le sous-réseau de la porte 3. A aucun moment, utiliser la porte 3 ne pourra se révéler plus avantageux que d'utiliser la porte 1. Elle est donc inutile et nous n'allons pas la conserver dans la bibliothèque.

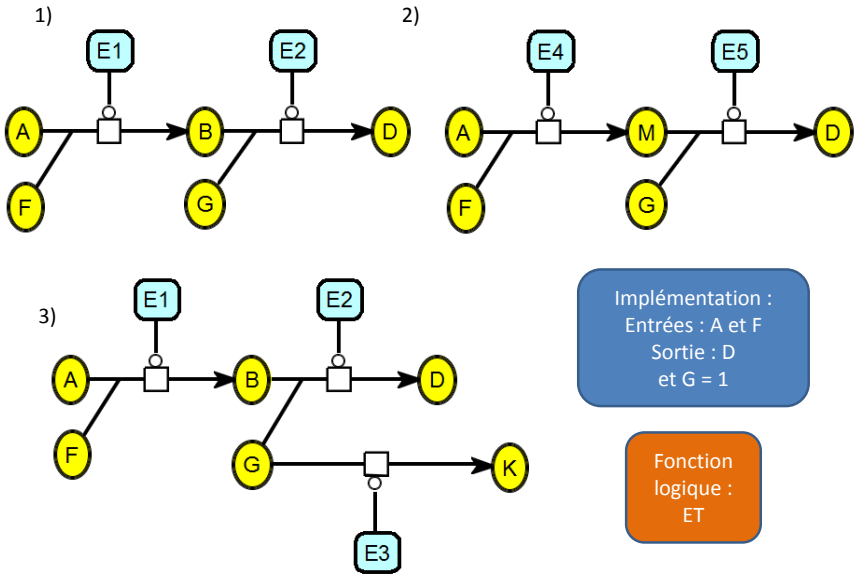


Figure 3.18 – Exemple de trois portes enzymatiques différentes implémentant la même fonction logique, mais utilisant des sous-réseaux différents.

A chaque fois qu'une porte sera trouvée, elle va être comparée à celles déjà stockées qui ont les mêmes entrées, sortie, extra-entrée et fonction logique pour déterminer si elle est meilleure. Si c'est le cas (par exemple, si la porte 3 est déjà stockée et que l'on vient de découvrir 1) elle va remplacer la porte déjà présente dans la bibliothèque. Si au contraire, il y a une porte meilleure déjà présente, la nouvelle

ne sera pas stockée dans la bibliothèque. Enfin, si aucune n'est meilleure, mais qu'elle n'est meilleure qu'aucune autre porte (cet ordre n'est pas total), elle est simplement ajoutée (par exemple, si la porte 1 est déjà stockée et on vient de découvrir 2).

Pour déterminer si la porte P_1 est meilleure que la porte P_2, nous allons utiliser des critères simples sur la taille des sous-réseaux, puis sur les extra-entrées fausses (puisque ce sont des interdictions de présence de métabolites) et enfin sur les extra-entrées vraies :

- Si toutes les réactions élémentaires du sous-réseau de P_1 sont comprises dans le sous-réseau de P_2 mais que ces sous-réseaux sont différents alors P_1 est meilleure
- Réciproquement, si toutes les réactions élémentaires du sous-réseau de P_2 sont comprises dans le sous-réseau de P_1 mais que ces sous-réseaux sont différents alors P_2 est meilleure
- Si les deux sous-réseaux sont identiques alors :
  - Si toutes les extra-entrées fausses de P_1 sont comprises parmi les extra-entrées fausses de P_2 et que le nombre d'extra-entrées fausses de P_2 est supérieur à celui de P_1 alors P_1 est meilleure
  - Réciproquement, si toutes les extra-entrées fausses de P_2 sont comprises parmi les extra-entrées fausses de P_1 et que le nombre d'extra-entrées fausses de P_1 est supérieur à celui de P_2 alors P_2 est meilleure
  - Si toutes les extra-entrées fausses de P_1 et P_2 sont identiques on va différencier sur les extra-entrées vraies :
    - Si toutes les extra-entrées vraies de P_1 sont comprises parmi les extra-entrées vraies de P_2 et que le nombre d'extra-entrées vraies de P_2 est supérieur à celui de P_1 alors P_1 est meilleure
    - Si toutes les extra-entrées vraies de P_2 sont comprises parmi les extra-entrées vraies de P_1 et que le nombre d'extra-entrées vraies de P_1 est supérieur à celui de P_2 alors P_2 est meilleure
    - Si toutes les extra-entrées vraies sont identiques, on considère que les portes sont complètement identiques (les deux peuvent être correctes) alors on décide de conserver P_1.

Ce procédé permet finalement d'obtenir une bibliothèque de portes logiques enzymatiques sans redondances inutiles.

3.1.3 Résultats

NetGate a été programmé dans le langage objet C++. Il a été compilé avec *gcc* sur les systèmes d'exploitation les plus répandus, Linux, MacOSX et Windows, sur des architectures 32 bits et 64 bits. Le programme source représente environ 5000 lignes de code. NetGate a été parallélisé en utilisant un processus par fonction booléenne cherchée. Il inclut par défaut dans sa liste de recherche, un jeu de sept fonctions logiques à deux entrées parmi les plus communément employées.

Nous avons lancé NetGate sur une version du métabolisme central du carbone de *B. subtilis* ayant 35 enzymes. Nous avons fixé la taille maximum des portes logiques enzymatiques à 4 interactions élémentaires. En moins de 16 secondes, sur un ordinateur *Apple MacBook Air* (proceseur Intel quad-core cadencé à 1.7 MHz), NetGate a trouvé 1759 portes logiques enzymatiques. Il a créé 820 sous-réseaux candidats, évalué 2,155,496 implémentations et à la fin obtenu 836 portes ET et 923 portes OU.

Nous avons simulé de façon fine avec HSIM plusieurs centaines des portes logiques enzymatiques trouvées par NetGate. Pour automatiser ces tests, nous avons écrit un utilitaire qui convertit la descriptions d'une porte logique enzymatique en modèle HSIM. Pour chaque porte, nous avons simulé

l'intégralité de sa table de vérité, donc effectué autant de simulations HSIM que de combinaisons des valeurs d'entrées.

Les conditions de simulation de toutes ces portes partagent plusieurs propriétés :

— Les réactions avaient lieu dans une vésicule virtuelle sphérique de 0.4 μm de diamètre (volume $\approx 0.0335\ \mu m^3$).

— La concentration initiale des métabolites pour les entrées représentant la valeur booléenne *vrai* à été fixée à approximativement $1mM$, (20000 copies).

— La concentration de chaque type d'enzyme est d'environ 2.5μM (50 copies).

— Le seuil utilisé pour déterminer qu'une concentration de métabolites représente la valeur booléenne *vrai* est d'environ 0.5 mM (10000 copies).

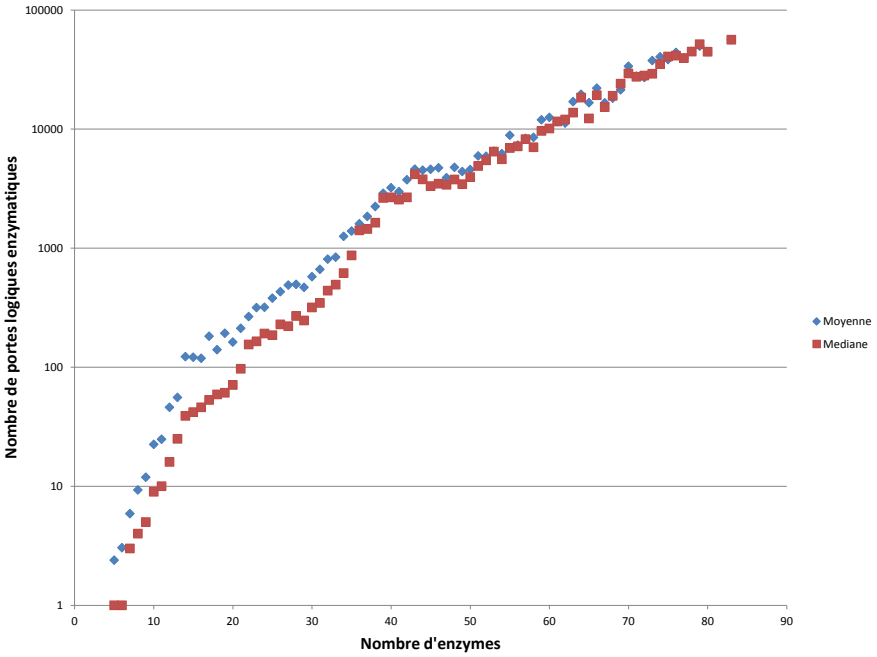


Figure 3.19 – Nombre portes trouvées en fonction de la taille du réseau initial.

Toutes ces simulations fines ont montré que les portes logiques enzymatiques trouvées par NetGate étaient correctes pour toutes les combinaisons d'entrées.

Ensuite, nous avons voulu tester les capacités de NetGate sur un ensemble de réseaux plus important. Nous avons téléchargé un jeu de réseaux métaboliques partir de la base de données *KEGG* (KEGG Pathway Database, `http://www.genome.jp/kegg/pathway.html`). Ces données comprenaient 72095 réseaux métaboliques de provenance variées ; la taille de ces réseaux variant de 1 à 80 réactions enzymatiques. Nous avons lancé NetGate sur tous les réseaux ayant plus de 5 réactions ce qui nous donne un total de 41668 réseaux métaboliques. Nous avons fixé la taille des portes logiques enzymatiques à chercher à 5 réactions au maximum.

Nous avons extrait plusieurs courbes à partir de ces résultats. Toutes les courbes utilisent une échelle semi-logarithmique et nous représentons à chaque fois les moyennes (en bleu) et les médianes (en rouge). Nous pouvons voir sur la figure 3.19 que plus un réseau métabolique comporte de réactions, plus NetGate va trouver de portes logiques enzymatiques. L'échelle logarithmique sur le nombre de portes et la courbe presque linéaire, nous montrent une progression exponentielle du nombre de portes.

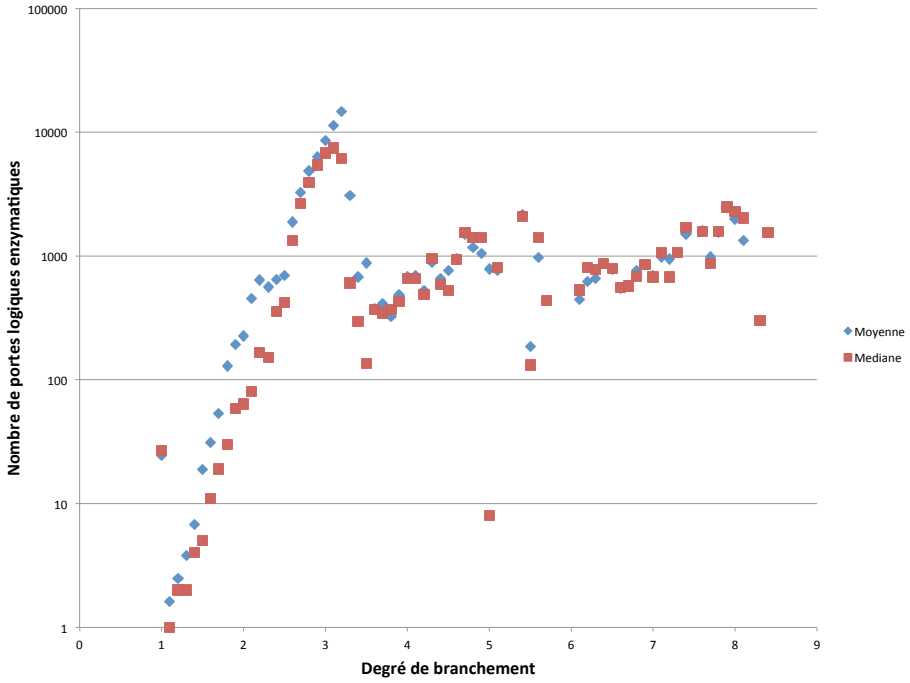


Figure 3.20 – Nombre portes trouvées en fonction du degré de branchement du réseau initial.

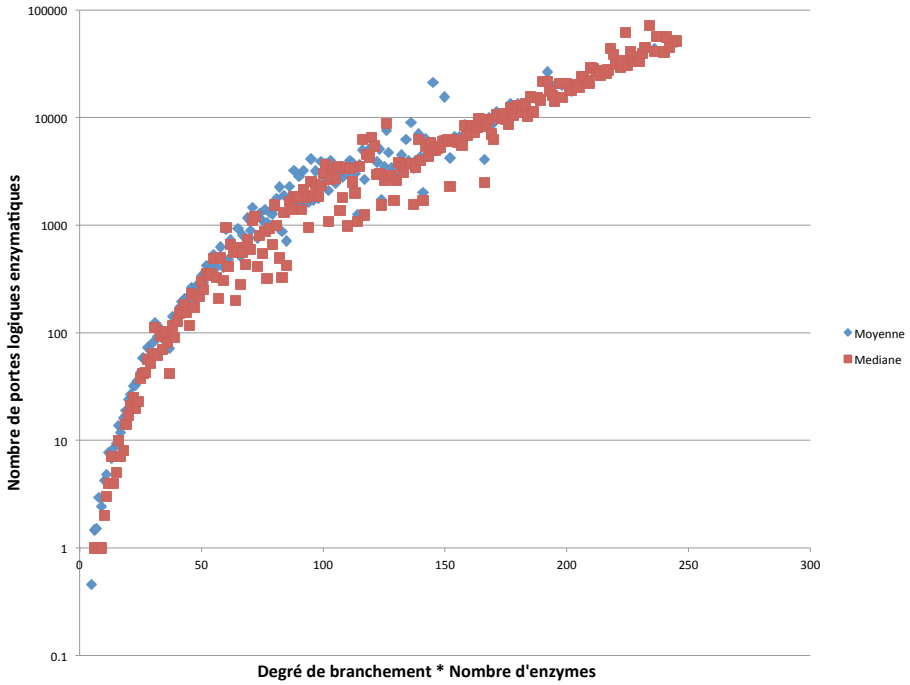


Figure 3.21 – Nombre portes trouvées en fonction du produit de la taille du réseau initial par son degré de branchement.

Lors du calcul de la complexité de l'algorithme nous avons aussi observé qu'en plus de la taille d'un réseau, il est nécessaire de connaitre sa topologie pour évaluer le nombre de portes. Ce facteur est représenté ici par le degré de branchement du réseau, plus il est élevé plus le réseau sera dense.

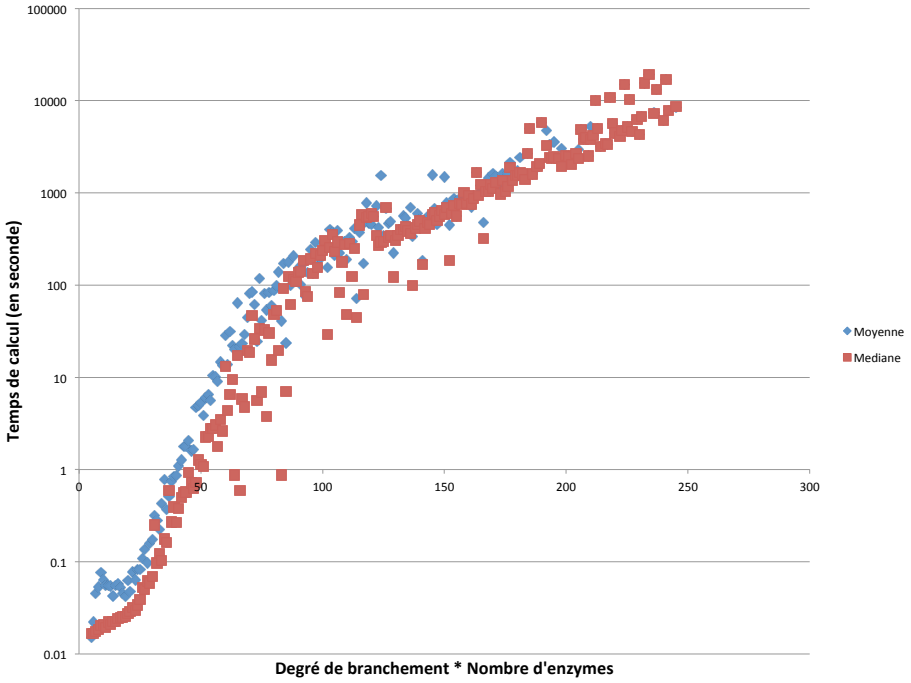


Figure 3.22 – Temps de calcul utilisé en fonction de la taille du réseau initial.

Nous avons ensuite tracé le nombre de portes en fonction du degré de branchement (voir fig. 3.20). Nous pouvons voir un phénomène intéressant, la courbe à une forme en dent de scie. Il y a trois pics qui correspondent aux degrés 3.2, 5.4 et 7.6. Il est difficile d'expliquer la raison de ce motif, mais nous supposons que les réseaux métaboliques ayant des degrés de branchement similaires à ceux des pics, d'avoir des propriétés spécifiques d'un point de vue logique. Pour obtenir une courbe plus claire, nous avons combiné ces deux descripteurs des réseaux métaboliques en en faisant le produit, puis nous avons tracé la courbe du nombre de porte logique enzymatique en fonction de ce nouveau paramètre (voir fig. 3.21). Nous obtenons une courbe exponentielle avec une pente raide au début, puis qui faiblit un peu ensuite. Enfin nous avons tracé sur la figure 3.22 la courbe du temps de calcul en fonction du degré de branchement multiplié par le nombre de réactions. Elle est aussi constituée de deux parties bien nettes, la première à forte pente représente l'augmentation rapide du temps de calcul à partir d'une valeur $x = 30$, puis, arrivé à $x = 100$ la courbe s'infléchie et prend une pente plus douce.

Enfin, on peut voir sur toutes ces courbes que les valeurs moyennes et médianes sont relativement proches, ce qui montre que le nombre de portes trouvées varie peu avec le réseau initial particulier d'où elles sont issues, mais plutôt avec sa taille et son degré de connectivité.

3.1.4 Validation

Nous savons grâce à des simulations fines que les portes trouvées par NetGate sont fiables, mais nous avons aussi essayé d'obtenir une validation expérimentale. Au cours de son post-doc à SysDiag, Liza Felicori a trouvé *manuellement* quelques implémentations prometteuses de portes logiques enzymatiques, en analysant un réseau métabolique hybride provenant d'une partie du métabolisme central du carbone de *B. subtilis* et d'un réseau synthétique. Liza, qui est une biologiste expérimentale, a réussi à valider *in vitro* sept portes parmi celles qu'elle a testées. On a lancé NetGate sur ce réseau de 35 réactions, et il été capable de retrouver ces sept portes, ainsi que de nombreuses autres. Nous avons repris ici deux de ces portes :

— Porte ET avec comme entrées du lactose et de l'ABTS et comme sortie de l'ABTS oxydé.
— Porte OU avec comme entrées du lactose et du lactate et comme sortie de la resorufine.

La porte ET est constituée de 3 réactions enzymatiques successives, catalysées par la β-galactosidase, la glucose oxydase et la peroxydase. Les entrées sont le lactose (substrat de la β-galactosidase) et ABTS, la sortie est ABTS oxydé. Il n'y a pas d'augmentation de l'absorbance à moins que les deux entrées ne soient présentes et donc que la porte soit dans l'état (1,1) comme illustré dans le graphique à barres.

La porte OU se compose de deux voies pouvant conduire à la formation de résorufine par l'intermédiaire de quatre réactions catalysées par la β-galactosidase, la glucose oxydase, la lactate oxydase et le peroxydase. Si l'une des entrées est présente, alors de la résorufine sera produite, conduisant à une augmentation de l'absorbance dans les configurations (0,1), (1,0) et donc (1,1). Il n'y a pas de variation d'absorbance en l'absence des deux entrées (0,0). Pour fonctionner cette porte a besoin d'Amplex Red, qui sera donc pour NetGate interprété comme une *extra-entrée* à l'état *vrai*.

Nous pouvons voir que ces deux portes logiques enzymatiques sont parfaitement fonctionnelles. L'augmentation de l'absorbance est bien corrélée à la table de vérité des portes. Nous avons donc bien une porte logique enzymatique ET, et une porte logique enzymatique OU.

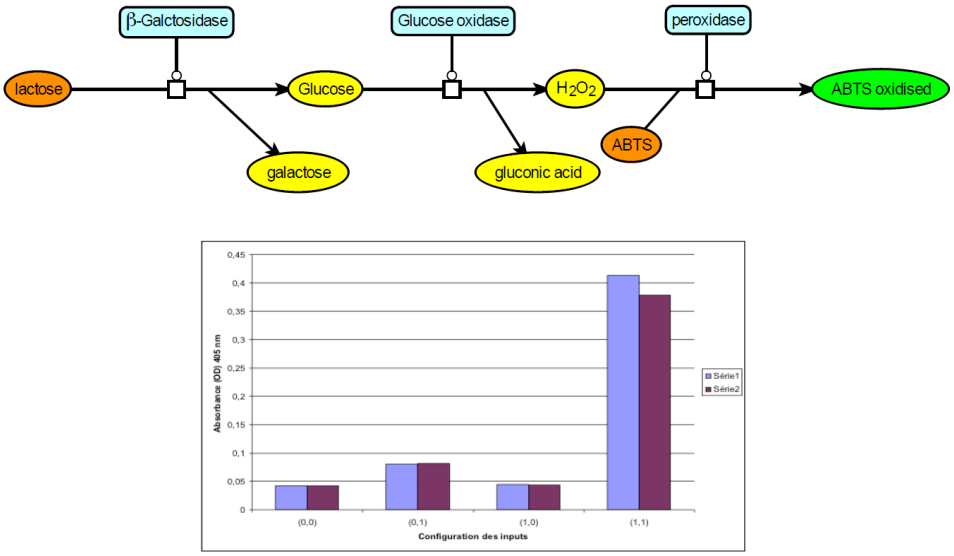


Figure 3.23 – Porte ET expérimentale : lactose et ABTS en entrée, ABTS oxydé en sortie. Le graphique à barres montre le bon fonctionnement de la porte.

3.1.5 Discussion

Un des points les plus importants et les plus discutés de notre approche est le simulateur interne, par réseau de Petri adapté, qui va évaluer les réseaux candidats pour chaque implémentation. C'est un processus critique en termes de temps et de performance. En effet c'est l'opération la plus consommatrice de temps calcul, et qui de plus, sera répétée un très grand nombre de fois.

Du fait que c'est ce sous-programme qui va évaluer si un sous-réseau candidat remplit bien une fonction booléenne donnée, il doit être le plus correct possible car il conditionne la *qualité* des portes trouvées par NetGate. Pour maintenir une bonne qualité des résultats, tout en conservant un temps

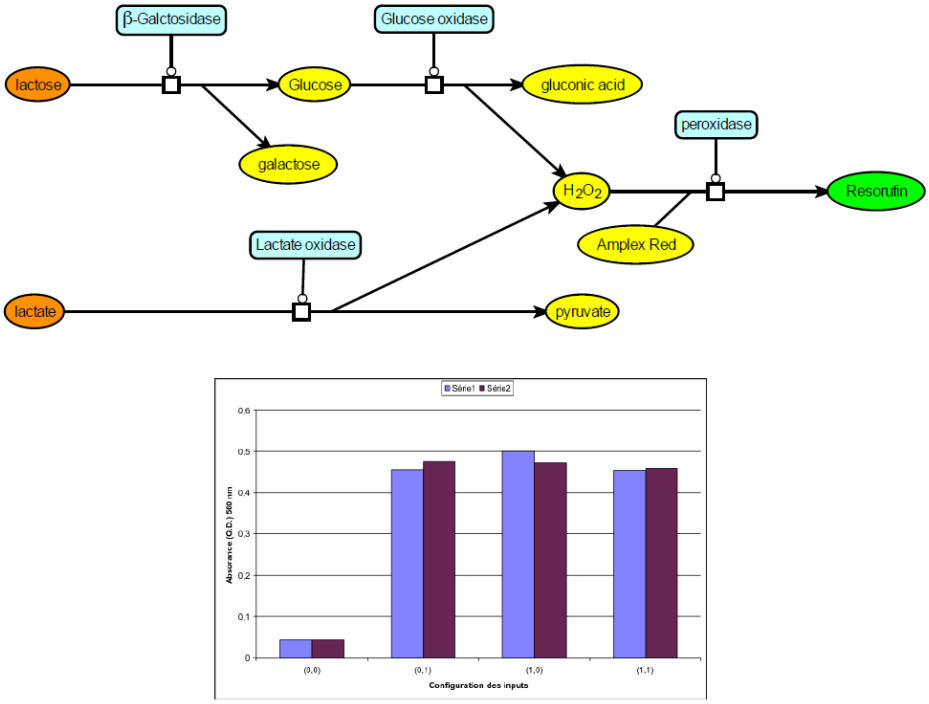


Figure 3.24 – Porte OU expérimentale : lactose et lactate en entrés, resorufin en sortie. Le graphique à barres montre le bon fonctionnement de la porte.

calcul raisonnable nous avons fait des compromis entre le niveau de détail des simulations avec les informations à notre disposition, et le temps de simulation.

