Intégration de l'inférence abductive et inductive pour la représentation des connaissances dans les réseaux de gènes

Tan Le

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Dévilré par l'Université Toulouse III – Paul Sabatier
Discipline ou spécialité : *Systèmes informatiques*

Présentée et soutenue par *LE Tan*
Le 28 Avril 2014

Titre : Intégration de l'inférence abductive et inductive pour la représentation des connaissances dans les réseaux de gènes

---

JURY

Gilles RICHARD	Président du jury	Professeur, Université Toulouse III
Jacques DEMONGEOT	Rapporteur	Professeur, Université Joseph Fourier, Grenoble
Yves LACROIX	Rapporteur	Professeur, Université de Toulon
Jean-Charles FAYE	Examinateur	Directeur de Recherche INSERM, Toulouse
Belaid BENHAMOU	Examinateur	Maître de conférences, Université de Provence, Marseille
Vincent RISCH	Examinateur	Maître de conférences, Université de Provence, Marseille
Andrei DONCESCU	Directeurs de thèse	Maître de conférences, Université Toulouse III
Pierre SIEGEL	Co-directeurs de thèse	Professeur, Université de Provence, Marseille

---

Ecole doctorale : EDSYS
Unité de recherche : LAAS-CNRS
Directeur(s) de Thèse : Andrei DONCESCU et Pierre SIEGEL
Rapporteurs : Jacques DEMONGEOT et Yves LACROIX

A ceux qui m'aiment, A ceux que j'aime,

A mes filles Hanh Quynh et Hanh Quyen, je vous aime !!!

Résumé

Le raisonnement diagnostique (abductif) et le raisonnement de prédiction (inductif) sont deux des méthodes de raisonnement qui permettent la découverte de connaissances nouvelles. Lorsque le raisonnement abductif est le processus permettant de trouver la meilleure explication (hypothèse) pour un ensemble d'observations (Josephson, 1994), le raisonnement de prédiction est le processus, à partir d'un ensemble d'observations, permettant de trouver tous les résultats possibles. Ces observations peuvent être les symptômes d'un patient, des expériences concernant les réseaux métaboliques et génomiques, etc.

Dans cette thèse, nous nous sommes intéressés à la représentation, l'analyse et la synthèse des réseaux de signalisation génomique en utilisant la logique des hypothèses. En fait, ce mémoire se focalise sur la modélisation des voies de signalisation en réponse à la cassure double-brin de l'ADN.

Pour implémenter l'abduction nous utilisons les algorithmes de production. Ensuite, la logique des défauts permet de construire des modèles de représentation minimale.

Ces algorithmes de découvertes de connaissances sont prouvés sur la carte de cassure double brin de l'ADN. Cette carte est minimale en tant que graphe de causalité biologique et elle permet d'intégrer les données biomoléculaires.

Mots clés : conclusion conséquente, abduction, champ de production, logique des défauts, raisonnement diagnostique, raisonnement de prédiction, cassure double-brin de l'ADN, voie de signalisation, voie métabolique, représentation, modélisation, connaissances biologiques.

Abstract

Diagnostic reasoning (abductive) and predictive reasoning (inductive) are two methods of reasoning that enable the discovery of new knowledge. When abductive reasoning is the process of finding the best explanation (hypothesis) for a set of observations (Josephson, 1994), the inductive reasoning is the process of predicting, from a set of observations, to find all possible results. These observations may be symptoms of a patient, experiments on genomic and metabolic networks, etc.

In this PhD thesis, we are interested in the representation, analysis and synthesis of genomic signaling networks using hypothetical logic. In fact, this thesis focuses on modeling of signaling pathways in response to the DNA double stranded break.

To implement the abduction, we use algorithms of production. Then, the default logic is used to build models of minimum representation.

These algorithms are proven knowledge discovery on the map of DNA double-strand break. This map is minimal as biological causality graph and allows integrating bio-molecular data.

Keywords: consistent conclusion, abduction, field of production, default logic, diagnostic reasoning, predictive reasoning, double-stranded break DNA, signaling pathway, metabolic pathway, representation, modeling, biological knowledge.

Remerciements

Le travail présenté dans ce rapport a été réalisé au sein du groupe Diagnostic, Supervision et Conduite (DISCO), du Laboratoire d'Analyse et d'Architecture et des Systèmes (LAAS) du Centre National de la Recherche Scientifique (CNRS).

Il a été effectué grâce au support financier du gouvernement vietnamien, et également du LAAS – CNRS, avec le support de la vie du Centre Régional des œuvres Universitaires et Scolaires (CROUS) de Toulouse.

Je tiens, en premier lieu, à adresser mes plus sincères remerciements et ma plus profonde reconnaissance à Monsieur Andrei DONCESCU et Monsieur Pierre SIEGEL, pour avoir bien voulu être mes directeurs de thèse, pour leur soutien et pour leur direction souple et sécurisante tout au long de ce travail.

Je souhaite exprimer ma plus profonde gratitude à Monsieur Jean ARLAT, Directeur du LAAS – CNRS, pour m'avoir accueilli au sein de laboratoire.

Je tiens à exprimer ma sincère gratitude à Messieurs :

Jean-Charles FAYE, Directeur de Recherche à l'Institut Claudius Regaud (ICR)

Olivier SORDET, Rechercheur à l'Institut Claudius Regaud

Barthélémy DWORKIN

Henri BONNABAU

pour leurs connaissances de la voie de réponse à la cassure double-brin de l'ADN.

Finalement, je tiens également à remercier spécialement tous les amis français et vietnamiens que je côtoie pendant mon séjour en France.

Table des matières

Liste des figures

Liste des tableaux

Introduction

Une fonction biologique n'est pratiquement jamais le produit d'une seule macromolécule mais plutôt le résultat de l'interaction d'un groupe de macromolécules (gènes, protéines). Comprendre les mécanismes complexes à l'œuvre dans la cellule requiert donc une approche intégrative de la modélisation de toutes les interactions entre les macromolécules.

Modéliser, identifier et éventuellement simuler les réseaux d'interactions entre macromolécules qui interviennent à différents niveaux dans la cellule forment les enjeux principaux d'une nouvelle discipline transversale qu'on appelle biologie de systèmes.

Les systèmes biologiques changent sans cesse. La reconstruction de réseaux biologiques à partir de données expérimentales constitue un des éléments clefs des objectifs scientifiques en biologie moléculaire: le biologiste s'intéresse souvent à la réponse cellulaire d'un organisme ou d'un certain tissu dans un organe à un signal ou stress donné. Il cherche par exemple à définir ou à compléter le réseau de régulation impliqué dans le contrôle de cette réponse en exploitant des données expérimentales (données d'expression de gènes etc.). L'apprentissage automatique intervient alors comme une des composantes de l'activité de découverte scientifique : à partir des données, à partir d'une classe de modèles de réseaux de régulation, un algorithme d'apprentissage permet de définir une ou plusieurs solutions candidates (graphe d'interaction et paramètres des modèles) que le biologiste peut ensuite tester en générant d'autres expériences pour vérifier telle ou telle particularité du modèle.

Pour cela, il est habituel de considérer qu'il y a deux modes de raisonnements, deux façons de progresser dans la connaissance : le raisonnement diagnostique, et le raisonnement de prédiction. Le raisonnement diagnostique (ou abductif) est une partie essentielle d'un grand nombre tâches du monde réel, par exemple le diagnostic médical, le débogage des programmes d'ordinateur, la découverte scientifique, etc. En règle générale, le raisonnement abductif est le processus permettant de trouver la meilleure explication pour un ensemble d'observations (Josephson, 1994). Ces observations peuvent être les symptômes d'un patient, les messages d'erreur d'un programme d'ordinateur, ou les résultats d'une expérience. La tâche de la résolution des problèmes dans chacun de ces domaines est de trouver un ensemble d'hypothèses élémentaires qui explique le mieux ces symptômes.

Une autre méthode est tout aussi importante pour la représentation et le traitement des connaissances biologiques, c'est le raisonnement de prédiction par la logique des défauts. Quand un système intelligent essaye de résoudre un problème, il peut être en mesure de s'appuyer sur des informations complètes sur ce problème, et sa tâche principale est de tirer la bonne conclusion par un raisonnement classique. Dans ce cas, la logique des prédicats classique peut être suffisante.

Cependant, dans de nombreuses situations, le système a seulement l'information incomplète, parce que certaines informations ne sont pas disponibles, ou bien parce qu'il doit répondre vite et n'a pas de temps de recueillir toutes les données pertinentes. La logique classique a en effet la capacité de représenter et raisonner avec certains aspects de

l'information incomplète. Mais il y a des occasions où l'information supplémentaire doit être remplie pour surmonter l'incomplétude, parce que certaines décisions doivent être prises. Dans ce cas, le système doit faire des conjectures plausibles, qui dans le cas du raisonnement par défaut sont basés sur des règles empiriques, appelé des défauts. Par exemple, un médecin d'urgence doit faire des conjectures sur les causes les plus probables des symptômes observés. Évidemment, il serait inapproprié d'attendre le résultat de tests éventuellement étendus et chronophages avant le début du traitement.

Puisque les décisions sont fondées sur des hypothèses, elles peuvent se révéler fausses face à de nouvelles informations qui seront disponibles, à savoir les examens médicaux peuvent conduire à un diagnostic modifié. Le phénomène d'avoir à reprendre certaines conclusions précédentes est appelé non-monotonie, ça veut dire que si une déclaration X découle d'une série de prémisses M, et M est un sous-ensemble de N, X ne découle pas nécessairement de N.

La logique des défauts, à l'origine présentée by Reiter [1980], fournit la méthode formelle pour soutenir ce genre de raisonnement. Elle est peut-être la méthode la plus importante pour le raisonnement non-monotone, essentiellement en raison de la simplicité de l'idée d'un défaut, et parce que les défauts prévalent dans de nombreux domaines d'application. Cependant, il existe plusieurs décisions de conceptions alternatives qui ont conduit à des variations de l'idée initiale. En fait, il y a une famille de méthodes de raisonnement par défauts qui partagent les mêmes fondements.

Le travail présenté dans ce mémoire se focalise sur le raisonnement par l'abduction et par la logique des défauts pour la modélisation des voies de signalisation en réponse à la cassure double-brin de l'ADN.

En fait, on a utilisé les algorithmes de production avec un champ de production pour faire le raisonnement diagnostique sur la carte d'interactions de Pommier. Ensuite, la logique des défauts a été utilisée pour faire le raisonnement de prédiction à partir de la carte. Toutefois, cette méthode ne nous permettait pas de connaitre l'ordre dans lequel se déroulaient les événements. Nous avons alors ajouté une variable temps à la logique des défauts ce qui nous a permis d'obtenir une chronologie des événements.

Dans le processus de travail, on se rend compte que les algorithmes ne fonctionnent pas bien avec les variables lorsqu'ils sont implémentés en Prolog, alors on a cherché à résoudre ce problème et nous avons fourni des solutions.

Plan de la thèse

Ce document s'organise en quatre chapitres suivants :

Le chapitre 1 introduit le contexte applicatif considéré dans le cadre de la thèse, en examinant les notions de gènes, de protéines et de métabolites, ainsi que les règles de base de comportement d'une cellule.

Le chapitre 2 présente les bases des outils mathématiques utilisés pour exécuter la synthèse des voies de signalisation. Ce chapitre se concentre sur la conclusion conséquence et la production.

Le chapitre 3 détaille la logique des défauts, l'outil est utilisé pour représenter les interactions dans les réseaux de régulation.

Le chapitre 4 propose une approche de représentation des connaissances – de la voie de signalisation cellulaire : utilisation de la production pour le raisonnement diagnostique ; et de la logique des défauts pour le raisonnement de prédiction.

En conclusion, nous présentons un bilan du travail réalisé et des orientations pour développer le sujet.

Chapitre 1. Contexte et cas d'application

Ce chapitre présente le contexte et le cas d'application de la thèse. Tout d'abord, il faut examiner les notions de gènes et leurs interactions dans les réseaux biologiques.

1.1. Contexte

Comme nous le savons, la plupart des médicaments anticancéreux qui sont encore actuellement utilisés en clinique ciblent l'ADN. L'action de ces médicaments anticancéreux sur les tissus tumoraux est probablement due au fait qu'ils peuvent supprimer les défauts tumeur-spécifiques des points de contrôle (checkpoint en anglais) du cycle cellulaire et améliorer la réparation de l'ADN, afin d'augmenter la réponse apoptotique des cellules tumorales. Nous nous intéresserons ici à l'interaction moléculaire qui apparaît dans la voie ATM-Chk2.

1.1.1. Connaissances biologiques

L'interaction moléculaire implique une liste de notions biologiques qui doivent être clarifiées.

D'abord, nous étudierons la notion de cycle cellulaire. Le cycle cellulaire est l'ensemble des phases par lesquelles une cellule passe entre deux divisions successives. Le cycle des cellules eucaryotes est divisé en quatre phases : G_1, S, G_2 et M. L'ensemble des trois premières phases est souvent appelé l'interphase.

Cellule : La cellule est une unité structurale et fonctionnelle de la plupart des organismes. Chaque organisme est structuré d'une ou plusieurs cellules. Des cellules ne sont produites qu'à partir des cellules précédentes. Toutes les fonctions vitales d'un organisme ont lieu dans la cellule. Les cellules contiennent des informations génétiques nécessaires pour diriger leurs fonctions et peuvent transmettre les matériaux génétiques aux générations suivantes.

Caractères de la cellule : Chaque cellule est un système ouvert, autonome et auto-productif. La cellule peut recevoir des nutriments, les convertir en énergie, exercer des fonctions spéciales, et produire des nouvelles cellules s'il est nécessaire. Chaque cellule contient un cryptage distinct dirigeant ses actions et a les capacités suivantes :

- Reproduction par la division.
- Métabolisme cellulaire : Recevoir des matières brutes et les transformer en substances nécessaires pour la cellule, produire les molécules à haute énergie et les sous-produits. Pour exercer leurs fonctions, les cellules ont besoin d'absorber et d'utiliser l'énergie chimique contenue dans les molécules organiques. Cette énergie est libérée dans les voies métaboliques.
- Faire la synthèse des protéines. Ce sont des molécules qui assument des fonctions fondamentales de la cellule, par exemple les enzymes.
- Répondre aux stimuli ou aux changements d'environnement extérieur tels que les changements de température ou de pH ou des éléments nutritifs.

- Déplacer des vésicules.

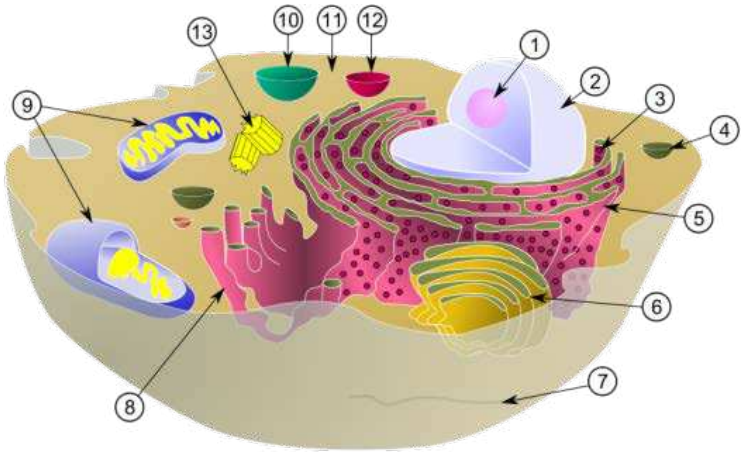

Figure 1.1. Structure d'une cellule animale eucaryote typique
(source : wikipedia.org)
(1) Nucléole, (2) Noyau, (3) Ribosome, (4) Vésicule, (5) Réticulum endoplasmique rugueux (granuleux), (6) Appareil de Golgi, (7) Microtubule, (8) Réticulum endoplasmique lisse, (9) Mitochondrie, (10) Lysosome, (11) Cytoplasme, (12) Peroxysome, (13) Centrosome

Composants d'une cellule : chaque cellule a une membrane plasmique pour l'envelopper, isoler l'intracellulaire de l'extérieur, contrôler strictement le transport des substances, maintenir le potentiel de membrane et la concentration des substances intérieures et extérieures. Chaque cellule contient des molécules d'ADN, matériels génétiques importants et des molécules d'ARN qui participent directement au processus de synthèse des protéines. À l'intérieur de la cellule, dans les temps donnés, la cellule synthétise une grande variété de molécules différentes.

- Membrane plasmique : l'enveloppe d'une cellule a la fonction d'encapsulation et de distinction de cellule avec le milieu environnant. La membrane est formée par une double couche de lipides et de protéines.
- Cytosquelette : un composant important compliqué et flexible. Il inclut un système de microtubules et de protéines et forme et maintient la forme de la cellule.
- Cytoplasme : le cytoplasme désigne le contenu d'une cellule vivante. Il s'agit de la totalité du matériel cellulaire du protoplasme délimité par la membrane plasmique et des organites.
- Matériel génétique : ce sont des molécules d'acides nucléides (ADN et ARN). L'information génétique de l'organisme est le code génétique qui prescrit toutes les protéines nécessaires pour toutes les cellules d'un organisme.
- Organites : les cellules ont souvent des petits organes appelés des organites, adaptés et différenciés pour une ou plusieurs fonctions vitales. Les organites se trouvent souvent dans les cellules eucaryotes et souvent ont leurs propres membranes.

Noyaux : les noyaux sont aussi entourés d'une membrane les isolant du cytoplasme et contiennent des acides nucléiques, ce sont des grandes molécules ayant la structure multimoléculaire, incluant plusieurs molécules de nucléotides. Il existe deux types d'acides nucléiques : l'acide désoxyribonucléique (ADN) et l'acide ribonucléique (ARN). L'ADN

contient l'information génétique lorsque l'ARN est la copie d'ADN, souvent en un seul brin alors que l'ADN a deux brins.

Nucléotide : Un nucléotide est une molécule organique. Certains nucléotides forment la base de l'ADN et de l'ARN, d'autres sont des cofacteurs ou coenzymes. Chaque molécule nucléotidique consiste en trois composants, ce sont une base azotée, un sucre et un groupement phosphate (ou acide phosphorique).

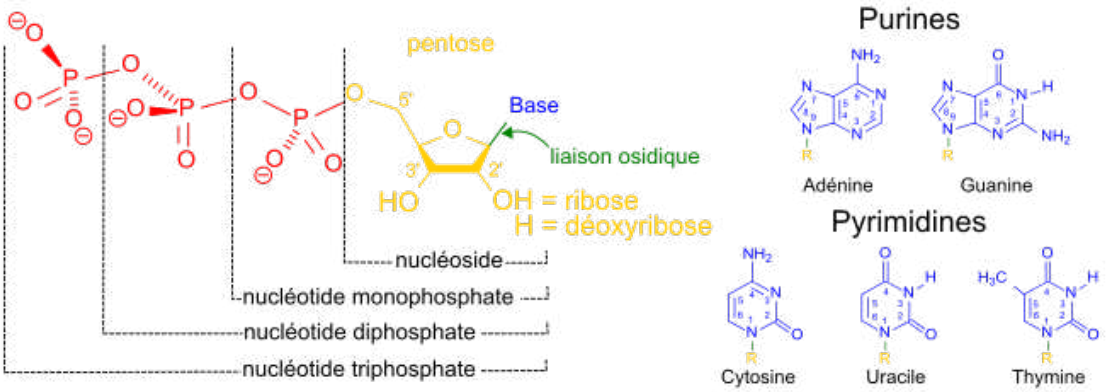

Figure 1.2. Structure des nucléotides
(source : wikipedia.org)

- Base azotée : Variable en fonction du type de nucléotide (purine ou pyrimidine) fixé à l'atome de carbone 1' du désoxyribose. Pour l'ADN, il existe quatre nucléotides différents correspondant aux quatre bases azotées différentes, ce sont l'adénine (A), la guanine (G), la cytosine (C), et la thymine (T). Dans l'ARN, la thymine (T) est remplacée par l'uracile (U).
- Sucre : Pentose (ou désoxyribose), les acides nucléiques se différencient par la base qui est fixée sur le sucre.
- Groupement phosphate (acide phosphorique) : Grâce à la liaison covalente entre l'acide phosphorique d'un nucléotide avec un sucre, des nucléotides se relient et forment une chaîne de poly-nucléotides. Cette liaison suit la règle complémentaire : une grande base est supplée par une petite base. Alors il existe deux types de liaison : le dAMP (adénine) avec le dTMP (thymine) en établissant deux liaisons hydrogènes (A-T), et le dCMP (cytosine) avec le dGMP (guanine) en établissant trois liaisons hydrogènes (G-C). Par exemple :

si on a un brin	A	G	G	C	T	A	C
alors l'autre brin est	T	C	C	G	A	T	G

ADN : L'acide désoxyribonucléique (ADN) est une molécule, elle est présente dans toutes les cellules vivantes, renferme l'ensemble des informations nécessaires au développement et au fonctionnement d'un organisme. C'est aussi le support de l'hérédité car il est transmis lors de la reproduction, de manière intégrale ou non. Il porte l'information génétique et constitue le génome des êtres vivants. La structure standard de l'ADN est une double-hélice droite, composée de deux brins complémentaires. Chaque brin d'ADN est constitué d'un enchaînement de nucléotides.

La fonction de l'ADN est de préserver et de transmettre des informations sur la structure des tous les types de protéines d'un organisme, de sorte qu'elles définissent les caractères de l'organisme. L'information génétique (l'information de structure des protéines) est codée dans l'ADN. Chaque section (tranche) de molécule ADN qui porte l'information régulant la séquence d'une protéine est appelée gène de structure (cistron). Chaque acide aminé de la molécule de protéine est régulé par trois nucléotides successifs dans l'ADN (codon), c'est le code triplet. La succession de trois nucléotides correspondant à un acide aminé est appelé l'unité de code (codon). Il est facile de voir que le nombre de combinaisons des trois à partir des quatre types de nucléotides est $4^3 = 64$. En fait, seulement 61 combinaisons sont utilisées pour coder 22 acides aminés. Il existe certains cas où un acide aminé correspond à plusieurs codons différents (dégénérescence du code génétique). D'autre part, il existe des codons qui ne régulent aucun acide aminé, ils ont juste la tâche de finir la synthèse de chaîne polypeptique. Ce sont UAA, UAG, et UGA.

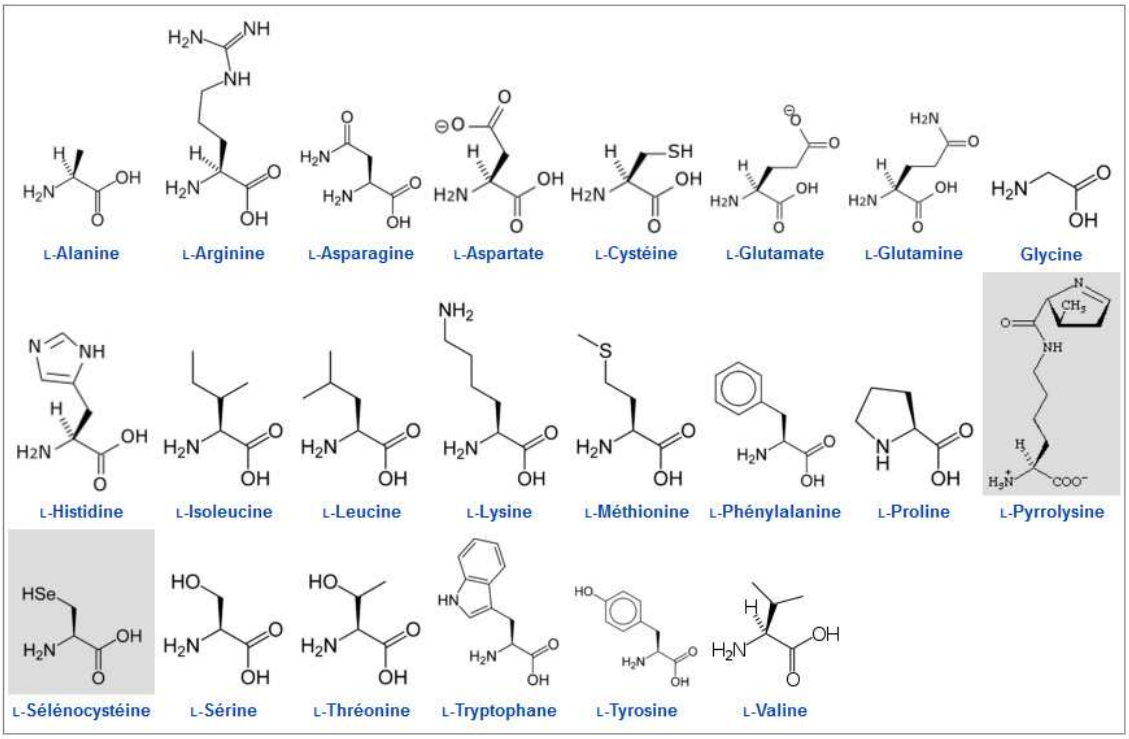

Figure 1.3. Structure des 22 acides aminés
(source : wikipedia.org)

Un caractère très important de l'ADN est la capacité de réplication (reproduction). Sous l'action d'enzymes, la double hélice de l'ADN est étirée, deux brins se détachent pas à pas. Chaque nucléotide dans chaque brin se combine avec un nucléotide libre suivant la règle (A-T, G-C) pour former une nouvelle double hélice, donnant deux molécules enfants.

Tableau 1.1. Les codes des 22 acides aminés

Code	Abrév	Acide aminé	Nature	Codons
Y	Tyr	Tyrosine	Polaire, aromatique	UAU, UAC
W	Trp	Tryptophane	Apolaire, aromatique	UGG (et UGA chez les mycoplasmes)
V	Val	Valine	Apolaire, aliphatique	GUU, GUC, GUA, GUG
U	Sec	Sélénocystéine	Polaire	UGA associé à un élément SECIS
T	Thr	Thréonine	Polaire	ACU, ACC, ACA, ACG
S	Ser	Sérine	Polaire	UCU, UCC, UCA, UCG, AGU, AGC
R	Arg	Arginine	Basique	CGU, CGC, CGA, CGG, AGA, AGG
Q	Gln	Glutamine	Polaire	CAA, CAG
P	Pro	Proline	Apolaire	CCU, CCC, CCA, CCG
O	Pyl	Pyrrolysine	Polaire	UAG associé à un élément PYLIS
N	Asn	Asparagine	Polaire	AAU, AAC
M	Met	Méthionine	Apolaire	AUG
L	Leu	Leucine	Apolaire, aliphatique	UUA, UUG, CUU, CUC, CUA, CUG
K	Lys	Lysine	Basique	AAA, AAG
I	Ile	Isoleucine	Apolaire, aliphatique	AUU, AUC, AUA
H	His	Histidine	Basique, aromatique	CAU, CAC
G	Gly	Glycine	Apolaire, aliphatique	GGU, GGC, GGA, GGG
F	Phe	Phénylalanine	Apolaire, aromatique	UUU, UUC
E	Glu	Acide glutamique	Acide	GAA, GAG
D	Asp	Acide aspartique	Acide	GAU, GAC
C	Cys	Cystéine	Polaire	UGU, UGC
A	Ala	Alanine	Apolaire, aliphatique	GCU, GCC, GCA, GCG

ARN : l'acide ribonucléique (ARN) est une molécule biologique trouvée pratiquement dans tous les organismes vivants, y compris dans certains virus. L'ARN est une molécule très proche chimiquement de l'ADN. Selon sa fonction, l'ARN est divisée en trois types : ARN messager (ARNm), ARN de transfert (ARNt), et ARN ribosomique (ARNr).

- ARNm : l'ARN messager est une chaîne poly nucléotidique copiant exactement un ou parfois plusieurs cistrons d'ADN, mais l'Uracile (U) remplace la Thymine (T). L'ARNm a la tâche de transmettre l'information génétique qui précise la séquence d'acides aminés de la protéine synthétisée.
- ARNt : l'ARN de transfert a la tâche de transporter des acides aminés au lieu de synthèse des protéines (le ribosome). L'ARN de transfert a une structure caractéristique en feuille de trèfle, composée de quatre tiges appariées. L'une de ces tiges est terminée par une boucle qui contient l'anticodon, le triplet de nucléotides qui s'apparie au codon lors de la traduction d'un ARNm par le ribosome. À l'autre extrémité, l'ARNt porte l'acide aminé correspondant attaché par une liaison ester à son extrémité 3'-OH. Cette estérification est catalysée par des enzymes spécifiques, les aminoacyl-ARNt synthétases.
- ARNr : L'ARN catalytique ou ribozyme constitue les composants du ribosome.

Chromosome : Un chromosome est un élément microscopique constitué de molécules d'ADN et de protéines. Chez les cellules eucaryotes, les chromosomes se trouvent dans le noyau où ils prennent la forme soit d'un bâtonnet, soit d'un écheveau, selon qu'ils sont condensés ou non. Chez les cellules procaryotes (sans noyau), le chromosome se trouve dans le cytoplasme.

Le chromosome est l'élément porteur de l'information génétique. Les chromosomes contiennent les gènes et permettent leur distribution égale dans les deux cellules enfants lors de la division cellulaire. Ils sont formés d'une longue molécule d'ADN, associée à des protéines (notamment les histones). Entre deux divisions, la séparation entre les molécules d'ADN différentes (chromosomes) est peu perceptible, l'ensemble porte alors le nom de chromatine. Ils se condensent progressivement au cours de la division cellulaire pour prendre une apparence caractéristique en forme de X à deux bras courts et deux bras longs, reliés par un centromère.

Au figuré, le mot chromosome est utilisé pour décrire son contenu en termes d'information génétique. Les chromosomes portent les gènes, supports de l'information génétique transmise des cellules mères aux cellules filles lors de la mitose ou de la méiose (reproduction sexuée).

Les chromosomes sont habituellement représentés par paires, en parallèle avec leur homologue. Ils sont souvent illustrés sous leur forme condensée et dupliquée (en métaphase de la mitose).

L'ensemble des chromosomes est représenté sur un caryotype, ou carte de chromosomes.

Processus de synthèse des protéines : L'ADN régule la séquence des protéines par l'ARNm, donc la synthèse de protéine se compose de deux étapes principales : la transcription et la traduction.

- Transcription : C'est le processus de synthèse de l'ARNm. La plupart de l'ARN est synthétisée sur le modèle de l'ADN. Sous l'action des enzymes ARN-polymérases, une section de molécule d'ADN correspondant à un ou plusieurs gènes est étirée, deux brins se détachent. Chaque nucléotide du brin d'origine est relié avec un ribo-nucléotide libre selon la règle (A-U, G-C), formant une chaîne polyribonucléotidique d'ARN. Après la synthèse, pour la cellule eucaryote, l'ARNm quitte le noyau vers le cytoplasme pour participer au processus de synthèse de protéine.
- Traduction : Elle inclut deux étapes principales, activation d'acide aminé et synthèse de polypeptide. Lors de l'étape de synthèse de l'acide aminé, des acides aminés libres dans le cytoplasme sont activés par la liaison avec des composés riches en énergie adénosines triphosphates (ATP) par l'action de certains enzymes. Ensuite, grâce à d'autres enzymes spécifiques, l'acide aminé activé est lié avec l'ARNt correspondante pour former l'acide aminé – ARNt (aa-ARNt). Dans la synthèse du polypeptide, d'abord, l'ARNm se lie avec le ribosome à la place du codon de départ. Ensuite, l'ARNt porte le premier acide aminé vers le ribosome, son codon coïncide avec le code de départ de l'ARNm par la règle (A-U, G-C). aa_1-ARNt arrive à la position suivante, son codon coïncide avec le codon du premier acide aminé d'ARNm. Par la

suite, la chaîne forme la structure de degré plus élevé pour former la protéine complète. Par exemple :

brin origine ADN	T	A	C	G	T	A	C	G	G	A	A	T	A	A	G
brin d'ARNm	A	U	G	C	A	U	G	C	C	U	U	A	U	U	C
Décodage	Méthionine			Histidine			Alanine			Leucine			Phénylalanine		

Mort cellulaire : La cellule ayant reçu un signal de son environnement va exprimer un programme qui entraîne sa mort (l'apoptose est un de ces mécanismes). Ce phénomène est nécessaire au développement des organismes pluricellulaires ; autant chez les végétaux (avec par exemple la mort des cellules formant le tube criblé), que chez les animaux (lors de la mise en place de la main chez l'homme : On a initialement une main palmée, la mort des cellules permet l'individualisation des doigts). Ce phénomène a aussi été découvert chez certaines bactéries (la mort cellulaire permet de limiter le nombre de bactéries lorsque les ressources sont insuffisantes).

La cellule, tant pour les êtres pluricellulaires que pour les unicellulaires, constitue une structure vouée avant tout à permettre la reproduction de l'organisme et donc la transmission d'une structure de base contenant un programme génétique. Ainsi, certains auteurs ont été amenés à formuler la théorie du gène égoïste, considérant les organismes (et donc les cellules) comme de simples structures destinées à assurer la transmission et la prolifération des gènes (le gène proliférant est qualifié d'égoïste). La mort cellulaire peut se faire par nécrose, par autophagie, ou par apoptose.

- Mort cellulaire par nécrose : La mort est causée par des dommages physiques ou chimiques. La nécrose est causée par des enzymes spéciales produites par les lysosomes, petites usines à enzymes de la cellule. La cassure de la membrane plasmique qui en résulte conduit à la libération dans le milieu extérieur du contenu cytoplasmique. La nécrose est accompagnée habituellement d'une réponse inflammatoire qui consiste en la présence d'exsudat et de cellules spécialisées du système hématopoïétique comme les lymphocytes et les macrophages. Une nécrose peut également être causée par une ischémie (plus ou moins) longue d'un membre.
- Mort cellulaire par autophagie : La dégradation d'une partie du cytoplasme de la cellule par ses propres lysosomes. Le terme 'autophagie' (se manger soi-même) regroupe plusieurs voies de dégradation de lysosome des constituants cellulaires, essentielles à l'homéostasie cellulaire. Il existe trois types différents d'autophagies : la micro-autophagie, l'autophagie réalisée par des protéines chaperonnées, et la macro-autophagie (c'est le type principal).
- Mort cellulaire par apoptose : Ou mort cellulaire programmée, c'est un processus par lequel des cellules déclenchent leur auto-destruction en réponse à un signal. C'est l'une des voies de mort cellulaire, génétiquement programmée, nécessaire à la survie des organismes multicellulaires. Elle est en équilibre constant avec la prolifération cellulaire. Contrairement à la nécrose, l'apoptose ne provoque pas d'inflammation : les membranes plasmiques ne sont pas détruites, du moins dans un premier temps, et la cellule émet des signaux.

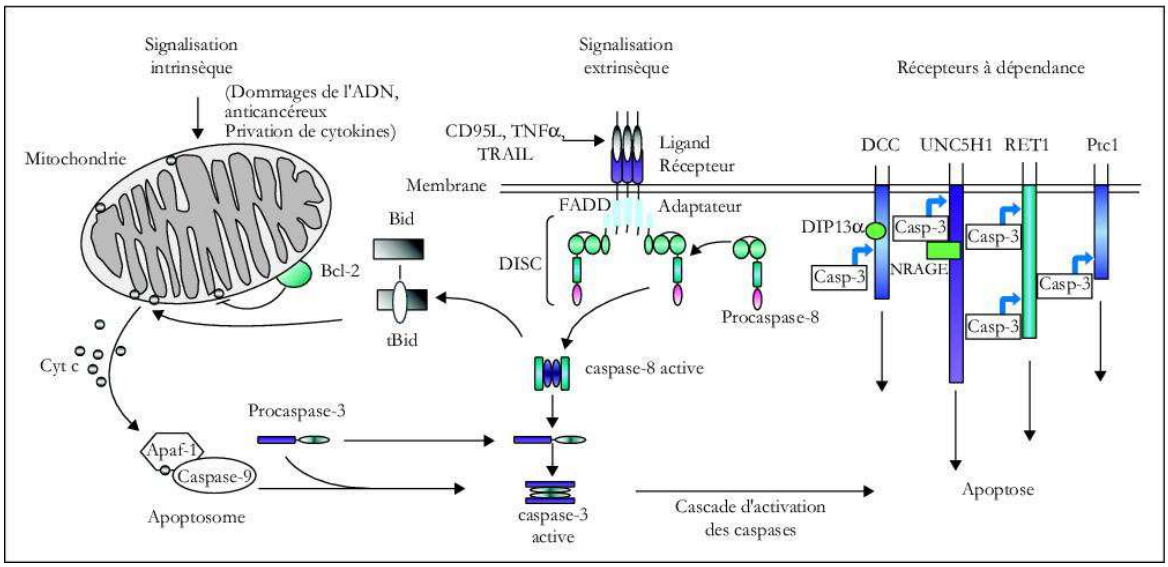

Figure 1.4. Mécanismes intracellulaires de régulation de l'apoptose
(source : wikipedia.org)

Reproduction cellulaire : Comme mentionné ci-dessus, une capacité importante de la cellule est la reproduction (réplication, ou bien duplication) par la division. Ceci est fait pendant un cycle appelé le cycle cellulaire (ou bien cycle de division cellulaire).

