Le système de maintenance de la vérité à propagation de contextes

Jérôme Euzenat

Le système de maintenance de la vérité à propagation de contextes

Jérôme EUZENAT

RR 779-I- Mai 1989

Le système de maintenance de la vérité à propagation de contextes

Jérôme Euzenat

Résumé

Les systèmes de maintenance de la vérité ont été conçus pour raisonner à l'aide de connaissance incomplète. Le CP-TMS est un système de maintenance de la vérité tentant de combiner les avantages des systèmes à propagation (TMS) — autorisant l'utilisation d'inférences non monotones — et des systèmes à contextes (ATMS) — considérant le raisonnement sous plusieurs contextes simultanément. Il maintient un graphe de dépendances entre les objets manipulés par un système de raisonnement et propage à travers ce graphe les contextes dans lesquels les nœuds sont valides. Ces contextes prennent en compte l'incomplétude des bases de connaissance et permettent d'exprimer des inférences non monotones. Une théorie de l'interprétation des contextes est présentée. Elle garantit certaines bonnes propriétés aux contextes manipulés par l'implémentation. Le système garantit la consistance des contextes manipulés et permet de répondre à des requêtes concernant différents contextes simultanément au regard de la base théorique ainsi posée.

Mots clés

Systèmes de maintenance de la vérité — Raisonnement non monotone — Raisonnement multimonde — Raisonnement hypothétique.

Context propagation truth maintenance system

Abstract

The CP-TMS is a truth maintenance system designed to merge advantages of both propagation oriented (TMS) — allowing nonmonotonic inferences — and contexts oriented (ATMS) — viewing the reasoning process under several contexts simultaneously — truth maintenance systems. It maintains a dependency graph over the objects manipulated by a problem solver, and propagates the contexts in which the nodes are valid through it. Those contexts take into account knowledge bases incompleteness and allow non monotonic inferences. Contexts are interpreted theoretically ensuring good properties of contexts while manipulated by the implementation. The system finally insures the consistency of manipulated contexts and is abble to answer requests considering several contexts simultaneously regarding this theoretical characterization.

Keywords

Truth maintenance systems — Nonmonotonic reasoning — Multiple-world reasoning — Hypothetical reasoning.

CR category: I.2.3 Deduction and theorem proving [ARTIFICIAL INTELLIGENCE]: *nonmonotonic reasoning and belief revision.*
General term: algorithms, theory.

Laboratoire ARTEMIS/Imag, BP 53X, F-38041 GRENOBLE Cedex
internet: jerome@gladiateur.imag.fr, uucp: euzenat@imag

Le système de maintenance de la vérité à propagation de contextes

Jérôme Euzenat

1.Introduction

Une inférence est définie comme une opération atomique permettant d'obtenir une nouvelle formule à partir de formules valides (voir figure 1). La propriété de monotonie, pour un système d'inférence, garantit la dérivation des théorèmes dérivés d'un certain nombre d'axiomes à partir d'un sur-ensemble de ces axiomes. Cette propriété s'exprime pour un système logique par «Si $A \vdash \alpha$ alors $A,\Gamma \vdash \alpha$».

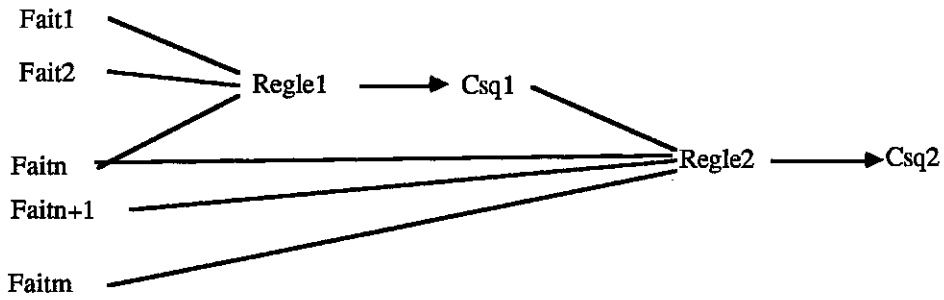

figure 1. Deux inférences successives: la première, en appliquant la règle Règle1 aux formules Fait1,...Faitn, produit la formule Csq1 — appelée conséquent —, la seconde, en appliquant la règle Règle2 aux formules Csq1, Faitn,...Faitm, produit la formule Csq2.

Les systèmes logiques classiques possèdent la propriété de monotonie. Divers systèmes, logiques ou non, qui ne possèdent pas cette propriété de monotonie ont été étudiés. Un grand nombre de systèmes implémentés — bases de données et mécanismes de défaut pour les plus répandus — ont un comportement réellement non monotone. Un exemple typique de ces inférences est l'inférence par défaut où l'absence d'une donnée dans la base — ou l'absence de réponse à une requête dans une période de temps finie — sert de fondement à une inférence. Il est alors clair que si la donnée est ajoutée ultérieurement à la base, l'inférence n'est plus valide.

La présence des inférences non monotones dans les bases de connaissance, conjuguée à la possibilité d'enregistrer les résultats de ces inférences, pose problème. En effet, la modification d'un fait ayant été utilisé dans une inférence peut entraîner l'invalidité de l'inférence. Pour maintenir la cohérence de la base il faut alors supprimer le fait inféré et ses conséquents de la base de connaissance, ce qui ne peut se faire facilement. Pour assurer cette tâche, il est possible d'utiliser des systèmes de maintenance de la vérité.

Les raisonnement menés tous les jours souffrent de l'incomplétude de la description du monde réel. C'est cette description connue de l'opérateur qui est fournie au programme. Diverses réponses ont été apportées pour pouvoir raisonner malgré cette incomplétude. Au niveau logique, il existe des logiques de défauts et des logiques non monotones qui prennent en compte le caractère non monotone du raisonnement mené. Au niveau algorithmique, diverses approches sont possibles, qui reviennent soit à compléter la description et à revenir dessus si une incohérence ou une inadéquation est détectée, soit à examiner en parallèle toutes les complétions possibles de la description. Ces deux approches se retrouvent dans deux types de systèmes de maintenance de la vérité: TMS et ATMS.

Ces dernières années, de tels systèmes se sont généralisés et beaucoup de logiciels commerciaux offrent la possibilité de gérer un raisonnement hypothétique soit en explorant plusieurs contextes à la fois, soit en procédant à une gestion de dépendances, soit en utilisant les deux approches. Un des problèmes avec les systèmes rencontrés est qu'ils ne présentent pas une sémantique très claire de leur action. L'ATMS, pourtant, est fondé sur une sémantique immédiate, et les logiques non monotones offrent un cadre attrayant pour les problèmes de non monotonie.

Le système présenté ici, le CP-TMS (pour «système de maintenance de la vérité à propagation de contextes»), se veut un système capable, à la fois de considérer plusieurs complétions de la description du monde en parallèle et d'autoriser l'utilisation d'inférences non monotones. Par ailleurs, le comportement de ce système est explicité par la signification de ce que sont les complétions (ou «environnements»).

Les trois parties suivantes introduisent les systèmes de maintenance de la vérité déjà existants. Les deux premières exposent le fonctionnement et la philosophie des systèmes de base que sont le TMS et l'ATMS. La quatrième partie est une brève revue des systèmes tentant de concilier les deux approches ainsi qu'une critique de l'approche la plus répandue.

Les parties de 5 à 10 exposent le CP-TMS proprement dit, elles concernent respectivement, les structures de données utilisées par l'algorithme, l'interprétation théorique des structures manipulées, l'algorithme détaillé et les propriétés de correction, complétude, consistance et minimalité de l'algorithme et enfin l'interprétation des requêtes par rapport au cadre théorique proposé.

Enfin, les deux dernières parties font le point sur les optimisations possibles des algorithmes proposés et la comparaison du CP-TMS avec divers systèmes préexistants. Après la conclusion, une annexe contient l'ensemble des démontrations nécessaires à cet exposé.

2.Systèmes de maintenance de la vérité à propagation

Le principe du système de maintenance de la vérité à propagation, ou TMS pour "truth maintenance system" [Doyle 79], est d'enregistrer pour chaque inférence la formule inférée au sein d'un nœud et de lui associer une justification représentant l'inférence. Une justification est un couple d'ensembles de nœuds, le premier ensemble étant l'ensemble des nœuds dont la présence dans la base est nécessaire à l'inférence (IN-liste) tandis que le second ensemble est celui des nœuds dont l'absence dans la base est nécessaire à l'inférence (OUT-liste). Les justifications ont donc la structure suivante: <{Fait1,...Faitn} {NFait1,...NFaitm}>. L'ensemble des nœuds considérés par le système, associés à leurs justifications, forme un graphe orienté nommé graphe de dépendances (voir figure 2).

Le système de maintenance de la vérité agit sur cette structure principalement quand une justification correspondant à une inférence lui est fournie. Il se charge alors de propager la validité nouvelle introduite par cette inférence au sein du graphe. Puis, si des faits contradictoires sont présents simultanément dans la base à la suite de cette inférence, le système invoque un mécanisme de rétrogression[1] chargé de supprimer cette contradiction.

Exemple 1:

La figure 2 représente un graphe de dépendances complet correspondant à l'ensemble de formules suivantes:

D	pr=	<{} {}>: D
¬A⇒B	j1=	<{} {A}>: B
¬B⇒A	j2=	<{} {B}>: A
B⇒C	j3=	<{} {B}>: C
(D∧¬E)⇒F	j4=	<{D} {E}>: F
¬(C∧F)	j5=	<{C F} {}>: ⊥
G⇒B	j6=	<{G} {}>: B

Chaque formule est suivie de la ou des justifications auxquelles elle a donné lieu — celles-ci étant nommées pour des raisons de commodité.

[1] Aussi appelé «retour-arrière».

La procédure de propagation agit elle aussi en deux passes. La première est chargée de déterminer si les noeuds ont une raison bien fondée d'être dans (IN) ou hors (OUT) de la base, c'est-à-dire soit une justification valide soit toutes ses justifications invalides en considérant que tous les nœuds qui ont le nœud inféré comme antécédent ne sont ni dans la base ni hors de la base. La seconde passe consiste à trouver des justifications faiblement fondées. Cette passe est utile quand il existe des cycles dans le graphe de dépendances. Le système cherche alors à attribuer arbitrairement, mais de manière consistante avec le graphe, l'état du nœud (IN ou OUT). Il faut distinguer entre cycles pairs — représentant une configuration de graphe valide — et cycles impairs — représentant un paradoxe — suivant la parité du nombre d'inverseurs à franchir pour aller d'un nœud à lui-même dans un cycle. Par exemple, dans le graphe de la figure 2, le cycle pair entre A et B autorise deux configurations du graphe.

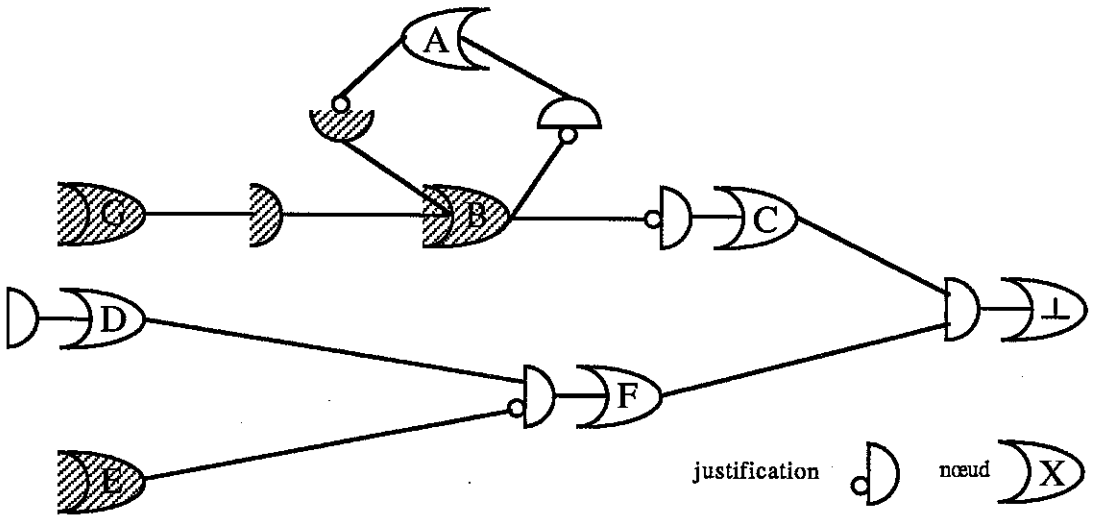

figure 2. Un graphe de dépendances est représenté comme un circuit logique dont les portes «ou» sont des nœuds et les portes «et» des justifications où les nœuds de la IN-liste arrivent directement, alors que les nœuds de la OUT-liste passent au travers d'un inverseur. Les nœuds dont une justification a ses deux listes vides représentent des formules toujours valides car elles n'ont pas besoin d'être inférées, ce sont les prémisses du raisonnement (D). Les nœuds et justifications en blanc sont considérés comme valides alors que ceux en grisé sont invalides. Bien sûr, la propagation de ces valeurs est faite selon les règles dictées par les composants du circuit. Les faits dans la base sont ainsi assurés d'avoir une justification valide — c'est-à-dire correspondant à une inférence valide.

La procédure de rétrogression est activée quand un nœud IN est marqué comme contradictoire, elle tente alors de supprimer ce nœud de la base. Pour cela elle choisit l'une des hypothèses de ce nœud qui va être invalidée. Une hypothèse est un nœud dont la justification courante possède sa OUT-liste non vide, c'est-à-dire que l'absence dans la base de certaines données entraîne la validité de cette hypothèse. Elle sera donc invalidée en ajoutant dans la base un des nœuds de cette seconde liste.

Les systèmes de maintenance de la vérité à propagation permettent donc, à partir d'une base de donnée initiale et d'un ensemble d'inférences produites, éventuellement non monotones, de garantir la validité des faits dans la base vis-à-vis de ces inférences et données initiales. Ils permettent en outre, en cas de violation de ce qui représente une contrainte d'intégrité, de rétablir la cohérence de la base en la complétant.

3.Systèmes de maintenance de la vérité à contextes

Le principe des systèmes de maintenance de la vérité à contextes, ou ATMS pour "assumption-based TMS" [De Kleer 86a], est d'enregistrer les contextes dans lesquelles elles sont valides, au lieu d'enregistrer la validité instantanée des formules. Ainsi, quand une inférence permet de conclure la formule C à partir des formules Fait1,...Faitn, un ensemble de données — appelé environnement — va être ajouté à l'ensemble des environnements de C — appelé étiquette. Cet environnement est l'intersection des environnements de Fait1,...Faitn. Les hypothèses sont enregistrées avec une étiquette contenant l'environnement composé d'eux-même. C'est donc sur celles-ci que seront bâtis les environnements des formules inférées. Les inférences gérées par les systèmes de maintenance de la vérité à contextes ne sont jamais non monotones, seules les formules nécessaires à l'inférence sont considérées.

Le comportement du système est très clairement défini puisque, pour une formule considérée, il construit des étiquettes avec les propriétés suivantes:

- Consistance: aucun des environnements de l'étiquette n'inclut un environnement connu comme inconsistant.
- Complétude: pour tout ensemble E connu d'hypothèses permettant de dériver la formule considérée, il existe un environnement de l'étiquette qui contient E.
- Correction: tous les environnements de l'étiquette permettent d'inférer la formule considérée.
- Minimalité: il n'existe pas d'environnement de l'étiquette qui en contienne un autre.

Ainsi, pour l'inférence représentée graphiquement dans la figure 1, le conséquent Csq2 a pour environnement [Fait1,...Faitn, Faitn+1,...Faitm] sachant que chaque Faiti ($1 \leq i \leq m$) a pour environnement [Faiti].

Les environnements peuvent être considérés comme étant structurés dans un treillis fondé sur la relation d'inclusion (voir figure 3). La condition de minimalité revient alors à signaler le plus petit environnement dans lequel l'inférence est valide et le treillis permet de considérer cette inférence comme valide dans tous les environnements supérieurs.

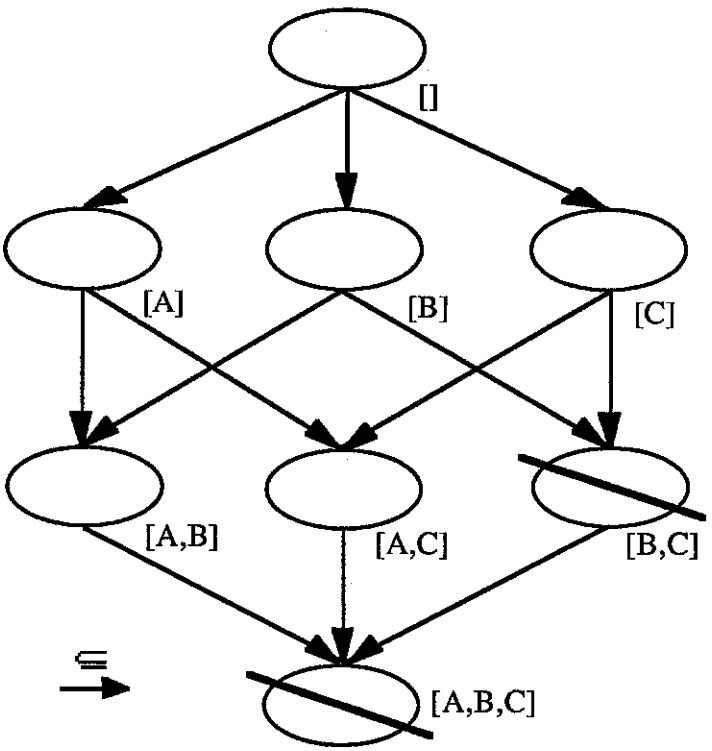

figure 3. Le treillis de contextes structuré par la relation d'inclusion et construit sur les hypothèses A, B et C dans lequel le contexte [B C] est connu comme inconsistant.

À chaque inférence produite, le système construit la nouvelle étiquette pour la formule inférée. Si cette formule est contradictoire, ce qui correspond à violer une contrainte d'intégrité, alors l'environnement construit est connu comme contradictoire et tous les environnements contenant celui-ci sont ôtés de toutes les étiquettes des autres nœuds.

Une requête est interprétée non plus dans l'absolu, comme c'était le cas avec le système à propagation et comme c'est le cas dans la plupart des bases de données, mais en relation avec un contexte courant. Le sens de la requête n'est donc plus «est-ce que C?» mais plutôt «est-ce que C si Fait1,...Faitn?». Si le contexte courant est inclus dans l'un des environnements de l'étiquette de C, le système répondra par l'affirmative.

Les systèmes de maintenance de la vérité à contextes permettent, à partir d'un ensemble d'inférences produites, nécessairement monotones, d'enregistrer les prémisses nécessaires à la validité de chaque formule. Ils enregistrent en outre les environnements conduisant à une violation des contraintes d'intégrité. Les réponses à une requête se font en fonction d'un contexte courant, fragment de la base, dans lequel se place l'utilisateur.

4.Systèmes étendus

Le système de maintenance de la vérité décrit par Jon Doyle ne développe pas la notion de contexte alors que le système défini par Johan De Kleer ne permet pas l'utilisation d'inférences non monotones. Aussi serait-il très avantageux de pouvoir créer un système disposant des deux notions; diverses tentatives ont déjà été faites dans ce sens.

4.1.Extensions de l'ATMS

C'est ainsi que plusieurs extensions de l'ATMS vers les inférences non monotones ont été proposées. Le but de telles extensions est de pouvoir exprimer des inférences non monotones et de les faire prendre en compte par l'ATMS. Le problème principal est que l'ATMS est fondé sur les propriétés de monotonie de l'inclusion ensembliste. Différents stratagèmes ont donc été imaginés pour tourner cette propriété de monotonie, ils nécessitent la génération de nombreux environnements dont certains seront inconsistants.

4.1.1. De Kleer 86

Johan De Kleer [De Kleer 86b] tente d'introduire les inférences non monotones à l'aide d'une primitive CHOOSE représentant la disjonction ($O \vee N$). Ainsi la justification non monotone $<\{i1,...in\}\{o1,...om\}>: c$[1] est traduite par:

$$\{\} : O \vee N$$
$$\{O, N\} : \perp$$[2]
$$\forall j;\ 1 \leq j \leq m,\ \{i1,...in,oj\} : O$$
$$\{i1,...in,N\} : c$$

Ce mécanisme en lui-même est déjà lourd à utiliser, mais la difficulté est encore renforcée par l'utilisation au sein de l'ATMS d'une règle d'«hyperrésolution» destinée à raisonner sur les environnements inconsistants à l'aide du CHOOSE.