Il n'est pas aisé de quantifier l'efficacité d'un processus d'évaluation volontairement approximatif. Nous aurions pu utiliser une méthode de simulation beaucoup plus précise, telle que celle implémentée dans HSIM, ou bien une version utilisant des constantes cinétiques pour les réactions, ou encore une toute autre approche par simple reconnaissance d'une série de patterns dans la structure des réseaux métaboliques candidats.

Aucune de ces méthodes n'aurait eu les qualités que nous recherchions, à la fois rapidité et exhaustivité. Nous évaluons les sous-réseaux uniquement sur leur résultats en tant que portes logiques, et non sur leur structure. Cet aspect *sans a priori* nous permet de trouver des portes logiques enzymatiques de n'importe quelle taille ainsi qu'avec des motifs exotiques ou non attendus (des boucles par exemple). C'est l'efficacité, en terme de temps calcul, et la qualité du modèle de simulation à gros grain choisi, qui nous ont permis d'obtenir cette exhaustivité dans la recherche des portes portentielles.

Les tests que nous avons effectués sur des centaines de réseaux nous ont permis de voir que notre approche à gros grain par réseaux de Petri remplit sa fonction de façon très satisfaisante. Cette méthode nous a apporté un moyen rapide et fiable d'évaluer des réseaux métaboliques sans nécessiter d'information sur la cinétique des réactions. Le but de NetGate, fournir une liste de portes logiques enzymatiques qui ont une grande probabilité de fonctionner, est à notre avis atteint.

3.1.5.1 Complexité

Pour calculer la complexité de notre algorithme, en termes de nombre d'étapes élémentaires à effectuer en fonction de la taille des données d'entrée, nous allons commencer par quantifier ces données d'entrée.

Nous supposerons que le réseau métabolique dont on veut extraire des portes logiques contient $nbRe$ réactions élémentaires, réparties en $nbIn$ interactions élémentaires. Ce réseau comporte $nbCp$ composants (*i.e.* nombre de types de métabolite et de types d'enzymes). La variable $degMoy$ représente le degré moyen de connectivité des interactions élémentaires dans le réseau.

L'algorithme est globalement séparé en deux parties : la première, un pré-traitement, est effectuée sur le réseau métabolique pour le rendre exploitable par la deuxième, qui elle va parcourir le réseau pour en extraire les sous-réseaux candidats. La première partie est rapide, son temps d'execution est proportionnel à $nbRe * nbCp$. Elle se borne à parcourir le réseau et à en stocker les informations sous un autre format. La deuxième partie est beaucoup plus longue et plus complexe, c'est elle qui va déterminer la complexité globale de l'algorithme.

On doit énumérer les sous-réseaux candidats qu'il faudra créer (et évaluer par la suite). Soit $nbSousReseaux$ leur nombre ; ce nombre dépend de la topologie du réseau d'entrée. Essayons de le calculer :

- Sous-réseaux de taille 1 : $nbIn$ (un pour chaque interaction élémentaire)
- Sous-réseaux de taille 2 : $nbIn * degMoy/2$. Pour chaque sous-réseau de niveau 1, on peut connecter $degMoy$ interactions élémentaires. Tous les sous-réseaux seront trouvés en double, donc on divise par 2.
- Sous-réseaux de taille 3 : $nbIn * degMoy/2 * ((degMoy - 1) + (degMoy - 1))/3$. Pour chaque sous-réseau de niveau 2, on peut connecter $degMoy - 1$ nouvelles interactions élémentaires. Tous les sous-réseaux seront trouvés en triple, donc on divise par 3.

A partir des réseaux taille 3 nous arrivons au cœur du problème du calcul exact du nombre moyen de sous-réseaux. On pourrait penser que le nombre de sous-réseaux de taille $n+1$ puisse être obtenu en multipliant le nombre de réseau de taille n par $(degMoy - 1)^n$. Cela représente la possibilité d'ajouter toutes les interactions élémentaires connexes aux n interactions du sous réseau de taille n dont on part, moins le fait que chaque interaction est déjà connectée au moins une fois au réseau de taille n. En réalité tout dépend fortement de la topologie du réseau, car chaque lien possible vers une nouvelle interaction, peut aussi être relié à une interaction déjà comprise dans le réseau de taille n, cette possibilité augmentant au fur et à mesure que n s'approche de $nbIn$. Nous avons néanmoins établi des bornes maximum et minimum :

- $nbSousReseaux \geq nbIn * (nbIn - 1) + 1$ qui représente le graphe connexe le plus simple, un cycle, avec $degré = 2$.
- $nbSousReseaux \leq \sum_{i=1}^{i=nbIn} \frac{!nbIn}{(nbIn-i)!*i!}$ qui représente le graphe connexe le plus dense, le graphe complet, avec $degré = nbIn$.

Pour chaque sous-réseau on doit compter toutes les implémentations possibles $nbImp$ des portes en fonction du nombre de sorties $nbSo$ et d'entrées $nbEn$, ici nous ne regarderons uniquement que les portes à 2 entrées (nous ne prenons pas en compte l'optimisation des descendants qui rendrait le calcul encore plus complexe) :

$$nbImp = nbSo * nbEn * (nbEn - 1) * 2^{nbEn-2}$$

D'où on obtient le nombre d'évaluations nécessaires :

$$nbEva = nbImp * 4 = nbSo * nbEn * (nbEn - 1) * 2^{nbEn-2} * 4$$

Chaque évaluation peut durer un nombre de rounds variable qui sera au maximum le nombre d'interactions élémentaires du réseau plus le maximum de jetons des entrées à l'initialisation de l'évaluation (extra-entrées non comprises).

Même sans calculer la complexité réelle de l'algorithme, nous pouvons nous douter qu'elle serait très élevée. C'est sa principale faiblesse. Le nombre de sous-réseaux, et donc le temps de calcul, va augmenter exponentiellement avec le nombre d'interactions élémentaires du réseau de départ. De même, plus les sous-réseaux candidats sont gros, plus le nombre d'implémentations possibles va augmenter. Le seul point positif c'est que nous pouvons en déduire qu'ajouter des réactions élémentaires, mais pas de nouvelles interactions, fera varier éventuellement le nombre d'entrées, mais n'augmentera pas autant le temps de calcul de façon drastique.

3.1.5.2 Limiter la taille des sous-réseaux

Nous avons vu que la complexité en temps de NetGate pouvait être une limitation forte pour l'analyse de grands réseaux. Notre solution pour pallier ce problème est de limiter la taille des sous-réseaux formés, à la valeur $limiteTailleSousReseau$ donnée en paramètre. Cette borne limitera donc aussi la taille des portes logiques enzymatiques qu'il sera possible de trouver. En majorant la taille des sous-réseaux candidats à ce paramètre, le nombre maximal de sous-réseaux possibles sera alors de :

$$nbSousReseaux <= \sum_{i=1}^{i=limiteTailleSousReseau} \frac{!nbIn}{(nbIn-i)! * i!}$$

ce qui est bien meilleur.

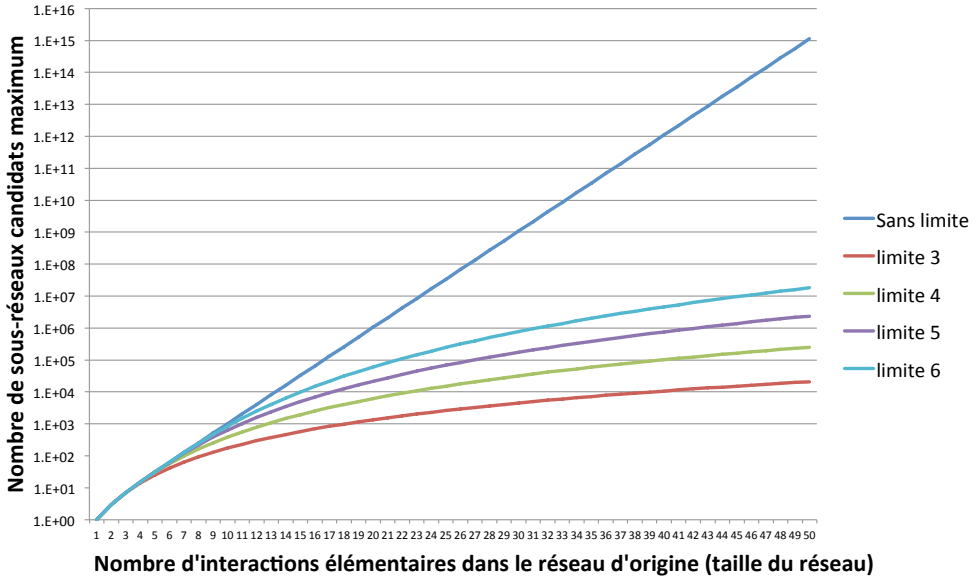


Figure 3.25 – Courbes représentant le nombre maximum de sous-réseaux en fonction de la taille du réseau d'origine et de la limite de taille par sous-réseau.

Cette limite a aussi une influence directe sur le nombre d'implémentations puisqu'il dépend de $nbEn$ qui lui, bien sûr, dépend de la taille du sous-réseau. Sur la figure 3.25 nous pouvons voir des courbes représentant les nombres maxima de sous-réseaux en fonction de la taille du réseau d'origine et de la limite de taille par sous-réseau. On peut voir que sans limite de taille, le nombre de sous-réseaux croît très vite : la courbe est linéaire avec une échelle semi-logarithmique (le nombre de sous-réseaux candidats est presque multiplié par 10 à chaque fois qu'on ajoute une interaction élémentaire au sous-réseau original). Lorsque nous utilisons une limite, la progression est nettement plus faible. Pour une valeur limite de 3, il est nécessaire de faire plus que doubler la taille du réseau d'entrée (passer d'un

réseau de taille 23 à un réseau de taille de 50) pour multiplier par 10 le nombre de sous-réseaux candidats (qui passe de 2047 à 20875). Même dans le cas d'un réseau métabolique d'entrée de taille 50, le nombre de sous-réseaux reste tout à fait calculable en un temps raisonnable.

Comme indiqué précédemment, la limitation de la taille maximum des sous-réseaux candidats va aussi limiter la taille des portes logiques enzymatiques qui seront trouvées. Dans la pratique ce n'est pas vraiment un problème, plus une porte logique enzymatique est petite plus elle sera utilisable. En effet, si elle ne nécessite que peu d'espèces moléculaires différentes, elle risquera peu d'entrer en conflit avec les autres portes présentes dans un circuit complexe. Dans la plupart de nos tests nous avons utilisé une taille limite de 3, voire 4 parfois, mais jamais supérieure. Cela permet d'utiliser NetGate sur des réseaux de grandes tailles sans problème de temps de calcul.

3.2 Implémenter une fonction booléenne avec un réseau métabolique : NetBuild

Nous avons vu dans la partie précédente comment chercher et trouver des portes logiques enzymatiques. Nous allons maintenant regarder comment les assembler pour qu'elles puissent former des circuits logiques enzymatiques calculant une expression booléenne donnée en entrée.

La méthode que nous allons utiliser se divise en deux étapes : la première construit toutes les formes alternatives de l'expression booléenne donnée en entrée, et la deuxième implémente à l'aide d'une bibliothèque de portes logiques enzymatiques chacune de ces formes d'expressions booléennes. Ces différentes formes alternatives vont nous permettre d'obtenir un plus grand choix d'implémentations du circuit. Nous obtiendrons à la fin une liste de réseaux logiques enzymatiques permettant de calculer l'expression booléennes originale (voir fig. 3.26).

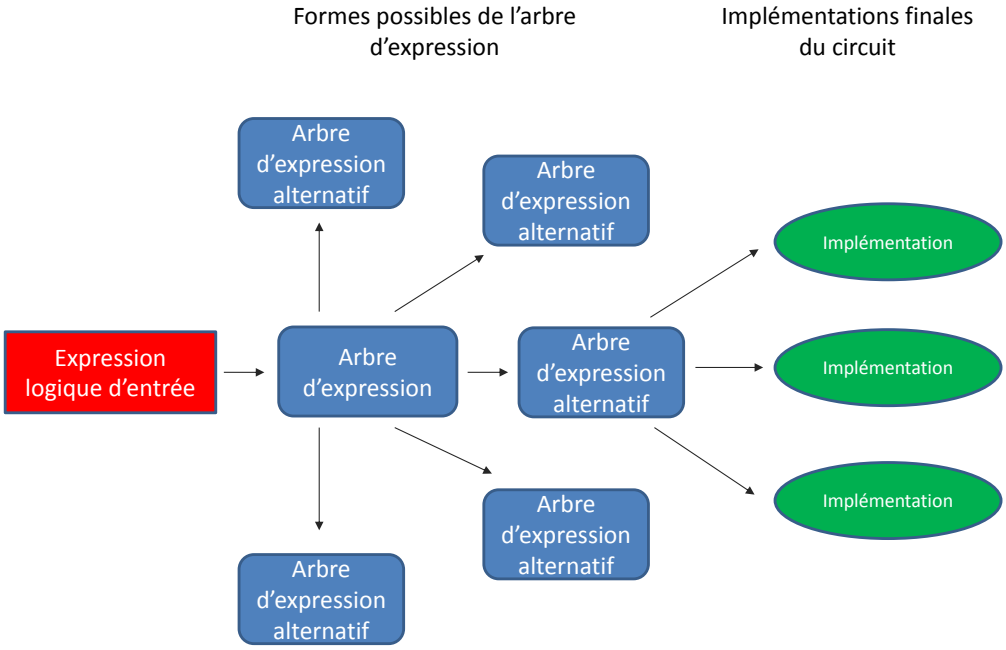


Figure 3.26 – Algorithme synthétique implémenté dans NetBuild.

La bibliothèque, issue de NetGate, décrivant les portes logiques enzymatiques est contenue dans un fichier au format CSV. Son format a été décrit dans l'introduction de la partie concernant NetGate. L'expression booléenne à implémenter est uniquement composée de variables, de parenthèses ainsi que des opérateurs ET, OU et NON. Les sorties de NetBuild seront des implémentations d'une fonction logique sous forme de réseau métabolique. Deux types d'informations seront disponibles pour chaque implémentation trouvée :

— la liste des portes logiques utilisées dans l'implémentation.
— un fichier au format SBML décrivant le réseau métabolique associé à l'implémentation.

Le ficher SBML produit contient des balises supplémentaires, non prévues dans la norme originale, pour décrire la catalyse et l'inhibition d'une réaction (balises *sboTerm="SBO:000046"* et *sboTerm="SBO:0000020"* respectivement). Ces nouvelles balises ont été créés par *l'EBI* dans le but de

développer des ontologies pour la modélisation et la représentation des systèmes biologiques (SBO Systems Biology Ontology, `http://www.ebi.ac.uk/sbo/main/`).

L'autre catégorie d'information que nous avons ajoutée utilise des annotations spécifiques pour les circuits logiques enzymatiques. Nous indiquons le rôle de certaines espèces moléculaires au sein du circuit logique enzymatique :

- Pour les entrées du circuit : "<netdraw:logic type="INPUT" />".
- Pour la sortie du circuit : "<netdraw:logic type="OUTPUT" />".
- Pour les espèces moléculaires devant être présentes initalement dans le circuit : "<netdraw:logic type="EXCESS" />".
- Pour les espèces moléculaires devant être en concentration nulle dans le circuit : "<netdraw:logic type="FORBIDDEN"/>".

Ces nouvelles balises sont conçues pour être interprétées par un logiciel d'édition de réseaux métaboliques. Nous allons maintenant décrire dans le détail comment fonctionne NetBuild.

3.2.1 Construction des expressions logiques équivalentes

3.2.1.1 Graphe des expressions

Nous allons commencer par convertir l'expression logique d'entrée en un graphe qui nous permettra de la manipuler plus facilement. Ces graphes d'expressions sont des arbres contenant deux types de nœuds : variables et opérateurs. Ces graphes ont plusieurs utilités : ils permettent d'obtenir une forme non-ambigüe de l'expression et l'ordre d'exécution des opérateurs devient explicite. Ensuite ils vont nous permettre de manipuler plus facilement la forme des expressions booléennes qu'ils représentent. Enfin leur forme en arbre est particulièrement bien adaptée à la création de circuits logiques enzymatiques, ce qui rend l'implémentation des expressions logiques plus simple.

Nous allons commencer par lire l'expression logique fournie en paramètre, puis la convertir en un graphe d'expression. Trois types d'opérateurs sont autorisés (ET, OU, NON), ils donneront trois types possibles de nœuds *opérateur*. Sur la figure 3.27, nous avons représenté la transformation d'une expression booléenne en arbre d'expression, ainsi que la liste des éléments utilisés dans ces graphes. Ces arbres d'expressions booléennes vont nous permettre de transformer les opérations de changement de forme des expressions logiques en opération de changement de topologie sur des arbres, ce qui sera plus pratique.

3.2.1.2 Une expression permet de multiples implémentations

Une expression booléenne, comme toute expression mathématique, peut prendre plusieurs formes ; Chacune de ces formes étant une écriture différente représentant la même expression booléenne et donc ayant la même table de vérité. Les arbres d'expression correspondants auront donc aussi des formes différentes et selon la forme de ces arbres, leur implémentation sous forme de circuits enzymatiques ne sera pas la même. Il va donc falloir décider quelle forme de l'expression booléenne d'entrée nous allons choisir. Nous pourrions considérer qu'obtenir l'expression logique comportant le moins d'opérateurs logiques est le meilleur choix, car cela signifierait que nous aurions besoin de moins de portes logiques enzymatiques pour l'implémenter ; en fait, c'est plus compliqué que ça. La principale contrainte des réseaux logiques enzymatiques est la difficulté à pouvoir trouver les portes logiques enzymatiques nécessaires pour les construire, or si nous avons plusieurs arbres d'expression, nous allons avoir plusieurs implémentations possibles pour réaliser la même fonction booléenne. Si nous conservons toutes ces formes possibles, nous allons devoir effectuer beaucoup plus de calculs, mais en contrepartie, nous allons augmenter fortement la probabilité de trouver des circuits logiques enzymatiques implé-

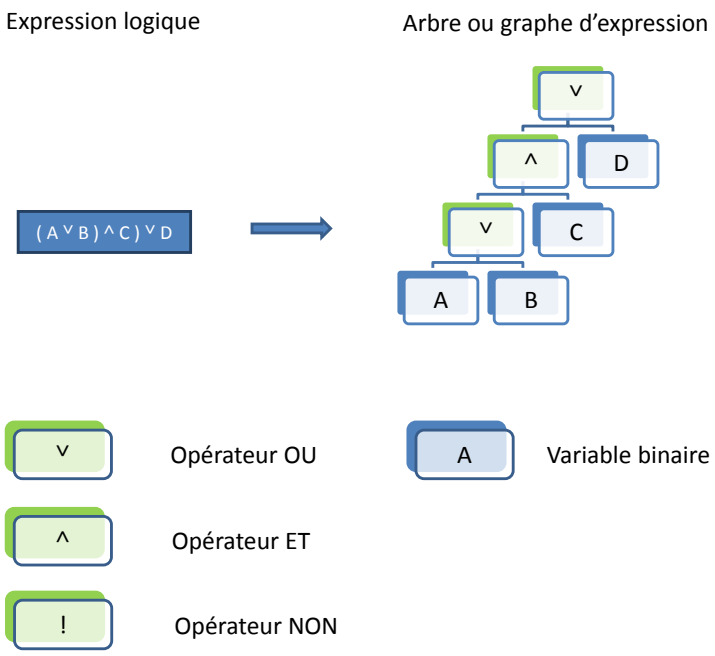


Figure 3.27 – Structure des arbres d'expressions, et représentation de la transformation d'une expression booléenne en arbre d'expression.

mentant l'une de ces formes. Nous allons détailler pourquoi des arbres d'expression tirés d'une même expression booléenne peuvent fournir plusieurs implémentations possibles. Deux arbres d'expression obtenus à partir d'une expression booléenne auront la même table de vérité, mais peuvent bien sur avoir des structures différentes (voir fig. 3.28).

Nous allons construire les arbres d'expressions en cherchant, pour chaque nœud associé à un opérateur logique, une porte logique qui à la même fonction. Nous aurons donc des arbres avec différentes structures nécessitant des portes différentes pour être implémentés. L'expression 1 (fig. 3.28) va nécessiter au moins une porte ET et une porte OU, car nous avons utilisé dans son arbre d'expression un nœud opérateur ET et un nœud opérateur OU. De la même façon, l'expression 2 va nécessiter deux portes ET et une porte OU. De cette façon, nous avons déjà multiplié par trois le choix de possibles implémentations avec un réseau logique enzymatique. Enfin l'expression 3 va nécessiter une porte ET, une porte OU ainsi que trois portes NON.

Pour créer ces différentes topologies d'arbres d'expression nous allons rechercher toutes les formes possibles que peut prendre l'expression originale. Nous trouverons des formes différentes qui auront donc des arbres d'expression différents, mais avec la même table de vérité. Pour cela, nous allons partir de l'expression originale sur laquelle nous allons appliquer plusieurs transformations qui conserveront sa fonction booléenne. Ces changements de forme sont effectués en utilisant les opérations suivantes :

- Opération de distribution : $C \wedge (A \vee B) = (C \vee A) \wedge (C \vee B)$
- Opération de factorisation : $(C \vee A) \wedge (C \vee B) = C \wedge (A \vee B)$
- Lois de De Morgan : $\neg(A \vee B) = (\neg A \wedge \neg B)$

En plus de ces opérations, plusieurs règles de simplification d'expressions booléennes seront prises en

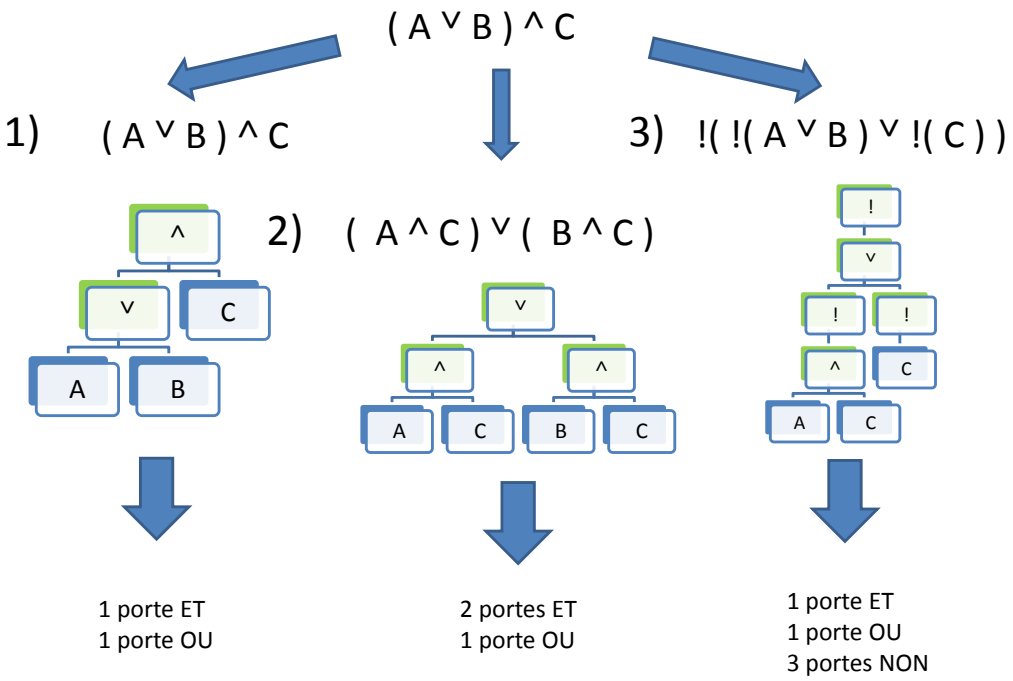


Figure 3.28 – Représentation de plusieurs arbres d'expression correspondant à une même expression booléenne. Nous pouvons voir que la topologie des arbres est différente et il faudra des jeux de portes différents pour les implémenter.

compte :

- $A \wedge A = A$
- $A \vee A = A$
- $\neg\ \neg A = A$
- $A \wedge 0 = 0$
- $A \wedge 1 = A$
- $A \vee 0 = A$
- $A \vee 1 = 1$

Toutes ces propriétés vont être utilisées pour construire trois opérateurs fonctionnant sur les expressions booléennes. Ces opérateurs vont produire de nouvelles expressions booléennes équivalentes.