Avant d'entrer en division cellulaire, une cellule doit accumuler des éléments nutritifs durant l'interphase. L'interphase procède en trois phases : G_1, S, et G_2. La division cellulaire fonctionne dans un cycle.

La première phase dans l'interphase est la phase G_1 (phase gap 1, ou post-mitotique phase), c'est une période principale de la croissance cellulaire. Au cours de cette phase, la biosynthèse, qui a été diminuée considérablement pendant la phase M précédente, recommence à haut taux. La cellule se prépare pour le processus de réplication de l'ADN. La cellule intègre les mitogènes, grossit et en fin de G_1 la cellule peut se mettre en pause ou quitter le cycle cellulaire.

Un point de contrôle important en G_1 a été identifié pour toutes les cellules de levure et de mammifère. C'est le moment où la cellule va entrer dans la phase de réplication de l'ADN avant l'achèvement d'un cycle cellulaire.

La phase suivante S commence quand la synthèse (réplication) de l'ADN commence. C'est une période au cours de laquelle le matériel chromosomique est doublé par réplication de l'ADN : duplication des chromosomes. Lorsque la phase est complète, tous les chromosomes ont été produits, c'est à dire chaque chromosome a deux chromatides enfants. Ainsi, au cours de la phase, la quantité d'ADN dans la cellule est effectivement doublée, quoique la ploïdie de la cellule ne change pas. Les vitesses de transcription et de synthèse protéiques sont très basses pendant cette phase.

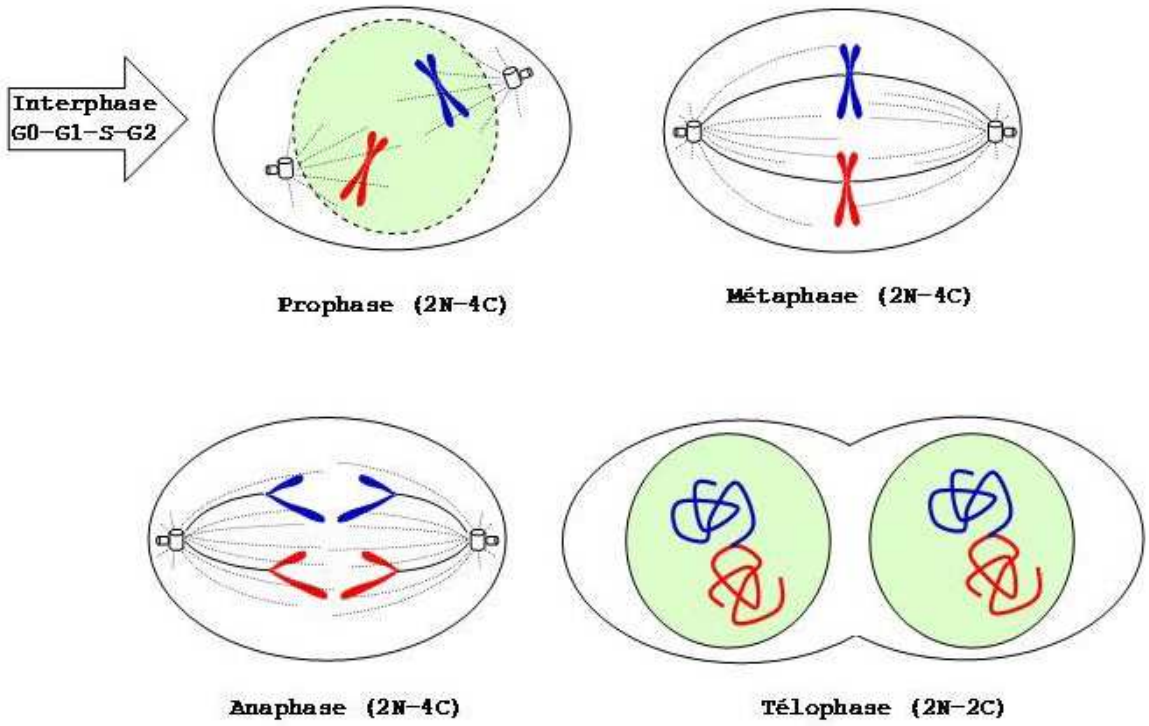

Figure 1.5. Les étapes du cycle cellulaire : la mitose
(source : wikipedia.org)

Ensuite, la phase G_2 est la seconde phase de croissance cellulaire, où la cellule se prépare à se diviser en deux cellules enfants. À l'issue de cette phase, chaque chromosome est parfaitement identique à son homologue sur le plan morphologique et du point de vue des gènes présents, mais chaque gène n'est pas nécessairement identique à son homologue (généralement plusieurs allèles existent).

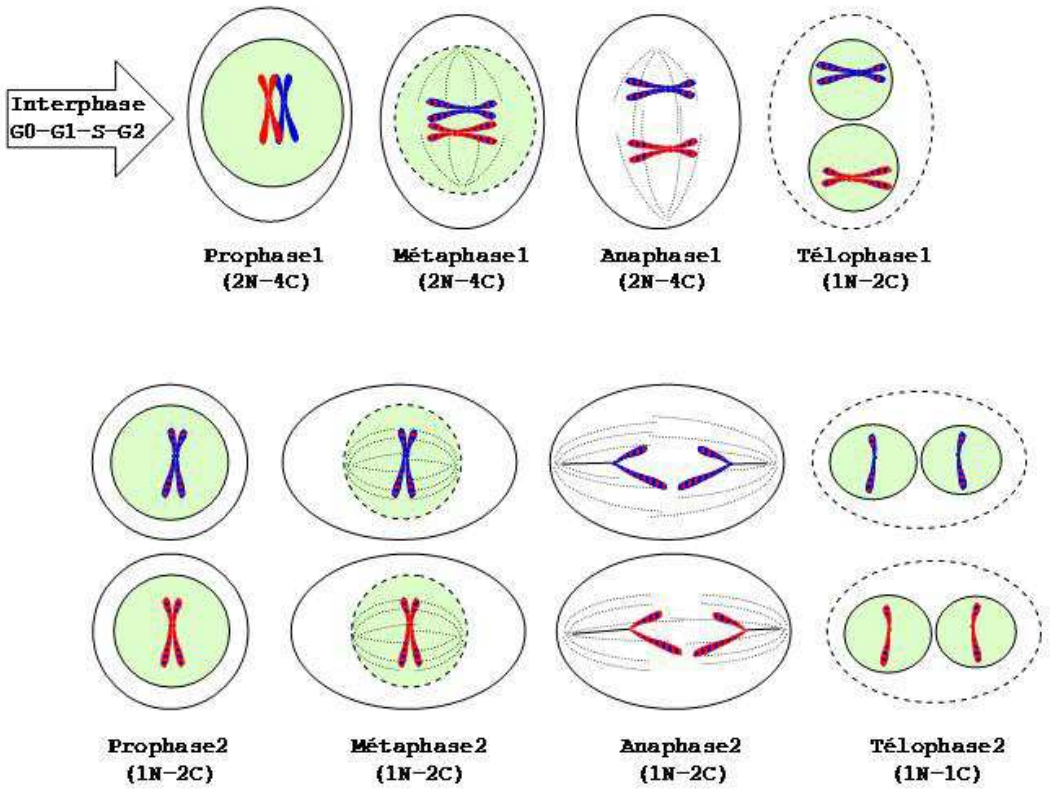

Figure 1.6. Les étapes du cycle cellulaire : la méiose
(source : wikipedia.org)

Enfin, c'est la phase M. La lettre M représente la mitose. La mitose est un processus dans lequel une cellule eucaryote sépare les chromosomes dans son noyau cellulaire en deux ensembles identiques dans deux noyaux. Il est généralement suivi immédiatement par la cytokinèse, qui divise le noyau, le cytoplasme, les organelles et la membrane cellulaire en deux cellules contenant des parts à peu près égales de ces composants cellulaires. L'ensemble de la mitose et la cytokinèse définissant la phase mitotique (phase M) du cycle cellulaire – la division de cellule parente en deux cellules enfantes, généralement identiques et ressemblent à leur parent. La phase M a été décomposée en plusieurs phases distinctes, successivement connues sous les noms prophase, métaphase, anaphase, télophase, et cytokinèse. Les erreurs dans une mitose peuvent tuer la cellule par l'apoptose ou causer la mutation qui peut mener au cancer.

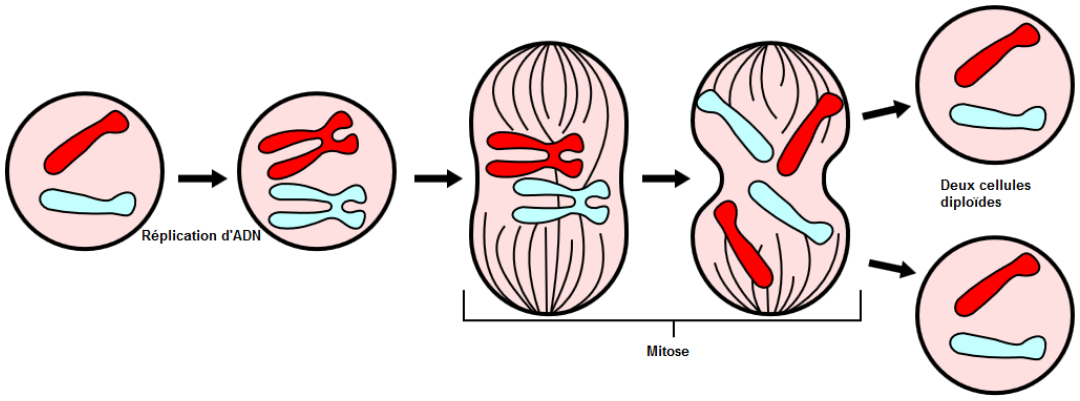

Figure 1.7. Division des chromosomes dans un noyau cellulaire
(source : wikipedia.org)

Pendant le cycle cellulaire, il existe une phase supplémentaire G_0. C'est une phase de repos où la cellule quitte le cycle et arrête de se diviser. Les cellules non-prolifératives dans les eucaryotes multicellulaires sont en phase G_0 de G_1 et peuvent rester tranquillement pour longtemps et peut être indéfiniment. La décrépitude de la cellule est un état qui a lieu en réponse au dommage de l'ADN ou à la dégradation de l'ADN qui peuvent faire un descendant cellulaire non-viable. Il est souvent une alternative biochimique à l'autodestruction par apoptose d'une telle cellule endommagée.

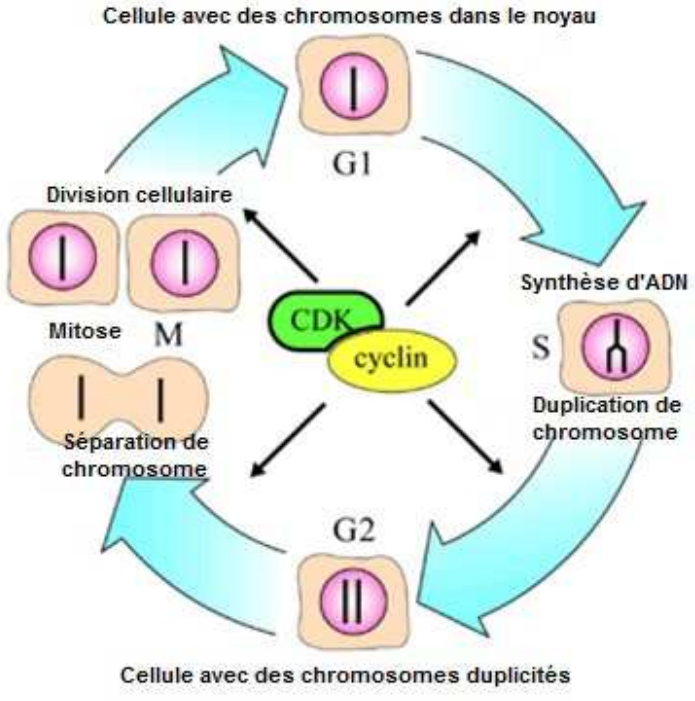

Figure 1.8. Les quatre phases du cycle cellulaire avec des chromosomes dans le noyau
(source : wikipedia.org)

Point de contrôle : Un terme très important qui a été rappelé est le point de contrôle. Le point de contrôle est utilisé par la cellule pour contrôler et régler le processus de cycle cellulaire. Les points de contrôle bloquent le processus de cycle cellulaire aux points spécifiques, laissent la vérification de phase nécessaire se faire et permettent de réparer les dommages à l'ADN. En d'autres mots, le point de contrôle traite l'ensemble des signaux

régulateurs reçus par la cellule et décide si elle doit se diviser. La cellule n'a pas pu passer à la phase suivante jusqu'à ce que le point de contrôle ait été passé.

Quelques points de contrôle sont désignés pour assurer que les ADN incomplets ou endommagés ne sont pas passés à la cellule enfante. Il existe deux points de contrôle principaux : le point G_1/S et le point G_2/M. La transition G_1/S est un pas dans le cycle cellulaire et est aussi appelé le point de restriction (non-retour). Le point G_1/S est le contrôle de l'état de l'ADN. Si c'est bon, il met en route la réplication par l'activation de facteurs inducteurs de la réplication. Par contre, le point G_2/M est le contrôle de l'état de l'ADN répliqué et le contrôle de la duplication des centrosomes. Si c'est bon, il met en route la mitose par l'activation de facteurs mitotiques.

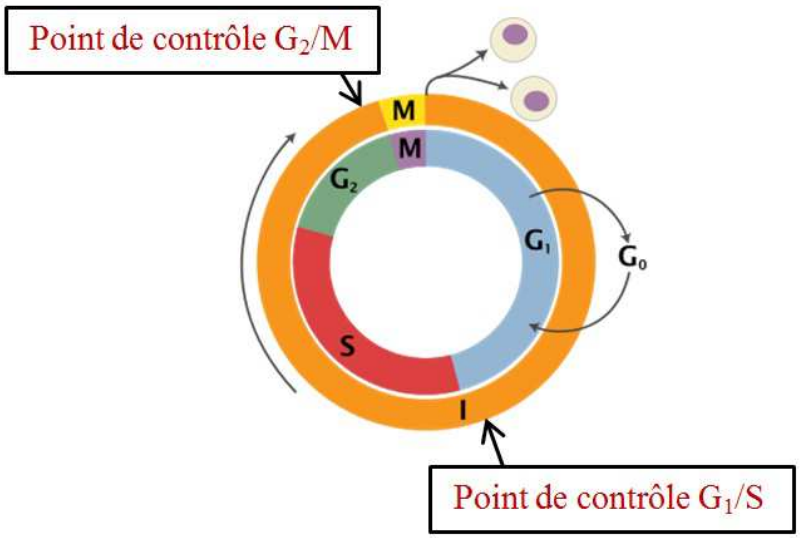

Figure 1.9. Cycle cellulaire avec des points de contrôle
(source : wikipedia.org)
Bague extérieure : I – interphase, M – Mitose ; Bague intérieure : M – Mitose, G_1 – Gap 1, G_2 – Gap 2, S – Synthèse ; Pas dans les bagues : G_0 – Gap 0/ Repos

Lorsque l'ADN génomique des cellules eucaryotes est endommagé par des processus spontanés, mutagènes chimiques ou l'exposition au soleil, la réplication de l'ADN endommagé déclenche une réponse cellulaire appelée un point de contrôle post-réplication. Cette réponse empêche la progression du cycle cellulaire jusqu'à ce que les processus de la réparation post-réplicative soient terminés et puissent contrôler l'activité de ces voies de réparation de l'ADN. Dans les types de cellules qui exécutent la phase S avant la mitose, comme la levure de fission et les cellules humaines, le point de contrôle post-réplicatif prend le temps de réparation en retardant le début de la mitose. Dans les types cellulaires où la mitose et la phase S sont concurrentes, tels que la levure bourgeonnante, les retards aux points de contrôle post-réplicatifs sont l'état d'avancement de la mitose à la métaphase.

Rôle de complexes cycline-CDK : Les complexes cycline-CDK sont des kinases capables de phosphoryler des protéines substrats. Ils sont constitués des deux sous-unités : la cycline (sous-unité activatrice de la kinase), et le CDK (sous-unité catalytique de la kinase). Les cyclines n'ont pas d'activité catalytique et les CDKs sont inactives en absence d'une

partenaire cycline. Les CDKs effectuent une commune réaction biochimique lorsqu'activées par une cycline liée appelée la phosphorylation ; ceci active ou ralentit les protéines cibles pour orchestrer l'entrée coordonnée à la phase suivante du cycle cellulaire. Des combinaisons cycline-CDK différentes déterminent les protéines ciblées en aval. Les CDKs sont exprimées constitutivement dans les cellules alors que les cyclines sont synthétisées à des étapes spécifiques du cycle, en réponse à divers signaux moléculaires.

Lors de la réception d'un signal pro-mitotique extracellulaire, les complexes cycline-CDK de G_1 deviennent actifs afin de préparer la cellule pour la phase S, en promouvant l'expression des facteurs de transcription qui à leur tour promeuvent l'expression des cyclines de S et des enzymes nécessaires pour la réplication de l'ADN. Les complexes cycline-CDK de G_1 promeuvent également la dégradation des molécules qui fonctionnent comme les inhibiteurs de la phase S en les ciblant pour l'ubiquitination. Une fois qu'une protéine a été ubiquitinilée, elle est dégradée par le protéasome.

Les complexes cycline-CDK actifs de S phosphorylent des protéines qui forment des complexes pré-réplications assemblés durant la phase G_1 sur les origines de réplication de l'ADN. La phosphorylation sert à deux objets : pour activer chaque complexe pré-réplication déjà-assemblé, et pour prévenir la formation de nouveaux complexes. Ceci assure que chaque portion de génome de la cellule sera reproduite une fois et une seule fois. La raison de prévention de gaps en réplication est claire, parce que les cellules enfants manquant de tout ou partie des gènes essentiels seront mortes. Toutefois, pour les raisons liées aux effets du nombre de copies de gènes, la possession de copies extra de certains gènes est aussi délétère pour les cellules enfants.

Les complexes mitotiques cycline-CDK, synthétisés mais inactifs pendant les phases S et G_2, promeuvent l'initiation de mitose en stimulant les protéines en aval impliquées dans la condensation des chromosomes et l'assemblage du fuseau mitotique. L'ubiquition ligase fait partie d'un complexe (APC) indispensable à la promotion de l'anaphase. Ce complexe chargé de dégrades les protéines vise également les cyclines mitotiques.

La cycline D est la première cycline produite dans le cycle cellulaire, en réponse aux signaux extracellulaires (facteurs de croissance, par exemple). La cycline D se lie à la CDK4 existante, forme le complexe de cycline D-CDK4 actif. Le complexe de cycline D-CDK4 à son tour phosphoryle la protéine du rétinoblastome susceptibilité (Rb). Le Rb hyper-phosphorylée se dissocie du complexe E2F/DP1/Rb, ceci active E2F. L'activation d'E2F entraîne la transcription de divers gènes, comme cycline E, cycline A, ADN polymérase, etc… La cycline E ainsi produite se lie à CDK2, ceci forme le complexe de cycline E-CDK2, qui pousse la cellule de phase G_1 à la phase S (transition G_1/S). La cycline B avec le complexe cdc2 (cdc2 – levures à fission, CDK1 – mammalien) forme le complexe de cycline B-cdc2, qui prend l'initiation de la transition G_2/M. L'activation du complexe cycline B-cdc2 cause la rupture de l'enveloppe nucléaire et l'initiation de la prophase. Et par la suite, sa désactivation entraîne la cellule à sortir de la mitose.

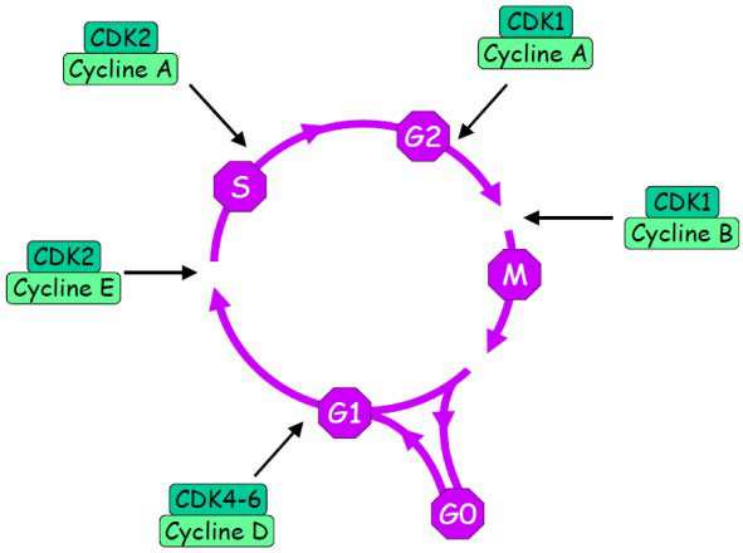

Figure 1.10. Molécules régulant intervenant dans le cycle cellulaire chez les mammifères (source : wikipedia.org)
cdc2 (levures) = CDK1 (mammifères)

Il semble qu'un réseau semi-autonome de la transcription agit en concert avec la machinerie cycline-CDK pour régler le cycle cellulaire. Plusieurs études d'expression génique ont identifié environs 800 à 1200 gènes dont l'expression change au cours du cycle cellulaire. Ils sont transcrits à des niveaux élevés à des points spécifiques du cycle cellulaire, et restent à des niveaux faibles dans le reste du cycle cellulaire. Tandis que l'ensemble des gènes identifiés diffère entre les études en raison de méthodes de calcul et des critères utilisés pour les identifier.

L'expression de nombreux gènes est modulée par des facteurs de transcription exprimés aussi périodiquement.

Les profils d'expression de ces facteurs de transcription sont modulés par les facteurs de transcription exprimés dans la phase précédente, et des modèles informatiques ont montré qu'un réseau CDK-autonome des facteurs de transcription est suffisant pour produire les oscillations dans l'expression de gènes.

Rôle dans la formation tumorale : Une déréglementation du cycle cellulaire peut conduire à la formation de tumeur. Certains gènes tels que les inhibiteurs du cycle cellulaire, RB, p53, etc., lorsqu'ils sont mutés, peuvent permettre à la cellule de se multiplier sans contrôle, provoquant alors l'apparition de tumeurs. Bien que la durée du cycle cellulaire dans les cellules tumorales soit égale ou supérieure au cycle cellulaire dans les cellules normales, la proportion des cellules qui se trouvent dans la division cellulaire activée (par rapport à des cellules quiescentes en phase G_0) dans les tumeurs est beaucoup plus élevée que dans les tissus normaux. L'ADN des cellules activées au cours du cycle cellulaire est ciblé dans le traitement du cancer car il est relativement exposé pendant la division cellulaire et donc susceptible d'être endommagé par des médicaments (intercalant de l'ADN) ou des radiations. La radiothérapie ou la chimiothérapie en suivant la procédure de réduction tumorale tuent les cellules qui sont récemment entrées dans le cycle cellulaire. En général, les cellules sont plus radiosensibles à la fin de M et de G_2, et plus résistantes à la fin de S. Le motif de la résistance

et de la sensibilité est en corrélation avec le niveau de composé de sulfhydrile dans la cellule. Les sulfhydriles sont les radio-protecteurs et tendent à être à leur plus haut niveau en phase S et à leur plus bas près de la phase M.

Le cycle cellulaire est contrôlé en plusieurs points de contrôle. Lorsque les cellules subissent un stress extracellulaire ou intracellulaire, ou les deux, ces points de contrôle, spécialement G_1/S et G_2/M qui sont contrôlés par un nombre de complexes composés des cyclines-CDKs et leurs régulateurs négatifs, incluant la famille Cip/Kip et la famille INK4a/ARF, sont activés. Le point G_1/S est la première surveillance du système pour arrêter la synthèse de l'ADN quand les cellules subissent un stress extracellulaire et il est un pas effectif de contrôle de la prolifération et l'apoptose cellulaire.

Le point de contrôle G_1/S est situé à la fin de phase G_1, juste avant la phase S, prend la décision importante de savoir si la cellule doit se diviser, retarder la division ou entrer en phase de repos. De nombreuses cellules s'arrêtent à ce stade et entrant en phase de repos appelée G_0. Les cellules du foie, par exemple, n'entrent uniquement en phase M qu'une ou deux fois par an.

Réseau de p53 : p53 (aussi appelé protéine 53 ou protéine tumorale 53) joue un rôle important en causant un contrôle mécanique aux deux points de contrôle G_1/S et G_2/M. La perte de contrôle de stabilité génomique est un élément central dans le développement du cancer, et p53, par la régulation des réponses normales au dommage de l'ADN et autres formes de stress gèno-toxique, est un élément très important dans le maintien de la stabilité génomique. Ainsi, il n'est pas surprenant que la protéine p53 fonctionnelle est perdue dans environ la moitié de tous les cancers humains. Qu'en est-il de l'autre moitié ? Une possibilité est que les mécanismes p53-indépendants de régulation ont été perdus. Une autre possibilité est que l'inactivation des voies p53-indépendantes peut arriver à n'importe lequel de plusieurs points différents et que la p53 elle-même n'est que la cible la plus courante. Par exemple, l'inhibiteur Mdm2 de p53 est surexprimé dans les tumeurs, indépendamment de la mutation de p53. Maintenant, vu les voies de signal qui arrivent à p53, en réponse à différentes formes de stress, et les voies de signal qui sortent de p53, déclenchés par p53 activée, il est clair que la p53 est le composant central d'un réseau complexe de voies de signalisation et que les autres composants de ces voies présentent les cibles alternatives pour l'inactivation.

- Signal d'entrée : La quantité de p53 augmente en réponse à une série de signaux, tels que le dommage de l'ADN, l'arrêt de la synthèse de l'ARN ou de l'ADN, ou la déplétion de nucléotides. Les mêmes stimuli activent également la p53, qui est souvent latente en l'absence de stress.

  Des évidences récentes ont également montré que la protéine Mdm2, qui se lie à p53, accélère la dégradation de p53, éventuellement par la voie d'ubiquitine. Le fait que le gène Mdm2 est une cible traductionnelle de p53 suggère une base moléculaire pour les protéines p53 mutantes. Par conséquent, la stabilité de ces protéines mutantes semble être due à leur incapacité à régler positivement l'expression de Mdm2, une protéine impliquée dans leur dégradation, plutôt qu'une propriété intrinsèque conférée par la résistance à la dégradation.

Une augmentation dans la transactivation de p53, sans augmentation dans le niveau de concentration de la protéine, a été trouvée dans les cellules traitées avec des doses basses de radiation d'UV, et la micro-injection d'un anticorps contre le domaine C-terminal stimule également la transcription p53-dépendente, même en l'absence de dommage à l'ADN. Plusieurs processus peuvent être impliqués dans l'activation de p53, ils incluent la phosphorylation, la glycosylation, la liaison aux protéines régulatrices, l'épissage alternatif, et l'acétylation.

Comment p53 « sent » les signaux ? Plusieurs protéines connues sont suspectées. La protéine kinase ADN-dépendante (DNAPK), un candidat plausible, se lie à et est activée par les extrémités cassées de l'ADN, elle peut phosphoryler des résidus 15 et 37 de p53 d'une manière ADN-dépendante. La phosphorylation de la serine 15 affecte la transactivation et les fonctions d'arrêt de croissance de p53 dans des cellules. Cependant, les cellules dépourvues de DNAPK ne montrent aucun défaut dans l'inhibition p53-médiée du cycle cellulaire, montrant que si DNAPK a un rôle dans la régulation de p53, d'autres composants doivent être en mesure de compenser sa perte.

De nombreuses protéines kinases ont été montrées pour phosphoryler p53 in vitro et sont des candidats pour être des régulateurs en amont. Cependant, très peu de données in vitro existent pour le rôle de la phosphorylation dans la régulation de p53. Les travaux récents qui montrent que p53 peut être acétylée in vitro sont intrigants et suggèrent la possibilité d'un mécanisme supplémentaire de régulation. Toutefois, il est nécessaire de montrer que l'acétylation se produit en réponse à un stress.

La poly (ADP-ribose) polymérase (PARP) a longtemps été connue pour avoir un rôle dans la reconnaissance de l'ADN endommagé et dans sa réparation. Les cellules PARP-KO de hamster chinois sont défectueuses dans l'activation de p53 et résistantes à l'apoptose induite par le dommage à l'ADN. Toutefois, des fibroblastes PARP-KO d'embryon des souris ont une réparation normale de l'ADN, et bien qu'il existe une diminution appréciable dans l'induction de p53 après le dommage de l'ADN ou la déplétion de nucléotide, il n'y a pas de changement dans l'activité de p53 dans les réponses cellulaires au stress. Par conséquent, bien que PARP soit impliquée dans l'augmentation de la quantité de p53 dans les fibroblastes de souris, d'autres voies de signalisation doivent être plus importantes dans l'activation de p53 en réponse aux dommages de l'ADN. La perte de l'ATM, le produit du gène de l'ataxie télangiectasie, ralentit l'induction de p53 en réponse à des ruptures de brins d'ADN causées par γ-radiation, mais pas en réponse à des dimères de pyrimidine causées par radiation d'UV. De même, p53 est induite normalement dans les cellules ATM-KO humaines après traitement avec N-(phosphonacétyl)-L-aspartate (PALA), qui bloque la biosynthèse de novo UMP, ou l'adriamycine, qui endommage l'ADN. La p53 et l'ATM peuvent à la fois être des composants de complexes qui fonctionnent dans la recombinaison. De même, le produit du gène impliqué dans le syndrome de la rupture de Nimègue (NBS) a également été placé en amont de p53 dans la voie qui répond aux rayonnements ionisants mais pas dans les réponses à d'autres agents endommageant l'ADN. Parce que les défauts dans l'induction de p53 dans les cellules ATM-KO sont

partiels ou sélectifs pour certains types de dommages de l'ADN, ces produits de gènes sont impliqués dans certains mais pas tous les signaux. Des knockouts doublés ou triplés doivent avoir un défaut plus profond dans l'induction de p53 en réponse aux dommages de l'ADN. Les défauts partiels similaires dans la signalisation de p53 ont été observés dans le syndrome de l'anémie de Fanconi (FAS) et le syndrome de Bloom (BLS) des fibroblastes, ceci suggère que de nombreuses voies régulent p53.

Récemment un rôle pour l'oncogène Ras et la voie des MAP kinases a été démontré dans la modulation et le fonctionnement de p53 dans les cellules humaines et de rongeurs. L'expression élevée de Ras ou l'activation de la voie de Mos/MAPK induit des niveaux de p53 de type sauvage et cause un arrêt de la croissance permanente, semblable à la sénescence cellulaire. Les cellules dépourvues de p53 peuvent tolérer des niveaux élevés de MAPK et affichent beaucoup d'arrêts p53-dépendants du cycle cellulaire et d'instabilité génomique. Dans une lignée de cellules défectueuses dans la voie des MAP kinases et dans l'expression de p53, une expression accrue de la MAP kinase, ERK2 restaure les niveaux normaux de p53, ce qui montre clairement que ERK2 est dans une voie qui régule le niveau d'expression de p53. Cette MAPK a été montrée pour phosphoryler les résidus 73 ou 83 de p53 in vitro, et cette phosphorylation peut être importante dans la stabilisation de la protéine. D'autres kinases, comme DNAPK II, cycline A-Cdc2, et cycline B-Cdc2, sont connues pour phosphoryler la protéine p53 in vitro et peuvent jouer un rôle dans la stabilisation de p53. Les mécanismes de l'induction de p53 en réponse aux types différents de stress sont encore largement méconnus.

- Signalisation post p53 : p53 est impliquée dans plusieurs aspects différents de l'arrêt du cycle cellulaire, l'apoptose, contrôle de l'intégrité du génome, et la réparation de l'ADN. Comment p53 régule de nombreux processus différents ? p53 est un tétramère qui peut se lier à des séquences spécifiques de l'ADN et donc transactiver de nombreux gènes. Plusieurs travaux ont montré que la p53 active est associée différemment sur les promoteurs, ceci entraîne la liaison et la transactivation différentielles de l'ADN. La p53 peut également inhiber l'expression de certains gènes. En outre, certains phénotypes p53-dépendents ne comportent pas de régulation de la transcription du tout.
- Contrôles du cycle cellulaire : La transition G_1/S – Les anticorps, qui reconnaissent l'extrémité terminale C de p53, empêchent des fibroblastes stimulés d'entrer dans la phase S. Ce résultat, à l'origine interprété comme une preuve qu'une fonction positive de p53 a été nécessaire, pose un paradoxe quand la surexpression de la p53 type-sauvage a été trouvée causant un arrêt de la croissance. Le paradoxe a été résolu quand on a constaté que ces anticorps inhibent plutôt que d'activer p53. Il est maintenant mieux compris que p53 est le médiateur pour l'arrêt en G_1 en réponse aux dommages de l'ADN causés par les UV ou la γ-radiation, les médicaments chimio-thérapeutiques ou la privation de nucléotides. La variabilité de type cellulaire dans l'arrêt p53-dépendant en G_1 est illustré par les études avec la γ-radiation, ce qui dans les fibroblastes diploïdes normaux provoque un arrêt long-terme p53-dépendant associé à l'induction prolongée de p21/Waf1. L'irréversibilité de cet arrêt dépend de

l'incapacité de ces cellules à réparer même un petit nombre de cassures double-brin de l'ADN, de sorte que le signal d'activation persiste. En revanche, la γ-irradiation de cellules HT1080, issues d'un fibrosarcome avec la p53 type-sauvage, provoque un arrêt en G_1 transitoire, tandis que la lignée tumorale colorectale RKO et la lignée tumorale mammaire MCF7, qui ont également une p53 type-sauvage, ne parviennent pas à arrêter en G1 après l'irradiation. Ces différences peuvent indiquer que la formation tumorale peut impliquer l'inactivation des composants en amont ou en aval de p53, ce qui provoque un échec de la réponse cellulaire aux dommages à l'ADN. Par exemple, la γ-irradiation active p53 pour activer la transcription de p21/Waf1, qui se lie à et inhibe les kinases cycline-dépendantes, provoque une hypo-phosphorylation de Rb, empêchant ainsi la libération de E2F et bloque la transition G_1/S. La modification d'un de ces composants en aval peut avoir un effet similaire à celui de l'inactivation de p53 lui-même dans la prévention de la voie de signalisation.

La p53 fuseau est impliquée dans un point de contrôle qui empêche la réplication de l'ADN lorsque le fuseau mitotique a été endommagé. Lorsque le contenu de l'ADN des fibroblastes d'embryons a été mesuré après le traitement avec du nocodazole ou autres inhibiteurs de l'assemblage des microtubules, il a été constaté que les fibroblastes normaux ont un contenu 4 N de l'ADN, tandis que les fibroblastes p53-KO atteignent des contenues de l'ADN de 8 ou 16 N. La destruction du fuseau peut bloquer la progression en mitose, ou la réplication peut être contrôlée en bloquant l'entrée à la phase S. Dans la lignée cellulaire murine avec p53 type-sauvage, la nocodazole cause un arrêt mitotique transitoire, à la suite de l'entrée en G_1 sans ségrégation des chromosomes.

