# Modèle ARCO : Apprentissages multiples et Raisonnement réflexif sur des Connaissances hOmogènes

Philippe Caillou

UNIVERSITE PARIS DAUPHINE
U.F.R SCIENCES DES ORGANISATIONS
LABORATOIRE D'ANALYSE ET MODELISATION DE SYSTEMES POUR L'AIDE A LA DECISION
(LAMSADE)

# Modèle ARCO : Apprentissages multiples et Raisonnement réflexif sur des Connaissances hOmogènes

THÈSE
pour l'obtention du titre de
DOCTEUR EN INFORMATIQUE

présentée et soutenue publiquement par

Philippe CAILLOU

**JURY**

| | |
|---|---|
| **Directeurs de thèse :** | **Monsieur Edwin DIDAY**<br>Professeur à l'Université Paris Dauphine |
| | **Madame Suzanne PINSON**<br>Professeur à l'Université Paris Dauphine |
| **Rapporteurs :** | **Monsieur Tristan CAZENAVE**<br>Maître de Conférence – HDR – à l'Université Paris 8 |
| | **Monsieur Richard ÉMILION**<br>Professeur à l'Université d'Orléans |
| **Suffragants :** | **Madame Anne NICOLLE**<br>Professeur à l'Université de Caen |
| | **Madame Sylvie KORNMAN**<br>Maître de Conférence à l'Université Paris Dauphine |

**Jeudi 9 décembre 2004**

# Table des matières

# Plan détaillé

# Chapitre 1

# Introduction

Lorsqu'il apprend un jeu, un être humain conçoit et utilise des concepts de plus en plus abstraits, que ce soit pour percevoir, agir ou réfléchir. Ainsi, aux échecs, des heuristiques de perception sont d'abord créées pour identifier rapidement les pièces, puis les ensembles de pièces qui correspondent à des structures connues. C'est pourquoi les champions d'échecs peuvent replacer plus de pièces sur un échiquier après l'avoir observé quelques secondes, uniquement si l'emplacement initial correspondait à une partie réelle. Au niveau de l'action, des concepts plus abstraits sont également créés et utilisés, tels que la menace de pièce. Ces concepts de plus en plus abstraits sont utilisés dans des heuristiques de raisonnements construites progressivement en fonction des situations rencontrées. Aux échecs, de telles heuristiques permettent notamment de n'étudier et évaluer que certains coups parmi les très nombreux possibles.

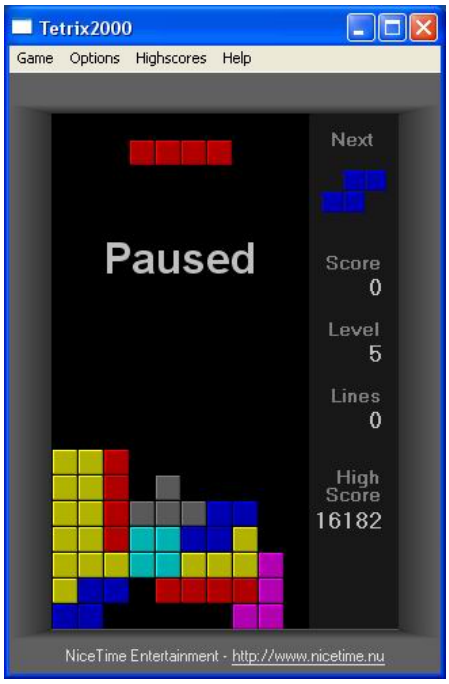



**Figure 1.1 : Exemple de jeu de Tetris**

De la même manière, dans des jeux temps réel tels que Tetris, l'apprentissage de concepts et d'heuristiques associés est obligatoire pour progresser. Dans ce jeu, des pièces

constituées de quatre briques adjacentes apparaissent en haut de l'écran et descendent jusqu'à ce qu'elles rencontrent un obstacle (voir figure 1.1). Le joueur peut déplacer la pièce qui descend vers la gauche ou la droite, ainsi que faire une ou plusieurs rotations. Une fois qu'une pièce s'est arrêtée, s'il existe des lignes complètes dans le jeu (une ligne horizontale sans trou) ces lignes disparaissent et toutes les briques sont décalées vers le bas. La vitesse de descente des pièces est de plus en plus rapide et le joueur doit donc repérer le plus vite possible le type de pièce et la placer où il le juge préférable.