- **Opérateur de distribution** : il appliquera toutes les opérations de distribution possibles sur une expression booléenne donnée en paramètre, chacune des nouvelles expressions booléennes seront simplifiées puis retournées en sortie de l'opérateur.
- **Opérateur de factorisation** : il appliquera toutes les opérations de factorisation possibles sur une expression booléenne, chacune des nouvelles expressions seront simplifiées puis retournées par l'opérateur.
- **Opérateur de De Morgan** : il appliquera les lois de De Morgan sur l'expression booléenne donnée en argument, chacune des nouvelles expressions booléennes eront simplifiées puis retournées par l'opérateur.

Chaque opérateur possède des conditions d'application, qui lorsqu'elles sont vérifiées permettent à l'opérateur d'agir. Sur la figure 3.29, on peut voir, pour chaque opérateur, ses conditions d'application

ainsi qu'un exemple d'arbre d'expression où ces conditions sont réalisées. Les conditions pour l'opérateur de De Morgan sont très simples, il pourra être appliqué presque partout, et va donc générer facilement un grand nombre de d'expressions équivalentes. L'opérateur de factorisation est celui dont les conditions sont les plus contraignantes, mais il a l'avantage de diminuer le nombre d'opérateurs logiques dans les expressions résultats.

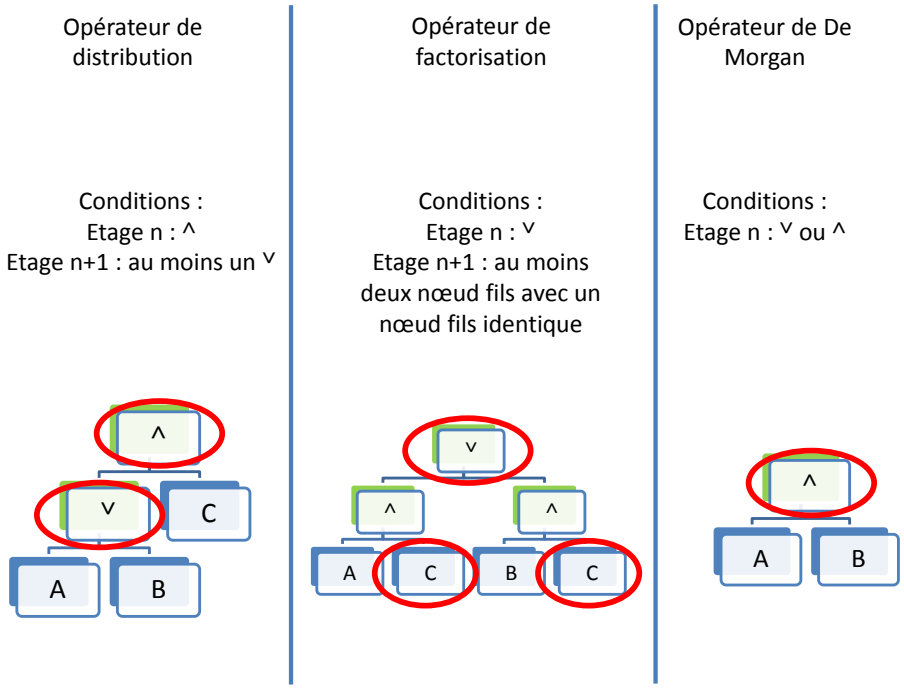


Figure 3.29 – Description des opérateurs, leurs conditions d'application et un exemple où elles sont remplies.

Nous avons détaillé sur la figure 3.30 un exemple d'application d'un des opérateurs, l'opérateur de distribution. On peut constater que sur cet exemple il y a deux possibilités d'application de l'opération de distribution. Nous avons représenté le cas le plus simple, celui entouré en jaune, il va donner une seule expression résultat.
Pour le cas entouré en rouge la situation est plus compliquée, elle va se diviser en deux expressions résultats. Chacun des deux membres à développer étant composé de deux nœuds fils, nous allons obtenir le développement de $(A \vee B) \wedge (((C \vee D) \wedge E) \vee F)$ sur le premier niveau :

— $(A \wedge (((C \vee D) \wedge E) \vee F)) \vee (B \wedge (((C \vee D) \wedge E) \vee F))$
— $((A \vee B) \wedge F) \vee ((A \vee B) \wedge ((C \vee D) \wedge E))$

Maintenant que nous avons établi quels opérateurs utiliser pour transformer la forme d'une expression booléenne, nous allons voir comment trouver toutes les formes possibles d'une expression particulière. Pour cela nous allons simplement appliquer l'un des trois opérateurs sur cette expression booléenne, pour obtenir toutes les expressions booléennes situées à distance un (un niveau d'opération) de l'expression booléennes d'entrée. Nous allons dire que nous avons *exploré* l'expression booléenne. Ensuite chaque nouvelle expression obtenue deviendra une nouvelle expression d'entrée. Nous allons progressivement trouver toutes formes à des distances d'opérateur de plus en plus grandes.

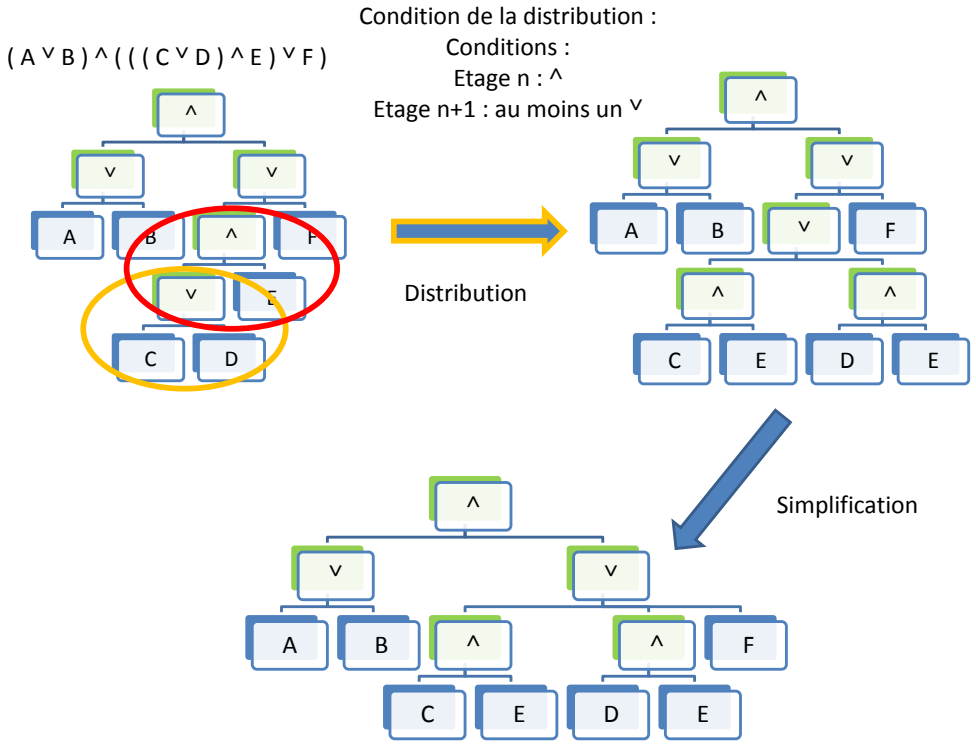


Figure 3.30 – L'opérateur de distribution peut s'appliquer à deux emplacements sur l'expression originale. On va obtenir deux expressions résultats différentes qui auront au préalable été simplifiées. Celle entourée en jaune est détaillée sur la figure.

Nous pouvons assimiler cet algorithme au parcours en largeur d'un graphe. Les nœuds sont les différentes formes de l'expression booléenne et les arcs, les opérateurs appliqués. Si deux formes ne diffèrent que de l'application d'un opérateur, elles sont reliées par un arc. Dans cette représentation la recherche de toutes les formes possibles revient à explorer l'intégralité du graphe. On va néanmoins limiter cette exploration car dans certains cas, le nombre de formes possibles devient extrêmement important, les formes vont être de plus en plus grosses, et comprendre de plus en plus de nœuds opérateurs. Ces formes ne seront d'aucune utilité, puisque plus la forme d'une expression est grosse, plus on aura de difficultés à l'implémenter avec des portes logiques enzymatiques. Nous allons donc imposer une limite à la taille des formes que nous explorons. Lorsqu'une forme de cette taille est atteinte, elle est gardée dans la liste des formes, mais on n'en repart plus pour continuer d'autres explorations. Nous avons représenté le fonctionnement de l'algorithme sur la figure 3.31. Le nœud rouge est le nœud de départ correspondant à l'expression booléenne d'origine. Les nœuds bleus (clairs et foncés) sont des formes trouvées par l'algorithme. Les nœuds bleus clairs sont ceux d'où on repart pour explorer un niveau de plus.

Nous avons maintenant une liste d'arbres d'expressions répondant tous à la même table de vérité, mais ces arbres ne sont pas encore prêts à être implémentés. En effet les nœuds opérateurs des arbres (ET, OU, NON) ne correspondent pas directement aux portes logiques enzymatiques que nous allons utiliser pour implémenter ces arbres. Nous allons devoir adapter les arbres dans ce but. Il y a deux différences majeures entre les portes et les opérateurs des expressions qui sont i) les portes complexes et ii) les opérateurs logiques à plus de deux entrées.

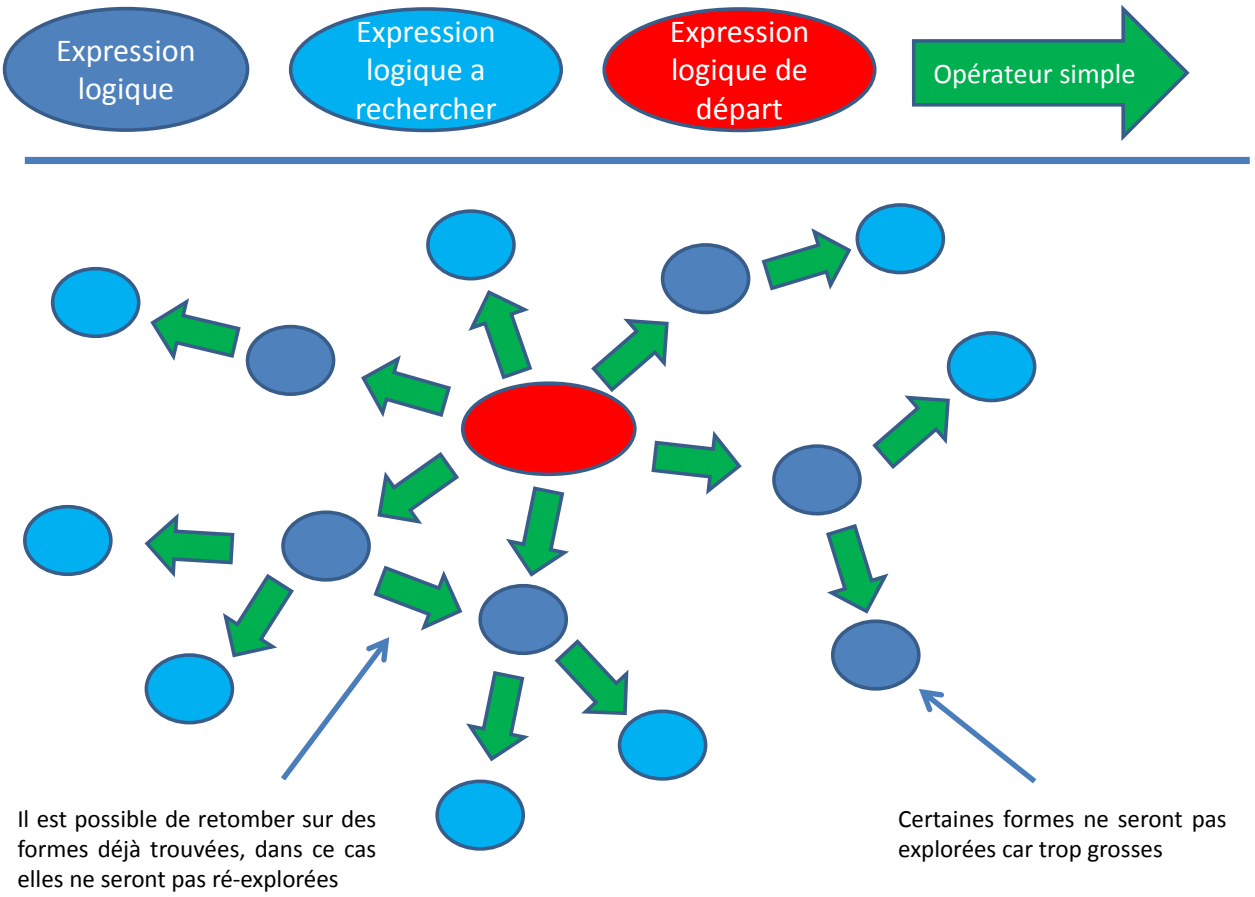


Figure 3.31 – Représentation graphique de l'algorithme de recherche de formes d'expressions booléennes.

Voici l'algorithme exact :
Les portes complexes sont des portes dont la fonction booléenne nécessite plusieurs opérateurs logiques de base pour les implémenter, par exemple la porte NON-ET correspondra à deux opérateurs booléens de base dans l'arbre d'expression : NON et ET. De plus, ces opérateurs devront respecter une certaine topologie dans l'arbre : le nœud ET devra être l'un des fils du nœud NON. Pour pouvoir utiliser des portes complexes, nous allons devoir détecter tous les arbres d'expression où cette situation existe, et ensuite les dupliquer pour en créer des variantes qui incorporeront de nouveaux nœuds opérateurs, tels que le NON-ET. Ces nœuds ne peuvent plus être manipulés par les opérateurs simples que nous avions utilisés précédemment, mais ils permettront une meilleure implémentation. Nous avons décrit un exemple de transformation d'un arbre d'expression pour les portes complexes sur la figure 3.32. Il y deux situations où il est possible de remplacer une combinaison NON et ET par un nœud opérateur NON-ET. Toutes les versions possibles doivent être référencées. Si deux remplacements sont possibles, on aura quatre versions de l'arbre d'expression (en comptant celle dont nous sommes partis).
Les opérateurs logiques ET et OU présents dans les arbres d'expressions n'ont pas de limite sur le nombre de leurs entrées, mais ce n'est pas le cas des portes logiques enzymatiques qui serviront à les implémenter. Celles-ci seront majoritairement des portes à deux et parfois à trois entrées. Il va donc falloir décomposer les opérateurs ET et OU à plus de deux entrées, en des associations d'opérateurs

```
1  Fonction RechercheToutesFormes( ExpressionLogique expressionLogiqueEntree , Entier
     limiteTailleExpression , ListeExpressionsLogiques listesExpressionsLogiquesTotal)
2      expressionLogiqueEntree.simplifier() ;
3      ListeExpressionsLogiques listeExpressionsLogiquesAChercher ;
4      listeExpressionsLogiquesAChercher.ajouter( expressionLogiqueEntree ) ;
5      tant que listeExpressionsLogiquesAChercher n'est pas vide faire
6          ListeExpressionsLogiques listeExpressionsLogiquesResultats ;
7          pour tous les ExpressionLogique expressionLogique de
             listeExpressionsLogiquesAChercher faire
8              listeExpressionsLogiquesResultats.ajouter(
                 expressionLogique.developpementSimple() ) ;
9              listeExpressionsLogiquesResultats.ajouter(
                 expressionLogique.FactorisationSimple() ) ;
10             listeExpressionsLogiquesResultats.ajouter( expressionLogique.DeMorganSimple()
                 ) ;
11         fin
12         listesExpressionsLogiquesTotal.ajouter( listeExpressionsLogiquesAChercher ) ;
13         listeExpressionsLogiquesAChercher.vider() ;
14         pour tous les ExpressionLogique expressionLogique de
             listeExpressionsLogiquesResultats faire
15             si listesExpressionsLogiquesTotal ne contient pas expressionLogique alors
16                 si expressionLogique.taille() < limiteTailleExpression alors
17                     listeExpressionsLogiquesAChercher.ajouter( expressionLogique ) ;
18                 fin
19             fin
20         fin
21     fin
22 fin fonction
```

Algorithm 4: Algorithme utilisé pour la recherche de toutes les formes possibles d'une expression booléenne.

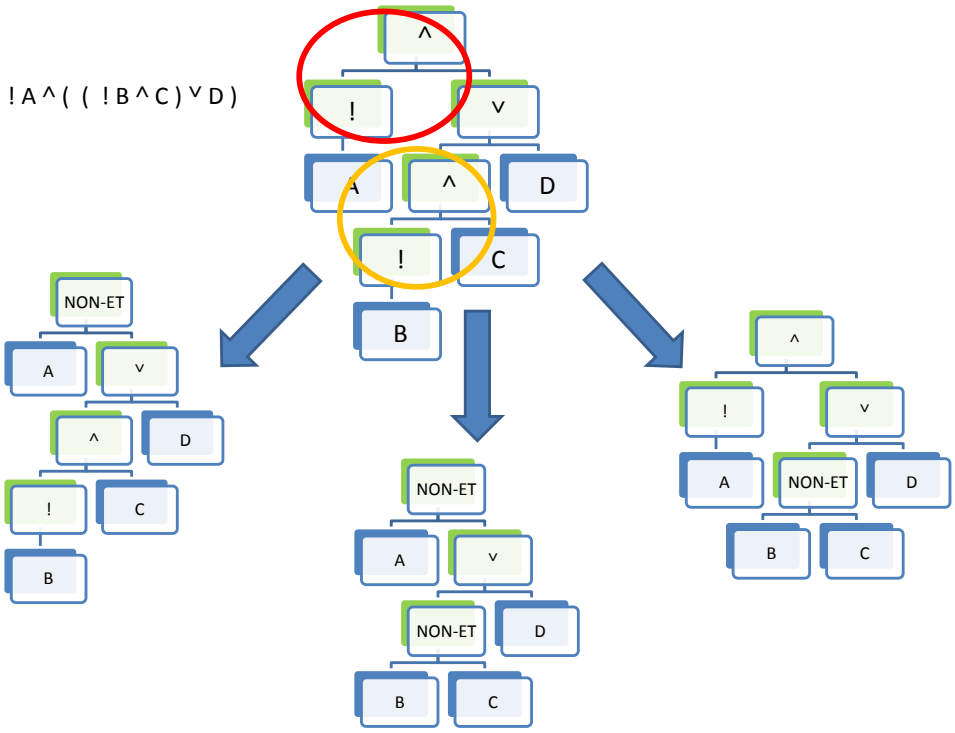


Figure 3.32 – Ajout d'arbres d'expression pour la prise en compte des portes complexes.

logiques ET et OU à deux entrées. On peut voir sur la figure 3.33 un exemple de transformation d'un arbre d'expression pour n'utiliser que des portes logiques enzymatiques à deux entrées.

Nous avons résumé toutes les étapes du processus de création de toutes les formes possibles d'une expression sur la figure 3.34. On part de la formule booléene d'origine, et on cherche toutes ses formes possibles grâce aux trois opérateurs, au développement simple, à la factorisation et à l'application des lois de De Morgan. On prend ensuite tous les arbres d'expression obtenus pour chercher si on peut remplacer certaines combinaisons de nœuds opérateurs de base par des nœuds opérateurs complexes. Quand c'est le cas, on ajoute les arbres d'expression appropriés. Enfin on cherche dans tous les arbres d'expressions résultants s'il y a des opérateurs logiques ternaires ou d'une arité plus élevée. Ces opérateurs seront alors décomposés en assemblages d'opérateurs logiques à deux entrées. Le résultat de cette dernière opération donne la liste finale des arbres d'expression à implémenter.

3.2.2 Recherche des implémentations

Nous allons maintenant décrire comment nous implémentons les arbres d'expressions avec des portes logiques enzymatiques. Notre but est d'être capable, pour chaque arbre d'expression, de construire un circuit logique enzymatique qui aura la même table de vérité et donc la même fonction logique. Ce circuit sera constitué d'un jeu de portes logiques enzymatiques fonctionnant ensemble. Nous avons représenté sur la figure 3.35 un résumé du processus d'implémentation d'un arbre d'expression. L'arbre d'expression correspondant à la formule $(A \wedge C) \vee (B \wedge C)$ comprend trois nœuds opérateurs. Il va donc falloir trouver (au moins) trois portes logiques enzymatiques qui, connectées entre elles, donneront un circuit calculant la même fonction logique. Ces portes logiques devront respecter certaines contraintes pour être capable de se connecter les unes aux autres et de fonctionner sans interactions destructives.

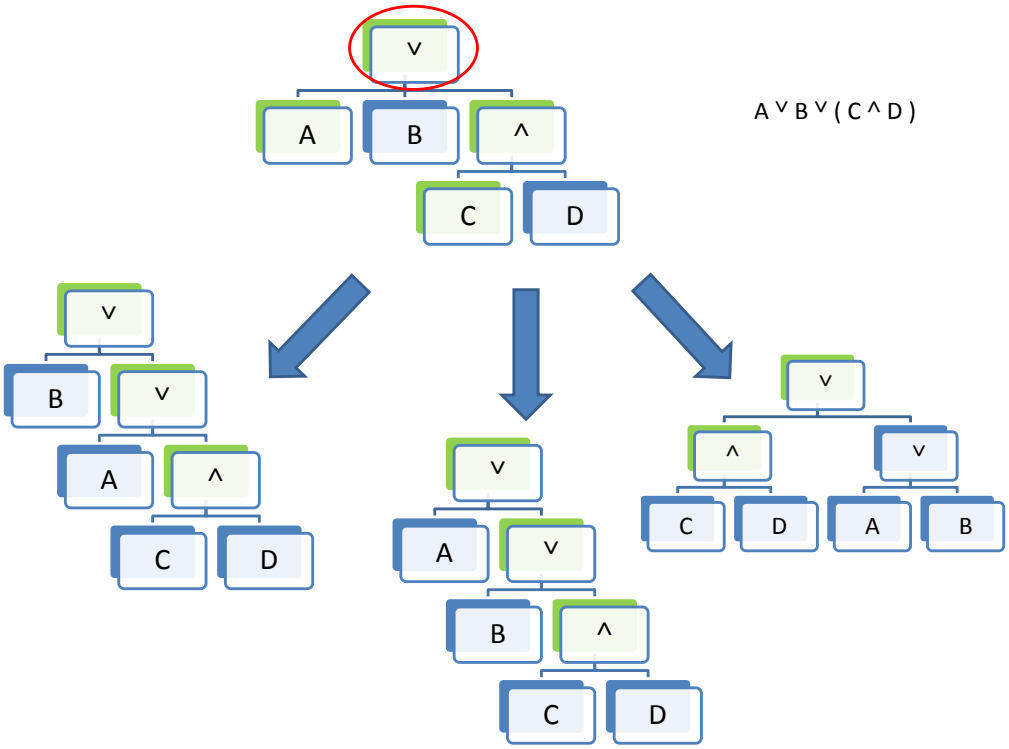


Figure 3.33 – Ajout d'arbres d'expression pour la prise en compte de portes à deux entrées seulement. L'opérateur que nous devons décomposer a trois entrées, donc il y aura deux portes à deux entrées successives. Nous allons avoir trois possibilités pour combiner ces opérateurs et trois nouveaux arbres d'expressions implémentables.

3.2.2.1 Arbre de recherche

Nous avons vu qu'un arbre d'expression sera implémenté par des portes logique enzymatiques, avec une porte correspondant à chaque nœud opérateur. Mais il est important de savoir dans quel ordre nous allons procéder, et donc quels seront les nœuds opérateurs qui vont être implémentés les premiers. Lorsque nous considérons un arbre d'expression non-implémenté nous avons déjà plusieurs informations sur le circuit logique enzymatique qui l'implémentera. En effet nous connaissons l'espèce moléculaire qui correspond à la sortie du circuit, elle devra être celle correspondant à sortie de la porte logique enzymatique implémentant le nœud *opérateur* situé à la racined de l'arbre. De même nous connaissons les espèces moléculaires des nœuds *variables* situées aux feuilles de l'arbre d'expression (entrées du circuit à réaliser). Il nous faut maintenant décider de l'ordre d'implémentation des nœuds opérateurs. Nous allons utiliser l'ordre des nœuds obtenu par un parcours en profondeur de l'arbre d'expression (voir fig. 3.36).

3.2.2.2 Règles d'assemblage des portes logiques enzymatiques

Pour qu'une porte logique enzymatique puisse être ajoutée à un circuit logique enzymatique sans causer de problème, plusieurs conditions qui doivent être respectées.