Après la mitose, p53 est induit. La conclusion selon laquelle p53 induite en réponse aux dommages du fuseau bloque l'entrée à la phase S a été obtenue par l'analyse de la synthèse de l'ADN dans les fibroblastes exposées au nocodazole ou à la colcémide. Il est intéressant de noter que, les fibroblastes des p21/Waf1-KO de souris ne reproduisent pas leur ADN lorsqu'ils sont traités avec des poisons du fuseau. Par conséquent, p53 doit également utiliser le mécanisme p21-indépendent pour arrêter les cellules en G_1 et donc inhiber la réplication en réponse à des poisons du fuseau.

L'homéostasie du centrosome – Les fibroblastes p53-KO d'embryon de souris acquièrent plus de deux centrosomes, conduisant à la mitose avec plus de deux pôles du fuseau. p53 est associée aux centrosomes et donc peut affecter directement la duplication des centrosomes. Comme alternatif, la duplication impropre des centrosomes peut induire l'activation de p53, ceci peut à son tour causer l'arrêt dans G_2 ou G_1. Il est curieux que les MAP kinases et Cdc2, capables de phosphoryler p53, sont également liées aux centrosomes, et, comme p53, la MAP kinase est importante pour l'homéostasie de centrosomes [25].

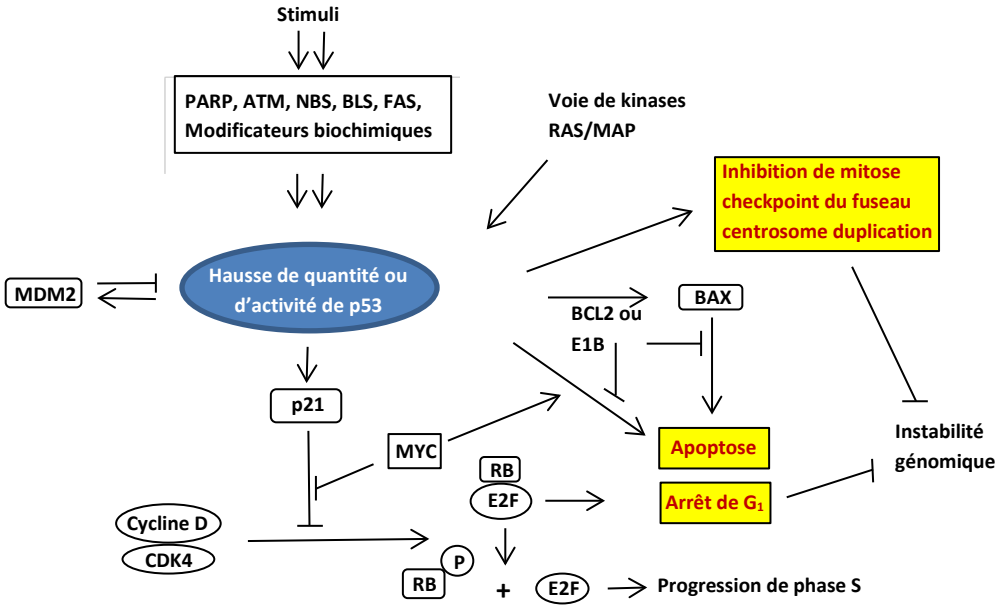

Figure 1.11. Composants de la voie de signalisation p53
(source : Munna et all 1998 [25])

p53 s'accumule et est modifiée et activée en réponse aux signaux générés par une variété de stresses génotoxiques. Plusieurs protéines, incluant ATM, PARP, FAS, BLS, et NBS sont impliquées dans l'activation, mais les voies qui conduisent à la modification sont largement inconnues. La voie RAS-MAP kinase est impliquée dans les niveaux de base de p53 et peut également affecter sa fonction. Certaines des fonctions cellulaires affectées par p53 peuvent être compromises par l'expression dérégulée de Myc, Bcl2, E1B, ou E2F. Le contrôle de l'activité de p53 comprend une boucle autorégulatrice liée à Mdm2. L'ensemble des voies p53-dépendentes aident à maintenir l'intégrité génomique par l'élimination des cellules endommagées, soit en les arrêtant de façon permanente ou par apoptose. p53 aide également à régler l'entrée en mitose, la formation du fuseau, et l'intégrité centrosome.

- Régulation de l'apoptose : p53 joue un rôle dans le déclenchement de l'apoptose dans certains types cellulaires, par exemple pour les cellules d'origine hématopoïétique. Les stimuli tels que les dommages à l'ADN, le retrait des facteurs de croissance, et l'expression de Myc ou E1A peuvent également causer l'apoptose p53-dépendente. La p53 doit être capable de fonctionner comme un facteur de transcription pour bloquer la transition G_1/S, mais l'apoptose ne requiert pas nécessairement l'activation de la transcription, car l'inhibition de la transcription par l'actinomycine D ou de la traduction par la cycloheximide n'affectent pas toujours l'apoptose p53-dépendente. En outre, les inhibiteurs de phosphatase induisent l'apoptose p53-dépendente en l'absence de transactivation. Cependant, les protéines pré-apoptotiques Bax et Igf-Bp3 sont des cibles de la transcription activée par p53, ceci suggère que la transactivation par p53 est importante dans l'induction de l'apoptose dans certaines circonstances. En outre, les anti-apoptotiques Bcl2 et la protéine adénovirus 19-kDa E1B peuvent prévenir l'apoptose p53-médiée. En réponse à un même stimulus p53 induit une apoptose dans certains types de cellules, mais un arrêt du cycle cellulaire dans les

autres. Plusieurs variables, telles que l'ampleur des dommages à l'ADN et les niveaux de p53, affectent également le choix entre l'arrêt du cycle cellulaire et l'apoptose. Également, le dialogue entre les voies de p53 et de Rb peut être important dans la détermination de ces réponses biologiques aux dommages à l'ADN. Par exemple, l'inactivation de Rb cause la perte de l'arrêt en G_1 et l'induction de l'apoptose après dommages à l'ADN. Ceci peut être expliqué par la libération de E2F, qui, lorsqu'il est surexprimé à lui seul peut induire l'apoptose. Par ailleurs, la surexpression de Rb bloque l'apoptose p53-médiée.

- Stabilité génomique : Le contrôle p53-dépendant du cycle cellulaire maintient l'intégrité génétique dans la population des cellules. L'amplification du gène est un modèle utilisé largement pour étudier l'intégrité génétique. Dans la plupart des cellules transformées ou immortalisées, des médicaments tels que PALA ou méthotrexate inhibent la synthèse de précurseurs nucléotidiques. La fonction de p53 est fréquemment perdue au cours du processus de tumorigenèse et dans l'immortalisation spontanée des cellules primaires, ceci indique que p53 peut être le facteur principal déterminant la permissivité pour l'amplification génique. En effet, les fibroblastes d'embryons de souris p53-KO sont permissives pour l'amplification génique, et des cellules humaines primaires de patients de Li-Fraumeni sont devenues permissives dès qu'elles ont perdu leur seul exemplaire de p53 de type sauvage. Quel signal est généré en tant que partie du mécanisme de l'amplification génique qui pourrait activer les voies p53-dépendantes et prévenir la propagation de cellules résistantes aux médicaments? Le modèle actuel de l'amplification implique, dans une première étape essentielle, de multiples cycles pont-cassé-fusions dans lesquels l'ADN cassé est formé tout au long d'une lignée entière des cellules enfants. L'importance des lésions de l'ADN dans la régulation des stades précoces de l'amplification génique a été démontrée avec des cellules REF52 transfectées avec un mutant de grand antigène T de SV40 sensible à la température. Lorsque ces cellules sont choisies avec PALA à la température basse, l'antigène T inactive p53, ceci rend les cellules permissives pour l'amplification génique. La restauration de p53 en inactivant l'antigène T à une température plus élevée très tôt dans le processus de formation des colonies PALA-résistantes arrête stablement toutes les cellules contenant l'ADN nouvellement amplifié. Des lignées cellulaires humaines peuvent acquérir une résistance à PALA par des mécanismes autres que l'amplification génique *in situ*, qui est de loin le mécanisme le plus commun dans les cellules de rongeurs. La plupart des colonies PALA-résistantes, provenant de plusieurs lignées cellulaires humaines, ne contiennent pas d'ADN. Cependant, dans les deux cas, les voies p53-dépendantes sont toujours impliquées. L'épuisement des nucléotides pyrimidiques causés par PALA génère un signal pour l'induction de p53 avant que tout endommagement de l'ADN ne se produise, en arrêtant les cellules et inhibant la formation des colonies PALA-résistantes.

Réseau de Chk2 : Pommier et al ont montré le rôle de Chk2 (et de Chk1 aussi) dans les cellules. Les agents ciblant l'ADN ont été utilisés dans la chimiothérapie du cancer, et peuvent guérir les patients atteints de leucémies, lymphomes, et cancers des testicules. Dans de nombreux autres cancers, les réponses à ces thérapies sont limitées et la rechute est

courante. Une approche consiste à rechercher les déterminants moléculaires de réponse tumorale et à extraire les paramètres moléculaires à utiliser pour prédire la réponse tumorale. Les paramètres moléculaires comprennent le point de contrôle du cycle cellulaire, les voies de la réparation de l'ADN et de l'apoptose.

Les points de contrôle cellulaires sont des voies moléculaires, qui sont activées en réponse aux dommages à l'ADN. L'Activation des points de contrôle peut tuer les cellules endommagées par apoptose ou arrêter la progression du cycle cellulaire pour réparer l'ADN avant la reproduction/division cellulaire. Deux voies bien connues sont les voies ATM-Chk2 et ATR-Chk1. La voie ATM-Chk2 est principalement activée par la cassure double-brin de l'ADN (DSB) produite par les rayonnements ionisants (IR) ou les agents endommageant de l'ADN tels que la bléomycine et la camptotécine. La voie ATR-Chk1 apparaît dans le cas de la DSB réplication-médiée, à côté de la voie ATM-Chk2. Les deux voies ATM-Chk2 et ATR-Chk1 semblent fonctionner comme des cascades de transduction parallèles des dommages à l'ADN. Il existe cependant un dialogue entre les deux voies, ATM peut également phosphoryler Chk1 en réponse à IR, et ATR peut phosphoryler Chk2 en réponse aux dommages réplication-associés. ATM et ATR appartiennent à la famille des kinases phosphatidylinositol-3 (PI3K), qui contient également la kinase ADN-dépendante (DNA-PK) et ATX. La fonction des PI3K est de former des grands complexes multi-protéiques dans le voisinage des dommages à l'ADN. L'Activation de la famille PI3K conduit à la phosphorylation/activation directe des effecteurs du point de contrôle (Cdc25A et Cdc25C, p53, BRCA1), et indirectement par la phosphorylation/activation des kinases Chk1 et Chk2, qui, à leur tour, phosphorylent les effecteurs du point de contrôle. Les Chk1 et Chk2 ont des rôles redondants partiellement pour l'arrêt/point de contrôle du cycle cellulaire, la réparation de l'ADN, et la transcription de l'ADN, car elles partagent une gamme de substrats, incluant p53, Cdc25A et Cdc25C. Pour la Chk2 humaine, le domaine SQ/TQ est dans la partie N-Terminale, et plus spécialement, T68 est une cible de PI3K's. La phosphorylation de T68 est une condition préalable pour l'autophosphorylation ultérieure sur T383 et T387, et l'activation de Chk2. Le domaine FHA fonctionne comme des sites de liaison pour les protéines phospho-thréonines (pT)-contenant, y compris les substrats de Chk2 aussi bien que Chk2 T68-phosphorylée. Le domaine de kinase catalytique (résidu 226-486) contient une boucle de l'activation autour de l'autophosphorylation/activation de résidus T383/387. Deux mutations, D347A ou D368N induisent une Chk2 kinase-morte. Le signal nucléaire de localisation (NLS) de Chk2 se trouve vers le C-Terminus. La phosphorylation de la Sérine 516 est impliquée dans l'apoptose en réponse aux dommages de l'ADN (DSB). Ainsi, DSB active Chk2, qui à son tour phosphoryle les substrats dont la phosphorylation réglemente des points de contrôle, qui soit tuent les cellules par l'apoptose ou arrêtent le cycle cellulaire et améliorent la survie cellulaire en laissant du temps pour la réparation.

- Activation de Chk2 par les dommages à l'ADN : En réponse aux DSB, Chk2 est phosphorylée par ATM (et ATR aussi) et DNA-PK, qui à son tour active Chk2 soit directement par phosphorylation de T68, soit indirectement lorsque ATM phosphoryle/active la Plk3. La phosphorylation de T68 est une condition préalable pour l'activation de Chk2. Elle est essentielle pour la dimérisation (une T68 phosphorylée d'une Chk2 fait l'interaction avec le domaine FHA d'autre), qui autorise

la trans-phosphorylation intermoléculaire de Chk2 sur T383/387 dans la boucle d'activation du domaine kinase et la cis-phosphorylation sur S516. La phosphorylation est nécessaire à l'activation complète de Chk2 vers ses substrats hétérologues. Le domaine FHA de Chk2 est essentiel pour que la mutation I157T diminue dans la dimérisation et l'activation de Chk2. La phosphorylation ATM-dépendante ne peut pas arriver librement dans le nucléoplasme, mais nécessite des complexes de protéines adaptatrices spécifiques DSB-associées contenant des motifs BRTC. Ainsi, la phosphorylation ATM-dépendante de Chk2 nécessite Nbs1 intacte, Brca1, la réparation fonctionnelle de mésappariement, et est stimulée par la liaison à la protéine liée, 53BP1. Aussi, la protéine partenaire Chk2-liée MDC1 est nécessaire pour les réponses Chk2-médiées aux dommages de l'ADN. MDC1, comme Chk2, contient un domaine FHA, et est phosphorylée d'une manière ATM/Chk2-dépendante. La suppression de l'induction de MDC1 entraîne la défection du point de contrôle de phase S. MDC1 est trouvée sur les sites de cassures de l'ADN, et associée à Chk2 T68-phosphorylée après dommages à l'ADN. Chk2 active ses cibles en aval, dont sept sont actuellement connues. Trois sont impliquées dans l'apoptose : PML, p53 et E2F1. Deux sont impliquées dans l'arrêt au point de contrôle du cycle cellulaire : Cdc25A et Cdc25C. Brca1 règle probablement la structure de la chromatine, l'induction de la réparation de l'ADN. La cible de Chk2 la plus récemment identifiée est la phosphatase-2A (PP2A), qui à son tour peut inactiver Chk2 par la déphosphorylation de T68.

- Cibles en aval de Chk2 impliquées dans l'apoptose : Chk2 peut régler l'apoptose par phosphorylation de PML. Les polymères PML sont normalement concentrés dans les corps nucléaires qui contiennent au moins 30 protéines différentes impliquées dans la réparation de l'ADN, le point de contrôle du cycle cellulaire, la dégradation de protéine et l'apoptose telles que p53, Nbs1, BLM, pRb, Dax. La deuxième cible de Chk2 impliquée directement dans l'apoptose est p53. Chk2 phosphoryle p53 sur la S20, la phosphorylation active p53 par prévention de la liaison de Mdm2 à p53, et donc la dégradation Mdm2-médiée de p53. Chk2 peut également induire l'activité de transcription de p53 par dissociation de Mdm2 à p53. La liaison de Chk2 à 53BP1 (un régulateur de transcription de p53), peut également contribuer à l'activation de la transcription de p53. Chk1 et Jnk peuvent également phosphoryler p53, ce qui explique que le knockout de Chk2 ne prévient pas la phosphorylation IR-inductive de p53 sur S20. Chk2 phosphoryle aussi E2F1. Un ensemble des gènes impliqués dans l'apoptose sont transactivés par la sur-induction d'E2F1, sont inclus INK4a/ARF, p73, Apaf-1, Noxa at PUMA. La phosphorylation d'E2F1 sur S364 par Chk2 augmente sa demi-vie en bloquant sa dégradation, qui améliore l'activité de transcription d'E2F1. E2F1 est également phosphorylée sur S31 et activée directement par ATM (et ATR) en réponse aux dommages de l'ADN. Les dommages à l'ADN peuvent diriger E2F1 sur ses gènes cibles apoptotiques.
- Cibles en aval de Chk2 impliquées dans l'arrêt du cycle cellulaire : Chk2 peut inactiver les kinases cycline-dépendantes et induire l'arrêt du cycle cellulaire par phosphorylation et donc inactivation de Cdc25A et Cdc25C, deux des trois kinases de Cdc25. La phosphorylation de Cdc25A induit sa dégradation, et bloque les cellules à

la frontière G_1/S. Les deux Chk1 et Chk2 peuvent effectivement phosphoryler Cdc25A sur S123. La phosphorylation qui en résulte conduit à l'ubiquitination et la dégradation rapide de Cdc25A par le protéasome. La phosphorylation Chk2-médiée de Cdc25C sur S216 inactive Cdc25C par induction de sa liaison à la protéine 14-3-3 et son exportation nucléaire. Donc, l'inactivation de Cdc25C prévient l'activation de Cdk1/cycline A et Cdk1/cycline B, et donc induit l'arrêt G_2/M. La phosphorylation/l'activation de p53 peut également fonctionner comme une réponse d'arrêt du cycle cellulaire G_1 retardée par induction de la transcription de p21 et de Gadd45.

- Rôle possible de Chk2 en réparation de l'ADN : Aucun rôle direct de Chk2 en réparation de l'ADN a encore été montré. Cependant, il y a des preuves de relier l'hyper-phosphorylation Chk2-médiée de BRCA1 au processus qui règle la réparation de DSB. BRCA1 est un gène vital de suppresseur tumoral dont l'inactivation augmente le risque de cancer du sein et de l'ovaire. Les cellules déficientes en BRCA1 présentent une hypersensibilité aux agents d'IR et d'ADN-réticulation. La phosphorylation de BRCA1 induit la dissociation de BRCA1 de Chk2, une situation appelée interaction Chk2/PML. BRCA1 libre pourrait alors fonctionner en réparation de l'ADN et en régulation de la transcription. L'évidence récente suggère que l'inhibition de la phosphorylation Chk2-médiée via la mutation de S988 sur BRCA1 prévienne la régulation positive BRCA1-dépendante de la recombinaison homologue (HR) et la régulation négative d'end-jonction non-homologue (NHEJ). En vertu de la phosphorylation de p53, Chk2 est potentiellement capable de régler un certain nombre d'événements en aval qui sont réglementés par l'activité transcriptionnelle de p53 et impliqués en réparation. Une preuve supplémentaire que Chk2 pourrait fonctionner en réparation de l'ADN provient de l'interaction de Chk2 avec Msh2, qui est impliquée dans la réparation de décalage, et avec Mus81 qui est impliquée dans la réparation de DSB.
- Autres substrats potentiels de Chk2 : Deux substrats potentiels supplémentaires de Chk2 sont la Phospho-kinase-3 (Plk3) et Mdm2. Plk3 forme une boucle d'activation positive avec Chk2, les deux kinases s'activent l'une l'autre. Par ailleurs, elles sont toutes les deux activées par ATM, et elles phosphorylent Cdc25C et p53. La phosphorylation de Mdm2 soit par Chk2, soit par Chk1 pourrait activer p53 en inactivant l'interaction p53-Mdm2, qui fonctionne comme un antagoniste de p53.

1.1.2. Inhibiteurs de Chk2 comme nouveaux agents anticancéreux

Pommier et al ont trouvé que l'inhibition de l'activité de p53 a amélioré la réponse apoptotique des cellules p53-défectueuses à l'IR. Ainsi, il semble rationnel de proposer l'utilisation des inhibiteurs de Chk2 pour les tumeurs p53-déficients.

1.1.3. Problématique

La régulation du cycle cellulaire est un processus décidant de la survie d'une cellule, contenant la détection et la réparation de dommage génétique ainsi que la prévention de la division cellulaire incontrôlée. Les événements moléculaires contrôlant le cycle cellulaire sont

ordonnés et directionnels. C'est à dire, chaque processus se déroule de manière séquentielle et il est impossible d'inverser le cycle.

Deux classes principales de molécules, cyclines et kinases cycline-dépendantes (CDKs), déterminent un progrès cellulaire dans le cycle. Un grand nombre de gènes codant pour les cyclines et CDKs sont conservés chez tous les eucaryotes, mais en général, les organismes plus complexes ont des systèmes de contrôle du cycle cellulaire plus élaborés qui intègrent des composants individuels.

1.1.4. Exigences

De nombreuses études se sont penchées sur la construction des réseaux de gènes grâce à des méthodes mathématiques ou logiques (Tomshine & Kaznessis, 2006 ; Brun et al, 2004). Le logiciel SOLAR [14] est un logiciel développé au Japon (Nabeshima et al, 2003), utilisant de la programmation logique et employé pour effectuer de la conséquence finding, c'est à dire trouver des conséquences logiques à partir d'un jeu de données. La logique basée sur l'abduction fait de ce logiciel un outil de choix pour une telle étude. D'autre part, c'est aussi et surtout parce que Katsumi Inoue a participé à son élaboration que le projet lui donne un rôle prépondérant. SOLAR a déjà été utilisé en biologie des systèmes, notamment sur p53 (Inoue et al, 2010), mais les résultats n'étaient pas aussi probants qu'on aurait pu espérer : un des objectifs de travail devait alors inclure une meilleure rigueur dans les données afin d'obtenir des résultats biologiquement significatifs.

1.2. Cas d'application

1.2.1. Contexte des gènes et leurs interactions

Le modèle biologique adopté a suivi celui indiqué dans la carte de Pommier [24], et a été complété par des informations trouvées dans la littérature. Dans la carte, les symboles et conventions sont définis ci-dessous

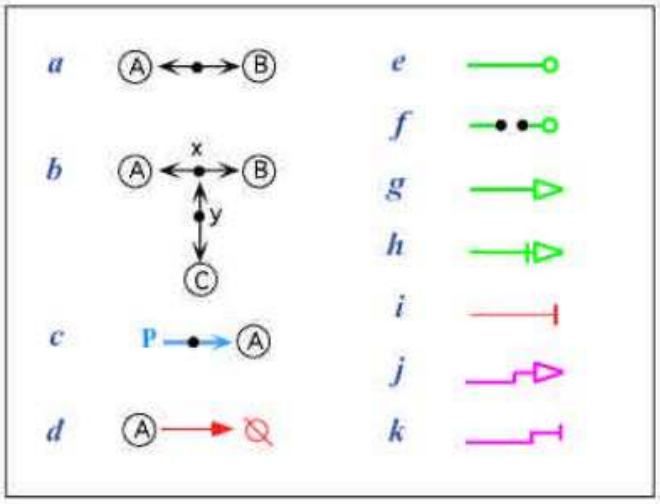

Figure 1.12. Définitions des symboles et des conventions de la carte
Couleurs : Noir – interaction de liaison ; bleu – modification de protéine ; vert – processus enzymatiques ou autres stimulants ; rouge – inhibitions ou autres effets négatifs ; pourpre – activation de transcription ou répression.

Symboles : (a) Protéines A et B peuvent se lier, le nœud représente le complexe A:B ; (b) Complexes multimoléculaires ; (c) Modification covalente de A ; (d) Dégradation de A ; (e) Stimulation enzymatique en transcription ; (f) Stimulation enzymatique en trans; plus spécifiquement autophosphorylation en trans ; (g) Symbole général de stimulation; (h) Une barre derrière la tête de flèche signifie la nécessité ; (i) Symbole général d'inhibition ; (j) Symbole de raccourci pour l'activation de transcription ; (k) Symbole de raccourci pour l'inhibition de transcription

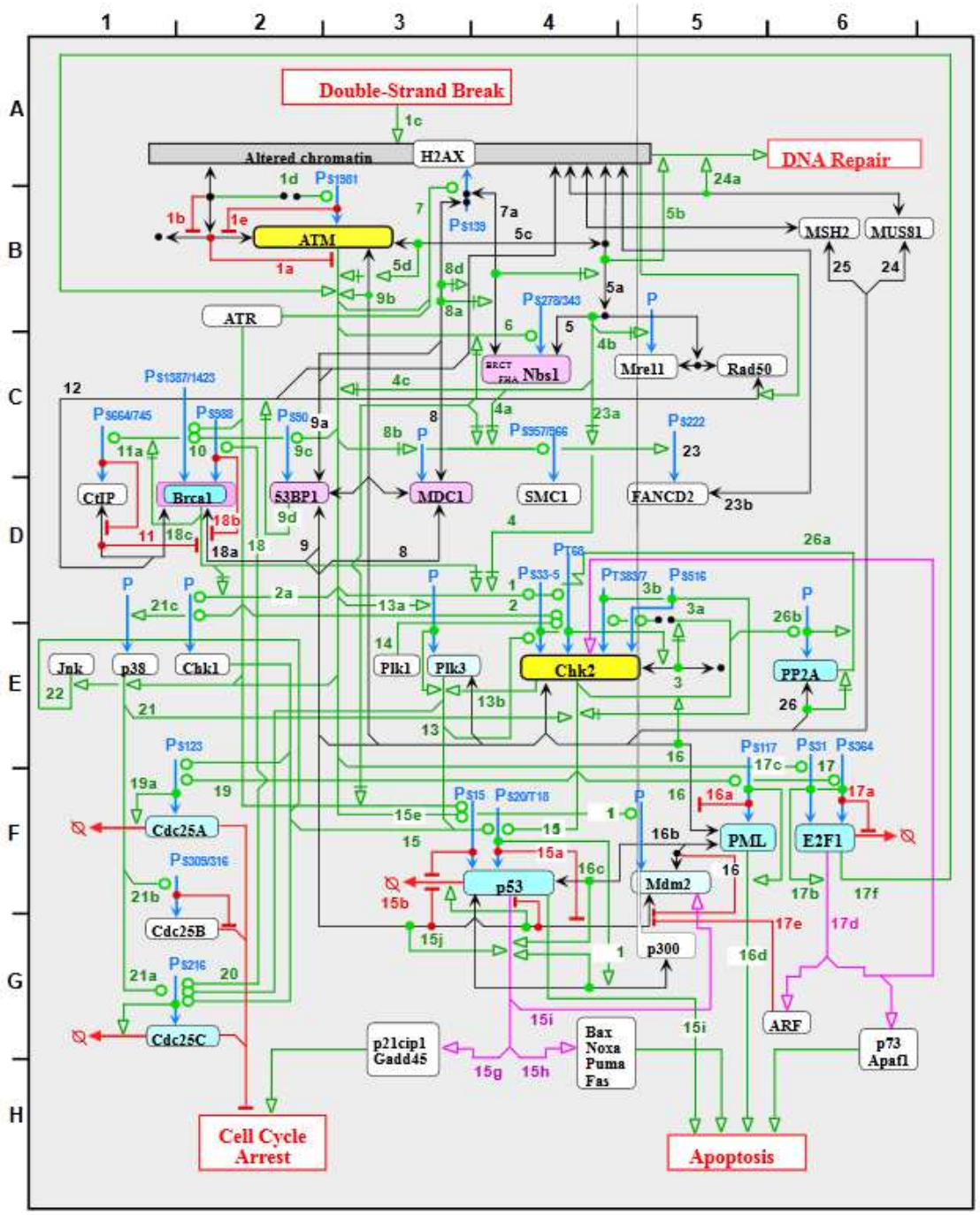

Figure 1.13. La carte d'interaction de la réponse ATM-dépendante de la cassure double-brin (source : Pommier et al, 2005[24])

Explication de la carte :

#1.D4 : Chk2 est activé suite à la phosphorylation par ATM en réponse à DSB (en anglais : double-strand break) et la réplication blocus menant à DSB de réplication-associée

#2.D4 : ATR phosphoryle et active Chk2 en réponse à UV (ultraviolet) et réplication des blocs mais pas IR (en anglais : ionizing radiation). Ainsi, Chk2 peut être activé par des voies ATM indépendants

#3.D5 : Chk2 est activé par homodimérisation et phosphorylé par des kinases PI3

#4.D4 : Nbs1 est nécessaire pour la phosphorylation ATM-dépendante et l'activation de Chk2.

#5.C4 : Chaque molécule Nbs1 recrute un dimère de Mre11 et Rad50 aux foyers DSB, où le complexe nucléase Mre11-Rad50 répare le DSB

#6.B4: Nbs1 est phosphorylée au S343 par ATM

#7.B3: H2AX est phosphorylée au S139 (γH2AX) par ATM en réponse à DSB et par ATR et DNA-PK en réponse à la réplication-médiation DSB

#8.D3: MDC1 forment les foyers nucléaires avec γ-H2AX et Chk2 T68-phosphorylée en réponse à l'IR.

#9.D3: 53BP1, une autre protéine BRCT-contenant, améliore également la phosphorylation à T68 de Chk2

#10.C2: BRCA1 est phosphorylée par ATM sur plusieurs sites

#11.D1: ATM phosphoryle CtIP sur S664 et S745, inhibant ainsi la liaison de CtIP à BRCA1

#12.C1: BRCA1 se lie à Rad50 en réponse à DSB et co-localise avec les foyers Mre11-Rad50-Nbs1 suivante l'IR

#13.E4: Polo kinase 3 (PLK3), qui, comme Chk2, est présente pendant tout le cycle cellulaire, est phosphorylée et activée dans les minutes suivant IR et MMS de manière ATM-dépendante

#14.D3: Kinase Polo-like (Plk1) phosphoryle Chk2 (à T68), co-IP avec Chk2, et co-localise avec Chk2 à centrosomes

#15.F4: Chk2 phosphoryle p53 sur T18 et S20, ceci empêche la liaison de p53-Mdm2 et de la dégradation de p53

#16.E5: Chk2 phosphoryle PML sur S117, qui se dissocie PML de Chk2 et active l'apoptose PML-médiation

#17.F6: Chk2 phosphoryle E2F1 dans les cellules traitées avec de l'étoposide

#18.C2: Chk2 phosphoryle BRCA1 au S988

#19.F2: La phosphorylation de Cdc25A par Chk2 et ATM au S123 induire la destruction de Cdc25A

#20.G2: La phosphorylation de Cdc25C par Chk2 et p38 MAPK inactive Cdc25C

#21.D1: ATM est requis pour l'activation de p38-gamma, ce qui conduit à l'activation de Chk2, et est suffisante pour activer un arrêt G2-M en réponse à IR

#22.E1: Jnk est activé par UV, et peut activer Cdc25C et inactiver p53 par la phosphorylation S20

#23.C5: La phosphorylation de FANCD2 en réponse à l'IR au S222 est ATM-dépendante (directe ou indirecte ne sait pas) et nécessite la phosphorylation au S343 de Nbs1

#24.D6: Mus81 se lie spécifiquement au domaine FHA de Chk2 et de Mus81

#25.D6: Msh2 lie à Chk2, et MLH1 lie à l'ATM

#26.E6: La protéine phosphatase 2A (PP2A) se lie à Chk2 et déphosphoryle T68, inactivant ainsi Chk2

1.2.2. Définitions

Quelques définitions sont données sur le langage de programmation utilisé dans la thèse : Prolog et logiciel SWI-Prolog.

Le langage a en premier été conçu par un groupe autour d'Alain Colmerauer à Marseille, France, dans les années 1970. Il a été un des premiers langages de programmation logique, et reste le plus populaire parmi ces langages aujourd'hui, avec beaucoup d'implémentations gratuites et commerciales disponibles, l'une d'elles est SWI Prolog.

Prolog (PROgramming in LOGic) est un langage de programmation de but général. Il est aussi appelé le langage de programmation de notation (symbolic programming) comme les langages de programmation fonctionnelle (functional programming) ou de programmation non-numérique (non-numerical programming). Ce langage de programmation est associé avec l'intelligence artificielle et la linguistique computationnelle. Il a ses racines en logique du premier ordre, une logique formelle, et contrairement à de nombreux autres langages de programmation. Prolog est déclaratif : le programme logique est exprimé en termes de relations : il est bien adapté pour résoudre les problèmes liés à des objets et des relations entre eux. Le principe de la programmation logique est basé sur les clauses de Horn. Une clause est appelée une clause de Horn si et seulement si elle a au plus un littéral positif. Par exemple, pour dire « Tous les hommes doivent mourir » et « Socrate est un homme », on peut déclarer:

Clause 1 : mourir(X) ← homme(X)

Clause 2 : homme(Socrate)

Ce sont des clauses de Horn. La première clause est une règle, la deuxième est un fait. Un programme Prolog peut être considéré comme une base des données qui contient des clauses de Horn en forme de faits et/ou de règles. L'utilisateur peut exécuter un programme Prolog en posant des questions sur la base des données, par exemple : est-ce que Socrate doit mourir « mourir(Socrate) »? Le système va exécuter le programme en raisonnant – cherchant

sur la connaissance – la base des données, pour donner une réponse qu'elle soit « vraie » ou « fausse ». Pour l'exemple ci-dessus, le système va répondre à la question « Est-ce que mourir(Socrate) est vrai ?» en répondant à la question « Est-ce que « homme(Socrate) est vrai ?» (Clause 1). La réponse est « oui » (Clause 2), alors mourir(Socrate) est vrai. Un autre exemple plus compliqué, *parent(Tom, Bob)* signifie que Tom est parent de Bob. C'est un fait, dans lequel *parent* est un prédicat, Tom et Bob sont des arguments. Si on a *parent(Tom, Bob)* et *parent(Bob, Liz)*, alors on a *grandparent(Tom, Liz)*, signifie que Tom est grand parent de Liz. Généralement, *X* est grand parent de *Z* si et seulement s'il existe un *Y*, tel que *X* est parent de *Y* et *Y* est parent de *Z*. On a une règle comme *grandparent(X, Z)* ← *parent(X, Y)* ∧ *parent(Y, Z)* qui signifie que si *X* est parent de *Y* et *Y* est parent de *Z*, alors *X* est grand parent de *Z*. Dans Prolog, la règle est représentée par *grandparent(X, Z)* :- *parent(X, Y), parent(Y, Z)*. Ensuite, une relation plus compliquée est définie, c'est l'ancêtre. *X* est ancêtre de *Y* si *X* est parent de *Y*, ou grand parent de *Y*, ou parent de grand parent de *Y*, etc. De façon récurrente, la relation peut être définie comme ça : *X* est ancêtre de *Y* si et seulement si (*X* est parent de *Y* ou *X* est parent de *Z* qui est ancêtre de *Y*). Dans Prolog, ceci est représenté par deux clauses consécutives :

ancestor(X,Y) :- *parent(X,Y)*

et

ancestor(X,Y) :- parent(X,Z), ancestor(Z, Y).

Pour écrire et exécuter des programmes, on utilise l'implémentation gratuite SWI-Prolog. SWI-Prolog appartient à la famille Prolog Edinburgh, il est en développement continu depuis 1987. Son principal auteur est Jan Wielemaker. Le nom est dérivé de SWI : Sociaal-Wetenschappelijke Informatica (Social Science Informatique), le nom ancien du groupe de l'Université d'Amsterdam, où Wielemaker est employé. SWI-Prolog dispose d'une bibliothèque des prédicats et de guidage riche. Il fonctionne dans l'environnement graphique d'objet orienté. SWI-Prolog est assez simple à utiliser en exploitant les caractéristiques interactives graphiques.

Après le téléchargement et l'installation de SWI-Prolog en ligne : http://www.swi-prolog.org/, et l'exécution de swipl-win.exe, la fenêtre de travail apparaît pour déposer des questions. SWI-Prolog fonctionne en mode interactif : l'utilisateur dépose des questions et le système répond. L'invite de commande de SWI-Prolog est un point d'interrogation avec un tiret, l'ordre indiqué pour surveiller le processus de travail de l'utilisateur, puis un curseur clignotant.

Après qu'un programme de Prolog soit compilé et chargé en mémoire, l'utilisateur peut déposer des questions (finies par un point). En fonction de la question, le système répond vrai (true) ou faux (false) avec une valeur X = < value > s'il y a une variable X quelconque dans la question. Dans le cas où il y a plusieurs réponses, après la première réponse, l'utilisateur peut taper un point-virgule s'il veut que le système continue à donner d'autres réponses, jusqu'à ce que le système réponde false (c'est à dire, il n'y a plus de réponse). Ou bien l'utilisateur peut

arrêter en tapant « entrée ». L'utilisateur peut également recevoir un message d'erreur s'il y a des problèmes dans la question déposée.