4.1.2. Dressler 88

Oskar Dressler propose une autre implémentation [Dressler 88]: pour chaque nœud N, un nœud OUT(N) est créé dont l'étiquette n'est jamais calculée — il s'agit de l'ensemble des contextes dans lesquels N n'est pas valide. Cette méthode introduit donc l'hypothèse du monde clos dans le système de maintenance de la vérité. Il est, par ailleurs, créé une hypothèse Out(N) et les justifications

$$\{Out(N)\} : OUT(N)$$
$$\{Out(N), N\} : \perp$$

sont fournies au système. Ainsi, pour la justification examinée plus haut, le système l'enregistre comme:

$$\forall j;\ 1 \leq j \leq m\ \{oj\} : O$$

---

[1] La formulation j: f, signifie que la formule f est justifiée par la justification j. Dans le cas de l'ATMS, les justifications ne contenant qu'une IN-liste, celle-ci est simplement placée devant la formule f.

[2] Dans tout ce qui suit, le symbole $\perp$ (lire inconsistant) permet d'exprimer une contrainte. Comme à tous les autres symboles fournis au système de maintenance de la vérité, il lui est associé un nœud marqué comme inconsistant (voir figure 2).

$$\{i1,\ldots in,\text{OUT}(O)\} : c.$$

Enfin, le traitement des requêtes est modifié de façon à répondre positivement quand il existe une extension du contexte d'interrogation dans lequel la formule est valide.

4.1.3. De Kleer 88

De Kleer [De Kleer 88] a récemment proposé un système semblable: à chaque hypothèse H utilisée dans la OUT-liste d'une justification non monotone, est associé un nœud ¬H ne pouvant être justifié. Il est ainsi possible de remplacer CHOOSE({O,N}) par

$$\{\neg O,\neg N\} : \bot$$

et l'algorithme de propagation se chargera de dériver les étiquettes correctes, cependant il n'assurera plus la complétude des étiquettes. Par ailleurs, à chaque découverte d'un environnement inconsistant E contenant des hypothèses H pour lesquelles ¬H a été défini, l'environnement E\{H} sera ajouté à l'étiquette de ¬H évitant ainsi la règle d'hyperrésolution. Ce dernier système n'est pas complètement décrit, en particulier Johan De Kleer ne mentionne pas la signification des étiquettes obtenues.

Ces trois premières extensions de l'ATMS se proposent d'introduire de nouveaux mécanismes en son sein et génèrent un nombre considérable de nœuds et de justifications gérant un grand nombre d'environnements. En particulier, les extensions multiples ne sont manipulables que parce que les membres de OUT-listes sont forcement des hypothèses. Ils modifient peu, par contre, le fonctionnement de l'ATMS et utilisent au contraire toutes ses propriétés.

4.2.Critique d'ART

Le logiciel ART [Williams 84, Clayton 86] a été développé parallèlement à l'ATMS. Bien que les algorithmes qu'il emploie ne soient pas connus, non plus d'ailleurs qu'une — hypothétique — caractérisation du fonctionnement du programme, on peut supposer que son fonctionnement est différent de celui de l'ATMS malgré une interface semblable. ART est certainement plus proche du système MBR [Martins& 83, 88].

Comme l'ATMS, ART autorise la création d'hypothèses (ou "viewpoints") et évalue ses inférences au sein des environnements les plus généraux dans lesquels elles peuvent prendre place. Mais il autorise aussi la présence d'antécédents négatifs dans les règles d'inférence. Par voie de conséquence, les environnements manipulés sont de la forme C|C1,...Cn correspondant à l'ensemble des "viewpoints" plus spécifiques que C sauf ceux plus spécifiques que C1,...Cn. Partant de là, le système semble se comporter de manière similaire à l'ATMS mais il faut remarquer un certain nombre de différences:

(1) Une justification représentant une inférence non monotone pourrait donner lieu à une étiquette similaire à celle donnée pour les justifications classiques. Cependant, l'algorithme permettant d'obtenir cette étiquette est très complexe, il doit utiliser des moyens permettant de trouver des environnements plus généraux et plus spécifiques à partir d'environnements avec restrictions. La structure des environnements s'éloigne donc de celle développée par De Kleer, le système ne peut exploiter les mêmes idées simples. Au lieu de persister à ne considérer qu'une étiquette par fait, le système ne considère pour un fait qu'un seul environnement avec plusieurs restrictions. Ceci oblige à dupliquer les faits quand un environnement est éclaté. C'est pénalisant car les environnements sont bien plus souvent éclatés que dans le simple ATMS. Ce principe revient, à l'instar de MBR, à ne plus considérer qu'une justification par nœud.

Exemple 2:

Soit la suite de règles suivantes activées l'une après l'autre:

```
(defrule ctxt1 () => (sprout (assert A))) -> C1
(defrule ctxt2 A => (sprout (assert B))) -> C1.1
(defrule ctxt3 A => (sprout (retract A))) -> C1.2
(defrule ctxt4 B => (sprout (retract B))) -> C.1.1.1
```

On a alors les contextes suivants A: C1|C1.2, B: C1.1|C1.1.1. Après l'utilisation de la règle

```
(defrule infC A (not B) => (assert C))
```

le contexte de C pose problème puisqu'il s'agit de C1.1.1 et C1|C1.1,C1.2, le fait C sera donc représenté par deux faits chacun ayant l'un de ces environnements pour étiquette.

(2) Le système offre la possibilité de masquer dans un environnement la présence d'une formule qui doit y être présente au vu du graphe des contextes et de celui des dépendances. Plus ce genre de possibilité est utilisée, plus le sens d'un environnement diffère du sens intuitif qui pouvait être rencontré chez De Kleer. Comme dans les systèmes à base de règles classiques, l'utilisation du `retract` n'a donc pratiquement aucune base logique.

(3) Le système développé par Johan De Kleer est totalement monotone, il tire toutes ses bonnes propriétés de sa monotonie, en particulier si une formule est valide dans un contexte, elle l'est dans tous les contextes plus complets. ART introduit la non monotonie avec l'utilisation des antécédents négatifs. Si la formule F est inférée grâce à la règle $F1 \wedge \neg F2 \Rightarrow F$ et que F2 n'a été trouvé valide nulle part, alors l'étiquette de F sera celle de F1. Si ultérieurement le fait F2 est inféré dans un contexte ou F1 est valide, il faudrait reconsidérer l'étiquette de F. Ceci n'est pas fait de façon automatique, mais peut être commandé par l'utilisation des dépendances logiques ("logicals"). Le système de dépendances logiques devrait être, en toute rigueur, un TMS.

(4) L'héritage de l'inconsistance n'obéit pas aux mêmes règles que l'héritage des faits: ART invalide tous les environnements plus spécifiques de ces derniers, or il peut en être qui ne soient pas inconsistants (voir exemple 3).

Exemple 3:

Soit la suite de règles suivantes activées l'une après l'autre:

```
(defrule hypA () => (hypothesize (assert A))) -> C1
(defrule hypB A => (hypothesize (assert B))) -> C1.1
(defrule poisA A (not B) => (poison ""))
```

Dans un tel programme le contexte de A devrait être déclaré comme inconsistant ("poisoned") mais pas celui de B. ART va supprimer le contexte de A ainsi que tous les contextes plus spécifiques que celui de A — entre autres celui de B.

Le problème soulevé ici est que l'instruction `poison` utilisée comme les autres instructions en partie droite n'a pas le même champ d'application que les autres instructions. Cela peut tout à fait être compris et utilisé proprement mais le manque d'homogénéité est flagrant.

À partir d'un ATMS ou d'un système similaire, l'idée initiale pour l'étendre vers un système intégrant des inférences non monotones est d'introduire la notion de dépendances entre les faits inférés et leurs antécédents. Mais cette notion est à elle seule insuffisante, il faut alors introduire un véritable TMS. Le problème est que certaines fonctions du TMS et de l'ATMS font double emploi. Il est donc tout naturel de concevoir un système intégré offrant les fonctionnalités d'un TMS et d'un ATMS.

Au lieu d'étendre l'ATMS par une couche au-dessus ou des mécanismes internes, afin qu'il puisse utiliser des justifications non monotones, l'approche présentée ici étend le TMS afin qu'il puisse utiliser les contextes. Cette approche suppose donc de modifier grandement le TMS afin qu'il ne tienne plus compte de l'état (IN ou OUT) dans lequel se trouve un objet mais qu'il se contente de propager l'étiquette de cet objet. Cette extension utilise des environnements particuliers qui tiennent compte de la nature des inférences non monotones. Par conséquent il faut définir clairement la signification des étiquettes des nœuds pour pouvoir leur associer les propriétés de correction, complétude, minimalité et consistance. Ce travail conduit à donner aussi diverses interprétation possibles aux requêtes.

5.Modifications de l'algorithme du système de maintenance de la vérité à propagation

La caractérisation des deux systèmes de maintenance de la vérité (TMS et ATMS) fournie dans [Brown& 87] est faite en terme de systèmes d'équations booléennes. Dans le cas du TMS, le système est résolu complètement et donne la valeur de chaque nœud (son état). Dans le cas de l'ATMS chaque nœud est évalué en une formule sous forme normale disjonctive (sans négations) dépendant des hypothèses (son étiquette).

L'algorithme présenté ici reprend ces idées et étend le traitement de l'ATMS aux formules avec négations (voir §12.4). À chaque nœud sera donc associée une formule, fonction propositionnelle des nœuds représentant les hypothèses: son étiquette.

5.1.Les étiquettes

L'étiquette d'un nœud est donc une formule qui est valide quand le nœud est valide et invalide lorsque le nœud est invalide. À l'instar de celle de Johan De Kleer, l'étiquette a aussi la propriété d'être consistante, c'est-à-dire que quand elle est valide, le nœud $\perp$ n'est pas valide. Alors que dans l'ATMS, un environnement est un ensemble d'hypothèses, il est nécessaire pour tenir compte des inférences non monotones de changer la structure des environnements. En effet, une justification ayant une partie négative dans ses prémisses n'est plus valide dans tout contexte où ses prémisses positives sont valides mais dans tout contexte où ses prémisses positives sont valides et où ses prémisses négatives sont invalides. Il faut donc introduire ces prémisses négatives dans les environnements correspondant aux inférences.

Un certain nombre de définitions vont permettre de fixer le vocabulaire utilisé dans ce qui suit: la principale modification des algorithmes concerne la propagation des étiquettes. L'*étiquette* est une formule, qui, comme toute formule, peut s'exprimer en forme normale disjonctive:

$(a11 \wedge \ldots a1p_1 \wedge \neg b11 \wedge \ldots \neg b1q_1) \vee \ldots (an1 \wedge \ldots anp_n \wedge \neg bn1 \wedge \ldots \neg bnq_n)$.

Un *environnement* sera l'un des membres de cette disjonction. Cette forme normale disjonctive peut s'exprimer par une forme compactée — unique — rassemblant tous les disjoints ayant le même ensemble de formules positives:

$(a11 \wedge \ldots a1p_1 \wedge ((\neg b111 \wedge \ldots \neg b11q_{11}) \vee \ldots (\neg b1m(1)1 \wedge \ldots \neg b1m(1)q_{1m(1)}))) \vee \ldots$
$(an1 \wedge \ldots anp_n \wedge ((\neg bn11 \wedge \ldots \neg bn1q_{n1}) \vee \ldots (\neg bnm(n)1 \wedge \ldots \neg bnm(n)q_{nm(n)})))$

chaque conjonction $ai1 \wedge \ldots aip_i \wedge ((\neg bi11 \wedge \ldots \neg bi1q_{i1}) \vee \ldots (\neg bim1 \wedge \ldots \neg bimq_{im}))$ sera nommée un *environnement global*, chaque conjonction $ai1 \wedge \ldots aip_i$ sera appelée un *ensemble d'axiomes*, chaque disjonction $(\neg bi11 \wedge \ldots \neg bi1q_{i1}) \vee \ldots (\neg bim1 \wedge \ldots \neg bimq_{im})$ sera appelée *ensemble de restrictions* et chaque conjonction $(\neg bij1 \wedge \ldots \neg bijq_{ij})$ sera nommée *restriction*. Un environnement est donc la conjonction d'un ensemble d'axiomes et d'une restriction.

C'est sous la forme compactée que seront manipulées et enregistrées les étiquettes dans l'implémentation. Les étiquettes seront représentées par des structures de la forme suivante:

$\{<\{a11 \ldots a1p_1\}\ \{\{b111 \ldots b11q_{11}\} \ldots \{b1m(1)1 \ldots b1m(1)q_{1m(1)}\}\}> \ldots$
$<\{an1, \ldots anp_n\}\ \{\{bn11 \ldots bn1q_{n1}\} \ldots \{bnm(n)1 \ldots bnm(n)q_{nm(n)}\}\}>\}$

Les étiquettes sont donc des ensembles de couples représentant les environnements globaux, le premier élément d'un environnement global est sa liste d'axiomes représentée par un ensemble de nœuds, son second élément est son ensemble de restrictions. L'ensemble de restrictions est un ensemble d'ensembles de nœuds représentant les restrictions. Les opérateurs ensemblistes seront donc principalement utilisés dans la description des algorithmes, deux fonctions supplémentaires `ax(Env)` et `rt(Env)` sont introduites qui rendent respectivement l'ensemble d'axiomes et l'ensemble de restrictions d'un environnement global.

Les notations choisies rejoignent celles utilisées pour décrire un système tel qu'ART [Clayton 86]. Un environnement n'est plus un ensemble d'hypothèses mais un ensemble d'hypothèses associé à une restriction. Une restriction étant elle-même un ensemble d'hypothèses. Ainsi $\{H1 \ldots Hn\} \mid \{H11 \ldots H1n_1\}, \ldots \{Hj1 \ldots Hjn_j\}$ est un environnement global embrassant tous les environnements plus spécifiques que $\{H1 \ldots Hn\}$ sauf ceux qui contiennent $\{H11 \ldots H1n_1\}$ ou... $\{Hj1 \ldots Hjn_j\}$.

Dans le texte, un environnement [A1,...An|R1,...Rm] désignera donc l'ensemble des environnements complets où A1,...An sont valides et R1,...Rm sont invalides. De tels environnements pourront aussi être désignés par [{A1,...An}/{R1,...Rm}]. Comme dans l'ATMS ils peuvent être classés grâce à la relation «est plus complet que» (voir figure 4).

Malheureusement, l'ensemble de tous les environnements sur un ensemble d'hypothèses $\mathcal{H}$ muni de cette relation ne constitue pas un treillis, et qui plus est, la nature non monotone des inférences interdit l'héritage, suivant la relation, des formules inférées. Il est donc nécessaire pour caractériser le comportement du programme d'aller plus loin dans la spécification et la signification des environnements.

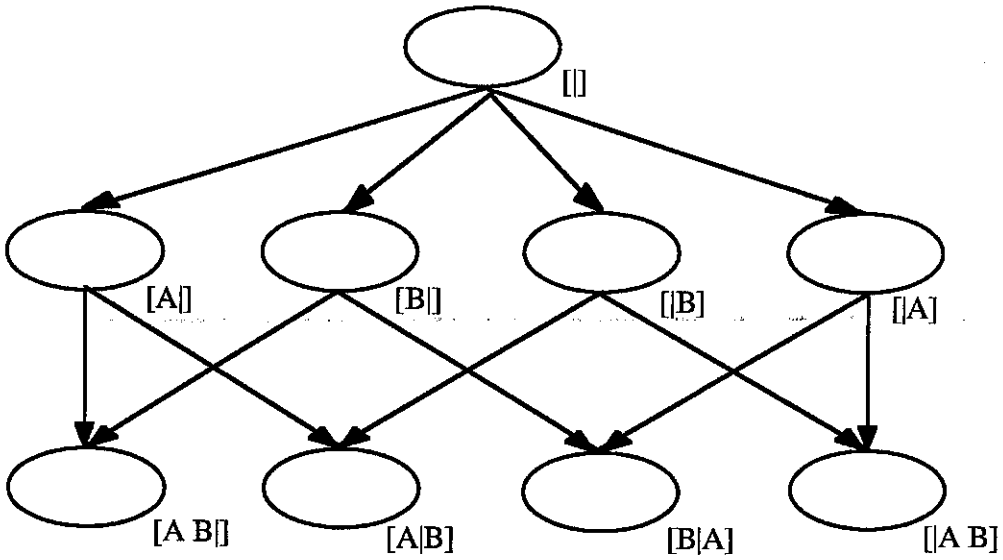

figure 4. Cette figure présente le graphe de la relation «est plus complet que» avec seulement deux hypothèses possibles: A et B. Il y a donc trois degrés de complétude suivant les statuts accordés aux deux hypothèses, le niveau de degré 2 (où les deux hypothèses sont présentes dans les environnements) constitue l'ensemble des environnements complets.

5.2.Les nœuds et justifications

Chaque inférence est fournie au système sous la forme d'une justification. Les justifications de ce système diffèrent de celles de l'ATMS car elles représentent des inférences non monotones, elles ont donc la même structure que dans le TMS, structure très simple:

- INLISTE: La liste des nœuds nécessaires à l'inférence.
- OUTLISTE: La liste des nœuds dont l'invalidité est nécessaire à l'inférence.

À la justification proprement dite sont rajoutés deux attributs présents dans la représentation de la justification:

- CONS: Le conséquent de l'inférence.
- ETIQ: L'étiquette correspondant à la justification.

Dans tous les systèmes de maintenance de la vérité, la structure principale représentant les données du système de raisonnement est le nœud. Le CP-TMS est inspiré du système de Jon Doyle, il en va de même de sa structure de nœud. Cependant, une grande partie des concepts développés pour le TMS reposent sur la notion de supports pour la validité — ou l'invalidité — d'un nœud. Une fois évacuée la notion de validité, il n'est plus besoin de support et des nombreuses relations de dépendance qui y sont liées. La composition des nœuds du CP-TMS se présente alors ainsi:

- FAIT: Contient le fait représenté par le nœud. À chaque fait manipulé par le système de raisonnement correspond un unique nœud.
- ETIQUETTE: C'est l'étiquette renfermant l'ensemble des environnements globaux dans lesquels le fait est valide.
- LISTEDEJUST: C'est l'ensemble des justifications représentant les inférences dont le conséquent est le fait.
- ANT: C'est l'ensemble des nœuds présents dans l'une des justifications du nœud.
  $ANT(n)=\{N; \exists J \in LISTEDEJUST[n]; N \in INLISTE[J] \vee N \in OUTLISTE[J] \}$
- ANT*: C'est la clôture transitive de ANT.
  $ANT^*(n)=\{N; N \in ANT[n] \vee \exists N' \in ANT[n]; N \in ANT^*[N']\}$
- CSQ: C'est l'ensemble des nœuds justifiés à l'aide du nœud.
  $CSQ(n)=\{N ; \exists J \in LISTEDEJUST[N]; n \in INLISTE[J] \vee n \in OUTLISTE[J] \}$
- CSQ*: C'est la clôture transitive de CSQ.
  $CSQ^*(n)=\{N; N \in CSQ[n] \vee \exists N' \in CSQ[n]; N \in CSQ^*[N'] \}$
- Les nœuds contiennent en outre un certain nombre de marqueurs utilisés par les différents algorithmes, ainsi que l'indicateur d'hypothèse (voir §5.3).

Cette description peut sembler circulaire, mais elle ne l'est pas car les justifications des prémisses ne contiennent aucun nœud et les nœuds non inférés n'ont pas de justification.

Une telle structure est beaucoup plus aisée à mettre à jour que celle de Doyle, tout d'abord à cause de sa moindre complexité mais surtout parce que les valeurs présentes dans la structure sont constantes et ne dépendent pas de la validité des nœuds. La mise à jour de la structure de nœud se fait donc simplement quand une justification est ajoutée dans le système. La première phase de cette mise à jour est la mise à jour des nœuds dans leur ensemble, la seconde est produite par l'algorithme de propagation des étiquettes, plus complexe, qui est décrit plus bas. L'algorithme de mise à jour des nœuds est le suivant:

```
MISE-A-JOUR( noeud just ) =
[ CONS[just] <- noeud
  LISTEDEJUST[noeud] <- LISTEDEJUST[noeud] ∪ { just }
  PourTout autre-noeud∈(INLISTE[just] ∪ OUTLISTE[just])
  Faire
    Si noeud∉CSQ[autre-noeud]
    Alors [CSQ[autre-noeud] <- CSQ[autre-noeud] ∪ { noeud }
              CSQ*[autre-noeud] <- CSQ*[autre-noeud] ∪ CSQ*[noeud]
              PourTout troisieme-noeud∈ANT*[autre-noeud]
              Faire CSQ*[troisieme-noeud]
                     <- CSQ*[troisieme-noeud] ∪ CSQ*[noeud]
              ANT*[noeud] <- ANT*[noeud] ∪ ANT*[autre-noeud]
              PourTout troisieme-noeud∈CSQ*[noeud]
              Faire ANT*[troisieme-noeud]
                     <- ANT*[troisieme-noeud] ∪ ANT*[autre-noeud] ]
  ANT[noeud] <- ANT[noeud] ∪ INLISTE[just] ∪ OUTLISTE[just] ]
```

5.3.L'ajout des hypothèses

Les hypothèses sont des faits que l'utilisateur peut considérer selon sa volonté comme valides ou invalides — en fait que l'utilisateur peut décider ou non de poser comme axiomes de son raisonnement. Une hypothèse est donc définie comme telle par l'utilisateur. Ce sont ces hypothèses et uniquement elles qui seront présentes dans les étiquettes. C'est en décidant qu'un certain nombre de ces hypothèses sont considérées comme vraies et les autres fausses — voire même sans présumer de la validité des autres — que l'utilisateur pourra interroger le système pour savoir quelles conclusions tirer de ces hypothèses. Il faut bien se rendre compte que les hypothèses, si elles représentent des formules bien formées du langage d'expression du système de raisonnement, ne sont rien de plus que des propositions pour le système de maintenance de la vérité. Le TMS ne descend donc jamais dans la structure de l'hypothèse.

