Les systèmes à base de connaissances

Florence Le Ber, Jean Lieber, Amedeo Napoli

Les systèmes à base de connaissances

Florence Le Ber, Jean Lieber et Amedeo Napoli

Mots-clés : systèmes à base de connaissances, représentation de connaissances, formalismes de représentation des connaissances, logiques, règles, logiques de descriptions, raisonnement, résolution de problèmes.

Résumé : Un système à base de connaissances s'appuie sur des connaissances relatives à un domaine donné pour résoudre des problèmes se posant dans ce domaine. Pour concevoir un tel système, il faut mettre au point des formalismes de représentation des connaissances et de raisonnement qui permettent de prendre en compte les niveaux syntaxique et sémantique des connaissances du domaine considéré. Dans ce chapitre, la problématique des systèmes à base de connaissances est tout d'abord introduite, puis sont détaillés des formalismes de représentation logiques (calculs des propositions et prédicats), le langage Prolog, les systèmes à base de règles et enfin les logiques de descriptions. Le chapitre se termine par un tour d'horizon des systèmes de référence et des domaines d'application dans lesquels ils opèrent.

Un jour, quelque part, nous arrivons à un carrefour où des routes se croisent et où ne figure aucune indication. Plusieurs solutions pour trouver le bon chemin — si tant est qu'un objectif ait été décidé pour ce voyage — s'offrent à nous : choisir une route au hasard, par exemple continuer tout droit coûte que coûte, quitte à retomber sur le même genre de carrefour ou encore dans un cul-de-sac (plus de route, un obstacle impossible à franchir), ou plutôt choisir une route en s'appuyant sur des connaissances disponibles sur le domaine, par exemple avec l'aide d'une carte de la région, d'un guide humain ou matériel, en repérant notre position par rapport au soleil ou avec une boussole, un sextant, etc. Beaucoup de problèmes se présentent comme celui-ci, où il s'agit de partir d'un état initial pour atteindre un état final, en passant par un ensemble d'états intermédiaires qu'il faut explorer et à partir desquels il faut faire des choix en fonction de connaissances plus ou moins disponibles et plus ou moins complètes. Le nombre de possibilités est la plupart du temps très vaste, si bien qu'il n'est pas envisageable de choisir au hasard, sous peine de ne jamais pouvoir résoudre le problème. La problématique des *systèmes à base de connaissances* repose sur de tels constats (Stefik, 1995) : comment procéder pour qu'un système informatique s'appuyant sur des connaissances puisse résoudre des problèmes dans un domaine donné ? Cela revient, pour un tel système, à faire preuve d'une *certaine forme d'intelligence*, à être capable de mettre en œuvre des stratégies et à faire des choix judicieux lorsque c'est nécessaire. Les connaissances dont il est question pour ces systèmes sont disponibles et exploitées sous forme électronique. Ainsi, la tâche principale d'un système à base de connaissances est d'exploiter des connaissances d'un domaine pour (aider à) résoudre un problème donné, d'identification, de classification, de reconnaissance, de diagnostic, de configuration, de planification, etc. Pour en revenir à notre voyage initial, de nombreux problèmes peuvent se poser en cours de route qu'il peut s'avérer fort utile de résoudre, comme identifier un animal qui vient à passer, dangereux ou pas, une plante ou un champignon qui poussent par là, comestible ou pas, soupçonner et détecter l'existence d'un point d'eau, diagnostiquer une panne de la voiture et réparer, trouver un chemin plus court ou plus pratique, ou encore plus long mais plus joli . . .

1 La notion de système à base de connaissances

1.1 La résolution de problèmes

Pour résoudre un problème, un être humain raisonne généralement sur des concepts abstraits qui modélisent les objets de l'univers du problème, en tire des conclusions qu'il interprète ensuite dans cet univers. La simulation d'un tel comportement pour un système informatique se décompose selon les étapes suivantes (cf. figure 1.1) :

- l'*abstraction* permet d'associer des structures aux éléments du domaine et du problème considéré ;
- les structures sont ensuite représentées en machine sous forme d'expressions symboliques (formules bien formées) à l'aide d'un langage de représentation des connaissances. Parmi ces expressions, certaines sont

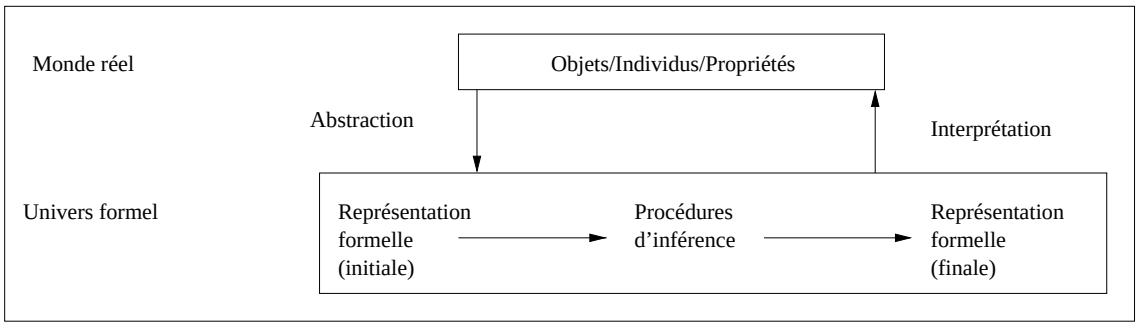

FIG. 1.1 – La représentation et la manipulation des connaissances dans un système à base de connaissances.

permanentes et constituent la mémoire à long terme ; d'autres sont temporaires et constituent la mémoire à court terme. La mémoire à long terme contient la connaissance dont dispose le système sur le domaine étudié et peut être considérée comme un modèle opérationnel et utilisable du domaine, ce qui s'appelle encore une *ontologie*. La mémoire à court terme contient les informations sur le problème particulier à résoudre ;
- les expressions symboliques sont combinées entre elles pour produire, par application de règles d'inférence, de nouvelles expressions symboliques ;
- les nouvelles expressions symboliques sont interprétées dans le cadre de l'univers du problème considéré et apportent des éléments de solution au problème courant.

Un système à base de connaissances est un programme qui s'appuie sur le schéma ci-avant pour résoudre des problèmes en utilisant explicitement des connaissances disponibles et stockées sous une forme exploitable. Dans son architecture classique, un tel système comprend : (i) une base de connaissances relatives à un domaine d'application, (ii) une base de faits contenant des faits ou données caractérisant le problème courant, (iii) un programme souvent appelé *moteur d'inférence*, qui manipule les bases, recherche les connaissances adéquates et mène un raisonnement (suite d'inférences) pour résoudre le problème courant.

1.2 Connaître, c'est décrire pour retrouver[1]

Concevoir un système à base de connaissances consiste à modéliser puis à représenter des connaissances propres à un domaine à l'aide d'un formalisme de représentation, puis à manipuler les connaissances par l'intermédiaire de règles d'inférence pour résoudre les problèmes posés (Kayser, 1997). La connaissance est alors considérée comme une *entité calculable*, ce qui pose un certain nombre de problèmes pratiques voire philosophiques (se reporter au chapitre sur la ££Connaissance££), en particulier :

- sous quelle forme doivent s'exprimer les connaissances ?
- comment un mécanisme de raisonnement peut-il exploiter au mieux un ensemble de connaissances forcément limité et comment peut-il tirer de cet ensemble toutes les connaissances implicites qu'il renferme ?
- comment les connaissances inférées influent-elles sur le comportement du système ?
- comment raisonner avec des informations incomplètes ou bruitées ?
- comment faire aboutir un raisonnement alors que l'éventail des possibilités de recherche d'une solution est virtuellement infini ?