Contraintes de connexion La première série de contraintes concerne celles liées au *raccordement*. Le respect de ces conditions permettra d'une part, de connecter la porte au reste du circuit, et d'autre

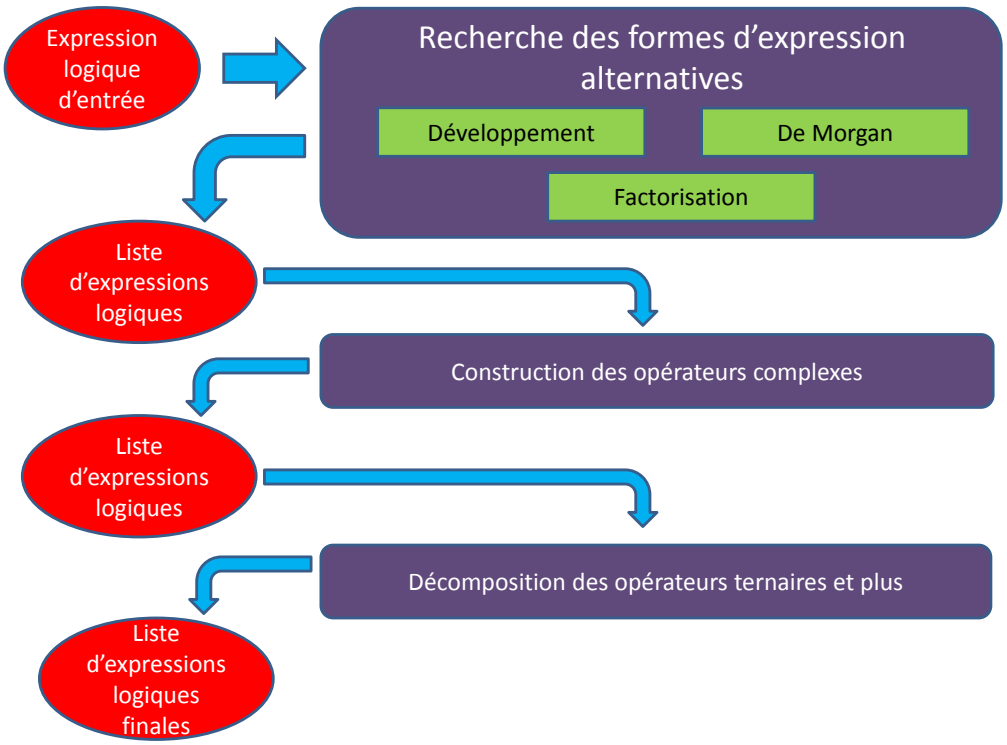


Figure 3.34 – Processus de création des formes possibles (jusqu'à une taille limite) d'une expression booléene.

part assurera que la fonction logique calculée par le circuit est correcte. Plus précisement, on va vérifier que la fonction logique effectivement implémentée par la porte qu'on ajoute est bien celle qui est représentée par le nœud opérateur. Ensuite, on va s'assurer que la porte est *connectable* au composant qui la précède et à ceux qui la suivent. Cette porte représentera un nœud opérateur au sein de l'arbre d'expression. Ce nœud peut avoir des nœuds voisins : un nœud père, et/ou des nœuds fils. On compte quatre possibilités selon la nature des nœuds voisins, opérateur ou variable, et leur relation avec le nœud considéré :

- Si ces nœuds voisins sont de type *opérateur*, alors la porte logique enzymatique sera reliée à d'autres portes logiques enzymatiques. La connexion entre portes logiques enzymatiques doit donc être assurée par une espèce moléculaire commune.
  - Si le nœud opérateur voisin est le père du nœud actuel, l'espèce moléculaire associée à la sortie de la porte que nous ajoutons doit être la même que l'espèce moléculaire associée à l'entrée de la porte implémentant le nœud père. C'est une condition de branchement *amont interne* (qui porte sur la connexion avec les composants implémentant des nœuds situés plus près de la racine de l'arbre d'expression).
  - Si le nœud opérateur voisin est un nœud fils du nœud actuel, l'espèce moléculaires associée à l'entrée de la porte logique que nous ajoutons doit être la même que l'espèce moléculaires associée à la sortie de la porte implémentant ce nœud fils. C'est une condition de branchement *aval interne* (qui porte sur la connexion avec les composants implémentant des nœuds situés plus près des feuilles de l'arbre d'expression).
- Si ces nœuds voisins sont de type *variable* alors ils sont fils du nœud actuel et la porte logique

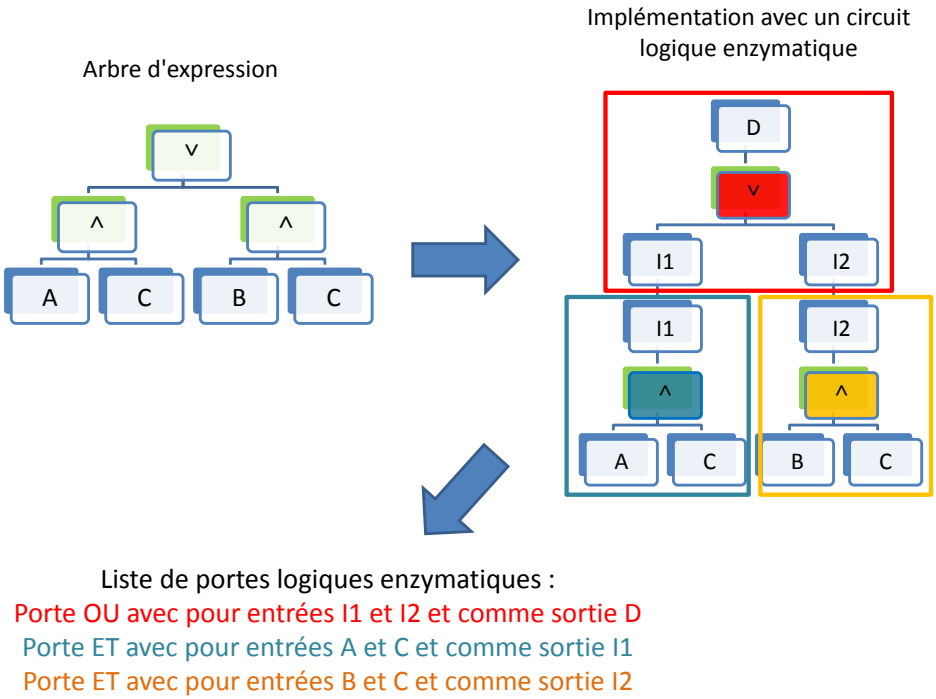


Figure 3.35 – Processus d'implémentation d'un arbre d'expression par un circuit logique enzymatique.

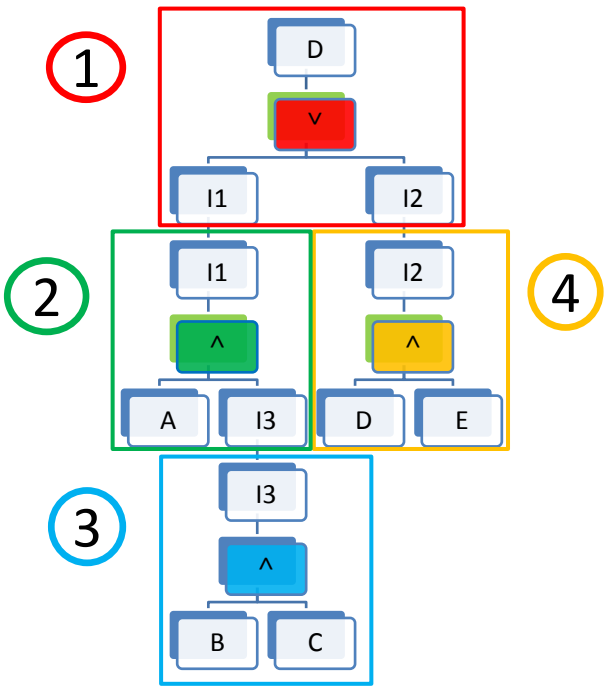


Figure 3.36 – Ordre d'implémentation d'un arbre d'expression : parcours en profondeur.

enzymatique est branchée sur des entrées du circuit. L'espèce moléculaire associée à l'une des entrées de la porte que nous ajoutons doit être la même que celle associée à l'entrée en question du circuit. C'est une condition de branchement *aval externe* (qui porte sur la connexion aux feuilles de l'arbre d'expression).

- Si il n'y a pas de nœud voisin père, le nœud actuel est la racine de l'arbre d'expression, donc la porte logique enzymatique est branchée sur la sortie finale du circuit. L'espèce moléculaire associée à la sortie de la porte que nous ajoutons doit être la même que l'espèce moléculaire associée à la sortie finale du circuit. C'est une condition de branchement *amont externe* (qui porte sur la connexion à la racine de l'arbre d'expression).

Il n'est néanmoins pas nécessaire d'utiliser toutes les conditions de branchement pour l'implémentation de tous les nœuds opérateurs. En effet pour vérifier les conditions aval, il faut que l'arbre soit implémenté en aval du nœud actuel. De même, si nous voulons vérifier les conditions amont alors il faut que l'arbre soit implémenté en amont du nœud actuel. Selon l'ordre dans lequel on va lire et implémenter les arbres d'expression et la position du nœud opérateur à implémenter dans cet arbre, on va utiliser les conditions de branchement aval ou amont. Nous avons vu précédemment qu'on commençait par la racine, puis qu'on descendait dans l'arbre. L'amont d'un nœud est donc toujours construit, par contre l'aval ne l'est presque jamais. Les seuls cas où nous avons des nœuds opérateurs dans l'arbre avec à la fois l'aval et l'amont déjà implémentés, sont lorsqu'on cherche à implémenter des nœuds *opérateur* situés juste avant les feuilles de l'arbre. Leurs avals sont les espèces moléculaires les nœuds *variable* des feuilles connectées au nœud *opérateur* qu'on cherche à implémenter.

De plus, il est également possible pour un nœud *opérateur* d'avoir des contraintes sur certains nœuds fils uniquement. Si l'un de ses fils est un nœud *variable* et l'autre, un nœud *opérateur*, alors il n'y aura de contrainte que sur le lien avec le nœud *variable*. Nous pouvons établir un algorithme simple pour décider des conditions de branchement à contrôler pour un nœud *opérateur* Nop_i :

— Si Nop_i est la racine de l'arbre d'expression alors il y aura une condition de branchement amont externe.
— Si Nop_i a un fils qui est un nœud variable alors il y aura une condition de branchement aval externe.
— Si Nop_i n'est pas la racine de l'arbre d'expression alors il y aura une condition de branchement amont interne.

Les conditions aval internes ne sont utilisées que si nous implémentons l'arbre d'expression en partant des feuilles vers la racine, ce qui n'est pas le cas choisi. Nous avons représenté sur la figure 3.37 deux possibilités d'ajout d'une nouvelle porte logique enzymatique avec, pour chaque situation, les conditions de branchement.

Contraintes de collisions La deuxième série de contraintes concerne celles liées aux *collisions*. Leur but est de vérifier que l'ajout au circuit d'une porte logique enzymatique ne provoquera pas de dysfonctionnement. Comme nous l'avons vu dans le chapitre 2, il est facile de provoquer des courts-circuits au sein d'un circuit logique enzymatique. Les fils de connexion étant implémentés par des espèces moléculaires spécifiques, si la même espèce moléculaire est utilisée à deux points différents du circuit, cela constituera une connexion entre ces deux points. Cette propriété est utilisée quand on veut connecter deux composants du circuit, mais si c'est accidentel, cela provoque un *court-circuit*.

La façon la plus simple de gérer les collisions au sein d'un circuit logique serait d'interdire, au circuit et à la porte que l'on ajoute, d'avoir des éléments communs (à part, bien sûr, l'espèce moléculaire qui connecte la porte au circuit). Dans un premier temps, nous avions adopté cette solution.

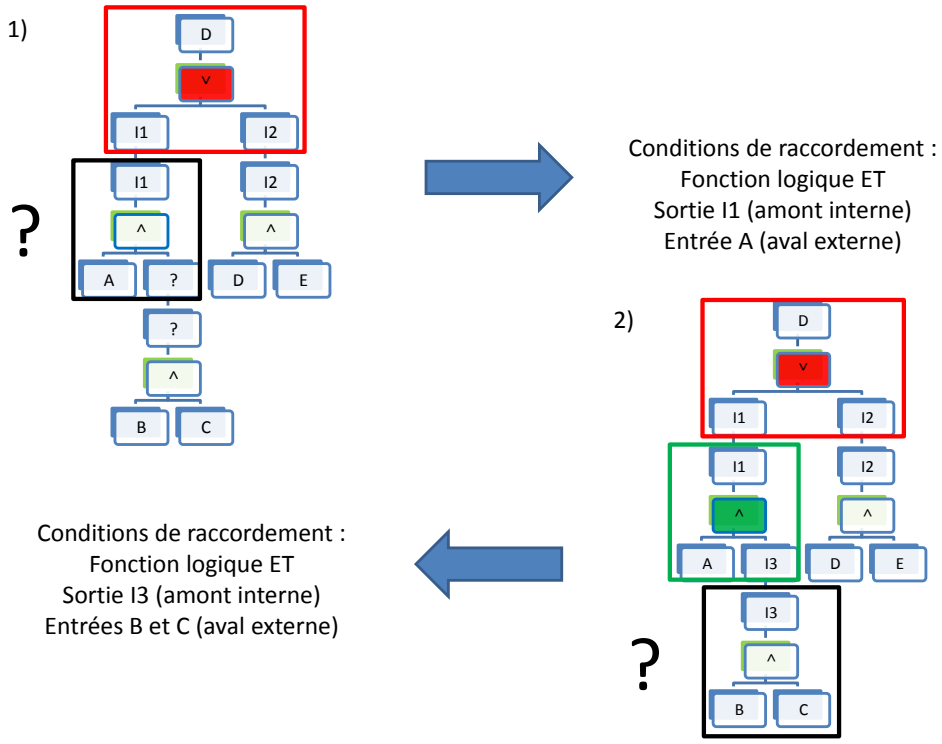


Figure 3.37 – Ajout de deux nouvelles portes logiques enzymatiques. Pour l'ajout **1** (encadré noir en haut à gauche) on applique les conditions de branchement sur la sortie de la porte ainsi que sur l'une des entrées, le nœud opérateur n'étant pas la racine, c'est une condition amont interne. L'un des nœuds fils est de type *variable*, l'entrée **A** du circuit, donc il y une condition aval externe. Pour l'ajout **2** (encadré noir en bas à droite), on applique les conditions de branchement sur la sortie de la porte ainsi que sur les deux entrées. Le nœud opérateur n'étant pas la racine, c'est une condition amont interne. Les deux nœud fils étant de type *variable*, les entrées **B** et **C**, il y deux conditions aval externes.

Nous avions de bons résultats, néanmoins une faiblesse est apparue : certains circuits fonctionnels conçus manuellement ne pouvaient pas être reproduits par NetBuild. A cette occasion, on s'est aperçu qu'il était tout à fait possible que deux portes logiques enzymatiques, bien qu'ayant des éléments en commun, fonctionnent ensemble correctement, à condition que ces éléments respectent plusieurs propriétés. On peut voir sur la figure 3.38 que le premier cas n'aurait pas pu être trouvé par cette version de NetBuild, malgré le fait que ce soit un circuit fonctionnel. Le métabolite **D** étant présent en grand quantité dans l'environnement, il peut donc être consommé par plusieurs réactions sans problèmes. Dans le deuxième cas, **D** n'etant pas un composant de la deuxième porte ET, sa compatibilité avec la première porte n'aurait pas été vérifiée, et cela aurait conduit à un circuit logique enzymatique faux. Nous avons donc utilisé des contraintes de collisions plus spécifiques. Nous avons créé une classification des éléments composants une porte logique enzymatique (voir fig. 3.39).

- **Éléments négatifs** : les espèces moléculaires qui doivent être absentes pour que la porte fonctionne.
- **Éléments constitutifs** : les espèces moléculaires qui sont produites et consommées au sein du réseau enzymatique, ainsi que celles des entrées de la porte logique.
- **Éléments libres** : les espèces moléculaires qui sont produites ou bien consommées au sein du réseau enzymatique, en excluant celles des entrées et de la sortie de la porte logique.
- **Enzymes** : les enzymes utilisées dans le réseau métabolique.

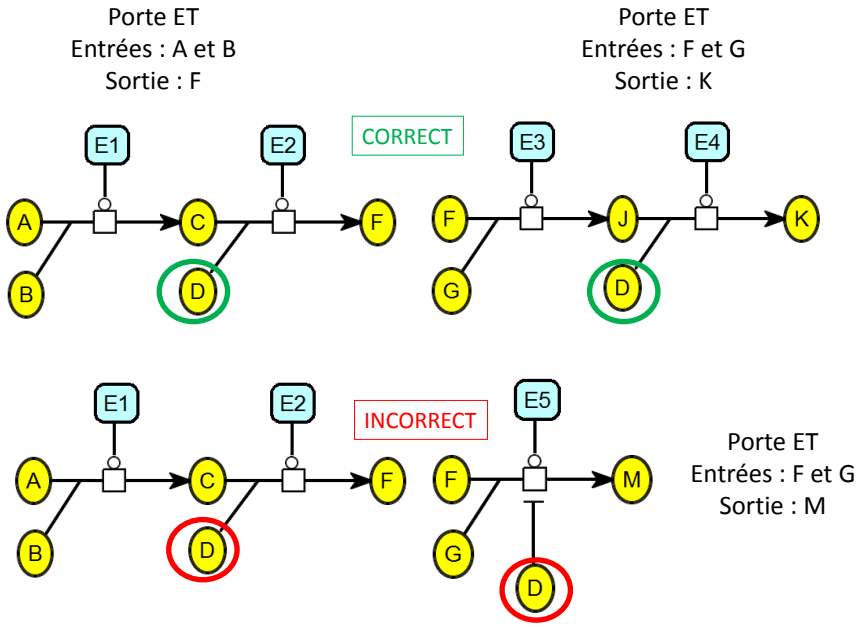


Figure 3.38 – Exemples d'ajout de porte où les contraintes simples ne suffisent pas. Le circuit du premier cas, bien qu'étant fonctionnel n'aurait pas été trouvé. Le circuit du deuxième cas, bien que non fonctionnel (à cause de **D** qui ne peut pas être à la fois présent et absent) aurait été proposé.

Un circuit logique enzymatique, de même qu'une porte logique enzymatique, aura les espèces moléculaires le composant réparties dans ces 4 catégories. Au lieu de comparer globalement l'ensemble des composants de la porte à ajouter à l'ensemble des composants du circuit pour vérifier sil y a des collisions, on va utiliser ces catégories pour effectuer des comparaisons plus fines. Le tableau 3.5 regroupe tous les tests à effectuer. Les éléments négatifs sont ceux qui empechent le fonctionnement du composant, il ne faut donc pas qu'ils fassent partie des molécules présentes dans les autres composants. Par contre, plusieurs composants peuvent partager des éléments négatifs. Les éléments constitutifs doivent absolument être uniques dans le réseau, car ce sont des éléments forcément présents à un moment donné, et porteurs d'une information. Enfin les éléments libres sont les éléments dont la quantité n'est pas importante, mais qui sont globalement présents dans le système. Ils ne doivent donc pas faire partie des éléments négatifs ou constitutifs, par contre, plusieurs composants peuvent partager des éléments libres.

Si toutes les conditions de branchement et de collisions sont respectées, la porte logique enzymatique peut être ajoutée au circuit logique enzymatique sans risque.

		Catégories pour la porte à ajouter		
		Négatifs	Constitutifs	Libres
Catégories pour le circuit	Négatifs	OUI	NON	NON
	Constitutifs	NON	NON	NON
	Libres	NON	NON	OUI

Table 3.5 – Conditions de collision pour l'ajout d'une porte logique enzymatique à un circuit.

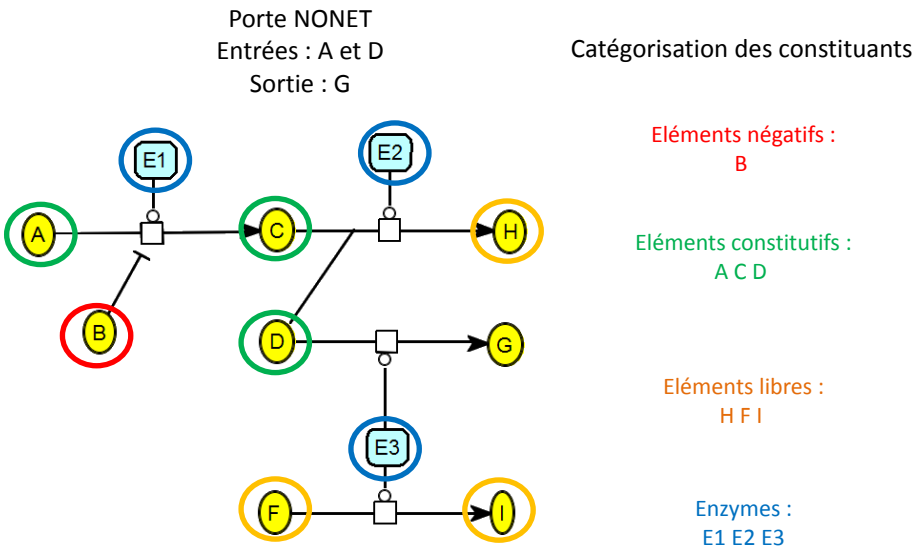


Figure 3.39 – Exemple de classification des composants d'une porte logique enzymatique. On peut noter que comme l'enzyme **E3** catalyse deux réactions, celle qui est inutile consomme et produit des métabolites *libres* : **F** et **I**.

3.2.2.3 Rétro-propagation des contraintes

Le processus d'implémentation d'un arbre d'expression se fait dans un ordre déterminé dans lequel, pour chaque nœud *opérateur*, on va chercher s'il existe une porte compatible. Cette recherche s'effectue en plusieurs étapes. Les portes sont stockées dans une table de hachage (la bibliothèque de portes) où la clef est constituée de la fonction logique de la porte ainsi que du métabolite correspondant à sa sortie. Lorsque la recherche dans la bibliothèque se fait, on peut déjà utiliser certaines des contraintes de branchements pour en extraire une liste de portes potentielles. Ensuite on va pouvoir vérifier le reste des contraintes de branchements, puis enfin les contraintes de collisions.

Il est possible (et souhaitable) qu'après toutes ces vérifications, plusieurs portes soient utilisables pour un même nœud *opérateur*. Il est aussi possible qu'aucune porte n'ait réussi à passer les tests. Ces deux cas de figures vont perturber le parcours de l'arbre d'expression.

Quand plusieurs portes sont disponibles pour implémenter le même nœud *opérateur*, on va sélectionner la première de la liste, puis continuer à suivre l'ordre d'implémentation des nœuds. Quand tous les nœuds auront été implémentés (on aura alors obtenu une première implémentation complète) on va retourner en arrière jusqu'au nœud le plus proche comportant des portes potentielles pas encore sélectionnées (*backtrack*). On va alors sélectionner une autre porte pour ce nœud et reprendre l'ordre normal de parcours de l'arbre. Ce processus va s'appliquer à tous les nœuds qui sont implémentables par plusieurs portes potentielles.

De même, si on ne trouve pas de porte dans la bibliothèque pour implémenter un nœud, on va retourner en arrière dans l'arbre d'expression jusqu'à un nœud qui a plusieurs implémentations

possibles et en choisir une autre (voir fig 3.40).

Nous avons vu toutes les grandes étapes de l'implémentation d'un arbre d'expression par un circuit enzymatique, néanmoins, il reste quelques points particuliers à expliquer. Il peut arriver de ne pas trouver de porte dans la bibliothèque ayant la bonne sortie et la bonne fonction logique. Dans ce cas, avant de revenir en arrière dans l'arbre, il reste quelques autres possibilités à explorer. Ces possibilités, *extensions*, sont en fait testées systématiquement, qu'une porte ait été trouvée ou non pour implémenter un nœud *opérateur*.