Une hypothèse est définie comme un nœud dont l'étiquette contient un environnement ayant pour ensemble d'axiomes le singleton composé de ce nœud:

H: {... [H...|...] ... }

elle est initialement déclarée en ajoutant l'environnement [H|] à son étiquette.

Tout pourrait être déclaré d'emblée comme hypothèse, mais cela alourdirait toutes les étiquettes et donc le système globalement. Les seules hypothèses déclarées sont donc celles de l'utilisateur. Toutefois des hypothèses pourront être déclarées implicitement comme telles, par exemple pour simuler le comportement du TMS de Doyle (voir §8.3). Si tous les objets ne sont pas des hypothèses, il est cependant possible de déclarer un fait comme une nouvelle hypothèse à n'importe quel moment. Par ailleurs, il est aisé de supprimer une hypothèse — au sens de faire en sorte que le nœud correspondant ne soit plus considéré comme une hypothèse — ce qui simplifie les étiquettes, les résultats fournis par le système et la tâche de celui-ci.

5.4. Algorithme de mise à jour des étiquettes

Le système de maintenance de la vérité, comme c'est le cas généralement, est sollicité par le système de raisonnement. Ces sollicitations peuvent être les suivantes:

- Un axiome est posé par le système de raisonnement comme un fait toujours valide. Il est fourni au système de maintenance de la vérité avec une justification contenant des IN- et OUT-listes vides et a donc dans son étiquette un environnement vide — c'est-à-dire sans axiome ni restriction. La justification est donc la suivante:

  $$<\{\}\{\}>: a$$

- Une hypothèse est posée afin de pouvoir être prise comme prémisse provisoire, elle devra apparaître dans les étiquettes. Dans l'absolu elle peut être fournie au système de maintenance de la vérité comme une justification avec une IN-liste composée du nœud justifié et d'une OUT-liste vide:

  $$<\{h\}\{\}>: h$$

  Mais, afin de rendre l'implémentation plus simple, les hypothèses seront marquées par un indicateur et son étiquette contient donc un environnement composé uniquement de l'hypothèse.

- Une inférence est fournie au système comme une justification, avec les antécédents positifs et négatifs de l'inférence dans les IN- et OUT-listes. C'est la configuration classique:

  $$<\{i1,\ldots in\}\{o1,\ldots om\}>: f$$

  Il est possible, au niveau du système de raisonnement de décider de ne fournir que certains antécédents au système de maintenance de la vérité — implémentant en cela les dépendances logiques — ou "logicals" — du logiciel ART [Williams 84]. Cela est indépendant du fonctionnement du système de maintenance de la vérité mais permet d'y faire appel quand cela se révèle nécessaire.

- Une contrainte permet d'interdire la présence dans la base d'un certain ensemble de formules. Elle est donc décrite comme une inférence ayant les nœuds de cet ensemble pour antécédents positifs et le fait $\perp$ pour conséquence. La justification introduite dans le système sera donc composée de la liste des formules inconsistantes dans sa IN-liste, une OUT-liste vide — ceci n'est pas obligatoire — et justifiera le fait $\perp$:

  $$<\{i1,\ldots in\}\{\}>: \perp$$

L'interface entre le système de raisonnement et le système de maintenance de la vérité a donc gardé toute sa simplicité: il ne s'agit jamais que de fournir une inférence au système. Cette inférence sera transformée en justification et cette justification insérée dans le graphe de dépendances. L'insertion de la justification se fait grâce à l'algorithme décrit plus haut (voir §5.2) et donne lieu à la propagation des étiquettes à travers le graphe.

Le système de maintenance de la vérité à propagation, propageait la validité à travers ce graphe. Le CP-TMS, lui, ne intéresse plus à la validité absolue des formules mais aux ensembles d'hypothèses sous lesquelles ces formules sont valides. La forme réduite de ces ensembles d'hypothèses est une étiquette. La propagation du système de Jon Doyle agit en deux phases: la première phase cherche des supports bien fondés pour les nœuds et la seconde, en désespoir de cause, cherche des supports circulaires. Ces deux phases étant liées à la recherche de la validité absolue, elles n'ont plus de raison d'être dans le système propageant les étiquettes et seront donc réduite à une phase unique décrite dans la suite (voir §7, 8 et 9).

Le système de Jon Doyle contient aussi une phase de rétrogression utilisée quand le nœud $\perp$ devient valide. Ce nœud n'étant plus valide absolument, la phase de rétrogression est remplacée par une phase de rétablissement de la consistance du graphe de dépendances (voir §8.2). En effet, au lieu de garantir que le nœud $\perp$ n'est pas valide dans l'environnement courant, le système se charge de garantir que l'utilisateur ne pourra pas se placer dans un quelconque environnement permettant de valider $\perp$ — il ne pourra pas utiliser d'ensembles d'hypothèses permettant de dériver $\perp$.

Une requête permet au système de raisonnement de consulter une partie du graphe à savoir la validité d'une formule sous un certain nombre d'hypothèses. Cet aspect du système est discuté plus loin (voir §10).

6.Approche théorique

Tout au long de l'implémentation, le programme du CP-TMS manipule des étiquettes et des environnements. Il est intéressant de savoir à quoi peuvent correspondre ces environnements. L'approche sémantique exposée ici est analogue à la sémantique couramment utilisée en logique.

Dans la suite on considèrera un ensemble $\mathcal{H}$ d'hypothèses et un ensemble $\mathcal{N}$ de nœuds ou formules (avec, bien entendu, $\mathcal{H}\subseteq\mathcal{N}$). L'ensemble $\mathcal{C}$ des nœuds déclarés comme inconsistants sera réduit, pour des raisons d'exposé, à l'ensemble $\{\bot\}$ (et donc $\bot\in\mathcal{N}\setminus\mathcal{H}$).Un environnement complet qui décrit donc complètement l'état du monde est un environnement où la validité de toutes les hypothèses de $\mathcal{H}$ est fixée. Classiquement, un environnement [A/R] n'est pas complet, il ne peut donc représenter complètement l'état du monde. La signification d'un tel environnement est qu'il représente l'ensemble des environnements complets plus complets que lui.

6.1.Valuations

Une *valuation* est un fonction propositionnelle ν: H$\longrightarrow${0,1}, H$\subseteq\mathcal{H}$ qui fixe la validité de certaines hypothèses et l'invalidité d'autres. Un environnement [A/R] d'une étiquette représente une valuation fournie en extension:

$$\begin{array}{lll} A\cup R & \longrightarrow & \{0,1\} \\ n & \longmapsto & \text{si } n\in A\ \nu(n)=1 \\ & & \text{si } n\in R\ \nu(n)=0. \end{array}$$

Une valuation est dite une *valuation complète* si H=$\mathcal{H}$.

Une valuation ν est dite *plus complète* qu'une valuation ν' ssi H'$\subseteq$H. L'ensemble des complétions de ν est l'ensemble des environnements dont la valuation correspondante est plus complète que celle correspondant à ν:

Comp([A/R])={ [A$\cup$A'/R$\cup$R']; (R$\cap$A')$\cup$(R'$\cap$A)$\cup$(A'$\cap$R')={} & R'$\cup$A'$\subseteq\mathcal{H}$ }.

La condition supplémentaire ajoutée dans la définition de l'ensemble permet d'éviter d'avoir des environnements ne signifiant rien comme ceux où des éléments de la restriction apparaissent dans l'ensemble d'axiomes.

Théorème 1: Comp([A/R]) = {[A$\cup$A'/R$\cup$R'];A'$\cup$R'$\subseteq\mathcal{H}\setminus$(A$\cup$R) & A'$\cap$R'={} }

L'ensemble des complétions maximales de [A/R] est l'ensemble des complétions complètes de [A/R] : Cmax([A/R]) = {[A$\cup$A'/R$\cup$R']$\in$Comp([A/R]); A$\cup$A'$\cup$R$\cup$R'=$\mathcal{H}$ }.

Théorème 2:

Cmax([A/R]) = {[A$\cup$A'/R$\cup$R']$\in$Comp([A/R]); A'$\cup$R'=$\mathcal{H}\setminus$(A$\cup$R) & A'$\cap$R'={} }

La *valuation complétée canoniquement* de ν est la valuation $\nu+$ où $\forall h\in H$ $\nu+(h)=\nu(h)$ et $\forall h\in\mathcal{H}\setminus H$ $\nu+(h)=0$, c'est-à-dire une complétion sous l'hypothèse du monde clos.

6.2.Interprétation

La notion d'interprétation recouvre celle présente dans les logiques classiques, elle va permettre d'étendre une valuation à l'ensemble de formules manipulables par le système. L'extension se fait par le biais du graphe de dépendances qui permet de propager la validité à partir des éléments de l'environnement.

Une *interprétation* à partir de la valuation ν est une fonction propositionelle I: $\mathcal{N}\longrightarrow${0,1} construite à l'aide des règles suivantes:

$\forall n\in\mathcal{N}$,
si $\nu(n)=1$ alors I(n)=1
sinon, si $\exists j\in$LISTEDEJUST[n]; $\forall h\in$INLISTE[j] I(h)=1 & $\forall h\in$OUTLISTE[j] I(h)=0
alors I(n)=1
sinon I(n)=0.

À une valuation peuvent correspondre plusieurs interprétations comme le montre cette définition récursive qui peut être transformée en équation de point fixe. L'ensemble des interprétations construites à partir de la valuation [A/R] sera noté IntInd([A/R]).

Exemple 4:

Un exemple classique est celui où l'ensemble des inférences correspond à l'ensemble $\{\neg a\Rightarrow b, \neg b\Rightarrow a\}$. Sous le simple environnement [|], il existe deux interprétations possibles: {a} et {b}.

À une interprétation correspond une *valuation interprétée* ν_I:$\mathcal{H} \longrightarrow \{1,0\}$; $\nu_I(h)=I(h)$.

Une interprétation I est dite *soutenable* (respectivement *insoutenable*) si et seulement si $I(\bot)=0$ (respectivement $I(\bot)=1$).

Une valuation ν est dite *consistante* si aucune des interprétations construites à partir de ν n'est soutenable. Si une valuation n'est pas consistante elle est dite *inconsistante*.

Une *valuation étendue* ν' d'une valuation ν est une valuation telle que

$$\forall h \in \mathcal{H};\ \nu(h)=1 \Rightarrow \nu'(h)=1,\ \forall h \in \mathcal{H};\ \nu(h)=0 \Rightarrow \nu'(h)=0,$$

et $\exists I'$, interprétation à partir de ν', I' soit soutenable. L'ensemble des valuations étendues de ν est caractérisé par

$$EVal(\nu)=\{\nu' \in Comp(\nu);\ \exists I \in IntInd(\nu'),\ I(\bot)=0\}.$$

Une *valuation étendue minimale* d'une valuation ν est une valuation étendue ν' de ν telle qu'il n'y ait pas d'autres valuations étendues de ν dont ν' soit une valuation étendue. Une valuation étendue minimale de ν est appelée une extension de ν. L'ensemble des extensions de ν est caractérisé par

$$Ext(\nu)=\{\nu' \in EVal(\nu);\ \neg(\exists \nu'' \in EVal(\nu);\ \nu' \in EVal(\nu''))\}$$

Théorème 3: Si ν est consistante, alors l'ensemble des valuations étendues minimales de ν est restreint à $\{\nu\}$.

Les étiquettes ne doivent contenir que des environnements représentant des valuations consistantes. Cette conception entraîne comme au sein de l'ATMS l'inconsistance pour tous les environnements plus complets qu'un environnement inconsistant précis et bien sûr uniquement pour ceux-là. Ainsi, tous les environnements plus généraux qu'un environnement plus spécifique de l'étiquette de $\bot$ devront être atomisés en plusieurs environnements représentant des valuations consistantes.

Le principe retenu est donc que, dans une étiquette, tout environnement représentant une valuation inconsistante doit être remplacé par les environnements représentant ses valuations étendues minimales.

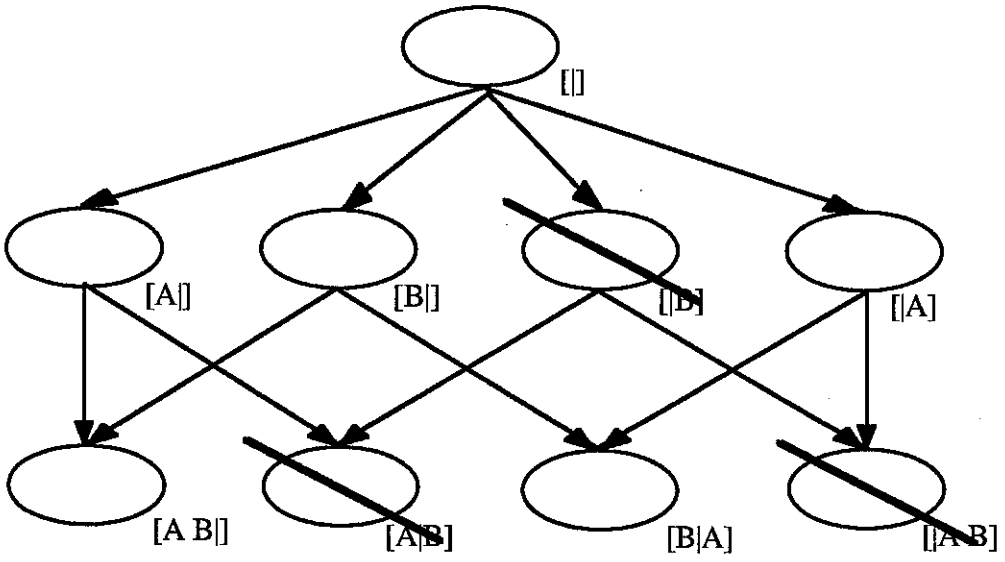

figure 5. Cette figure présente le graphe de la figure 3 où un environnement est inconsistant (celui où B est invalide). Trois des environnements représentés sont donc inconsistants (ceux impliquant l'invalidité de B). Et seuls trois environnements peuvent apparaître au sein des étiquettes: [B|], [B|A] et [B,A|] (ceux dont aucune complétion n'est inconsistante).

À une valuation peuvent correspondre plusieurs valuations étendues et plusieurs valuations étendues minimales.

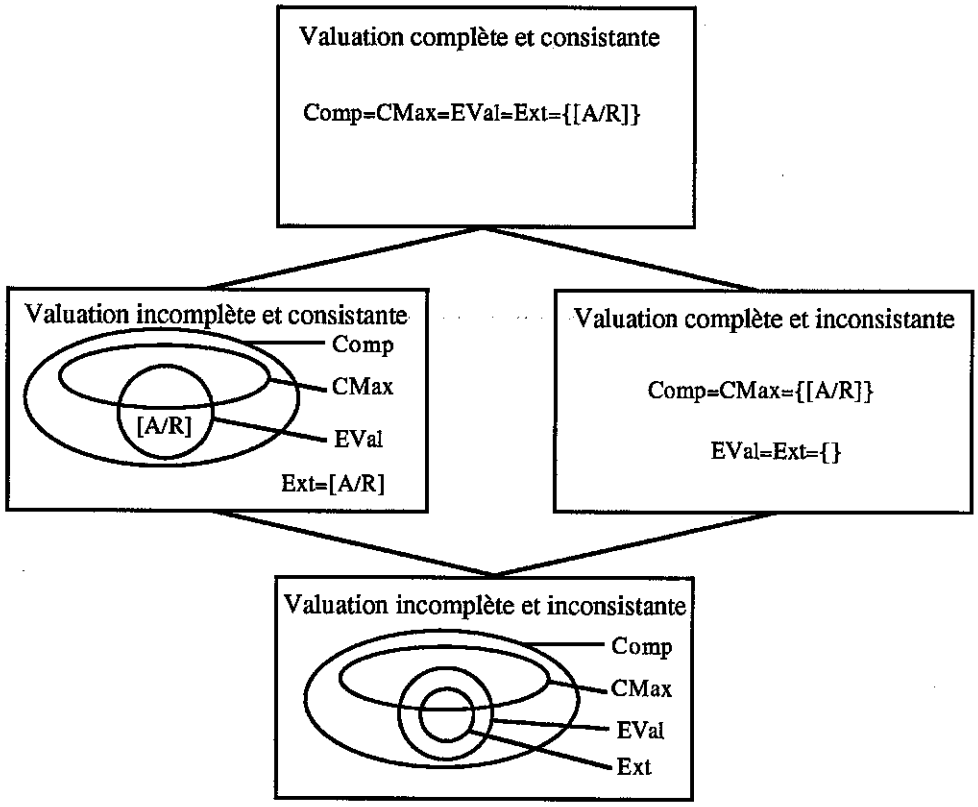

figure 6. Suivant la nature de la valuation [A/R] considérée, les rapports entre ses différents ensembles caractéristiques sont modifiés.

6.3.Les étiquettes

Une étiquette d'un nœud N représente un ensemble de valuations ν qui permettent de construire une interprétation I soutenable et considérant N comme valide. L'étiquette est donc construite sur la définition suivante:

$$\text{ETIQUETTE}[n] = \{C\ ;\ \forall \nu \in \text{Comp}(C), \exists I \in \text{IntInd}(\nu);\ I(n)=1\ \}.$$

En fait, la définition utilisée permet d'avoir une définition «minimale» de l'étiquette:

$$C \in \text{ETIQUETTE}[n] \Leftrightarrow \forall \nu \in \text{Comp}(C), \exists I \in \text{IntInd}(\nu);\ I(n)=1$$

Si un environnement E est dans l'étiquette du nœud N, cela signifie que N est valide dans une interprétation compatible induite par la valuation correspondant à E. En d'autres termes la complétude des étiquettes des nœuds est réalisée si

$$\forall \nu, \forall n \in \mathcal{N}, \exists I \in \text{IntInd}(\nu);\ I(n)=1 \Rightarrow \exists C \in \text{ETIQUETTE}[n];\ \nu \in \text{Comp}(C).$$

La correction quant à elle s'exprime par

$$\forall \nu, \forall n \in \mathcal{N}, \exists C \in \text{ETIQUETTE}[n] \Rightarrow \forall \nu \in \text{Comp}(C), \exists I \in \text{IntInd}(\nu);\ I(n)=1$$

Le travail du CP-TMS sera donc d'assurer à travers tout le graphe de dépendances la complétude, la correction et la consistance des étiquettes des nœuds au regard des définitions données ci-dessus.

7.Propagation des étiquettes

La phase de propagation permet de s'assurer de la correction, de la complétude, de la minimalité et d'une partie de la consistance des étiquettes contenues dans le graphe, alors que la phase de rétablissement de la consistance se charge de garantir une seconde partie de la consistance.

7.1.Algorithme

Les principes des algorithmes utilisés sont les mêmes que ceux de l'algorithme de Jon Doyle, à savoir une phase de propagation activée à chaque fois qu'une nouvelle justification est fournie suivie d'une phase de rétablissement de la consistance activée quand l'étiquette d'un nœud inconsistant est modifiée.

Le principe de la propagation des étiquettes est simple: Pour chaque justification, l'étiquette est la conjonction des étiquettes des nœuds de sa IN-liste avec la conjonction des négations des étiquettes des nœuds de sa OUT-liste. Pour chaque nœud, l'étiquette est calculée en faisant la disjonction de ses justifications.

Si ce principe est simple, il n'est pas directement applicable pour trois raisons: des circularités peuvent être présentes dans le graphe de dépendances qui feraient boucler le programme, les contextes ainsi obtenus ne seraient ni minimaux ni consistants et enfin l'exécution prendrait un temps considérable. L'algorithme présenté ici tente de remédier à ces trois problèmes:

- Les cycles pairs dans le graphe sont correctement traités alors que l'algorithme présenté ici ne supporte pas les cycles impairs.
- Les étiquettes obtenues sont vérifiées comme consistantes et dans un certain sens minimales. Les algorithmes permettant cela sont présentés plus loin (voir §8 et 9), mais les points d'entrée de ceux-ci sont décrits ici.
- Les étiquettes des justifications et des nœuds ne sont recalculées que si cela est nécessaire.