Un langage de représentation des connaissances se caractérise par une *syntaxe* et une *sémantique* : les expressions du langage dérivent d'un procédé de construction de formules bien formées (syntaxe) et se voient associées une sémantique, sur laquelle repose la validité des inférences effectuées par le système. La sémantique est généralement donnée par une fonction d'interprétation qui met en correspondance les éléments de la représentation et les éléments d'un domaine d'interprétation (sémantique dénotationnelle). Les opérations de manipulation des connaissances — essentiellement recherche et inférence — sont dépendantes du formalisme de représentation lui-même. Les concepteurs d'une base de connaissances doivent choisir les unités de connaissances du domaine qu'il faut représenter et les coder de

[1] L'expression est de Gaston Bachelard *in Essai sur la connaissance approchée*, 1927.

façon à ce que la base construite soit un modèle opérationnel de l'univers considéré.

La plupart des formalismes de représentation des connaissances sont *déclaratifs* : les connaissances expriment ce que l'on sait, le « quoi » de l'univers étudié, par opposition à un formalisme *procédural*, où est explicité le « comment » des choses. L'aspect déclaratif repose sur une certaine indépendance vis-à-vis du contexte d'utilisation — il est possible d'appréhender les unités de connaissances isolément et chaque unité est intelligible pour et par elle-même — et vis-à-vis des traitements ; il n'est pas nécessaire de « faire tourner le système » pour savoir ce qu'il renferme et ce qu'il est censé faire. Une « échelle » des degrés de déclarativité des formalismes courants de représentation est proposée dans (Laurière, 1987).

2 L'architecture et la mécanique des systèmes de connaissances

2.1 La notion de système formel

La notion de *système formel* permet d'appréhender la forme et le fonctionnement d'un système à base de connaissances. Un *système formel* se compose d'un ensemble non vide de symboles, d'un procédé de construction de formules (à partir des symboles), d'un ensemble d'axiomes (formules prédéfinies) et d'un ensemble de *règles d'inférence* ou de *dérivation* notées $\frac{\alpha_1\alpha_2...\alpha_n}{\alpha_{n+1}}$, qui se lisent « à partir de l'ensemble de formules $\alpha_1\alpha_2...\alpha_n$ se dérive la formule α_{n+1} ». Une formule est un *théorème* si c'est un axiome ou bien si elle résulte de l'application d'une règle d'inférence à des théorèmes. Un problème général qui se pose dans le cadre d'un système formel est de savoir si une formule ϕ est un théorème, ce qui revient à rechercher l'enchaînement des dérivations (preuve ou démonstration) qui conduit des axiomes à ϕ.

Il est possible d'associer une *interprétation* — une *sémantique* — à une formule ϕ, sur la base d'un domaine d'interprétation Δ et d'une fonction d'interprétation $\mathtt{I}$. La fonction d'interprétation établit une correspondance entre les symboles de ϕ et les éléments de Δ, sur la base de laquelle se calcule la valeur que l'interprétation $\mathtt{I}$ confère à ϕ (qui se note encore $\phi^{\mathtt{I}}$ et qui peut être `vrai` ou `faux`). Ainsi, la formule ϕ est *satisfiable* s'il existe une interprétation $\mathtt{I}$ qui lui donne la valeur `vrai`. De plus, lorsque toutes les formules d'un ensemble de formules $\mathtt{F}$ sont vraies pour une interprétation $\mathtt{I}$, alors $\mathtt{I}$ est un *modèle* de $\mathtt{F}$. En particulier, à tout formalisme de représentation est associé un procédé qui permet de tester la satisfiabilité d'une formule ou de construire un modèle d'un ensemble de formules. Les tâches d'un système à base de connaissances se ramènent globalement à vérifier la satisfiabilité d'une formule ou d'un ensemble de formules et à construire alors un modèle associé.

Considérons un système formel[2] où l'ensemble des symboles est $\{\mathtt{p}, \mathtt{e}, \mathtt{u}\}$, le procédé de construction de formules est la concaténation des symboles, où le seul axiome est `upueuu` et les deux règles d'inférences sont $\frac{\mathtt{AeB}}{\mathtt{uAeBu}}$ (« si une expression de la forme `AeB` est un théorème alors `uAeBu` est un théorème ») et $\frac{\mathtt{AeB}}{\mathtt{AueuB}}$ (« si une expression de la forme `AeB` est un théorème alors `AueuB` est un théorème »). Il est possible de tester dans ce système formel si les formules `uupuueuuuu`, `uupueuuuu` et `upupueuuu` sont des théorèmes. La réponse est *oui* pour la première (deux arbres de dérivation peuvent être produits), et *non* pour les deux formules suivantes, car un théorème ne peut pas contenir un nombre impair de `u` et plus d'un `p`.

Une interprétation peut être associée à ce système formel, avec comme domaine d'interprétation $\Delta = \mathbb{N}$ (ensemble des entiers naturels), et une fonction d'interprétation `I` qui à `u` fait correspondre l'entier 1 (`I(u)` = "1"), à `e` fait correspondre l'égalité (`I(e)` = "=") et à `p` fait correspondre l'addition (`I(p)` = "+"). Dans ce cas, la formule `uupuueuuuu` s'interprète comme l'« addition » `11 + 11 = 1111` (à la façon des chiffres romains). Le fait que la formule `upupueuuu` ne soit pas un théorème alors qu'il peut lui correspondre une interprétation, en l'occurrence `1 + 1 + 1 = 111`, montre que le système formel est incomplet : il n'est pas possible de prouver tout ce qui est vrai parce qu'il manque des règles d'inférence pour le faire. Le problème de l'incomplétude d'un système formel est difficile à résoudre et se retrouve à l'identique dans un système à base de connaissances : il manque des connaissances et/ou des règles d'inférence pour résoudre certains des problèmes posés.

2.2 Les concepts et les ontologies

En représentation des connaissances, le terme *ontologie* fait référence à un modèle opérationnel utilisé pour décrire un domaine particulier du monde réel (Fensel et al., 2003 ; Staab and Studer, 2004). Dans cet ordre d'idées, les *concepts* apparaissent comme des briques de base des ontologies (Ganter et Wille, 1999) : ils possèdent une *intension* qui se définit par l'ensemble des propriétés caractéristiques, ou attributs, du concept et une *extension* qui recouvre l'ensemble des individus ou objets « instances » du concept. L'intension peut s'appréhender comme l'ensemble des condi-

[2] Cet exemple est tiré de la page Web http ://www710.univ-lyon1.fr/~fouet/DEA/main.html de J.-M. Fouet. Un système formel analogue noté « MIU » est proposé dans (Hofstadter, 1985).

tions nécessaires et suffisantes devant être vérifiées par un objet pour être instance du concept. Les intensions et les extensions sont emboîtées en sens inverse l'une de l'autre : plus un concept est général plus il recouvre d'individus et moins il recouvre de propriétés, et réciproquement.

Pratiquement, une ontologie $\mathcal{O}$ se présente comme un système formel constitué d'un ensemble de concepts et d'un ensemble de relations binaires spécifiées par des couples de concepts $(\mathtt{D}, \mathtt{R})$ de *domaines* et de *codomaines*, d'un ensemble $\mathtt{A}$ d'axiomes, d'une relation de *spécialisation* ou *subsomption* notée $\sqsubseteq$, qui est généralement réflexive, antisymétrique et transitive, qui permet d'organiser les concepts et les relations en une hiérarchie, et qui autorise les inférences. En particulier, $\mathtt{C}_1 \sqsubseteq \mathtt{C}_2$ signifie que $\mathtt{C}_1$ est un sous-concept de $\mathtt{C}_2$: l'extension de $\mathtt{C}_1$ est contenue dans celle de $\mathtt{C}_2$ tandis que l'intension de $\mathtt{C}_1$ contient celle de $\mathtt{C}_2$. Il est possible alors d'inférer qu'un individu est instance d'un certain concept ou qu'un concept partage certaines propriétés avec un autre concept. Suivant ce schéma, une base de connaissances s'appuie sur un couple constitué d'une ontologie et d'une base d'assertions ou de faits (dans lesquels interviennent les individus). Ainsi, une base de connaissances contient des unités de formes et de niveaux d'abstraction différents, comme par exemple des concepts, des instances, mais aussi des règles manipulant des faits, des stratégies (*heuristiques*) exprimant la façon de se servir des connaissances de la base.

