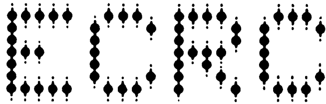
in : Compte-Rendus des $2^{èmes}$ Journées
Bases de Données Avancées. 22-25 Avril 1986.
Giens (Var. France).



Internal Report KB-14
12 décembre 1985

Sur la Validité des Schémas de Bases de Données

François Bry
Rainer Manthey

Résumé

Les mises à jour d'un schéma de base de données peuvent porter sur ses trois constituants, (1) les déclarations de structures, (2) les règles de cohérence (et de déduction) et (3) les transactions. Ces constituants doivent vérifier certaines propriétés, définissant les schémas *valides*. Un système de gestion de schémas doit pouvoir vérifier la validité des schémas (indépendamment de toute extension de la base) et détecter les modifications des schémas violant les conditions de validité. Dans cet article nous décrivons tout d'abord l'approche que nous avons suivi pour réaliser un système de gestion des déclarations de structures, premier composant disponible d'un système de gestion de schémas en cours de développement. Cette approche fait de la validité de déclarations de structures un problème semblable à la vérification des règles de cohérence dans une base de données. Nous montrons ensuite que la validité des règles (de cohérence et de déduction) correspond à une propriété logique plus forte que la consistence, la *satisfaisabilité finie*. Dans certains cas, cette propriété peut être détectée par des méthodes de réfutation de preuve automatique de théorèmes. Considérant la résolution, sur laquelle s'appuient les procédures de réfutation les plus performantes, nous caractérisons les additions nécessaires pour étendre une procédure de réfutation en une méthode complète pour la satisfaisabilité finie. Une telle extension de la résolution est décrite.

European Computer-Industry
Research Centre GmbH
Arabellastr. 17
D-8000 Muenchen 81
West Germany

1. Introduction

On distingue classiquement dans une base de données entre le schéma, qui comprend des informations structurelles, et l'extension (ou état) qui est un ensemble d'informations élémentaires (ou faits). Une base est dite cohérente [Delobel 82] si son extension respecte les conditions imposées par le schéma. A un même schéma peuvent correspondre, de manière cohérente, successivement plusieurs extensions. La plupart des mises à jour d'une base de données consiste précisement à définir une nouvelle extension. Le schéma d'une base peut également être modifié, bien qu'en général moins fréquemment que l'extension. Les mises à jour d'un schéma doivent, d'une part être compatibles avec l'extension courante (ou entraîner sa modification), et d'autre part respecter certaines conditions invariantes. Ces conditions, qui découlent en partie du modèle de données considéré, définissent les schémas *valides*. De même que les différentes extensions successivement associées à un schéma en sont des réalisations, les schémas successifs d'une base sont des réalisations du modèle de données qui peut être vu comme un méta-schéma (cf. figure 1 ci-dessous).

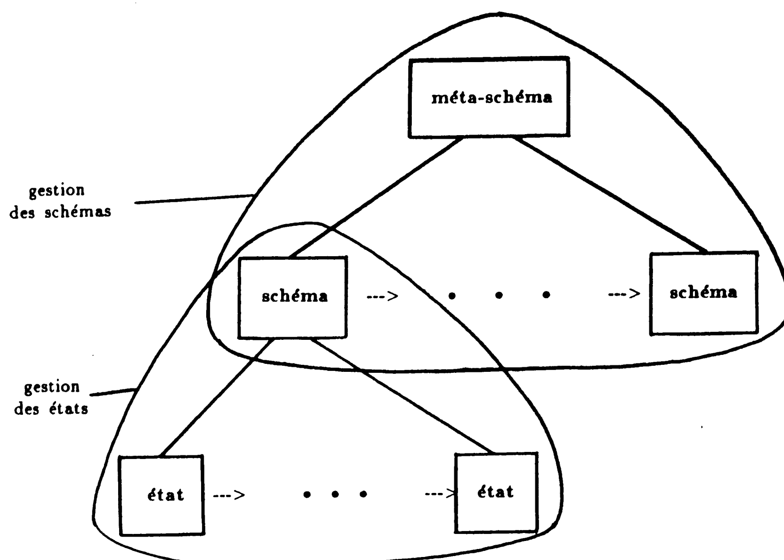


figure 1

On peut distinguer deux niveaux de gestion : celui de gestion des états qui maintient la cohérence entre l'extension et le schéma, et le niveau de gestion des schémas, qui, de manière semblable, maintient la cohérence entre le schéma et le méta-schéma. Trois niveaux d'information peuvent être distingués, celui des états, des schémas et celui du méta-schéma. Seules, les informations de ce dernier niveau sont invariables. Détecter la validité ou la non-validité d'un schéma est un problème distinct de celui de la vérification de la cohérence d'une base. En particulier, le premier n'est pas décidable alors que le second l'est. Si des méthodes conçues pour vérifier la cohérence d'une base sont utilisable pour détecter partiellement la validité d'un schéma (cf. section 2), ce dernier problème demande des solutions nouvelles (cf. section 3).

Nous présentons dans cet article certains éléments d'un système de gestion de schémas de bases de données, destiné à fournir des aides tant lors de la définition que durant la vie de la base. Un schéma de base de données (conventionnelle ou déductive) peut être vu comme formé de trois parties :

- un ensemble de *déclarations de structures.*
  Si, par exemple, on considère le modèle relationnel, cette première partie fournira la liste des relations, de leurs attributs et des domaines.

- un ensemble de *règles.*
  Dans le cas d'une base conventionnelle, il s'agit uniquement de règles de cohérence (ou contraintes d'intégrité). Une base déductive comprend également des règles de déduction.

- un ensemble de *transactions.*
  Cette dernière partie d'un schéma définie les opérations élémentaires de mise à jour possibles (licites) de l'extension de la base, ainsi que des opérations composées (définie par exemple par les usagers).