Retourner au problème principal. Ayant pour base la carte de Pommier, le choix du prédicat le plus simple a été de se baser sur les types d'interactions déjà existantes sur cette carte : *stimulation* (une réaction enzymatique générale), *phosphorylation*, *autophosphorylation*, *inhibition*, *nécessité* (désigne une interaction nécessaire en tant que condition prérequis pour qu'une autre interaction soit réalisable), *liaison*, *activation* de *transcription*, *dégradation* et *déphosphorylation*.

Le deuxième choix de stratégie est de vouloir représenter les interactions de la voie de signalisation de façon linéaire et chronologique, c'est à dire de vouloir créer un modèle dans lequel une interaction peut être issue d'une autre interaction, et va probablement entraîner une autre encore. Il s'agit alors de trouver une façon de représenter une succession chronologique des interactions. En effet, la carte d'interaction de Pommier ne montre que les interactions en elles-mêmes, de manière générale, sans distinction quant à leur ordre d'apparition dans la cascade d'événements. Le choix est alors d'inclure le facteur temps dans le modèle d'une manière indirecte, sur les conséquences de la succession des événements dans les états différents des protéines. En connaissant leur ordre chronologique d'apparition, on peut connaître l'évolution des événements en transposant chacun de ces états dans le modèle logique [22]. Par exemple : si 'ATM → Chk2 → p53 → Apoptose' est lu dans la carte, après descriptions on a alors 'ATM/ATM (se phosphoryle) → $_{p}$ATM (ATM phosphorylé) → Chk2 (phosphoryle) → $_{p}$Chk2 (Chk2 phosphorylé) → Chk2/Chk2 (se lie) → p53 (phosphoryle) → $_{p}$p53 (p53 phosphorylé) → Transcription → Apoptose'

L'étape suivante dans la réalisation de la thèse est de construire un modèle contenant les interactions présentes dans la voie de signalisation en réponse à la cassure double brin de l'ADN. Ce modèle doit avant tout être un modèle biologique avant d'être un modèle logique, car il faut dans un premier temps lister les interactions avant de les implémenter dans Prolog. La nécessité de définir un modèle biologique a entraîné de faire des choix dans les informations apportées à ce modèle. Pour pouvoir établir ce modèle, il faut une compréhension de la voie de réponse à la cassure double brin, or à l'heure actuelle, elle n'est pas encore bien connue. Des informations certaines restant encore à découvrir.

Pour passer du modèle biologique au modèle logique, il faut considérer plusieurs contraintes : comment modéliser les interactions, bien faire attention au sens des implications logiques, respecter l'ordre chronologique préalablement défini et vérifier la cohérence des informations. Mais l'étape la plus importante à ce niveau d'avancement a été de bien définir les prédicats. Les prédicats choisis ont été calqués sur ceux de la carte de Pommier, à savoir *stimulation*, *phosphorylation*, *autophosphorylation*, *inhibition*, *nécessité*, *liaison*, *activation* de transcription, *dégradation* et *déphosphorylation*.

Au départ, les prédicats ont été conceptualisés, pour la majorité d'entre eux, de la façon suivante : *produit* ← *réaction(enzyme, substrat)*. La *réaction* peut être *stimulation*, *phosphorylation*, *déphosphorylation*, *liaison*, *activation* ou *dégradation*. Par exemple, $_{p*}$-Y ←

phosphorylation(X, Y). Les autres prédicats qui ne sont pas de type *produit* ← *réaction(enzyme, substrat)* sont modélisés séparément :

$_{p*}$-Y ← *autophosphorylation(Y)*

¬Y ← *inhibition(X, Y)* : si *X* est vrai, alors *Y* ne peut pas l'être

Y ← *nécessité(X, Y)* : pour que *Y* est vrai, *X* doit être vrai

En effet, certains prédicats ont été modifiés, ou bien supprimés, au cours d'implémentations, tandis que d'autres ont été rajoutés grâce aux mises à jours effectuées dans le modèle biologique et avec l'avancement de son implémentation dans Prolog. Par exemple, les prédicats ajoutés sont *ubiquitination*, *méthylation* et *dissociation*.

Une fois l'inventaire des réactions réalisé, un modèle biologique adopté, les prédicats choisis et réactualisés, il faut implémenter les informations dans Prolog. Pour la thèse, le logiciel SWI-Prolog a été utilisé. Ce programme utilise un fichier qui doit posséder une syntaxe propre. En effet, le fichier doit contenir des informations en forme de clauses, plus précisément des clauses en Forme Normale Conjonctive (CNF). Une CNF est une conjonction de clauses, de laquelle chaque clause est une disjonction de littéraux.

Exemple : Soient *A, B, C,* et *D* des littéraux, contenus dans les clauses *(A ∨ B)* et *(C ∨ D)*. Une CNF possible pourra alors être *[(A ∨ B) ∧ (C ∨ D)]*.

1.2.3. Règles de comportements

Il y a des règles de comportement entre les protéines dans le réseau de gènes. Par exemple, s'il y a une relation causale directe d'une protéine *A* à une protéine *B*, elle est définie que si *cause(A, B)* est vrai. En autres mots, si *A* cause *B*, et si *A* est vrai, alors *B* est vrai. Le fait qu'il n'y a pas de relation causale directe d'une protéine *A* à une protéine *B* est représenté par un contraire de la forme ¬*cause(A, B)*. Similairement, s'il y a une relation de blocage directe d'une protéine *A* à une protéine *B*, elle est définie que *bloque(A, B)* est vrai, etc. Le prédicat *bloque* peut être expliqué ci-dessous : si *A* bloque *B*, est si *A* est vrai, alors *B* est faux.

Dans ce contexte, pour donner les liens entre les relations *cause* et *bloque*, la logique classique est utilisé. La solution élémentaire est alors de donner explicitement les deux règles de comportements [3].

Règle 1 : *cause(A, B) ∧ cause(B, C) → cause(A, C)*

Règle 2 : *cause(A, B) ∧ bloque(B, C) → bloque(A, C)*

La règle 1 veut dire que si *A* cause *B*, et si *B* cause *C*, alors *A* cause *C*. Similairement, la règle 2 veut dire que si *A* cause *B*, et si *B* bloque *C*, alors *A* bloque *C*.

Chapitre 2. Conclusion conséquente et la production

2.1. Conclusion conséquente et SOL

2.1.1. Conclusion conséquente

La conclusion conséquente est la tâche de calcul des théorèmes entraînés par un ensemble d'axiomes [10]. Son importance réside dans le fait que les tâches de raisonnement sont des cas particuliers de conclusions conséquentes [11]. Par exemple, la démonstration d'un théorème est équivalente à démontrer que la clause vide est une conséquence logique des axiomes et de la négation du théorème à démontrer. Un autre exemple est le raisonnement abductif, qui consiste à construire des hypothèses, lorsqu'est ajoutée à une théorie donnée une série d'observations [16]. Mais la génération des hypothèses par raisonnement abductif est équivalente à nier les conséquences de la théorie et de la négation des observations. De cette façon, la conclusion conséquente fournit un cadre général pour résoudre une variété de problèmes de raisonnement [17].

La plupart des applications de conclusion conséquente exigent les théorèmes générés pour satisfaire certaines propriétés syntaxiques. Par exemple, dans la démonstration de théorèmes, la seule conséquence d'intérêt est la clause vide, tandis que, dans l'abduction, les conséquences doivent en général être broyant des clauses unitaires dont les prédicats sont parmi un ensemble d'abductibles. En outre, pour éviter de produire un nombre infini de clauses redondantes et intérêt, il est généralement nécessaire de générer uniquement les conséquences qui ne sont pas englobées par d'autres [18,19].

2.1.2. SOL

G. Bossu et P. Siegel ont présenté un algorithme basé sur la SL-résolution pour implémenter une logique non-monotone, la sub-implication [29,30]. Cet algorithme est basé sur la génération de clauses caractéristiques. En [26] les clauses caractéristiques sont généralisées par P. Siegel pour définir les champs de production et leurs propriétés [26,31]. La thèse de 1987 [26] donne aussi un algorithme de production qui est une méthode complète pour calculer toutes les clauses impliquées par un ensemble de clauses et appartenant à un champ de production. Ces travaux ont été repris par Inoue [12] pour proposer la SOL-résolution comme une méthode complète pour trouver les clauses caractéristiques d'une théorie. Ensuite, Iwanuma et al. [14] ont reformulé la SOL-résolution dans le cadre des tableaux de la connexion et proposé plusieurs méthodes de taille pour élaguer des branches redondantes de l'espace de recherche.

Tableaux de la connexion : Dans la logique du premier ordre, une clause est un multi-ensemble de littéraux, écrite comme une disjonction $L_1 \vee \ldots \vee L_n$. La clause vide est notée $\varnothing$. Un littéral L est *général maximum*, si les arguments de L sont des variables distinctes. Par exemple, *p(X, Y, Z)* est un littéral général maximum. Un soulignement «_» est utilisé pour représenter une variable distincte si le nom de la variable n'est pas nécessaire, par exemple, *p(_, _, _)*. L'ensemble de toutes les instances d'un littéral L est défini comme *inst(L)*. Pour un

ensemble de littéraux $L = \{L_1, \ldots, L_n\}$, alors $inst(L) = inst(L_1) \cup \ldots \cup inst(L_n)$. Le nombre d'éléments dans un ensemble S est noté $|S|$. $Occ(e, S)$ est le nombre d'occurrences de e dans S. Le signe $\subseteq_{ms}$ désigne la relation d'inclusion sur les multi-ensembles, c'est-à-dire $S_1 \subseteq_{ms} S_2$ si et seulement si $Occ(e, S_1) \leq Occ(e, S_2)$ pour tout e contenu dans S_1 ou S_2. Lorsque C et D sont des clauses, C *subsume* D s'il y a une substitution θ telle que $C\theta \subseteq_{ms} D$. C *subsume correctement* D si C subsume D mais D ne subsume pas C. Pour un ensemble Σ de clauses, $\mu\Sigma$ désigne l'ensemble des clauses de Σ non subsumées correctement par une clause dans Σ. $Th(\Sigma)$ représente l'ensemble des conséquences logiques de Σ.

Les clauses caractéristiques sont destinées à représenter les clauses «intéressantes», et sont construites sur un sous-vocabulaire du langage sous-jacent appelé un champ de production.

Définition 2.1: Un *champ de production* P est une paire (L, *Cond*), où L est un ensemble de littéraux et *Cond* est une condition veut être satisfaite. Si *Cond* n'est pas spécifié, P est juste noté (L). Une clause C appartient à $P = (L, Cond)$ si chaque littéral dans C appartient à $inst(L)$ et C satisfaite *Cond*. Pour un ensemble de clauses Σ, l'ensemble des conséquences logiques de Σ appartenant à P est noté $Th_p(\Sigma)$. Un champ de production P est *stable* si, pour deux clauses des C et D telles que C subsume D, la clause D appartient à P seulement si C appartient à P.

Définition 2.2: Les *clauses caractéristiques* de Σ par rapport à P sont $Carc(\Sigma, P) = \mu Th_p(\Sigma)$. Soit C une clause. Les nouvelles clauses caractéristiques de C par rapport à Σ et P sont $Newcarc(\Sigma, C, P) = \mu[Th_p(\Sigma \cup \{C\}) \setminus Th(\Sigma)]$.

Proposition 2.1: Une clause est une nouvelle clause caractéristique de C par rapport à Σ et P si et seulement si elle est la clause caractéristique de $\Sigma \cup \{C\}$ mais pas la clause caractéristique de Σ. À savoir, $Newcarc(\Sigma, C, P) = Carc(\Sigma \cup \{C\}, P) \setminus Carc(\Sigma, P)$.

Soit $F = C_1 \wedge \ldots \wedge C_m$ comme une forme normale conjonctive (CNF) de formule. Alors,

$$Newcarc(\Sigma, F, P) = \mu\left[\bigcup_{i=1}^{m} Newcarc(\Sigma_i, C_i, P)\right]$$

où $\Sigma_1 = \Sigma$ et $\Sigma_{i+1} = \Sigma_i \cup \{C_i\}$ pour $i = 1, \ldots, m - 1$.

Noter que Σ n'est pas consistent si et seulement si $Carc(\Sigma, P) = \{\varnothing\}$ pour tout champ de production stable $P = (L, Cond)$. Cela signifie que la conclusion de preuve est un cas particulier de conclusion conséquente. D'autre part, si $\Sigma \not\models L$ (Σ n'entraîne pas L) et $\Sigma \not\models \neg L$ (Σ n'entraîne pas $\neg L$) pour les littéraux L et $\neg L$ appartenant à P, $Carc(\Sigma, P)$ contient une tautologie $L \vee \neg L$ tant que $L \vee \neg L$ satisfait *Cond*. Les clauses caractéristiques $Carc(\Sigma, P)$ peuvent être exprimées en utilisant progressivement les opérations *Newcarc*. Soit $Taut(L)$ l'ensemble des tautologies de la forme $L \vee \neg L$ tel que L et $\neg L$ appartiennent à $inst(L)$. Alors, pour un ensemble Σ de clauses, une clause C et un champ de production stable $P = (L, Cond)$, il vient que :

$Carc(\varnothing, P) = \mu\{T \mid T \in Taut(L)$ et T satisfait $Cond\}$,

$Carc(\Sigma \cup \{C\}, P) = \mu[Carc(\Sigma, P) \cup Newcarc(\Sigma, C, P)]$.

Pour calculer $Newcarc(\Sigma, C, P)$, SOL se concentre sur la seule production des théorèmes appartenant à P, et traite une clause C nouvellement ajoutée comme la *clause de départ*. Ces caractéristiques sont importantes pour la conclusion conséquente car la procédure peut directement déduire les théorèmes relatifs à l'information ajoutée.

Définition 2.3: Un *tableau de clauses T* est un arbre ordonné étiqueté, où chaque nœud non racine de T est marqué par un littéral. Un nœud est défini avec son étiquette (soit un littéral) si aucune confusion ne se pose. Si les successeurs immédiats d'un nœud sont des littéraux $L_1, \ldots, L_n$, alors la clause $L_1 \vee \ldots \vee L_n$ est appelé une *clause de tableau*. La clause de tableau en dessous de la racine est appelée la *clause de départ*. T est un *tableau de clauses* pour un ensemble Σ de clauses si chaque clause de tableau C en T est une instance d'une clause D dans Σ. Dans ce cas, D est appelé une *clause d'origine* de C dans Σ.

Un *tableau de connexion (serré)* est un tableau de clauses tel que, pour chaque nœud non-feuille L sauf la racine, il y a un successeur immédiat marqué avec $\neg L$. Un tableau marqué T est un tableau de clauses tel que des feuilles sont marquées avec des étiquettes *fermé* ou *sauté*. Les feuilles non marquées sont appelées *sous-buts*. Un nœud N dans T est dit être résolu si N lui-même est un nœud de feuille marqué ou tous les nœuds de feuilles de branches à N de T sont marqués. T est résolu si toutes les feuilles sont marquées. *Sauté(T)* désigne l'ensemble des littéraux de nœuds marqués sauté.

Définition 2.4: Un tableau T est *régulier* si chaque nœud sur une branche en T est étiqueté avec un littéral différent. T est dit *libre-tautologie* si aucune clause de tableau en T n'est une tautologie. T est dit *libre-complément* si deux nœuds non-feuilles sur une branche en T ne sont jamais étiquetés avec des littéraux complémentaires. Un tableau T est *sauté-régulier* s'il n'y a pas de nœud L en T tel que $\neg L \in sauté(T)$. T est *TCS-libre* (TCS-Tableau de Clauses Subsumées libre) pour un ensemble de clauses Σ si aucune clause tableau C en T n'est subsumée par une clause dans Σ autre que les clauses d'origine de C.

Définition 2.5: Une *fonction φ de sélection* est une application attribution d'un sous-but à chaque tableau. φ est dite *profondeur d'abord* si φ choisit de tout tableau T un sous-but avec une profondeur maximale. φ est *stable par substitution* si, pour tout tableau T et toute substitution, $\varphi(t) = \varphi(T\sigma)$ où $T\sigma$ est le tableau qui est construit en appliquant σ à tous les littéraux de T.

Définition 2.6: Soit Σ un ensemble de clauses, C une clause, P un champ de production, et φ une fonction de sélection. Un *SOL-déduction dérivée d'une clause S de Σ + C et P* via φ est constituée d'une séquence de tableaux $T_0, T_1, \ldots, T_n$ satisfaisant:

(1) T_0 se compose seulement de la clause de départ C. Tous les nœuds de feuille de T_0 ne sont pas étiquetés.

(2) T_n est un tableau résolu, et $sauté(T_n) = S$.

(3) Pour chaque T_i *(i = 0, ..., n)*, T_i est régulier, libre-tautologie, libre-complément, sauté-régulier, et TCS-libre pour $\Sigma \cup \{C\}$.

(4) Pour chaque T_i *(i = 0, ..., n),* la clause *sauté*(T_i) appartient à P.

(5) T_{i+1} est construit à partir de T_i comme suit. Sélectionner un sous-but K par φ, puis appliquer une des règles suivantes à T_i pour obtenir T_{i+1}:

(a) Si *sauté*(T_i) $\cup \{K\}$ appartient à P, alors marquer K avec sauté.

(b) Si *sauté*(T_i) contient un littéral L, et K et L sont unifiables (K et L peuvent être l'inverse l'un de l'autre) avec θ, marquer K avec sauté, et appliquer θ à T_i.

(c) Sélectionner une clause B de $\Sigma \cup \{C\}$ et obtenir une variante $B' = L_1 \vee \ldots \vee L_m$ en renommant. S'il y a un littéral L_j tel que $\neg K$ et L_j sont unifiables avec θ, puis ajouter les nouveaux nœuds $L_1, \ldots, L_m$ à K comme les successeurs immédiats. Ensuite, marquer L_j avec fermé et appliquer θ au tableau prolongée.

(d) Si K a un nœud ancêtre L, et $\neg K$ et L sont unifiables avec θ, alors marquer K avec fermé, et appliquer θ à T_i.

Théorème 2.1: (Correction et exhaustivité de SOL) Pour toute fonction φ de sélection, ce qui suit s'applique:

(1) **Correction** : S'il y a un SOL-déduction d'une clause S de $\Sigma + C$ et P par φ, alors S appartient à $Th_p(\Sigma \cup \{C\})$.

(2) **Exhaustivité** : Si une clause F n'appartient pas à $Th_p(\Sigma)$, mais appartient à $Th_p(\Sigma \cup \{C\})$, alors il existe une SOL-déduction d'une clause S de $\Sigma + C$ et P par φ tels que S subsume F.

Exemple 2.1: **U**ne clause de départ C, un ensemble de clauses Σ et un champ de production P sont définis comme suit. Soit φ une fonction de sélection.

$C = p(X) \vee s(X)$,

$\Sigma = \{q(X) \vee \neg p(X), \neg s(Y), \neg p(Z) \vee \neg q(Z) \vee r(Z)\}$

$P = (L^+$, *la longueur est inférieure à 2)*

La **Figure 2.1** montre trois tableaux résolus qui sont dérivés par SOL-déduction sur $\Sigma + C$ et P par φ. Dans le tableau T_a, le nœud $p(X)$ est sauté depuis le littéral positif $p(X)$ qui appartient à P, et $s(X)$ est étendu à l'aide de la clause d'unité $\neg s(Y)$. Noter que $s(X)$ ne peut pas être sauté puisque le champ de production P limite la longueur maximale des conséquences à 1. La conséquence est *sauté*(T_a) = $\{p(X)\}$. T_b représente un autre tableau dérivé dont le nœud $p(X)$ est prolongé par $q(X) \vee \neg p(X)$, et le nœud $q(X)$ est sauté. La conséquence de T_b est *sauté*(T_b) = $\{q(X)\}$. Dans T_c, le nœud $p(X)$ est prolongé par $q(X) \vee \neg p(X)$ et $q(X)$ par $\neg p(Z) \vee \neg q(Z) \vee r(Z)$ respectivement. Le nœud du bas $\neg p(X)$ est fermé par réduction avec l'ancêtre $p(X)$, et $r(X)$ est sauté. La conséquence est *sauté*(T_c) = $\{r(X)\}$. Dans cet exemple, il y a six

tableaux résolus. Car les tableaux restés génèrent la même conséquence que T_c, ils sont redondants. Comme résultats, trois nouvelles clauses caractéristiques sont obtenues dans cet exemple :

Newcarc(Σ, C, P) = {p(X), q(X), r(X)}.

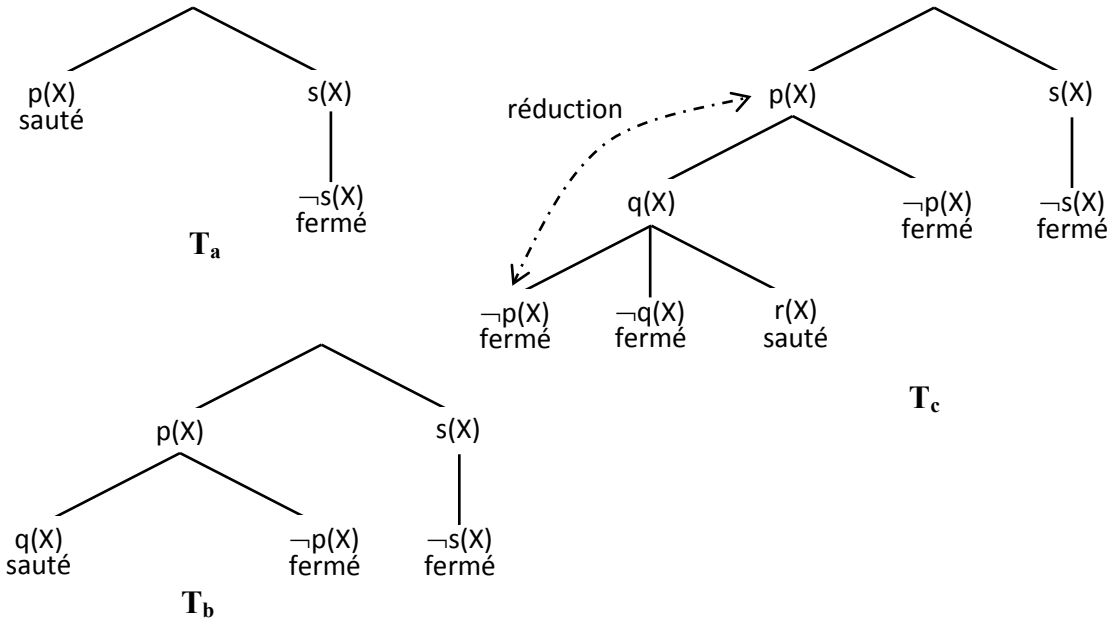

Figure 2.1 Tableaux résolus pour l'Exemple 2.1

La tâche de conclusion conséquente est de trouver toutes les (nouvelles) clauses caractéristiques. Cela signifie que même si un tableau résolu est trouvé, il faut continuer à chercher d'autres tableaux résolus pour trouver toutes les clauses caractéristiques. En conclusion de preuve, le processus de calcul peut être arrêté immédiatement si une réfutation apparait. Ceci est la différence importante entre la conclusion conséquente et la conclusion de preuve.

Définition 2.7: L'ensemble de *SOL-dérivées* de $\Sigma + C$ et P par φ est:

$$\Delta(\Sigma, C, P, \varphi) = \{S | S \text{ est dérivée utilisant une SOL–déduction de } \Sigma + C \text{ et } P \text{ par } \varphi\}$$

L'ensemble des clauses de production de $\Sigma + C$ et P par φ est:

$Prod(\Sigma, C, P, \varphi) = \mu\Delta(\Sigma, C, P, \varphi)$.

Proposition 2.2: Pour toute fonction φ de sélection,

$Newcarc(\Sigma, C, P) = Prod(\Sigma, C, P, \varphi) - Th_p(\Sigma)$.

Afin de clarifier l'espace de recherche du processus de conclusion conséquente, l'arbre de recherche SOL est défini pour explicitement toutes les SOL-déductions possibles.

Définition 2.8: *L'arbre de recherche SOL* de $\Sigma + C$ et P par φ est un arbre T marqué avec les tableaux comme suit. Un nœud est identifié avec son étiquette (c'est à dire un tableau) si aucune confusion ne se pose:

(1) La racine de T est un tableau qui se compose seulement de la clause de départ C.

(2) Chaque nœud non-feuille N dans T a autant de nœuds successeurs qu'il existe d'applications réussies d'une étape d'inférence unique appliquée au sous-but sélectionné dans N par φ, et les nœuds successeurs de N sont les tableaux de résultantes respectives.

La *procédure conséquente de recensement SOL* visite constructivement tous les nœuds dans l'arbre de recherche SOL, et énumère l'ensemble des clauses de production.

Définition 2.9: La *procédure conséquente de recensement SOL* de $\Sigma + C$ et P via φ est définie comme suit:

(1) T_0 est un tableau qui comprend seulement la clause de départ C, et *Conqs* est initialisée comme {} (*Conqs* est une variable globale).

(2) Appeler la procédure *EnumConqs*(T_0) dans la **Figure 2.2**.

(3) Renvoyer l'ensemble de clauses de production *Conqs*.

La correction de la procédure conséquente de recensement SOL tient évidemment par les définitions de clauses de production et d'EnumConqs.

Proposition 2.3: Laisser Conqs être un ensemble de conséquences obtenues de la procédure conséquente de recensement SOL de $\Sigma + C$ et P par φ. Alors,

Conqs = Prod(Σ, C, P, φ).

```
EnumConqs(T)
  T: un tableau
Commencer
  si Conqs = {∅} alors revenir; fin          / * trouvé la conséquence la plus générale * /
  si T est un tableau résolu alors
    C := sauté(T);                           / * trouvé une nouvelle conséquence * /
    Conqs := μ(Conqs ∪ {C});                 / * supprimer conséquences non minimales * /
    revenir;
  fin
  K := φ(T);                                 / * sélectionner un sous-but de T * /
  R := un ensemble de règles d'inférence qui sont applicables à K;
  pour chaque r ∈ R faire
    T' := le tableau obtenu à partir de T en appliquant r de K;
    EnumConqs(T') ;          / * Correspondant à l'extension de l'arbre de recherche SOL * /
  fin
fin
```

Figure 2.2. La procédure conséquente de recensement SOL.

2.2. Production pour l'abduction et l'induction

2.2.1. Définition

Selon la thèse de P.Siegel [26], une *production PR* de *C* (un ensemble des clauses) est une suite finie de couples $<p_i, A_i>$. Chacun de ces couples est une étape (ou une inférence) de *PR* dans laquelle p_i est une clause, la clause en production et A_i un arbre. Cette suite a les propriétés *A, B* et *C* :

A. La première étape de *PR* est $<p_1, A_1> = <0, (l_1) \ldots (l_m)>$

   où les l_j sont tous les littéraux d'une même clause *c* de *C*, l'origine de la production. Chacun de ces littéraux apparaissant une et une seule fois. La clause en production est vide.

B. La dernière étape est $<p_n, A_n>$ si A_n est l'arbre vide.

   La clause p_n est alors la clause produite par la production.

C. Si la *k-ième (k<n)* étape est $<p, A> = <p, (l\ s)\ B>$

   où *(l s)* est une branche de feuille *l* (le littéral à effacer), *B* un arbre et *p* la clause en production, alors l'étape suivante a l'une des forme *C1* ou *C2*.

   C1. On met en production le littéral *l* si l'étape suivante est : $<l\ p, B>$

   C2. On effectue une résolution si l'étape suivante est : $<p, (l'_1\ l\ s)\ (l'_2\ l\ s) \ldots (l'_t\ l\ s)\ B>$

   où les littéraux l'_i sont tels que les conditions de *C2a* et de *C2b* sont vérifiées.

   C2a. Il existe une clause *c'* de *C*, la clause appelée à cette étape,

   - qui contient l'opposé de *l*,
   - dont aucun littéral n'est dans la branche *(l s)* (les littéraux de cette branche sont les ascendants ou a-ancêtres)
   - dont aucun littéral n'a pour opposé une feuille de l'arbre *B* (les frères de *l*)
   - dont aucun littéral n'a pour opposé un littéral de la clause en production *p* (les littéraux de cette clause sont les littéraux en production).

   Les trois dernières conditions sont les conditions de non répétition.

   C2b. Les l'_i sont alors les littéraux de cette clause *c'*, qui ne sont pas **immédiatement effaçables**, c'est à dire ceux :

   - dont l'opposé n'est pas dans la branche *(l s)*,
   - qui ne sont pas égaux à une feuille de l'arbre *B*,
   - qui ne sont pas égaux à un littéral de la clause en production.

2.2.2. Exemple d'utilisation

Soit l'ensemble C de cinq clauses :

$\neg a \quad b \quad c$

$\neg b \quad d \quad e$

$f \quad e \quad \neg a \quad \neg d \quad c$

$\neg c \quad e \quad f$

$\neg e \quad f \quad g$

L'ensemble $C \cup \{a\ f\}$ implique la clause $f\ g$ et C ne l'implique pas. Il existe donc au moins une production d'origine $a\ f$ qui produit $f\ g$. C'est par exemple le cas de :

1.						*<0*	*,(a)*	*(f)>*		
2.	*¬a*	*b*	*c*			*<0*	*,(ba)*	*(ca)*	*(f)>*	
3.	*¬b*	*d*	*e*			*<0*	*,(dba)*	*(eba)*	*(ca)*	*(f)>*
4.	*f*	*e*	*¬a*	*¬d*	*c*	*<0*	*,(eba)*	*(ca)*	*(f)>*	
5.	*¬e*	*f*	*g*			*<0*	*,(geba)*	*(ca)*	*(f)>*	
6.						*<g*	*,(ca)*	*(f)>*		
7.	*¬c*	*e*	*f*			*<g*	*,(eca)*	*(f)>*		
8.	*¬e*	*f*	*g*			*<g*	*,(f)>*			
9.						*<g f*	*,0>*			

Pour visualiser globalement toutes les étapes d'une production, il est possible de faire un dessin du type :

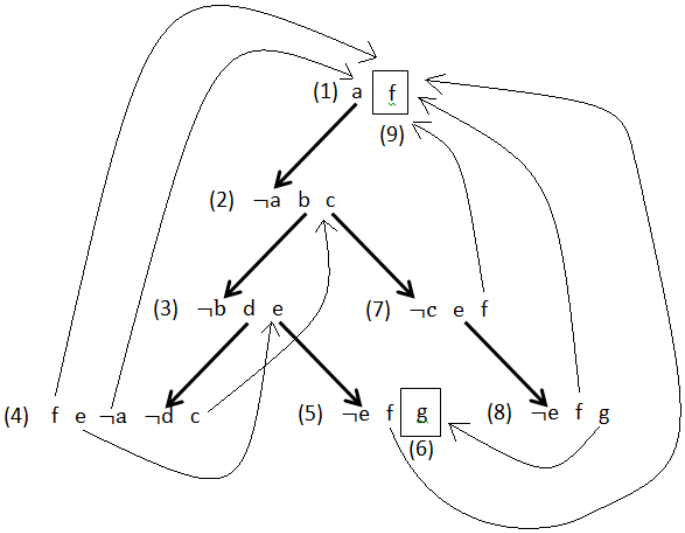

Figure 2.3. Les étapes d'une production

2.3. Champ de production

La notion de champ de production dépassant le cadre restreint du calcul propositionnel, cette partie en donnera la définition dans le cadre général du calcul des prédicats. Une « formule » sera donc ici une formule de la logique du premier ordre et, bien entendu, une « formule propositionnelle » une formule du calcul propositionnel.

2.3.1. Définition

Définition 2.10:

Un *champ de production P* est un ensemble de formules dont chacune est un résultat possible dont on a besoin de savoir s'il est vrai ou non dans un certain état de connaissance.

Le but du jeu est de trouver les formules de cet ensemble vraies pour une connaissance donnée. Il s'agit donc d'une notion assez naturelle qui pourra se définir, correctement d'un point de vue logique, de manière simple.

De manière très générale, deux problèmes se posent :

Problème 2.1:

Soit :

- *L* un langage du premier ordre,
- *P* un ensemble de formules de *L*, le champ de production,
- *C* un ensemble de formules de *L*, la connaissance,

Trouver l'ensemble, *produites(C, P)*, des formules sémantiquement impliquées par *C* et appartenant à *P*.

Si *th(C)* représente toutes les formules sémantiquement impliquées par *C*, on a donc :

$$produites(C, P) = th(C) \cap P$$

Problème 2.2:

Soit :

- *L, P, C* définis comme précédemment,
- *C'* un nouvel ensemble de formules de *L*, la nouvelle connaissance,

Trouver toutes les formules de *P* impliquées par $C \cup C'$, qui appartiennent à *P* et qui ne sont pas impliquées par *C*.

Cet ensemble des formules se notera *nouv-prod(C', C, P)* :

$$nouv\text{-}prod(C', C, P) = produites(C \cup C', P) - produites(C, P).$$

Définition 2.11:

Si c_1 et c_2 sont des clauses, on dira que c_1 est une *sous clause* de c_2, ou que c_1 *subsume* c_2 si tout littéral de c_1 est un littéral de c_2. On dira également que c_1 subsume strictement de c_2 (est une sous clause stricte) si c_1 subsume c_2 et c_2 ne subsume pas c_1.

Si *F* est un ensemble de formules qui ne sont pas des tautologies, et *G* un sous ensemble de *F*, on dit que :

- *G* est un *précurseur* de *F* si toute formule de *F* est impliquée par une formule de *G*. « Pour toute *f* dans *F* il existe *g* dans *G* telle que $g \vDash f$ »

- G est un *plus petit précurseur* de F (ou *précurseur minimal* de F), si de plus aucune formule de G n'implique une autre formule de G. « Si G est un précurseur de F, et si g_1 et g_2 sont dans G et $g_1 \models g_2$ alors $g_1 = g_2$ ».

Propriété 2.1:

Si G_1 et G_2 sont deux précurseurs minimaux de F, il existe une bijection b_{ij} de G_1 dans G_2 telle que, pour toute formule g_1 de G_1, il existe un unique g_2 de G_2 tel que g_1 et $b_{ij}(g_2)$ sont logiquement équivalentes.

Définition 2.12:

Un champ de production P est *stable* pour l'implication sémantique si toute formule qui implique sémantiquement une formule de P est dans P.

Propriété 2.2:

Si P est un ensemble de clauses propositionnelles, alors P est stable si et seulement si toute sous clause d'une clause de P est dans P.

2.3.2. Exemple d'utilisation

Si L est un langage propositionnel contenant une infinité de propositions $p_1, p_2, ...,$ et si la connaissance est un ensemble de deux clauses :

$C = \neg p_1 \vee p_2 \vee p_3$ 'si p_1, alors p_2 ou p_3 est vrai'

p_1 'p_1 est vrai'

Si le champ de production est l'ensemble des clauses de L ne contenant que des littéraux positifs (clauses positives), on a :

$produites(C, P) = p_1$ 'p_1 est vrai'

$p_2 \vee p_3$ 'p_2 ou p_3 est vrai'

Si maintenant la nouvelle connaissance se compose des deux unaires :

$C' = p_4$ 'p_4 est vrai'

$\neg p_3$ 'p_3 est faux'

alors on obtient :

$nouv\text{-}prod(C', C, P) = p_4$ 'p_4 est vrai'

p_2 'p_2 est vrai'

2.4. Algorithmes de calcul de production

2.4.1. Description simplifiée

Notre problème pratique est dont de calculer toutes les clauses impliquées par un ensemble fini *C* de clauses et appartenant à un champ de production *P*. Comme *P* est en général très grand, il est évident que la technique consistant à vérifier, pour toute clause de *P*, si elle est impliquée par *C* ou non, est totalement inapplicable. Cette partie va étudier un algorithme basé sur la *SOL* résolution qui essaie de résoudre partiellement ce problème.

Pour bien faire son travail, il doit s'arrêter, ne donner que des résultats corrects et donner tous les résultats. On s'assure de ceci en démontrant trois propriétés.

Propriété 2.3:

Un ensemble fini de clauses a un ensemble fini de productions (*l'algorithme s'arrête*).