Les ontologies ont une place d'importance croissante dans des domaines comme la gestion des connaissances, les systèmes coopératifs, l'intégration et la recherche d'information, le commerce électronique et bien sûr le Web sémantique (se reporter au chapitre sur le ££Web sémantique££). Les ontologies sont appelées à jouer là un rôle clé en établissant une terminologie commune entre les agents — logiciels et humains — qui peuvent ainsi partager la même sémantique sur les concepts et les relations manipulés. Les ontologies sont également au cœur de la gestion des connaissances, où, à l'image de la gestion de bases de données, ce ne sont plus simplement des données (syntaxiques) mais des connaissances (munies d'une syntaxe et d'une sémantique) qui sont considérées et manipulées.

Toutefois, les objets du quotidien obéissent rarement à des lois rigoureuses, comme les êtres humains et au contraire des objets mathématiques. Intégrer les différentes natures des objets du quotidien dans un formalisme de représentation pose des problèmes difficiles à résoudre comme la représentation de modalités (statut et degré de vérité des informations), la représentation de connaissances typiques et exceptionnelles, la représentation de connaissances incomplètes, évolutives, interdépendantes, etc. Ainsi, de nombreux formalismes de représentation et de raisonnement ont été mis au point pour prendre en compte les natures diverses et variées des connaissances (Haton et al., 1991). Quelques-uns de ces formalismes, parmi les plus couramment utilisés, les logiques des propositions et des prédicats, le langage Prolog, les règles de production et les logiques de descriptions, sont détaillés dans les paragraphes qui suivent. Il existe bien sûr d'autres formalismes de représentation, qu'il n'est pas possible de traiter ici faute de place, comme par exemple les graphes conceptuels (Sowa, 1984) ou les représentations de connaissances par objets (Ducournau et al, 1998). Le lecteur curieux ou intéressé peut se rapporter aux deux livres référencés.

2.3 L'ingénierie et la gestion des connaissances

L'ingénierie d'un système à base de connaissances s'appuie sur un ensemble d'opérations qui se retrouvent dans la conception de tout logiciel d'envergure :

- initialisation et modélisation : ces étapes recouvrent la spécification d'un modèle des éléments du domaine à représenter et une mise en œuvre du modèle du domaine ; des méthodologies de modélisation telles que KADS ou COMMON KADS ont été mises au point pour ces besoins (Schreiber et al., 1999) ;
- représentation et raffinement : ces étapes recouvrent la phase de représentation proprement dite, le choix d'un langage de représentation, l'implantation du modèle et le raffinement du modèle après les premières opérations de test ;
- évaluation : cette étape suit et complète l'utilisation du système à base de connaissances dans des applications et recouvre la mise en place d'environnements logiciels adaptés aux besoins spécifiques et l'évaluation du fonctionnement du système dans la pratique ;
- maintenance : les connaissances évoluent, les modes de raisonnement aussi, ce qui fait évoluer d'autant la spécification du système, qui doit être mis à jour pour garantir la cohérence et la compatibilité des connaissances, anciennes et nouvelles ;
- diffusion : les connaissances doivent être partagées et transmises si besoin est sous une forme opérationnelle : c'est là un des fondamentaux de la conception des ontologies et donc des bases de connaissances. Le fait que les connaissances soient codées à l'aide d'un langage de représentation muni

d'une syntaxe et d'une sémantique bien définies garantit que ce sont bien les mêmes éléments de connaissances qui sont envoyés et réceptionnés, au sens où ils peuvent servir à résoudre des problèmes de même nature de part et d'autre.

Du côté du monde industriel, ce sont les notions de gestion des connaissances et de mémoire d'entreprise qui émergent effectivement (Hatchuel et Weil, 1992 ; Dieng et al., 2001 ; Zacklad et Grundstein, 2001). Les problèmes à résoudre consistent pour l'essentiel à gérer les connaissances liées à l'entreprise : les recenser, les mémoriser, les utiliser, les transmettre et les faire croître. Les éléments essentiels de la mémoire d'entreprise sont le savoir-faire, l'expertise, les documents scientifiques et techniques ; les différents « agents » sont ici les personnes, les connaissances, les documents et les actions (la dynamique, le flot des informations, etc.). Un des problèmes récurrents qui se pose est celui de « trouver la bonne information » : qui sait ou peut savoir où elle se trouve, comment s'en servir et sinon comment faire.

Les principales étapes de la conception d'une mémoire d'entreprise sont calquées sur le schéma ci-dessus d'ingénierie d'un système à base de connaissances : la détection des besoins et la conception du modèle, la construction de la mémoire, l'utilisation, l'évaluation et la diffusion de la mémoire. Il faut faire ressortir ici plusieurs points particuliers : l'utilisation effective de la mémoire d'entreprise par des personnes aux statuts différents et donc aux besoins différents (décideurs et techniciens par exemple), l'accès nécessaire à des systèmes et à des bases d'informations qui sont associés ou qui dépendent de la mémoire d'entreprise, le partage et la diffusion des éléments de la mémoire sur un plan interne mais aussi externe, et enfin le « retour sur expérience » qui autorise une mise au point et un raffinement progressifs de la mémoire d'entreprise.

Ci-après nous détaillons un ensemble de formalismes de représentation parmi les plus utilisés et les plus importants, dans l'ordre les logiques des propositions et des prédicats, le langage Prolog, les règles de production et enfin les logiques de descriptions.

3 Les formalismes logiques et Prolog

Le *calcul des propositions* (CP0) correspond à la logique où les constituants des formules sont des propositions, c'est-à-dire des expressions qui prennent l'une des deux valeurs {`faux`, `vrai`} (ou {0, 1}), à l'image d'une fonction booléenne. Une proposition peut correspondre à une forme du langage comme `sujet-verbe-complément`, insécable au sens où elle forme un « tout non décomposable » : `il-fait-beau` ou `le-chat-est-noir`. Le calcul des propositions permet de formaliser des énoncés de problèmes donnés en langage naturel et une certaine partie du raisonnement humain. Les problèmes principaux qui se posent en CP0 sont (*p1*) de représenter des énoncés avec des formules du CP0, (*p2*) de savoir si une formule est toujours vraie et (*p3*) de savoir si une formule se déduit d'un ensemble de formules. Ces problèmes recouvrent les tâches qui doivent être accomplies par un système à base de connaissances (Delahaye, 1987).

Ci-dessous, nous décrivons brièvement les grands principes du CP0 et nous montrons les rapports existant avec la problématique des systèmes à base de connaissances. Une formule du CP0 ou formule propositionnelle se construit à partir d'un ensemble de symboles (ou variables) propositionnels et d'un ensemble de connecteurs booléens exprimant (entre autres) la négation ($\neg$), la conjonction ($\wedge$), la disjonction ($\vee$) et l'implication ($\rightarrow$). Une sémantique peut être associée à une formule propositionnelle ϕ par l'intermédiaire d'une interprétation ou valuation $\mathtt{I}$ qui fait correspondre à une formule ϕ une valeur de vérité dans $\{0, 1\}$, notée $\phi^{\mathtt{I}}$, une fois fixée la valeur de vérité des symboles propositionnels qui composent ϕ. La formule ϕ est alors *satisfiable* s'il existe une valuation $\mathtt{I}$ telle que $\phi^{\mathtt{I}} = 1$; une telle valuation s'appelle un *modèle* de ϕ. Les problèmes *p2* et *p3* s'expriment alors de la façon suivante : pour *p2*, une formule ϕ est toujours vraie lorsque toute valuation $\mathtt{I}$ est un modèle de ϕ ; pour *p3*, une formule ϕ est une conséquence logique d'un ensemble de formules $\mathtt{S} = \{\phi_1, \phi_2, ..., \phi_n\}$ si et seulement si chaque fois que les formules de $\mathtt{S}$ sont vraies simultanément alors ϕ est vraie, ce qui revient encore à dire que tous les modèles de $\mathtt{S}$ sont des modèles de ϕ.