L'algorithme mémorise les étiquettes de chaque nœud de façon à modifier le minimum à chaque justification ajoutée. Les caractéristiques de l'algorithme sont les suivantes:

- La stratégie du support est abandonnée. Cette stratégie permettait de ne recalculer l'étiquette d'un nœud que si un antécédent précis responsable de l'état actuel du nœud voit son état modifié. Les états d'un nœud n'existant plus, il n'est plus question de maintenir une telle stratégie et l'étiquette du nœud doit être constamment remise à jour dès que l'étiquette de l'un de ses antécédents change. La propagation est toujours incrémentale mais suite à l'abandon de la stratégie du support elle concerne cependant tous les nœuds conséquents. Cette modification de l'incrémentalité permet de résoudre le problème (3) de ART.
- La décomposition du graphe de dépendances en composantes fortement connexes utilisée par James Goodwin [Goodwin 87] est conservée. Elle permet de ne remettre à jour les nœuds d'une composante que si tous les nœuds des composantes précédentes ont déjà été examinés.
- La rétrogression ne se fait plus en changeant l'état d'un nœud mais en rétablissant la consistance des étiquettes au sein de tout le graphe.

La façon de déterminer l'étiquette d'une justification est la suivante:

```
JUST-ETIQ( just ) =
∧etiq( {E; ∃N ∈ INLISTE[just]; E = ETIQUETTE[N] }
       ∪ {E; ∃N ∈ OUTLISTE[just]; E = ¬etiq( ETIQUETTE[N] ) } )
```

Pour une justification `just` définie comme

```
INLISTE[just] == {a1,… an}
OUTLISTE[just] == {a'1,… a'n'}
CONS[just] == A
```

le résultat obtenu est

`ETIQUETTE[a1]∧… ETIQUETTE[an]∧ ¬ETIQUETTE[a'1]∧…¬ETIQUETTE[a'n']`

qui correspond bien à la signification de la justification, sachant que pour tout `a` intervenant dans la justification

`ETIQUETTE[a] ⇒ a.`

L'algorithme `CALCULER-JUST-ETIQ` invoqué par le programme permet de ne recalculer cette étiquette qu'en cas de besoin, de la minimiser, de la rendre consistante et enfin de la mémoriser au sein de la structure de justification.

```
CALCULER-JUST-ETIQ( just ) =
[Si ETIQ[just] == {}
  Alors ETIQ[just] <- JUST-CONSISTANCE( MINIMISER( JUST-ETIQ( just ) ) )
<= ETIQ[just]]
```

Le découpage du graphe de dépendances en composantes fortement connexes utilisé dans l'algorithme de James Goodwin a été repris ici. Cependant, une constatation s'impose, les critères de validité n'étant plus importants, le graphe doit être découpé suivant les critères plus stricts de précédence dans tout le graphe et non plus seulement dans le graphe induit par les supports des nœuds (voir §5.2). En fait, il est maintenant aisé de déterminer la composante fortement connexe d'un nœud N d'après ses attributs puisque la définition en est:

$$Cfc(N) = CSQ^*[N] \cap ANT^*[N]$$

Ce découpage n'est pas utile pour la correction et l'arrêt du programme dans l'implémentation actuelle mais peut être utilisé pour optimiser la propagation. La partition du graphe en composantes fortement connexes permet d'examiner celui-ci composante après composante sachant qu'une composante ne sera abordée que si celles qui lui sont antérieures ont été examinées. Ainsi, au moment d'aborder une composante, toutes les justifications dont l'étiquette doit être recalculée ont leur attribut ETIQ vidé. L'algorithme utilisé pour obtenir les composantes est celui décrit dans [Aho& 74]. Il permet d'obtenir l'ensemble des composantes dans l'ordre de précédence qui sera utilisé pour les examiner. Cet algorithme a été modifié pour rendre l'ensemble des justifications ayant des nœuds extérieurs à la composante comme antécédents et aboutissant à celle-ci. L'algorithme de propagation au sein d'une composante est donc le suivant:

```
PROPAGER-ETIQUETTE-DANS-CFC( cfc justl ) =
PourTout just∈justl
Faire Si ETIQ[just] == {}
  Alors PROPAGER-ETIQ(CONS[just] cfc)
```

L'algorithme parcourt alors tous les nœuds de la composante, en partant des justifications à recalculer et en suivant les dépendances, recalculant pour chaque nœud ses justifications si cela est nécessaire. Il n'y a pas de danger de bouclage en cas de cycle pair car avant de modifier l'étiquette d'un nœud il est vérifié si la nouvelle étiquette n'est pas la même que l'étiquette précédemment stockée. Si elle est la même, il n'est pas besoin de propager plus loin car cette propagation de l'étiquette a déjà été faite. C'est le même principe qui est appliqué aux composantes lorsque les étiquettes des justifications incidentes n'ont pas été modifiées.

L'algorithme de propagation au sein d'un nœud est le suivant:

```
PROPAGER-ETIQ( noeud cfc ) =
[ NouvEtiq <-
     NOEUD-CONSISTANCE(
       MINIMISER(
         ∨etiq({ CALCULER-ETIQ( just ); just∈L-JUST[noeud] }))
       noeud)
  Si ~( NouvEtiq == ETIQUETTE[noeud] )
  Alors
     [ ETIQUETTE[noeud] <- NouvEtiq
       Si INCONSISTANT[noeud] == Vrai
       Alors ETIQ-CONSISTANCE( NouvEtiq )
       Sinon PourTout just∈INJUST[noeud]∪OUTJUST[noeud]
             Faire [  ETIQ[just] <- {}
                      Si CONS[just]∈cfc
                      Alors PROPAGER-ETIQ( CONS[just] ) ]]]
```

Ce qui correspond bien au calcul de l'étiquette du nœud car, sachant que

$$\forall j \in LISTDEJUST[n],\ ETIQUETTE[j] \Rightarrow n$$

et que

$$LISTEDEJUST[n]=\{j1,\ldots jm\},$$

alors

$$ETIQUETTE[j1] \vee \ldots ETIQUETTE[jm] \Rightarrow n.$$

L'idée de la propagation est de compléter toutes les étiquettes des nœuds. Cet algorithme garantit que toutes les composantes seront abordées — sauf si cela n'apportait aucune modification — et que tous les nœuds le seront aussi. La décomposition en composantes fortement connexes apporte un gain de rapidité en établissant les étiquettes de toutes les composantes antécédentes avant d'aborder une nouvelle composante. Il n'est cependant pas un modèle de rapidité et certaines optimisations peuvent être envisagées (voir §11).

Par ailleurs, l'algorithme peut boucler en cas de cycle impair au sein du graphe comme le montre l'exemple 5.

Exemple 5:

Si $\mathcal{N}$={A,B} et que deux justifications j1=A->B et j2=¬B->A sont ajoutés, les étiquettes des deux nœuds restent inchangées. Si ensuite le nœud B est posé comme hypothèse (j3=B->B) la propagation se fait ainsi:

B:{[B[]} j1:{[[B]} A:{[[B]} j2:{[[B]}
B:{[B[],[[B]]={[[]} j1:{} A:{} j2:{}
B:{[B[]}...

7.2.Complétude et correction de l'algorithme

Deux des propriétés importantes des étiquettes de l'algorithme de Johan De Kleer sont la complétude — tout environnement permettant de dériver le nœud est au moins un environnement plus spécifique qu'un des environnements qui composent l'étiquette de ce nœud — et la correction — tout environnement plus spécifique (au sens de plus complet) qu'un environnement de l'étiquette d'un nœud permet effectivement de dériver ce nœud. Ces propriétés sont aussi celles des algorithmes utilisés ici pour calculer les étiquettes lors de la propagation.

Les algorithmes développés pour opérer la propagation sont donc des algorithmes capables d'opérer des négations, disjonctions et conjonctions d'étiquettes. Ils sont simplement dérivés des algorithmes permettant de transformer une formule quelconque en forme normale disjonctive avec certaines optimisations liées au fait que la forme de la formule initiale est connue a priori (l'algorithme n'itère donc pas jusqu'à obtenir la forme voulue, les opérations à effectuer sont connues à l'avance). Ces algorithmes sont décrits sous une forme ensembliste[1], puis la correction et la complétude de ces expressions sont montrées à l'aide de l'équivalence des formules logiques représentant les arguments et le résultat de la fonction. L'algorithme utilisé correspondant à des transformations valides de formules logiques, la correction et la complétude des étiquettes des nœuds est alors garantie.

```
∧etiq( etiquette1 etiquette2 ) =
{ ∧env( env1 env2 ) ; env1∈etiquette1 & env2∈etiquette2 }
```

Cet algorithme est complet et correct car:

etiquette1∧etiquette2
== (env11∨...env1n1)∧(env21∨...env2n2)
== (env11∧env21)∨...(env11∧env2n2)∨...(env1n1∧env21)∨...(env1n1∧env2n2)
== env1∨...envn [car ∧env: env×env⟶env] : étiquette

```
∨etiq( etiquette1 etiquette2 ) =
{ env ; env∈etiquette1 ∨ env∈etiquette2 }
```

etiquette1∨etiquette2
== (env11∨...env1n1)∨(env21∨...env2n2)
== env11∨...env1n1∨ env21∨...env2n2 : étiquette

```
¬etiq( etiquette ) =
∧etiq( { ¬env( env ) ; env∈etiquette } )
```

¬etiquette
== ¬(env1∨...envn)
== ¬env1∧...¬envn
== etiquette1∧...etiquetten [car ¬env: env⟶etiquette] : étiquette

[1] «Env» dans la description des algorithmes dénote un environnement global.

```
∧env( env1 env2 ) =
{ ax( env1 )∧ax( env2 )∧rt1∧rt2 ; rt1∈rt( env1 ) & rt2∈rt( env2 ) }
```
env1∧env2
$= ax1 \wedge (rt11 \vee \ldots rt1n_1) \wedge ax2 \wedge (rt21 \vee \ldots rt2n_2)$
$= ax1 \wedge ax2 \wedge ((rt11 \wedge rt21) \vee \ldots (rt11 \wedge rt2n_2) \vee \ldots (rt1n_1 \wedge rt21) \vee \ldots (rt1n_1 \wedge rt2n_2))$: étiquette

```
∨env( env1 env2 ) =
{ env1 env2 }
```
env1∨env2 : étiquette

```
¬env( env ) =
{ ¬a ; a∈ax( env ) }∪{ R; ∀rt∈rt( env ) ; ∃r∈rt; r∈R}
```
$\neg(a1 \wedge \ldots an \wedge ((\neg r11 \wedge \ldots \neg r1n_1) \vee \ldots (\neg rm1 \wedge \ldots \neg rmn_m)))$
$= \neg a1 \vee \ldots \neg an \vee \neg((\neg r11 \wedge \ldots \neg r1n_1) \vee \ldots (\neg rm1 \wedge \ldots \neg rmn_m))$
$= \neg a1 \vee \ldots \neg an \vee (\neg(\neg r11 \wedge \ldots \neg r1n_1) \wedge \ldots \neg(\neg rm1 \wedge \ldots \neg rmn_m))$
$= \neg a1 \vee \ldots \neg an \vee ((r11 \vee \ldots r1n_1) \wedge \ldots (rm1 \vee \ldots rmn_m))$
$= \neg a1 \vee \ldots \neg an \vee ((r11 \vee \ldots r1n_1) \wedge \ldots (rm1 \vee \ldots rmn_m))$
$= \neg a1 \vee \ldots \neg an \vee (r11 \wedge r21 \wedge \ldots rm1) \vee \ldots (r11 \wedge r21 \wedge \ldots rmn_m)$
$\vee \ldots (r1n_1 \wedge r21 \wedge \ldots rm1) \vee \ldots (r1n_1 \wedge r2n_2 \wedge \ldots rmn_m)$

8.Traitements de l'inconsistance

L'étiquette d'un nœud ne doit pas contenir d'environnement inconsistant — c'est-à-dire d'environnement comparable à un environnement d'un nœud inconsistant — et le système garantit que de tels environnements ne peuvent apparaître au cours de la phase de propagation. Pour cela, les diverses causes d'inconsistances sont isolées qui donnent lieu à divers algorithmes permettant d'y remédier.

8.1.Propagation de la consistance interne

La consistance interne est ainsi nommée car elle ne dépend pas des contraintes présentes dans le système. Une telle consistance se borne à cerner des étiquettes bien formées sans considérer si ces étiquettes contiennent des environnements caractérisant un environnement inconsistant. Plutôt que les configurations inconsistantes ce sont les configurations impossibles — ou paradoxales — qui sont éliminées.

La première cause d'inconsistance interne dans un graphe est le cycle impair, et en particulier quand une hypothèse fait partie d'un cycle impair. Si c'est le cas, il existe dans l'étiquette de ce nœud un environnement contenant le nœud lui-même dans sa restriction: il faut éliminer cet environnement. Un exemple d'environnement à éliminer est le suivant:

ETIQUETTE[a]={..., [...|...a...], ...}.

Pour cela, toutes les restrictions contenant ce nœud sont ôtées de leur environnement global, si cela conduit un environnement global à ne plus contenir aucune restriction, alors il est ôté de l'étiquette:

```
CI-CONSISTANCE( etiquette noeud ) =
[ PourTout env∈etiquette
  Faire PourTout rt∈rt( env )
    Faire Si noeud∈rt
      Alors etiquette <- etiquette\env
  <= etiquette ]
```

Par ailleurs, il faut aussi vérifier que le système ne propage pas d'environnements insensés ("meaningless"), il s'agit d'environnements comprenant dans leur restriction un nœud de leurs axiomes. C'est le cas par exemple de l'environnement

[...a...|...a...].

Pour rétablir la consistance interne, toutes les restrictions créant une inconsistance interne sont ôtées de leur environnement. L'algorithme ainsi conçu intervient avant d'établir les étiquettes des nœuds pour la première et après la conjonction d'étiquette pour la seconde. Ceux-ci sont également supprimés:

```
{}-CONSISTANCE( etiquette ) =
[ PourTout env∈etiquette
  Faire Si ax( env )∩rt( env ) ≠ {}
    Alors etiquette <- etiquette\env
  <= etiquette ]
```

Ces deux causes d'inconsistances n'apparaissaient pas dans le système de Johan De Kleer, elles sont toutes deux conséquences de la présence de restrictions dans les environnements.

8.2.Découverte d'une inconsistance

L'inconsistance externe est l'inconsistance ne dépendant pas des nœuds auxquels sont attachées les étiquettes, mais de l'étiquette de ⊥. Le traitement de l'inconsistance externe se fait de deux façons: à la propagation en interdisant d'ajouter des environnements inconsistants (voir §8.4) et en fin de propagation, quand ⊥ a été dérivé, en supprimant du graphe tous les environnements inconsistants.

Dans le second cas, c'est-à-dire lors de la découverte de nouveaux environnements dans l'étiquette du nœud ⊥, le réseau n'est possiblement plus consistant. Il faut alors rétablir la consistance externe en supprimant ces nouveaux environnements de toutes les étiquettes des nœuds où ils se trouvent.

Pour cela plusieurs approches peuvent être imaginées suivant l'état de consistance exigé du graphe de dépendances.

8.2.1. Approche minimale

L'approche minimale s'apparente totalement au travail effectué dans le cadre de l'ATMS: de toutes les étiquettes sont supprimés les environnements plus généraux qu'un des environnements de l'étiquette de ⊥ — sans poser l'hypothèse du monde clos. Tout d'abord, il faut trouver, pour tous les environnements E de l'étiquette, tous les nœuds qui ont dans leur étiquette un environnement contenant tous les axiomes de E. Cela constitue un sous-ensemble de l'intersection de tous les CSQ* des axiomes de E. L'algorithme est alors le suivant:

```
ETIQ-CONSISTANCE( incetiq ) =
PourTout incdisj∈incetiq
Faire PourTout noeud∈∩N∈ax( incdisj )∪rt( incdisj )CSQ*[N]
       Faire PourTout disj∈ETIQUETTE[noeud]
              Faire Si ax( incdisj )⊆ax( disj )
                       & rt( incdisj )⊆rt( disj )
                     Alors ETIQUETTE[noeud] <- ETIQUETTE[noeud]\{disj}
```

Cet algorithme, le plus simple, considère chaque environnement comme incomplet et permet — faute de complétude — de raisonner momentanément au sein d'un environnement inconsistant.

8.2.2. Approche étendue

Il est cependant possible d'étendre la portée de cet algorithme de façon à disposer de plus de connaissance dans chaque extension. Ainsi, si l'étiquette {[a,b|c,d]} est inconsistante et que dans une autre étiquette se trouve l'environnement [a,b,e|d], cet environnement sera avantageusement remplacé par [a,b,e,c|d], il est en effet hors de question que le nœud soit valide dans un environnement contenant a, b et ne contenant ni c, ni d.

L'implémentation de cette deuxième approche consiste à trouver les nœuds dont l'ensemble d'axiomes d'un des environnements de leur étiquette contient un des ensembles d'axiomes de l'étiquette de ⊥. Une fois trouvés ces nœuds, les environnements E' de ces nœuds sont remplacés par l'ensemble des environnements constitués de E'∧r où r appartient à la restriction de l'environnement.

```
ETIQ-CONSISTANCE( incetiq ) =
PourTout incenv∈incetiq
Faire PourTout noeud∈∩N∈ax( incenv )CSQ*[N]
       Faire PourTout env∈ETIQUETTE[noeud]
              Faire Si ax( incenv )⊆ax( env )
                     Alors [ ETIQUETTE[noeud] <- ETIQUETTE[noeud]\env
                             PourTout r∈∪rt∈rt( incenv )rt
                             Faire AJ-ETIQ(ax(env)∧r∧rt(env)
                                    ETIQUETTE[noeud])]
```

Cet algorithme fait bien le travail de son prédécesseur, mais pose tout de même un problème: si a,b⇒f et que a,b,¬c⇒⊥, l'étiquette de f contiendra l'environnement [a,b,c|] ce qui signifierait que f n'est dérivable que dans un environnement où a,b et c sont présents; ce n'est pas ce qu'expriment les formules. L'utilisation d'un tel algorithme — ainsi que du suivant — entraîne la modification de la définition de l'étiquette qui ne contient que des environnements *consistants* permettant de dériver la formule.

8.2.3. *Approche canonique*

L'approche canonique va beaucoup plus loin par rapport à la précédente. Elle consiste à exiger que tout environnement dans l'étiquette d'un nœud ait toutes ses complétions valides. Les étiquettes doivent donc être purgées des environnements inconsistants. Le principe est simple: la partie valide d'un environnement E correspond à la formule

$$E \wedge \neg \text{ETIQUETTE}[\bot]$$

L'algorithme naïf est le suivant pour chaque environnement `[ax/rt]` d'une étiquette et pour chaque environnement `[incax/incrt]` de l'étiquette de ⊥.

```
RENDRE-VALIDE( incax ax incrt rt ) =
Si incrt == {}
Alors Si incax == {}
     Alors <= {}
     Sinon Soit a∈incax
            Faire <= RENDRE-VALIDE( incax\{a} ax∪{a} incrt rt)
                     ∪{<ax rt∪{a}>}
Sinon Soit r∈incrt
       Faire <= RENDRE-VALIDE( incax ax incrt\{r} rt∪{r})
                ∪{<ax∪{r} rt>}
```

Cet algorithme correspond aux spécifications suivantes:

$E \wedge \neg \text{ETIQUETTE}[\bot]$

$$\begin{aligned}
&== a1 \wedge \ldots an \wedge \neg b1 \wedge \ldots \neg bm \wedge \neg incdisj1 \wedge \ldots \neg incdisjp \\
&== a1 \wedge \ldots an \wedge \neg b1 \wedge \ldots \neg bm \wedge \neg (a1,1 \wedge \ldots a1,n_1 \wedge \neg b1,1 \wedge \ldots \neg b1,m_1) \\
&\qquad\qquad \wedge \ldots \neg (ap,1 \wedge \ldots ap,n_p \wedge \neg bp,1 \wedge \ldots \neg bp,m_p) \\
&== a1 \wedge \ldots an \wedge \neg b1 \wedge \ldots \neg bm \wedge (\neg a1,1 \vee \ldots \neg a1,n_1 \vee b1,1 \vee \ldots b1,m_1) \\
&\qquad\qquad \wedge \ldots (\neg ap,1 \vee \ldots \neg ap,n_p \vee bp,1 \vee \ldots bp,m_p) \\
&== (a1 \wedge \ldots an \wedge \neg b1 \wedge \ldots \neg bm \wedge \neg a1,1 \wedge \ldots \neg ap,1) \vee \ldots \\
&\quad (a1 \wedge \ldots an \wedge \neg b1 \wedge \ldots \neg bm \wedge \neg a1,1 \wedge \ldots bp,m_p) \vee \ldots \\
&\quad (a1 \wedge \ldots an \wedge \neg b1 \wedge \ldots \neg bm \wedge b1,m_1 \wedge \ldots \neg ap,1) \vee \ldots \\
&\quad (a1 \wedge \ldots an \wedge \neg b1 \wedge \ldots \neg bm \wedge b1,m_1 \wedge \ldots bp,m_p) \\
&== \{[ax(E) \cup A \,/\, rt(E) \cup R];\ \forall E' \in incetiq\ (\exists r \in rt(E');\ r \in ax(E) \cup A \text{ ou } \exists a \in ax(E');\ a \in rt(E) \cup A)\}
\end{aligned}$$

Cependant, l'algorithme est légèrement optimisé par la réinjection des formules utilisées dans la suite de l'environnement à traiter afin de ne pas obtenir trop d'environnements complétions les uns des autres. En effet, cette réinjection permet d'obtenir des environnements qui ne sont que les complémentaires des environnements déjà obtenus par rapport aux nouveaux environnements.