Le méta-schéma, dont tout schéma peut être vu comme une réalisation, comprend également trois parties. Les méta-structures expriment les types d'objets du modèle de données (par exemple, le type "relation" pour le modèle relationnel, le type "entité" pour le modèle Entité-Association). Elles définissent également la syntaxe des règles de cohérence et de déduction, ainsi que celle des transactions. Les méta-règles sont les conditions de valité d'un schéma. Celles concernant les déclarations de structures découlent du modèle de données. D'autres portent sur les règles et les transactions. L'une d'entre elle, clairement nécessaire, impose à l'ensemble des règles d'être logiquement consistent. Les méta-transactions sont les primitives de manipulation des schémas, i.e. les opérations d'insertion, de suppression et de mise à jour des trois parties d'un schéma. Les rôles respectifs des constituants du méta-schéma sur ceux d'un schéma sont résumés par la figure 2 :

	déclarations de méta- structure	**méta-règles**	**méta-transactions**
déclarations de structures	**syntaxe des déclarations de structure**	**conditions de validité des déclarations de structures**	**manipulations de déclarations de structures**
règles	**syntaxe des règles**	**conditions de validité des règles**	**manipulations de règles**
transations	**syntaxe des transactions**	**conditions de validité des transactions**	**manipulations de transactions**

figure 2

Nous proposons, dans cet article, des méthodes de vérification de la validité des deux premiers

constituants d'un schéma, les déclarations de structures et les règles. Les problèmes liés à la validité des transactions ne sont pas abordés ici. Dans une première partie, tirant profit de la "vision bases de données" de la gestion des schémas présentée plus haut, la validité des déclarations de structures est traitée comme un problème de vérification de règles de cohérence, bien que de complexité moindre. Les principes du système de gestion de déclarations de structure que nous avons développé sont exposés. Une seconde partie est consacrée à la validité des ensembles de règles de cohérence et de déduction. Nous montrons tout d'abord que la consistence logique, bien que condition nécessaire, n'est pas suffisante. Les règles d'une base de données, qu'elle soit déductive ou conventionnelle, doivent en fait vérifier une propriété logique plus forte, la *satisfaisabilité finie*. Lors de la recherche d'une méthode efficace de vérification de cette propriété, nous avons été amené à considérer les méthodes de réfutation (voir, par exemple, [Loveland 78]), introduites en preuve automatique de théorèmes, qui dans certains cas, détectent la satisfaisabilité finie. Cependant, aucune de ces méthodes n'est complète pour cette propriété. A partir du principe de résolution, sur lequel se fondent les démonstrateurs de théorèmes actuellement les plus performants, nous montrons quels types d'addition sont nécessaires pour étendre une procédure de réfutation en une méthode saine et complète pour la satisfaisabilité finie. Finallement, une telle méthode fondée sur le principe de résolution est décrite et commentée.

2. Gestion de Déclarations de Structures Valides

Les structures déclarées dans un schéma sont les types (conceptuels) des données de la base. Ces types sont, entre autres, les différentes relations d'une base relationnelle, les différents ensembles ("sets") d'une base CODASYL ou les classes d'entités si le schéma suit le modèle Entité-Association. Ces structures sont déclarées à l'aide des "méta-types" (ou types de structure) du modèle de données considéré : méta-type "relation" du modèle relationnel, méta-type "set" en CODASYL, etc... Conformément au paradigme proposé dans l'introduction et illustré par la figure 1, la déclaration d'une structure peut être vue comme l'insertion d'un fait dans l'extension d'une base. La déclaration d'une classe d'associations "**emploie**" entre deux classes d'entités "**personnes**" et "**entreprises**" revient à insérer un ou plusieurs faits traduisant cette classe d'associations. Une telle insertion n'est possible que sous certaines conditions, par exemple que les classes d'entités "**personnes**" et "**entreprises**" aient auparavant été déclarées. Ces conditions peuvent être vues comme les règles de cohérence, semblables à celles d'une base de données. Afin de montrer que cette comparaison est féconde, nous allons considérer un exemple concret. A cette fin, nous décrivons rapidemment un modèle sémantique [Manthey 85], proposé pour le système de gestion de schémas que nous développons.

Sans reprendre toutes les propositions qui ont été faites en matière de modélisation, ce modèle est un exemple representatif de la capacité d'expression nécessaire en bases de données. Il se fonde sur le modèle Entité-Association de Chen [Chen 79] qui offre d'interessantes possibilités de représentations

graphiques et qui est très proche du modèle relationnel. Le modèle original est enrichi d'un concept de "sous-classes d'entités", destiné à permettre l'expression de "généralisations" [Smith 77], ainsi que par celui de "rôle" d'une classe d'associations. Certaines particularités du modèle de Chen, telle que la possibilité de définir des cardinalités d'associations, n'ont pas été conservées, car, selon nous, elles s'expriment mieux par des règles de cohérence.

Le modèle que nous considérons offre trois sortes de classes, les classes d'entités, les classes d'associations et les classes de valeurs, ainsi que deux types de sélecteurs, attributs et rôles. Une classe d'entités peut être déclarées comme étant une sous-classe d'une autre classe d'entités (cf. figure 3 ci-dessous).

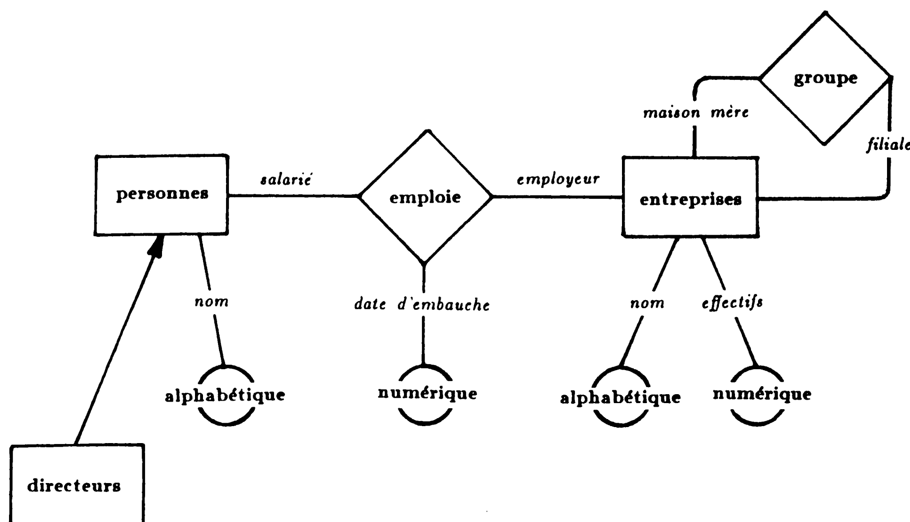


figure 3

Une classe d'associations ("**emploie**", "**groupe**") relie deux (ou plus de deux) classes d'entités ("**personnes**", "**directeurs**", "**entreprises**") qui ne sont pas nécessairement distinctes. Des sélecteurs, les rôles ("*salarié*", "*employeur*", "*filiale*", "*maison mère*"), identifient les participations des classes d'entités aux classes d'associations. D'autres sélecteurs, les attributs ("*nom*", "*date d'embauche*", "*effectifs*"), permettent d'attacher des valeurs à des entités. Dans notre exemple, l'ensemble image des deux attributs "*nom*" est la classe de valeurs "**alphabétique**". Ces deux attributs de même nom sont distingués par leurs origines, la classe d'entités "**personnes**" pour l'un, la classe d'entités "**entreprises**" pour l'autre. La classe d'entités "**directeurs**" est une sous-classe de la classe "**personnes**".