Il existe plusieurs façons de prouver qu'une formule admet un modèle, en utilisant des méthodes de tableaux sémantiques ou la résolution par exemple (Gochet et Gribomont, 1991). En particulier, la règle de résolution s'écrit $\frac{\neg \mathtt{a} \vee \mathtt{b}, \mathtt{a} \vee \mathtt{c}}{\mathtt{b} \vee \mathtt{c}}$ et se lit « la formule $\mathtt{b} \vee \mathtt{c}$ se déduit à partir des formules $\neg \mathtt{a} \vee \mathtt{b}$ et $\mathtt{a} \vee \mathtt{c}$ » (les expressions sont sous forme de clause, c'est-à-dire d'une disjonction de symboles propositionnels positifs comme $\mathtt{a}$ ou négatifs comme $\neg \mathtt{a}$). De façon pratique, une formule ϕ se déduit de l'ensemble de formules $\mathtt{S}$ si et seulement si $\mathtt{S} \cup \{\neg\phi\}$ est non satisfiable. La règle de résolution se spécialise d'une part en *modus ponens*, $\frac{\neg \mathtt{a} \vee \mathtt{b}, \mathtt{a}}{\mathtt{b}}$, qui se lit « si $\mathtt{a}$ et $\mathtt{a} \rightarrow \mathtt{b}$ sont vrais alors $\mathtt{b}$ est vrai » (où $\mathtt{a} \rightarrow \mathtt{b}$ est l'abréviation de $\neg \mathtt{a} \vee \mathtt{b}$), et d'autre part en *modus tollens*, $\frac{\neg \mathtt{a} \vee \mathtt{b}, \neg \mathtt{b}}{\neg \mathtt{a}}$, qui se lit « si $\neg \mathtt{b}$ et $\mathtt{a} \rightarrow \mathtt{b}$ sont vrais alors $\neg \mathtt{a}$ est vrai ». Le *modus ponens* et le *modus tollens* synthétisent le fonctionnement d'un système à base de connaissances lors d'une résolution de problème. D'une

part, le *modus ponens* consiste à partir des faits disponibles dans une base de faits pour en déduire de nouveaux faits, jusqu'à ce que les faits relatifs à la solution du problème courant soient déduits ; le problème est alors déclaré résolu, sinon la résolution du problème est en échec. D'autre part, le *modus tollens* consiste à partir d'un but à satisfaire et à le décomposer en sous-buts jusqu'à obtenir des faits connus ; si tous les sous-buts sont satisfaits le problème est déclaré résolu, sinon la résolution du problème est en échec.

Considérons par exemple un problème de configuration de composants et la base de formules qui en découle : « un composant non principal est de couleur orange », « un composant est générateur ou n'est pas de couleur orange », « un composant actif n'est pas bidirectionnel », « un composant est bidirectionnel si et seulement si il est principal », « un composant générateur est principal et actif », « un composant principal est générateur ». La base de formules associée est : $\neg \texttt{P} \rightarrow \texttt{CO}$ qui s'écrit (1) $\texttt{P} \vee \texttt{CO}$, avec `P = principal` et `CO = couleur-orange`, (2) $\texttt{G} \vee \neg \texttt{CO}$ avec `G = générateur`, $\texttt{A} \rightarrow \neg \texttt{B}$ qui s'écrit (3) $\neg \texttt{A} \vee \neg \texttt{B}$, avec `A = actif` et `B = bidirectionnel`, $\texttt{B} \longleftrightarrow \texttt{P}$ qui se traduit par les deux formules (4.1) $\neg \texttt{B} \vee \texttt{P}$ et (4.2) $\neg \texttt{P} \vee \texttt{B}$, $\texttt{G} \rightarrow \texttt{P} \wedge \texttt{A}$ qui se traduit encore par deux formules (5.1) $\neg \texttt{G} \vee \texttt{P}$ et (5.2) $\neg \texttt{G} \vee \texttt{A}$, et enfin $\texttt{P} \rightarrow \texttt{G}$ qui s'écrit (6) $\neg \texttt{P} \vee \texttt{G}$. Il est possible de montrer que la clause vide peut se dériver avec la règle de résolution à partir de cet ensemble de formules, où un ordre d'application possible est : (1) et (2) donnent $\texttt{P} \vee \texttt{G}$ (7) ; (3) et (5.2) donnent $\neg \texttt{B} \vee \neg \texttt{G}$ (8) ; (4.2) et (5.1) donnent $\texttt{B} \vee \neg \texttt{G}$ (9) ; (8) et (9) donnent $\neg \texttt{G}$ (10) ; (7) et (10) donnent P (11) ; (6) et (11) donnent G (12), ce qui conduit à une contradiction entre (10) qui est $\neg \texttt{G}$ et (12) qui est G. Cela signifie que l'ensemble des règles de configuration donné ci-dessus n'est pas satisfiable, ce qui s'interprète encore comme le fait qu'il n'est pas possible de mettre en place une configuration de composants avec de telles règles.

Le *calcul des prédicats du premier ordre*, abrégé en CP1, étend le CP0 avec des variables (à valeurs non nécessairement booléennes), des fonctions qui retournent des valeurs (non booléennes), des prédicats qui généralisent les propositions et des quantificateurs qui déterminent la portée des variables. Par exemple, le prédicat unaire `masculin(x)` prend la valeur `vrai` chaque fois que la variable `x` désigne un individu de sexe masculin ; le prédicat binaire `est-mère-de(x,y)` prend la valeur `vrai` chaque fois que la variable `x` désigne la `mère` de l'individu désigné par la variable `y`. Le calcul des prédicats devient indispensable dès lors qu'il est nécessaire de représenter des formules où interviennent des quantifications, pour décrire par exemple des propriétés qui concernent un ou tous les individus étudiés.