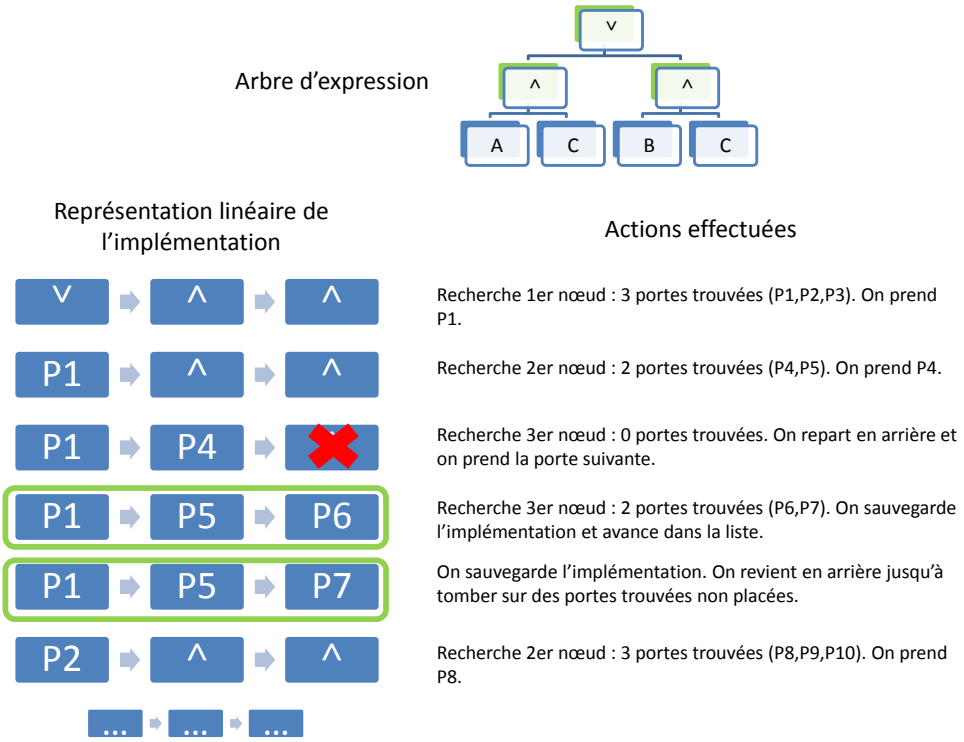


Figure 3.40 – Processus d'implémentation d'un arbre d'expression. Seules les premières étapes sont affichées. L'implémentation par 3 portes est représentée sous forme linéaire, plus pratique à visualiser.

Extension par porte OUI La première de ces possibilités est l'extension par porte OUI. La porte enzymatique OUI réalise le changement de nature de l'espèce moléculaire représentant un fil de connexion. Cela permet de changer des contraintes de branchement liées d'un nœud *opérateur*.

On va essayer d'insérer une porte OUI devant le nœud que nous cherchons à implémenter en décalant ce nœud d'un cran vers le bas de l'arbre (voir fig. 3.41). Cela permet de multiplier les possibilités de portes par changement de la nature des entrées et des sorties de portes existantes et donc de trouver des implémentations qui ne l'auraient pas été sinon. Bien sûr, cela aurait pu être déja fait par *NetGate*, mais il aurait fallu augmenter la taille maximum des portes cherchées et par conséquent le temps de recherche (qui est exponentiel en fonction de la taille des portes).

Extension par porte OU abstraite La deuxième possibilité d'extension consiste à ce que NetBuild construise *au vol* des portes logiques enzymatiques OU. Ces portes OU seront spéciales en ce sens qu'elles sont construites sur le shéma $A = A \vee A$. Cette porte est utilisable dès qu'il y a présence de

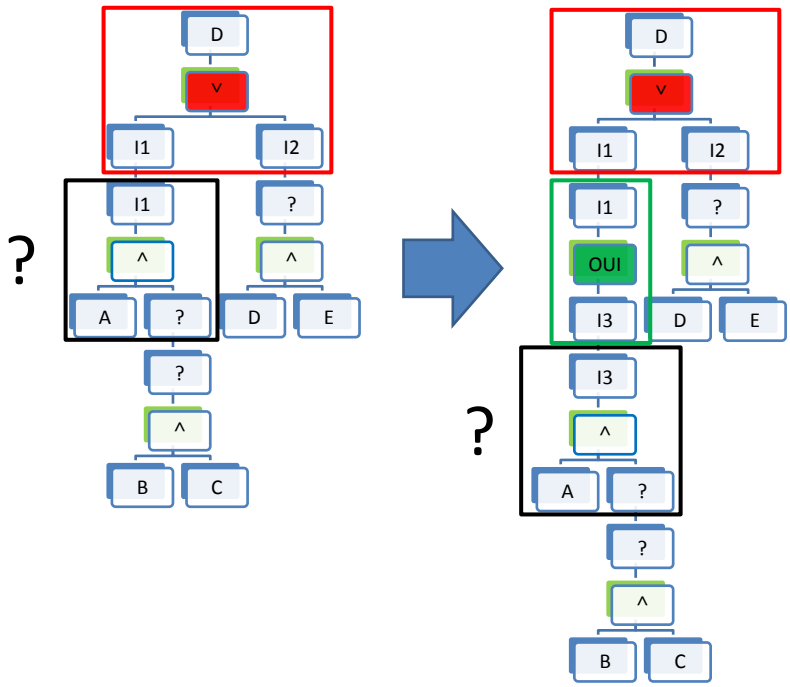


Figure 3.41 – Exemple d'utilisation d'une porte OUI ajoutée entre deux nœuds opérateurs. Cela permet d'utiliser une porte ET qui sort le métabolite I3 pour alimenter la porte OU qui a besoin de I1.

A. Cela peut paraitre inutile à première vue, mais va permettre de décomposer un nœud opérateur OU à l'aide de portes OUI. En effet, nous savons qu'une implémentation simple de la porte OU est faite de deux réactions consommant des espèces moléculaires différentes et produisant la même espèce moléculaire. Avec des portes OUI devant l'une et/ou l'autre des entrées, NetBuild va pouvoir construire ces portes directement (voir fig. 3.42).

3.2.3 Validation

NetBuild a été programmé dans le langage objet C++. Il a été compilé avec *gcc* sur les systèmes d'exploitation les plus répandus, Linux, MacOSX et Windows, sur des architectures 32 bits et 64 bits. Le programme source représente environ 4000 lignes de code.

Nous avons utilisé la bibliothèque de portes trouvées par NetGate à partir d'un réseau composé du métabolisme central du carbone de *B. subtilis* et d'un réseau artificiel, ayant 35 enzymes. En moins d'une seconde, sur un laptop standard, NetBuild a pu fournir des implémentations de quelques expressions logiques données en entrée. Nous avons aussi testé NetBuild à partir de réseaux où d'autres chercheurs avaient trouvé manuellement des circuits logiques enzymatiques correspondant à des fonctions booléenes simples. NetBuild a pu retrouver tous ces circuits de façon automatique.

3.2.4 Intégration avec NetGate

NetBuild a été construit pour être utilisé avec NetGate, mais ce n'est bien sûr pas une obligation. Toutes les bibliothèques de portes logiques enzymatiques pourront être utilisées par NetBuild de la même façon. Ces deux logiciels ont néanmoins été conçus pour accomplir ensemble un but bien précis,

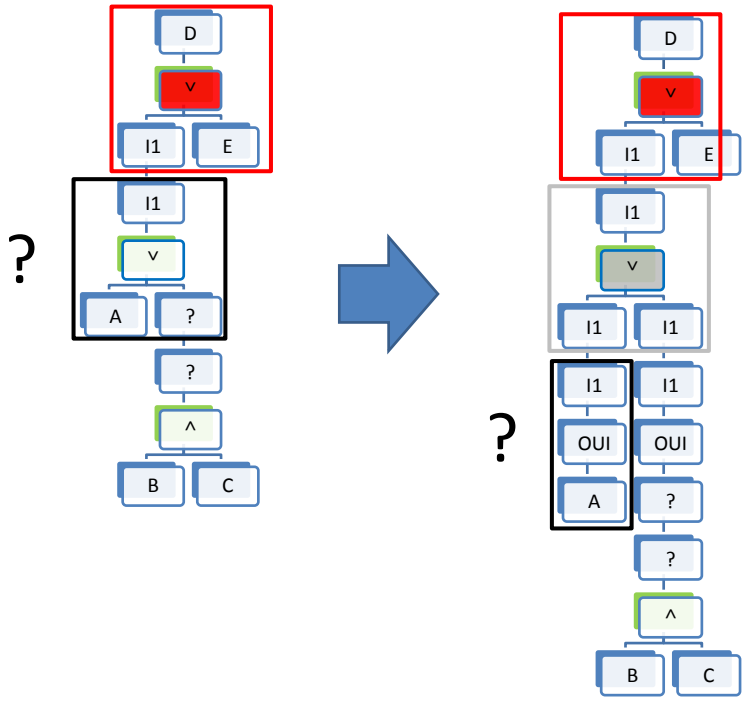


Figure 3.42 – Ajout d'une porte OU abstraite pour construire une porte OU *au vol*.

pouvoir construire des circuits logiques enzymatiques répondant à une expression logique spécifiée en entrée, à partir de réseaux métaboliques existants. Il existe donc des synergies et des liens particuliers qui les unissent. Nous avons vus lors de la description du fonctionnement de NetGate que l'algorithme qu'il implémente pourrait aussi construire des circuits logiques enzymatiques puisque tout circuit logique peut être assimilé à une porte logique complexe, mais que le temps qu'il mettrait pour accomplir cette tâche le rendrait inutilisable. NetBuild est donc indispensable pour pouvoir construire des circuits logiques enzymatiques car il permet de trouver des circuits très rapidement et sans difficultés.
Si on fait abstraction du temps calcul, on pourrait constater que l'utilisation de NetGate uniquement n'est pas équivalente à l'utilisation de NetGate suivi de NetBuild. L'association NetGate plus NetBuild trouvera moins de circuit que NetGate tout seul. Cette différence est due à la méthode de recherche qui est fondamentalement différente. NetGate va chercher à associer des fonctions logiques à des réseaux métaboliques qu'il va simuler, c'est une méthode de test *boite noire*, cela fonctionnera quelle que soit la topologie interne du réseau. NetBuild lui, ne peut trouver des circuits logiques enzymatiques que s'ils ont une forme arborescente. NetGate est fait pour trouver des portes logiques enzymatiques avec très peu de contrainte sur la forme de ces portes (seule la table de vérité importe). NetBuild est fait pour organiser et construire des circuits respectant un modèle précis, il est moins flexible, mais il est beaucoup plus rapide.

Il n'est néanmoins pas nécessaire de faire un choix tranché entre NetGate et l'association NetGate-NetBuild. D'une part parce que NetGate tout seul est rapidement inutilisable dès que le réseau initial prend un peu d'ampleur et qu'on recherche des portes avec beaucoup de réactions, et d'autre part parce que nous pouvons régler le degré du couplage entre NetGate et NetBuild. En effet, dans NetGate, nous pouvons contrôler la taille maximale des portes logiques enzymatiques recherchées. Plus cette taille sera petite plus NetGate sera rapide, mais plus NetBuild va devoir *construire* et donc le circuit logique enzymatique final sera *ordonné*. À l'inverse, plus cette taille sera grande, plus NetGate sera lent, et

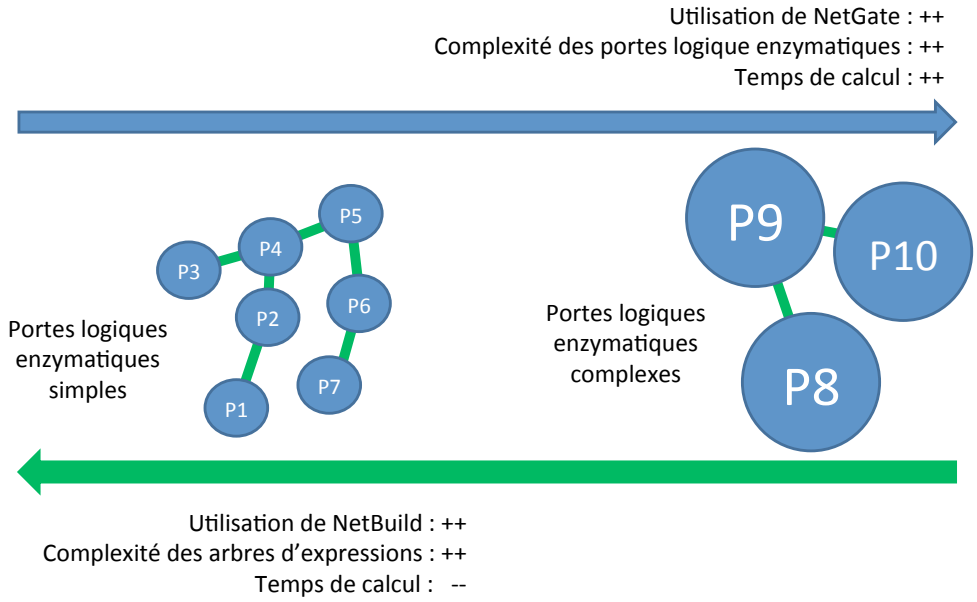


Figure 3.43 – Propriétés de la relation entre NetGate et NetBuild.

moins NetBuild va devoir construire ; en conséquence, le circuit logique enzymatique final sera moins *ordonné*. Nous avons représenté sur la figure 3.43 ces compromis entre NetGate et NetGate+NetBuild.

3.3 Disposition lisible d'un réseau enzymatique : NetPlace

Un des moyens le plus intuitif pour représenter des informations est d'utiliser un graphe : il y a des éléments, et ces éléments ont des relations entre eux. Il existe un grand nombre de genres de graphes, chacun adapté à la représentation d'un type particulier d'informations. Pour que ces informations portées par des graphes soient comprises par des humains, on va avoir besoin de les visualiser de façon *lisible*. Visualiser un graphe consiste à afficher ses nœuds (les éléments) et ses arcs (les relations) à l'écran ou sur papier, de façon à ce que l'utilisateur puisse rapidement appréhender l'information contenue. L'affichage d'un graphe est une opération qui peut être divisée en deux parties :

— le dessin en lui-même : la forme, la couleur, etc. des nœuds et des arcs,
— le placement des nœuds sur un plan (une fenêtre à l'écran) qui va conditionner la forme générale du graphe.

De nombreux logiciels sont très performants pour effectuer la première partie, le dessin, mais le placement est souvent fait *à la main*. Notre but ici est de concevoir une méthode automatique de placement, bien adaptée à l'affichage des réseaux métaboliques.

3.3.1 Les graphes des réseaux métaboliques

La définition du graphe général est la donnée d'un ensemble de nœuds et d'un ensemble d'arcs qui connectent les nœuds entre eux. Un réseau métabolique a une structure plus contrainte que celle d'un graphe général : il est constitué de réactions (souvent catalysées par des enzymes) qui transforment des espèces moléculaires (substrats) en d'autres espèces moléculaires (produits).

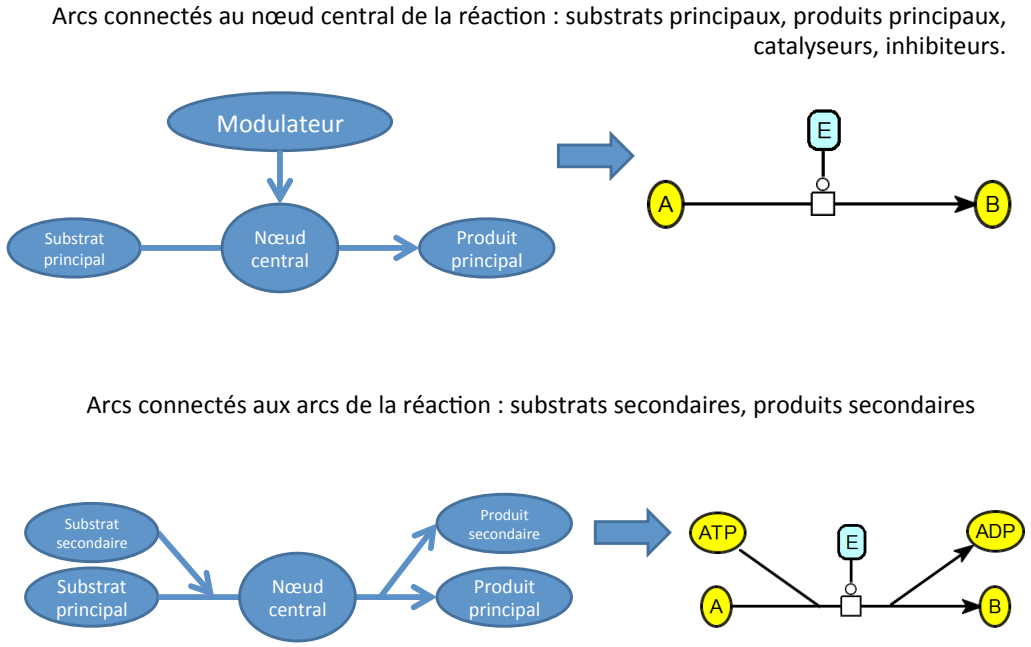


Figure 3.44 – Représentation graphique d'une réaction enzymatique. Ces deux exemples montrent que les réseaux métaboliques sont des hyper-graphes. Sur la partie gauche on voit la forme schématique de la réaction et sur la partie droite un tracé possible.

Si on considère que les espèces moléculaires sont les nœuds et les réactions les arcs, cela ne sera pas suffisant, en effet, une réaction peut être catalysée par une enzyme (ou inhibée par une espèce moléculaire) et peut également avoir plus d'un substrat ou plus d'un produit. Enfin les réactions

enzymatiques possèdent une forme graphique bien particulière qu'il va falloir respecter pour les afficher (e.g. SBGN Process Description language [57]). C'est donc un graphe non trivial à représenter, un hyper-graphe, où les arcs connectant les espèces moléculaires consommées et produites peuvent aussi posséder un nœud interne qui pourra être connecté à d'autres espèces moléculaires qui agiront comme modulateurs de la réaction. Nous avons représenté deux situations fréquemment rencontrées dans les réseaux métaboliques sur la figure 3.44. Sur la gauche nous avons une représentation purement schématique de la réaction enzymatique et sur la droite un exemple possible de tracé lui correspondant. Par ailleurs, les réactions biochimiques peuvent être unidirectionnelles ou bien réversibles.

Nous allons faire une distinction entre tous les substrats d'une réaction. Nous pouvons voir sur le schéma que certains substrats et certains produits sont dans le même axe, alors que les autres se greffent sur cet axe principal. Nous appellerons les substrats et produits principaux, ceux qui sont dans l'axe principal, et substrats et produits secondaires les autres.

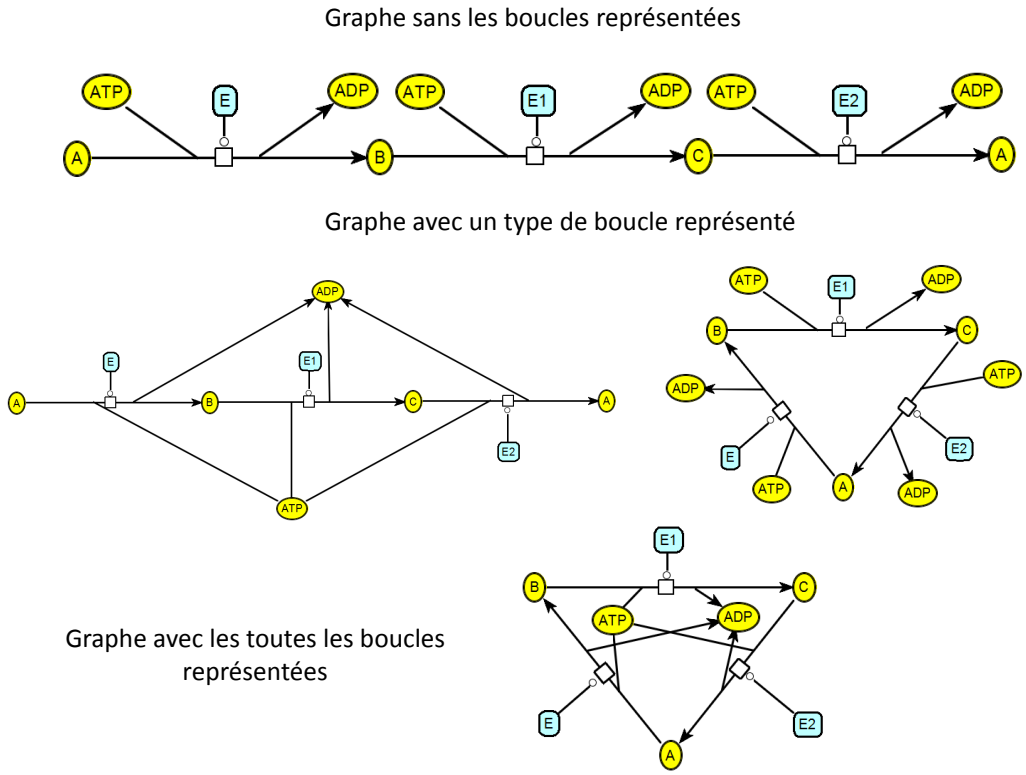


Figure 3.45 – Différentes façons de dessiner les boucles/cycles d'un réseau métabolique.

Un autre point important concernant les réseaux métaboliques est la présence de boucles qui ont des rôles bien spécifiques. La boucle (cycle ou bien circuit) est une topologie importante dans tous les graphes et il est nécessaire de lui porter une attention particulière. En effet, la représentation des boucles est en général une contrainte forte sur la lisibilité des graphes. Dans les réseaux métaboliques, certains types de molécules participent à beaucoup de réactions enzymatiques et vont donc être représentés comme des nœuds très connectés, ce qui va surcharger le tracé du graphe. Nous allons donc différencier deux types de boucles : celles qui montrent une information utile et celles qui représentent des informations déjà connues implicitement par l'utilisateur. Cette distinction est déjà respectée lors du tracé manuel des réseaux métaboliques pour privilégier l'information utile. Sur la figure 3.45 on

peut voir le même réseau représenté de différentes façons. Ce réseau comporte deux types de boucles : une que nous allons considérer utile, le triangle A-B-C, et une autre, en fait un cycle si on ignore l'orientation des arcs, mettant en jeu l'ATP et l'ADP, qui sera considérée comme moins importante. On peut constater que selon la façon dont le réseau est tracé, et les nœuds placés, le rendu visuel est plus ou moins lisible.

Nous pouvons en général différencier les boucles utiles des autres en fonction des molécules impliquées et de leur rôle. En première approximation, les boucles utiles sont celles qui font intervenir les substrat ou produit principaux d'une réaction enzymatique, alors que les celles qui utilisent des substrats ou produits secondaires sont de moindre importance.

3.3.1.1 Qu'est-ce qu'un beau graphe ?

Avant de chercher à savoir comment dessiner un graphe, nous devons répondre à une question essentielle : quelle forme doit-il prendre ? Il existe de nombreuses possibilités. Un graphe étant constitué de nœuds et d'arcs, les nœuds peuvent être placés n'importe où dans l'espace de dessin, et les arcs peuvent avoir un grand nombre de formes. Notre but est d'afficher un graphe afin de pouvoir représenter une information sous une forme spécifique, et la qualité primordiale que nous recherchons est la lisibilité. Bien entendu, ce critère peut paraitre subjectif, mais plusieurs auteurs ont établi des objectifs à atteindre pour tracer un *beau* graphe [58] :

1. **nombre d'arcs qui se croisent** : plus il y a de croisement et moins le graphe est lisible, néanmoins tous les graphes ne sont pas planaires c'est donc l'un des critère le plus difficile à respecter.
2. **taille totale du graphe** : plus le graphe est compris dans une petite zone, plus il sera facile de l'appréhender dans son ensemble.
3. **symétrie** : le graphe doit être le plus symétrique possible, l'esprit humain aime la symétrie.
4. **forme des arcs** : les arcs doivent avoir la forme la plus simple possible. Les arcs rectilignes sont les plus faciles à comprendre.

Nous avons représenté graphiquement ces critères sur la figure 3.46. On voit que même si certains peuvent se combiner avec synergie (3 et 4) ce n'est pas le cas de tous, et ce sera d'autant plus vrai que les graphes que nous voulons modéliser seront complexes.

Ces critères sont valables pour toutes les sortes de graphes, quelle que soit l'information qu'ils serviront à représenter. Mais chaque type d'informations peut avoir des spécificités supplémentaires. Un graphe modélisant une machine a état ne se sera pas tracé avec les mêmes objectifs qu'un workflow décrivant le cycle de développement d'une application web par exemple. Dans notre cas de circuits logiques enzymatiques, l'information principale est le réseau métabolique qui les constitue. Nous allons donc nous intéresser plus spécifiquement aux critères d'esthétisme associé aux réseaux métaboliques. Ces critères ont été définis par plusieurs auteurs [59, 60] et nous allons les résumer ici :

— Les nœuds du graphe doivent refléter la nature et le rôle des espèces moléculaires qu'ils vont représenter.
— La forme des réactions enzymatiques doit être caractéristique. Comme nous l'avons vu dans la partie sur le tracé des graphes, les réactions enzymatiques ont une forme bien particulière qui doit être respectée, cette forme va induire des contraintes fortes sur le placement des nœuds.
— Le graphe doit refléter les voies métaboliques d'intérêt du réseau. Les voies les plus pertinentes pour le lecteur doivent être représentées de façon évidente.