Les conditions de validité de déclarations d'un schéma sont les suivantes :

1. Deux classes différentes ont des noms différents (une classe d'entités et une classe d'associations ne peuvent par exemple pas porter le même nom).

2. L'ensemble image d'un attribut est une classe de valeur. L'ensemble image d'un rôle est une classe d'associations.

3. L'origine d'un attribut est soit une classe d'entités, soit une classe d'associations. L'origine d'un rôle est une classe d'associations.

4. Une classe d'associations est l'origine d'au moins deux rôles distincts.

5. Les sélecteurs (attributs ou rôles) de même origine doivent avoir des noms différents, mais un même nom peut désigner des sélecteurs d'origines différentes.[1]

6. La relation "sous-classe" est définie sur les classes d'entités uniquement.

7. La relation "sous-classe" est anti-réflexive et transitive (i.e. c'est un ordre strict, non nécessairement total).

8. Si $\mathbf{E_1}$ est une classe d'entités sous-classe d'une classe d'entités $\mathbf{E_2}$, alors $\mathbf{E_1}$ hérite des attributs définis sur $\mathbf{E_2}$.

9. Deux classes d'entités distinctes ayant une même sous-classe d'entités en commun ne peuvent pas avoir d'attribut de même nom.[2]

10. Une classe d'entité est l'origine d'au moins un attribut, sauf si elle est une sous-classe d'une autre classe d'entités.

Ces dix règles se traduisent aisemment en logique, si l'on considère trois prédicats unaires "entité(x)", "association(x)" et "valeur(x)" pour distinguer les différentes sortes de classes, un prédicat binaire "sous-classe(x, y)" pour exprimer qu'une classe d'entités x est une sous-classe d'une classe d'entités y, et deux predicats ternaires "attribut(x, y, z)" et "rôle(x, y, z)" pour exprimer qu'un attribut (respectivement rôle) x a pour origine la classe y et pour image la classe z. Dans ce formalisme, les termes (i.e. les variables et les constantes) sont les structures déclarées dans un schéma particulier. Les dix règles données ci-dessus s'expriment par les formules logiques suivantes :

1. $\forall x\ [\ \text{entité}(x) \Rightarrow (\neg\text{association}(x) \wedge \neg\text{valeur}(x))\ ]$
   $\forall x\ [\ \text{association}(x) \Rightarrow (\neg\text{entité}(x) \wedge \neg\text{valeur}(x))\ ]$
   $\forall x\ [\ \text{valeur}(x) \Rightarrow (\neg\text{association}(x) \wedge \neg\text{entité}(x))\ ]$

2. $\forall x \forall y \forall z\ [\ \text{attribut}(x, y, z) \Rightarrow \text{valeur}(z)\ ]$
   $\forall x \forall y \forall z\ [\ \text{rôle}(x, y, z) \Rightarrow \text{entité}(z)\ ]$

3. $\forall x \forall y \forall z\ [\ \text{rôle}(x, y, z) \Rightarrow \text{association}(y)\ ]$
   $\forall x \forall y \forall z\ [\ \text{attribut}(x, y, z) \Rightarrow (\text{entité}(y) \vee \text{association}(y))\ ]$

4. $\forall x\ [\ \text{association}(x) \Rightarrow$
   $(\exists y_1 \exists y_2 \exists z_1 \exists z_2\ [\ y_1 \neq y_2 \wedge \text{rôle}(y_1, x, z_1) \wedge \text{rôle}(y_2, x, z_2)\ ])\ ]$

5. $\forall x \forall y \forall z_1 \forall z_2\ [\ \text{attribut}(x, y, z_1) \wedge \text{attribut}(x, y, z_2) \Rightarrow z_1 = z_2\ ]$
   $\forall x \forall y \forall z_1 \forall z_2\ [\ \text{rôle}(x, y, z_1) \wedge \text{rôle}(x, y, z_2) \Rightarrow z_1 = z_2\ ]$

---

[1] Le même nom peut être donné à des sélecteurs d'origines différentes mais d'image identique ("nom" de personne et "nom" d'entreprise en figure 3).

[2] Cette règle est nécessaire pour éviter toute ambiguïté dans l'héritage d'attributs par une sous-classe.

$\forall x \forall y \forall z_1 \forall z_2 \neg[$ attribut$(x, y, z_1) \wedge$ rôle(x, y, z_2) $]$

6. $\forall x \forall y$ [sous-classe(x, y) => entité(x) $\wedge$ entité(y)]

7. $\forall x$ [$\neg$sous-classe(x, x)]
   $\forall x \forall y \forall z$ [sous-classe(x, y) $\wedge$ sous-classe(y, z) => sous-classe(x, z)]

8. $\forall x_1 \forall x_2 \forall y \forall z$ [sous-classe$(x_1, x_2) \wedge$ attribut(y, x_2, z) => attribut(y, x_1, z)]

9. $\forall x_1 \forall x_2 \forall x_3$ [$(x_1 \neq x_2 \wedge$ sous-classe$(x_3, x_1) \wedge$ sous-classe$(x_3, x_2))$ =>
   $\neg(\exists y \exists z_1 \exists z_2$ [attribut$(y, x_1, z_1) \wedge$ attribut(y, x_2, z_2)]$)$]

10. $\exists x_1$ [entité$(x_1) \wedge \neg(\exists y \exists z$ attribut$(y, x_1, z))$ => $\exists x_2$ [sous-classe(x_1, x_2)]]

Avec les mêmes prédicats, l'exemple donné en figure 3 se traduit par la liste suivante d'atomes complétement instanciés :

entité(personnes)
entité(directeurs)
entité(entreprises)

attribut(nom, personnes, alphabétique)
attribut(nom, directeurs, alphabétique)
attribut(nom, entreprises, alphabétique)
attribut(effectifs, entreprises, numérique)

sous-classe(directeurs, personnes)

association(emploie)

attribut(date d'embauche, emploie, numérique)
rôle(salarié, emploie, personnes)
rôle(employeur, emploie, entreprises)

association(groupe)

rôle(filiale, groupe, entreprises)
rôle(maison mère, groupe, entreprises)

Cette liste peut être vue comme l'extension d'une base de données relationnelle. Une modification des déclarations de structures du schéma correspond à une modification de cette base. Les structures résultantes sont valides si et seulement si les dix règles de cohérence données plus haut sont respectées. Cette vue des déclarations de structure d'un schéma comme extension d'une base de données permet d'appliquer des méthodes semblables à celles décrites dans [Blaustein 81, Decker 86, Nicolas 82] pour optimiser les vérifications de validité de déclarations de structures lors de mises à jour d'un schéma.