En particulier, la représentation de phrases du langage naturel s'appuie sur des formes génériques universelles et existentielles, où interviennent explicitement les quantificateurs (Kleene, 1971 ; Kayser, 1997). Ainsi, dans le célèbre syllogisme *Tous les hommes sont mortels, Socrate est un homme, donc Socrate est mortel*, la première phrase illustre une quantification universelle, la deuxième une instanciation et la troisième une conséquence logique des deux précédentes. C'est à représenter ce type d'énoncés avec des formules et à prouver qu'une formule est une conséquence logique d'un ensemble de formules que s'attache le calcul des prédicats. En supposant pour simplifier que tous les prédicats sont unaires, la forme universelle positive « tous les S sont des P » se traduit par $(\forall \texttt{x})(\texttt{S}(\texttt{x}) \rightarrow \texttt{P}(\texttt{x}))$, la forme universelle négative « aucun S n'est un P » se traduit par $(\forall \texttt{x})(\texttt{S}(\texttt{x}) \rightarrow \neg \texttt{P}(\texttt{x}))$ ou encore par $\neg((\exists \texttt{x})(\texttt{S}(\texttt{x}) \wedge \texttt{P}(\texttt{x})))$, les formes existentielles positives « certains S sont des P » et négatives « certains S ne sont pas des P » se traduisent respectivement par $(\exists \texttt{x})(\texttt{S}(\texttt{x}) \wedge \texttt{P}(\texttt{x}))$ et $(\exists \texttt{x})(\texttt{S}(\texttt{x}) \wedge \neg \texttt{P}(\texttt{x}))$. Pour en revenir à l'exemple ci-dessus, les deux premières phrases du syllogisme se traduisent par les deux formules : $(\forall \texttt{x})(\texttt{homme}(\texttt{x}) \rightarrow \texttt{mortel}(\texttt{x}))$, $\texttt{homme}(\texttt{Socrate})$, qui se transforment sous forme de clauses en $\neg \texttt{homme}(\texttt{x}) \vee \texttt{mortel}(\texttt{x})$, $\texttt{homme}(\texttt{Socrate})$. En appliquant la règle de résolution — comme pour le CP0 — moyennant la substitution de la variable `x` par `Socrate`, il est possible de déduire que $\texttt{mortel}(\texttt{Socrate})$ est une conséquence logique des deux premières phrases. De plus, des relations complexes comme « Tous les (certains) éléments de type S sont en relation (R) avec tous les (certains) éléments de type P » peuvent également être représentées. Par exemple, l'expression « toutes les femmes aiment tous les hommes » se traduit par $(\forall \texttt{x})(\forall \texttt{y})(\texttt{femme}(\texttt{x}) \wedge \texttt{homme}(\texttt{y}) \rightarrow \texttt{aime}(\texttt{x}, \texttt{y}))$.

Par rapport aux systèmes à base de connaissances, le CP1 permet l'utilisation de variables et donc de traiter une catégorie de problèmes plus vaste que ne le permet le seul CP0. En contrepartie, la modélisation des énoncés et le raisonnement s'avèrent plus complexes. Du point de vue pratique, l'utilisation du CP1 pour concevoir un système à base de connaissances repose le plus souvent sur le langage de programmation logique Prolog, qui adopte les principes mêmes du CP1 et qui a été mis au point dans les années soixante-dix (Colmerauer et al., 1983). Les formules du langage Prolog se composent de constantes, de variables, de fonctions, de prédicats — où certaines fonctions et certains prédicats sont prédéfinis — et où sont également disponibles des structures de don-

nées comme la liste par exemple. Trois formules génériques permettent de modéliser des règles, des faits et des questions :

Une règle se présente sous la forme $\mathtt{A(x)} \leftarrow \mathtt{B_1(x), B_2(x), ..., B_n(x)}$ et se lit A(x) si $\mathtt{B_1(x)}$ et $\mathtt{B_2(x)}$ et ... $\mathtt{B_n(x)}$ (les prédicats sont supposés unaires pour simplifier, A(x) s'appelle la tête de la règle et la conjonction des $\mathtt{B_i(x)}$ le corps de la règle). Par exemple la règle $\mathtt{repas(e,p,d)} \leftarrow \mathtt{horsDoeuvre(e), plat(p), dessert(d)}$ représente la règle «un repas est composé d'un hors d'œuvre, d'un plat et d'un dessert», où e, p et d, sont des variables. Un fait représente un élément de connaissance toujours vrai (une règle sans condition). Un fait A(x) se note $\mathtt{A(x)} \leftarrow$. Par exemple, $\mathtt{horsDoeuvre(carotte)} \leftarrow$, $\mathtt{plat(couscous)} \leftarrow$ et $\mathtt{dessert(pomme)} \leftarrow$ sont des faits. Une question se note $?\,\mathtt{B_1(x), B_2(x), ..., B_n(x)}$ et se compose d'une conjonction de prédicats dont il faut établir la vérité, plus précisément rechercher les valeurs des variables qui satisfont simultanément chaque prédicat $\mathtt{B_i}$. Par exemple, les questions $?\,\mathtt{repas(e,p,d)}$ et $?\,\mathtt{repas(e, couscous, tarte)}$ s'interprètent respectivement comme «pour quelles valeurs de e, p et d, le prédicat repas(e,p,d) est-il satisfait ?» et «pour quelles valeurs de e le prédicat $\mathtt{repas(e, couscous, tarte)}$ est-il satisfait (deux variables ont des valeurs fixées)».

Un programme Prolog se présente donc comme une suite de faits et de règles, qui correspondent à la base de faits et à la base de connaissances. L'utilisation d'un système Prolog consiste à poser des questions et à obtenir des réponses par l'intermédiaire d'une instanciation des variables qui figurent dans les questions. Pour satisfaire la question $?\,\mathtt{q(x)}$, le système Prolog recherche une formule (fait ou règle) de tête q(.) avec laquelle le prédicat de la question peut s'apparier, puis il retourne la substitution des variables qui autorise l'appariement. Par exemple, $?\,\mathtt{plat(p)}$ peut s'apparier avec plat(couscous), ce qui produit la réponse $\{\mathtt{p = couscous}\}$. Lorsque la tête d'une règle s'apparie avec une question, la tête *s'efface* et les prédicats du corps de la clause deviennent à leur tour autant de questions, dans lesquels les substitutions de variables effectuées lors de l'appariement se sont propagées. Le processus se continue jusqu'à ce que tous les faits et toutes les règles aient été traités. Par exemple, la question $?\,\mathtt{repas(e,p,d)}$ donne naissance à trois sous-questions qui sont $?\,\mathtt{horsDoeuvre(e)}$, $?\,\mathtt{plat(p)}$ et $?\,\mathtt{dessert(d)}$, qui donnent la réponse $\{\mathtt{e = carotte, p = couscous, d = pomme}\}$. Le processus est en échec si un appariement n'est pas possible et qu'une tête de règle ne peut pas s'effacer (le système effectue un retour-arrière si nécessaire), sinon le processus retourne la suite des substitutions associées à tous les appariements réalisés.

Le langage Prolog a eu une importance assez considérable dans le développement de systèmes à base de connaissances et reste encore et toujours un langage d'actualité pour l'ingénierie des connaissances. Des expériences ont également été menées pour coupler Prolog avec des langages de contraintes, ce qui a accru le potentiel du langage en termes de conception de systèmes à base de connaissances (Van Hentenryck, 1989).

4 Les systèmes à base de règles

Les *systèmes à base de règles (de production)*, plus connus sous le nom de *systèmes experts* (Hayes-Roth et al., 1983 ; Buchanan et Shortliffe, 1984 ; Delahaye, 1987 ; Laurière, 1987), constituent un moyen privilégié de coder des connaissances d'experts. En effet, il s'avère qu'une bonne partie des éléments d'expertise courants et utiles en résolution de problèmes, par exemple pour une analyse, un diagnostic, une reconnaissance, une classification, s'exprime sous la forme de règles de production. Fondamentalement, les inférences réalisées par les systèmes à base de règles sont régies par les principes de la logique, mais c'est le recueil d'expertise qui fait ici la différence. Une règle de production se compose d'une partie gauche — la prémisse — et d'une partie droite — la conclusion — et se présente sous la forme *si condition(s) alors conclusion(s)*. Le langage de description des règles peut être plus ou moins complexe, comporter des conjonctions, des disjonctions, des négations, des variables, etc.