Par ailleurs, les environnements ajoutés doivent être vérifiés comme consistants (de manière interne cette fois-ci). Ceci peut d'ailleurs aisément être vérifié en s'assurant que $ax(incenv) \cap rt(env) \cup ax(env) \cap rt(incenv) = \{\}$ avant d'utiliser `RENDRE-VALIDE`.

La procédure rétablissant la consistance après modification de l'étiquette de ⊥ est donc la suivante:

```
ETIQ-CONSISTANCE( incetiq ) =
PourTout incdisj∈incetiq
Faire PourTout noeud
     Faire PourTout ETIQUETTE[noeud]
       ETIQUETTE[noeud]
         <- vetiq({RENDRE-VALIDE(ax(incdisj)
                                 ax(disj)
                                 rt(incdisj)
                                 rt(disj)); disj∈ETIQUETTE[noeud]})
```

En résumé, les trois premières approches regardent l'étiquette {[a,b|c,d]}:⊥ sous des jours différents:

- La première approche considère qu'un environnement [A/R] est inconsistant — et le supprime donc — uniquement si {a,b}⊆A et {c,d}⊆R. Cette approche permet de garantir qu'aucun environnement n'est présent dans une étiquette si tous les environnements plus complets que lui sont inconsistants. Il supprime donc tous les environnements plus complets qu'un environnement inconsistant.
- Dans la deuxième approche, l'étiquette {[a,b|]} est considérée comme inconsistante sous l'hypothèse du monde clos qui est utilisée pour répondre aux requêtes. L'étiquette {[a,b|]} est donc restreinte à l'étiquette maximale consistante plus spécifique: {[a,b,c|],[a,b,d|]}. L'étiquette {[a|]} ne subit par contre aucune modification. En fait, cette façon de faire est plutôt contre l'esprit de l'ATMS qui se contente d'éliminer les environnements inconsistants sans se soucier de transformer ceux-ci en environnements consistants. Cette approche garantit qu'aucun environnement n'est présent dans la base si sa complétion canonique est inconsistante. L'inconsistance n'est donc rétablie que vis-à-vis des axiomes.
- Dans le troisième cas, l'étiquette {[a,b|]} n'est pas inconsistante car il existe des extensions de cette étiquette — nommément {[a,b,c|]},... qui ne sont pas inconsistantes. L'étiquette {[a,b|]} subit le même traitement que dans le cas précédent alors que l'étiquette {[a|]} est transformée en {[a|b],[a,c|],[a,d|]}. Cette approche résout intelligemment un des problèmes des systèmes alliant contextes et non monotonie qui est qu'un contexte peut être inconsistant alors qu'aucun de ses fils — contextes moins généraux — ne l'est (voir l'exemple 3). L'idée de cette approche est de remplacer tout environnement ayant des complétions inconsistantes par ses extensions — c'est-à-dire ses complétions minimales (valuations étendues minimales).

Dans l'implémentation actuelle du CP-TMS, c'est la troisième solution qui a été implémentée. C'est en effet la seule qui garantisse qu'une formule soit valide dans toutes les complétions de son étiquette, tout en garantissant la consistance de ces complétions.

Certains systèmes ont adopté une attitude plus radicale — et à notre avis incorrecte — vis-à-vis des environnements inconsistants: ils invalident tous les environnements plus spécifiques de ces derniers (voir l'exemple 3).

8.3.Traiter l'inconsistance par la rétrogression

Le CP-TMS ne traite pas mieux ces interprétations que le TMS classique. En effet, la présence de cycles pairs dans le graphe de dépendances peut occasionner diverses interprétations construites à partir d'une valuation — ceci seulement si aucun nœud du cycle n'est déclaré comme hypothèse. Dans un tel cas, les réponses aux requêtes fondées sur l'existence d'une interprétation peuvent sembler quelque peu insuffisantes. Ce problème conduit à examiner diverses réponses:

- La réponse immédiate consiste simplement à forcer tous les membres de OUT-listes à être des hypothèses. Ainsi, toutes les extensions seront présentes sous forme d'environnements. Cette solution est, entre autre, adoptée pour toutes les extensions de l'ATMS [Dressler 88, De Kleer 88].
- Il est possible de diriger le système lors de la «rétrogression» puisque c'est là que peuvent apparaître de multiples extensions. À partir des techniques utilisées lors de la rétrogression du TMS, des techniques de rétrogression choisissant les extensions peuvent être envisagées (c'est ce qui est exposé dans cette partie). Une de ces techniques est utilisée dans l'extension de Johan De Kleer [De Kleer 88].

- Il serait intéressant de rendre compte autrement des interprétations de façon à considérer qu'un environnement doit être dans l'étiquette d'un nœud si et seulement si ce dernier est valide dans toutes les interprétations construites à partir de la valuation correspondant à l'environnement. En tout cas, cette option devrait donner beaucoup plus de travail au système.
- Enfin, un dernier terrain d'investigation est la construction et la maintenance explicite d'environnements structurés en un graphe permettant l'héritage des formules inférées. Un tel système s'éloigne énormément de l'ATMS et du TMS, mais devrait permettre de considérer toutes les extensions de manière automatique. Cette dernière solution, qui se rapproche beaucoup du système ART, fera l'objet d'un autre travail.

Il est possible de traiter l'inconsistance externe grâce à une rétrogression tel que cela est réalisé dans le TMS de Jon Doyle. Une telle approche n'est pas très naturelle quand elle est utilisée avec des contextes. Elle peut cependant s'avérer très attrayante.

8.3.1. Rétrogression sur les causes premières

Cette extension vise à opérer une rétrogression similaire à celle produite par le TMS. Elle va, en effet, empêcher qu'un environnement de l'étiquette de $\perp$ puisse être présent dans un autre environnement. Chaque environnement de l'étiquette de $\perp$ contient une restriction, il suffit, pour empêcher cet environnement de devenir valide, d'ajouter une justification imposant la présence d'une des restrictions quand tout le reste de l'environnement est valide.

Concrètement, pour tout nœud dans une restriction de l'étiquette, la justification correspondant à son environnement sans lui lui est ajoutée. Tout ces nœuds sont en effet des hypothèses, ce qui est ajouté est donc le fait que cette hypothèse doit être posée quand suffisamment d'autres hypothèses sont posées pour engendrer une inconsistance. Pour un environnement de la forme [A/R] et pour tout r dans R, la rétrogression ajoute la justification

<A R\{r}>: r

```
ETIQ-CONSISTANCE( incetiq ) =
PourTout incenv∈incetiq
Faire [PourTout rt∈rt( incenv )
       Faire PourTout r∈rt
                 Faire PROPAGER( ax( incenv )∧rt\{r} r )
ETIQUETTE[noeud])]]
```

Il est possible — comme cela est fait dans l'algorithme — de remettre en cause tous les fautifs de façon à pouvoir explorer toutes les extensions du contexte inconsistant, mais il est encore possible comme cela est fait dans le système de Jon Doyle de ne considérer qu'un seul fautif à remettre en cause.

La phase de propagation de l'algorithme s'arrêtera au moins devant la justification de $\perp$ car l'étiquette deviendra alors inconsistante — de manière interne. L'appel récurent de PROPAGER ne fera pas boucler l'algorithme général PROPAGER, même si plusieurs niveaux de propagation peuvent apparaître. En effet, comme c'était le cas pour l'algorithme initial du système de maintenance de la vérité cet algorithme conduit toujours à «moins de possibilités d'inconsistance» et converge donc.

Un tel algorithme risque de mener à l'introduction de cycles impairs dans le graphe. Il devrait alors être possible d'utiliser un algorithme du genre de celui de Charles Petrie [Petrie 87] pour les éviter.

Cet algorithme implique qu'une hypothèse soit posée si toutes les autres sont présentes ou absentes dans le sens de l'environnement. Cela revient à faire de la remise en cause d'hypothèses en cas d'inconsistance comme le fait de TMS mais la définition d'une hypothèse a changée puisqu'une telle hypothèse n'est pas forcement dans la OUT-liste d'une justification — c'est-à-dire dans les antécédents négatifs de la règle qui mène à l'inférence. Réciproquement, toutes les hypothèses au sens de Jon Doyle ne sont pas des hypothèses dans le sens considéré ici.

L'effet de ce maintient de la consistance peut être observé par rapport au diagramme de la figure 2.

Si les hypothèses posées sont G et E, l'étiquette du nœud ⊥ sera (¬G∧¬E). Bien que A et B soient pris dans un cycle pair, ce ne sont pas forcément eux qui sont remis en cause car ce ne sont pas des hypothèses et que leur valeur dépend de l'hypothèse G. L'interprétation du graphe n'est donc pas équivalente avec celle de Doyle. Au cours du rétablissement de la consistance ce sont les nœuds G et E qui voient leurs conditions de validité — leurs étiquettes — restreintes.

8.3.2. Approche «à la TMS»

Pour recréer l'équivalent de la rétrogression Doylienne, il suffirait de déclarer comme hypothèses tous les nœuds présents dans une OUT-liste de justification — c'est-à-dire, pour le graphe de la figure 2, A, B et E. La rétrogression du TMS serait alors obtenue simultanément pour toutes les hypothèses — au lieu d'en choisir une. Dans un tel cas le système interdit l'absence simultanée des deux hypothèses A et B et propose à l'utilisateur les deux extensions correspondantes. Le comportement est bien alors celui du système de Doyle sauf qu'ici le système ne choisit pas l'extension dans laquelle il se place. Il est normal qu'en cas de règles avec antécédents négatifs, le système déclare de tels antécédents comme des hypothèses — bien que chez Doyle ce soient leurs conséquents qui sont des hypothèses, ici ce sont les défauts car ce sont eux qui conditionnent la validité de l'hypothèse. Le conséquent est en fait posé sous l'hypothèse que le défaut ne soit pas dérivable. Cette dernière optique est à rapprocher des travaux d'Oskar Dressler (voir §4.1.2 et 12.3)

Il est possible de se rapprocher encore plus du comportement du système de Jon Doyle en conservant un mécanisme de rétrogression. Ainsi, à la dérivation de ⊥, pour chaque environnement inconsistant le système de rétrogression pourra choisir d'ajouter *un* seul des défauts à l'environnement. Cela permet pour chaque environnement inconsistant de résoudre l'inconsistance en se plaçant dans une extension précise de l'environnement. C'est la réalisation d'un système où le traitement des inférences monotones est indéterministe et celui des inférences non monotone déterministe.

Le grand avantage de cette méthode est qu'il n'est plus nécessaire de propager les environnements consistants, ils le sont en effet tous et cela simplifie grandement l'algorithme de propagation. Il est cependant plus réaliste dans le cadre d'un système de maintenance de la vérité à propagation de contextes de mélanger les deux méthodes de gestion de l'inconsistance. Auquel cas, si la rétrogression est choisie, l'environnement inconsistant ne doit plus se trouver dans l'étiquette de ⊥, car toutes les précautions ont été prises pour que celui-ci ne soit plus dérivé.

Il est cependant des règles qui ne doivent pas être traitées par la rétrogression, ce sont celles qui correspondent aux règles de complétion des bases de données. Elles signifient que si l'utilisateur n'a pas dit que A était vrai ce n'est pas parce qu'il a oublié ou parce qu'il ne sait pas, c'est que A n'est pas vrai et toutes les règles de complétion vont faire des inférences en tirant parti de ce fait. Ces règles servent en fait à ne pas obliger l'utilisateur à ajouter de manière fastidieuse un grand ensemble de faits. Les inférences faites à partir de tels défauts sont donc toujours valides et les défauts ne doivent pas être compris comme des hypothèses du raisonnement ni surtout être examinés lors d'une rétrogression. Il existe donc deux sortes de règles à antécédents négatifs: les inférences par défaut qui posent une hypothèse en cas d'information incomplète et les règles de complétion qui complètent une base de façon sûre et correcte. Les secondes relèvent en fait plus de l'interface utilisateur que des systèmes de raisonnement hypothétique et ne doivent donc pas être maintenues ni signalées au système de maintenance de la vérité. Cela est possible en jouant sur un mécanisme tel que les dépendances logiques (voir §4.2).

Quoiqu'il en soit, ces différentes approches relèvent beaucoup plus du système de raisonnement que du système de maintenance de la vérité. Elles ne seront donc pas développées plus avant.

8.4.Propagation de la consistance externe.

Les environnements doivent posséder la propriété de consistance externe — c'est-à-dire qu'ils ne doivent pas permettre de dériver ⊥. Il est donc inutile de propager des environnements inconsistants et c'est pourquoi l'algorithme de propagation garantit l'utilisation d'environnements consistants.

Pour s'assurer, à la propagation, que toutes les étiquettes présentes dans le système sont consistantes, les fonctions utilisées lors de la propagation sont modifiées de façon très simple: à chaque étiquette calculée, celle-ci est vérifiée comme consistante en s'assurant qu'aucun environnement d'étiquette ne permet de dériver $\bot$. La méthode utilisée consiste a disloquer tous les environnements plus généraux qu'un environnement de l'étiquette de $\bot$ de façon à n'en conserver que la partie disjointe de l'étiquette de $\bot$. Cela peut être fait en utilisant les différents types de consistance d'une étiquette défini au §8.2. L'algorithme présenté ici concerne plus spécialement l'option choisie dans l'implémentation actuelle.

L'algorithme naïf est le suivant:

```
⊥-CONSISTANCE( etiquette ) =
PourTout incdisj∈ETIQUETTE[⊥]
Faire PourTout env∈etiquette
     Faire Si ax( env )∩rt( incdisj )={}
            Alors PourTout rt∈rt( env )
               Faire Si rt∩ax( incdisj )={}
                      Alors etiquette
                             <- etiquette
                                  \env
                                  ∪RENDRE-VALIDE( ax(incdisj)
                                    ax(env) rt(incdisj) rt )
```

D'un point de vue optimisation, les trois fonctions de propagation de la consistance peuvent bien sûr fonctionner conjointement de façon à ne pas parcourir plusieurs fois les étiquettes. Une autre approche possible, c'est d'ajouter la vérification de la consistance lors de la propagation au sein des opérations élémentaires présentées plus haut: $\vee$etiq propage la consistance car l'union d'un certain nombre d'étiquettes consistantes donnera une étiquette consistante. Il n'en va pas de même de $\wedge$etiq et de $\neg$etiq, il faut alors y rajouter de petites fonctions vérifiant la consistance des résultats.

Les fonctions de plus haut niveau peuvent être décrites comme suit:

```
JUST-CONSISTANCE( etiquette ) =
⊥-CONSISTANCE( {}-CONSISTANCE( etiquette ) )

NOEUD-CONSISTANCE( etiquette noeud ) =
CI-CONSISTANCE( etiquette noeud )
```

elles bénéficient des propriétés de la disjonction d'étiquettes qui préservent la $\bot$-consistance et la {}-consistance.

Dans l'implémentation, l'étiquette des nœuds inconsistants est stockée dans la variable `*incetiq*` plus rapidement accessible. L'étiquette physique de ces nœuds, quant à elle, est toujours remise à {} par les algorithmes de rétablissement de la consistance ce qui permet de ne pas faire de cas particuliers au sein de ces algorithmes.

Ceci apporte un autre avantage: le rétablissement de la consistance externe au sein de tout le réseau de dépendances ne se fait plus sur l'ensemble de l'étiquette de $\bot$ (qui est contenue dans `*incetiq*`), mais seulement sur les nouvelles manières de dériver $\bot$.

9.Minimalité

L'algorithme de l'ATMS garantit la minimalité des étiquettes ce qui permet de meilleures performances des algorithmes les manipulant. Le principe retenu par Johan De Kleer est qu'une étiquette est minimale s'il n'existe pas dans cette étiquette deux environnements comparables l'un à l'autre. Ce principe garantit une forme unique des étiquettes équivalentes ayant un nombre minimum d'environnements, ces environnements ayant un nombre minimum d'hypothèses. Le problème est qu'il existe plusieurs formes normales disjonctives pour une formule quelconque comme celles considérées ici. Qui plus est, il existe plusieurs formes normales «minimales» possibles. Le problème de garantir la minimalité des formules en forme normale est très complexe (dur au sens combinatoire). Toutes les vérifications doivent être faites à chaque étape de la propagation.

Le principe retenu a donc été celui dit de minimalité faible: deux environnements d'une étiquette ne peuvent avoir de complétions communes. Cette formulation est équivalente à celle de De Kleer si elle est utilisée avec l'ATMS, mais elle n'a pas les mêmes conséquences sur les étiquettes manipulées par le CP-TMS. Pour obtenir la faible minimalité il ne faut pas que dans une étiquette se trouve un environnement tel qu'un autre environnement soit plus général. Autrement dit,

min(étiquette) = {env∈étiquette; ∀env'∈étiquette env≠env' ⇒ ¬(env'⊇env) }

en utilisant la définition suivante de la relation de généralité — «est plus général» correspond alors à «est moins complet» —:

$$[A1/R1] \sqsupseteq [A2/R2] \Leftrightarrow A2 \subseteq A1 \;\&\; R2 \subseteq R1.$$

Supposant que les étiquettes des nœuds sont faiblement minimales, si la conjonction d'une étiquette et d'une étiquette minimale rend une étiquette minimale, il est possible de conserver cette minimalité. L'algorithme d'union d'étiquettes peut être transformé ainsi:

```
vetiq( etiq1, etiq2 ) =
PourTout env1∈etiq1
Faire [ dumenv <- env1
        PourTout env2∈etiq2
        Faire Si ax( env1 ) == ax( env2 )
                Alors [ PourTout rt1∈rt( env1 )
                        Faire  PourTout rt2∈rt( env2 )
                               Faire [Si rt2⊆rt1
                                      Alors dumenv <- dumenv\{rt1}
                                      Si rt1⊂rt2
                                      Alors env2 <- env2\{rt2} ]
                        rt( env2 ) <- rt( env2 ) ∪ rt( dumenv )
                        dumetiq <- dumetiq\{env1}∪dumenv
                        dumenv <- {} ]
                Sinon Si ax( env1 )⊂ax( env2 )
                        Alors [PourTout rt1∈rt( env1 )
                               Faire  PourTout rt2∈rt( env2 )
                                      Faire Si rt1⊆rt2
                                              Alors rt( env2 ) <- rt( env2
)\{rt2}
                               Si rt( env2 ) == {}
                               Alors etiq2 <- etiq2\{env2} ]
                        Sinon Si ax( env2 )⊂ax( env1 )
                               Alors PourTout rt1∈rt( env1 )
                               Faire  PourTout rt2∈rt( env2 )
                                      Faire Si rt2⊆rt1
                                              Alors rt( dumenv )
                                                      <- rt( dumenv )\{rt1}
        etiq2 <- etiq2∪dumenv
        dumetiq <- dumetiq\{env1}∪{dumenv} ]
<= dumetiq
```

Le problème de cet algorithme par rapport aux algorithmes de Johan De Kleer, plus simples, est qu'il ne s'agit plus seulement de comparer des ensembles de faits mais des ensembles de faits restreints.

Cet algorithme rend des étiquettes faiblement minimales dans le sens où il ne rajoute pas d'environnements qui sont comparables à d'autres environnements. Par contre, il ne fait jamais de fusion d'environnements globaux. Par exemple, l'ajout de l'étiquette {[a,b|]} à l'étiquette {[a|b], [|a]}, toutes deux minimales donnera l'étiquette {[a|], [|a]} au lieu de {[|]} qui peut sembler plus minimale.