Propriété 2.4:

Toute clause produite par une production de *C* est impliquée par *C* (*les résultats sont valides*).

En particulier, si une production de *C* produit la clause vide, alors *C* est inconsistant.

Propriété 2.5:

Si une clause *d* est impliquée par *C*, il existe une production de *C* dont la clause produite subsume *d* (*les résultats sont tous trouvés*).

En particulier, si *C* est inconsistant, il existe une production de *C* qui produit la clause vide.

Propriété 2.6:

Si *C* n'implique pas *d* et $C \cup \{c\}$ implique *d*, il existe une production de $C \cup \{c\}$, d'origine *c* dont la clause produite subsume *d*.

En particulier, si $C \cup \{c\}$ est inconsistant, et *C* ne l'est pas, il existe une production de *C* d'origine *c* qui produit la clause vide.

Il faut maintenant décrire l'algorithme qui génère toutes les clauses impliquées par un ensemble *C* de clauses et appartenant à un champ de production *P*. Cet algorithme a été implanté en SWI Prolog. Les clauses, branches et arbres sont représentés classiquement :

- un littéral *x* s'écrira *plus(x)* s'il est positif et *moins(x)* s'il est négatif.
- une liste de littéraux (clause ou branche) se notera de la manière habituelle $l_1.\ l_2...\ l_n.$ *nil* où *nil* représentant la liste vide.
- un arbre est une liste $s_1.\ s_2...\ s_m.$ *nil* de branches. Les s_i sont donc des listes non vides de littéraux (donc différentes de *nil*).
- un ensemble de clauses est identifié à une liste $c_1.\ c_2...\ c_p.$ *nil* de clauses.

Description simplifiée :

Pour produire les clauses impliquées par une liste *C = c. C'* (*c* est une clause et *C'* une liste de clauses éventuellement vide) appartenant à un champ de production *P*, il faut produire les clauses impliquées par *C'* puis produire les clauses impliquées par *c.C'* et non par *C'*. Pour ce faire on utilise le prédicat *grande-production* :

grande-production(nil, P) → ;

grande-production(c.C, P) →

grande-production(C, P)

construire-arbre(c, nil, nil, nil, A)

production(nil, A, C, P) ;

Le cœur de l'algorithme est la définition d'un prédicat à quatre arguments *production(p, A, C, P)*, où *p* est la clause en production, *A* un arbre, *C* l'ensemble de clauses et *P* le champ de production. L'arbre *A* origine est donné par le prédicat construire-arbre, qui transforme la clause origine

$c = l_1. l_2... l_n. nil$

en un arbre dont les branches sont composées d'un littéral de cette clause :

$A = (l_1.nil).(l_2.nil) .... (l_n. nil).nil$

On définit production par trois clauses Prolog :

```
(1) production(p, nil, C, P) →
        utiliser(p) ;
(2) production(p, (l.s).B, C, P) →          /* Forme C2*/
        dans(c', C)                         /* Prendre une clause c' de C */
        bonne-clause(c', p, (l.s).B)        /* Conditions C2a */
        construire-arbre(c', p, (l.s).B, B') /* Construire l'arbre B' à partir de la
                                               clause c', conditions C2b*/
        production(p, B', C, P) ;
(3) production(p, (l.s).B, C, P) →          /* Forme C1*/
        bon-debut-de-clause(l.p, P)         /* l.p satisfaisante le champ de
                                               production P*/
        production(l.p, B, C, P) ;

dans(x, x.r) → ;

dans(x, y.r) → dans(x,r) ;
```

La clause (*1*) est le cas terminal d'une production. Ici, on l'ajoutera sous de forme d'une clause Prolog : *clause-produite(p) → ;*

utiliser(p) → assert(clause-produite(p), nil) ;

Pour être en accord avec la définition, *bonne-clause* doit vérifier les conditions de non répétition. Il faut donc :

1- S'assurer que l'opposé de la feuille de la première branche de l'arbre est bien dans la clause candidate.
2- Vérifier que tous les littéraux de cette clause :
   - ne sont pas dans la première *l.s* de l'arbre,
   - n'ont pas leur opposé dans la clause en production *p*,
   - n'ont pas leur opposé égal à une feuille de l'arbre *B*.

Ceci s'effectue de manière naturelle :

bonne-clause(c', p, (l.s).B) →

oppose(l, l')

dans(l', c')

bons-littéraux(c', p, (l.s), B)

bons-littéraux(nil, p, A) → ;

bons-littéraux(l.r, p, s.B) →

bons-littéraux(r, p, s.B)

pas-dans(l, s)

oppose(l, l')

pas-dans(l', p)

pas-dans-feuilles(l', B) ;

oppose(plus(x), moins(x)) → ;

oppose(moins(x), plus(x)) → ;

pas-dans(x, nil) → ;

pas-dans(x, y.r) →

dif(x, y)

pas-dans(x, r) ;

pas-dans-feuilles(x, nil) → ;

pas-dans-feuilles(x, (l.s).B) →

pas-dans(x, l.s)

pas-dans-feuilles(x, B) ;

Il faut maintenant définir *construire-arbre* qui, à partir de la clause

$c' = l_1. l_2... l_n. nil$

de la clause en production *p* et de l'arbre *(l.s).B* va construire l'arbre

$A' = (l'_1.l.s) ... (l'_p.l.s).B$

les l'_i étant les littéraux de *c'* non immédiatement effaçables.

construire-arbre(nil, p, s.B, B) → ;

construire-arbre(l.r, p, s.B, (l.s).B') → /* Conditions C2a sont vérifiées*/

oppose(l, l')

pas-dans(l', s)

pas-dans(l, p)

pas-dans-feuilles(l, B)

/

construire-arbre(r, p, s.B, B') ;

construire-arbre(l.r, p, A, A') → /* Conditions C2a ne sont pas vérifiées*/

construire-arbre(r, p, A, A') ;

Pour vérifier qu'une clause est dans un champ de production défini par une quadruplé de bases quelconque, on peut écrire :

bon-debut-de-clause(c, P.l-iste-P) →

dans-le-champ(c, P)

/ ;

bon-debut-de-clause(c, P.l-iste-P) →

bon-debut-de-clause(c, l-iste-P) ;

dans-le-champ(l.c, <s-igne, l-ong, a-lpha, s-ous-cl>) →

eq(l, s-igne)

dans(l, a-lpha)

longueur-inferieure(l.c, l-ong)

dans(c', s-ous-cl)

subsume(c, c')

/ ;

subsume(nil, c) →

/ ;

subsume(l.c', c) →

subsume(c', c)

dans(l, c) ;

longueur-inferieure(c, infini) →

/ ;

longueur-inferieure(nil, l) → ;

longueur-inferieure(a.c, l) →

/

impasse ;

longueur-inferieure(a.c, l) →

val(sub(l, l), l')

longueur-inferieure(c, l') ;

2.4.2. Algorithme avec coupure

A toute étape de production, on essaie d'effacer le premier littéral de la première branche d'arbre (le littéral à effacer). Pour ce faire, il faut choisir une clause candidate contenant l'opposé de ce littéral et satisfaisant aux conditions de non répétition, en ôter tous les littéraux immédiatement effaçables (qui ne satisfont pas aux conditions *C2b*), puis essayer d'effacer la clause restante. Quand tous ces choix ont été effectués et résolus, il reste un choix supplémentaire qui est de mettre en production le littéral à effacer.

Or, si le littéral à effacer a été, pour un certain choix, complètement effacé sans avoir eu besoin d'installer des littéraux en productions supplémentaires, il est inutile d'effectuer les choix restants. On peut formaliser ceci par une définition et une propriété.

Définition 2.13:

S'il existe un début de production, *D*, de dernière étape (*j*) :

.....

(i)<p_1, (l.s) B_1>

.....

(j)<p_2, B_2>

telle que $B_1 = B_2$ et $p_1 = p_2$, on dira que le littéral *l* a été complètement effacé.

Propriété 2.7:

Dans ce cas toute production égale à *D* jusqu'à l'étape (*i*) incluse, ne pourra produire que des clauses subsumées par d'autres clauses produites par les productions de début *D*.

Donc, d'après la structure de l'algorithme, qui à partir de l'étage (*i+1*), calcule toutes les productions ayant ces (*i*) premières étapes avant de faire les autres choix pour *l* à l'étape (*i+1*), il sera inutile si ce littéral est complètement effacé de faire ces autres choix. C'est principalement pour cette raison que l'on essaie d'abord d'effectuer toutes les résolutions possibles sur un littéral avant de mettre ce littéral en production (avant d'essayer d'effacer de manière classique un littéral, on regarde s'il n'existe pas une clause contenant son opposé et dont tous les littéraux sont immédiatement effaçables). Le choix inverse est possible mais donne de moins bons résultats.

Démonstration :

Soit *PR1* une production égale à *D* jusqu'à (*i*). Cette production aura une étape (*j'*) dont l'arbre est *B1*, et peut donc s'écrire :

.....

(i)<p_1, (l.s) B_1>

.....

(j')<q p_1, B_1>

.....

(n')<r q p_1, 0>

PR1 produit la clause $p = r\ q\ p_1$ dans laquelle *q* et *r* sont des clauses éventuellement vides (la notation inclut le cas où *j' = n'*). Pour prouver le théorème, il faut construire à partir de *PR1* une production, *PR2*, dont le début est *D* et dont la clause produite subsume *p*.

Le début de *PR2* est donc :

.....

(i)<p_1, (l.s) B_1>

.....

(j)<p_1, B_1>

L'important est que, à l'étape (*j'*) de *PR1* et à l'étape (*j*) de *PR2*, les arbres sont égaux. À l'étape (*j*), la clause en production, p_1, de *PR2* est une sous clause de la clause en production à l'étape (*j'*), *q* p_1, de *PR1*.

Les étapes de *PR2* qui suivent (*j*) sont construites à partir des étapes de *PR1* qui suivent (*j'*) telles que, si la clause appelée contient un littéral de *r q* alors ce littéral n'est pas immédiatement effacé mais inséré dans l'arbre. Il sera mis en production à l'étape où il sera littéral à effacer de l'arbre courant. Plus précisément :

1. L'étape (*j+1*) de *PR2* est construite à partir de l'étape (*j'+1*) de *PR1* telle que :
   - si (*j'+1*) met en production le littéral à effacer, alors (*j+1*) mettra également en production ce même littéral à effacer.
   - si (*j'+1*) effectue une résolution sur une clause *c*, alors (*j+1*) effectuera également une résolution sur cette même clause *c*. Les branches ajoutées à l'arbre *B1* (dans *PR2*) sont obtenues à partir des littéraux de *c* non immédiatement effaçables (dans *PR1*). Mais les littéraux de c immédiatement effaçables dans *PR2*, sont les littéraux de *c* immédiatement effaçables dans *PR1*, plus les littéraux de *c*, égaux à un des littéraux de *q* (les ascendants et frères sont les mêmes dans *PR1* et *PR2* car les arbres sont égaux à l'étape (*i*)). Donc les branches à ajouter dans *PR2* sont les branches à ajouter dans *PR1*, plus éventuellement un certain nombre de branches dont les racines sont les littéraux de *c* qui ne sont pas dans *q*.
2. On répète l'opération pour les étapes suivantes de *PR2*, qui sont construites à partir des étapes de *PR1*. Les clauses appelées sont les mêmes dans *PR1* et *PR2*. La seule différence est que dans les arbres correspondants (dans *PR2*) sont insérées un certain nombre de branches supplémentaires, branches dont les racines seront toujours des éléments de *q*. Quand le littéral à effacer (dans *PR2*) portera sur une de ces branches supplémentaires, ce littéral sera mis en production. On insère donc un certain nombre d'étapes qui sont toutes des mises en production de littéraux de *q*.

La suite de couples ainsi formée est bien une production, car comme *PR1* en est une, on en déduit que pour toutes les étapes de résolution de *PR2* les clauses appelées satisfont aux conditions de non répétition. En effet, ces étapes sont telles que :

- l'étape associée dans *PR1* satisfait aux conditions de non répétition
- les ascendants sont les mêmes dans *PR1* et *PR2* car ces étapes sont des étapes de résolution et ne portent donc pas sur les branches supplémentaires (on a dit que les étapes portant sur les branches supplémentaires mettent en production un littéral).
- l'union des frères et littéraux en production de *PR2* est incluse dans l'union des frères et littéraux en production de *PR2*.

PR2 est donc bien une production et la clause produite par *PR2* subsume bien celle produite par *PR1* car les littéraux mis en production dans *PR1* et non dans *PR2* sont tous dans *q*.

Exemple 2.2:

Soit un ensemble de clause :

$C = \{a\ b\ c, \neg a\ b, \neg a\ u, \neg u\ v, \neg v\ w\}$

On veut calculer toutes les productions d'origine $a\ b\ c$. L'algorithme construira, par ses appels récursifs, la première production qui produit la clause $c\ b$:

1.			*<0 ,(a) (b) (c)>*
2.	*¬a*	*b*	*<0 ,(b) (c)>*
3.			*<b ,(c)>*
4.			*<c b ,0>*

Cette production est donnée par le choix de $\neg a\ b$ à la deuxième étape. Il reste deux choix à cette étape : soit appeler la clause $\neg a\ u$, soit mettre a en production. En fait ces choix sont inutiles car ils ne pourront générer que des clauses subsumées par d'autres clauses produites par la production effectuée auparavant. Le premier choix donne trois productions dont la première est :

1.			*<0 ,(a) (b) (c)>*
2.	*¬a*	*u*	*<0 ,(ua) (b) (c)>*
3.	*¬u*	*v*	*<0 ,(vua) (b) (c)>*
4.	*¬v*	*w*	*<0 ,(wvua) (b) (c)>*
5.			*<w ,(b) (c)>*
6.			*<b w ,(c)>*
7.			*<c b w ,0>*

En fait, $c\ b\ w$ est subsumée par $c\ b$.

2.4.3. Algorithme en Prolog

Pour produire les clauses d'un champ de production P impliquées par un ensemble C on appellera comme auparavant *grande-production(C, P)* :

```
grande-production(nil, P) → ;

grande-production(c.C, P) →

        grande-production(C, P)

        production(c, nil, nil, nil, p, C, P)

        utiliser(p) ;

production(nil, A, B, p, p, C, P) → ;          //le cas terminal

production(l.c, A, B, p1, p3, C, P) →

        conc(c, B, B1)                         // B1=c.B, construire la liste des frères

        oppose(l, l')

        dans(c1, C)

        dans(l', c1)

        bonne-clause(c1, l.A, B1, p1)          // Conditions C2a
```

```
        simplifier-clause(c1, l.A, B1, p1, c2)     // Conditions C2b, enlever les littéraux
        production(c2, l.A, B1, p1, p2, C, P)      //Concaténant aux ascendants le littéral
        (p1 == p2 *→ !; true)                      //couper(p1, p2)
        production(c, A, B, p2, p3, C, P) ;        //si p1 = p2 ou p1≠ p2
production(l.c, A, B, p1, p3, C, P) →              //mettre en production si p1≠ p2
        bon-debut-de-clause(l.p1, P)
        production(c, A, B, l.p1, p3, C, P) ;
bonne-clause(nil, A, B, p) → ;
bonne-clause(l.c, A, B, p) →
        pas-dans(l , A)
        oppose(l, l')
        pas-dans(l', B)
        pas-dans(l', p)
        bonne-clause(c, A, B, p) ;
simplifier-clause(nil, A, B, p, nil) → ;
simplifier-clause(l.c, A, B, p, l.c') →
        oppose(l, l')
        pas-dans(l', A)
        pas-dans(l, B)
        pas-dans(l, p)
        /
        simplifier-clause(c, A, B, p, c') ;
simplifier-clause(l.c, A, B, p, c') →
        simplifier-clause(c, A, B, p, c') ;
conc(nil, y, y) → ;
conc(a.x, y, a.z) →
        conc(x, y, z) ;
```

Où :

production(clause, ascendants, frères, produit-initiale, produit-finale, ensemble, champ)

- La clause à effacer, *clause*, est un bout d'une clause de $c \cup C$ qui contient le premier littéral (littéral à effacer).
- *ascendants* est la liste des ascendants de ce premier littéral.
- *frères* est la liste des frères de ce premier littéral, autres que les littéraux de *clause*.
- A l'appel de *production*, *produit-initiale* donne la clause en production à ce moment.

L'argument *produit-finale* donne alors la clause en production résultante.

Pour que ces algorithmes travaillent avec les variables, il faut les améliorer. Par exemple, pour représenter l'idée : *a* est une protéine et *b* un substrat ; tous les substrats sont activés par les protéines, ceci doit être écrit dans la basée de données comme ci-dessous :

```
proteine(a).

substrat(b).

ecrit :-

    proteine(X),

    substrat(Y),

    assert(clause(active(X,Y))).
```

Chapitre 3. La logique des défauts

3.1. Introduction

Quand un algorithme d'Intelligence Artificielle doit résoudre un problème, il peut être en mesure de s'appuyer sur des informations complètes. Sa tâche principale est alors de donner de bonnes conclusions par un raisonnement classique. Dans ce cas, la logique des prédicats peut être suffisante.

Cependant, souvent l'information est incomplète, car certaines informations ne sont pas disponibles. Parfois, il peut aussi être nécessaire de répondre rapidement et il est alors impossible de recueillir toutes les données pertinentes en un temps raisonnable. Dans ce cas, le système doit faire des conjectures plausibles, c'est la problématique des logiques non-monotones [7,9]. Pour la logique des défauts ces conjectures sont basées sur des règles empiriques, appelées défauts. Par exemple, un médecin d'urgence doit faire des conjectures sur les causes les plus probables des symptômes observés et il est impossible d'attendre le résultat de tests éventuellement étendus et chronophages avant le début du traitement.

Lorsque les décisions sont fondées sur des hypothèses, elles peuvent se révéler fausses face à de nouvelles informations qui seront disponibles [13]. Par exemple les examens médicaux peuvent conduire à un diagnostic modifié. Il faut alors ajouter de nouvelles informations qui peuvent être contradictoires, avec les conclusions (le diagnostic) précédent. Cet ajout est impossible en logique classique, qui a la propriété de monotonie. Intuitivement cette propriété dit que si une conclusion C est déductible d'un ensemble de prémices P_1, et que l'on ajoute des informations P_2 à P_1, alors C est démontrable de P_1 et P_2. Dans le cas de la logique du premier ordre P_1, P_2 et C sont des ensembles de formule et la propriété de monotonie s'écrit :

$$P_1 \vdash C \quad \Rightarrow \quad P_1 \cup P_2 \vdash C$$

Si cette propriété n'est pas vérifiée, alors on a une logique non-monotone. La logique des défauts présentée par Reiter en 1980 est une des premières logiques non-monotones. Elle est très intéressante, car simple à comprendre et à utiliser.

3.2. Notion de défaut

La logique des défauts définie par Reiter formalise le raisonnement par défaut. Elle permet de traiter les règles admettant des exceptions sans être obligé de remettre en cause les règles précédemment établies à chaque fois qu'une nouvelle exception apparaît. Une théorie des défauts est un couple $\{W, D\}$ où W est un ensemble de formules classiques de la logique du premier ordre et D un ensemble de défauts, qui sont des règles d'inférence spécifiques. Les défauts permettent de gérer les informations incomplètes [8].

Par exemple, en hiver, une règle générale utilisée par les arbitres de football pourrait être : «Un match de football doit avoir lieu, à moins qu'il n'y ait de la neige sur le stade ». Cette règle de base est représentée par un défaut :

$$\frac{football: \neg neige}{avoir\_lieu}$$

L'interprétation du défaut est la suivante : si on n'a pas l'information explicite qu'il y aura de la neige dans le stade, il est raisonnable de supposer qu'il n'y aura pas de neige (*¬neige)* et de conclure que le match aura lieu. On peut donc préparer le match. Mais s'il y a une forte chute de neige pendant la nuit avant le match, cette hypothèse n'est plus valide. On sait qu'il y a de la neige, et il est donc impossible d'assumer *¬neige*, donc le défaut ne peut pas être appliqué. Dans ce cas, il faut renoncer à la conclusion précédente (le match aura lieu). Le raisonnement est donc non-monotone.

La logique classique n'est pas appropriée pour modéliser cette situation. En effet, on pourrait tenter d'utiliser la formule

$$football \wedge \neg neige \rightarrow avoir\_lieu$$

Le problème avec cette règle, c'est qu'il faut établir de façon définitive qu'il n'y aura pas de neige dans le stade avant d'appliquer la règle.

Pour résoudre ce problème, le même exemple aurait pu être représenté par le défaut

$$\frac{football: avoir\_lieu}{avoir\_lieu}$$

avec la règle de logique classique

$$neige \rightarrow \neg avoir\_lieu.$$

Si la *neige* est certaine alors on déduit *¬avoir_lieu* par la logique classique, donc on ne peut pas inférer *avoir_lieu* par le défaut. Dans cette représentation, le défaut est une règle qui dit que les matchs se déroulent généralement et les *exceptions* de cette règle sont représentées par les règles classiques telles que celle ci-dessus.

Les défauts peuvent être utilisés pour modéliser le raisonnement prototypique, ce qui signifie que la plupart des instances d'un concept ont une certaine propriété. Un exemple est l'énoncé « Typiquement, l'enfant a des parents (vivants) », qui peut être exprimé par le défaut :

$$\frac{enfant(X): avoir\_parents(X)}{avoir\_parents(X)}$$

Une autre forme de raisonnement par défaut est le *raisonnement sans-risque*. Il s'agit de situations où une conclusion peut être tirée, même si ce n'est pas le plus probable, car une autre décision peut conduire à une catastrophe. Par exemple « En l'absence de preuve contraire présumer que l'accusé est innocent, s'exprime par :

$$\frac{accusé(X): innocent(X)}{innocent(X)}$$

On donne aussi des hiérarchies avec des exceptions qui sont en biologie. L'exemple classique est « En règle générale, les oiseaux volent », « les manchots sont les oiseaux », « les manchots ne volent pas ». On a alors le défaut:

$$\frac{oiseau(X): vole(X)}{vole(X)}$$

avec la règle de logique classique :

$$manchot(X) \rightarrow oiseau(X) \wedge \neg vole(X)$$

Les défauts peuvent être aussi utilisés pour modéliser l'hypothèse du monde clos de Reiter, utilisée pour les bases de données et pour la programmation logique. Selon cette hypothèse, un domaine d'application est décrit par un ensemble de formules logiques *F*. En simplifiant, l'hypothèse du monde clos dit qu'une information élémentaire positive (un atome) φ est considéré comme faux si *F* n'implique pas logiquement φ. Ceci peut se représenter par un défaut normal sans prérequis :

$$\frac{vrai: \neg\varphi}{\neg\varphi}$$

On encore, s'il est consistant d'assumer $\neg\varphi$ (ce qui est équivalent à ne pas avoir de preuve pour φ), on conclue $\neg\varphi$

3.3. Syntaxe de la logique des défauts.

Une théorie des défauts Δ = *(D, W)* est une paire *(D, W),* contenant un ensemble *W* de formules de la logique des prédicats (appelées faits ou axiomes) et un ensemble dénombrable *D* de défauts. Dans sa forme la plus générale, un défaut *d* s'écrit :

$$d = \frac{A: B_1, B_2, \ldots B_n}{C}$$

où *A,* $B_1, \ldots, B_n$ et *C* sont des formules logiques bien formées du premier ordre. La formule *A* est le *prérequis,* les formules $B_1, \ldots, B_n$ sont les *justifications* et *C* est le *conséquent*.

Un défaut signifie informellement : si *A* est vérifié, et s'il est possible que $B_1, \ldots, B_n$ soient vrais alors *C* est inféré.

Un défaut *d* est appelé *normal* si et seulement si sa justification est égale à son prérequis, donc s'il est de la forme :

$$d = \frac{A: C}{C}$$

Un défaut peut contenir des variables libres, par exemple :

$$\frac{oiseau(X): vole(X)}{vole(X)}$$

Ces défauts sont appelés défauts ouverts. Dans ce cas le défaut ouvert est considéré comme l'ensemble des défauts où *X* a été remplacée par tous les termes terminaux (sans variables) du langage. Un défaut ouvert représente donc un ensemble de défauts fermés qui peut éventuellement être infini.

3.4. Extensions.

L'utilisation des défauts augmente le nombre de formules déduites de la base de connaissance *W* : nous obtenons alors des extensions qui sont des ensembles de théorèmes dérivables de façon monotone. Intuitivement une extension est obtenue en utilisant un ensemble maximal consistant de défauts possibles. Cette définition va entrainer qu'il pourra exister plusieurs extensions éventuellement contradictoires.

3.4.1. Extensions - définition formelle.

Une extension de la théorie des défauts $\Delta = (D,W)$ est un ensemble *E* de formules, clos pour la déduction, contenant *W* et vérifiant la propriété suivante : si *d* est un défaut de *D* dont le prérequis est dans *E*, sans que la négation de sa justification soit dans *E*, alors le conséquent de *d* est dans *E*. Plus formellement, les extensions sont définies de la façon suivante.

Définition 3.1: *E* est une extension de Δ si et seulement si $E = \bigcup_{i=0,\infty} E_i$, avec :

1) $E_0 = W$

2) Pour tout *i*, $E_{i+1} = Th(E_i \cup \{C/d = (\frac{A:B}{C}) \in D, A \in E_i, \neg B \notin E\})$

où *Th(F)* désigne l'ensemble des théorèmes obtenus en logique classique à partir de *F*, c'est à dire $Th(F) = \{w/F \vdash w\}$.

Remarque : Il est important de noter que *E* apparaît dans la définition de E_{i+1}. Donc dans le cas général il peut ne pas être possible de construire *E* car à cause de la condition $\neg B \notin E$, il faut déjà connaître *E* pour construire *E*. La définition n'est donc pas constructive. Ceci peut être très gênant mais si l'on utilise uniquement des défauts normaux, la condition $\neg B \notin E$ se transforme en $\neg B \notin E_i$. Cette fois, il suffit de vérifier que la négation de la justification n'appartient pas à E_i. Un algorithme récursif peut donc être utilisé pour calculer *E*. De plus lorsque tous les défauts sont normaux et que *W* est satisfaisable, l'existence d'au moins une extension est assurée.

Exemple 3.1: Soit $\Delta = (D, W)$, où $W = \{a\}$, et *D* contient les défauts normaux suivants :

$$d_1 = \frac{a:\neg b}{\neg b}$$

$$d_2 = \frac{b:c}{c}$$

On obtient avec la définition des extensions une extension $E = Th(\{a, \neg b\})$ en utilisant le défaut d_1. Dans ce cas d_2 ne pas être utilisé car comme $\neg b \in E$, la justification de d_2 n'est pas vérifiée.

Exemple 3.2: C'est l'exemple classique. On sait que *Les manchots sont des oiseaux* et que *Titi est un manchot*. Ceci va s'exprimer par *W* composé de deux formules de la logique des prédicats :

$W = \{ \forall x, Manchot(x) \rightarrow Oiseau(x), Manchot(Titi)\}$

On sait aussi que *Sauf exception, les oiseaux volent* et que *Sauf exception, les manchots ne volent pas*. On exprime ceci par un ensemble de deux défauts normaux $D = \{d_1, d_2\}$:

$d_1 = Oiseau(x):Vole(x)/Vole(x)$

$d_2 = Manchot(x):\neg Vole(x)/\neg Vole(x)$

Le défaut d_1 peut aussi s'exprimer par : *Si x est un oiseau et qu'il est consistant de supposer que x peut voler, alors on conclue que x peut voler*. On voit que si l'on utilisait les deux défauts en même temps on obtiendrait que *Vole(Titi)* et $\neg Vole(Titi)$ ce qui est insatisfaisable. Pour cet exemple, on aura deux extensions. On a deux cas :

Cas 1: Dans ce cas on va commencer à construite l'extension en utilisant le défaut d_1

- On part de $E_0 = W = Th(\{Manchot(Titi) \rightarrow Oiseau(Titi), Manchot(Titi), Oiseau(Titi)\})$

- On utilise alors d_1. Le prérequis de d_1, *Oiseau(Titi)* est dans E_0. La négation de la justification de d_1, $\neg vole(Titi)$, n'est pas dans E_0. Donc avec la définition d'une extension, on peut ajouter *Vole(Titi)* à E_0 pour obtenir E_1.

$E_1 = Th(\{E_0) \cup Vole(Titi)\})$

- Ensuite on essaie d'utiliser d_2 pour compléter E_1. Le prérequis *Manchot(Titi)* est bien dans E_1. Mais *Vole(Titi)* qui est la négation de la justification de d_2, est dans E_1. Il est impossible d'utiliser d_2. Le calcul s'arrête et la première extension est E_1. Dans cette extension, Titi vole.

Cas 2: On va maintenant commencer la construction de l'extension en utilisant d_2

- On part de $E_0 = W = Th(\{Manchot(Titi) \rightarrow Oiseau(Titi), Manchot(Titi), Oiseau(Titi)\})$

- On utilise alors d_2. Le prérequis de d_2, *Manchot(Titi)* est dans E_0. La négation de la justification de d_2, *Vole(Titi)* n'est pas dans E_0. Donc avec la définition d'une extension, on peut ajouter $\neg Vole(Titi)$ à E_0 pour obtenir E_1.

$E_1 = Th(\{E_0) \cup \neg Vole(Titi)\})$

- Ensuite on essaie d'utiliser d_1 pour compléter E_1. Le prérequis *Manchot(Titi)* est bien dans E_1. Mais *Vole(Titi)* qui est la négation de la justification de d_2, est dans E_1. Il est impossible

d'utiliser d_2. Le calcul s'arrête et on a une deuxième extension dans laquelle Titi ne vole pas.

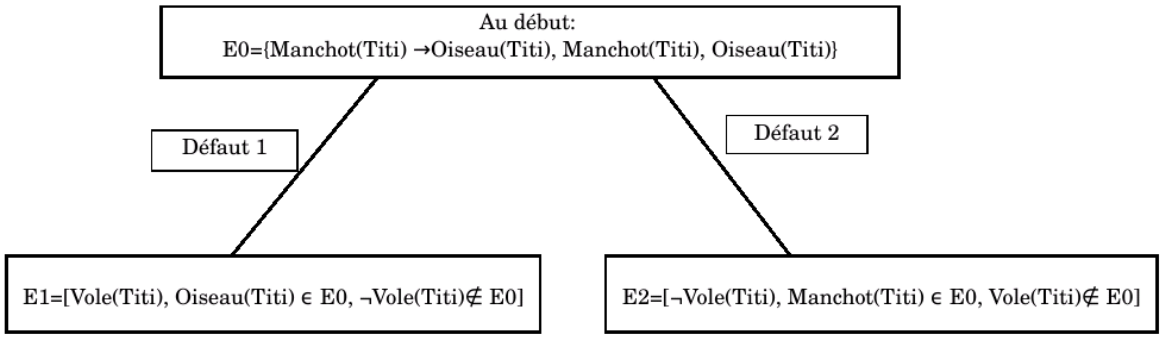

Figure 3.1. Arbre de recherche des solutions pour le calcul d'extensions

On obtient donc deux extensions qui sont contradictoires. Si nous cherchons à répondre à la question : "Est-ce que Titi vole ?", il faut pouvoir choisir entre ces deux extensions. On peut par exemple préférer les défauts les plus particuliers, ou encore établir des préférences entre les défauts.

Comme nous venons de voir dans cet exemple, les théories des défauts peuvent avoir plusieurs extensions. Il y a également des cas où elles n'ont pas d'extension. Dans certains cas, ces défauts classiques peuvent donc poser des problèmes. Mais il est démontré qu'il existe toujours une extension si *W* est satisfaisable et tous les défauts sont normaux [23].

3.6 Algorithme de calcul d'extensions.

Avant de décrire l'algorithme, on généralise la définition d'une extension en considérant qu'un défaut peu avoir plusieurs prérequis. Pour que la logique des défauts travaille dans le domaine de temps discret, il faut ajouter un argument temporal t_i. L'ensemble de faits $W = \{w_1, w_2, ...\}$ est représenté avec l'argument comme $W = \{(w_1,t_0), (w_2,t_0), ...\}$. Une extension de la théorie des défauts $\Delta = (D, W)$ est un ensemble E de formules, clos pour la déduction, contenant W et vérifiant la propriété suivante : si d est un défaut de D dont les prérequis $A(X)$ (avec t_k) sont dans E, sans que la négation des justifications $B_i(X)$ ne soient dans E, alors le conséquent $C(X)$ (avec t_{k+1}) de d est dans E. Formellement, les extensions sont définies de la façon suivante :

E est une extension de Δ si et seulement si $E = \bigcup_{i=0,\infty} E_i$, avec

$E_0 = W$ *(avec* t_0*)* et pour $i > 0$,

$$E_i = Th(E_{i-1}) \cup \{(C(X), \mathrm{t}_{k+1})/(\frac{A(X): B_j(\mathrm{X})}{C(X)}) \in D, (A(X), \mathrm{t}_k), \in E_{i-1}, \neg B_j(\mathrm{X}) \notin E_{i-1}\}$$

où $Th(E_{i-1})$ désigne l'ensemble des théorèmes obtenus de façon monotone à partir de E_{i-1}: $Th(E_{i-1}) = \{w / E_{i-1} \vdash w\}$.

Pour une théorie des défauts $\Delta = (D, W)$, avec D l'ensemble des défauts et W la base de connaissance, le calcul d'extension se fait par l'algorithme :

Entrée :

D ; (ensemble des défauts).

$E = \emptyset$; (ensemble d'extension).

Sortie : $E = \cup_{i=0,N} E_i$.

calcul_extension(E_i) : {

$E_i := W$ (en moment t_0);

tantque il y a un défaut $d = \frac{A(X):B_j(X)}{C(X)}$ qui n'a pas encore été inspecté **faire**

- Sélectionner ce défaut d,

- Vérifier que les prérequis $A(X)$ sont vrais avec E_i (en moment t_k),

- Vérifier que les justifications $B_j(X)$ sont consistantes avec E_i (utiliser la négation par échec en cas d'inconsistance),

- Ajouter $(C(X), t_{k+1}))$ à E_i

fin tantque

Fin du calcul pour une extension.

Backtracking (Suppression des $(C(X), t_{k+1})$ ajoutés à E_i).

calcul_extension(E_i).

}

Chapitre 4. Approche proposée et résultats

4.1. Utilisation de l'algorithme de production de clauses.

On revient à la carte d'interactions de Pommier. Pour passer du modèle biologique au modèle logique, il faut considérer plusieurs contraintes : comment modéliser les interactions, bien faire attention aux sens des implications logiques, respecter l'ordre chronologique préalablement défini et vérifier la cohérence des informations [15]. L'étape initiale est de définir correctement les prédicats. Ici les prédicats ont été calqués sur ceux de la carte de Pommier, à savoir *stimulation*, *phosphorylation*, *autophosphorylation*, *inhibition*, *nécessité*, *liaison*, *activation* de transcription, *dégradation* et *déphosphorylation*.

Au départ, les prédicats ont été conceptualisés, pour la majorité d'entre eux, de la façon suivante : *produit* ← *réaction*(enzyme, substrat). La *réaction* peut être *stimulation*, *phosphorylation*, *déphosphorylation*, *liaison*, *activation* ou *dégradation*. Par exemple, $_{p*}$-*Y* ← *phosphorylation(X, Y)*. Les autres prédicats qui ne sont pas de type *produit* ← *réaction(enzyme, substrat)* sont modélisés séparément :

$_{p*}$-*Y* ← *autophosphorylation(Y)*

¬*Y* ← *inhibition(X, Y)* : si *X* est vrai, alors *Y* ne peut pas l'être

Y ← *nécessité(X, Y)* : pour que *Y* soit vrai, *X* doit être vrai

En effet, certains prédicats ont été modifiés, ou bien supprimés, au cours de l'implémentation, tandis que d'autres ont été rajoutés grâce aux mises à jours effectuées dans le modèle biologique et avec l'avancement de son implémentation dans Prolog. Par exemple, les prédicats ajoutés sont *ubiquitination*, *méthylation* et *dissociation*.