Un système à base de règles se compose de trois modules séparés et interdépendants (cette séparation permet de respecter le caractère déclaratif de la représentation). La base de règles contient les connaissances du domaine mémorisées sous forme de règles de production. Les règles sont intelligibles pour et par elles-mêmes, et elles sont indépendantes les unes des autres (aucune règle de production ne fait appel à une autre règle). La base de faits contient les faits relatifs au problème courant à résoudre. Le moteur d'inférence permet d'enchaîner et de contrôler les cycles d'applications des règles, en partant des faits pour atteindre la solution du problème courant. Le cycle d'un moteur d'inférence est à trois temps : sélection des règles applicables, choix d'une règle et déclenchement. La figure 1.2 montre un exemple de base de règles relatives à des déplacements de personnes et une base de faits associée. Les premières règles (de (r1) à (r4)) traitent le choix du moyen de déplacement en fonction de la distance et les autres règles sont relatives au budget pour le déplacement. La base de faits décrit le problème particulier d'un déplacement de 456 km pour un universitaire disposant d'un certain budget.

```
(r1) Si distance ≤ 2 km alors aller-à-pied
(r2) Si 2 km < distance ≤ 20 km alors aller-à-vélo
(r3) Si 20 km < distance alors aller-en-train
(r4) Si 400 km < distance et budget-avion alors aller-en-avion
(r5) Si aller-en-train ou aller-en-avion alors besoin-billet
(r6) Si universitaire et budget alors ordre-mission
(r7) Si ordre-mission et besoin-billet alors billet-prépayé
(r8) Si universitaire et distance ≤ 1000 km alors ¬ budget-avion
(r9) Si besoin-billet et ¬ ordre-mission alors payer-billet

Base de faits = {universitaire, distance = 456 km, budget}
```

FIG. 1.2 – Un exemple de base de règles relative à des déplacements et une base de faits associée.

Étant donné une base de faits et une base de règles, une règle `R` est applicable si ses conditions sont satisfaites, autrement dit si la base de faits contient les informations nécessaires pour valider les prémisses de la règle `R`. Dans l'exemple ci-dessus, les règles `(r3)`, `(r6)` et `(r8)` sont applicables. Le choix d'une règle se fait parmi les règles sélectionnées sur la base de certains critères, car plusieurs règles peuvent être en compétition lors de ce choix (d'où le nom d'ensemble des conflits donné quelquefois à l'ensemble des règles applicables). Des *métarègles* (ou des méta-connaissances (Pitrat, 1990)) peuvent être utilisées pour résoudre les conflits, c'est-à-dire aider à choisir une règle à appliquer parmi plusieurs règles possibles, comme par exemple *MR1* « N'appliquer qu'une seule fois une règle sur les mêmes données » ; *MR2* « Ne pas appliquer une règle qui conclut sur des faits déjà connus » (ces deux métarègles permettent également d'éviter les boucles infinies) ; *MR3* « Appliquer la règle de rang le plus bas si aucune des deux métarègles *MR1* et *MR2* ne s'applique ». Une fois une règle choisie, elle est appliquée et les éléments de sa conclusion sont ajoutés à la base de faits. Par exemple, ici, la métarègle *MR3* s'applique, la règle `(r3)` est choisie et le fait `aller-en-train` est validé, ce qui permet la sélection future de la règle `(r5)`, et ainsi de suite.

En ce qui concerne la stratégie d'application des règles, le *chaînage avant* est dirigé par les données et se distingue du *chaînage arrière* qui est dirigé par les buts. Pour le chaînage avant, étant donné une base des faits disponibles et un but à résoudre, le moteur d'inférence enchaîne les cycles en sélectionnant les règles dont les prémisses sont satisfaites (*modus ponens*). Les nouveaux faits sont insérés dans la base de faits et le moteur s'arrête avec succès lorsque la base contient le but ou en échec lorsque plus aucune règle n'est applicable et que le but n'est toujours pas atteint. Par exemple, une suite d'applications de règles peut être la suivante : `(r3),(r5),(r6),(r7),(r8)`, ce qui exprime que le déplacement de 456 km pour notre universitaire peut se faire en train avec un billet prépayé. Pour le chaînage arrière, le moteur d'inférence recherche les règles qui concluent sur le but à résoudre : les prémisses des règles sélectionnées constituent autant de nouveaux sous-buts à résoudre (à la façon du *modus tollens*, il s'agit de « remonter » des conclusions vers les prémisses). Soit un sous-but est dans la base de faits, soit il faut le prouver à son tour en recherchant des règles qui concluent sur ce sous-but. Le moteur s'arrête avec succès lorsque l'ensemble des sous-buts à résoudre est vide et en échec lorsqu'il existe un sous-but qui ne peut être démontré. De cette façon, en partant de la même base de faits et d'un but comme `aller-en-avion`, le système va sélectionner la règle `(r4)`. La première condition est satisfaite (`400 km < distance`) et il faut fournir des informations supplémentaires au système pour qu'il puisse aller plus loin, ici savoir si un budget avion existe pour le déplacement, et ainsi de suite.

Une analyse des caractéristiques générales des systèmes à base de règles montre que cette famille de systèmes est plutôt bien adaptée à la résolution de problèmes de classification, de reconnaissance, de diagnostic, de configuration, pour lesquels il n'est pas nécessaire de manipuler des structures et de construire une solution de toutes pièces. C'est justement pour faciliter la représentation et la manipulation de structures qu'ont été développés les formalismes de représentation par objets (Ducournau et al., 1998) et les logiques de descriptions dont il est question ci-après.

5 Les logiques de descriptions

Le formalisme des logiques de descriptions joue actuellement un rôle fondamental pour concevoir des systèmes à base de connaissances et des ontologies pour le Web sémantique (Baader

et al., 2003 ; Fensel et al., 2003 ; Staab et Studer, 2004). Dans une logique de descriptions, le concept est l'élément fondamental des formules ; il représente un ensemble d'individus — l'extension du concept — et il est muni de *rôles*, qui représentent des relations binaires entre les individus et qui composent l'intension du concept. Un concept et un rôle possèdent une description structurée, élaborée à partir d'un ensemble de *constructeurs*. Une sémantique extensionnelle est associée à chaque description de concept et de rôle par l'intermédiaire d'une interprétation ensembliste. En particulier, un concept `C` est satisfiable si et seulement si son interprétation ensembliste — autrement dit son extension — n'est pas vide. La relation de *subsomption* permet d'organiser concepts et rôles suivant leur niveau de généralité : un concept `D` *subsume* un concept `C` si `D` est plus général que `C` au sens où l'extension de `D` contient l'extension de `C` dans toute interprétation. Les connaissances sont prises en compte selon deux niveaux : la représentation des concepts relève du niveau ontologique, tandis que la représentation des individus et les assertions dans lesquelles ils interviennent relèvent du niveau assertionnel ou factuel. Les opérations qui sont à la base du raisonnement sont la satisfiabilité d'un concept ou d'une base de connaissances (la recherche d'un modèle), la détection de relations de subsomption (classification de concepts) et d'instanciation (classification d'instances).

Les langages de la famille des logiques de descriptions se distinguent par l'ensemble des constructeurs qu'ils autorisent. Les constructeurs les plus courants pour les concepts sont la conjonction ($\mathtt{C} \sqcap \mathtt{D}$, où `C` et `D` désignent des concepts), la disjonction ($\mathtt{C} \sqcup \mathtt{D}$), la négation ($\neg\mathtt{C}$), la quantification universelle ($\forall \mathtt{r}.\mathtt{C}$, où `r` désigne un rôle et `C` le codomaine du rôle), la quantification existentielle ($\exists \mathtt{r}.\mathtt{C}$), les restrictions sur la cardinalité des rôles sans codomaine ($\geq \mathtt{n}\,\mathtt{r}$ et $\leq \mathtt{n}\,\mathtt{r}$) et avec codomaine ($\geq \mathtt{n}\,\mathtt{r}.\mathtt{C}$ et $\leq \mathtt{n}\,\mathtt{r}.\mathtt{C}$). La famille de logique $\mathcal{AL}$ se caractérise à partir de l'ensemble de constructeurs suivants : $\mathcal{ALC} = \{\mathtt{A}, \mathtt{r}, \top, \bot, \mathtt{C} \sqcap \mathtt{D}, \mathtt{C} \sqcup \mathtt{D}, \neg\mathtt{C}, \forall \mathtt{r}.\mathtt{C}, \exists \mathtt{r}.\mathtt{C}\}$, où `A` dénote un nom de concept, `r` un nom de rôle, $\top$ le concept le plus général et $\bot$ le concept le plus spécifique. Peuvent venir s'ajouter à $\mathcal{ALC}$ des constructeurs portant sur les rôles qui permettent la définition d'une hiérarchie de rôles, les rôles inverses, la transitivité sur les rôles, la conjonction, disjonction et négation de rôles, etc.