Il serait souhaitable d'ajouter au présent algorithme un algorithme maintenant une minimalité plus forte opérant la fusion d'environnements globaux, cet algorithme doit être activé à chaque fois que l'étiquette d'un nœud est déterminée. Il est tentant de poser le principe de minimalité forte: une étiquette est fortement minimale s'il n'existe pas d'étiquette logiquement équivalente comportant un nombre moindre d'environnements. Ce principe ne garantit toujours pas un minimum unique (par exemple {[a,b|],[|a]} ⇔ {[b|],[|a]}).

Une procédure pour calculer ce minimum fort peut être échafaudée:

```
MIN( etiq ) =
PourTout <A, R>∈etiq
Faire Si <A' R'>∈etiq ; R∩A'≠{}
  Alors PourTout f∈R∩A'
    Faire Si A'∪A∩R'∪R\{f}={}
      Alors <= <A∩A'\{f}, R∩R'\{f}>
  Sinon <= <A, R>
```

Cependant une telle procédure doit agir en une succession de passes (cela peut se vérifier sur l'exemple précédent) comme un opérateur de point fixe. Malheureusement, il n'est pas certain qu'un tel point fixe existe toujours et dès lors que l'algorithme s'arrête.

10.Traitement des requêtes

Une requête est une question de la forme «la formule f est-elle valide dans le contexte [A/R]?» (ce qui sera noté [A/R]?f). Avant de pouvoir répondre à une telle requête, il faut savoir ce que signifient les étiquettes des nœuds, les contextes et les requêtes.

10.1.Les contextes d'interrogation

Le contexte fourni lors d'une requête a la forme [A/R]. Il n'y pas a priori de raison pour que ce contexte détermine une valuation complète. Le problème est de savoir à quelle interprétation renvoie ce contexte d'interrogation, car un tel contexte détermine plusieurs interprétations. Indépendamment de l'interprétation choisie, il est possible de mettre en évidence différents ensembles correspondant à un contexte.

L'ensemble des formules dérivées de [A/R] est l'ensemble des nœuds qui sont toujours valides sous les hypothèses [A/R], c'est-à-dire les nœuds dont [A/R] est une complétion d'un environnement de leur étiquette:

$$Ctxt(\diamond) = \cup_{C;\diamond\in Comp(C)} \{n\in\mathcal{N};\ C\in ETIQUETTE[n]\ \}$$

Cette définition rejoint celle de Johan De Kleer [De Kleer 86a]. Il faut remarquer que cette définition n'exige pas que les restrictions de $\diamond$ soient exclues de Ctxt($\diamond$).

L'ensemble des formules nécessaires dans <A {R}> est l'ensemble des nœuds qui doivent être valides dans toutes les interprétations construites à partir de <A {R}>:

$$Nec(\diamond) = Ctxt(\diamond)\cup\cap_{C\in Comp(\diamond)\setminus\{\diamond\}} Nec(C)$$

Conséquence: Si $x\in Ctxt(\diamond)$ alors $x\in Nec(\diamond)$

Cette définition qui correspond à une équation de point fixe peut être rendue plus opérationnelle grâce au théorème suivant.

Théorème 4: $Nec(\diamond) = \cap_{C\in Cmax(\diamond)} Ctxt(C)$

Il ressort de ce théorème que l'ensemble des formules nécessaires dans un environnement correspond à l'ensemble des formules valides dans le contexte de tous les environnements qui sont plus complets que lui.

Théorème 5: $\forall n\in\mathcal{N}$, $(n\in Nec(\diamond) \Rightarrow \forall \diamond'\in Cmax(\diamond), \exists I\in IntInd(\diamond'), I(n)=1)$

Ce théorème permet d'assurer une certaine complétude entre le concept de nécessité et l'interprétation des environnements telle qu'elle a été décrite plus haut.

L'ensemble des nœuds possibles dans $\diamond$ est l'ensemble des nœuds qui sont valides dans une interprétation construite à partir de $\diamond$:

$$Pos(\diamond) = \cup_{C\in Comp(\diamond)} Ctxt(C)$$

Théorème 6: $Pos(\diamond) = \cup_{C\in Cmax(\diamond)} Ctxt(C)$

Encore une fois, ce théorème permet de faire correspondre la notion de possibilité à l'intuition vis-à-vis des contextes complets:

Théorème 7: $\forall n\in\mathcal{N}$, $(n\in Pos(\diamond) \Rightarrow \exists \diamond'\in Cmax(\diamond); \exists I\in IntInd(\diamond'), I(n)=1)$

Il assure, ici aussi, la complétude entre interprétations et environnements.

Théorème 8: $Ctxt(\diamond) \subseteq Nec(\diamond) \subseteq Pos(\diamond)$

Ce dernier théorème met en évidence la hiérarchie entre les ensembles définis ci-dessus et permet donc de se faire une idée de la précision et de la portée des résultats qu'ils permettent d'obtenir.

Il est possible que certaines complétions du contexte [A/R] soient inconsistantes. Il est alors commun de se rabattre sur les extensions d'un tel contexte, c'est-à-dire les complétions minimales de [A/R] dont toutes les complétions sont consistantes — c'est-à-dire encore les valuations étendues minimales de la valuation correspondant à [A/R].

10.2.Signification possible des requêtes

Le problème est de savoir interpréter une requête, la forme générale d'une requête doit être [A/R]?f. Il est clair que ces requêtes sont faites en fonction du monde réel mais le contexte d'interrogation est incomplet par rapport à celui-ci. Plusieurs attitudes peuvent être adoptées vis-à-vis de cette incomplétude:

1- La formule doit être un théorème dans toutes les complétions du contexte d'interrogation.
2- La formule doit être un théorème dans la complétion canonique du contexte d'interrogation (hypothèse du monde clos appliquée au contexte d'interrogation).
3- La formule doit être un théorème dans une complétion quelconque du contexte d'interrogation.

Des complications peuvent être introduites si non content d'être incomplet, le contexte d'interrogation est inconsistant (au sens de «une complétion du contexte d'interrogation est inconsistante»). Il est alors possible de:

1- Simplement signaler à l'utilisateur cette inconsistance en lui demandant de compléter son contexte de façon non contradictoire.
2- Calculer les extensions de son contexte d'interrogation pour lui donner une réponse. Deux nouvelles possibilités s'offrent alors:
   1- La formule doit être un théorème dans toutes les extensions du contexte d'interrogation.
   2- La formule doit être un théorème dans une extension du contexte d'interrogation.

   Et ici encore, les trois premiers choix sont possibles pour déterminer ce que signifie être un théorème dans une extension qui est un contexte d'interrogation comme un autre (mais forcement consistant).

Certaines significations sont exposées ci-dessous dans un ordre qui va de la plus difficile à la plus facile à satisfaire:

a) F est-elle un théorème dans toutes les complétions de la valuation correspondant à [A/R]?
b) F est-elle un théorème dans toutes les extensions (au sens de toutes les complétions de toutes les extensions) de la valuation correspondant à [A/R]?
c) F est-elle un théorème dans certaines complétions de toutes les extensions de la valuation correspondant à [A/R]?
d) F est-elle un théorème monotoniquement sous l'hypothèse du monde clos et dans un contexte contenant [A/R] — ce qui est noté $[A/R] \vdash_{-} f$?
e) F est-elle un théorème dans la valuation complétée canoniquement à partir de la valuation correspondant à [A/R]? Il est à noter que cette valuation peut elle-même être inconsistante.
f) F est-elle un théorème dans une extension (au sens de toutes les complétions d'une extension) de la valuation correspondant à [A/R]?
g) F est-elle un théorème dans certaines complétions d'une extension de la valuation correspondant à [A/R]?
h) F est-elle un théorème dans une complétion de la valuation correspondant à [A/R]?

Les interprétations (d) et (e) étant peu aisées à vérifier elles ne seront pas considéré systématiquement ici.

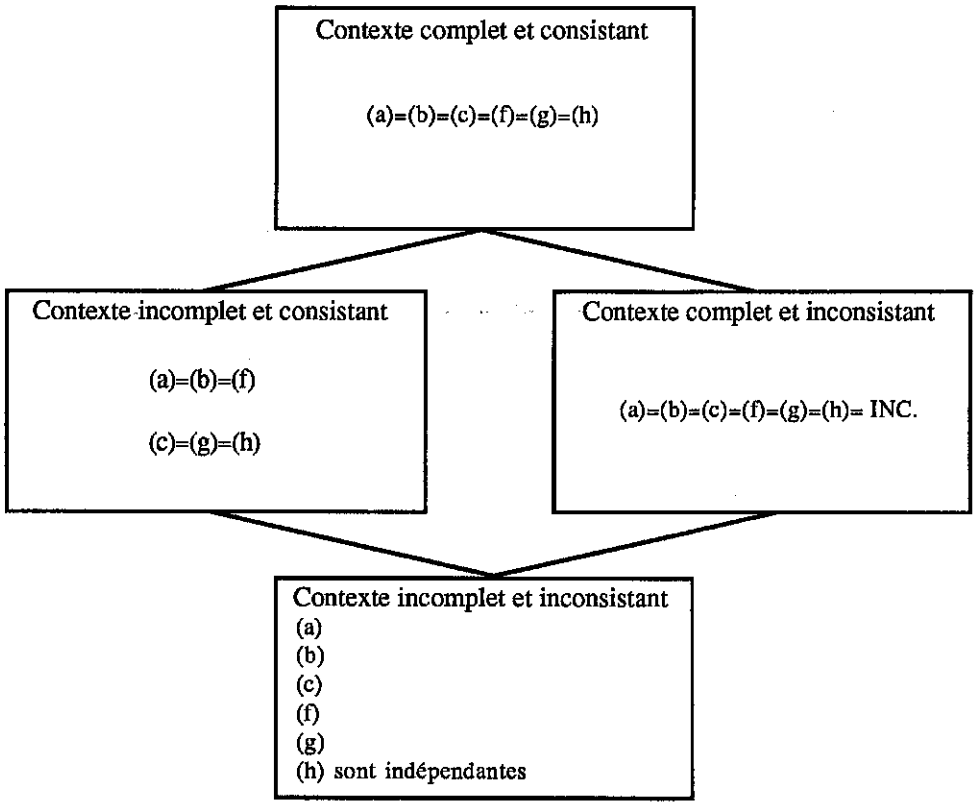

figure 7. Lien entre les diverses interprétations d'une requête suivant le contexte d'interrogation. Il est clair que la réponse positive en (a), entraîne la réponse positive dans toutes les autres questions ($F \equiv Ctxt([A/R])$). De même (b) entraîne (c), (f), (g) et (h), et même (e) si la valuation complétée canoniquement est consistante. (c) entraîne (f), (g) et (h). (f) entraîne (g) et (h). Et enfin (g) entraîne (h). Il est clair par ailleurs, que si [A/R] est consistant il n'a qu'une extension: lui-même et par conséquent les réponses aux questions (a), (b), (f), d'une part et (c), (g) et (h) d'autre part sont identiques.

L'interprétation (d) est difficile à justifier dans la mesure où les étiquettes ne contiennent pas d'informations quand à la nature du raisonnement tenu.

Les interprétations (a) et (h) sont les plus extrêmes, (a) exige que toutes les complétions de [A/R] permettent de dériver F. (h) demande simplement s'il est possible, dans un contexte où la connaissance est plus complète que F soit un théorème.

(b) et (f) correspondent aux interprétations de logiques des défauts de Reiter [Reiter 80], c'est-à-dire que le système considère les extensions de [A/R], c'est-à-dire que si [A/R] est inconsistant ([A/R] étant incomplet il doit donc être complété pour redevenir consistant et fidèle à la réalité), il est complété par le système afin d'évaluer la requête dans une extension correspondant plus à la réalité. Il faut noter que (a) et (b) d'une part et (h) et (g) d'autre part sont équivalents si une complétion inconsistante est considérée comme permettant de dériver n'importe quelle formule.

(c) et (g) considèrent les extensions comme (b) et (f), mais exigent qu'il n'y ait qu'une complétion des extensions dans lesquelles F soit un théorème comme (h).

Il faut noter que ce type d'interprétation des contextes d'interrogation pourrait être étendu à l'interprétation des étiquettes si celle-ci n'était fixée d'avance. Il y aurait alors $2^6=36$ interprétations possibles d'une requête.

10.3.Manipulation du contexte d'interrogation

Les différentes fonctions décrites ici permettent d'obtenir des renseignements sur la requête. Elles sont suffisamment puissantes pour permettre de répondre aux différentes interprétations d'une requête. Dans les parties suivantes, ces fonctions seront utilisées dans la spécification des procédures de réponse aux requêtes comme s'il s'agissait de données.

10.3.1. Conditions de dérivabilité

Il est possible de fournir tout simplement à l'utilisateur l'ensemble des conditions de dérivabilité de f, c'est-à-dire l'ensemble des hypothèses à ajouter à [A/R] pour que f soit valide. Cet ensemble se présente comme une étiquette, telle que pour chaque environnement [A'/R'], f soit valide dans [A∪A'/R∪R'] et (A∩A'∪R∩R')∪(A∩R'∪A'∩R)={}. Cette approche est analogue à celle prise par Marie-Odile Cordier avec l'ATMS pour son logiciel Sherlock [Cordier 87].

L'algorithme chargé de fournir cette étiquette est le même que celui utilisé pour connaître la validité de f dans [A/R].

```
DIFFERENCE( Ax Rt Etiq) =
[ diff <- {}
  PourTout env≡Etiq
  Faire Si Rt∩ax(env)={}
    Alors diff <- vetiq( { diff,
                           {< ax(env)\Ax
                             {r\Rt; r≡rt(env) & rt(env)∩Ax={}}>})
  <= diff ]
```

10.3.2. Consistance

Il est parfois nécessaire de vérifier la consistance du contexte de requête. Comme pour tout environnement, il faut vérifier la consistance interne ({}-consistance qui n'est pas nécessaire si R={}) et enfin la consistance externe (⊥-consistance).

```
CONSISTANTp( A R ) =
{}-CONSISTANCE({<A {R}>}) & ⊥-CONSISTANCE({<A {R}>})
```

Par ailleurs, il faudra aussi vérifier la consistance vis-à-vis de la formule de requête (ci-consistance).

```
R-CONSISTANTp( A R f ) =
CI-CONSISTANCE({<A {R}>} f)
```

10.3.3. Extensions

Un certain nombre d'interprétations des requêtes — et en particulier celles autorisant des environnements de requête inconsistants — font appel à la notion d'extension. La fonction calculant les extensions est donc largement utilisée. Il est clair que si [A/R] est un contexte consistant, il n'a qu'une extension: lui-même.

D'autre part, notre définition des valuations étendues nous permet d'utiliser le théorème 9 pour calculer les valuations étendues.

Théorème 9:
EVal([A/R]) =

$$\{[A'/R'];\ A\subseteq A',\ R\subseteq R',\ A'\cap R'=\{\}\ \&\ \forall[A''/R'']\equiv ETIQUETTE[\bot];\ A\subseteq A''\ \&\ R\subseteq R''\Rightarrow (A'\cap R'')\cup(A''\cap R')\neq\{\}\}$$

La méthode pour calculer les extensions n'est pas justifiée plus avant, elle est la suivante:

```
EXTENSIONS( Ax Rt ) =
vetiq({RENDRE-VALIDE(ax(incdisj)
                     Ax
                     rt(incdisj)
                     Rt);  incdisj≡*incetiq*})
```

10.4.Requêtes

L'ensemble des algorithmes permettant de répondre à l'une quelconque des interprétations possibles d'une requête est présenté ici. L'utilisateur peut alors employer la fonction correspondant à son besoin réel.

10.4.1. Dérivabilité dans toutes les complétions du contexte d'interrogation

Rechercher si une formule est dérivable dans le contexte proposé par l'utilisateur correspond à l'attitude à adopter a priori. L'utilisateur est alors considéré comme connaissant la nature du contexte dans lequel il se place. Il faut cependant que $\perp$ ne soit pas dérivable de [A/R], auquel cas il serait bon de se rabattre sur une de ses extensions (valuations étendues minimales).

Il n'est pas difficile de répondre à une requête si la valuation correspondant au contexte de requête est complète, il est alors possible de répondre en la comparant avec ceux définis par l'étiquette. Cela n'est pas non plus difficile si cette valuation est seulement complète vis-à-vis de l'étiquette, ce qui est défini comme suit:

Un contexte [A/R] est complet vis-à-vis d'une étiquette L s'il existe un environnement [A'/R'] de L tel que $A' \subseteq A$ et $R' \subseteq R$, ou si pour tout environnement [A'/R'] de L $A \cap R' \cup R \cap A' \neq \{\}$. Dans le premier cas, la réponse à la requête est affirmative, dans le second elle est négative. La procédure permettant de vérifier la validité de F dans le contexte [A/R] peut donc être spécifiée comme suit:

```
VALIDE1(A R f) =
<{}{{}}>∈DIFFERENCE(A R f)
```

Si le contexte d'interrogation n'est pas complet vis-à-vis de l'étiquette de f, alors les problèmes posés par l'information incomplète resurgissent. Les solutions envisagées précédemment sont à examiner.

10.4.2. Dérivabilité dans toutes les complétions de toutes les extensions

Ce type de requête peut être formulé en vérifiant que le premier type de requête obtient une réponse positive pour chaque extension du contexte d'interrogation.

```
VALIDE2(A R f) =
∀<A' {R'}>∈EXTENSIONS(A R), <{}{{}}>∈DIFFERENCE(A' R' f)
```

10.4.3. Dérivabilité dans certaines complétions de toutes les extensions

Ici, il faut qu'il existe une complétion de chaque extension du contexte d'interrogation dans lequel la formule soit dérivable.

```
VALIDE3(A R f) =
∀<A' {R'}>∈EXTENSIONS(A R), DIFFERENCE(A' R' f)≠{}
```

10.4.4. Dérivabilité dans la complétion canonique

Pour qu'une formule soit valide dans la complétion canonique d'un contexte, il faut et il suffit qu'il existe une complétion de ce contexte dans lequel seules des restrictions sont ajoutées dans lequel la formule soit valide. Il faut donc qu'il n'y ait pas d'axiomes à ajouter au contexte d'interrogation pour que la formule soit valide dans celui-ci, ce qui s'exprime par:

```
VALIDE5(A R f) =
∃<{}{R'}>∈DIFFERENCE(A' R' f)
```

Dans certains cas, ce type de requête — du moins tel qu'il est défini ici — donnera des résultats surprenants. De tels exemples (voir l'exemple 6), sont dus à la présence de cycles pairs exclusifs, c'est-à-dire contenant un couple de nœuds entre lesquels existe un chemin aller et un chemin retour traversant un nombre impair d'antécédents négatifs. Un tel cycle valide l'un des nœuds en l'absence de l'autre.

Exemple 6:

Cet exemple est la réalisation de l'exemple 4. A et B sont posés comme hypothèses:

(addhyp A)
(addhyp B)

Puis A peut se déduire de l'absence de B et vice-versa:

(inf () (A) B)
(inf () (B) A).

La réponse à la requête sera positive pour A et négative pour B dans la complétion canonique de []:

(valide5 () () A) -> ()
(valide5 () () B) -> t

Il est possible de forcer la validité de A de manière extérieure au cycle par:

(addhyp C)
(inf () (C) A)

ou même (addft A) et alors

(valide5 () () A) -> t
(valide5 () () B) -> ()

L'explication est que la complétion canonique possède plusieurs interprétations l'une dans laquelle A est valide et B invalide et l'autre dans laquelle B est valide et A invalide. Le système choisit donc par lui-même — et ce au cours de la propagation — l'interprétation qui sera considérée pour cette réponse. Si par contre, la validité de l'un des nœuds est forcée, alors l'interprétation choisie par l'utilisateur est adoptée.

10.4.5. Dérivabilité dans toutes les complétions d'une extension

Les trois derniers types de requêtes ont même forme que les trois premiers. La différence réside dans l'exigence d'une propriété particulière au lieu d'une propriété générale.

```
valide6(A R f) =
∃<A' {R'}>∈extensions(A R); <{}{{}}>∈difference(A' R' f)
```

10.4.6. Dérivabilité dans une complétion d'une extension

```
VALIDE7(A R f) =
∃<A' {R'}>∈EXTENSIONS(A R), DIFFERENCE(A' R' f)≠{}
```

10.4.7. Dérivabilité dans une complétion du contexte d'interrogation

```
VALIDE8(A R f) =
DIFFERENCE(A' R' f)≠{}
```

Dans le système actuellement implémenté, toutes les approches cohabitent, c'est-à-dire qu'il est possible d'appeler la fonction correspondant à l'interprétation que l'on désire donner à sa requête. Bien sûr, ces fonctions ne sont pas implémentées sous la forme décrite ici, en particulier, les fonctions générales (définies au §10.3) sont rarement évaluées dans leur ensemble.