L'expression assertionnelle d'un modèle (dans notre exemple à l'aide des prédicats "entité", "association", "valeur", etc...) fournit un cadre à l'expression de modifications du modèle, qui correspondent à des mises à jour d'un schéma relationnel. Cette approche permet de disposer de la

souplesse du modèle relationnel dans des oprérations de types création de vues, ajout de relations, etc... au niveau des modifications de modèles.

Cette approche, implicite dans certains travaux, permet une réalisation extrèmement simple en Prolog d'un système de gestion et de vérification de la validité de déclaration de structures. Il suffit de considérer les prédicats définissant les méta-types du modèle (dans l'exemple "entité", "association", "valeur", etc...) comme des prédicats Prolog.[3] Le système de gestion de déclarations de structures que nous avons développé en Prolog [Manthey 85], est fondé sur ce principe.

La section suivante est consacrée à la validité des ensembles de règles (de cohérence ou de déduction). En fait, la validité de chaque constituant n'est pas suffisante à assurer la validité du schéma dans son ensemble. Les différents constituants doivent être compatibles entre eux. Les conditions de compatibilité sont simples : en particulier, les règles d'un schéma sont compatibles avec les déclarations de structure si et seulement si elles se rapportent aux mêmes types de structures. En d'autre termes, l'ensemble des déclarations de structures est la définition d'un langage (du premier ordre), dans lequel les régles doivent être exprimées.

3. Vérifier la Validité des Règles

3.1. Règles Valides = Règles Finiment Satisfaisables

La consistence logique des règles d'un schéma est la condition de validité la plus immédiate. En effet, d'après le théorème de complétude de Gödel [Mendelson 79], elle est équivalente à l'existence d'un modèle, ou satisfaisabilité, qui en terme de bases de données correspond à l'existence d'une extension. A un ensemble inconsistent de règles de cohérence ou de déduction aucune extension ne peut être associée: En raison du lien entre existence de modèles (satisfaisabilité) et existence d'extensions de base de données, nous utiliseront de préférence les termes de "satisfaisabilité" et "non-satisafaisabilité" au lieu de ceux, strictement équivalents, de "consistence" et "inconsistence".

Si la satisfaisabilité est une condition nécessaire, elle n'est pas suffisante en bases de données. L'extension d'une base conventionnelle est forcement un ensemble fini de faits, et correspond donc à un modèle fini des règles du schéma. Les règles de déduction d'une base déductive peuvent être supposées correspondre à des formules définies,[4] selon l'hypothèse la plus commune. Or, tout ensemble de formules définies est non seulement satisfaisable [Gallaire 84], mais ses modèles

[3]En pratique d'autres prédicats sont préférables, par exemple pour que la base relationnelle soit normalisée.

[4]L'expression "formule définie" ("definite formula") a différentes acceptions. Nous lui donnons le même sens que [Gallaire 84], i.e. celui de formule correspondant à un ensemble de clauses ayant chacune exactement un litéral positif.

minimaux sont finis. C'est la restriction aux modèles finis [Ullman 85, Reiter 78, Vieille 86] qui permet en particulier d'assurer qu'aucune interrogation de bases déductives ne peut avoir de réponse infinie. L'existence de modèles finis est donc, pour les bases déductives comme pour les bases conventionnelles, une condition de validité supplémentaire que les règles doivent vérifier. La propriété pour un ensemble de formules logiques du premier ordre d'accepter un modèle fini s'appelle la *satisfaisabilité finie*. L'intéret de la satisfaisabilité finie en bases de données n'est à notre connaissance (brièvement) mentionné que dans [Fagin 84].

La satisfaisabilité finie implique la satisfaisabilité : un ensemble de formules acceptant un modèle fini est satisfaisable. La réciproque est fausse, car il existe des (ensembles de) formules, appelés axiomes de l'infini, qui sont satisfaisables, mais dont tous les modèles sont infinis. La formule $\forall x\ [\neg p(x,x) \wedge \exists y\ [p(x,y)]] \wedge \forall u \forall v \forall w\ [\ (p(u,v) \wedge p(v,w)) \Rightarrow p(u,w)\ ]$ en est un exemple (le prédicat p peut être interprété comme l'ordre strict <, si les variables prennent leurs valeurs dans l'ensemble des entiers naturels). Reconnaître les ensembles de règles qui ne sont pas satisfaisables n'est donc pas suffisant. Il est également nécessaire de distinguer entre les axiomes de l'infini, à rejeter, et les ensembles de formules finiment satisfaisables (cf. figure 4 ci-dessous), qui seuls ont un sens dans un contexte de base de données.

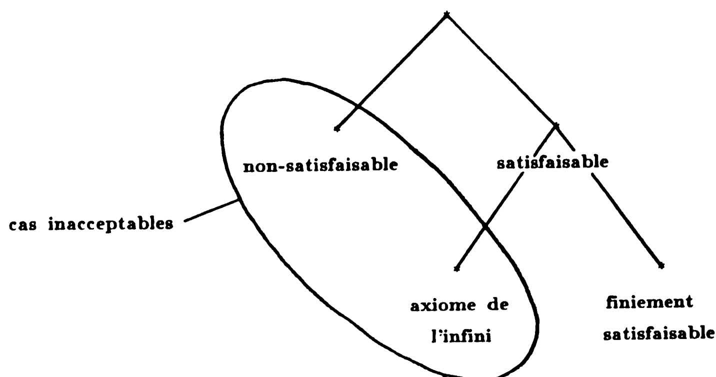


figure 4

Il existe des méthodes permettant de détecter tous les cas où un ensemble de formules logique du premier ordre n'est pas satisfaisable. Ces méthodes, appelées méthodes de réfutation, ont été étudiées intensivement depuis plus de vingt ans en preuve automatique de théorèmes. Aucune méthode de réfutation n'est assurée de s'arrêter dans tous les cas, car la non-satisfaisabilité n'est pas une propriété décidable. Elle est cependant semi-décidable, c'est à dire qu'il existe des algorithmes capables de détecter en temps fini (mais indéfini) la non-satisfaisabilité. Appliqués à des ensembles

satisfaisables, ces algorithmes peuvent, dans certains cas qui ne sont pas tous reconnaissables *a priori*, ne jamais s'arrêter. Puisque sa négation est semi-décidable, la satisfaisabilité est non seulement indécidable, mais aussi non-semi-décidable. Cela signifie qu'aucune méthode ne peut détecter, dans tous les cas, cette propriété. Il est heureux que la condition de validité des ensembles de règles des schémas de bases de données ne soit pas la satisfaisabilité en général, mais une propriété plus restrictive, la satisfaisabilité finie, qui elle est semi-décidable [Trachtenbrot 50] !