Nous pouvons voir que ces critères ne sont plus seulement esthétiques comme l'était les critères précédents mais qu'ils dépendent spécifiquement du public auquel le graphe est destiné. Donc deux graphes

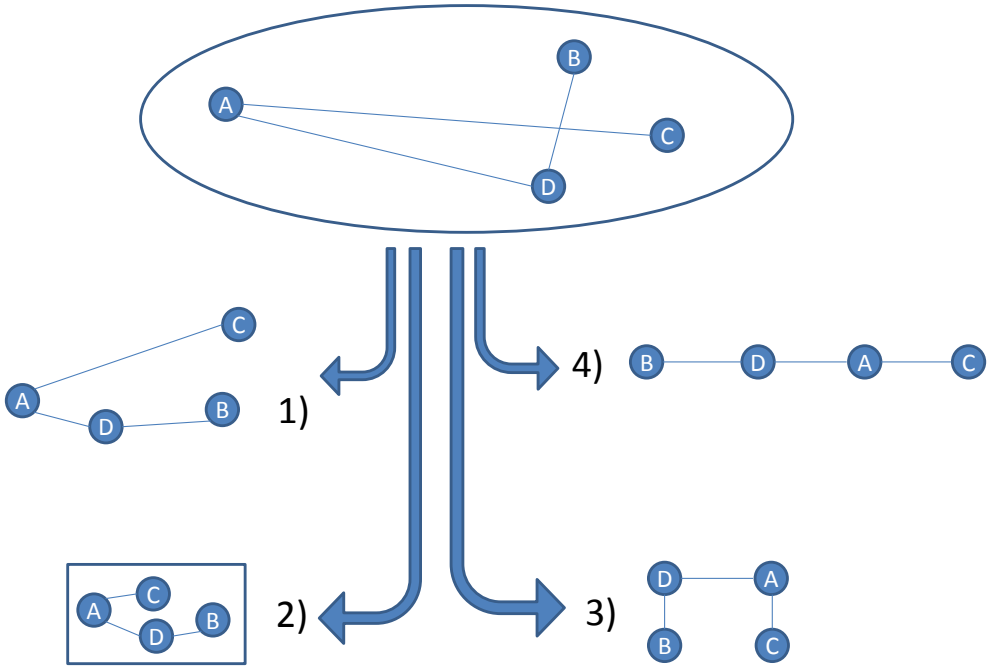


Figure 3.46 – Représentation graphique des critères pour obtenir des *beau* graphe. Les quatre critères sont 1) Le nombre d'arcs qui se croisent, 2) La taille total du graphe, 3) Le symétrie du dessin et 4) Les arcs doivent être de la forme a plus simple possibles.

identiques en nature et comprenant exactement les mêmes éléments à décrire donc les mêmes réactions enzymatiques, pourraient prendre des formes très différentes selon le but que les auteurs des graphes cherche à atteindre.

3.3.1.2 Algorithmes de tracé de réseaux métaboliques existants

Il existe déjà de nombreuses méthodes de tracé automatique de réseaux, elles n'ont en général pas été conçues spécifiquement pour les réseaux métaboliques, mais plutôt adaptées et paramétrés pour donner des graphes ressemblant à des réseaux métaboliques dessinés manuellement. Deux grandes méthodes dominent le placement automatisé de réseaux métaboliques, nous allons les décrire très succinctement :

— **Placement dirigé par les forces** [59] : ce type de placement est basé sur une simulation physique, les nœuds sont reliés entre eux par des fils agissant comme des ressorts, ils se repousseront s'ils trop proches et s'attireront s'ils sont trop éloignés.
— **Dessins hiérarchique** [60] : cette méthode va considérer le graphe à représenter comme un arbre, elle va partir de la racine pour placer les nœuds relativement à cette racine.

Bien que ces algorithmes produisent des résultats intéressants, ils ne vont pas satisfaire les contraintes que nous nous sommes fixées : les réactions seront en général difficilement reconnaissables puisque de tracé non-uniforme. La première méthode va produire des réactions sans forme homogène car chaque élément de la réaction pourra être déplacé librement par rapport aux autres. La deuxième méthode se rapproche plus de ce que nous souhaitons, mais malgré tout, les réactions seront aussi étirées à travers

le graphe. Notre but étant d'avoir un algorithme de placement pour représenter des circuits logiques enzymatiques, il est impératif que les réactions soient représentées le plus clairement possible. Nous allons donc concevoir une méthode originale de placement respectant ces objectifs.

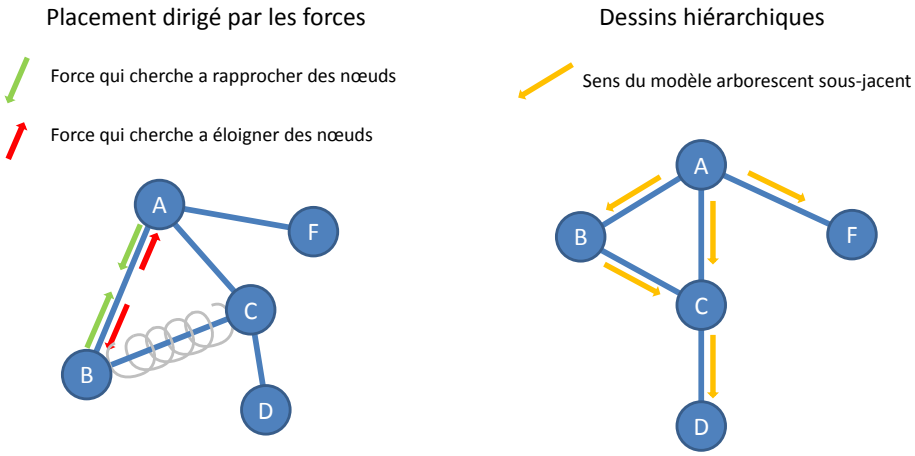


Figure 3.47 – Représentation schématique des deux méthodes de placement de graphe présentées.

3.3.2 Comment tracer des graphes métaboliques

En considérant que les arcs qui relient les nœuds du réseau sont rectilignes, placer un réseau métabolique revient à placer les nœuds représentant les différentes espèces moléculaires le constituant. Une réaction sera tracée comme un segment de droite entre le substrat et le produit principal. Le nœud interne de la réaction sera automatiquement placé au milieu de ce segment. Les substrats secondaires, produits secondaires et modulateurs seront reliés par des arcs au nœud interne de la réaction associée. C'est une approximation qui nous suffira pour le calcul du placement. Dans les faits, *NetPlace* calcule le placement de nœuds ainsi que leur rôle, à un logiciel de dessin de graphe indépendant qui tracera réellement les nœuds et les arcs à partir des informations fournies.

Sachant que nous n'avons à placer que les nœuds, nous allons utiliser une grille qui va organiser le placement des éléments. La présence de cette grille implique que le placement des nœuds ne sera pas complètement libre : au lieu de calculer leur position réelle sur le dessin, nous allons utiliser leur coordonnées sur la grille (voir fig. 3.48). Ce type de placement va nous permettre de gagner en homogénéité, les nœuds vont être alignés verticalement et horizontalement.

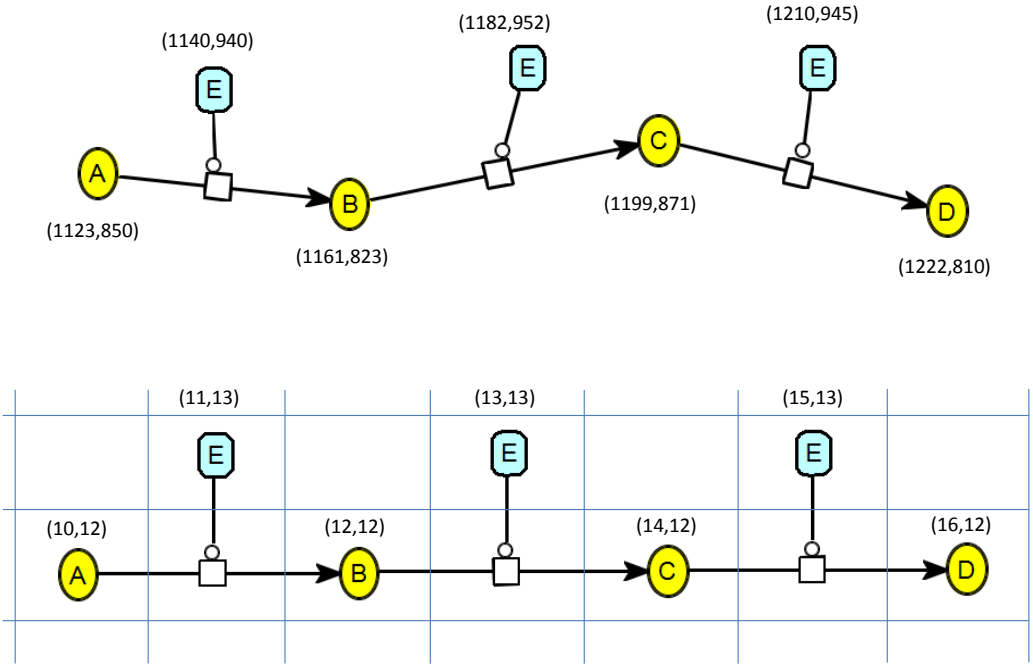


Figure 3.48 – Exemple de graphe placé avec une grille sous-jacente par rapport au même graphe sans la grille. Les nœuds sont centrés dans les mailles de la grille.

3.3.2.1 Briques de construction

La première étape consiste à construire une structure homogène pour la représentation des réactions enzymatiques. Cette structure doit permettre de reconnaître facilement les réactions enzymatiques et aussi d'être capable de s'adapter à toutes les formes qu'elles peuvent prendre. Nous allons nommer ces structures des *pavés* de réaction. Ces pavés vont contenir les arcs et les nœuds, qui seront reliés ensemble. Les pavés sont décrits par un motif comprenant plusieurs cases de la grille de placement. Ce motif peut changer selon la forme de la réaction enzymatique et les molécules avec lesquelles elle interagit. Nous avons représenté ces motif sur la figure 3.49. Le haut de la figure montre un pavé complet, c'est le motif le plus grand que nous puissions dessiner pour une réaction enzymatique. Il occupe cinq cases de large sur trois cases de haut. Il permet le placement du substrat principal, de deux substrats secondaires, du produit principal, de deux produits secondaires, ainsi que de deux modulateurs. Nous nous sommes volontairement limités à trois substrats (et produits), car des réactions en ayant plus sont rares. Ce motif peut être simplifié lorsque moins d'éléments sont présents et dans ce cas il prendra moins de place. Nous avons représenté sur la figure deux versions de motifs incomplets, le premier comprenant uniquement un substrat et trois produits, et le deuxième avec seulement un substrat et un produit. Nous avons dessiné ces motifs horizontalement de gauche à droite, mais ils peuvent aussi exister verticalement et dans les deux sens : droite à gauche, haut en bas et bas en haut. On n'a pas prévu d'autre possibilité de tracer une réaction enzymatique, chaque réaction sera dessinée soit verticalement, soit horizontalement.

3.3.2.2 Niveau des nœuds

Maintenant que nous savons comment nous allons dessiner une réaction enzymatique, nous allons nous intéresser au choix des substrats et produits principaux. Nous avons vu précédemment que les réactions enzymatiques discriminaient leurs substrats et leurs produits, l'un est principal et les autres

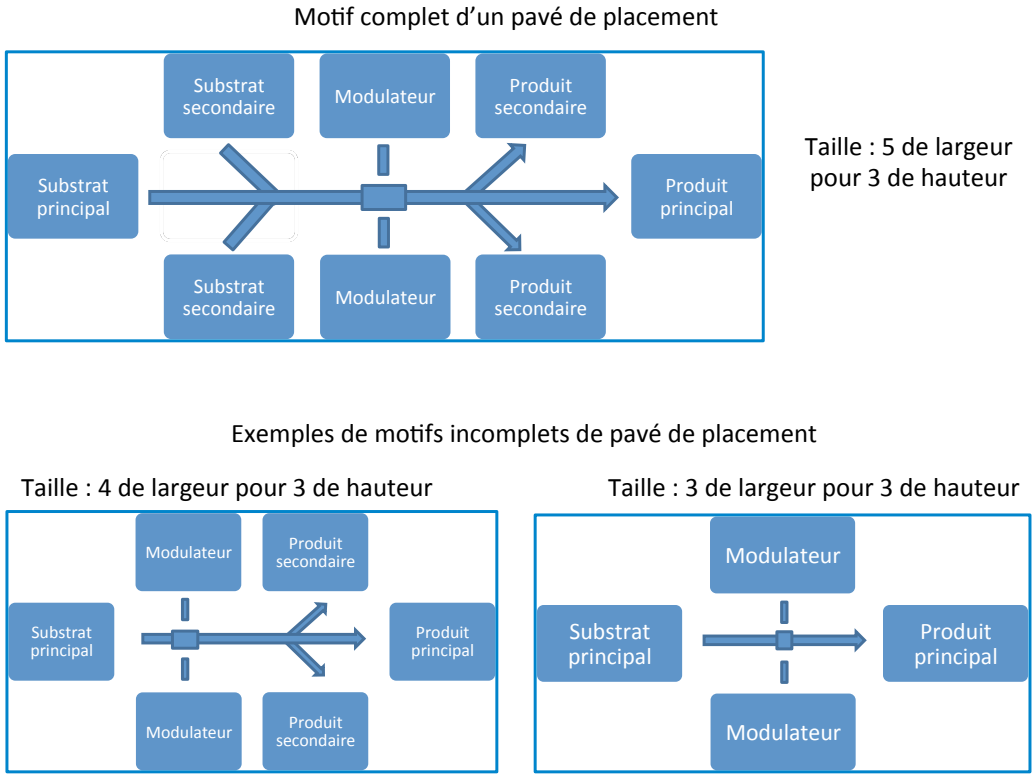


Figure 3.49 – Différents motifs du pavé de placement d'une réaction métabolique : le motif complet (en haut), deux motifs incomplets (en bas).

sont secondaires. Ce choix est dicté par des considérations biochimiques pour certaines réactions et par des raisons pratiques pour d'autres. Dans les deux cas, c'est une information à laquelle nous n'avons pas forcement accès (absente des modèles SBML par exemple) mais que nous allons devoir obtenir. C'est nécessaire pour savoir par quelles espèces moléculaires passent les voies du réseau : en général, une voie métabolique est une suite de réactions enzymatiques connectées par leur éléments principaux. Pour les circuits logiques enzymatiques, les voies seront les chemins qui vont connecter les entrées du circuit à sa sortie. Nous avons vu précédemment que la représentation claire de ce chemin était un critère important de notre méthode de placement. Connaître quels sont les éléments principaux d'une réaction permet de savoir comment la représenter et comment affecter correctement les espèces moléculaires dans le pavé de placement.

Nous avons mis au point une méthode pour déterminer les substrats et produits principaux dans une réaction. Nous allons calculer le niveau de chaque espèce (nombre d'occurrences), plus cette mesure sera élevée plus l'espèce moléculaire aura de probabilité d'être secondaire. Lorsque les niveaux de toutes les espèces seront calculés, alors pour chaque réaction l'espèce avec le plus bas niveau deviendra substrat ou produit principal. Pour calculer le niveau d'une espèce S, on regarde si elle est souvent associée à une autre espèce T dans toutes réactions auxquelles S participe. Plus ce couple d'espèces sera présent et plus leur niveau sera élevé. Cette méthode se base sur le fait que les éléments secondaires apparaissent souvent par paire lorsqu'ils sont utilisés dans des réactions. Nous allons donc essayer de détecter ces associations pour calculer le niveau. Voici l'algorithme que nous utilisons :

Pour calculer le niveau d'une espèce S, on va commencer par calculer la matrice de ses associations $MatriceNiveau(S, E_i)$. Cette matrice va répertorier combien de fois l'espèce S est utilisée dans une

réaction en conjonction de l'espèce E_i. On définit :

$$Niveau(S) = sup(MatriceNiveau(S, E_i)) \text{ pour toutes les espèces } i$$

Nous avons représenté un exemple de calcul du niveau sur un réseau métabolique sur la figure 3.50. La plupart des nœuds sont de niveau 1, ce qui est normal sachant qu'ils ne sont présents qu'une seule fois dans chaque association. Une paire de nœuds est présente deux fois, ses composants ont donc un niveau 2.

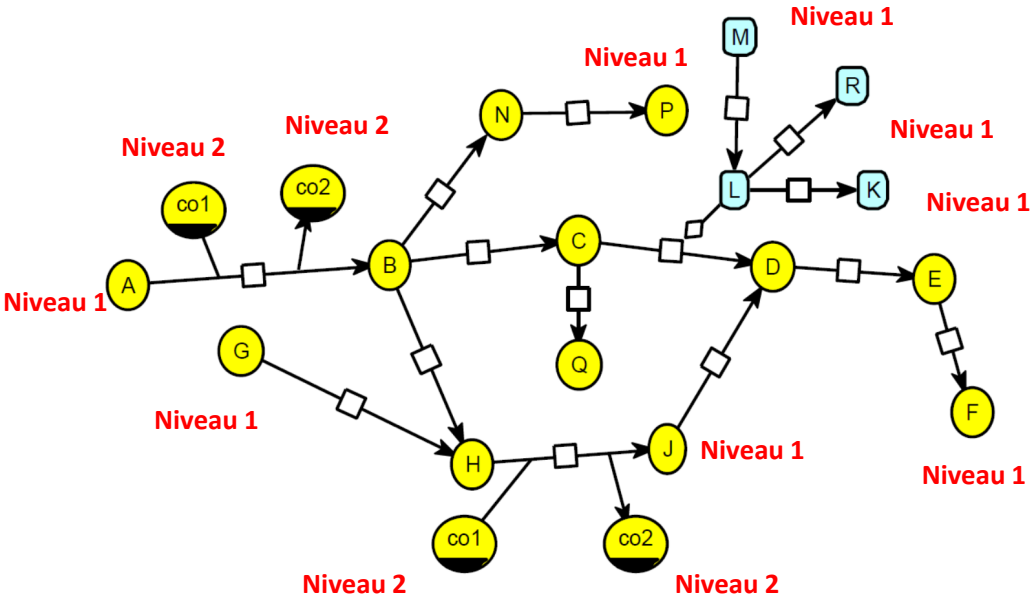


Figure 3.50 – Calcul du niveau des espèces moléculaires dans un réseau métabolique.

3.3.3 Approche par croissance arborescente

Nous voulons obtenir une méthode de placement qui montrera le sens du réseau métabolique. Bien que ce ne soit pas le cas de tous les réseaux métaboliques, les réseaux logiques enzymatiques ont par construction une forme presque arborescente. Nous avons vu précédemment que bien que les portes logiques enzymatiques puissent comporter des boucles, le schéma dans lequel elles sont organisées dans un circuit n'en comporte pas. Nous allons donc considérer le réseau métabolique à placer comme un arbre. Il va falloir trouver une racine pour cet arbre, elle sera le commencement de l'algorithme de placement par croissance arborescente (voir fig. 3.51).

Nous allons partir des entrées du réseau métabolique. Cet ensemble constituera toutes les racines possibles, ensuite nous allons lancer sur tous ces nœuds un algorithme de recherche arborescente en largeur, modifié pour nos besoins. Ces recherches vont donner, pour chaque racine potentielle, le nombre de nœuds parcourus ainsi que la profondeur maximale atteinte. Ces deux indications vont permettre d'évaluer la *qualité* de la racine. La meilleure racine (plus grands nombre de nœuds et profondeur) sera alors nommée racine d'ordre 1 du graphe. Tous les chemins partant de cette racine et allant vers les feuilles seront calculés. Ensuite, nous allons retirer du graphe tous les nœuds principaux compris dans les chemins. Il va rester un série de sous-graphes où certains nœuds sont marqués, ces nœuds sont non-principaux, ils sont devenus des racines d'ordre 2. Nous allons ensuite relancer l'algorithme sur ces nœuds racines d'ordre 2. Au fur et à mesure, tous les nœuds du graphe seront

explorés, et nous allons obtenir une liste de chemins qui seront ensuite tracés.

La dernière étape va consister à convertir la position des nœuds dans la grille, en position sur le dessin. Pour cela, nous allons calculer la taille d'une case de la grille en prenant comme étalon la taille de la représentation du nœud du réseau qui occupe la plus grande place.

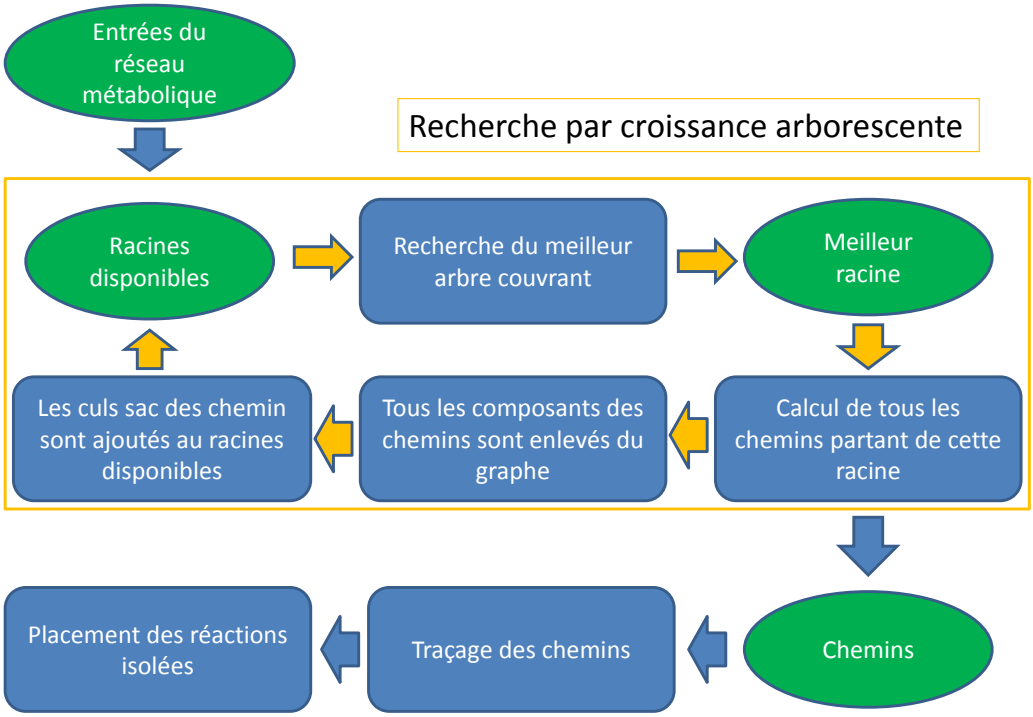


Figure 3.51 – Algorithme par croissance arborescente.

Nous allons maintenant voir en détail toutes les parties de cet algorithme.

3.3.3.1 Rechercher la racine principale

Pour trouver la racine principale, nous allons évaluer et comparer toutes les racines potentielles : les entrées du réseau métabolique. Un parcours en largeur standard part de la racine, puis trouve tous les nœuds à distance de 1, puis ceux à distance de 2 et ainsi de suite explorer tout le graphe. Nous allons le modifier en plaçant des conditions sur sa capacité à explorer un nœud à partir d'un voisin. Selon la nature du nœud d'où il part, seuls certains nœuds fils seront explorables. De plus, il faut, pour pouvoir être explorés, que les nœuds ne dépassent pas un certain *niveau* (au sens défini précédemment). Voici le détail de ces conditions :

— Si le nœud père est un produit, alors les nœuds substrats visibles par l'intermédiaire d'une réaction peuvent être explorés.
— Si le nœud père est un substrat, alors les nœuds produits visibles par l'intermédiaire d'une réaction peuvent être explorés.
— Si le niveau du nœud fils est supérieur à une certaine limite, il ne peut pas être exploré.

La première information que cette recherche apporte est le nombre de nœuds marqués : ce sont tous les nœuds *visibles* depuis l'exploration par recherche en largeur modifié. Pour être visible, un nœud doit appartenir à une réaction dont à la fois le substrat et le produit principal ont été explorés. La deuxième information est la profondeur maximale atteinte dans le graphe : la plus grande distance

entre la racine et un nœud exploré. A partir de ces informations, nous allons pouvoir juger de la qualité de la racine. La meilleure racine sera celle qui aura le plus de nœuds marqués, et, s'il y a égalité, celle qui aura la plus grande profondeur. Ces critères vont nous assurer que la racine choisie donnera le meilleur arbre possible, celui qui permettra de tracer le maximum de chemins et de placer le maximum de nœuds.

Nous allons prendre comme exemple le graphe de la figure 3.50. Nous avons trois racines potentielles, A, G et M, qui sont les entrées du réseau. Nous allons lancer l'algorithme sur chacun de ces nœuds. Sur la figure 3.52 est représenté le résultat de l'algorithme lancé sur le nœud G. Nous pouvons voir que le nombre de nœuds marqués est de 13 et la profondeur de 5. On constate que certains nœuds n'ont pas été parcourus (M,R,K).