Dans ce programme, *cnf(nom_de_clause, type_de_clause, [littéraux])* indique une clause. Pour la partie contenant des littéraux, chaque disjonction doit être indiquée par une virgule, une négation par un signe moins, le signe plus peut être « oublié ». Voici un exemple qu'on souhaite mettre au format de Prolog :

- D'abord, *γH2AX* se lie avec *Mdc1*.
- Ensuite, *ATM* peut phosphoryler *Mdc1* liée à *γH2AX*.
- *Mdc1*, une fois phosphorylée, va se lier avec *Rnf8*.
- Enfin, *Rnf8* va se lier avec *Ubc13*.

C'est modélisé par les équations logiques suivant les prédicats :

- *Produit(γH2AX) → liaison(γH2AX, MDC1).*
- *Liaison(γH2AX, MDC1) → produit(γH2AX/MDC1).*
- *Produit(γH2AX/MDC1) → phosphorylation(ATM, γH2AX/MDC1).*
- *Phosphorylation(ATM, γH2AX/MDC1) → produit(p*-MDC1).*
- *Produit(p*-MDC1) → liaison(p*-MDC1, MRF8).*

- *Liaison(p*-MDC1, MRF8) → produit(MRF8_liée).*
- *Produit(MRF8_liée) → liaison(MRF8_liée, UBC13).*
- *Liaison(MRF8_liée, UBC13) → produit(MRF8/UBC13)*

De façon plus intuitive, ceci peut être présenté par l'image ci – dessous :

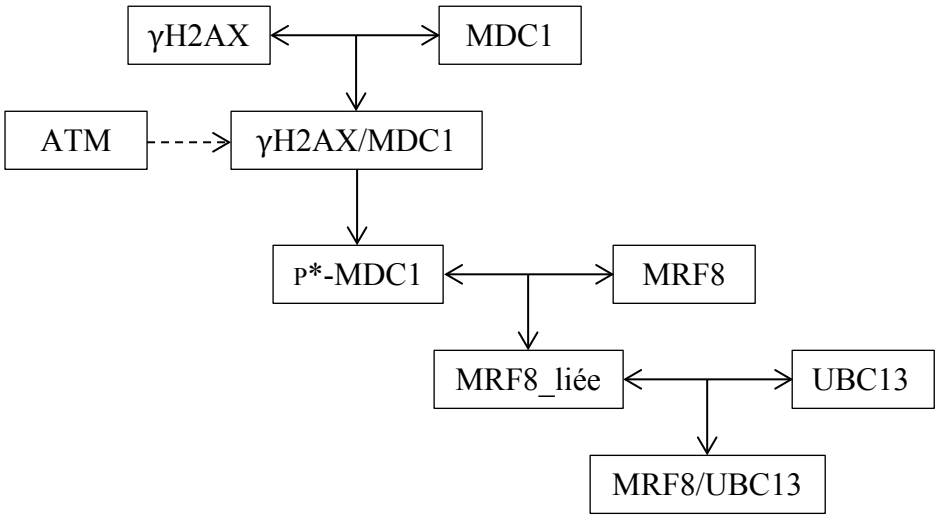

Figure 4.1. Interactions de la carte de Pommier

Dans la **Figure 4.1**, une flèche à deux têtes présente une liaison, une flèche à une tête présente une production, et une flèche brisée est une phosphorylation.

Suite, on replace les implications en utilisant les équivalences ($A \rightarrow B \equiv \neg A \vee B$) :

- *¬Produit(γH2AX) ∨ liaison(γH2AX, MDC1).*
- *¬Liaison(γH2AX, MDC1) ∨ produit(γH2AX/MDC1).*
- *¬Produit(γH2AX/MDC1) ∨ phosphorylation(ATM, γH2AX/MDC1).*
- *¬Phosphorylation(ATM, γH2AX/MDC1) ∨ produit(p*-MDC1).*
- *¬Produit(p*-MDC1) ∨ liaison(p*-MDC1, MRF8).*
- *¬Liaison(p*-MDC1, MRF8) ∨ produit(MRF8_liée).*
- *¬Produit(MRF8_liée) ∨ liaison(MRF8_liée, UBC13).*
- *¬Liaison(MRF8_liée, UBC13) ∨ produit(MRF8/UBC13)*

Enfin, on adapte les équations au format de Prolog :

- *cnf(mdc1_1, axiom, [-product(gamma_h2ax),binding(gamma_h2ax,mdc1)]).*
- *cnf(mdc1_2, axiom, [-binding(gamma_h2ax,mdc1),product(h2ax_mdc1)]).*
- *cnf(mdc1_3,axiom,[product(h2ax_mdc1), phosphorylation(p_atm_bound,h2ax_mdc1)]).*
- *cnf(mdc1_4, axiom, [-phosphorylation(p_atm_bound,h2ax_mdc1),product(p_mdc1)]).*
- *cnf(rnf_01, axiom, [-product(p_mdc1),binding(p_mdc1,rnf8)]).*
- *cnf(rnf_02, axiom, [-binding(p_mdc1,rnf8),product(rnf8_bound)]).*
- *cnf(rnf_03, axiom, [-product(rnf8_bound),binding(rnf8_bound,ubc13)]).*
- *cnf(rnf_04, axiom, [-binding(rnf8_bound,ubc13),product(rnf8_ubc13)]).*

À partir de la carte, avec la modélisation, on a adapté la base de données au format de Prolog. Les lignes ci-dessous sont celles utilisées dans le modèle et qui relatent les événements biologiques précédemment indiques :

```
/************************************/

cnf(mrn_0, axiom, [-product(dsb),stimulation(dsb,dna)]).
cnf(mrn_1, axiom, [-stimulation(dsb,dna),product(altered_dna)]).
cnf(mrn_2, axiom, [-product(altered_dna),binding(mre11,rad50)]).
cnf(mrn_3, axiom, [-binding(mre11,rad50),product(mre11_rad50)]).
cnf(mrn_4, axiom, [-product(mre11_rad50),binding(mre11_rad50,nbs1)]).
cnf(mrn_5, axiom, [-binding(mre11_rad50,nbs1),product(mrn)]).
cnf(mrn_6, axiom, [-product(mrn),binding(mrn,altered_dna)]).
cnf(mrn_7, axiom, [-binding(mrn,altered_dna),product(mrn_bound_to_dna)]).
cnf(atm_1, axiom, [-product(mrn_bound_to_dna),binding(mrn_bound_to_dna,atm_atm)]).
cnf(atm_2, axiom, [-binding(mrn_bound_to_dna,atm_atm),product(atm_bound_to_mrn)]).
cnf(atm_3, axiom, [-product(atm_bound_to_mrn),-product(atm_atm)]).
cnf(atm_4, axiom, [-product(atm_bound_to_mrn),autophosphorylation(atm_bound_to_mrn)]).
cnf(atm_5, axiom, [-autophosphorylation(atm_bound_to_mrn),product(p_atm_atm_bound)]).
cnf(atm_6, axiom, [-product(p_atm_atm_bound),-product(atm_atm)]).
cnf(atm_7, axiom, [-product(p_atm_atm_bound),dissociation(p_atm_atm_bound)]).
cnf(atm_8, axiom, [-dissociation(p_atm_atm_bound),product(p_atm_bound)]).
cnf(atm_9, axiom, [-dissociation(p_atm_atm_bound),product(p_atm_free)]).

%Chk1 Phosphorylation

cnf(chk1_1, axiom, [-product(p_atm_free),phosphorylation(p_atm_free,chk1)]).
cnf(chk1_2, axiom, [-phosphorylation(p_atm_free,chk1),product(p_chk1)]).
cnf(chk1_3, axiom, [phosphorylation(atr,chk1)]).
cnf(chk1_4, axiom, [-phosphorylation(atr,chk1),product(p_chk1)]).

%Other MRN needs

cnf(mrn_add_1, axiom, [-product(mrn_bound_to_dna),product(p_smc1)]).
cnf(mrn_add_2, axiom, [-product(mrn_bound_to_dna),product(p_mre11)]).

%gamma-H2AX

cnf(h2ax_1, axiom, [-phosphorylation(atr,h2ax),product(gamma_h2ax)]).
cnf(h2ax_2, axiom, [-phosphorylation(p_atm_bound,h2ax),product(gamma_h2ax)]).

%MDC1, Ubiquitination and BRCA1

cnf(mdc1_1, axiom, [-product(gamma_h2ax),binding(gamma_h2ax,mdc1)]).
cnf(mdc1_2, axiom, [-binding(gamma_h2ax,mdc1),product(h2ax_mdc1)]).
cnf(mdc1_3, axiom, [-product(h2ax_mdc1),phosphorylation(p_atm_bound,h2ax_mdc1)]).
cnf(mdc1_4, axiom, [-phosphorylation(p_atm_bound,h2ax_mdc1),product(p_mdc1)]).
cnf(rnf_01, axiom, [-product(p_mdc1),binding(p_mdc1,rnf8)]).
cnf(rnf_02, axiom, [-binding(p_mdc1,rnf8),product(rnf8_bound)]).
cnf(rnf_03, axiom, [-product(rnf8_bound),binding(rnf8_bound,ubc13)]).
cnf(rnf_04, axiom, [-binding(rnf8_bound,ubc13),product(rnf8_ubc13)]).
cnf(rnf_05, axiom, [-product(rnf8_ubc13),ubiquitination(rnf8_ubc13,h2a)]).
cnf(rnf_06, axiom, [-ubiquitination(rnf8_ubc13,h2a),product(ub_h2a)]).
cnf(rnf_07, axiom, [-product(ub_h2a),binding(ub_h2a,rnf168)]).
cnf(rnf_08, axiom, [-binding(ub_h2a,rnf168),product(rnf168_bound)]).
cnf(rnf_09, axiom, [-product(rnf168_bound),binding(rnf168_bound,ubc13)]).
```

cnf(rnf_10, axiom, [-binding(rnf168_bound,ubc13),product(rnf168_ubc13)]).
cnf(rnf_11, axiom, [-product(rnf168_ubc13),ubiquitination(rnf168_ubc13,h2a)]).
cnf(rnf_12, axiom, [-ubiquitination(rnf168_ubc13,h2a),product(poly_ub_h2a)]).
cnf(rnf_13, axiom, [-product(poly_ub_h2a),stimulation(poly_ub_h2a,dna)]).
cnf(rnf_14, axiom, [-stimulation(poly_ub_h2a,dna),product(changed_struct_dna)]).
cnf(rap_01, axiom, [-product(poly_ub_h2a),binding(poly_ub_h2a,rap80)]).
cnf(rap_02, axiom, [-binding(poly_ub_h2a,rap80),product(rap80_bound)]).
cnf(rap_03, axiom, [-product(rap80_bound),binding(rap80_bound,abraxas)]).
cnf(rap_04, axiom, [-binding(rap80_bound,abraxas),product(abraxas_bound)]).
cnf(rap_05, axiom, [-product(abraxas_bound),binding(abraxas_bound,bre)]).
cnf(rap_06, axiom, [-binding(abraxas_bound,bre),product(bre_bound)]).
cnf(rap_07, axiom, [-product(abraxas_bound),binding(abraxas_bound,brcc36)]).
cnf(rap_08, axiom, [-binding(abraxas_bound,brcc36),product(brcc36_bound)]).
cnf(rap_09, axiom, [-product(brcc36_bound),binding(brcc36_bound,merit40)]).
cnf(rap_10, axiom, [-binding(brcc36_bound,merit40),product(merit40_bound)]).
cnf(rap_10, axiom, [-product(merit40_bound),binding(merit40_bound,brcc36_bound)]).
cnf(rap_11, axiom, [-binding(merit40_bound,brcc36_bound),product(brcc36_merit40)]).
cnf(rap_12, axiom, [-product(brcc36_merit40),binding(brcc36_merit40,brca1)]).
cnf(rap_13, axiom, [-binding(brcc36_merit40,brca1),product(brca1_bound_to_rap80_complex)]).

%53BP1 binding

cnf(mmset_1, axiom, [-product(changed_struct_dna),phosphorylation(p_atm_bound,mmset)]).
cnf(mmset_2, axiom, [-phosphorylation(p_atm_bound,mmset),product(p_mmset)]).
cnf(mmset_3, axiom, [-product(h2ax_mdc1),binding(h2ax_mdc1,mdc1)]).
cnf(mmset_4, axiom, [-binding(h2ax_mdc1,mdc1),product(mdc1_multi)]).
cnf(mmset_5, axiom, [-product(mdc1_multi),-product(p_mmset),binding(mdc1_multi,p_mmset)]).
cnf(mmset_6, axiom, [-binding(mdc1_multi,p_mmset),product(mmset_mdc1)]).
cnf(mmset_7, axiom, [-product(mmset_mdc1),methylation(mmset_mdc1,h4)]).
cnf(mmset_8, axiom, [-methylation(mmset_mdc1,h4),product(h4k20me2)]).
cnf(p53bp1_1,axiom,[-product(h4k20me2),-product(brca1_bound_to_rap80_complex),
binding(h4k20me2,p53bp1)]).
cnf(p53bp1_2, axiom, [-binding(h4k20me2,p53bp1),product(p53bp1_bound)]).
cnf(p53bp1_3, axiom, [-product(p53bp1_bound),phosphorylation(p_atm_bound,p53bp1_bound)]).
cnf(p53bp1_4, axiom, [-phosphorylation(p_atm_bound,p53bp1_bound),product(p_53bp1)]).

%Chk2 formation with Plk3 action (and binding with 53bp1)

cnf(plk3_1, axiom, [-product(p_atm_free),phosphorylation(p_atm_free,plk3)]).
cnf(plk3_2, axiom, [-phosphorylation(p_atm_free,plk3),product(p_plk3)]).
cnf(chk2_01, axiom, [-product(p_atm_free),phosphorylation(p_atm_free,chk2)]).
cnf(chk2_02, axiom, [-phosphorylation(p_atm_free,chk2),product(p_s33_35_chk2)]).
cnf(chk2_03, axiom, [-product(p_s33_35_chk2),phosphorylation(p_plk3,p_s33_35_chk2)]).
cnf(chk2_04, axiom, [-phosphorylation(p_plk3,p_s33_35_chk2),product(p_s33_35_s62_73_chk2)]).
cnf(chk2_05,axiom,[-product(p_s33_35_s62_73_chk2),
phosphorylation(p_atm_free,p_s33_35_s62_73_chk2)]).
cnf(chk2_06, axiom, [-phosphorylation(p_atm_free,p_s33_35_s62_73_chk2),product(p_t68_chk2)]).
cnf(chk2_07,axiom,[-product(p_s33_35_s62_73_chk2),
phosphorylation(atr,p_s33_35_s62_73_chk2)]).
cnf(chk2_08, axiom, [-phosphorylation(atr,p_s33_35_s62_73_chk2),product(p_t68_chk2)]).
cnf(chk2_09, axiom, [-product(p_t68_chk2),binding(p_t68_chk2,p_t68_chk2)]).
cnf(chk2_10, axiom, [-binding(p_t68_chk2,p_t68_chk2),product(chk2_chk2)]).
cnf(chk2_11, axiom, [-product(chk2_chk2),autophosphorylation(chk2_chk2)]).
cnf(chk2_12, axiom, [-autophosphorylation(chk2_chk2),product(p_active_chk2_chk2)]).
cnf(chk2_13,axiom,[-product(p_active_chk2_chk2),-product(p_53bp1),
binding(p_active_chk2_chk2,p_53bp1)]).

cnf(chk2_13, axiom, [-binding(p_active_chk2_chk2,p_53bp1),product(chk2_53bp1)]).

%BRCA1 regulation by CtIP and Chk2

cnf(ctip_1, axiom, [binding(brca1,ctip)]).
cnf(ctip_2, axiom, [-binding(brca1,ctip),product(brca1_ctip)]).
cnf(ctip_3, axiom, [-product(brca1_ctip),-product(chk2_53bp1),binding(brca1_ctip,chk2_53bp1)]).
cnf(ctip_4, axiom, [-binding(brca1_ctip,chk2_53bp1),product(chk2_53bp1_bound_to_brca1)]).
cnf(ctip_5, axiom, [-binding(brca1_ctip,chk2_53bp1),product(brca1_ctip_bound_to_chk2)]).
cnf(ctip_6,axiom,[-product(chk2_53bp1_bound_to_brca1),-product(brca1_ctip_bound_to_chk2),
phosphorylation(chk2_53bp1_bound_to_brca1,brca1_ctip_bound_to_chk2)]).
cnf(ctip_7,axiom,[-phosphorylation(chk2_53bp1_bound_to_brca1,brca1_ctip_bound_to_chk2),
product(p_brca1_ctip_bound_to_chk2)]).
cnf(ctip_8, axiom, [-product(p_brca1_ctip_bound_to_chk2),product(p_s988_brca_ctip)]).
cnf(ctip_9, axiom, [-product(chk2_53bp1_bound_to_brca1),product(chk2_53bp1)]).
cnf(brca1_0,axiom,[-product(p_s988_brca_ctip), phosphorylation(p_atm_bound,p_s988_brca_ctip)]).
cnf(brca1_1, axiom, [-phosphorylation(p_atm_bound,p_s988_brca_ctip),product(brca_p_ctip)]).
cnf(brca1_2, axiom, [-product(brca_p_ctip),-product(brca1_ctip)]).
cnf(brca1_3, axiom, [-product(brca_p_ctip),dissociation(brca_p_ctip)]).
cnf(brca1_4, axiom, [-dissociation(brca_p_ctip),product(brca1)]).
cnf(brca1_5, axiom, [-dissociation(brca_p_ctip),product(p_ctip)]).
cnf(brca1_6, axiom, [-product(brca1),phosphorylation(p_atm_bound,brca1)]).
cnf(brca1_7, axiom, [-phosphorylation(p_atm_bound,brca1),product(p_brca1)]).
cnf(brca1_8, axiom, [-product(brca1),phosphorylation(atr,brca1)]).
cnf(brca1_9, axiom, [-phosphorylation(atr,brca1),product(p_brca1)]).

%DNA repair proposed mecasism

cnf(repa_1, axiom, [-product(p_brca1),-product(p_53bp1),binding(mrn_bound_to_dna,p_brca1)]).
cnf(repa_2, axiom, [-binding(mrn_bound_to_dna,p_brca1),product(brca1_bound_to_mrn)]).

%p53 pathway and regulation including PML

cnf(pml_1, axiom, [-product(chk2_53bp1),phosphorylation(chk2_53bp1,pml)]).
cnf(pml_2, axiom, [-phosphorylation(chk2_53bp1,pml),product(p_pml)]).
cnf(pml_3, axiom, [-product(p_pml),stimulation(p_pml,cell)]).
cnf(pml_4, axiom, [-stimulation(p_pml,cell),product(apoptosis)]).
cnf(pml_5, axiom, [-product(p_pml),binding(p_pml,mdm2)]).
cnf(pml_6, axiom, [-binding(p_pml,mdm2),product(pml_mdm2)]).
cnf(pml_7, axiom, [-product(pml_mdm2),-binding(mdm2,p53)]).
cnf(mdm2_1, axiom, [product(p53)]).
cnf(mdm2_2, axiom, [-product(mdm2),-product(p53),binding(mdm2,p53)]).
cnf(mdm2_3, axiom, [-binding(mdm2,p53),product(mdm2_p53)]).
cnf(mdm2_4, axiom, [-product(mdm2_p53),stimulation(p53_degradation_effectors,mdm2_p53)]).
cnf(mdm2_5,axiom,[-stimulation(p53_degradation_effectors,mdm2_p53),
product(p53_degradation)]).
cnf(p53_01,axiom,[-product(p_atm_free),-product(mdm2_p53),
phosphorylation(p_atm_free,mdm2_p53)]).
cnf(p53_02, axiom, [-phosphorylation(p_atm_free,mdm2_p53),product(p_s15_p53_mdm2)]).
cnf(p53_03, axiom, [-product(mdm2_p53),phosphorylation(atr,mdm2_p53)]).
cnf(p53_04, axiom, [-phosphorylation(atr,mdm2_p53),product(p_s15_p53_mdm2)]).
cnf(p53_05,axiom,[-product(p_s15_p53_mdm2),-product(chk2_53bp1),
phosphorylation(chk2_53bp1,p_s15_p53_mdm2)]).
cnf(p53_06, axiom, [-phosphorylation(chk2_53bp1,p_s15_p53_mdm2),product(p_p_p53_mdm2)]).
cnf(p53_07,axiom,[-product(p_s15_p53_mdm2),-product(p_chk1),
phosphorylation(p_chk1,p_s15_p53_mdm2)]).

cnf(p53_08, axiom, [-phosphorylation(p_chk1,p_s15_p53_mdm2),product(p_p_p53_mdm2)]).
cnf(p53_09, axiom, [-product(p_s15_p53_mdm2),-product(p53_degradation)]).
cnf(p53_10, axiom, [-product(p_p_p53_mdm2),-product(p53_degradation)]).
cnf(p53_11, axiom, [-product(p_p_p53_mdm2),dissociation(p_p_p53_mdm2)]).
cnf(p53_12, axiom, [-dissociation(p_p_p53_mdm2),product(p_p_p53)]).
cnf(p53_13, axiom, [-dissociation(p_p_p53_mdm2),product(mdm2)]).
cnf(p53_14, axiom, [-product(p_atm_free),-product(mdm2),phosphorylation(p_atm_free,mdm2)]).
cnf(p53_15, axiom, [-phosphorylation(p_atm_free,mdm2),product(p_mdm2)]).
cnf(p53_16, axiom, [-product(p_p_p53),binding(p300,p_p_p53)]).
cnf(p53_17, axiom, [-product(p_pml),-binding(p300,p_p_p53),product(active_p53)]).
cnf(p53_18,axiom,[-product(active_p53), transcription_activation(active_p53,prom_p21_gadd45)]).
cnf(p53_19,axiom,[-transcription_activation(active_p53,prom_p21_gadd45),
product(p21_and_gadd45)]).
cnf(p53_19a, axiom, [-product(p21_and_gadd45),stimulation(p21_and_gadd45,cell_cycle)]).
cnf(p53_20, axiom, [-stimulation(p21_and_gadd45,cell_cycle),product(cell_cycle_arrest)]).
cnf(p53_21, axiom, [-product(active_p53),transcription_activation(active_p53,prom_bnpf)]).
cnf(p53_22,axiom,[-transcription_activation(active_p53,prom_bnpf), product(box_nas_puma_fas)]).
cnf(p53_23, axiom, [-product(box_nas_puma_fas),stimulation(box_nas_puma_fas,cell)]).
cnf(p53_24, axiom, [-stimulation(box_nas_puma_fas,cell),product(apoptosis)]).
cnf(p53_21, axiom, [-product(active_p53),transcription_activation(active_p53,prom_mdm2)]).
cnf(p53_22, axiom, [-transcription_activation(active_p53,prom_mdm2),product(mdm2)]).

%E2F1 action

cnf(e2f1_00, axiom, [product(e2f1)]).
cnf(e2f1_01, axiom, [-product(e2f1),-product(chk2_53bp1),phosphorylation(chk2_53bp1,e2f1)]).
cnf(e2f1_02, axiom, [-phosphorylation(chk2_53bp1,e2f1),product(p_e2f1)]).
cnf(e2f1_03, axiom, [-product(e2f1),stimulation(e2f1_degradation_effectors,e2f1)]).
cnf(e2f1_04, axiom, [-stimulation(e2f1_degradation_effectors,e2f1),product(e2f1_degradation)]).
cnf(e2f1_05, axiom, [-product(p_e2f1),-product(e2f1_degradation)]).
cnf(e2f1_06, axiom, [-product(p_e2f1),phosphorylation(p_atm_free,p_e2f1)]).
cnf(e2f1_07, axiom, [-phosphorylation(p_atm_free,p_e2f1),product(p_p_e2f1)]).
cnf(e2f1_08, axiom, [-product(p_e2f1),phosphorylation(atr,p_e2f1)]).
cnf(e2f1_09, axiom, [-phosphorylation(atr,p_e2f1),product(p_p_e2f1)]).
cnf(e2f1_10, axiom, [-product(p_p_e2f1),transcription_activation(p_p_e2f1,prom_chk2)]).
cnf(e2f1_11, axiom, [-transcription_activation(p_p_e2f1,prom_chk2),product(chk2)]).
cnf(e2f1_12, axiom, [-product(p_p_e2f1),transcription_activation(p_p_e2f1,prom_arf)]).
cnf(e2f1_13, axiom, [-transcription_activation(p_p_e2f1,prom_arf),product(arf)]).
cnf(e2f1_14, axiom, [-product(p_p_e2f1),transcription_activation(p_p_e2f1,prom_p73_apaf1)]).
cnf(e2f1_15, axiom, [-transcription_activation(p_p_e2f1,prom_p73_apaf1),product(p73_apaf1)]).
cnf(e2f1_16, axiom, [-product(p73_apaf1),stimulation(p73_apaf1,cell)]).
cnf(e2f1_17, axiom, [-stimulation(p73_apaf1,cell),product(apoptosis)]).
cnf(e2f1_18, axiom, [-product(arf),binding(arf,mdm2)]).
cnf(e2f1_19, axiom, [-binding(arf,mdm2),product(arf_mdm2)]).
cnf(e2f1_20, axiom, [-product(arf_mdm2),-product(mdm2_p53)]).
cnf(e2f1_21, axiom, [-product(p_p_e2f1),stimulation(p_p_e2f1,unknown_atm_way)]).
cnf(e2f1_22, axiom, [-stimulation(p_p_e2f1,unknown_atm_way),product(p_atm_free)]).

%p38 phosphorylation (usefull for cdc25c and cdc25b)

cnf(p38_1, axiom, [-product(p_atm_free),phosphorylation(p_atm_free,p38)]).
cnf(p38_2, axiom, [-phosphorylation(p_atm_free,p38),product(p_p38)]).

%Cdc25 regulation

cnf(cdc25a_0, axiom, [-stimulation(cdc25a,cell),-product(cell_cycle_arrest)]).

```
cnf(cdc25a_1, axiom, [-product(chk2_53bp1),phosphorylation(chk2_53bp1,cdc25a)]).
cnf(cdc25a_2, axiom, [-phosphorylation(chk2_53bp1,cdc25a),product(p_cdc25a)]).
cnf(cdc25a_3, axiom, [-product(p_chk1),phosphorylation(p_chk1,cdc25a)]).
cnf(cdc25a_4, axiom, [-phosphorylation(p_chk1,cdc25a),product(p_cdc25a)]).
cnf(cdc25a_5, axiom, [-product(p_cdc25a),stimulation(p_cdc25a,cdc25a_degradation_effectors)]).
cnf(cdc25a_6,axiom,[-stimulation(p_cdc25a,cdc25a_degradation_effectors),
      product(cdc25a_degradation)]).
cnf(cdc25a_7, axiom, [-product(cdc25a_degradation),-stimulation(cdc25a,cell)]).
cnf(cdc25c_00, axiom, [-stimulation(cdc25c,cell),-product(cell_cycle_arrest)]).
cnf(cdc25c_01, axiom, [-product(chk2_53bp1),phosphorylation(chk2_53bp1,cdc25c)]).
cnf(cdc25c_02, axiom, [-phosphorylation(chk2_53bp1,cdc25c),product(p_cdc25c)]).
cnf(cdc25c_03, axiom, [-product(p_plk3),phosphorylation(p_plk3,cdc25c)]).
cnf(cdc25c_04, axiom, [-phosphorylation(p_plk3,cdc25c),product(p_cdc25c)]).
cnf(cdc25c_05, axiom, [-product(p_chk1),phosphorylation(p_chk1,cdc25c)]).
cnf(cdc25c_06, axiom, [-phosphorylation(p_chk1,cdc25c),product(p_cdc25c)]).
cnf(cdc25c_07, axiom, [-product(p_p38),phosphorylation(p_p38,cdc25c)]).
cnf(cdc25c_08, axiom, [-phosphorylation(p_p38,cdc25c),product(p_cdc25c)]).
cnf(cdc25c_09, axiom, [-product(p_cdc25c),stimulation(p_cdc25c,cdc25c_degradation_effectors)]).
cnf(cdc25c_10,axiom,[-stimulation(p_cdc25c,cdc25c_degradation_effectors),
      product(cdc25c_degradation)]).
cnf(cdc25c_11, axiom, [-product(cdc25c_degradation),-stimulation(cdc25c,cell)]).
cnf(cdc25b_1, axiom, [-stimulation(cdc25b,cell),-product(cell_cycle_arrest)]).
cnf(cdc25b_2, axiom, [-product(p_p38),phosphorylation(p_p38,cdc25b)]).
cnf(cdc25b_3, axiom, [-phosphorylation(p_p38,cdc25b),product(p_cdc25b)]).
cnf(cdc25b_4, axiom, [-product(p_cdc25b),-stimulation(cdc25b,cell)]).

%JNK actions (mechanisms quite unclear)

cnf(jnk_1, axiom, [-stimulation(jnk,cdc25c),product(p_cdc25c)]).
cnf(jnk_2, axiom, [-phosphorylation(jnk,p_s15_p53_mdm2),product(p_p_p53_mdm2)]).

%FANCD2 first steps

cnf(fancd2_1, axiom, [-product(p_atm_bound),phosphorylation(p_atm_bound,nbs1)]).
cnf(fancd2_2, axiom, [-phosphorylation(p_atm_bound,nbs1),product(p_nbs1)]).
cnf(fancd2_3, axiom, [-product(p_nbs1),phosphorylation(p_atm_bound,fancd2)]).
cnf(fancd2_4, axiom, [-phosphorylation(p_atm_bound,fancd2),product(p_fancd2)]).
cnf(fancd2_5, axiom, [-product(p_fancd2),stimulation(p_fancd2,cell)]).
cnf(fancd2_6, axiom, [-stimulation(p_fancd2,cell),-product(cell_cycle_arrest)]).

%Mus81 actions on DNA Repair

cnf(mus81_1, axiom, [-product(chk2_53bp1),binding(chk2_53bp1,mus81)]).
cnf(mus81_2, axiom, [-binding(chk2_53bp1,mus81),product(mus81_chk2)]).
cnf(mus81_3, axiom, [-product(mus81_chk2),binding(mus81_chk2,altered_dna)]).
cnf(mus81_4, axiom, [-binding(mus81_chk2,altered_dna),product(mus81_bound_to_dna)]).
%cnf(mus81_5, axiom, [-binding(mus81_chk2,altered_dna),product(mus81_bound_to_dna)]).

/**************************************/
```

Le résultat : Avec le champ de production *champ([_,1,[stimulation(_,_)],_])*, après l'exécution de programme, le résultat est donné

Pour activer product(cell_cycle_arrest):

-stimulation(p21_and_gadd45,cell_cycle)

% 261,290 inferences, 0.156 CPU in 20.794 seconds (1% CPU, 1674925 Lips)

Pour bloquer product(cell_cycle_arrest) :

-stimulation(cdc25a,cell)

-stimulation(cdc25c,cell)

-stimulation(cdc25b,cell)

-stimulation(dsb,dna)

-stimulation(p_fancd2,cell)

% 18,008 inferences, 0.047 CPU in 7.993 seconds (1% CPU, 384784 Lips)

Pour activer product(apoptosis) :

-stimulation(p_pml,cell)

-stimulation(box_nas_puma_fas,cell)

-stimulation(p73_apaf1,cell)

% 154,139 inferences, 0.094 CPU in 20.310 seconds (0% CPU, 1646774 Lips)

Pour bloquer product(apoptosis), le résultat est un ensemble vide de clauses.

4.2. Utilisation de la logique des défauts

4.2.1. Dans le cas général

La **Figure 4.2** représente un exemple très simplifié d'interactions dans une cellule.

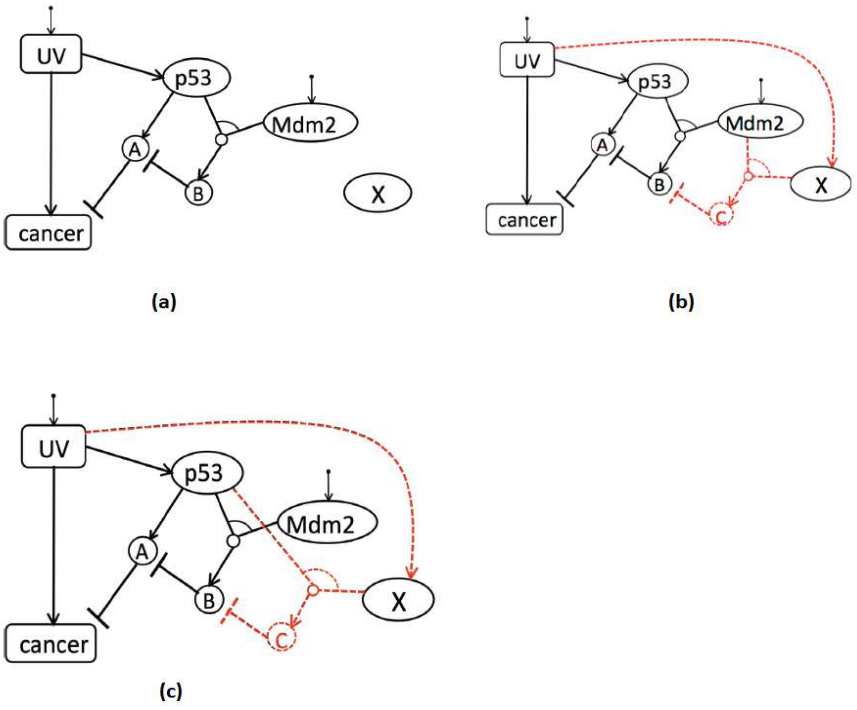

Figure 4.2. Un exemple d'interactions dans une cellule

Par des mécanismes divers non indiqués ici, les ultraviolets (*UV*) mettent la cellule en apostase (elle devient de fait immortelle) d'où le cancer. Ceci est représenté par une flèche. D'un autre coté les *UV* activent la production de la protéine *p53*. Cette protéine va activer une protéine *A* qui va bloquer le cancer. Mais *p53* liée à la protéine *Mdm2* va produire *B*, qui va bloquer *A*. Pour un biologiste, la question est donc de bloquer le cancer en bloquant *B*. Les expériences biologiques ont montré que le *X* pourrait être un candidat pour ce blocage. Les figures (*b*) et (*c*) donnent deux types d'interactions possibles avec *X* pour expliquer ce blocage de *B*. Le problème est d'utiliser l'informatique pour compléter le graphe [3].

Pour décrire les interactions entre les gênes dans la figure, on part d'un langage *L* de logique classique (propositionnel ou du premier ordre). Dans *L*, la proposition *A* (resp $\neg A$) signifie que *A* est *vrai* (*faux*). Les interactions entres les gênes sont une forme très simple de causalité. Pour exprimer ces interactions il est courant d'aller à l'essentiel en donnant deux relations binaires *cause(A, B) et bloque(A, B)*. La première relation veut dire, par exemple, qu'une protéine *A* déclenche la production d'une protéine *B*, la deuxième est l'inhibition. De manière classique, ces relations sont représentées dans le réseau de gênes, par $A \rightarrow B$ et $A \dashv B$. Bien entendu, ces causalités sont élémentaires et de nombreux travaux très savants ont été écrits pour les représenter.

Pour représenter les liens entre les relations, dans le langage *L*, il faut faire deux choses : décrire les propriétés internes aux relations *cause* et *bloque* ; décrire les liens entre ces relations.

Les premières propriétés que l'on a envie de donner peuvent s'exprimer naturellement, par des règles du type :

(1) Si *A cause B* et si *A* est vrai, alors *B* est vrai.

(2) Si *A bloque B* et si *A* est vrai, alors *B* est faux.

La solution élémentaire est alors de donner explicitement les deux schémas d'axiomes :

(C1) $cause(A,B) \wedge cause(B, C) \rightarrow cause(A, C)$

(C2) $cause(A, B) \wedge bloque(B, C) \rightarrow bloque(A, C)$

La première idée est d'exprimer ces lois en logique classique par les axiomes :

$cause(A, B) \wedge A \rightarrow B$

$bloque(A, B) \wedge A \rightarrow \neg B$

On peut aussi les exprimer plus faiblement par des règles d'inférences proches du modus ponens :

$cause(A, B) \wedge A \ / \ B$

$bloque(A, B) \wedge A \ / \neg B$

Mais ces deux formulations posent problème dès qu'il y a conflit. Si par exemple on a un ensemble F de quatre formules $F = \{A, B, cause(A, C), bloque(B, C)\}$, alors on va dans les deux approches données ci-dessus inférer de F, C et $\neg C$ ce qui est inconsistant. Pour résoudre ce conflit, on peut utiliser la logique des défauts.