Les ensembles Nec(◇) et Pos(◇) correspondent à l'ensemble des nœuds pour lesquels la réponse aux requêtes (a) et (h) est positive. Ce résultat est obtenu grâce aux théorèmes 5 et 7 et à la complétude et correction des étiquettes des nœuds. La caractérisation de Nec(◇) et Pos(◇) ne se préoccupe pas de la consistance des valuations, il est possible d'obtenir l'équivalent de Nec(◇) et Pos(◇) pour les requêtes (b), (c), (f) et (g) en introduisant les notions d'extensions.

Il est à remarquer que tous les algorithmes utilisés par le système sont indépendants du critère de validité de l'utilisateur ce qui apporte une très grande flexibilité au système. Cela peut, par contre, être un handicap en terme d'efficacité des algorithmes.

11.Implémentation et optimisation

Tous les algorithmes exposés ici le sont sous forme naïve. Il est possible d'envisager un certain nombre d'optimisations.

11.1.Représentation des étiquettes

11.1.1. Vecteurs de bits

Il est possible de profiter de l'optimisation proposée par Johan De Kleer pour son système c'est-à-dire de représenter les ensembles d'hypothèses par des vecteurs de bits. Ainsi toutes les procédures de comparaison (inclus $\subseteq$, disjoints $\cap=\{\}$...) et de manipulation (union $\cup$, inter $\cap$, neg...) seront beaucoup plus rapides à s'exécuter. Une telle optimisation pourra se faire comme dans Token [De Brabant 88] où l'utilisateur choisit au début de sa session un nombre maximum d'hypothèses autorisées cela permettant de ne pas avoir à augmenter *tous* les vecteurs de bits contenus dans le système ce qui coûte énormément de temps. Il faudra alors une structure permettant pour chaque hypothèse de connaître sa position dans le vecteur de bits. Cela permettra, entre autre, de définir facilement une forme canonique pour les ensembles d'hypothèses et même les environnements. Il n'y aura, par contre, toujours pas de forme canonique pour les étiquettes mais il est possible d'en envisager une pour les ensembles d'environnements.

11.1.2. Listes ordonnées

Il est possible de stocker les hypothèses dans un ordre défini donné par la position de ces hypothèses dans la variable `*listhyp*`. Les opérations d'union, d'intersection et de différence se font alors en fonction de cet ordre. Dans le même ordre d'idée, les restrictions peuvent être classées dans les ensembles de restrictions suivant le même ordre, ce qui les rend bien plus aisé à retrouver.

11.2.Propagation incrémentale

L'algorithme décrit ici n'est cependant pas un modèle d'incrémentalité, il ne vise qu'à être correct. Un algorithme bien plus incrémental ne propageant que des différences d'étiquettes a été développé, malheureusement, à cause des inférences non monotones, ce dernier système est incorrect dans le cas général. Il l'est cependant s'il est utilisé de concert avec un système de raisonnement produisant un raisonnement linéaire ou si le nombre de justifications par nœud est restreint à une unique justification — comme c'est le cas pour ART ou MBR. Un tel système a de meilleures performances et l'avantage de détecter les cycles impairs.

11.3.Restauration de la consistance incrémentale

11.3.1. Restauration paresseuse de la consistance

La restauration paresseuse de la consistance externe consiste à ne pas rétablir celle-ci au sein du réseau mais à supprimer toutes les étiquettes du réseau et à ne les recalculer qu'à des fins d'interrogation ou de propagation. Une telle manière de faire est, bien sûr, contraire à l'idée d'utiliser un système de maintenance de la vérité. Il en est une autre, plus simple, qui consiste à supprimer les étiquettes inconsistantes des nœuds et à ne recalculer celles-ci qu'en cas de besoin.

11.3.2. Chercher l'inconsistance

Si lors de la propagation du nœud n, le fait $\bot$ appartient à CSQ*[n], il va falloir procéder à un réaménagement et il serait avantageux d'aller directement sur cet inconsistant ce qui simplifierait les calculs ensuite. Il faut alors explorer d'abord les composantes fortement connexes dont CSQ* contient $\bot$. Cela est utile pour synchroniser une implémentation parallèle de l'algorithme et permet de savoir s'il est possible d'examiner une nouvelle inférence car il n'y a plus de rétrogression possible.

11.3.3. Abandonner la consistance

Vu la façon d'évaluer les requêtes, il ne semble plus indispensable de garantir la consistance des étiquettes. En effet, les requêtes ne considèrent plus que les extensions — ou au moins les complétions valides — des contextes d'interrogation, si bien que les réponses se feront toujours en fonction de valuations valides.

Il reste à prouver que cette modification du système rendrait un système équivalent, mais une telle optimisation promet d'être très valable.

11.4.Détection des cycles impairs

Il est possible d'adapter l'algorithme à la détection des cycles impairs. Cela peut se faire en adoptant la solution proposée par James Goodwin [Goodwin 87]. Ainsi, lors de la propagation dans une composante fortement connexe, l'algorithme conserve le nom du nœud duquel il est parti et la parité des dépendances traversées. Il est ainsi capable de se rendre compte quand il revient sur le nœud de départ si le cycle est pair ou impair.

12.Relation avec les systèmes préexistants

12.1.Le TMS de Jon Doyle

Si toutes les étiquettes sont {} (OUT) ou {[|]} (IN) le travail de propagation est le même que celui fait par le TMS. Cependant, l'efficacité est moindre car il est nécessaire de tenir compte des opérations sur les contextes. En cas de cycles impairs, le système présenté bouclera contrairement à l'implémentation de James Goodwin. La notion de nœuds supportants étant évacuée, la procédure de propagation restera cependant assez efficace puisqu'elle s'arrêtera dès qu'une étiquette n'est pas modifiée. Elle n'aura pourtant pas toute l'efficacité du système de Doyle car il faudra examiner toutes les composantes fortement connexes.

Les différences entre le TMS et le CP-TMS concernent les deux pans de celui-ci:

- La propagation ne concerne plus la validité absolue mais des environnements de validité, ce qui entraîne l'abandon de la stratégie des nœuds supportants.
- La rétrogression est remplacée par une phase de rétablissement de la consistance du graphe de dépendances.

La différence principale est dans la rétrogression. Il ne semble plus possible de faire de la rétrogression de la même manière que le faisait le système de Jon Doyle. En particulier, le système présenté ici est incapable de rétablir la consistance après qu'une inconsistance ait été découverte. Il vaut mieux, pour faire de la rétrogression, utiliser la notion d'hypothèse présente dans le système, il est alors possible d'implémenter des mécanismes spécifiques de rétablissement de la consistance qui font un travail similaire à celui de la rétrogression (voir §8.3).

L'avantage du CP-TMS est bien sûr de pouvoir raisonner simultanément dans plusieurs contextes.

Il est à noter que la simulation d'un TMS doylien par le CP-TMS en déclarant comme hypothèses tous les nœuds apparaissant dans des OUT-listes donnera plutôt comme résultat l'implémentation de la logique des défauts de Raymond Reiter. On aura en effet une théorie, contenant les nœuds posés comme hypothèses et création de plusieurs extensions de la théorie uniquement s'il y a découverte de son inconsistance.

12.2.L'ATMS de Johan De Kleer

Le système présenté perd aussi en généralité par rapport au système de maintenance de la vérité à contextes de De Kleer [De Kleer 86a], en effet, il n'y a plus une forme unique — correspondant à la forme normale disjonctive — pour exprimer un contexte mais plusieurs formes, cela ne facilite pas les algorithmes de recherche. Le treillis perd alors beaucoup de son intérêt puisqu'il ne suffit plus de comparer les environnements (ou plutôt leurs ensembles d'axiomes) de l'étiquette d'un nœud au contexte courant pour savoir s'il est valide mais qu'il faut ensuite comparer les exceptions.

Il est possible d'utiliser le CP-TMS à la place de l'ATMS. Tous les environnements seront alors sans restriction. Le CP-TMS sera moins efficace car il devra tenir compte des opérations sur les restrictions et ne profitera pas des structures de vecteurs ni surtout de la minimalité.

Mais, de manière surprenante, cette extension du TMS est en fait une extension de l'ATMS, toutes les procédures évoquées ici se retrouvent en effet dans un récent article de Johan De Kleer [De Kleer 88] plus complet quant à l'exposition de la propagation que la définition de l'ATMS [De Kleer 86a]:

- PROPAGATE correspond à l'algorithme général de propagation d'étiquette.
- NOGOOD correspond au rétablissement de la consistance dans tout le réseau.
- WEAVE correspond au calcul de l'étiquette affectée à une justification avec vérification de la minimalité et de la $\perp$-consistance.
- Enfin UPDATE correspond au calcul et à la mise à jour de l'étiquette d'un nœud avec vérification de la minimalité.

La différence avec l'algorithme de l'ATMS est que la partie propagation de contextes est bien plus complexe à cause de la présence des justifications non monotones et donc bien plus exposée que dans les premiers papiers de Johan De Kleer où elle était implicite. Les problèmes posés par ces types d'environnements ont entraîné:

- Une nouvelle formulation de la minimalité.
- Une notion de consistance interne qui était absente dans les environnements simples et une redéfinition de la consistance externe.
- Une multiplication des interprétations données à une requête.

L'avantage principal du CP-TMS est qu'il permet d'exprimer simplement des inférences avec antécédents négatifs.

Cette extension du TMS vers les contextes est aussi une extension de l'ATMS vers les justifications non monotones. En fait le système ressemble beaucoup plus au second qu'au premier et superficiellement à ART. Le CP-TMS tente de concilier les avantages des deux systèmes et permet donc l'expression de manière simple et naturelle d'inférences de type défaut et la manipulation de contextes.

12.3.L'ATMS étendu d'Oskar Dressler

Une réponse au problème des multiples interprétations induites par une valuation a été proposée (voir §8.3). Elle consiste simplement à forcer tous les membres de OUT-listes à être des hypothèses. Ainsi, toutes les extensions seront présentes sous forme d'environnements. Cette solution rend le CP-TMS très proche de ce qui a été proposé par Oskar Dressler (voir §4.1.2).

En effet, déclarer chaque membre de OUT-liste comme hypothèse va permettre d'obtenir des environnements contenant ce membre de OUT-liste en restriction, ce qui n'est pas possible si ce membre de OUT-liste n'est pas une hypothèse. Le système ne peut plus alors choisir ses extensions comme il le faisait précédemment car elles sont toutes représentées par un environnement. Les réponses aux requêtes sont toutes correctes car alors il n'existe plus qu'une unique interprétation par valuation complète.

De surcroît, une simple conséquence de la proposition précédente est que la «définition» de l'environnement peut passer de l'existence d'une interprétation induite satisfaisant le nœud à la garantie que toutes les interprétations induites satisfont le nœud. Cela permet de rapprocher encore plus la définition des ensembles Pos(⋄) et Nec(⋄) de l'intuition.

Quand cette option est choisie pour le CP-TMS, on obtient alors une équivalence entre les environnements de CP-TMS:

[i1,...in|o1,...om]

et les environnements de l'extension de l'ATMS:

{i1,...in, Out(o1),...Out(om)}.

Il est certain que cette équivalence n'est pas aussi complète que cela, car supposons qu'un nœud a ait pour étiquette {[a1,...an|r1,...rm]} (respectivement {{a1,...an,Out(r1),... Out(rm)}}) et que ce nœud soit utilisé dans une justification:

<{}{a}>:b.

Le CP-TMS est alors capable de trouver une étiquette pour b, il s'agit de {[r1|],...[rm|],[|a1],...[|an]}, tandis que l'extension d'Oskar Dressler se trouve plutôt démunie. Il est cependant possible d'étendre le système de Dressler pour obtenir {r1,...rm,Out(a1),...Out(an)} mais alors tous les nœuds sont dédoublés (à cause de leur Out).

Le CP-TMS bien qu'il soit, dans ce cas particulier, équivalent au système de Dressler présente certains avantages sur celui-ci:

- Le système reste homogène; tous les nœuds sont traités de la même façon ce qui n'est pas le cas des nœuds Out(N) qui ne peuvent être justifiés.
- En conséquence le caractère propositionnel du système a été préservé. Il n'est pas besoin de se préoccuper de la structure des faits associés aux nœuds.
- Il évite la multiplication des nœuds et hypothèses supplémentaires qui n'ont pas été soumis par l'utilisateur ou le système de raisonnement.
- Enfin, le CP-TMS est plus souple dans sa forme la plus générale, il n'oblige aucun nœud (pas même les nœuds présents dans les OUT-listes) à être déclarés comme hypothèses.

12.4.Systèmes d'équations booléennes

Le travail présenté ici est issu de travaux destinés à trouver le cadre formel permettant d'exprimer à la fois TMS et ATMS. Le point de départ est l'expression de chaque système indépendamment comme résolvant, à sa manière, des systèmes d'équations booléennes [Brown& 87].

Pour Allen Brown, un ensemble de nœuds peut être représenté sous forme de variables booléennes. À chaque variable est associée une équation booléenne, fonction des autres variables, sous forme normale disjonctive. Cette équation correspond à la disjonction des justifications. Un nœud n sans justification est représenté par l'équation n=0. À un ensemble de justifications et de nœuds correspond donc un système à |𝒩| équations. Ce système doit être, de plus, si on le veut consistant, contraint par l'équation ⊥=0.

Exemple 7:

L'ensemble de justifications de l'exemple 1 permet de construire le système d'équations suivant:

$$\begin{cases} A=\sim B \\ B=\sim A|G \\ C=\sim B \\ \bot=C\&F \\ D=1 \\ E=0 \\ F=D\&\sim E \\ G=0 \end{cases}$$

Il existe des algorithmes permettant d'obtenir les solutions d'un tel système — ne contenant pas de cycle impair — grâce aux opérations de substitution et de normalisation. À l'ensemble des solutions d'un tel système correspond l'ensemble Γ(𝒩,𝒥,𝒞) — un élément de Γ(𝒩,𝒥,𝒞) contenant tous les nœuds d'une solution dont la variable est égale à 1 —, mais le modèle va plus loin puisqu'il permet d'exprimer la notion de solution bien fondée pour un nœud. De plus la spécification algébrique rend compte aisément d'optimisations du type de l'algorithme de James Goodwin [Goodwin 87].

L'intérêt du système algébrique est qu'il permet d'exprimer aussi le comportement du système de maintenance de la vérité à contextes. En effet, un ensemble d'hypothèses peut-être considéré comme des nœuds n'apparaissant pas dans la partie gauche d'une équation. Ces hypothèses combinées par l'opération de conjonction forment un treillis ordonné par la relation d'implication.

Exemple 8:

Cet exemple reprend en l'aménageant l'exemple précédent (7). Il est donné par l'ensemble de règles suivantes:

{G}:B
POSER-HYPOTHESE(A)
POSER-HYPOTHESE(B)
{A}:¬B
{B}:¬A
{¬B}:C
{}:¬E
{}: ¬G
{}:D
{D, ¬E}:F
{F, C}:⊥

Vis-à-vis de la négation, la transcription en équation n'est pas très rigoureuse, il manque des équations. Il faut cependant se rendre compte que le symbole ¬X est une proposition totalement indépendante de X (la négation booléenne étant exprimée par ~X). Le premier système d'équation est:

$$\begin{cases} B = G \\ \neg B = A \\ \neg A = B \\ C = \neg B \\ \neg E = 1 \\ G = 0 \\ D = 1 \\ F = D \,\&\, \neg E \\ \bot = F \,\&\, C \end{cases}$$

Le second système montre que l'ensemble des résultats du système dépend de la validité de l'hypothèse A:

$$\begin{cases} B = 0 \\ \neg B = A \\ \neg A = 0 \\ C = A \\ E = 0 \\ G = 0 \\ D = 1 \\ F = 1 \\ \bot = A \end{cases}$$

Les solutions des systèmes d'équations ne seront donc plus réduites à 1 ou 0 mais à une formule booléenne fonction des hypothèses. Appelons S(n) la formule booléenne correspondant au nœud n, cette fonction peut être mise en forme normale disjonctive; ainsi déterminer la validité du nœud n dans un environnement E revient à savoir si l'environnement permet de déduire la valeur 1 pour la solution de l'équation en n:

$$\text{n valide dans l'environnement E} \Leftrightarrow E \vdash S(n).$$

Le système rend compte de manière indépendante du TMS et de l'ATMS. L'idée du CP-TMS étant de fusionner les deux systèmes, il semble intéressant de pouvoir représenter celui-ci dans le cadre des systèmes d'équations booléennes. Le rapport entre les deux travaux est évident, c'est ce qui a conduit à justifier les algorithmes présentés ici en tant que manipulations de formules propositionnelles.

La solution est en effet relativement simple, elle considère comme dans la modélisation du TMS des équations avec négations mais comme dans celle de l'ATMS ces équations ne sont résolues qu'en fonction des hypothèses. Cependant, les problèmes se posant avec le CP-TMS se reproduisent dans la résolution de ces équations. En effet, en cas de cycles pairs dont aucun des membres n'est une hypothèse il n'est pas possible de trouver une solution (au moins unique) en fonction d'hypothèses. Par contre, dans le cas de systèmes ne présentant pas cette particularité, on obtient comme solution de chaque équation une formule exprimable sous forme normale disjonctive — correspondant à l'étiquette du nœud. Les algorithmes pour obtenir de telles formules en formes normales sont exactement les mêmes, aux optimisations près, que ceux utilisés par le CP-TMS.

13.Conclusion

L'extension du système de maintenance de la vérité à propagation qui a été présenté permet d'unifier les concepts présents à la fois dans le TMS et dans l'ATMS. Conçu comme une extension du système de maintenance de la vérité à propagation il se révèle être simultanément une extension du système à contextes. Il permet donc de raisonner dans divers contextes simultanément tout en manipulant des inférences non monotones. Le CP-TMS est caractérisé théoriquement par l'explicitation de la signification des environnements manipulés et par l'utilisation d'opérations pertinentes par rapport à cette signification. Les résultats obtenus permettent d'expliquer les réponses aux requêtes fournies par le système.

Cependant les inférences non monotones ne sont pas mieux traitées que par le TMS. En effet, la présence de cycles pairs dans le graphe de dépendances peut occasionner diverses interprétations construites à partir d'une valuation — ceci seulement si aucun nœud du cycle n'est déclaré comme hypothèse. Dans un tel cas, les réponses aux requêtes fondées sur l'existence d'une interprétation peuvent sembler quelque peu insuffisantes. Ce problème conduit à examiner diverses réponses inspirées de la rétrogression présente au sein du TMS.

Quoiqu'il en soit, le CP-TMS est un premier pas vers l'implémentation de systèmes de maintenance de la vérité liant TMS et ATMS dont l'action est clairement définie. Il a été implémenté en Lisp et répond aux caractéristiques décrites ici. Il permet de déclarer hypothèses et inférences et répond à différents types de requêtes.

Annexe: Preuves

Définition 1. Une *valuation* est une fonction propositionnelle ν: H$\longrightarrow${0,1}, H$\subseteq\mathfrak{U}$ qui fixe la validité de certaines hypothèses et l'invalidité d'autres. Un environnement [A/R] d'une étiquette représente une valuation fournie en extension:

$$\begin{array}{lll} A\cup R & \longrightarrow & \{0,1\} \\ n & \longmapsto & \text{si } n\in A\ \nu(n)=1 \\ & & \text{si } n\in R\ \nu(n)=0. \end{array}$$

Définition 2. Une valuation est dite une *valuation complète* si H=$\mathfrak{U}$.

Définition 3. Une valuation ν est dite *plus complète* qu'une valuation ν' ssi H'$\subseteq$H. L'ensemble des complétions de ν est l'ensemble des environnements dont la valuation correspondante est plus complète que celle correspondant à ν:

Comp([A/R]) ={ [A$\cup$A'/R$\cup$R']; (R$\cap$A')$\cup$(R'$\cap$A)$\cup$(A'$\cap$R')={} & R'$\cup$A'$\subseteq\mathfrak{U}$ }.

Théorème 1:

Comp([A/R]) = {[A$\cup$A'/R$\cup$R'];A'$\cup$R'$\subseteq\mathfrak{U}\backslash$(A$\cup$R) & A'$\cap$R'={} }

preuve. immédiate

Définition 4. L'ensemble des complétions maximales de [A/R] est l'ensemble des complétions complètes de [A/R] :

Cmax([A/R]) = {[A$\cup$A'/R$\cup$R']$\in$Comp([A/R]); A$\cup$A'$\cup$R$\cup$R'=$\mathfrak{U}$ }.

Théorème 2:

Cmax([A/R]) =

{[A$\cup$A'/R$\cup$R']$\in$Comp([A/R]); A'$\cup$R'=$\mathfrak{U}\backslash$(A$\cup$R) & A'$\cap$R'={} }

preuve.