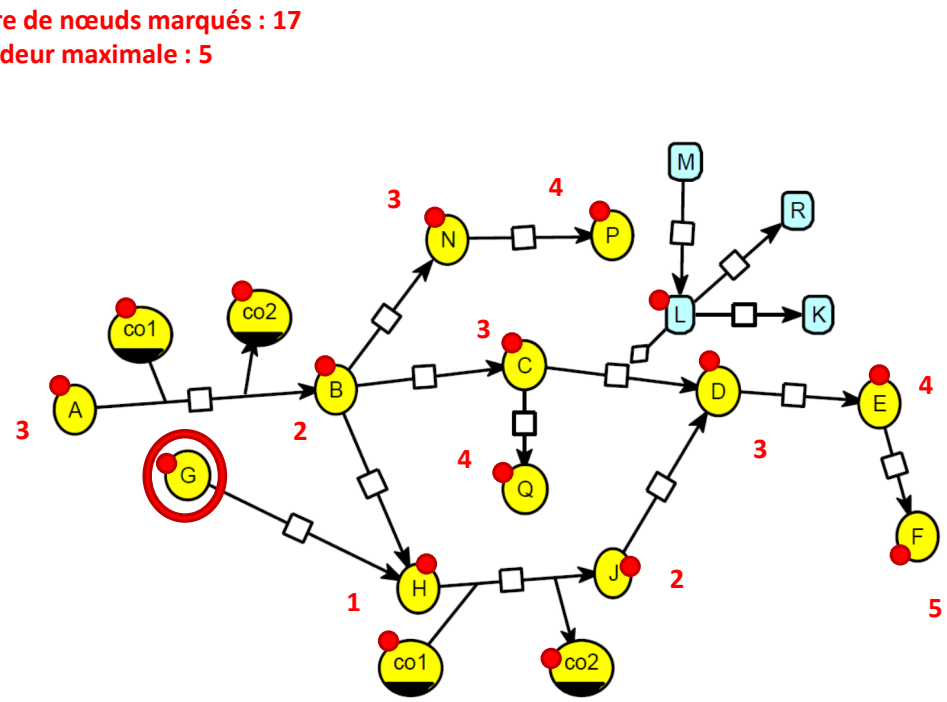


Figure 3.52 – Parcours en largeur adapté lancé sur le nœud G. Les points rouges représentent les nœuds marqués, les numéros la profondeur des nœuds explorés.

Lorsque toutes les racines potentielles ont été examinées, on les classe pour garder la meilleure :

- Racine A : 17 nœuds marqués, profondeur maximale 6.
- Racine G : 17 nœuds marqués, profondeur maximale 5.
- Racine M : 4 nœuds marqués, profondeur maximale 2.

La racine d'ordre 1 est donc A. Nous allons maintenant calculer les chemins qui partent de cette racine.

3.3.3.2 Trouver les chemins

Pour trouver les chemins, nous allons commencer par refaire un parcours en largeur à partir de la racine d'ordre 1. Ensuite, nous allons partir des feuilles qui ont été trouvées par ce parcours, quand elles sont des éléments principaux, pour tracer des chemins des feuilles vers la racine. Ces chemins ne seront constitués que d'élément principaux reliés par des réactions enzymatiques. Tous ces éléments vont ensuite être retirés du graphe des éléments possibles à explorer, ils seront considérés comme étant

déjà placés. Certains éléments vont être marqués, mais pas explorés : ce sont les éléments secondaires des réactions explorées. Comme il est possible que tous les nœuds principaux ne soient pas accessibles à partir de la racine d'ordre 1, tous les éléments secondaires marqués vont devenir des nouvelles racines, étiquetées d'ordre 2. Des parcours en largeur adaptés seront lancés à partir d'elles, et de nouveaux chemins seront trouvés. Ce processus peut se prolonger avec des racines d'ordre de plus en plus élevé, jusqu'à ce que tous les nœuds du graphe aient été marqués.

Les chemins étant issus d'un arbre, ils ne contiendront aucune boucle, ce qui fait que toutes les boucles néanmoins présentes dans le réseau métabolique seront représentées coupées (deux occurrences différentes du même métabolite).

Nous avons représenté le déroulement de cet algorithme sur notre réseau métabolique exemple de la figure 3.53. Nous pouvons voir qu'il y a deux racines : une d'ordre 1, A, et une d'ordre 2, L. Il y a 5 chemins qui partent de A et 3 qui partent de L :

— Racine A : A-B-N-P
— Racine A : A-B-C-D-E-F
— Racine A : A-B-C-Q
— Racine A : A-B-H-J
— Racine A : A-B-H-G
— Racine L : L-M
— Racine L : L-R
— Racine L : L-K

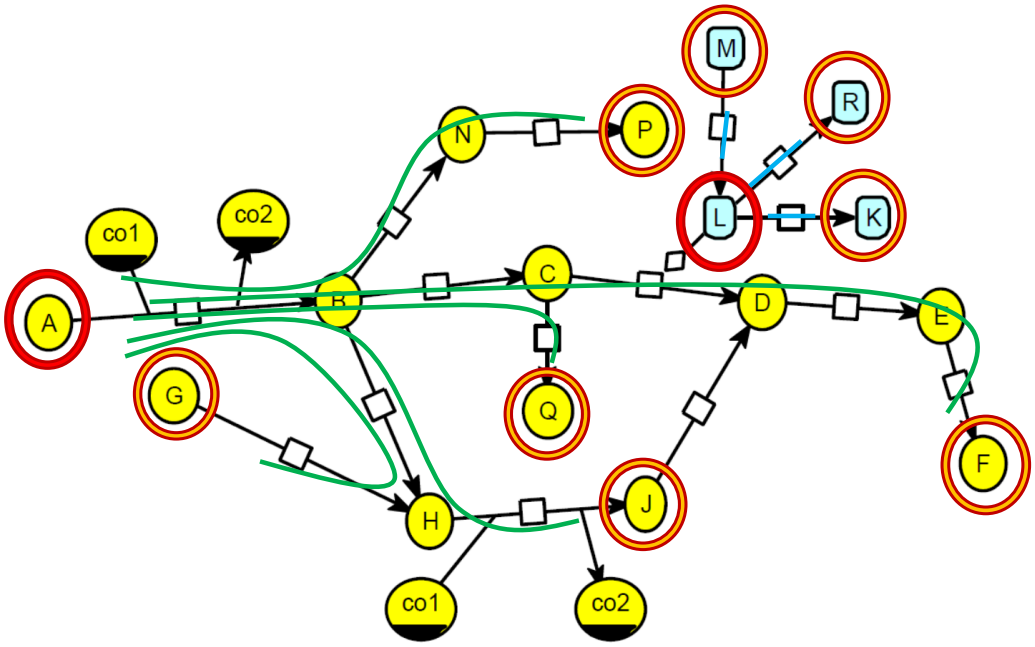


Figure 3.53 – Recherche des chemins dans un réseau métabolique. Les racines (**A** et **L**) sont entourées en rouge, les feuilles en jaune et rouge. Les chemins venant de la racine d'ordre 1, **A**, sont dont dessinés en vert, ceux de la racine d'ordre 2, **L**, en bleu.

Ces chemins vont ensuite être ordonnés selon deux critères : l'ordre de la racine dont ils sont issus, et leur longueur. Ce classement va déterminer l'ordre dans lequel ils vont être placés sur la grille. Voici le classement obtenu pour notre réseau exemple :

1. A-B-C-D-E-F , ordre 1, taille 6
2. A-B-H-G, ordre 1, taille 4
3. A-B-H-J, ordre 1, taille 4
4. A-B-N-P, ordre 1, taille 4
5. A-B-C-Q, ordre 1, taille 4
6. L-M, ordre 2, taille 2
7. L-R, ordre 2, taille 2
8. L-K, ordre 2, taille 2

3.3.3.3 Tracer les chemins

Le placement des chemins se fait dans l'ordre du classement précédemment établi. Chaque chemin est constitué de plusieurs réactions enzymatiques, chacune représentée par un pavé de placement. Ces pavés peuvent avoir quatre orientations, vers le haut, le bas, la gauche ou la droite. Ces directions sont ordonnées : à chaque fois qu'une réaction doit être placée, on se pose la question de la direction et de l'emplacement du pavé. L'emplacement sera déterminé en fonction de la nature des éléments principaux de la réaction : si l'un de ces éléments est déjà placé, alors on greffe la réaction sur cet élément, sinon elle sera placée à distance dans une nouvelle position, séparée du reste du graphe. Ensuite, chaque direction sera testée dans l'ordre pour placer le pavé. Chaque élément étant placé sur une case de la grille de placement, il est très facile de détecter les éventuelles collisions par comparaison selon les cases de la grille de placement.

On va commencer par placer le premier chemin. Celui-ci ne pouvant pas être gêné par d'autres, il va pouvoir être rectiligne. Ensuite, pour le deuxième chemin, on va regarder s'il commence par des réactions qui ont déjà été placées. Si c'est le cas, on parcourt le chemin sans rien faire tant que les réactions sont déjà placées. Lorsqu'on arrive à une réaction qui n'a pas encore été placée, on commence à placer le chemin.

Sur les figures 3.54 et 3.55 nous avons représenté le placement des quatre premiers chemins. Le placement du premier chemin ne pose aucun problème. Dans le deuxième chemin, A et B sont déjà placés, donc le placement du chemin commence à partir de H. La place étant occupée dans les deux premières directions (droite et gauche), on prend la troisième (bas) pour placer le nœud H. Le placement des troisième et quatrième chemins suit le même principe.

Au premier abord, le placement du chemin 5, A - B - C - Q (voir fig. 3.56) n'était pas possible car la place était occupée dans les quatre directions. Pour résoudre ce problème, on va effectuer un autre test pour choisir l'une des quatre directions possibles. Au lieu de regarder si l'emplacement est disponible, on va chercher à savoir si l'emplacement peut être rendu disponible en faisant un *décalage*. Un décalage est une modification du placement du graphe où l'on va tracer un trait imaginaire vertical ou horizontal, qui se situera sur une des lignes de la grille de placement (il ne peut donc pas couper de nœuds). Ensuite, l'une des deux parties du graphe situées de part et d'autre du trait va être décalée pour faire une place au milieu. Bien sûr, les réactions coupant le trait seront allongées d'autant.

Si grâce à un décalage, le placement est possible dans une direction, on effectue ce décalage et on place la réaction. Ce n'est pas toujours le cas : par exemple, aucun décalage horizontal n'aurait pu rendre les directions gauche ou droite envisageables pour placer la réaction $C \longrightarrow Q$.

Les trois autres chemins (6, 7 et 8) ne posant aucun problème pour être placés, nous ne les détaillerons pas.

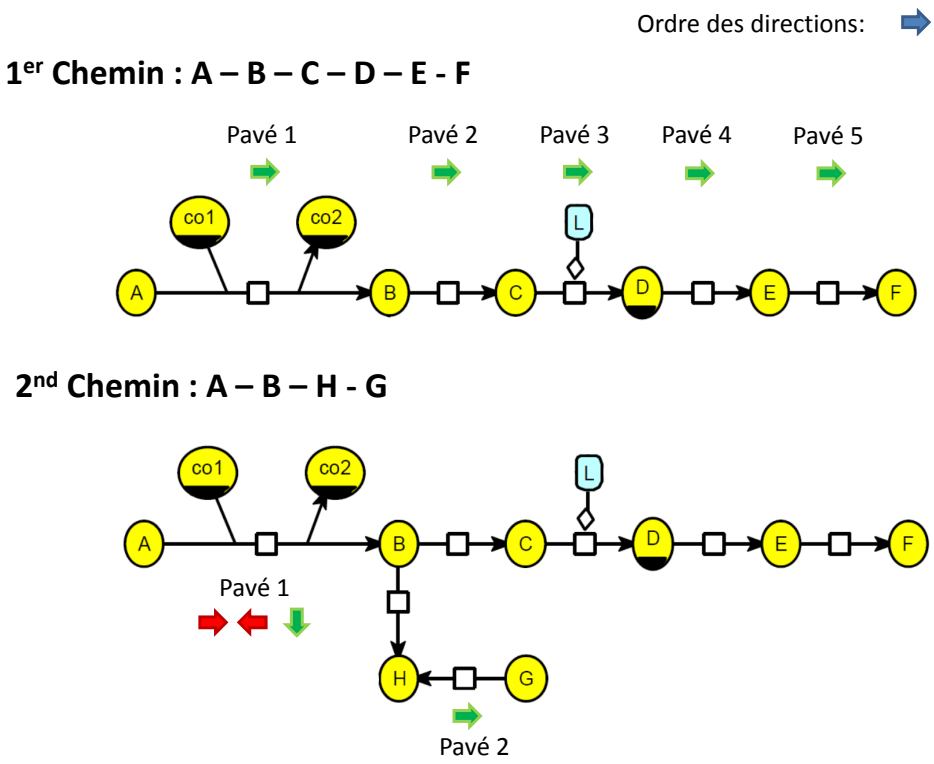


Figure 3.54 – Placement des deux premiers chemins. Pour chaque réaction, on a représenté les directions tentées et échouées en rouge et celle réussie en vert.

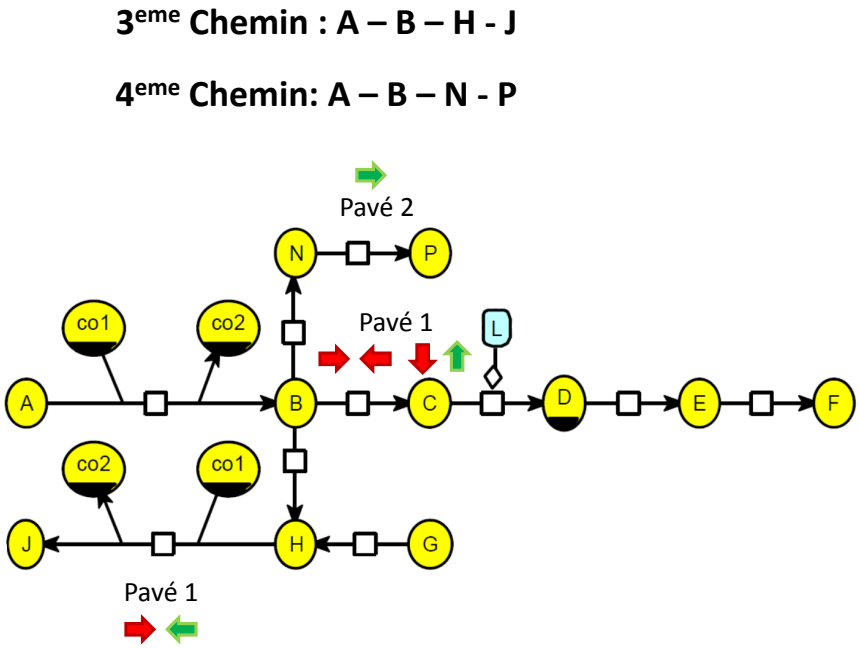


Figure 3.55 – Placement des chemins 3 et 4.

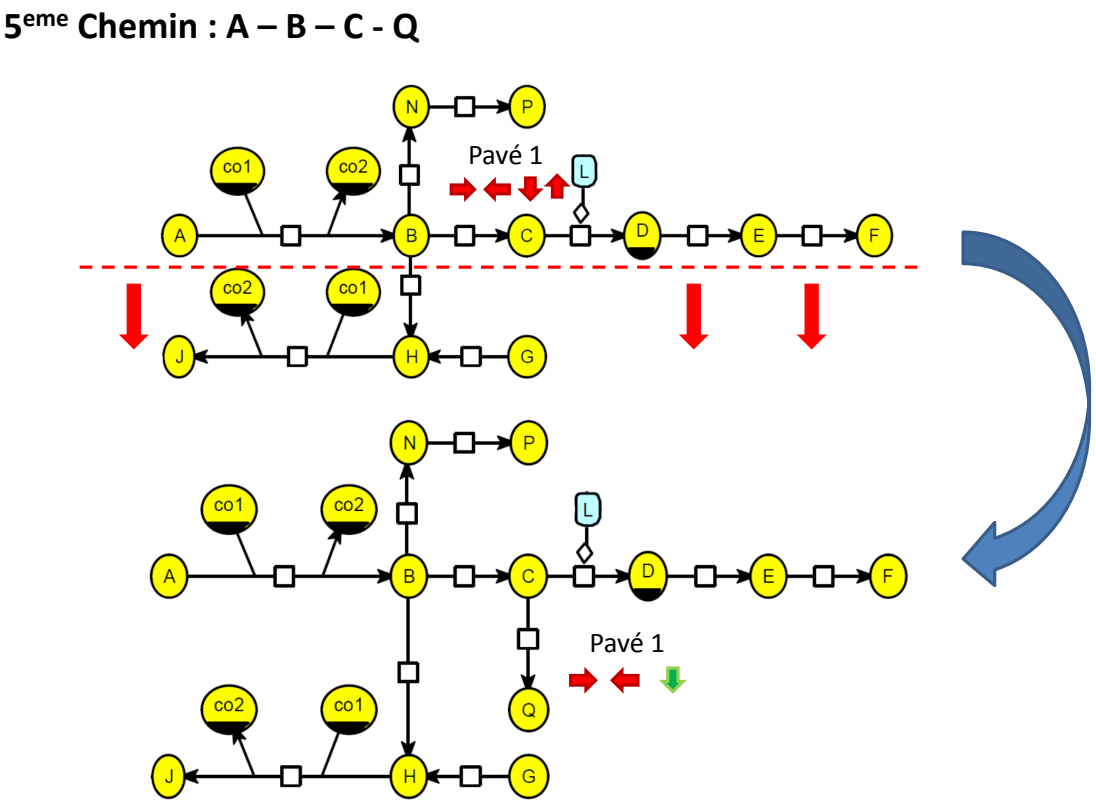


Figure 3.56 – Placement du dernier chemin. Il a été nécessaire de décaler vers le bas la deuxième ligne de réactions pour pouvoir placer ce chemin.

3.3.3.4 Placer les réactions isolées.

La dernière étape sera de placer les réactions isolées. Ces réactions n'appartiennent à aucun chemin et sont dues en général à la coupure d'une boucle. Dans notre exemple, on avait une boucle avec les nœuds B-C-D-J-H-B, cette boucle a été coupée au niveau de la réaction entre J et D. Le problème est que cette réaction n'est présente dans aucun des chemins : il va être donc nécessaire de l'ajouter explicitement. En plaçant ces réactions, il est très probable que certains nœuds soient dupliqués. La dernière étape de notre algorithme consiste donc à vérifier que toutes les réactions ont bien été affichées et sinon de placer les réactions manquantes.

3.3.4 Discussion

Cet algorithme est assez rapide, la complexité du parcours en largeur modifié est la même que celle du parcours en largeur standard : $O(S + A)$, S étant le nombre de sommets et A, le nombre d'arcs. Nous allons effectuer autant de parcours que nous avons d'entrées, et pour chaque nouvel ordre de racine, effectuer une nouvelle série de parcours. Mais il faut se souvenir que les ordres n'ont pas de nœuds parcourus en commun, donc pour un réseau d'une certaine taille, s'il y a beaucoup de parcours, chacun sera petit. La complexité cumulée des parcours sera donc bornée par $N_e * O(S + A)$, avec N_e, le nombre d'entrée du sous-réseau. Les autres opérations ne vont pas changer la complexité de l'algorithme. Trouver les chemins prendra un temps proportionnel à la taille de chaque chemin, et placer les chemins revient en fait à placer une et une seule fois chaque réaction. Les opérations de décalages peuvent augmenter un peu la durée, mais cela restera négligeable comparé aux parcours.

L'algorithme est linéaire en sommets et en arcs, ce qui permet de l'utiliser sur de grands réseaux sans difficulté.

Cette méthode présente quelques limitations : les boucles sont systématiquement coupées et le nœud de jointure dupliqué. Étant donné qu'il n'y a que quatre directions possibles pour le placement d'une réaction enzymatique sur un élément principal déjà placé, s'il y a dans le réseau des éléments principaux de degré supérieur à quatre, alors les nœuds en question seront eux aussi dupliqués et les réactions supplémentaires seront placées à partir des clones.

Néanmoins, cet algorithme répond aux besoins qu'on s'était fixés en matière de placement. Il permet de disposer rapidement et esthétiquement des réseaux métaboliques et les circuits logiques enzymatiques produits par NetBuild. Nous avons réussi à respecter la plupart des contrainte de départ : il n'y a pas de croisement, les réactions sont clairement visibles et respectent les conventions de placement standard, et enfin, la direction générale du réseau est apparente.

NetPlace a été programmé dans le langage objet C++ et compilé pour les principales plateformes (Linux, Windows et MacOSX). Le programme source représente environ 3600 lignes de code.

Chapitre 4

Conclusions et perspectives

L'utilisation d'éléments biologiques pour implémenter des systèmes logiques est une voie novatrice, avec des applications potentielles particulièrement intéressantes. Nous avons vu que plusieurs moyens était possibles, génétique et enzymatique. Les éléments logiques génétiques ont été recherchés par de nombreux auteurs et constituent un champ de recherche particulièrement prolifique. Ces dernières années, les efforts sur le sujet se sont tournés de l'exploration et de la découverte vers une vision plus applicative, signe d'un début de maturité du domaine. En revanche, l'implémentation d'éléments logiques utilisant des réactions enzymatiques reste relativement vierge. Quelques travaux ont été publiés, mais ce champ de recherche reste majoritairement inexploré. Il n'y a pas de théorie générale, mais plutôt des suites de cas particuliers étudiés indépendamment. Nous avons voulu remédier à cette situation et construire des bases solides pour les futures recherches sur les éléments logiques enzymatiques.

Au cours de cette thèse, une contribution significative a été apportée aux implémentations de systèmes logiques utilisant des réseaux métaboliques. Un cadre théorique a été mis en place pour la création et l'utilisation de portes logiques implémentées avec des réactions enzymatiques. J'ai utilisé un concept connu, le codage d'un flux d'information par la présence / absence d'une espèce moléculaire et j'ai l'ai développé en une théorie complète pouvant être réutilisée pour d'autres applications. Cette théorie permet de concevoir et d'utiliser des portes logiques enzymatiques, et aussi de les assembler pour former des circuits logiques enzymatiques répondant à des fonctions booléennes complexes. La plupart des résultats obtenus ont pu être validés par simulation et beaucoup l'ont été expérimentalement. Cela montre la robustesse de l'approche.

J'ai ensuite conçu et développé deux logiciels utilisant la théorie des éléments logiques enzymatiques. Ces programmes, utilisés en tandem, permettent de créer automatiquement des réseaux métaboliques artificiels qui vont implémenter une fonction booléenne donnée.

Le premier, *NetGate*, va pouvoir créer de façon automatique des bibliothèques contenant des centaines de portes logiques enzymatiques. C'est un outil nouveau et efficace, le premier de son genre. Auparavant, il était nécessaire d'analyser manuellement des réseaux métaboliques pour pouvoir découvrir chaque exemplaire d'un type de porte donné. *NetGate* permet d'augmenter fortement les applications potentielles utilisant des portes logiques enzymatiques, car on peut les obtenir facilement, en grande quantité et avec une très forte probabilité d'avoir un fonctionnement correct et robuste.

Le deuxième programme, *NetBuild*, va utiliser une bibliothèque de portes (par exemple celle créée par NetGate) pour construire des circuits qui seront capables de répondre à des questions logiques complexes. Il a un but applicatif direct, par exemple, si pour un test diagnostic, on a besoin d'un réseau métabolique qui calcule une fonction booléenne précise, à partir de métabolites d'entrée donnés et fournissant une réponse colorimétrique, *NetBuild* va pouvoir fournir, à partir d'une bibliothèque

de portes, tous les réseaux métaboliques artificiels qui en sont capables. Là encore, aucun précédent n'existait.

Nous avons conçus ces programmes à partir de spécifications issues des résultats de travaux précédents. Nous les avons testés dans de nombreux cas pour nous assurer de leur correction et de leur robustesse. De ce point de vue, nous avons atteint les objectifs que nous nous étions fixés.

Comme le domaine est varié et relativement neuf, lors de mes trois années de travail sur les circuits logiques enzymatiques, j'ai du faire des choix, et parfois arrêter de chercher dans certaines directions pour me concentrer sur mon objectif : un pipeline NetGate-NetBuild fonctionnel. Nous avons par exemple songé à concevoir et utiliser d'autres éléments que les portes logiques dans nos circuits, comme par exemple des oscillateurs, des mémoires ou encore des machines à état. Nous avons aussi réfléchi à des extensions qui pourraient être faites à NetGate et Netbuild. Nous allons en détailler ici quelques unes.

4.1 Utilisation répétée d'un réseau logique enzymatique

Nos circuits logiques enzymatiques sont conçus pour une utilisation unique. Une fois que les entrées ont été mises en présence du réseau, on attend que l'information se propage, que les réactions enzymatiques se fassent, pour obtenir le résultat. Le réseau est ensuite inutilisable car certains métabolites nécessaires ont été complètement consommés et d'autres ont été produits. Bien entendu, cette caractéristique n'est pas un problème pour l'utilisation pour laquelle ils ont été conçus, mais c'est un frein à la diversification des utilisations possibles. Nous allons donc voir quelles seraient les modifications à apporter si on voulait adapter nos circuits pour qu'ils soient capables d'effectuer plusieurs itérations de calcul.