On peut chercher à restreindre par des conditions syntaxiques le type de formules logiques autorisées pour l'expression des règles de cohérence, afin de se trouver dans un cas où non-satisfaisabilité et satisfaisabilité finie sont toutes deux des propriétés décidables. La restriction des règles de déduction à des formules définies est, en fait, de ce type. Malheureusement, si de nombreuses telles classes de formules, appelées classes solvables, sont connues (voir par exemple [Dreben 79]), aucune n'est assez expressive pour des règles de cohérence. Comme il a souvent été remarqué [Gallaire 84], des formules de tous types sont nécessaires pour l'expression de règles de cohérence.

La satisfaisabilité finie étant semi-décidable, il est possible de disposer d'une méthode apte à reconnaître les ensembles de règles valides (i.e. finiment satisfaisables) d'un schéma, même en l'absence de restriction à certaines classes de formules. Pour des raisons d'efficacité autant que pour pouvoir expliquer les causes de non-validité, il est souhaitable que les cas ou le test de validité ne s'arrête pas soient aussi rares que possible. Puisque la non-satisfaisabilité peut être reconnue (par une méthode de réfutation), cet objectif peut être atteint en appliquant parallèlement au test de satisfaisabilité finie, une méthode de réfutation à l'ensemble de règles en question. Dans les cas où cet ensemble s'avère non-satisfaisable, il n'est pas nécessaire de poursuivre le test de satisfaisabilité finie. Symétriquement, la détection de la satisfaisabilité finie autorise l'arrêt du test de non-satisfaisabilité. Les seuls cas où une telle méthode pourrait ne pas s'arrêter seraient ceux des axiomes de l'infini (cf. figure 4, en page précédente). Le problème qui se pose est donc celui de la définition d'une méthode complète pour la satisfaisabilité finie, i.e. capable de détecter tous les cas de satisfaisabilité finie.

Une telle méthode simple, mais peu efficace, consiste à tenter la construction d'un modèle de cardinalité finie n, successivement pour tout entier n. A notre connaissance, aucune autre méthode n'a été proposée qui soit complète. En cherchant à définir une méthode (complète) plus efficace, il nous est apparut que les optimisations possibles consistent précisement à détecter les cas de non-satisfaisabilité. Cette constatation empirique nous a amené à étudier comment une méthode de réfutation peut servir de base à la définition d'une méthode complète pour la satisfaisabilité finie.

Nous nous proposons, dans cette section, de décrire le type d'ajouts qu'il est indispensable de faire à une méthode de réfutation pour pouvoir reconnaître les ensembles de formules finiment

satisfaisables. Comme exemple concret de méthode de réfutation nous considérons ici la résolution. Cette méthode, sur laquelle s'appuie les démonstrateurs de théorèmes les plus performants, est en effet bien connue. A titre d'exemple, une méthode originale, complète pour la satisfaisabilité finie et fondée sur la résolution est présentée. Cette méthode est discutée, et d'autres directions de recherches que nous avons explorées sont introduites.

A notre connaissance, le seul autre travail consacré à la vérification de la validité des règles d'une base de données est [Kung 85]. Il ne relève pas l'importance de la satisfaisabilité finie et propose une méthode complète en réfutation et (nécessairement) non complète pour la satisfaisabilité, étendant la méthode des tableaux de Beth et Hintikka (voir [Smullyan 68]). Les extensions proposées consistent en fait à détecter certains cas de satisfaisabilité finie, mais sans atteindre la complétude pour cette propriété.

Une connaissance intuitive du principe de résolution est suffisante à la compréhension de cette section. Il est présenté, ainsi que d'autres méthodes de réfutation, dans les ouvrages [Chang 73, Loveland 78]. Nous considérons des clauses, au lieu de formules logiques. Ce n'est pas là une restriction, car toute formule logique admet une forme clausale qui vérifie les mêmes propriétés de satisfaisabilité et de non-satisfaisabilité [Loveland 78]. Il est de plus facile de prouver que la mise sous forme clausale (par mise en forme prénexe normale conjonctive, puis skolémisation) préserve aussi la satisfaisabilité finie. Nous considérons un calcul des prédicats du premier ordre dont toutes les fonctions (d'arités non nulles) sont des fonctions de Skolem. Les variables sont représentées par des minuscules, constantes et fonctions par des majuscules. Si F est une fonction de Skolem d'arité 1, on notera F(x) par Fx si aucune ambiguïté n'est possible. F^1 étant défini comme égal à F, on défini récursivement F^n par FF^{n-1}, pour tout $n \geq 2$.

3.2. Evaluation Fonctionnelle

Puisque la non-satisfaisabilité n'est pas une propriété décidable, nous ne sommes pas assurés que la méthode de résolution (même contrainte par une stratégie) s'arrête lorsqu'appliquée à un ensemble satisfaisable de clauses. De tels cas sont cependant possibles. Nous avons établi [Bry 86] que dans ces cas l'ensemble de clauses est nécessairement finiment satisfaisable. En d'autre termes, la résolution ne s'arrête jamais lorsqu'appliquée à un axiome de l'infini. Cependant, il existe également des ensembles de clauses finiment satisfaisables à partir desquels une infinité de clauses peuvent être dérivées par résolution.

Considérons par exemple deux prédicats "marié(x, y)" et "inconnu(x, y)" destinés à exprimer respectivement que deux personnes sont mariées ou inconnues l'une de l'autre. Un "monde" où toute personne serait mariée, où chacun connaîtrait les connaissances de son conjoint et où une certaine personne "A" connaîtrait tout le monde peut être axiomatisé par les clauses suivantes :

marié(x, Fx)

inconnu(x, y) ¬inconnu(Fx, y)
¬inconnu(x, A)

Cet ensemble de clauses admet des modèles finis, par exemple un modèle de cardinalité 1, correspondant aux clauses unitaires complétement instanciés "marié(A, A)", "¬inconnu(A, A)". Cependant, pout tout entier n $\geq$ 1, les clauses

¬inconnu(F^nx, A)

sont dérivables par résolution. Aucune de ces clauses n'est une variante d'une autre d'entre elles ou d'une des clauses de l'ensemble initial. Aucune de ces clauses n'est non plus subsumée par une autre clause. Si la clause "marié(x, Fx)" est interprétée comme exprimant que toute personne est mariée à quelqu'un, la fonction F peut être interprétée comme l'application[5] "conjoint". La génération sans fin de termes fonctionnels "imbriqués" vient de l'incapacité de la résolution à identifier une personne par exemple avec son conjoint (conduisant à "FA = A" et au modèle de cardinalité 1 donné plus haut), ou bien à identifier le conjoint du conjoint d'un individu (F^2) avec cet individu (conduisant à un modèle de cardinalité 2), etc... De telles identifications de termes fonctionnels sont des évaluations de fonctions.