Considérons l'exemple suivant, tiré de (Nebel, 1990) : *(1)* un homme et une femme sont des personnes ; *(2)* aucune femme n'est un homme et vice-versa ; *(3)* une équipe est définie comme un ensemble ayant au moins deux membres qui sont tous des personnes ; *(4)* une petite équipe est définie comme une équipe ayant au plus cinq membres ; *(5)* une équipe moderne est définie comme une équipe ayant au plus quatre membres, ayant au moins un chef et dont tous les chefs sont des femmes. Cet ensemble d'énoncés peut se représenter comme indiqué à la figure 1.3. Une introduction de concept (ou de rôle) se ramène à déclarer une relation de subsomption ($\sqsubseteq$) entre le nouveau concept et des concepts préexistants. Un concept peut être primitif, comme `Ensemble` et `Personne`, et introduit par référence au concept le plus général ($\top$) ou à d'autres concepts primitifs, comme `Homme` et `Femme`. Un concept peut être défini, comme **Équipe**, `Petite-équipe` et `Équipe-moderne`, avec une définition donnée par l'intermédiaire de $\equiv$: $\mathtt{C} \equiv \mathtt{D}$ signifie alors que $\mathtt{C} \sqsubseteq \mathtt{D}$ et $\mathtt{D} \sqsubseteq \mathtt{C}$. Une façon de spécifier l'incompatibilité entre les concepts `Homme` et `Femme` revient à déclarer qu'ils ne peuvent pas avoir de sous-concept commun différent de $\bot$. Comme un concept, un rôle peut être primitif ou défini. Ici, les deux rôles sont primitifs et `aPourMembre` est plus général que `aPourChef`. Une base de connaissances se compose alors d'un ensemble d'introductions de concepts et de rôles, qui s'organisent en une hiérarchie grâce à la relation de subsomption. Ainsi, dans l'exemple, la hiérarchie des concepts définis est $\mathtt{Équipe\text{-}moderne} \sqsubseteq \mathtt{Petite\text{-}équipe} \sqsubseteq \mathtt{Équipe}$. Un ensemble d'assertions composées d'instanciations de concepts ou de rôles peut venir compléter la base de connaissances (et correspondre à une base de faits).

Les logiques de descriptions offrent de nombreuses constructions pour concevoir des systèmes à base de connaissances opérationnels et efficaces, mais des problèmes de représentation subsistent, parce que difficiles à résoudre si l'on veut rester en parfaite adéquation avec le formalisme de ces logiques (par exemple les relations entre rôles, l'absence de procédures, de fonctions, de métaniveau, etc.). Toutefois, pour l'heure, les logiques de descriptions ont fourni la base du langage de représentation d'ontologies pour le Web sémantique OWL. Le langage OWL possède trois dialectes, OWL FULL, OWL DL et OWL LITE, qui se distinguent par l'ensemble des constructeurs utilisés (Staab et Studer, 2004). En particulier, OWL DL repose sur la logique de descriptions $\mathcal{SHOIN}(\mathcal{D})$, qui regroupe les constructeurs de $\mathcal{ALC}$ ($\mathcal{S}$), $\mathcal{H}$ pour l'existence d'une hiérarchie de rôles, $\mathcal{O}$ pour les types énumérés (ensembles d'instances), $\mathcal{I}$ pour les rôles inverses, auxquels s'ajoute la transitivité des rôles, $\mathcal{N}$ pour la cardinalité sans codomaine sur les rôles et enfin ($\mathcal{D}$) pour l'existence de types de données.

$\texttt{Personne} \sqsubseteq \top$
$\texttt{Ensemble} \sqsubseteq \top$
$\texttt{Homme} \sqsubseteq \texttt{Personne}$
$\texttt{Femme} \sqsubseteq \texttt{Personne}$
$\texttt{Homme} \sqcap \texttt{Femme} \sqsubseteq \bot$
$\texttt{aPourChef} \sqsubseteq \texttt{aPourMembre}$
$\texttt{Équipe} \equiv \texttt{Ensemble} \sqcap \forall \texttt{aPourMembre}.\texttt{Personne} \sqcap (\geq 2\, \texttt{aPourMembre})$
$\texttt{Petite-équipe} \equiv \texttt{Équipe} \sqcap (\leq 5\, \texttt{aPourMembre})$
$\texttt{Équipe-moderne} \equiv \texttt{Équipe} \sqcap (\leq 4\, \texttt{aPourMembre}) \sqcap (\geq 1\, \texttt{aPourChef}) \sqcap \forall \texttt{aPourChef}.\texttt{Femme}$

FIG. 1.3 – Une représentation possible en logique de descriptions de l'énoncé sur les équipes et leurs propriétés.

6 Histoires, applications et enjeux actuels

L'histoire des systèmes à base de connaissances n'est pas si longue mais elle a vu naître un bon nombre de systèmes dont certains ont été et sont encore des modèles du genre dans leurs spécialités (Hayes-Roth et al., 1983 ; Marcus, 1988 ; Puppe, 1993). Parmi ces systèmes, beaucoup sont restés des prototypes et ont été construits plus pour valider des idées et des résultats de recherche que pour être véritablement opérationnels dans le monde industriel. Parmi les tâches effectuées par ces systèmes, figurent, en reprenant la division de (Stefik, 1995), la classification, la configuration et la diagnostic. Les domaines d'applications sont très variés pour les trois tâches, avec, aux premiers rangs, la biologie, la chimie, la géologie, la médecine, mais aussi l'agronomie, l'environnement, le marketing, les mathématiques, la sidérurgie, etc.

La classification recouvre l'identification d'un objet ou phénomène quelconque en tant que membre d'une classe connue. Pour cette tâche, les systèmes qui ont fortement marqué leur époque sont MYCIN, DENDRAL et PROSPECTOR (Buchanan et Shortliffe, 1984). Le premier système, le plus célèbre, travaille dans le cadre des maladies infectieuses et met en relation une maladie infectieuse à un ensemble de symptômes. Le système MYCIN a fait couler beaucoup d'encre et a été à l'origine de très nombreux travaux de recherches et d'un engouement assez spectaculaire dans le domaine des systèmes à base de connaissances, ce qui, paradoxalement, a relativement contribué à desservir la thématique par la suite. Les tâches réalisées par les autres systèmes sont pour DENDRAL l'élucidation de structures moléculaires en spectrométrie de masse (chimie) et pour PROSPECTOR l'identification de la nature des terrains en vue de leur prospection (géologie).

La configuration recouvre l'arrangement de parties ou de composants en satisfaisant un ensemble de contraintes donné. Le système R1 et son successeur XCON permettent de configurer les composants d'un ordinateur et ont été les premiers d'une lignée de systèmes de configuration d'ordinateurs. Dans le système MYCIN comme dans d'autres systèmes de classification ou de diagnostic, il existe un module de configuration, qui consiste en l'occurrence à mettre une thérapie en place par rapport à un diagnostic effectué. Proche de la configuration se trouve la conception qui consiste à construire une structure à partir d'éléments de base. Dans cette catégorie peuvent se placer des systèmes dits de CAO ou « conception assistée par ordinateur ». L'architecture de ces systèmes est souvent procédurale et pas toujours conforme à l'architecture déclarative introduite ci-avant. En particulier, citons le système MOLGEN de planification d'expériences génétiques en biologie, qui met en œuvre un module de satisfaction de contraintes, et les systèmes de planification de synthèses en chimie organique LHASA et RESYN, ce dernier s'appuyant sur un formalisme de représentation par objets (Ducournau et al., 1998).