Les flux d'information sont des espèces moléculaires qui sont stables dans le temps si elles ne sont pas utilisées par une réaction du réseau. Donc, un signal représentant la valeur booléenne *vrai* la représentera longtemps, s'il n'est pas utilisé. Par exemple si nous avons une porte ET consommant du A et du B, et que pendant la première itération de calcul par le circuit, seul A est présent, la porte n'est pas activée et donc A n'est pas consommé. Si nous faisions une deuxième itération, A serait donc toujours présent et l'information qu'il représente aurait encore la valeur *vrai*, et cela quelle que soit la valeur qu'elle aurait dû avoir au début de cette itération pour assurer un bon fonctionnement du circuit logique enzymatique.

C'est le premier problème : après une itération de calcul, certaines espèces moléculaires intermédiaires peuvent être encore présentes dans le circuit, ce qui risque de fausser les calculs qui seront effectués par la suite. Il est donc nécessaire de pouvoir *purger* le système.

Le deuxième problème est lié aux réactions d'inhibition. Quand elles fonctionnent par désactivation des enzymes, cette désactivation ne s'arrête pas forcément à la fin d'une itération de calcul. Il va donc être nécessaire de réactiver ces enzymes.

Le dernier problème est plus facile à résoudre : les *extra-entrées* qui doivent être à la valeur *vrai* tout le long du calcul. Ces entrées peuvent être consommées lors d'une itération et pour certaines, ne plus représenter *vrai* au début de l'itération suivante. Il va donc être nécessaire de ravitailler le système.

En résumé, pour permettre au circuit de pouvoir être utilisé plusieurs fois, une série d'actions doivent être faisables :

1. purger le système pour se débarrasser des réactifs résiduels de l'itération précédente,
2. réinitialiser le système pour que les réactions inhibitrices soient de nouveau faisables,

3. ravitailler le système pour que les métabolites nécessaires soient encore présents en quantité suffisante.

Ravitailler le système est relativement facile à faire tant que le circuit est accessible et que le mécanisme utilisé ne perturbe pas le milieu ambiant. Réinitialiser les réactions d'inhibition peut être fait en déclenchant d'autres réactions qui consomment l'inhibiteur, pour qu'après un certain temps les enzymes redeviennent fonctionnelles. On peut voir que cela revient en fait à purger le système des inhibiteurs restants et donc cela pourrait être résolu de la même manière que pour le premier problème. On peut aussi essayer d'éviter d'utiliser des réactions d'inhibition quand c'est possible.

Le principal défi va être de purger sélectivement le système. Pour cela, on peut construire un circuit parallèle à celui calculant la fonction logique. Ce circuit ne devra fonctionner que lorsque l'itération courante est terminée, un signal servant de déclencheur. Le circuit de purge prend en entrée toutes les espèces moléculaires à purger ainsi que le déclencheur. On peut utiliser des porte ET, chacune prenant en entrée une des espèces à purger ainsi que le déclencheur (voir fig. 4.1). Ces portes ne doivent en outre ne produire que des espèces moléculaires qui n'interagissent pas avec le reste des circuits. Il est aussi envisageable d'utiliser plusieurs déclencheurs pour faciliter la conception du circuit de purge.

À la fin de l'itération courante, ce circuit principal doit produire les métabolites déclenchant le circuit de purge. Ces métabolites seront consommés par le circuit de purge, mais il faut s'assurer qu'il n'en reste plus avant de faire l'itération suivante. On peut prévoir d'autres réactions qui, après un certain temps, consommeraient les molécules de déclencheur résiduelles. Cela nécessite alors un circuit *retardateur* pour déclencher cette dégradation.

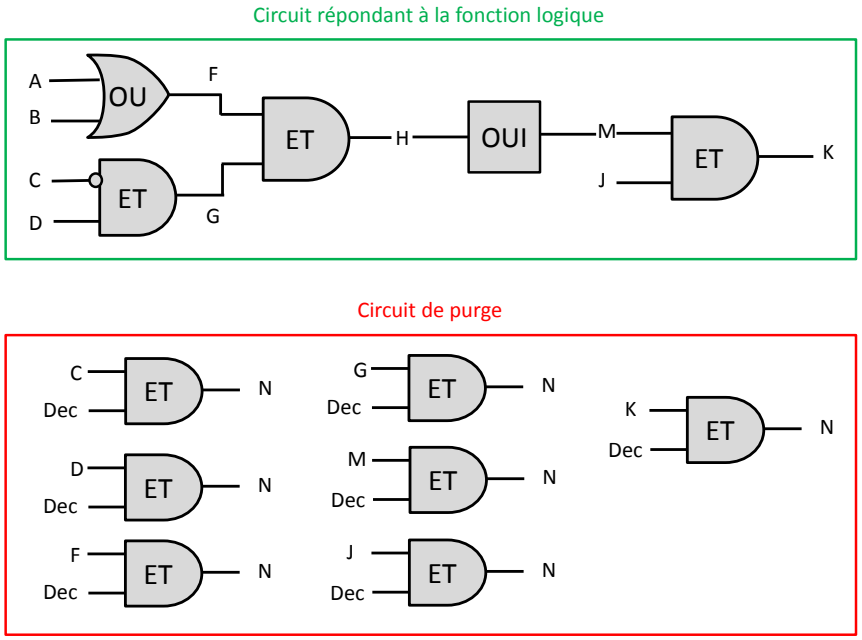


Figure 4.1 – Exemple de circuit de purge. Les entrées **A** et **B** sont nécessairement transformées en **F** et n'ont donc pas besoin de purge. De même pour **H**, qui est transformé en **M**. Toutes les autres espèces moléculaires doivent être purgées.

Même s'il est envisageable de trouver, pour un circuit enzymatique donné, un ensemble de réactions qui permette de le réinitialiser, il reste à pouvoir automatiser la conception du circuit de purge. Mais

les principales questions qui se posent quand on veut réutiliser nos circuits itérativement sont i) *d'où vient l'énergie* qui permet le calcul ? ii) combien d'itérations pouvons-nous faire ?

L'énergie qui permet de calculer est l'énergie potentielle contenue dans les métabolites présents initialement : les entrées et les *extra-entrées* à *vrai*. De même, l'énergie qui permet de purger le circuit de calcul provient de la présence des métabolites qui permettent le déclenchement du circuit de purge. Ces métabolites ne sont pas en quantité infinie et après un certain temps, toutes les réactions arriveront à l'équilibre. Dans le meilleur des cas, en supposant un système fermé, le calcul ne pourrait être itéré qu'un petit nombre de fois.

On peut peut bien sûr apporter de l'énergie depuis l'extérieur du système, par exemple en faisant diffuser des métabolites dans la vésicule qui contient les réseaux métaboliques. Il faudrait aussi prévoir l'extraction des métabolites *inutiles* hors de la vésicule.

On constate donc que la réutilisation itérative des circuits logiques enzymatiques constitue un problème non trivial qui nécessite une étude approfondie.

4.2 Portes multi-fonctions

Nous avons vu dans la partie 2 qu'une même porte logique enzymatique pouvait implémenter plusieurs fonctions booléennes, et que ce sont les valeurs des seuils qui vont déterminer quelle est cette fonction. D'un certain point de vue, cela constitue un *défaut* que l'on contourne en faisant des simulations sur le circuit final. C'est un aspect que nous avons ignoré dans *NetBuild*, où nous considérons que les portes logiques n'ont qu'une seule fonction.

Un solution simple serait de construire le diagramme fonctionnel de chaque porte logique enzymatique trouvée par *NetGate*. Bien qu'il puisse être important, le nombre de diagrammes est suffisamment petit pour que le processus soit réalisé dans un temps raisonnable. Ces diagrammes permettraient de trouver les bornes de validité de la fonction booléenne associée à chaque porte par Netgate. Nous aurions toujours une seule fonction par porte, mais nous saurions exactement dans quelles conditions cette porte est fonctionnelle. Ensuite, nous pourrions, pour chaque implémentation proposée par NetBuild, calculer les quantités de métabolites nécessaires sur les entrées pour que le circuit fonctionne correctement. Pour effectuer ce calcul, on pourrait partir des contraintes de la porte calculant la sortie, et remonter le réseau jusqu'aux entrées du circuit en passant par chaque porte. Ce calcul pourrait probablement être effectué par un solveur linéaire puisque chaque domaine de validité peut être exprimé sous forme de contrainte linéaire. On obtiendrait ainsi des circuits logiques enzymatiques plus robustes.

Nous n'avons pour l'instant que très peu exploité la propriété de multi-fonctionnalité des portes logiques enzymatiques. Nous nous contentons plutôt de les analyser pour éviter ces cas de multiples fonctions. Néanmoins, il est probable que cette capacité de changement de fonction pourrait permettre de faire des circuits ayant des fonctionnalités beaucoup plus avancées. En effet, on pourrait imaginer qu'une porte puisse changer de fonction selon la façon dont elle est alimentée et que ce changement fasse partie intégrante du fonctionnement du circuit logique global. Il serait tout à fait possible de construire une extension de NetBuild qui prendrait en compte les différentes fonctions booléennes associées à une porte. Cette extension pourrait prendre plusieurs formes selon le niveau de complexité souhaité.

4.3 Mémoires

Nous avons aussi travaillé sur la capacité de mémorisation des circuits logiques moléculaires. Dans les circuits électroniques, on utilise des montages spécifiques, comme les bascules RS, pour mémoriser un état logique. Dans les réseaux métaboliques, l'aspect flux de matière constitue un avantage. Les circuits moléculaires ont une capacité naturelle à stocker l'information. Tant que les molécules associées à un fil ne sont pas consommées, l'information binaire va rester disponible. Le problème est que la lecture supprime l'information. Une solution simple consiste à utiliser un amplificateur qui dédouble l'information avant la lecture.

Bien entendu, la mémorisation avec plusieurs cycles de lecture / écriture n'est principalement utile qu'en cas de calcul itératif, et donc cette capacité est largement associée au problème de l'énergie nécessaire au calcul itératif vu précédemment.

Bibliographie

[1] Clément Fromentin. L'art du diagnostic : sémiologie et clinique à l'âge de la psychiatrie classique. *L'Évolution Psychiatrique*, 81(2) :460–475, 2016.

[2] S.R.F. Meneses, A.P. Goode, A.E. Nelson, J. Lin, J.M. Jordan, K.D. Allen, K.L. Bennell, L.S. Lohmander, L. Fernandes, M.C. Hochberg, M. Underwood, P.G. Conaghan, S. Liu, T.E. McAlindon, Y.M. Golightly, and D.J. Hunter. Clinical algorithms to aid osteoarthritis guideline dissemination. *Osteoarthritis and Cartilage*, 2016.

[3] J.C. Toha, S. Vatsquez, P. Fuentes, and M.A. Soto. Algorithm for assisting medical diagnosis. *Computer Methods and Programs in Biomedicine*, 93 :303–309, 1993.

[4] Lopes, Maria Helena Baena de Moraes, Ortega Neli Regina Siqueira, Silveira Paulo Sérgio Panse, Massad Eduardo, Higa Rosângela, and Marin Heimar de Fátima. Fuzzy cognitive map in differential diagnosis of alterations in urinary elimination : A nursing approach. *International Journal of Medical Informatics*, 82(3) :201–208, 2016.

[5] Cristina Casado-Lumbrerasa, Alejandro Rodríguez-Gonzálezb, José María Álvarez Rodríguezc, and Ricardo Colomo-Palaciosb. PsyDis : towards a diagnosis support system for psychological disorders. *Expert Systems with Applications*, 39(13) :11391–11403, 2012.

[6] Enes Elvin Gul MD, Kjell C. Nikusb, Halil I. Erdoganc, and Kurtulus Ozdemirc. Differential diagnostic dilemma between pulmonary embolism and acute coronary syndrome. *Journal of Arrhythmia*, 32(2) :160–161, 2016.

[7] Marc Fakhoury. Autistic spectrum disorders : A review of clinical features, theories and diagnosis. *International Journal of Developmental Neuroscience*, 43 :70–77, 2015.

[8] Anwar R. Padhani, Guoying Liu, Dow Mu-Koh, Thomas L. Chenevert, Harriet C. Thoeny, Taro Takahara, Andrew Dzik-Jurasz, Brian D. Ross, Marc Van Cauteren, David Collins, Dima A. Hammoud, Gordon J.S. Rustin, Bachir Taouli, and Peter L. Choyke. Diffusion-weighted magnetic resonance imaging as a cancer biomarker : Consensus and recommendations. *Neoplasia*, 11(2) :102–125, 2009.

[9] Kim Henriksena, Sid E. O'Bryantb, Harald Hampelc, John Q. Trojanowskid, Thomas J. Montinee, Andreas Jerominf, Kaj Blennowg, Anders Lönneborgh, Tony Wyss-Corayi, Holly Soaresj, Chantal Bazenetk, Magnus Sjögrenh, William Hul, Simon Lovestonek, Morten A. Karsdala, Michael W. Weinerm, and Blood-Based Biomarker Interest Group. The future of blood-based biomarkers for Alzheimer's disease. *Alzheimer's and Dementia*, 10(1) :115–131, 2014.

[10] Jay L. Koyner MD, Steven G. Coca DO MS, Heather Thiessen-Philbrook, Uptal D. Patel MD, Michael G. Shlipak MD, Amit X. Garg MD, and Chirag R. Parikh MD. Urine biomarkers and perioperative acute kidney injury : The impact of preoperative estimated GFR. *American Journal of Kidney Diseases*, 66(6) :1006–1014, 2015.

[11] Aditya Bagrodiaa, Eugene K. Chaa, John P. Sfakianosg, Emily C. Zaborb, Bernard H. Bochnera, Hikmat A. Al-Ahmadiec, David B. Solitd, and Jonathan A. Colemana. Genomic biomarkers for the prediction of stage and prognosis of upper tract urothelial carcinoma. *The Journal of Urology*, 2016.

[12] Gustav Holmgrena, Jane Synnergrena, Christian X. Anderssonc, Anders Lindahlb, and Peter Sartipya. MicroRNAs as potential biomarkers for doxorubicin-induced cardiotoxicity. *Toxicology in Vitro*, 34 :26–34, 2016.

[13] Johan Palmfeldt, Kim Henningsenb, Stine Aistrup Eriksenb, Heidi K. Müllerb, and Ove Wiborgb. Protein biomarkers of susceptibility and resilience to stress in a rat model of depression. *Molecular and Cellular Neuroscience*, 2016.

[14] Simon J.S. Camerona, Keir E. Lewisb, Manfred Beckmanna, Gordon G. Allisona, Robin Ghosalb, Paul D. Lewisc, and Luis A.J. Mura. The metabolomic detection of lung cancer biomarkers in sputum. *Lung Cancer*, 94 :88–95, 2016.

[15] Ballman KV. Biomarker : Predictive or prognostic ? *J Clin Oncol*, 33(33) :3968–71, 2015.

[16] Eric Stern, Aleksandar Vacic, Nitin K. Rajan, Jason M. Criscione, Jason Park, Bojan R. Ilic, David J. Mooney, Mark A. Reed, and Tarek M. Fahmy. Label-free biomarker detection from whole blood. *Nature Nanotechnology*, 5 :138–142, 2009.

[17] Ackermann BL, Hale JE, and Duffin KL. The role of mass spectrometry in biomarker discovery and measurement. *Curr Drug Metab*, 7(5) :525–39, 2006.

[18] Yuji Imafuku, Gilbert S. Omenn, , and Samir Hanash. Proteomics approaches to identify tumor antigen directed autoantibodies as cancer biomarkers. *Dis Markers*, 20(3) :149–153, 2004.

[19] Leila H. Choe, Brenda G. Werner, , and Kelvin H. Lee. Two-dimensional protein electrophoresis : From molecular pathway discovery to biomarker discovery in neurological disorders. *NeuroRx*, 3(3) :327–335, 2012.

[20] Zangar RC, Daly DS, and White AM. ELISA microarray technology as a high-throughput system for cancer biomarker validation. *Expert Rev Proteomics*, 3(1) :37–44, 2006.

[21] Hye Jin Lee, Alastair W. Wark, and Robert M. Corn. Microarray methods for protein biomarker detection. *HHS Author Manuscripts*, 133(8) :975–983, 2008.

[22] Anand Gururajan, Gerard Clarke, Timothy G. Dinan, and John F. Cryan. Molecular biomarkers of depression. *Neuroscience and Biobehavioral Reviews*, 64 :101–133, 2016.

[23] E. V. Huntington. New sets of independent postulates for the algebra of logic, with special reference to Whitehead and Russell's Principia mathematica. *Trans. Amer. Math. Soc*, pages 274–304, 1933.

[24] R. E. Funk. *Understanding Buffered and Unbuffered CD4xxxB Series Device Characteristics.* Texas Instruments, Texas Instruments Post Office Box 655303 Dallas, Texas 75265, 2002.

[25] Robert W. Keyes. What makes a good computer device ? *Science*, 11(230) :138–144, 1985.

[26] N. Kinsey, C. DeVault, J. Kim, M. Ferrera, V. M. Shalaev, , and A. Boltasseva. Epsilon-near-zero Al-doped ZnO for ultrafast switching at telecom wavelengths. *Optica*, 2(7) :616–622, 2015.

[27] Junjie Baoa, Jun Xiaoa, Lin Fana, Xiaoxu Lib, Yunfei Haia, Tong Zhanga, and Chunbo Yanga. All-optical NOR and NAND gates based on photonic crystal ring resonator. *Optics Communications*, 329 :109–112, 2014.

[28] K. M. Johnson, M. A. Handschy, and L. A. Pagano-Stauffer. Optical computing and image processing with ferroelectric liquid crystals. *Optical Engineering*, 26(5) :385–391, 1987.

[29] David A. B. Miller. Are optical transistors the logical next step ? *Nature Photonics*, 4 :3–5, 2010.

[30] S. Leduc. La biologie de synthèse, étude de biophysique. 1912.

[31] Dellomonaco C. and al. The path to next generation biofuels : successes and challenges in the era of synthetic biology. *Microb. Cell Fact*, 9(3), 2010.

[32] Ro D.K. and al. Production of the antimalarial drug precursor artemisinic acid in engineered yeast. *Nature*, 440 :940–943, 2006.

[33] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266 :1021–1024, 1994.

[34] Benenson Y., Gil B., Ben-Dor U., Adar R., and Shapiro E. An autonomous molecular computer for logical control of gene expression. *Nature*, 429 :423–429, May 2004.

[35] Simon Ausländer, David Ausländer, Marius Müller, Markus Wieland, and Martin Fussenegger. Programmable single-cell mammalian biocomputers. *Nature*, 487 :123–127, 2012.

[36] Alec A. K. Nielsen, Bryan S. Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A. Strychalski, David Ross, Douglas Densmore, and Christopher A. Voigt. Genetic circuit design automation. *Science*, 352(6281) :53–66, 2016.

[37] Cases I. and de Lorenzo V. Genetically modified organisms for the environment : stories of success and failure and what we have learned from them. *Int. Microbiol*, 8 :213–222, 2005.

[38] Gonen Ashkenasy and M. Reza Ghadiri. Boolean logic functions of a synthetic peptide network. *J. Am. Chem. Soc.*, 126 (36) :11140–11141, 2004.

[39] Ronan Baron, Oleg Lioubashevski, Eugenii Katz, Tamara Niazov, , and Itamar Willner. Logic gates and elementary computing by enzymes. *The Journal of Physical Chemistry A*, 110(27) :8548–8553, 2006.

[40] T. Niazov, E. Katz R. Baron, O. Lioubashevski, I. Willner, and Proc. Natl. Acad. Concatenated logic gates using four coupled biocatalysts operating in series. *PNAS*, 103 :17160–17163, 2006.

[41] Reshma P Shetty, Drew Endy, and Thomas F KnightJr. Engineering biobrick vectors from biobrick parts. *Journal of Biological Engineering*, 2(5), 2008.

[42] Stéphanie Rialle, Liza Felicori, Camila Dias-Lopes, Sabine Pérès, Sanaâ El Atia, Alain R. Thierry, Patrick Amar, and Franck Molina. BioNetCAD : design, simulation and experimental validation of synthetic biochemical networks. *Int. Microbiol*, 26(18) :2298–2304, 2010.

[43] Dan Boneh, Christopher Dunwortha, Richard J. Liptona, and Jirí Sgall. On the computational power of DNA. *Discrete Applied Mathematics*, 71(1-3) :79–94, 1996.

[44] Tarkeshwar Gupta and Milko E. van der Boom. Redox-active monolayers as a versatile platform for integrating boolean logic gates. *Angewandte Chemie*, 120(29) :5402–5406, 2008.

[45] Guinevere Strack, Marcos Pita, Maryna Ornatska, and Evgeny Katz. Boolean logic gates that use enzymes as input signals. *ChemBioChem*, 9(8) :1260–1266, 2008.

[46] Keith J. Laidler and Branko F. Peterman. Temperature effects in enzyme kinetics. *Methods in Enzymology*, 63 :234–257, 1979.

[47] Victor Henri. Théorie générale de l'action de quelques diastases. *Comptes Rendus Hebdomadaires Séances Académie des Sciences*, 135 :916–919, 1902.

[48] Leonor Michaelis and Maud Menten. Die Kinetik der Invertinwirkung. *Biochemische Zeitschrift*, 49 :333–369, 1913.

[49] Briggs and Haldane J.B.S. A note on the kinetics of enzyme action. *Biochem. J.*, 19 :338–339, 1925.

[50] Ute Deichmann, Stefan Schuster, Jean-Pierre Mazat, and Athel Cornish-Bowden. Commemorating the 1913 Michaelis-Menten paper "Die Kinetik der Invertinwirkung" three perspectives. *The FEBS Journal*, 281 :435–463, 2014.

[51] Athel Cornish-Bowden, Jean-Pierre Mazat, and Serge Nicolas. Victor Henri : 111 years of his equation. *Biochimie*, 107 :161–166, 2014.

[52] Patrick Amar, Guillaume Legent, Michel Thellier, Camille Ripoll, Gilles Bernot, Thomas Nystrom, Milton Saier Jr, and Vic Norris. A stochastic automaton shows how enzyme assemblies may contribute to metabolic efficiency. *BMC Systems Biology*, 2(27), 2008.

[53] Patrick Amar and Loïc Paulevé. HSIM : an hybrid stochastic simulation system for systems biology. In *The Third International Workshop on Static Analysis and Systems Biology (SASB 2012)*, Deauville, France, September 2012.

[54] Michel Thellier, Guillaume Legent, Patrick Amar, Vic Norris, and Camille Ripoll. Steady-state kinetic behaviour of functioning-dependent structures. *The FEBS Journal (Federation of European Biochemical Societies)*, 273 :4287–4299, 2006.

[55] Marc Bouffard, Franck Molina, and Patrick Amar. Extracting logic gates from a metabolic network. In Patrick Amar, François Kepés, and Vic Norris, editors, *Proceedings of the Strasbourg Spring School in advances in Systems and Synthetic Biology*, pages 63–77, March 2015.

[56] VN Reddy and ML Mavrovouniotis ans MN Liebman. Petri net representations in metabolic pathways. *ISMB*, 1993.

[57] Nicolas Le Novere, Michael Hucka, Huaiyu Mi, Stuart Moodie, Falk Schreiber, Anatoly Sorokin, Emek Demir, Katja Wegner, Mirit I Aladjem, Sarala M Wimalaratne, Frank T Bergman, Ralph Gauges, Peter Ghazal, Hideya Kawaji, Lu Li, Yukiko Matsuoka, Alice Villeger, Sarah E Boyd, Laurence Calzone, Melanie Courtot, Ugur Dogrusoz, Tom C Freeman, Akira Funahashi, Samik Ghosh, Akiya Jouraku, Sohyoung Kim, Fedor Kolpakov, Augustin Luna, Sven Sahle, Esther Schmidt, Steven Watterson, Guanming Wu, Igor Goryanin, Douglas B Kell, Chris Sander, Herbert Sauro, Jacky L Snoep, Kurt Kohn, and Hiroaki Kitano. The Systems Biology Graphical Notation. *Nat. Biotech.*, 27(8) :735–741, 2009.

[58] Di Battista et al. Aesthetics. Section 2.1.2 :14–16, 1997.

[59] F. Schreiber. High quality visualization of biochemical pathways in BioPath. *In Silico Biology*, 2(2) :59–73, 2002.

[60] Christian Bachmaier, Ulrik Brandes, and Falk Schreiber. Biological Networks. *Handbook of Graph Drawing and Visualization*, 13(52) :621–651, 2014.