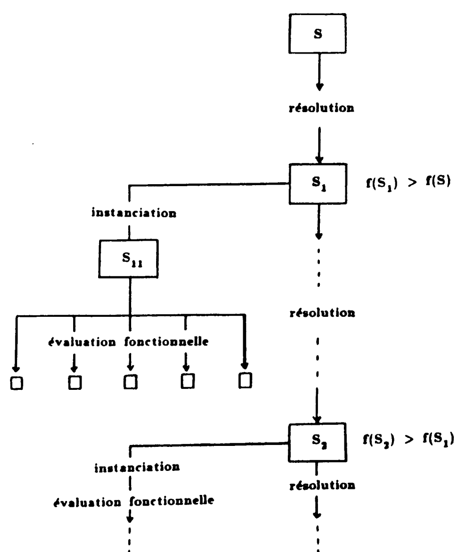
L'incapacité de la résolution à réaliser ces évaluations ne doit pas être vu comme un défaut. La génération illimitée de nouveaux termes que permet l'absence d'évaluation peut être nécessaire. Si par exemple, un ensemble de clauses ne contient qu'une seule constante, mais n'est néanmoins satisfaisable que par des modèles de cardinalités supérieure à 2, cette capacité de construire de nouveaux termes permettra aux méthodes utilisant la résolution de reconnaître un cas de satisfaisabilité finie. Dans le cas d'axiomes de l'infini, cette génération illimitée a également une signification. Considèrons par exemple l'ensemble formé des clauses résultant de l'axiome de l'infini, donné plus haut ainsi que de la clause unitaire "p(0, x)" exprimant que 0 est strictement inférieur à tout entier x. A la variable quantifiée existentiellement correspond une fonction de Skolem, soit S. Les termes S(0), S^2(0), ..., S^n(0), pour tout n $\in$ $\mathbf{N}^*$, vont être produits. Cette suite n'est autre que la suite infinie des entiers : la fonction S s'interprête comme la fonction "successeur" de l'axiomatique de Peano.

L'évaluation des termes fonctionnels ne peut être réalisée que pour les termes complétement instanciés. En effet, un terme fonctionnel non instancié tel que Fx correspond à différentes fonctions. L'instanciation des variables qui sont arguments de fonctions doit donc être réalisée, afin de permettre l'évaluation fonctionnelle.

---

[5] Les "fonctions de Skolem" sont en fait interprétées non pas par des fonctions quelconques, mais par des applications. Nous suivons l'usage et utilisons le terme de fonction.

Instanciation et évaluation fonctionnelle peuvent être ajoutés à la résolution de la manière suivante : la résolution est appliquée à l'ensemble de clauses considéré, tant qu'aucun terme de hauteur fonctionnel 2 (i.e. tel que $F^2(\ )$, ou $F(G(\ ))$) n'est produit. Lorsqu'un tel terme apparait dans une résolvente ou dans un facteur, l'ensemble des constantes figurant dans les clauses initiales est utilisé pour instancier les variables figurant dans les termes fonctionnels. Il en résulte un ensemble de termes complétement instanciés de hauteur 0 (constantes), 1, ou 2. Les termes de hauteur 2 sont alors évalués (i.e. identifiés) selon une analyse de cas par les termes de hauteur 0 ou 1. (Par exemple, deux constantes a et b, et un symbole fonctionnel unaire F permettent la formation des termes complêtement instanciés F(a) et F(b). Le terme (complètement instancié) de hauteur 2 $F^2(a)$ peut être évalué de quatre manières, consistant respectivement à le remplacer par a, b, F(a) ou F(b).) Si une de ces évaluation ne conduit pas à une contradiction, un modèle fini de l'ensemble initial a été trouvé. Sinon, il faut revenir en arrière *(backtrack)* à l'ensemble de clauses produit par la résolution avant la phase d'instanciation, et poursuivre la résolution, jusqu'à ce qu'un terme de hauteur fonctionnelle 3 soit produit, donnant lieu à un nouveau processus d'instanciation et d'évaluation fonctionnelle (cf. figure 5 ci-dessous).



S est l'ensemble initial de clauses. Les ensembles S_1, S_2, etc... sont les ensembles formés en étendant S avec les résolventes et facteurs successifs. $f(S_i)$ désigne la hauteur fonctionnelle maximum des termes apparaissant dans l'ensemble de clauses S_i. S_{11} est formé depuis S_1 par instanciation des variables apparaissant dans les termes fonctionnels.

figure 5

Cette méthode préserve la complétude en réfutation : appliquée à un ensemble non-satisfaisable de clauses, aucun modèle fini ne sera trouvé dans les "branches de gauche", et la clause vide sera dérivée par résolution dans la "branche principale".

3.3. Compactification

L'accroissement de la hauteur fonctionnelle n'est pas la seule cause de génération illimitée de clauses par résolution. Considérons trois prédicats "homme(x)", "femme(x)" et "mariés(x, y)". L'ensemble de clauses suivante décrit un "monde" où le conjoint d'un homme est une femme, et réciproquement :

homme(x) ¬marié(x, y) ¬femme(y)
femme(x) ¬marié(x, y) ¬homme(y)

Cet ensemble de clauses admet des modèles finis, par exemple un modèle de cardinalité 1, contenant un homme qui ne serait pas marié. Cependant, et malgré l'absence de fonction de Skolem, une infinité de clauses est dérivables par résolution, les clauses

homme(x_1) ¬marié(x_1, x_2) ¬marié(x_2, x_3) ... ¬marié(x_{n-1}, x_n) ¬homme(x_n)

pour tout n impair et supérieur à 3, sont produites par résolution. Aucune d'entre elles n'est subsumée ou est variante d'une autre clause. Là encore, cette génération infinie de clauses vient de l'incapacité de la résolution à identifier des termes. Considérons par exemple la clause

homme(x_1) ¬marié(x_1, x_2) ¬marié(x_2, x_3) ¬homme(x_3)

Elle exprime que si x_3 est un homme marié à x_2, et si x_2 est marié à x_1, alors x_1 est un homme. Les cas où après l'énumération d'un certain nombre d'individus on retrouve nécessairement un individu déjà rencontré, i.e. en particulier les cas correspondant à des modèles finis, ne sont pas reconnus.