{[A$\cup$A'/R$\cup$R']$\in$Comp([A/R]); A$\cup$A'$\cup$R$\cup$R'=$\mathfrak{U}$ }

= { [A$\cup$A'/R$\cup$R']$\in$Comp([A/R]);
A$\cup$A'$\cup$R$\cup$R'=$\mathfrak{U}$ & A'$\cup$R'$\subseteq\mathfrak{U}\backslash$(A$\cup$R) & A'$\cap$R'={} }

= { [A$\cup$A'/R$\cup$R']$\in$Comp([A/R]); A'$\cup$R'=$\mathfrak{U}\backslash$(A$\cup$R) & A'$\cap$R'={} }

Lemme 1: $\exists$C, $\nu\in$Cmax(C) $\Rightarrow$ Cmax(ν)={ν}

preuve.

[A/R]$\in$Cmax(C) $\Rightarrow$ A$\cup$R=$\mathfrak{U}$ (définition 4)
$\Rightarrow$ $\mathfrak{U}\backslash$(A$\cup$R)={}
$\Leftrightarrow$ Cmax([A/R])={[A$\cup$A'/R$\cup$R']; A'$\cup$R'={}} (Théorème 2)
$\Leftrightarrow$ Cmax([A/R])={[A/R]}

Lemme 2: $\nu\in$Cmax(ν) $\Leftrightarrow$ Cmax(ν)={ν}

preuve.

$\Rightarrow$ (Lemme 1)
$\Leftarrow$ immédiat

Définition 5. Une valuation ν' est une *complétion de degré i* d'une valuation ν ssi $\nu' \in Comp(\nu)$ et $|H'|=i$. L'ensemble des complétions de degré i d'une valuation ν est définie par

$$Comp^i(\nu)=\{\nu' \in Comp(\nu);\ |A' \cup R'|=i\}$$

Conséquences: $\forall \nu$ valuation, $Comp^{|H|}(\nu)=\{\nu\}$ & $Comp^{|\mathfrak{B}|}(\nu)=Cmax(\nu)$

Lemme 3: $\forall C \in Comp^i(\nu)$, $Comp^j(C) \subseteq Comp^j(\nu)$

preuve.

$[A'/R'] \in Comp^i([A/R]) \Rightarrow [A'/R']=[A \cup A''/R \cup R'']$

et donc

$Comp^j([A'/R']) = Comp^j([A \cup A''/R \cup R''])$

$= \{[A \cup A'' \cup A'''/R \cup R'' \cup R''']; |A \cup A'' \cup A'''/R \cup R'' \cup R'''|=j\}$

$\subseteq \{[A \cup A''''/R \cup R'''']; |A \cup A''''/R \cup R''''|=j\}$

$= Comp^j([A/R])$

Définition 6. L'ensemble des complétions de degré supérieur ou égal à i d'une valuation ν est dénoté par

$$Comp^{*i}(\nu)=\cup_{i \le j \le |\mathfrak{B}|} Comp^j(\nu)$$

Lemme 4: $\forall C \in Comp^i(\nu)$, $Comp(C) \subseteq Comp^{*i}(\nu)$

preuve.

$\forall C \in Comp^i(\nu)$,

$Comp(C) = Comp^i(C) \cup Comp^{*i+1}(C)$

$\subseteq Comp^i(C) \cup Comp^{*i+1}(\nu)$ (Lemme 3)

$= \{C\} \cup Comp^{*i+1}(\nu)$

$\subseteq Comp^{*i}(\nu)$

Définition 7. Une *interprétation* à partir de la valuation ν est une fonction propositionelle I: $\mathbb{N} \longrightarrow \{0,1\}$ construite à l'aide des règles suivantes:

$\forall n \in \mathbb{N}$, si $\nu(n)=1$ alors $I(n)=1$

sinon, si $\exists j \in LISTEDEJUST[n]$; $\forall h \in INLISTE[j]$ $I(h)=1$ & $\forall h \in OUTLISTE[j]$ $I(h)=0$

alors $I(n)=1$

sinon $I(n)=0$.

Définition 8. L'ensemble des interprétations construites à partir de la valuation [A/R] sera noté IntInd([A/R]).

Définition 9. À une interprétation correspond une *valuation interprétée* $\nu_I: \mathfrak{B} \longrightarrow \{1,0\}$; $\nu_I(h)=I(h)$.

Lemme 5: $IntInd(\nu) \subseteq \cup_{C \in Cmax(\nu)} IntInd(C)$

preuve.

$\forall I \in IntInd([A/R])$ $\exists \nu'=[A_I \backslash R/R_I \cup R] \in Cmax([A/R])$ où $[A_I/R_I]=\nu_I$

car $I \in IntInd([A/R]) \Rightarrow \forall h \in A$, $I(h)=1$ (définition 7)

or $\forall h \in \mathfrak{B}$; $I(h)=1$

soit $h \in A_I \backslash R \Rightarrow \nu'(h)=1$

soit $h \in R \backslash R_I$

$\exists$ $j \in LISTEDEJUST[h]$; $\forall n \in INLISTE[j]$, $I(n)=1$

et $\forall n \in OUTLISTE[j]$, $I(n)=0$

car $h \in R$, $I(h)=1$ et $I \in IntInd([A/R])$

$\Rightarrow I \in IntInd(\nu')$

Définition 10. Une interprétation I est dite *soutenable* (respectivement *insoutenable*) si et seulement si $I(\bot)=0$ (respectivement $I(\bot)=1$).

Définition 11. Une valuation ν est dite *consistante* si toutes les interprétations construites à partir de ν sont soutenables. Si une valuation n'est pas consistante elle est dite *inconsistante*.

Définition 12. Une interprétation construite à partir d'une valuation [A/R] est dite *compatible* si $\forall h \in A$, $I(h)=1$ et $\forall h \in R$, $I(h)=0$.

Conséquence: Toute interprétation construite à partir d'une valuation [A/{}] est compatible.

preuve. Par la définition de l'interprétation (définition 7).

Définition 13. Une *valuation étendue* ν' d'une valuation ν est une valuation telle que $\forall h \in \mathcal{H}$; $\nu(h)=1 \Rightarrow \nu'(h)=1$, $\forall h \in \mathcal{H}$; $\nu(h)=0 \Rightarrow \nu'(h)=0$, et $\forall I'$, interprétation à partir de ν', I' soit soutenable. L'ensemble des valuations étendues de ν est caractérisé par

$$EVal(\nu)=\{\nu' \in Comp(\nu);\ \forall I \in IntInd(\nu'),\ I(\bot)=0\}.$$

Définition 14. Une *valuation étendue minimale* d'une valuation ν est une valuation étendue ν' de ν telle qu'il n'y ait pas d'autres valuations étendues de ν dont ν' soit une valuation étendue. Une valuation étendue minimale de ν est appelée une extension de ν. L'ensemble des extensions de n est caractérisé par

$$Ext(\nu)=\{\nu' \in EVal(\nu);\ \neg(\exists \nu'' \in EVal(\nu);\ n' \in EVal(\nu''))\}$$

Théorème 3: Si $\forall I$, interprétation à partir de ν, I est soutenable, alors l'ensemble des valuations étendues minimales de ν est restreint à $\{\nu\}$. Il peut être formulé ainsi:

$$\forall I \in IntInd(\nu),\ I(\bot)=0 \Rightarrow Ext(\nu)=\{\nu\}$$

preuve. ν est bien une valuation étendue d'elle-même. Par ailleurs, s'il existe une valuation étendue ν' de ν différente de ν dont ν soit une valuation étendue, cela signifie que:

$$\nu'(h)=1 \Leftrightarrow \nu(h)=1 \text{ et } \nu'(h)=0 \Leftrightarrow \nu(h)=0$$

ce qui revient à dire que $\nu=\nu'$ ce qui est contraire à l'hypothèse initiale et donc $\forall \nu' \in Ext(\nu)$, $\nu'=\nu$. Autre formulation:

$\nu \in EVal(\nu)$ car $\nu \in Comp(\nu)$ et $\forall I \in IntInd(\nu)$, $I(\bot)=0$
par ailleurs,
si $\exists \nu' \in EVal(\nu)$; $\nu \in EVal(\nu')$
alors $\nu'(h)=1 \Leftrightarrow \nu(h)=1$ et $\nu'(h)=0 \Leftrightarrow \nu(h)=0$
et donc $\nu'=\nu$
$\Rightarrow Ext(\nu)=\{\nu\}$

Définition 15. Une étiquette d'un nœud n est définie comme l'ensemble des valuations dont toutes les complétions sont soutenables et ont au moins une interprétation qui considère n comme valide:

$$ETIQUETTE[n] = \{C\ ;\ \forall \nu \in Comp(C), \exists I \in IntInd(\nu);\ I(n)=1\ \}$$

Définition 16. L'ensemble des formules dérivées de [A/R], c'est l'ensemble des nœuds qui sont toujours valides sous les hypothèses [A/R], c'est-à-dire les nœuds dont [A/R] est une complétion d'un environnement de leur étiquette:

$$Ctxt(\nu) = \cup_{C;\nu \in Comp(C)} \{n \in \mathcal{N};\ C \in ETIQUETTE[n]\ \}$$

Lemme 6: $Ctxt(\nu) \subseteq \cap_{C \in Cmax(\nu)} Ctxt(C)$

preuve.
$n \in Ctxt(\nu) \Leftrightarrow \exists C;\ \nu \in Comp(C);\ C \in ETIQUETTE[n]$
or $\forall C;\ \nu \in Comp(C)$ et $\forall C' \in Cmax(\nu)$ on a $C' \in Comp(C)$
et donc
$\forall C' \in Cmax(\nu)\ \exists C;\ C' \in Cmax(C);\ C \in ETIQUETTE[n]$
$\Leftrightarrow \forall C' \in Cmax(\nu),\ n \in Ctxt(C')$

Définition 17. L'ensemble des formules nécessaires dans [A/R] est l'ensemble des nœuds qui doivent être valides dans toutes les interprétations construites à partir de [A/R]:

$$Nec(\nu) = Ctxt(\nu) \cup \cap_{C \in Comp(\nu) \setminus \{\nu\}} Nec(C)$$

Conséquence: Si $x \in Ctxt(\nu)$ alors $x \in Nec(\nu)$

Théorème 4: $Nec(\nu) = \cap_{C \in Cmax(\nu)} Ctxt(C)$

preuve.
i) $Nec(\nu) \subseteq \cap_{C \in Cmax(\nu)} Ctxt(C)$:
a) $Cmax(\nu)=\{\nu\}$
$n \in Nec(\nu) \Leftrightarrow n \in Ctxt(\nu) \Leftrightarrow n \in \cap_{C \in Cmax(\nu)} Ctxt(C)$

b) $Cmax(\diamond)\neq\{\diamond\} \Rightarrow Cmax(\diamond)\subseteq Comp(\diamond)\setminus\{\diamond\}$ (contraposée du lemme 2)
. $n\in Ctxt(\diamond) \Rightarrow n\in\bigcap_{C\in Cmax(\diamond)}Ctxt(C)$ (lemme 6)
. $n\in\bigcap_{C\in Comp(\diamond)\setminus\{\diamond\}}Nec(C)$
$\Rightarrow n\in\bigcap_{C\in Cmax(\diamond)}Nec(C)$
or $C\in Cmax(\diamond) \Rightarrow C\in Cmax(C)$ (par lemme 2) et donc $Nec(C)=Ctxt(C)$
$\Rightarrow n\in\bigcap_{C\in Cmax(\diamond)}Ctxt(C)$
ii) $Nec(\diamond)\supseteq\bigcap_{C\in Cmax(\diamond)}Ctxt(C)$:[1]
pas 0)
$\forall C\in Comp^{|\mathfrak{R}|}(\diamond)$,
$Cmax(C)=\{C\}=Comp(C)$
$\Rightarrow Nec(C) = Ctxt(C)\cup\bigcap_{C'\in Comp(C)\setminus\{C\}}Nec(C')$
$= Ctxt(C)$
$=\bigcap_{C'\in Cmax(C)}Ctxt(C')$
pas d'induction)
$\forall C\in Comp^{i}(\diamond)$,
$\forall C'\in Comp^{*i+1}(\diamond)$, $Nec(C')=\bigcap_{C''\in Cmax(C')}Ctxt(C'')$ (hypothèse d'induction)
alors $Nec(C)$
$= Ctxt(C)\cup\bigcap_{C'\in Comp(C)\setminus\{C\}}Nec(C')$ (définition 17)
$= Ctxt(C)\cup\bigcap_{C'\in Comp^{*i+1}(C)}Nec(C')$ (définition 6)
$= Ctxt(C)\cup\bigcap_{C'\in Comp^{*i+1}(C)}\bigcap_{C''\in Cmax(C')}Ctxt(C'')$ (hypothèse d'induction)
$= Ctxt(C)\cup\bigcap_{C'\in Cmax(C)}\bigcap_{C''\in Cmax(C')}Ctxt(C'')$
car $\forall C'\in Comp^{*i+1}(C)$, $Cmax(C')\subseteq Cmax(C)$ (lemme 3, $j=|\mathfrak{R}|$)
$= Ctxt(C)\cup\bigcap_{C'\in Cmax(C)}Ctxt(C')$
$=\bigcap_{C'\in Cmax(C)}Ctxt(C')$ (lemme 6)

Théorème 5: $\forall n\in\mathbb{N}$, $(n\in Nec(\diamond) \Rightarrow \forall\diamond'\in Cmax(\diamond), \exists I\in IntInd(\diamond'), I(n)=1)$

preuve.
$n\in Nec(\diamond)$
$\Leftrightarrow$ $\forall C\in Cmax(\diamond), n\in Ctxt(C)$ (théorème 4)
$\Leftrightarrow$ $\forall C\in Cmax(\diamond), \exists C'; C\in Comp(C'); C'\in ETIQUETTE[n]$ (définition 16)
$\Leftrightarrow$ $\forall [A/R]\in Cmax(\diamond), \exists A'\subseteq A, R'\subseteq R; [A'/R']\in ETIQUETTE[n]$ (définition 3)
$\Leftrightarrow$ $\forall C\in Cmax(\diamond), \exists I\in IntInd(C), I(n)=1$

Définition 18. L'ensemble des nœuds possibles dans $\diamond$ est l'ensemble des nœuds qui sont valides dans une interprétation construite à partir de $\diamond$:

$$Pos(\diamond) = \bigcup_{C\in Comp(\diamond)} Ctxt(C)$$

Théorème 6: $Pos(\diamond) = \bigcup_{C\in Cmax(\diamond)} Ctxt(C)$

preuve.
$Pos(\diamond)$
$= \bigcup_{C\in Comp(\diamond)} Ctxt(C)$ (définition 18)
$= \bigcup_{C\in Comp(\diamond)} \bigcup_{C'; C\in Comp(C')} \{n\in\mathbb{N}; C'\in ETIQUETTE[n]\}$ (définition 16)
$= \bigcup_{C\in Cmax(\diamond)} \bigcup_{C'; C\in Comp(C')} \{n\in\mathbb{N}; C'\in ETIQUETTE[n]\}$
Car $\forall C\in Comp(\diamond)\setminus Cmax(\diamond)\ \forall C''\in Cmax(C)$
$C''\in Cmax(\diamond)$ et $\{C'; C\in Comp(C')\}\subseteq\{C'; C''\in Comp(C')\}$
$= \bigcup_{C\in CMax(\diamond)} Ctxt(C)$

Théorème 7: $\forall n\in\mathbb{N}$, $(n\in Pos(\diamond) \Rightarrow \exists\diamond'\in Cmax(\diamond), \exists I\in IntInd(\diamond'), I(n)=1)$

preuve.
$n\in Pos(\diamond)$
$\Leftrightarrow$ $\exists\diamond'\in Cmax(\diamond); n\in Ctxt(\diamond')$ (théorème 6)
$\Leftrightarrow$ $\exists\diamond'\in Cmax(\diamond); \exists\diamond''\in Cmax(\diamond'); \diamond''\in ETIQUETTE[n]$ (définition 16)
$\Leftrightarrow$ $\exists\diamond'\in Cmax(\diamond); \diamond'\in ETIQUETTE[n]$
$\Leftrightarrow$ $\exists\diamond'\in Cmax(\diamond); \forall\diamond'''\in Comp(\diamond'); \exists I\in IntInd(\diamond'''), I(n)=1$ (définition 15)
$\Leftrightarrow$ $\exists\diamond'\in Cmax(\diamond); \exists I\in IntInd(\diamond'), I(n)=1$

[1] Recherche désespérément démonstration plus élégante (ne faisant pas intervenir les $Comp^i$). À noter que cette démonstration de la seconde partie du théorème est suffisante pour démontrer tout le théorème.

Théorème 8: $Ctxt(\diamond) \subseteq Nec(\diamond) \subseteq Pos(\diamond)$

preuve.
$Ctxt(\diamond) \subseteq Nec(\diamond)$: par définition.
$Nec(\diamond) \subseteq Pos(\diamond)$: s'appuie sur le fait que si $\forall x \in X, \exists y \in Y; x \subseteq y$ alors $\cap Y \subseteq \cup X$.

Théorème 9:
$EVal([A/R]) =$
$\{[A'/R']; A \subseteq A', R \subseteq R', A' \cap R' = \{\}$
$\quad \& \forall [A''/R''] \in ETIQUETTE[\bot]; A \subseteq A'' \& R \subseteq R'' \Rightarrow (A' \cap R'') \cup (A'' \cap R') \neq \{\}\}$

preuve.
$\{[A'/R']; A \subseteq A', R \subseteq R', A' \cap R' = \{\}$
$\quad \& \forall [A''/R''] \in ETIQUETTE[\bot]; A \subseteq A'' \& R \subseteq R'' \Rightarrow (A' \cap R'') \cup (A'' \cap R') \neq \{\}\}$
$= \{[A'/R'] \in Comp([A/R]);$
$\quad \forall [A''/R''] \in ETIQUETTE[\bot], [A'/R'] \notin Comp([A''/R'']) \}$ (définition 1)
$= \{[A'/R'] \in Comp([A/R]);$
$\quad \neg(\exists I \in IntInd([A'/R']); I(\bot)=1) \}$ (définition 15)
$= \{[A'/R'] \in Comp([A/R]); \forall I \in IntInd([A'/R']), I(\bot)=0 \}$ (définition 7)
$= EVal([A/R])$ (définition 13)

Références

[Aho& 74]
Alfred Aho, John Hopcroft, Jeffrey Ullman
The design and analysis of computer algorithms
Addison-Wesley, Reading (MA US) (rep. Data structures and algorithms, 1983; trad. InterÉditions, Paris (FR), 1987), 1974

[Brown& 87]
Allen Brown, Dale Gaucas, Dan Benanav
An algebraic foundation for truth maintenance
Actes 10ième IJCAI, pp973-980, Milan (IT), 1987

[Clayton 86]
Bruce Clayton
ART programming tutorial 2: a first look at viewpoints
Inference corporation, Los Angeles (CA US), 1986

[Cordier 87]
Marie-Odile Cordier
Unification contextuelle et raisonnement hypothétique: le moteur d'inférence SHERLOCK.
Actes 6ième RFIA, pp787-797, Antibes (FR), 1987

[De Brabant 88]
Philippe De Brabant
TOKKEN: manuel de référence
Rapport interne, Cognitech, Paris (FR), 1988

[De Kleer 86a]
Johan De Kleer
An assumption-based TMS
Artificial intelligence 28-II, pp127-162, 1986

[De Kleer 86b]
Johan De Kleer
Extending the ATMS
Artificial intelligence 28-II, pp163-196, 1986

[De Kleer 88]
Johan De Kleer
A general labeling algorithm for assumption-based truth maintenance
Actes 7ième AAAI, pp188-192, Saint-Paul (MN, US), 1988

[Doyle 79]
Jon Doyle
A truth maintenance system
Artificial intelligence 12-III, pp231-272, 1979

[Dressler 88]
Oskar Dressler
Extending the basic ATMS
Actes 8ième ECAI, pp535-540, München (DE), 1988

[Goodwin 87]
James Goodwin
A theory and system for non monotonic reasoning
Linköping studies in science and technology 165, 1987

[Martins& 83]
João Martins, Stuart Shapiro
Reasoning in multiple belief spaces
Actes 8ième IJCAI, pp370-373, Karlsruhe (DE), 1983

[Martins& 88]
João Martins, Stuart Shapiro
A model for belief revision
Artificial intelligence 35-I, pp25-79, 1988

[Petrie 87]
Charles Petrie
Revised dependency directed backtracking for default reasoning
Actes 6ième AAAI, pp167-172, Seatle (WA US), 1987

[Reiter 80]
Raymond Reiter
A logic for default reasoning.
Artificial Intelligence 13-I, pp81-132, 1980

[Williams 84]
Chuck Williams
ART: The advanced reasoning tool, conceptual overview
Inference corporation, Los Angeles (CA US), 1984