Pour résoudre les conflits vus ci-dessus, l'idée intuitive est d'affaiblir la formulation des règles de causalité en :

(1') Si *A cause B*, et si *A* est vrai, et s'il est *possible* que *B* est vrai, alors *B* est *vrai*.

(2') Si *A bloque B*, et si *A* est vrai, et s'il est *possible* que *B* est faux, alors *B* est *faux*.

Dans la logique des défauts, les règles (1') et (2') vont s'exprimer intuitivement.

(1") Si *A cause B*, et si *A* est vrai, et si *B n'est pas contradictoire*, alors *B* est vrai.

(2") Si *A bloque B*, et si *A* est vrai, et si $\neg B$ *n'est pas contradictoire,* alors $\neg B$ est vrai.

Ces règles vont se représenter par des défauts normaux et s'écrire :

$d_1 : cause(A, B) \wedge A : B / B$

$d_2 : bloque(A, B) \wedge A : \neg B / \neg B$

Pour le cas élémentaire qui précède, si *A* et *B* sont vrais, on a :

$W = \{A, B, cause(A,C), bloque(B,C)\}$

$D = \{d_1, d_2\}$

et on obtient deux extensions :

E_1 contient C (en appliquant d_1)

E_2 contient $\neg C$ (en appliquant d_2)

Le conflit est donc résolu, mais se pose le problème des extensions à préférer ; est-ce que *C* est induit ou bloqué? En fait ceci va vraiment dépendre du contexte. On peut par exemple préférer les interactions positives par rapport aux négatives ; ou bien utiliser des méthodes statistiques ou probabilistes.

Plus en détail, pour l'application, on a [28]:

$W = \{UV, Mdm2\}$

$D = \{ \quad d_1 = UV : cancer / cancer$

$d_2 = UV : p53 / p53$

$d_3 = p53 : A / A$

$d_4 = p53 \wedge Mdm2 : B / B$

$d_5 = B : \neg A / \neg A$

$d_6 = A : \neg cancer / \neg cancer$

$d_7 = C : \neg B / \neg B$

}

On a encore deux défauts généraux :

$d_8 = Y : x / x ;$ où $Y \in \{UV, Mdm2\}$

et $d_9 = Z \wedge x: C / C;$ où $Z \in \{UV, Mdm2, p53, A, B\}$

Comme on a dit précédemment, pour que l'algorithme travaille avec des variables, il faut l'améliorer. Dans ce cas, les défauts d_8 et d_9 peuvent être écrits :

```
source(uv).                 % UV est une source
source(mdm2).
substance(uv).              % UV est une substance
substance(mdm2).
substance(p53).
substance(a).
substance(b).
ecrit:
    source(Y),              % « ordre 1 »
    assert(cl(def(Y,x))),   % « ordre 2 »
    substance(Z),           % « ordre 3 »
    assert(cl(def(joint(Z,x),c))).  % « ordre 4 »
```

où « ordre 1 » et « ordre 2 » peuvent dire : pour toutes les sources *Y*, ajouter les défauts *Y:x/x* à la base de données ; « ordre 3 » et « ordre 4 » peuvent dire : pour toutes les substances *Z*, ajouter les défauts $Z \wedge x:C/C$ à la base de données.

En utilisant l'algorithme de calculer des extensions, on a 18 extensions ci-dessous (l'extension 12 correspond à la **Figure 4.2**-b, et l'extension 13 correspond à la **Figure 4.2**-c) :

EXTENSION 1:
uv -> p53
p53 -> a

joint(p53,mdm2) -> b
uv -> x
joint(uv,x) -> c
a -> -cancer

EXTENSION 2:
uv -> p53
p53 -> a
joint(p53,mdm2) -> b
uv -> x
joint(mdm2,x) -> c
a -> -cancer

EXTENSION 3:
uv -> p53
p53 -> a
joint(p53,mdm2) -> b
uv -> x
joint(p53,x) -> c
a -> -cancer

EXTENSION 4:
uv -> p53
p53 -> a
joint(p53,mdm2) -> b
uv -> x
joint(a,x) -> c
a -> -cancer

EXTENSION 5:
uv -> p53
p53 -> a
joint(p53,mdm2) -> b
uv -> x
joint(b,x) -> c
a -> -cancer

EXTENSION 6:
uv -> p53
p53 -> a
joint(p53,mdm2) -> b
mdm2 -> x
joint(uv,x) -> c
a -> -cancer

EXTENSION 7:
uv -> p53
p53 -> a
joint(p53,mdm2) -> b
mdm2 -> x
joint(mdm2,x) -> c
a -> -cancer

EXTENSION 8:
uv -> p53
p53 -> a
joint(p53,mdm2) -> b
mdm2 -> x
joint(p53,x) -> c
a -> -cancer

EXTENSION 9:
uv -> p53
p53 -> a
joint(p53,mdm2) -> b
mdm2 -> x
joint(a,x) -> c
a -> -cancer

EXTENSION 10:
uv -> p53
p53 -> a
joint(p53,mdm2) -> b
mdm2 -> x
joint(b,x) -> c
a -> -cancer

EXTENSION 11:
uv -> p53
p53 -> a
uv -> x
joint(uv,x) -> c
c -> -b
a -> -cancer

EXTENSION 12:
uv -> p53
p53 -> a
uv -> x
joint(mdm2,x) -> c

c -> -b
a -> -cancer

EXTENSION 13:
uv -> p53
p53 -> a
uv -> x
joint(p53,x) -> c
c -> -b
a -> -cancer

EXTENSION 14:
uv -> p53
p53 -> a
uv -> x
joint(a,x) -> c
c -> -b
a -> -cancer

EXTENSION 15:
uv -> p53
p53 -> a
mdm2 -> x
joint(uv,x) -> c
c -> -b
a -> -cancer

EXTENSION 16:
uv -> p53
p53 -> a
mdm2 -> x
joint(mdm2,x) -> c
c -> -b
a -> -cancer

EXTENSION 17:
uv -> p53
p53 -> a
mdm2 -> x
joint(p53,x) -> c
c -> -b
a -> -cancer

EXTENSION 18:
uv -> p53

p53 -> a
mdm2 -> x
joint(a,x) -> c
c -> -b
a -> -cancer
% 599,907 inferences, 1.700 CPU in 2.402 seconds (71% CPU, 352801 Lips)

4.2.2. Utilisation dans le domaine temporel discret

Bien que les cellules aient des morphologies et des structures différentes et que leurs rôles dans les organismes différents soient variés, leurs fonctionnalités de base sont les mêmes [1,4]. Une de ces fonctionnalités est d'assurer la survie de la cellule. Cette activité dans son ensemble peut être résumée en deux points [21]. Premièrement, une cellule a besoin de trouver l'énergie nécessaire pour son activité. Cette énergie est principalement obtenue par la dégradation des ressources minérales ou organiques, c'est le catabolisme. Deuxièmement, les cellules doivent fabriquer les molécules simples nécessaires à leur survie, c'est l'anabolisme. Ces deux grandes activités sont regroupées sous le nom de métabolisme, et résultent d'un grand nombre de mécanismes et de réactions biochimiques. La plupart des réactions, qui se déroulent dans une cellule, sont catalysées par des molécules spéciales appelées enzymes. Tout ceci est représenté comme un réseau, appelé voie métabolique [20].

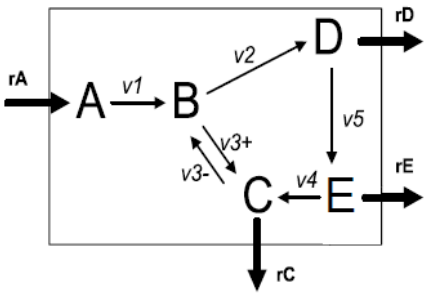

Figure 4.3. Système équilibré de masse

La **Figure 4.3** représente la topologie simple d'une voie métabolique dans une cellule, qui se compose de cinq métabolites *A, B, C, D* et *E* et six réactions, dont chacune relie deux métabolites. Chaque flux est placé sur la réaction correspondante. Bien que les concentrations de *A, C, D* et *E* soient expérimentalement mesurables, la concentration de *B* ne peut pas être mesurée. Par conséquent, *B* est le métabolite intracellulaire. Sur la base de la cinétique enzymatique, le comportement dynamique du flux d'une réaction enzymatique peut être représenté comme l'équation différentielle suivante:

$$\frac{dC_x}{dt} = v_{in} - v_{out} - \mu C_x \qquad (4.2)$$

où C_x est la concentration d'un métabolite X, v_{in} (resp. v_{out}) est la somme des flux de réactions pour la production (resp. consommation) de X, et μC_X représente le taux de croissance de la biomasse dans une cellule. Si tous les métabolites sont à l'état stationnaire, le terme de gauche de l'équation (4.2) doit être nul car il n'y a pas de changements de séries chronologiques des concentrations, et aussi, on peut supposer que la dilution des éléments à cause de la croissance

de la biomasse (correspondant à la durée de la dernière équation (4.2)) est négligée. Ce fait signifie que pour chaque métabolite *X*, les flux de consommation de *X* sont équilibrés avec les flux de production de *X* à l'état stationnaire. L'équilibrage de flux métabolique est basé sur cette notion simple. Par exemple, son équilibrage dans la **Figure 4.3** peut être représenté comme les équations linéaires suivantes:

$$v_1 = rA, v_2 + v_{3^+} = v_{3^-} + v_1, v_5 = v_2 + rD, v_4 + rE = v_{5,} v_{3^-} + rC = v_{3^+} + v_4 \quad (4.3)$$

Ensuite, nous pouvons analyser la distribution du flux sur la base des équations (4.3) avec les flux mesurables *rA, rC, rD* et *rE*. En général, ces équations ne peuvent pas être résolues de façon déterministe parce que le nombre de valeurs inconnues comme $v_1, \ldots, v_5$ correspondant aux flux de réactions enzymatiques intracellulaires devient plus grand que le nombre de valeurs connues correspondant aux flux mesurables. Des méthodes ont été proposées précédemment telles que l'analyse en mode primaire et extrêmes de la voie, des fonctions d'analyse d'utilisation d'optimisation afin de résoudre les équations. Ces fonctions introduites sont généralement construites en supposant que la maximisation de la croissance cellulaire ou la minimisation de la consommation d'énergie. Toutefois, dans le cas d'une grande échelle de la voie métabolique, nous ne pouvons pas résoudre la distribution de flux avec ces méthodes d'approximation en raison du coût de calcul énorme.

Le système métabolique cellulaire dispose d'un mécanisme sophistiqué pour contrôler dynamiquement les activités des enzymes pour répondre aux besoins d'une cellule. Ce mécanisme de régulation peut être représenté comme des relations causales entre les activités enzymatiques et la concentration de métabolites changeants. Ici, nous considérons trois voies métaboliques simples: la première se compose de deux réactions avec trois métabolites, la seconde se compose de deux réactions avec trois métabolites aussi, et la troisième se compose d'une réaction avec deux métabolites. Noter que dans les figures suivantes, nous décrivons activé et non-activé par des réactions comme les cercles et barres obliques plus flèches correspondant aux réactions, respectivement. Et aussi, une hausse (resp. baisse) flèche représente l'augmentation (resp. la diminution) à une concentration de métabolites.

La **figure 4.4** correspond à la voie métabolique composée de trois métabolites *X, Y* et *Z*, et deux réactions. La figure dit que si la concentration de *Y* tend à augmenter à un certain moment, et si la réaction enzymatique *Y*→*X* (resp. *X*→*Z*) est activée (resp. non-activée) dans un état, alors la concentration de *X* va aussi augmenter. Cette relation causale est prise en compte par l'équation (4.2). Supposons que l'augmentation de la concentration de *X* est observée, ce qui est représentée par une flèche en pointillés dans les figures. Un cas possible est que la concentration de *Y* augmente, la réaction de *Y*→*X* est activée et la réaction *X*→*Z* n'est pas activée. Ceci parce que la production de *X* à partir de *Y* ne peut pas être consommée pour générer le *Z*.

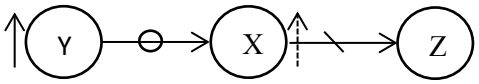

Figure 4.4. La première relation entre les états de réaction et les changements de concentration de métabolites

La **Figure 4.5** correspond à la voie métabolique composée de trois métabolites *X, Y* et *Z*, et deux réactions. La figure montre que si la concentration de *Y* tend à augmenter à un certain moment, et si la réaction enzymatique $Y \rightarrow X$ (resp. $X \rightarrow Z$) est non-activée (resp. activée) dans un l'état, alors la concentration de *X* va diminuer. Cette relation causale est prise en compte par l'équation (4.2). Supposons que la diminution de la concentration de *X* est observée, ce qui est désignée par une flèche en pointillés dans les figures. Un cas possible est que la concentration de *Y* augmente, la réaction de $Y \rightarrow X$ n'est pas activée et la réaction $X \rightarrow Z$ est activée. C'est parce que la production de *Z* à partir de *X* peut être consommée.

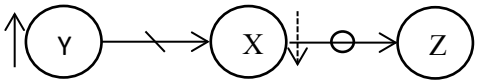

Figure 4.5. La deuxième relation entre les états de réaction et les changements de concentration de métabolites

Ensuite, nous considérons la **Figure 4.6** qui représente une voie métabolique composée de deux métabolites *X* et *Y*, et une réaction. La figure montre que, même si la réaction $Y \rightarrow X$ est activée, la concentration de *X* doit diminuer en ce que la concentration de *Y* diminue. Par conséquent, si nous observons que la concentration de *X* diminue, on peut supposer une diminution de la concentration de *Y* et la réaction $Y \rightarrow X$ est activée comme un cas possible.

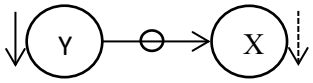

Figure 4.6. La troisième relation entre les états de réaction et les changements de concentration de métabolites

Les métabolites et les protéines sont considérés comme un même objet. On va souvent se restreindre ici à une représentation propositionnelle. Dans la pratique l'étude fine des interactions va demander de représenter des augmentations ou des diminutions de concentration de protéines. On sort donc du cadre propositionnel mais les problèmes de base sont les mêmes. Pour représenter une variation de concentration il est par exemple possible d'utiliser des prédicats de type "augmente" ou "diminue" et de limiter l'utilisation de ces prédicats.

Pour décrire les interactions entre les métabolites dans la cellule, on part d'un langage *L* de logique classique (propositionnel ou du premier ordre). Dans *L*, la proposition *A* (resp $\neg A$) signifie que *A* est vrai (faux). On pourra utiliser par exemple $con(A, up, t_i)$ pour dire que la

concentration de *A* est augmentée au moment t_i, ou encore, *con(A, down, t_j)* pour dire que la concentration de *A* est diminuée au moment t_j. On est dans un cadre logique, donc il est possible de représenter à peu près tout ce que l'on veut de manière naturelle. Le prix à payer, si l'on utilise la totalité du langage peut être l'explosion combinatoire des algorithmes.

Les réactions entres métabolites sont une forme très simple de causalité. Pour exprimer ces réactions, il est courant d'aller à l'essentiel en donnant deux relations binaires *act(A, B)* et ¬*act(A, B)*. La première relation veut dire, par exemple, qu'un métabolite *A* est consommé pour la production d'un métabolite *B*, la deuxième n'est pas consommé. De manière classique ces relations sont représentées dans le réseau de gênes, par $A \rightarrow B$ et $A \dashv B$. Bien entendu cette causalité est élémentaire et de nombreux travaux très savants ont été écrits pour représenter les causalités.

(C1) $con(A,up,t_i) \wedge react(A,B) \wedge active(A,B) \wedge react(B,C) \wedge non\_active(B,C) \rightarrow con(B,up,t_{i+1})$

(C2) $con(A,up,t_j) \wedge react(A,B) \wedge non\_active(A,B) \wedge react(B,C) \wedge active(B,C) \rightarrow con(B,down,t_{j+1})$

(C3) $con(A,down,t_k) \wedge react(A,B) \wedge active(A,B) \rightarrow con(B,down,t_{k+1})$

L'expression (C1) veut dire que si la concentration de *A* est augmentée au moment t_i, s'il y a une réaction entre *A* et *B* dans laquelle le métabolite *A* est consumée pour la production du métabolite *B*, et si pour toutes les réactions entre *B* et *C* le métabolite *B* n'est pas consommé pour la production du métabolite *C,* alors la concentration de *B* est augmentée au moment t_{i+1}.

L'expression (C2) veut dire que si la concentration de *A* est augmentée au moment t_j, s'il y a une réaction entre *A* et *B* dans laquelle le métabolite *A* n'est pas consommé pour la production du métabolite *B*, et si pour une réaction entre *B* et *C* le métabolite *B* est consommé pour la production du métabolite *C,* alors la concentration de *B* est diminuée au moment t_{j+1}.

L'expression (C3) veut dire que si la concentration de *A* est diminuée au moment t_k, s'il y a une réaction entre *A* et *B* dans laquelle le métabolite *A* est consommé pour la production du métabolite *B*, alors la concentration de *B* est diminuée au moment t_{k+1}.

Mais ces deux formulations posent problème dès qu'il y a un paradoxe. Si par exemple on a un ensemble *F* de sept formules *F = {con(A, up, t_i), react(A, B), react(B, C), active(A, B), non_active(B, C), non_active(A, B), active(B, C)},* alors on va dans les deux approches données ci-dessus inféré de *F*, *con(B, up, t_{i+1})* et *con(B, down, t_{i+1})* ce qui est paradoxe, parce que la concentration de *B* est augmentée et diminuée au même moment t_{i+1}. Pour résoudre ce conflit, on peut utiliser la logique des défauts.

Dans notre exemple, pour fournir des liens entre ces relations actives et non actives, l'idée intuitive est donc d'affaiblir la formulation de règles de causalité:

(1) Si $con(A,up,t_i)$, *react(A,B)*, *react(B,C), active(A,B)*, et *non_active(B,C)* sont vrais, et s'il est possible que $con(B,up,t_{i+1})$, alors $con(B,up,t_{i+1})$ est conclu
(2) Si $con(A,up,t_j)$, *react(A,B)*, *react(B,C), non_active(A,B)*, et *active(B,C)* sont vrais, et s'il est possible que $con(B,down,t_{j+1})$, alors $con(B,down,t_{j+1})$ est conclu

(3) Si *con(A,down,t_k)*, *react(A,B),* et *active(A,B)* sont vrais, et s'il est possible que *con(B,down,t_{k+1})*, alors *con(B,down,t_{k+1})* est conclu

La question est alors de formellement décrire le *possible*. Dans la logique des défauts, les règles (1), (2) et (3) seront exprimées de manière intuitive.

$$d1: \frac{con(A, up, t_i) \wedge react(A, B) \wedge react(B, C) \wedge active(A, B) \wedge non\_active(B, C): con(B, up, t_{i+1})}{con(B, up, t_{i+1})}$$

$$d2: \frac{con(A, up, t_j) \wedge react(A, B) \wedge react(B, C) \wedge non\_active(A, B) \wedge active(B, C): con(B, down, t_{j+1})}{con(B, down, t_{j+1})}$$

$$d3: \frac{con(A, down, t_k) \wedge react(A, B) \wedge active(A, B): con(B, down, t_{k+1})}{con(B, down, t_{k+1})}$$

Le paradoxe a été résolu.

Pour une théorie des défauts Δ = *(D, W),* où *D* l'ensemble des défauts et *W = {con(A, up, t_0)}*, en utilisant l'algorithme de calcul des extensions, on a 12 extensions ci-dessous [27]:

EXTENSION 1:

joint(con(a,up, t0), act(a,b), non_act(b,c)) → con(b,up, t1)

joint(con(b,up, t1), act(b,d), non_act(d,e)) → con(d,up, t2)

joint(con(d,up, t2), act(d,e), non_act(e,c)) → con(e,up, t3)

joint(con(e,up, t3), act(e,c), non_act(c,b)) → con(c,up, t4)

EXTENSION 2:

joint(con(a,up, t0), act(a,b), non_act(b,c)) → con(b,up, t1)

joint(con(b,up, t1), act(b,d), non_act(d,e)) → con(d,up, t2)

joint(con(d,up, t2), act(d,e), non_act(e,c)) → con(e,up, t3)

joint(con(e,up, t3), non_act(e,c), act(c,b)) → con(c,down, t4)

EXTENSION 3:

joint(con(a,up, t0), act(a,b), non_act(b,c)) → con(b,up, t1)

joint(con(b,up, t1), act(b,d), non_act(d,e)) → con(d,up, t2)

joint(con(d,up, t2), non_act(d,e), act(e,c)) → con(e,down, t3)

joint(con(e,down, t3), act(e,c)) → con(c,down, t4)

EXTENSION 4:

joint(con(a,up, t0), act(a,b), non_act(b,c)) → con(b,up, t1)

joint(con(b,up, t1), non_act(b,d), act(d,e)) → con(d,down, t2)

joint(con(d,down, t2), act(d,e)) → con(e,down, t3)

joint(con(e,down, t3), act(e,c)) → con(c,down, t4)

EXTENSION 5:

joint(con(a,up, t0), non_act(a,b), act(b,c)) → con(b,down, t1)

joint(con(b,down, t1), act(b,c)) → con(c,down, t2)

joint(con(b,down, t1), act(b,d)) → con(d,down, t2)

joint(con(d,down, t2), act(d,e)) → con(e,down, t3)

EXTENSION 6:

joint(con(a,up, t0), non_act(a,b), act(b,c)) → con(b,down, t1)

joint(con(b,down, t1), act(b,d)) → con(d,down, t2)

joint(con(d,down, t2), act(d,e)) → con(e,down, t3)

joint(con(e,down, t3), act(e,c)) → con(c,down, t4)

EXTENSION 7:

joint(con(a,up, t0), act(a,b), non_act(b,d)) → con(b,up, t1)

joint(con(b,up, t1), act(b,d), non_act(d,e)) → con(d,up, t2)

joint(con(d,up, t2), act(d,e), non_act(e,c)) → con(e,up, t3)

joint(con(e,up, t3), act(e,c), non_act(c,b)) → con(c,up, t4)

EXTENSION 8:

joint(con(a,up, t0), act(a,b), non_act(b,d)) → con(b,up, t1)

joint(con(b,up, t1), act(b,d), non_act(d,e)) → con(d,up, t2)

joint(con(d,up, t2), act(d,e), non_act(e,c)) → con(e,up, t3)

joint(con(e,up, t3), non_act(e,c), act(c,b)) → con(c,down, t4)

EXTENSION 9:

joint(con(a,up, t0), act(a,b), non_act(b,d)) → con(b,up, t1)

joint(con(b,up, t1), act(b,d), non_act(d,e)) → con(d,up, t2)

joint(con(d,up, t2), non_act(d,e), act(e,c)) → con(e,down, t3)

joint(con(e,down, t3), act(e,c)) → con(c,down, t4)

EXTENSION 10:

joint(con(a,up, t0), act(a,b), non_act(b,d)) → con(b,up, t1)

joint(con(b,up, t1), non_act(b,d), act(d,e)) → con(d,down, t2)

joint(con(d,down, t2), act(d,e)) → con(e,down, t3)

joint(con(e,down, t3), act(e,c)) → con(c,down, t4)

EXTENSION 11:

joint(con(a,up, t0), non_act(a,b), act(b,d)) → con(b,down, t1)

joint(con(b,down, t1), act(b,c)) → con(c,down, t2)

joint(con(b,down, t1), act(b,d)) → con(d,down, t2)

joint(con(d,down, t2), act(d,e)) → con(e,down, t3)

EXTENSION 12:

joint(con(a,up, t0), non_act(a,b), act(b,d)) → con(b,down, t1)

joint(con(b,down, t1), act(b,d)) → con(d,down, t2)

joint(con(d,down, t2), act(d,e)) → con(e,down, t3)

joint(con(e,down, t3), act(e,c)) → con(c,down, t4)

Cet algorithme a une complexité très grande. Il sera sans doute nécessaire de réduire les possibilités d'expression du langage pour arriver à des temps de calcul raisonnables. On retrouve ici la même problématique que pour la résolution pratique des problèmes NP-Complets. Pour la carte de Pommier, après une coupe de l'algorithme pour réduire les extensions équivalentes, on a la base de données et le résultat ci-dessous, où un fait *w* dans *W* est défini « *dur : --> w* », et un défaut *d = A:C/C* dans *D* est défini *« def : A --> C »*.

LA BASE DE DONNÉES:

1 dur : --> phosphorylation(atr,h2ax)
2 dur : --> stimulation(dsb,dna)
3 def : stimulation(dsb,dna) --> product(altered_dna)
4 def : product(altered_dna) --> binding(mre11,rad50)
5 def : binding(mre11,rad50) --> product(mre11_rad50)
6 def : product(mre11_rad50) --> binding(mre11_rad50,nbs1)
7 def : binding(mre11_rad50,nbs1) --> product(mrn)
8 def : product(mrn) --> binding(mrn,altered_dna)
9 def : binding(mrn,altered_dna) --> product(mrn_bound_to_dna)
10 def : product(mrn_bound_to_dna) --> binding(mrn_bound_to_dna,atm_atm)
11 def : binding(mrn_bound_to_dna,atm_atm) --> product(atm_bound_to_mrn)
12 def : product(atm_bound_to_mrn) --> bloq-product(atm_atm)
13 def : product(atm_bound_to_mrn) --> autophosphorylation(atm_bound_to_mrn)
14 def : autophosphorylation(atm_bound_to_mrn) --> product(p_atm_atm_bound)

15 def : product(p_atm_atm_bound) --> bloq-product(atm_atm)
16 def : product(p_atm_atm_bound) --> dissociation(p_atm_atm_bound)
17 def : dissociation(p_atm_atm_bound) --> product(p_atm_bound)
18 def : dissociation(p_atm_atm_bound) --> product(p_atm_free)
19 def : product(p_atm_free) --> phosphorylation(p_atm_free,chk1)
20 def : phosphorylation(p_atm_free,chk1) --> product(p_chk1)
21 dur : --> phosphorylation(atr,chk1)
22 def : phosphorylation(atr,chk1) --> product(p_chk1)
23 def : product(mrn_bound_to_dna) --> product(p_smc1)
24 def : product(mrn_bound_to_dna) --> product(p_mre11)
25 def : phosphorylation(atr,h2ax) --> product(gamma_h2ax)
26 def : phosphorylation(p_atm_bound,h2ax) --> product(gamma_h2ax)
27 def : product(gamma_h2ax) --> binding(gamma_h2ax,mdc1)
28 def : binding(gamma_h2ax,mdc1) --> product(h2ax_mdc1)
29 def : product(h2ax_mdc1) --> phosphorylation(p_atm_bound,h2ax_mdc1)
30 def : phosphorylation(p_atm_bound,h2ax_mdc1) --> product(p_mdc1)
31 def : product(p_mdc1) --> binding(p_mdc1,rnf8)
32 def : binding(p_mdc1,rnf8) --> product(rnf8_bound)
33 def : product(rnf8_bound) --> binding(rnf8_bound,ubc13)
34 def : binding(rnf8_bound,ubc13) --> product(rnf8_ubc13)
35 def : product(rnf8_ubc13) --> ubiquitination(rnf8_ubc13,h2a)
36 def : ubiquitination(rnf8_ubc13,h2a) --> product(ub_h2a)
37 def : product(ub_h2a) --> binding(ub_h2a,rnf168)
38 def : binding(ub_h2a,rnf168) --> product(rnf168_bound)
39 def : product(rnf168_bound) --> binding(rnf168_bound,ubc13)
40 def : binding(rnf168_bound,ubc13) --> product(rnf168_ubc13)
41 def : product(rnf168_ubc13) --> ubiquitination(rnf168_ubc13,h2a)
42 def : ubiquitination(rnf168_ubc13,h2a) --> product(poly_ub_h2a)
43 def : product(poly_ub_h2a) --> stimulation(poly_ub_h2a,dna)
44 def : stimulation(poly_ub_h2a,dna) --> product(changed_struct_dna)
45 def : product(poly_ub_h2a) --> binding(poly_ub_h2a,rap80)
46 def : binding(poly_ub_h2a,rap80) --> product(rap80_bound)
47 def : product(rap80_bound) --> binding(rap80_bound,abraxas)
48 def : binding(rap80_bound,abraxas) --> product(abraxas_bound)
49 def : product(abraxas_bound) --> binding(abraxas_bound,bre)
50 def : binding(abraxas_bound,bre) --> product(bre_bound)
51 def : product(abraxas_bound) --> binding(abraxas_bound,brcc36)
52 def : binding(abraxas_bound,brcc36) --> product(brcc36_bound)
53 def : product(brcc36_bound) --> binding(brcc36_bound,merit40)
54 def : binding(brcc36_bound,merit40) --> product(merit40_bound)
55 def : product(merit40_bound) --> binding(merit40_bound,brcc36_bound)
56 def : binding(merit40_bound,brcc36_bound) --> product(brcc36_merit40)
57 def : product(brcc36_merit40) --> binding(brcc36_merit40,brca1)
58 def : binding(brcc36_merit40,brca1) --> product(brca1_bound_to_rap80_complex)
59 def : product(changed_struct_dna) --> phosphorylation(p_atm_bound,mmset)
60 def : phosphorylation(p_atm_bound,mmset) --> product(p_mmset)
61 def : product(h2ax_mdc1) --> binding(h2ax_mdc1,mdc1)
62 def : binding(h2ax_mdc1,mdc1) --> product(mdc1_multi)

63 def : product(mdc1_multi), product(p_mmset) --> binding(mdc1_multi,p_mmset)
64 def : binding(mdc1_multi,p_mmset) --> product(mmset_mdc1)
65 def : product(mmset_mdc1) --> methylation(mmset_mdc1,h4)
66 def : methylation(mmset_mdc1,h4) --> product(h4k20me2)
67 def : product(h4k20me2), product(brca1_bound_to_rap80_complex) --> binding(h4k20me2,p53bp1)
68 def : binding(h4k20me2,p53bp1) --> product(p53bp1_bound)
69 def : product(p53bp1_bound) --> phosphorylation(p_atm_bound,p53bp1_bound)
70 def : phosphorylation(p_atm_bound,p53bp1_bound) --> product(p_53bp1)
71 def : product(p_atm_free) --> phosphorylation(p_atm_free,plk3)
72 def : phosphorylation(p_atm_free,plk3) --> product(p_plk3)
73 def : product(p_atm_free) --> phosphorylation(p_atm_free,chk2)
74 def : phosphorylation(p_atm_free,chk2) --> product(p_s33_35_chk2)
75 def : product(p_s33_35_chk2) --> phosphorylation(p_plk3,p_s33_35_chk2)
76 def : phosphorylation(p_plk3,p_s33_35_chk2) --> product(p_s33_35_s62_73_chk2)
77 def : product(p_s33_35_s62_73_chk2) --> phosphorylation(p_atm_free,p_s33_35_s62_73_chk2)
78 def : phosphorylation(p_atm_free,p_s33_35_s62_73_chk2) --> product(p_t68_chk2)
79 def : product(p_s33_35_s62_73_chk2) --> phosphorylation(atr,p_s33_35_s62_73_chk2)
80 def : phosphorylation(atr,p_s33_35_s62_73_chk2) --> product(p_t68_chk2)
81 def : product(p_t68_chk2) --> binding(p_t68_chk2,p_t68_chk2)
82 def : binding(p_t68_chk2,p_t68_chk2) --> product(chk2_chk2)
83 def : product(chk2_chk2) --> autophosphorylation(chk2_chk2)
84 def : autophosphorylation(chk2_chk2) --> product(p_active_chk2_chk2)
85 def : product(p_active_chk2_chk2), product(p_53bp1) --> binding(p_active_chk2_chk2,p_53bp1)
86 def : binding(p_active_chk2_chk2,p_53bp1) --> product(chk2_53bp1)
87 dur : --> binding(brca1,ctip)
88 def : binding(brca1,ctip) --> product(brca1_ctip)
89 def : product(brca1_ctip), product(chk2_53bp1) --> binding(brca1_ctip,chk2_53bp1)
90 def : binding(brca1_ctip,chk2_53bp1) --> product(chk2_53bp1_bound_to_brca1)
91 def : binding(brca1_ctip,chk2_53bp1) --> product(brca1_ctip_bound_to_chk2)
92 def : product(chk2_53bp1_bound_to_brca1), product(brca1_ctip_bound_to_chk2) --> phosphorylation(chk2_53bp1_bound_to_brca1,brca1_ctip_bound_to_chk2)
93 def : phosphorylation(chk2_53bp1_bound_to_brca1,brca1_ctip_bound_to_chk2) --> product(p_brca1_ctip_bound_to_chk2)
94 def : product(p_brca1_ctip_bound_to_chk2) --> product(p_s988_brca_ctip)
95 def : product(chk2_53bp1_bound_to_brca1) --> product(chk2_53bp1)
96 def : product(p_s988_brca_ctip) --> phosphorylation(p_atm_bound,p_s988_brca_ctip)
97 def : phosphorylation(p_atm_bound,p_s988_brca_ctip) --> product(brca_p_ctip)
98 def : product(brca_p_ctip) --> bloq-product(brca1_ctip)
99 def : product(brca_p_ctip) --> dissociation(brca_p_ctip)
100 def : dissociation(brca_p_ctip) --> product(brca1)
101 def : dissociation(brca_p_ctip) --> product(p_ctip)
102 def : product(brca1) --> phosphorylation(p_atm_bound,brca1)
103 def : phosphorylation(p_atm_bound,brca1) --> product(p_brca1)
104 def : product(brca1) --> phosphorylation(atr,brca1)
105 def : phosphorylation(atr,brca1) --> product(p_brca1)
106 def : product(p_brca1), product(p_53bp1) --> binding(mrn_bound_to_dna,p_brca1)

107 def : binding(mrn_bound_to_dna,p_brca1) --> product(brca1_bound_to_mrn)
108 def : product(chk2_53bp1) --> phosphorylation(chk2_53bp1,pml)
109 def : phosphorylation(chk2_53bp1,pml) --> product(p_pml)
110 def : product(p_pml) --> stimulation(p_pml,cell)
111 def : stimulation(p_pml,cell) --> product(apoptosis)
112 def : product(p_pml) --> binding(p_pml,mdm2)
113 def : binding(p_pml,mdm2) --> product(pml_mdm2)
114 def : product(pml_mdm2) --> bloq-binding(mdm2,p53)
115 dur : --> product(p53)
116 def : product(mdm2), product(p53) --> binding(mdm2,p53)
117 def : binding(mdm2,p53) --> product(mdm2_p53)
118 def : product(mdm2_p53) --> stimulation(p53_degradation_effectors,mdm2_p53)
119 def : stimulation(p53_degradation_effectors,mdm2_p53) --> product(p53_degradation)
120 def : product(p_atm_free), product(mdm2_p53) --> phosphorylation(p_atm_free,mdm2_p53)
121 def : phosphorylation(p_atm_free,mdm2_p53) --> product(p_s15_p53_mdm2)
122 def : product(mdm2_p53) --> phosphorylation(atr,mdm2_p53)
123 def : phosphorylation(atr,mdm2_p53) --> product(p_s15_p53_mdm2)
124 def : product(p_s15_p53_mdm2), product(chk2_53bp1) --> phosphorylation(chk2_53bp1,p_s15_p53_mdm2)
125 def : phosphorylation(chk2_53bp1,p_s15_p53_mdm2) --> product(p_p_p53_mdm2)
126 def : product(p_s15_p53_mdm2), product(p_chk1) --> phosphorylation(p_chk1,p_s15_p53_mdm2)
127 def : phosphorylation(p_chk1,p_s15_p53_mdm2) --> product(p_p_p53_mdm2)
128 def : product(p_s15_p53_mdm2) --> bloq-product(p53_degradation)
129 def : product(p_p_p53_mdm2) --> bloq-product(p53_degradation)
130 def : product(p_p_p53_mdm2) --> dissociation(p_p_p53_mdm2)
131 def : dissociation(p_p_p53_mdm2) --> product(p_p_p53)
132 def : dissociation(p_p_p53_mdm2) --> product(mdm2)
133 def : product(p_atm_free), product(mdm2) --> phosphorylation(p_atm_free,mdm2)
134 def : phosphorylation(p_atm_free,mdm2) --> product(p_mdm2)
135 def : product(p_p_p53) --> binding(p300,p_p_p53)
136 def : product(p_pml), binding(p300,p_p_p53) --> product(active_p53)
137 def : product(active_p53) --> transcription_activation(active_p53,prom_p21_gadd45)
138 def : transcription_activation(active_p53,prom_p21_gadd45) --> product(p21_and_gadd45)
139 def : product(prom_and_gadd45) --> stimulation(prom_and_gadd45,cell_cycle)
140 def : stimulation(p21_and_gadd45,cell_cycle) --> product(cell_cycle_arrest)
141 def : product(active_p53) --> transcription_activation(active_p53,prom_bnpf)
142 def : transcription_activation(active_p53,prom_bnpf) --> product(box_nas_puma_fas)
143 def : product(box_nas_puma_fas) --> stimulation(box_nas_puma_fas,cell)
144 def : stimulation(box_nas_puma_fas,cell) --> product(apoptosis)
145 def : product(active_p53) --> transcription_activation(active_p53,prom_mdm2)
146 def : transcription_activation(active_p53,prom_mdm2) --> product(mdm2)
147 dur : --> product(e2f1)
148 def : product(e2f1), product(chk2_53bp1) --> phosphorylation(chk2_53bp1,e2f1)
149 def : phosphorylation(chk2_53bp1,e2f1) --> product(p_e2f1)
150 def : product(e2f1) --> stimulation(e2f1_degradation_effectors,e2f1)
151 def : stimulation(e2f1_degradation_effectors,e2f1) --> product(e2f1_degradation)
152 def : product(p_e2f1) --> phosphorylation(p_atm_free,p_e2f1)