Ce second problème peut être résolu de manière semblable au premier, en utilisant cette fois le nombre de variables dans une clause comme indicateur pour entrer dans une "branche de gauche". Lorsqu'une clause C est produite par résolution contenant plus de n = v(S) variables, où v(S) est le nombre maximum de variables dans l'ensemble S des clauses déjà obtenues, l'existence d'un modèle de cardinalité n est testé. Toute évaluation de C dans un tel modèle associe nécessairement la même valeur à deux de ses variables. Cette remarque est à l'origine de la méthode utilisée, appelée "compactification". La clause C est "compactifiée à hauteur n", c'est à dire qu'elle est remplacée par l'ensembles de toutes les clauses que l'on peut former depuis C en identifiant deux de ses variables. Par exemple, la compactification à hauteur 2 de la clause

homme(x_1) ¬marié(x_1, x_2) ¬marié(x_2, x_3) ¬homme(x_3)

est l'ensemble de clauses

$homme(x_1)\ \neg marié(x_1,\ x_1)\ \neg marié(x_1,\ x_3)\ \neg homme(x_3)$
$homme(x_1)\ \neg marié(x_1,\ x_2)\ \neg marié(x_2,\ x_1)\ \neg homme(x_1)$
$homme(x_1)\ \neg marié(x_1,\ x_2)\ \neg marié(x_2,\ x_2)\ \neg homme(x_2)$

La résolution est ensuite appliquée à l'ensemble résultant. D'autres compactifications sont réalisées si des clauses avec plus de n variables sont produites. Les termes fonctionnels sont finalement évalués (en accord avec l'hypothèse de cardinalité n). Si la clause vide est dérivée, le test de cardinalité n échoue : il faut alors revenir au point où l'on est entré dans ce test *(backtrack)*, et reprendre la résolution jusqu'à ce que soit dérivée une clause accroissant encore le nombre de variables. La figure 6 ci-dessous illustre ce processus.

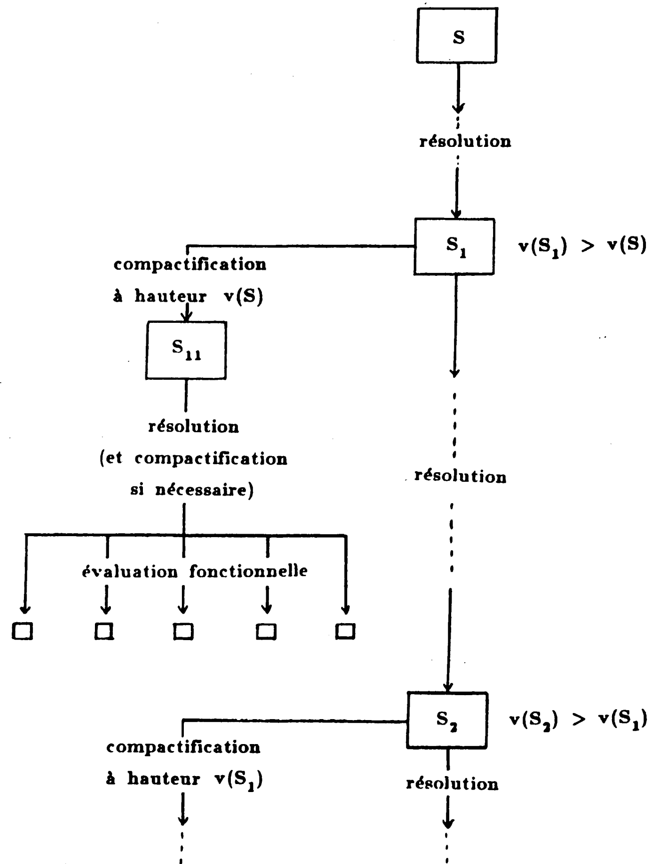


figure 6

En combinant évaluation fonctionnelle et compactification, on obtient une méthode semi-décidable et complète pour la satisfaisabilité finie qui est également complète pour la non-satisfaisabilité [Bry 85a].

3.4. Commentaires

L'extension de la résolution décrite dans les deux paragraphes précédents est compatible avec toute stratégie proposée pour améliorer la résolution, pourvu que la complétude en réfutation soit préservée. Une telle stratégie, proposée dans [Joyner 76], est spécialement interessante lorsqu'on cherche à détecter les cas de satisfaisabilité finie. Cette stratégie évite en effet la génération de termes fonctionnels de hauteur supérieure ou égale à deux, sous réserve que l'on ait mis l'ensemble initial de clause dans une certaine forme (qui préserve les propriété de satisfaisabilité et de satisfaisabilité finie) [Bry 85a]. Malheureusement, l'évaluation fonctionnelle reste indispensable, ainsi bien sur que la compactification. Cette dernière technique est extrèmement couteuse. Elle accroît en effet de manière considérable le nombre de clauses, augmentant ainsi les nombre de résolvantes et de facteurs potentiels.

L'intéret de la résolution en réfutation, et une des principales raisons pour lesquelles cette méthode a supplanté les autres approches suivies avant sa découverte, vient de ce qu'elle ne nécessite pas de phase d'instanciation. En conséquence, elle ne demande pas d'analyse de cas. Or, nous avons vu que pour détecter la satisfaisabilité finie, l'instanciation est nécessaire. Les méthodes de réfutation antérieures à la résolution ne sont donc pas nécessairement "hors de compétition" comme base à la détection de la satisfaisabilité finie.

La première de ces méthodes est la "méthode des tableaux" (décrite dans [Smullyan 68]). Dans [Kung 85], cette méthode est étendue en une méthode qui, sans être complète, détecte certains cas de satisfaisabilité finie. Une méthode complète pour la satisfaisabilité finie (et la non-satisfaisabilité), mais peu efficace, peut être définie à partir de la méthode des tableaux [Bry 85b]. La méthode de Davis et Putnam [Davis 60], qui peut être vue comme une optimisation de la méthode des tableaux, est une meilleure candidate. Nous avons défini, à partir de cette méthode, une méthode, complète pour la satisfaisabilité finie et la non-satisfaisabilité [Manthey 86]. Un grand avantage de cette méthode est qu'elle utilise la "unit-résolution", évitant ainsi les risques d'expension du nombre de variables dans les clauses produites. Elle ne nécessite donc pas de compactification. Pour cette raison, et bien qu'elle repose sur une analyse de cas, comme la méthode des tableaux et celle de Davis et Putnam, elle semble préférable à la méthode fondée sur la résolution.