La recherche d'un diagnostic consiste à observer un dysfonctionnement et à en déterminer les causes, que ce soit pour un être vivant, une personne, un animal, une plante, etc. ou un artefact de toute nature comme une voiture, un circuit électrique, etc. La recherche des causes d'un dysfonctionnement est à la base du raisonnement par abduction. Les systèmes de diagnostic sont très nombreux parmi les systèmes à base de connaissances et il n'est pas étonnant de retrouver aux premiers rangs des systèmes dont le champ d'activité est la médecine (Buchanan et Shortliffe, 1984 ; Clancey et Shortliffe, 1984), comme par exemple la famille des systèmes INTERNIST spécialisés dans le diagnostic de maladies cliniques (hépatites, infections, fièvres, etc.), le système CASNET/GLAUCOMA pour le diagnostic et le traitement du glaucome et enfin, dans un autre domaine, le système SOPHIE-III sur la compréhension de pannes de circuits électriques.

Pour terminer ce bref tour d'horizon, il faut citer la démarche originale du système CYC, qui se veut un système à base de connaissances universel, capable de raisonner sur des problèmes de la vie quotidienne (Lenat et Guha, 1990). Une telle entreprise est de longue haleine et le système CYC risque d'être en perpétuelle construction, mais n'est-ce pas le propre de l'homme lui-même ?

Dans le contexte actuel, plusieurs défis et enjeux se posent pour les systèmes à base de connaissances, surtout par rapport à la problématique du Web sémantique (Fensel et al., 2003). La diffusion des connaissances est essentielle et ne peut pas se faire sous n'importe quelle forme : ce sont les ontologies qui vont donner une forme et un sens aux connaissances représentées, qui vont servir de véhicules et qui vont permettre à des agents de tirer parti des connaissances pour résoudre des problèmes, manipuler des connaissances distribuées, rechercher et exploiter des documents en fonction de leur contenu, assurer l'interopérabilité entre différents services, etc. Pour l'heure, il est acquis que la structure et la sémantique des documents – informations, connaissances – doit être diffusée par l'intermédiaire de langages de représentation comme OWL, compatibles avec les standards du Web que sont devenus XML et RDF(S), pour la description des documents et des ressources (Staab et Studer, 2004). Il est nécessaire pour les systèmes à base de connaissances d'avoir une syntaxe et une sémantique clairement définies, et de montrer des possibilités de représentation et de raisonnement réelles et efficaces. Est-ce qu'un « super système à base de connaissances universel », dans la lignée de CYC, verra bientôt le jour, avec un ensemble de dimensions spécifiques, une pour décrire les objets du monde, une pour les documents, une pour les structures de données, une pour extraire des connaissances, une pour les traitements et les services, une pour les raisonnements de toutes natures ? Dans cette perspective de « gratte-ciel des connaissances », chacun des étages a son rôle à jouer et doit collaborer avec les autres — via l'ascenseur ou l'escalier — pour résoudre les problèmes posés par les visiteurs...

Bibliographie

Baader et al. (2003), F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, Cambridge, UK, 2003.

Buchanan et Shortliffe (1984), B.G. Buchanan and E.H. Shortliffe, editors. *Rule-Based Expert Systems : The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading (MA), USA, 1984.

Clancey et Shortliffe (1984), W.J. Clancey and E.H. Shortliffe, editors. *Readings in Medical Artificial Intelligence : The First Decade*. Addison-Wesley, Reading (MA), USA, 1984.

Colmerauer et al., (1983), A. Colmerauer, H. Kanoui, et M. Van Caneghem. PROLOG, bases théoriques et développements actuels. *Techniques et Sciences Informatiques*, 2(4) :271–311, 1983.

Delahaye (1987), J.-P. Delahaye. *Systèmes experts : organisation et programmation des bases de connaissance en calcul propositionnel*. Eyrolles, Paris, 1987.

Dieng et al., (2001), R. Dieng, O. Corby, A. Giboin, J. Golebiowska, N. Matta, et M. Ribiere. *Méthodes et outils pour la gestion des connaissances*. Dunod, Paris, 2001.

Ducournau et al., (1998), R. Ducournau, J. Euzenat, G. Masini, et A. Napoli, éditeurs. *Langages et modèles à objets — État des recherches et perspectives*. Collection Didactique D-019. INRIA, Le Chesnay, 1998.

Fensel et al., (2003) D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors. *Spinning the Semantic Web*. The MIT Press, Cambridge, Massachusetts, 2003.

Ganter et Wille (1999), B. Ganter and R. Wille. *Formal Concept Analysis*. Springer, Berlin, 1999.

Gochet et Gribomont (1991) P. Gochet et P. Gribomont. *Logique*, volume 1. Hermès, Paris, 1991.

Hatchuel et Weil (1992), A. Hatchuel et B. Weil. *L'expert et le système*. Economica, Paris, 1992.

Haton et al., (1991), J.-P. Haton, N. Bouzid, F. Charpillet, M.-C. Haton, B. Lâasri, H. Lâasri, P. Marquis, T. Mondot, et A. Napoli. *Le raisonnement en intelligence artificielle*. InterEditions, Paris, 1991.

Hayes-Roth et al., (1983), F. Hayes-Roth, D. A. Waterman, and D. Lenat. *Building Expert Systems*. Addison-Wesley, Reading (MA), USA, 1983.

Hofstadter (1985), D. Hofstadter. *Gödel, Escher, Bach, les brins d'une guirlande éternelle*. InterEditions, Paris, 1985.

Kayser (1997), D. Kayser. *La représentation des connaissances*. Hermès, Paris, 1997.

Kleene (1971), S.C. Kleene. *Logique mathématique*. Collection U. Armand Colin, Paris, 1971.

Laurière (1987), J.L. Laurière. *Intelligence Artificielle. Tome 1 : résolution de problèmes par l'Homme et la machine*. Eyrolles, Paris, 1987.

Lenat et Guha (1990), D.B. Lenat and R.V. Guha. *Building Large Knowledge-Based Systems*. Addison Wesley, Reading, Massachusetts, 1990.

Marcus (1988), S. Marcus, editor. *Automatic Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers, Boston, 1988.

Nebel (1990), B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Lecture Notes in Artificial Intelligence 422. Springer-Verlag, Berlin, 1990.

Pitrat (1990), J. Pitrat. *Métaconnaissance*. Hermès, Paris, 1990.

Puppe (1993), F. Puppe. *Systematic Introduction to Expert Systems*. Springer-Verlag, Berlin, 1993.

Schreiber et al., (1999), G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. van de Velde, and B. Wielinga. *Knowledge Engineering and Management : the CommonKADS Methodoloy*. The MIT Press, Cambridge, MA, 1999.
Sowa (1984), J.F. Sowa. *Conceptual Structures : Information Processing in Mind and Machine*. Addison Wesley, Reading, Massachusetts, 1984.
Staab et Studer (2004), S. Staab and R. Studer, editors. *Handbook on Ontologies*. Springer, Berlin, 2004.
Stefik (1995), M. Stefik. *Introduction to Knowledge Systems*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1995.
Van Hentenryck (1989), P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge (MA), USA, 1989.
Zacklad et Grundstein (2001), M. Zacklad et M. Grundstein, editors. *Management des connaissances (modèles d'entreprise et applications)*. Hermès, Paris, 2001.

Index