153 def : phosphorylation(p_atm_free,p_e2f1) --> product(p_p_e2f1)
154 def : product(p_e2f1) --> phosphorylation(atr,p_e2f1)
155 def : phosphorylation(atr,p_e2f1) --> product(p_p_e2f1)
156 def : product(p_p_e2f1) --> transcription_activation(p_p_e2f1,prom_chk2)
157 def : transcription_activation(p_p_e2f1,prom_chk2) --> product(chk2)
158 def : product(p_p_e2f1) --> transcription_activation(p_p_e2f1,prom_arf)
159 def : transcription_activation(p_p_e2f1,prom_arf) --> product(arf)
160 def : product(p_p_e2f1) --> transcription_activation(p_p_e2f1,prom_p73_apaf1)
161 def : transcription_activation(p_p_e2f1,prom_p73_apaf1) --> product(p73_apaf1)
162 def : product(p73_apaf1) --> stimulation(p73_apaf1,cell)
163 def : stimulation(p73_apaf1,cell) --> product(apoptosis)
164 def : product(arf) --> binding(arf,mdm2)
165 def : binding(arf,mdm2) --> product(arf_mdm2)
166 def : product(arf_mdm2) --> bloq-product(mdm2_p53)
167 def : product(p_p_e2f1) --> stimulation(p_p_e2f1,unknown_atm_way)
168 def : stimulation(p_p_e2f1,unknown_atm_way) --> product(p_atm_free)
169 def : product(p_atm_free) --> phosphorylation(p_atm_free,p38)
170 def : phosphorylation(p_atm_free,p38) --> product(p_p38)
171 def : product(chk2_53bp1) --> phosphorylation(chk2_53bp1,cdc25a)
172 def : phosphorylation(chk2_53bp1,cdc25a) --> product(p_cdc25a)
173 def : product(p_chk1) --> phosphorylation(p_chk1,cdc25a)
174 def : phosphorylation(p_chk1,cdc25a) --> product(p_cdc25a)
175 def : product(p_cdc25a) --> stimulation(cdc25a,cell)
176 def : stimulation(cdc25a,cell) --> bloq-product(cell_cycle_arrest)
177 def : product(p_cdc25a) --> stimulation(p_cdc25a,cdc25a_degradation_effectors)
178 def : stimulation(p_cdc25a,cdc25_degradation_effectors) --> product(cdc25a_degradation)
179 def : product(cdc25a_degradation) --> bloq-stimulation(cdc25a,cell)
180 def : product(chk2_53bp1) --> phosphorylation(chk2_53bp1,cdc25c)
181 def : phosphorylation(chk2_53bp1,cdc25c) --> product(p_cdc25c)
182 def : product(p_plk3) --> phosphorylation(p_plk3,cdc25c)
183 def : phosphorylation(p_plk3,cdc25c) --> product(p_cdc25c)
184 def : product(p_chk1) --> phosphorylation(p_chk1,cdc25c)
185 def : phosphorylation(p_chk1,cdc25c) --> product(p_cdc25c)
186 def : product(p_p38) --> phosphorylation(p_p38,cdc25c)
187 def : phosphorylation(p_p38,cdc25c) --> product(p_cdc25c)
188 def : product(p_cdc25c) --> stimulation(cdc25c,cell)
189 def : stimulation(cdc25c,cell) --> bloq-product(cell_cycle_arrest)
190 def : product(p_cdc25c) --> stimulation(p_cdc25c,cdc25c_degradation_effectors)
191 def : stimulation(cdc25b,cell) --> bloq-product(cell_cycle_arrest)
192 def : product(p_p38) --> phosphorylation(p_p38,cdc25b)
193 def : phosphorylation(p_p38,cdc25b) --> product(p_cdc25b)
194 def : product(p_cdc25b) --> bloq-stimulation(cdc25b,cell)
195 def : stimulation(jnk,cdc25c) --> product(p_cdc25c)
196 def : phosphorylation(jnk,p_s15_p53_mdm2) --> product(p_p_p53_mdm2)
197 def : product(p_atm_bound) --> phosphorylation(p_atm_bound,nbs1)
198 def : phosphorylation(p_atm_bound,nbs1) --> product(p_nbs1)
199 def : product(p_nbs1) --> phosphorylation(p_atm_bound,fancd2)
200 def : phosphorylation(p_atm_bound,fancd2) --> product(p_fancd2)

201 def : product(p_fancd2) --> stimulation(p_fancd2,cell)
202 def : stimulation(p_fancd2,cell) --> bloq-product(cell_cycle_arrest)
203 def : product(chk2_53bp1) --> binding(chk2_53bp1,mus81)
204 def : binding(chk2_53bp1,mus81) --> product(mus81_chk2)
205 def : product(mus81_chk2) --> binding(mus81_chk2,altered_dna)
206 def : binding(mus81_chk2,altered_dna) --> product(mus81_bound_to_dna)

LE RÉSULTAT

def 3 : (stimulation(dsb,dna), up, t0) ==> (product(altered_dna), up, t1)
def 4 : (product(altered_dna), up, t1) ==> (binding(mre11,rad50), up, t2)
def 5 : (binding(mre11,rad50), up, t2) ==> (product(mre11_rad50), up, t3)
def 6 : (product(mre11_rad50), up, t3) ==> (binding(mre11_rad50,nbs1), up, t4)
def 7 : (binding(mre11_rad50,nbs1), up, t4) ==> (product(mrn), up, t5)
def 8 : (product(mrn), up, t5) ==> (binding(mrn,altered_dna), up, t6)
def 9 : (binding(mrn,altered_dna), up, t6) ==> (product(mrn_bound_to_dna), up, t7)
def 10 : (product(mrn_bound_to_dna), up, t7) ==> (binding(mrn_bound_to_dna,atm_atm), up, t8)
def 11 : (binding(mrn_bound_to_dna,atm_atm), up, t8) ==> (product(atm_bound_to_mrn), up, t9)
def 12 : (product(atm_bound_to_mrn), up, t9) ==> (product(atm_atm), down, t10)
def 13 : (product(atm_bound_to_mrn), up, t9) ==> (autophosphorylation(atm_bound_to_mrn), up, t10)
def 14 : (autophosphorylation(atm_bound_to_mrn), up, t10) ==> (product(p_atm_atm_bound), up, t11)
def 16 : (product(p_atm_atm_bound), up, t11) ==> (dissociation(p_atm_atm_bound), up, t12)
def 17 : (dissociation(p_atm_atm_bound), up, t12) ==> (product(p_atm_bound), up, t13)
def 18 : (dissociation(p_atm_atm_bound), up, t12) ==> (product(p_atm_free), up, t13)
def 19 : (product(p_atm_free), up, t13) ==> (phosphorylation(p_atm_free,chk1), up, t14)
def 20 : (phosphorylation(p_atm_free,chk1), up, t14) ==> (product(p_chk1), up, t15)
def 23 : (product(mrn_bound_to_dna), up, t7) ==> (product(p_smc1), up, t8)
def 24 : (product(mrn_bound_to_dna), up, t7) ==> (product(p_mre11), up, t8)
def 25 : (phosphorylation(atr,h2ax), up, t0) ==> (product(gamma_h2ax), up, t1)
def 27 : (product(gamma_h2ax), up, t1) ==> (binding(gamma_h2ax,mdc1), up, t2)
def 28 : (binding(gamma_h2ax,mdc1), up, t2) ==> (product(h2ax_mdc1), up, t3)
def 29 : (product(h2ax_mdc1), up, t3) ==> (phosphorylation(p_atm_bound,h2ax_mdc1), up, t4)
def 30 : (phosphorylation(p_atm_bound,h2ax_mdc1), up, t4) ==> (product(p_mdc1), up, t5)
def 31 : (product(p_mdc1), up, t5) ==> (binding(p_mdc1,rnf8), up, t6)
def 32 : (binding(p_mdc1,rnf8), up, t6) ==> (product(rnf8_bound), up, t7)
def 33 : (product(rnf8_bound), up, t7) ==> (binding(rnf8_bound,ubc13), up, t8)
def 34 : (binding(rnf8_bound,ubc13), up, t8) ==> (product(rnf8_ubc13), up, t9)
def 35 : (product(rnf8_ubc13), up, t9) ==> (ubiquitination(rnf8_ubc13,h2a), up, t10)
def 36 : (ubiquitination(rnf8_ubc13,h2a), up, t10) ==> (product(ub_h2a), up, t11)
def 37 : (product(ub_h2a), up, t11) ==> (binding(ub_h2a,rnf168), up, t12)
def 38 : (binding(ub_h2a,rnf168), up, t12) ==> (product(rnf168_bound), up, t13)
def 39 : (product(rnf168_bound), up, t13) ==> (binding(rnf168_bound,ubc13), up, t14)
def 40 : (binding(rnf168_bound,ubc13), up, t14) ==> (product(rnf168_ubc13), up, t15)
def 41 : (product(rnf168_ubc13), up, t15) ==> (ubiquitination(rnf168_ubc13,h2a), up, t16)
def 42 : (ubiquitination(rnf168_ubc13,h2a), up, t16) ==> (product(poly_ub_h2a), up, t17)
def 43 : (product(poly_ub_h2a), up, t17) ==> (stimulation(poly_ub_h2a,dna), up, t18)
def 44 : (stimulation(poly_ub_h2a,dna), up, t18) ==> (product(changed_struct_dna), up, t19)
def 45 : (product(poly_ub_h2a), up, t17) ==> (binding(poly_ub_h2a,rap80), up, t18)

def 46 : (binding(poly_ub_h2a,rap80), up, t18) ==> (product(rap80_bound), up, t19)
def 47 : (product(rap80_bound), up, t19) ==> (binding(rap80_bound,abraxas), up, t20)
def 48 : (binding(rap80_bound,abraxas), up, t20) ==> (product(abraxas_bound), up, t21)
def 49 : (product(abraxas_bound), up, t21) ==> (binding(abraxas_bound,bre), up, t22)
def 50 : (binding(abraxas_bound,bre), up, t22) ==> (product(bre_bound), up, t23)
def 51 : (product(abraxas_bound), up, t21) ==> (binding(abraxas_bound,brcc36), up, t22)
def 52 : (binding(abraxas_bound,brcc36), up, t22) ==> (product(brcc36_bound), up, t23)
def 53 : (product(brcc36_bound), up, t23) ==> (binding(brcc36_bound,merit40), up, t24)
def 54 : (binding(brcc36_bound,merit40), up, t24) ==> (product(merit40_bound), up, t25)
def 55 : (product(merit40_bound), up, t25) ==> (binding(merit40_bound,brcc36_bound), up, t26)
def 56 : (binding(merit40_bound,brcc36_bound), up, t26) ==> (product(brcc36_merit40), up, t27)
def 57 : (product(brcc36_merit40), up, t27) ==> (binding(brcc36_merit40,brca1), up, t28)
def 58 : (binding(brcc36_merit40,brca1), up, t28) ==> (product(brca1_bound_to_rap80_complex), up, t29)
def 59 : (product(changed_struct_dna), up, t19) ==> (phosphorylation(p_atm_bound,mmset), up, t20)
def 60 : (phosphorylation(p_atm_bound,mmset), up, t20) ==> (product(p_mmset), up, t21)
def 61 : (product(h2ax_mdc1), up, t3) ==> (binding(h2ax_mdc1,mdc1), up, t4)
def 62 : (binding(h2ax_mdc1,mdc1), up, t4) ==> (product(mdc1_multi), up, t5)
def 63 : (product(mdc1_multi), up, t5), (product(p_mmset), up, t21) ==> (binding(mdc1_multi,p_mmset), up, t22)
def 64 : (binding(mdc1_multi,p_mmset), up, t22) ==> (product(mmset_mdc1), up, t23)
def 65 : (product(mmset_mdc1), up, t23) ==> (methylation(mmset_mdc1,h4), up, t24)
def 66 : (methylation(mmset_mdc1,h4), up, t24) ==> (product(h4k20me2), up, t25)
def 67 : (product(h4k20me2), up, t25), (product(brca1_bound_to_rap80_complex), up, t29) ==> (binding(h4k20me2,p53bp1), up, t30)
def 68 : (binding(h4k20me2,p53bp1), up, t30) ==> (product(p53bp1_bound), up, t31)
def 69 : (product(p53bp1_bound), up, t31) ==> (phosphorylation(p_atm_bound,p53bp1_bound), up, t32)
def 70 : (phosphorylation(p_atm_bound,p53bp1_bound), up, t32) ==> (product(p_53bp1), up, t33)
def 71 : (product(p_atm_free), up, t13) ==> (phosphorylation(p_atm_free,plk3), up, t14)
def 72 : (phosphorylation(p_atm_free,plk3), up, t14) ==> (product(p_plk3), up, t15)
def 73 : (product(p_atm_free), up, t13) ==> (phosphorylation(p_atm_free,chk2), up, t14)
def 74 : (phosphorylation(p_atm_free,chk2), up, t14) ==> (product(p_s33_35_chk2), up, t15)
def 75 : (product(p_s33_35_chk2), up, t15) ==> (phosphorylation(p_plk3,p_s33_35_chk2), up, t16)
def 76 : (phosphorylation(p_plk3,p_s33_35_chk2), up, t16) ==> (product(p_s33_35_s62_73_chk2), up, t17)
def 77 : (product(p_s33_35_s62_73_chk2), up, t17) ==> (phosphorylation(p_atm_free,p_s33_35_s62_73_chk2), up, t18)
def 78 : (phosphorylation(p_atm_free,p_s33_35_s62_73_chk2), up, t18) ==> (product(p_t68_chk2), up, t19)
def 79 : (product(p_s33_35_s62_73_chk2), up, t17) ==> (phosphorylation(atr,p_s33_35_s62_73_chk2), up, t18)
def 81 : (product(p_t68_chk2), up, t19) ==> (binding(p_t68_chk2,p_t68_chk2), up, t20)
def 82 : (binding(p_t68_chk2,p_t68_chk2), up, t20) ==> (product(chk2_chk2), up, t21)
def 83 : (product(chk2_chk2), up, t21) ==> (autophosphorylation(chk2_chk2), up, t22)
def 84 : (autophosphorylation(chk2_chk2), up, t22) ==> (product(p_active_chk2_chk2), up, t23)
def 85 : (product(p_active_chk2_chk2), up, t23), (product(p_53bp1), up, t33) ==>

(binding(p_active_chk2_chk2,p_53bp1), up, t34)
def 86 : (binding(p_active_chk2_chk2,p_53bp1), up, t34) ==> (product(chk2_53bp1), up, t35)
def 88 : (binding(brca1,ctip), up, t0) ==> (product(brca1_ctip), up, t1)
def 89 : (product(brca1_ctip), up, t1), (product(chk2_53bp1), up, t35) ==>
(binding(brca1_ctip,chk2_53bp1), up, t36)
def 90 : (binding(brca1_ctip,chk2_53bp1), up, t36) ==> (product(chk2_53bp1_bound_to_brca1), up, t37)
def 91 : (binding(brca1_ctip,chk2_53bp1), up, t36) ==> (product(brca1_ctip_bound_to_chk2), up, t37)
def 92 : (product(chk2_53bp1_bound_to_brca1), up, t37), (product(brca1_ctip_bound_to_chk2), up, t37) ==>(phosphorylation(chk2_53bp1_bound_to_brca1,brca1_ctip_bound_to_chk2), up, t38)
def 93 : (phosphorylation(chk2_53bp1_bound_to_brca1,brca1_ctip_bound_to_chk2), up, t38) ==>
(product(p_brca1_ctip_bound_to_chk2), up, t39)
def 94 : (product(p_brca1_ctip_bound_to_chk2), up, t39) ==> (product(p_s988_brca_ctip), up, t40)
def 96 : (product(p_s988_brca_ctip), up, t40) ==>
(phosphorylation(p_atm_bound,p_s988_brca_ctip), up, t41)
def 97 : (phosphorylation(p_atm_bound,p_s988_brca_ctip), up, t41) ==> (product(brca_p_ctip), up, t42)
def 99 : (product(brca_p_ctip), up, t42) ==> (dissociation(brca_p_ctip), up, t43)
def 100 : (dissociation(brca_p_ctip), up, t43) ==> (product(brca1), up, t44)
def 101 : (dissociation(brca_p_ctip), up, t43) ==> (product(p_ctip), up, t44)
def 102 : (product(brca1), up, t44) ==> (phosphorylation(p_atm_bound,brca1), up, t45)
def 103 : (phosphorylation(p_atm_bound,brca1), up, t45) ==> (product(p_brca1), up, t46)
def 104 : (product(brca1), up, t44) ==> (phosphorylation(atr,brca1), up, t45)
def 106 : (product(p_brca1), up, t46), (product(p_53bp1), up, t33) ==>
(binding(mrn_bound_to_dna,p_brca1), up, t47)
def 107 : (binding(mrn_bound_to_dna,p_brca1), up, t47) ==> (product(brca1_bound_to_mrn), up, t48)
def 108 : (product(chk2_53bp1), up, t35) ==> (phosphorylation(chk2_53bp1,pml), up, t36)
def 109 : (phosphorylation(chk2_53bp1,pml), up, t36) ==> (product(p_pml), up, t37)
def 110 : (product(p_pml), up, t37) ==> (stimulation(p_pml,cell), up, t38)
def 111 : (stimulation(p_pml,cell), up, t38) ==> (product(apoptosis), up, t39)
def 112 : (product(p_pml), up, t37) ==> (binding(p_pml,mdm2), up, t38)
def 113 : (binding(p_pml,mdm2), up, t38) ==> (product(pml_mdm2), up, t39)
def 114 : (product(pml_mdm2), up, t39) ==> (binding(mdm2,p53), down, t40)
def 148 : (product(e2f1), up, t0), (product(chk2_53bp1), up, t35) ==>
(phosphorylation(chk2_53bp1,e2f1), up, t36)
def 149 : (phosphorylation(chk2_53bp1,e2f1), up, t36) ==> (product(p_e2f1), up, t37)
def 150 : (product(e2f1), up, t0) ==> (stimulation(e2f1_degradation_effectors,e2f1), up, t1)
def 151 : (stimulation(e2f1_degradation_effectors,e2f1), up, t1) ==> (product(e2f1_degradation), up, t2)
def 152 : (product(p_e2f1), up, t37) ==> (phosphorylation(p_atm_free,p_e2f1), up, t38)
def 153 : (phosphorylation(p_atm_free,p_e2f1), up, t38) ==> (product(p_p_e2f1), up, t39)
def 154 : (product(p_e2f1), up, t37) ==> (phosphorylation(atr,p_e2f1), up, t38)
def 156 : (product(p_p_e2f1), up, t39) ==> (transcription_activation(p_p_e2f1,prom_chk2), up, t40)
def 157 : (transcription_activation(p_p_e2f1,prom_chk2), up, t40) ==> (product(chk2), up, t41)
def 158 : (product(p_p_e2f1), up, t39) ==> (transcription_activation(p_p_e2f1,prom_arf), up, t40)

def 159 : (transcription_activation(p_p_e2f1,prom_arf), up, t40) ==> (product(arf), up, t41)
def 160 : (product(p_p_e2f1), up, t39) ==> (transcription_activation(p_p_e2f1,prom_p73_apaf1), up, t40)
def 161 : (transcription_activation(p_p_e2f1,prom_p73_apaf1), up, t40) ==> (product(p73_apaf1), up, t41)
def 162 : (product(p73_apaf1), up, t41) ==> (stimulation(p73_apaf1,cell), up, t42)
def 164 : (product(arf), up, t41) ==> (binding(arf,mdm2), up, t42)
def 165 : (binding(arf,mdm2), up, t42) ==> (product(arf_mdm2), up, t43)
def 166 : (product(arf_mdm2), up, t43) ==> (product(mdm2_p53), down, t44)
def 167 : (product(p_p_e2f1), up, t39) ==> (stimulation(p_p_e2f1,unknown_atm_way), up, t40)
def 169 : (product(p_atm_free), up, t13) ==> (phosphorylation(p_atm_free,p38), up, t14)
def 170 : (phosphorylation(p_atm_free,p38), up, t14) ==> (product(p_p38), up, t15)
def 171 : (product(chk2_53bp1), up, t35) ==> (phosphorylation(chk2_53bp1,cdc25a), up, t36)
def 172 : (phosphorylation(chk2_53bp1,cdc25a), up, t36) ==> (product(p_cdc25a), up, t37)
def 173 : (product(p_chk1), up, t15) ==> (phosphorylation(p_chk1,cdc25a), up, t16)
def 175 : (product(p_cdc25a), up, t37) ==> (stimulation(cdc25a,cell), up, t38)
def 176 : (stimulation(cdc25a,cell), up, t38) ==> (product(cell_cycle_arrest), down, t39)
def 177 : (product(p_cdc25a), up, t37) ==> (stimulation(p_cdc25a,cdc25a_degradation_effectors), up, t38)
def 180 : (product(chk2_53bp1), up, t35) ==> (phosphorylation(chk2_53bp1,cdc25c), up, t36)
def 181 : (phosphorylation(chk2_53bp1,cdc25c), up, t36) ==> (product(p_cdc25c), up, t37)
def 182 : (product(p_plk3), up, t15) ==> (phosphorylation(p_plk3,cdc25c), up, t16)
def 184 : (product(p_chk1), up, t15) ==> (phosphorylation(p_chk1,cdc25c), up, t16)
def 186 : (product(p_p38), up, t15) ==> (phosphorylation(p_p38,cdc25c), up, t16)
def 188 : (product(p_cdc25c), up, t37) ==> (stimulation(cdc25c,cell), up, t38)
def 190 : (product(p_cdc25c), up, t37) ==> (stimulation(p_cdc25c,cdc25c_degradation_effectors), up, t38)
def 192 : (product(p_p38), up, t15) ==> (phosphorylation(p_p38,cdc25b), up, t16)
def 193 : (phosphorylation(p_p38,cdc25b), up, t16) ==> (product(p_cdc25b), up, t17)
def 194 : (product(p_cdc25b), up, t17) ==> (stimulation(cdc25b,cell), down, t18)
def 197 : (product(p_atm_bound), up, t13) ==> (phosphorylation(p_atm_bound,nbs1), up, t14)
def 198 : (phosphorylation(p_atm_bound,nbs1), up, t14) ==> (product(p_nbs1), up, t15)
def 199 : (product(p_nbs1), up, t15) ==> (phosphorylation(p_atm_bound,fancd2), up, t16)
def 200 : (phosphorylation(p_atm_bound,fancd2), up, t16) ==> (product(p_fancd2), up, t17)
def 201 : (product(p_fancd2), up, t17) ==> (stimulation(p_fancd2,cell), up, t18)
def 203 : (product(chk2_53bp1), up, t35) ==> (binding(chk2_53bp1,mus81), up, t36)
def 204 : (binding(chk2_53bp1,mus81), up, t36) ==> (product(mus81_chk2), up, t37)
def 205 : (product(mus81_chk2), up, t37) ==> (binding(mus81_chk2,altered_dna), up, t38)
def 206 : (binding(mus81_chk2,altered_dna), up, t38) ==> (product(mus81_bound_to_dna), up, t39)

Ce résultat est représenté dans la **Figure 4.7**. Les chiffres rouges sont les défauts utilisés dans le résultat.

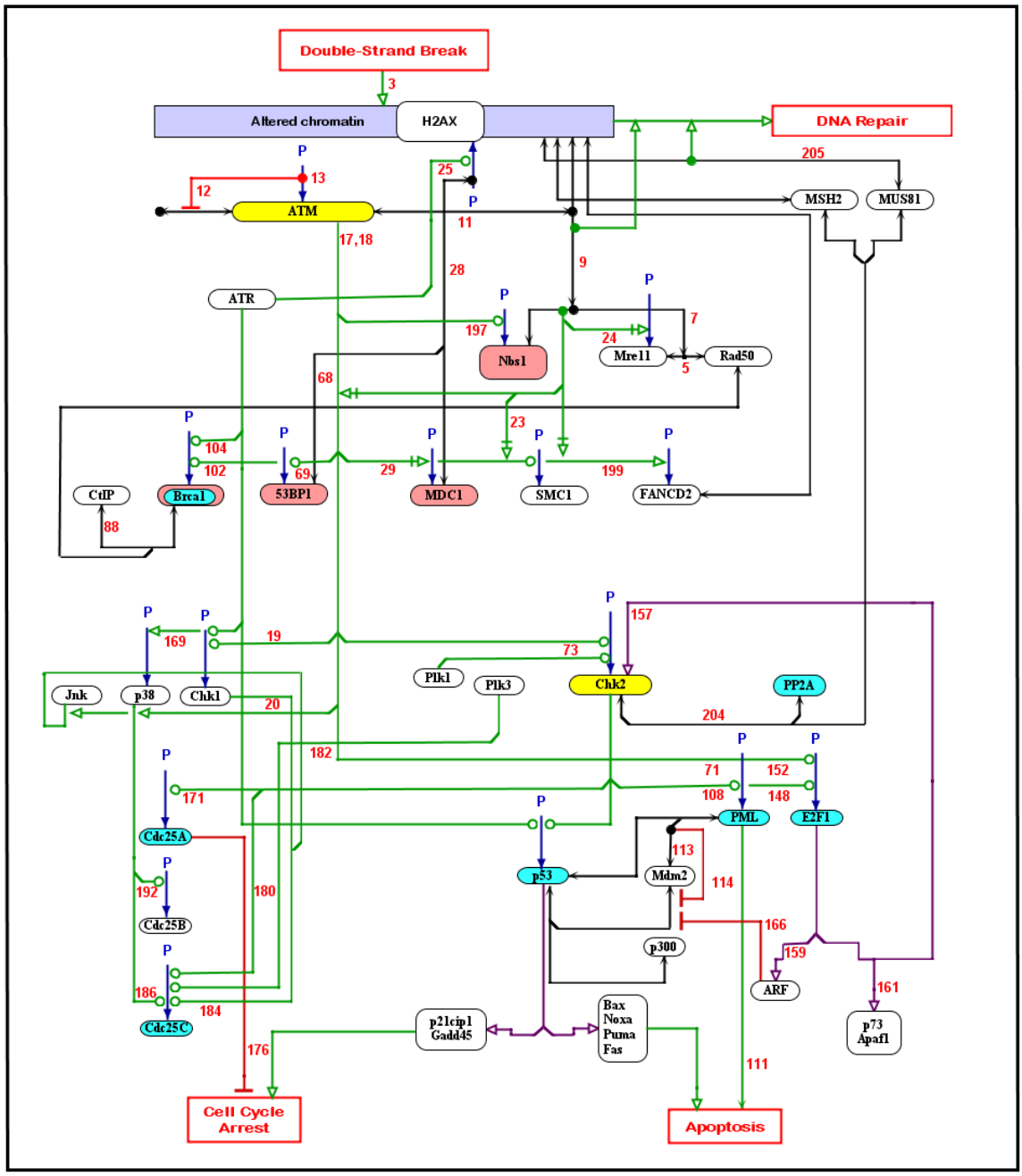

Figure 4.7. Une extension de la carte de Pommier

Conclusion

Les voies de signalisation sont très importantes pour la compréhension des mécanismes des fonctions biologiques.

Cette thèse étudie la synthèse automatique des voies de signalisation à partir d'informations trouvées dans des bases de connaissances. Une composante essentielle de cette approche est l'utilisation du raisonnement logique pour représenter la connaissance biologique de la communication intracellulaire. En choisissant la représentation adéquate des connaissances biologiques la partie « raisonnement » est capable d'assigner un ordre dans l'acquisition des faits et d'extraire les interactions nécessaires à la synthèse des voies métaboliques ou de signalisation.

La première partie de la thèse est consacrée à la synthèse des voies pharmaco-cinétiques par des algorithmes de production. Cet algorithme est décidable, cohérent et complet. La deuxième partie utilise la logique des défauts pour représenter les interactions dans les réseaux de gènes.

Le problème de l'implémentation et de la complexité algorithmique n'a pas été abordé. Il sera sans doute nécessaire de réduire les possibilités d'expression du langage pour arriver à des temps de calcul raisonnables. On retrouve ici la même problématique que pour la résolution pratique des problèmes NP-Complets. Bien entendu le problème est ici plus compliqué. La complexité pratique peut alors rester accessible en contrôlant l'écriture des règles. Par exemple dans un premier temps on peut essayer de se restreindre à quelque chose qui ressemble à des clauses de Horn (clauses qui contiennent au plus un littéral positif [2]).

Bibliographie

[1] Demongeot J, "Multi-stationarity and cell differentiation", J. Biol. Systems., 6, 1-2 (1998).

[2] Doncescu A. , Inoue K. and Yamamoto, "Knowledge-based discovery in systems biology using CF-induction". New Trends in Applied Artificial Intelligence: Proceedings of the 20th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA / AIE 2007), Lecture Notes in Artificial Intelligence, volume 4570, pages 395-404, Springer-Verlag.

[3] Doncescu A, and Siegel P, "Utilisation de la logique des hypothèses pour la modélisation des voies de signalisation dans la cellule", JIAF 11, Lyon 8-10, June 2011.

[4] Doncescu A., Waissman J.,Richard G.,Roux G. "Characterization of bio-chemical signals by inductive logic programming", Knowledge-Based Systems 15 (1), 129-137, 2002.

[5] Christophe Chassagnole, Juan Carlos, A Rodriguez, Andrei Doncescu, Laurence T Yang "Differential evolutionary algorithms for in vivo dynamic analysis of glycolysis and pentose phosphate pathway in Escherichia Coli", Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies, 59-78, John Wiley & Sons, Inc.,2006.

[6] Montseny E., Doncescu A., "Operatorial Parametrizing of Controlled Dynamic Systems-Application to the Fed-Batch Bioreactor Control Problem", 17th World Congress The International Federation of Automatic Control. Seoul, Korea, June 2008.

[7] Forget L, Rish V. , P Siegel. "Preferential Logics are X-logics" Journal of Computational Logic, 10, 2000, pp. 1-13.

[8] Ginsberg, ML, Smith, DE (July 1988). "Reasoning about action II: the qualification problem". Artificial Intelligence Vol. 35 No. 3 pp.311-342.

[9] Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., & Turner, H. (March 2004). "Nonmonotonic causal theories". Artificial Intelligence No. 1-2 vol.153 pp.49-104.

[10] Inoue K, "Induction as Consequence Finding". Machine Learning, 55 (2) :109-135, 2004.

[11] Inoue K, Saito H. "Circumscripta Policies for Induction" Proceedings of 14th Conf. on Inductive Logic Programming, LNAI 3194, pp.164-179, Springer, September 2004.

[12] K. Inoue, T. Sato, M. Ishihata, Y. Kameya and H. Nabeshima, "Evaluating abductive hypotheses using an em algorithm on bdds", in: Proceedings of IJCAI-09, Pasadena, CA, 2009, pp. 810–815.

[13] Kayser D., Levy F. "Modeling symbolic causal reasoning", Intellecta 2004 / 1, 38, pp 291-232

[14] Nabeshima H. , Iwanuma K., Inoue K. Ray O. "SOLAR: An automated deduction system for Finding consequence". AI Commun. 23 (2-3): 183-203 (2010)

[15] Roux-Rouquié M., L. Hood, Imbeaud S., Auffray C. "Issues in Computational Methods for Functional Genomics and Systems Biology". CMSB 2003 : 182-186

[16] Schwind P. , Siegel P: "Modal Logic for Hypothesis Theory", Fundamentae Informaticae, cal 21, No. 1-2 89-101.

[17] Synnaeve G, Inoue K, Doncescu A, Nabeshima N, Kameya Y, Ishihata M., Sato T, "Kinetic models and qualitative abstraction for relational learning in systems biology", BIOSTEC Bioinformatics 2011

[18] Siegel P. : "A modal language for Nonmonotonic Reasonning", Proc. Workshop DRUMS / EEC Marseille 24-27 February 90.

[19] P. Siegel , C. Schwind, "Modal logic based theory for nonmonotonic reasoning". Journal of Applied Non-classical Logic, Volume 3 - No. 1 / 1993, P 73-92.

[20] Synnaeve G., Doncescu A., Inoue K., "Kinetic models for logic-based hypothesis finding in metabolic pathways", Int'l Conf. on Inductive Logic Programming (ILP-09), 2009.

[21] Tran N. , C. Baral (2007) "Hypothesizing and reasoning about signaling networks". Journal of Applied Logic 7 (2009) 253-274

[22] Barthélémy DWORKIN “Utilisation de SOLAR dans la voie de signalisation ATM-dépendante de la cassure double-brin de l'ADN”. Mémoire de master 2 / Université Paul Sabatier, Toulouse 2011.

[23] Toulgoat I. “Modélisation du comportement humain dans les simulations de combat naval”. Thèse doctorat / Université du Sud, Toulon-Var [s.n] 2011.

[24] Pommier Y et al. “Targeting Chk2 kinase: Molecular interaction maps and therapeutic rationale”. 2005

[25] Munna L. Agarwal et al. “The p53 Network”, Journal of Biological Chemistry, Vol. 273, No 1, Issue of January 2, pp. 1-4, 1998

[26] P. Siegel, “Représentation et utilization de la connaissance en calcul propositionnel”, Thèse d’État, Université d’Aix-Marseille II, Luminy, France, 1987.

[27] Tan Le, A. Doncescu, P. Siegel, “Default Logic for Diagnostic of Discrete Time System”, 8th international conference BWCCA, October 28-30 2013, Compiegne, France, pp. 488-493.

[28] Tan Le, A. Doncescu, P. Siegel, “Utilization of Default Logic for Analyzing a Metabolic System in Discrete Time”, 13th international conference ICCSA, June 24-27 2013, Ho Chi Minh City, Vietnam, pp. 130-136.

[29] G. Bossu, P. Siegel, “Nonmonotonic Reasoning and databases”, Workshop Logic and Data Bases. CERT - Toulouse - Décembre 1982

[30] G. Bossu, P. Siegel, “Saturation, Nonmonotonic Reasoning and the Closed World Assumption”, Artificial Intelligence 25, Janvier 1985, p. 13-63

[31] J.M. Boi, E. Innocenti, A. Rauzy, P. Siegel, “Production Fields : a New approach to Deduction Problems and two Algorithms for propositional Calculus”, Revue d'Intelligence Artificielle, vol 6 - n° 3/1992 , p 235, 255.