Une présentation et une comparaison des deux approches, fondées respectivement sur la résolution et la méthode de Davis et Putnam, est donné dans [Bry 85c]. La seconde section du présent article reprend, en une une présentation moins abstraite, une partie du matériel de ce dernier rapport.

4. Conclusion

Dans une première partie de cet article consacré à la vérification de la validité des schémas de bases de données (indépendamment de toute extension), nous tirons profit d'une formalisation en logique des conditions de validité des déclarations de structures pour les voir comme les règles de cohérence d'une base relationnelle. Les déclarations de structures apparaissent alors comme les tuples de cette base, permettant une réalisation simple en Prolog d'un système les gérant et maintenant leur validité. Cette approche est décrite et justifiée.

Les règles de cohérence et de déduction d'un schéma sont valides, non seulement si elles sont consistentes mais si elles sont finiment satisfaisable. Dans une seconde partie, ce point est justifié. Il est ensuite montré quels sont les additions à faire à une méthode de réfutation pour l'étendre en une méthode complète pour la satisfaisabilité finie. Deux techniques, l'évaluation fonctionnelle et la compactification, permettent de reconnaître des cas de satisfaisabilité finie où de nouvelles clauses sont dérivées sans fin par la résolution. Il est montré comment coupler ces techniques à la résolution, conduisant à une méthode complète pour la satisfaisabilité finie. Le choix de la résolution est finallement discuté, et les autres approches que nous avons explorées sont citées.

Remerciements

Les auteurs expriment leur reconnaissance à Jean-Marie Nicolas pour l'aide qu'il leur a apporté par ses conseils, durant leur travaux de recherche et dans la rédaction de cette communication.

References

[Blaustein 81] Blaustein, B. T.
Enforcing Database Assertions: Techniques and Applications.
PhD thesis, Computer Science Dept., Harvard Univ., Cambridge, Mass., USA, Aug., 1981.

[Bry 85a] Bry F.
The Compactification Method.
Internal Report KB-6, E.C.R.C., Sept., 1985.

[Bry 85b] Bry, F.
Note on Consistency Checking of Databases Schemas.
Internal Report KB-4, E.C.R.C., July, 1985.

[Bry 85c] Bry, F. and Manthey, R.
Checking Consistency of Database Constraints: A Logical Basis.
Internal Report KB-16, E.C.R.C., December, 1985.

[Bry 86] Bry F.
On Resolution and Finite Satisfiability.
Internal Report, E.C.R.C., Dec., 1986.
en preparation.

[Chang 73] Chang, C.-L. and Lee R. C.-T.
Symbolic Logic and Mechanical Theorem Proving.
Academic Press, New York, 1973.

[Chen 79] Chen, P.
The Entity-Relationship Model: Toward a Unified View of Data.
ACM Trans. on Database Systems 4(1), Dec., 1979.

[Davis 60] Davis, M. and Putnam, H.
A Computing Procedure for Quantification Theory.
Journal of the ACM 7(3), July, 1960.

[Decker 86] Decker H.
Integrity Enforcement on Prolog-Based Deductive Databases.
In *Proceedings of the First International Conference on Expert Database Systems (to appear).* April 1-4 (Charleston, South Carolina), 1986.
egalement ECRC Internal Report KB-7 (septembre 1985).

[Delobel 82] Delobel, C. et Adiba, M.
Bases de Donnees et Systemes Relationnels.
Dunod, Paris, 1982.

[Dreben 79] Dreben, B. and Goldfarb, W.
The Decision Problem. Solvable Classes of Quantificational Formulas.
Addison-Wesley, Reading, Massachusetts, 01867, 1979.

[Fagin 84] Fagin, R. and Vardi M. Y.
The Theory of Data Dependencies - An Overview.
In *Proceedings of ICALP '84.* 1984.

[Gallaire 84] Gallaire, H., Minker J. and Nicolas, J.-M.
Logic and Databases: A Deductive Approach.
ACM Computing Surveys 16(2):153-185, June, 1984.

[Joyner 76] Joyner, W. H.
Resolution Strategies as Decision Procedure.
Journal of the ACM 23(3):398-417, Jul., 1976.

[Kung 85] Kung, C. H.
A Tableaux Approach for Consistency Checking.
In *Working Conference on Theoretical and Formal Aspects of Information Systems.* I.F.I.P. WG-8.1, April 16-18, 1985.
Sitges, Spain.

[Loveland 78] Loveland, D. W.
Automated Theorem Proving: a Logical Basis.
North-Holland, Amsterdam, New York, Oxford, 1978.

[Manthey 85] Manthey, R.
A Computer Aided System for DB Schema Design.
Technical Report KB-1, E.C.R.C., March, 1985.

[Manthey 86] Manthey, R.
A Generalized Davis-Putnam Procedure Able to Detect Finite Satisfiability.
Internal report KB-13, E.C.R.C., 1986.
en preparation.

[Mendelson 79] Mendelson, E.
Introduction to Mathematical Logic.
Van Nostrand, New York, 1979.

[Nicolas 82] Nicolas, J.-M.
Logic for Improving Integrity Checking in Relational Databases.
Acta Informatica 18(3):227-253, 1982.

[Reiter 78] Reiter, R.
Deductive Question-Answering on Relational Data Bases.
In Gallaire, H. and Minker, J. (editor), *Logic and Data Bases*, pages 149-177. Plenum Press, New York, 1978.
egalement, Proceedings of the Symposium on Logic and Data Bases, C.E.R.T., Toulouse, France, novembre 16-18, 1977.

[Smith 77] Smith, J.M. and Smith, D.C.P.
Database Abstractions: Aggregations and Generalizations.
ACM Trans. on Database Systems 2(2), June, 1977.

[Smullyan 68] Smullyan, R.M.
First-Order Logic.
Springer Verlag, New York, 1968.

[Trachtenbrot 50]
Trachtenbrot, B. A.
Impossibility of an Algorithm for the Decision Problem in Finite Classes.
Dokl. Acad. Nauk. SSSR 70, 1950.
en russe, traduit en anglais dans Amer. Soc. Translations, Series 2, Vol. 23, 1963, pp. 1-5.

[Ullman 85] Ullman, J.
Implementation of Logical Query Langages for Databases.
ACM Transactions on Database Systems 10(3), 1985.
egalement Report STAN-CS-84-1000 (May 1984), Stanford University.

[Vieille 86] Vieille, L.
Recursive Axioms in Deductive Databases: The Query-Subquery Approach.
In *Proceedings of the First International Conference on Expert Database Systems (to appear)*. April 1-4 (Charleston, South Carolina), 1986.
egalement ECRC Internal Report KB-10 (septembre 1985).