Pour réaliser l'identification, il a tout d'abord intérêt à apprendre les 7 types de pièces et leur forme. Il peut ensuite se servir de leur couleur pour déduire rapidement à partir de cette seule information la forme de la pièce, et ce quelle que soit sa rotation. Connaissant les différents types de pièces disponibles, il peut également modifier sa façon de percevoir le mur inférieur. Avant même de connaître la pièce – notamment quand la pièce précédente est encore en train de tomber – il peut analyser la forme du mur, non pour la retenir précisément, mais pour identifier des formes qui peuvent correspondre aux différentes pièces connues. Ce type de comportement au jeu de Tetris, ainsi que des heuristiques d'action comme pour la méthode de placement des pièces, ont été observés notamment par [Kirsh et Maglio 1994]. Le joueur a adapté sa sémantique (les concepts) et son comportement (façon de réagir, de percevoir, d'agir et de raisonner) à l'environnement.

## 1.1. Importance et définition de l'adaptabilité

Adapter sa sémantique et son comportement à l'environnement permet à un agent – humain ou artificiel – de raisonner à partir de ces nouveaux concepts, de créer des règles particulières correspondant aux situations identifiées. Certains considèrent même l'adaptabilité comme une définition possible de l'intelligence, en faisant un des objectifs logiques de l'intelligence artificielle. Ainsi, Pei Wang définit et situe l'adaptabilité de la façon suivante ([Wang 1995: p.23]):

> To adapt means that the system learns from its experiences. It carries out tasks and adjusts its internal structure to improve its resource efficiency, under the assumption that future situations will be similar to past situations. Not all information processing systems adapt to their environment. For instance, a traditional computing system gets all of its knowledge during its design phase, or before its *birth*. After that, its experience contains tasks only, and the results do not further contribute to the experience of the system. Indeed, to acquire new knowledge, such a system would have to be redesigned, which certainly cannot be done by communicating with a human user in its interface language. On the other hand, not all experience-related changes can be called *adaptation*. Adaptation

> takes place only if the change involved helps to make the system work better, provided that the environment is relatively stable.

L'importance de l'adaptabilité semble particulièrement forte dans le cas d'agents intelligents. On peut définir le terme d'agent intelligent en suivant Wooldridge [Wooldridge 1999] : « un système informatique situé dans un environnement et qui est capable d'agir de façon autonome et flexible sur cet environnement pour atteindre ses objectifs prédéfinis ». L'agent évolue rarement dans un environnement qu'il connaît parfaitement. Il doit donc le découvrir et s'adapter s'il veut être efficace. Il doit modifier sa façon de percevoir et d'agir en fonction de l'environnement. Par exemple, un robot martien ne sait pas ce qu'il va découvrir à l'origine. Il lui serait intéressant d'apprendre à percevoir les différents types d'obstacle qu'il peut rencontrer pour pouvoir les identifier rapidement et ainsi se mouvoir et agir plus efficacement. De même un agent négociateur peut avoir intérêt à identifier et à reconnaître des types de partenaires, afin de pouvoir adapter ses propositions ou anticiper leur comportement.

Fonctionnant en temps réel, l'agent informatique, comme l'agent humain, n'est souvent pas capable de déterminer à chaque instant quel est l'action optimale en fonction des circonstances. Sa rationalité est limitée. Plus précisément, sa rationalité déclarative, c'est-à-dire sa capacité à trouver la meilleure solution possible, est limitée. Sa rationalité procédurale, c'est-à-dire sa capacité à déterminer une procédure de choix adaptée, devient alors fondamentale [Simon 1976, Nicolle 2002].

Pour être efficace dans un environnement inconnu, un système doit pour s'adapter, non seulement découvrir de nouveaux concepts, mais aussi s'autoprogrammer en fonction de cet environnement. Autrement dit, il doit construire des heuristiques adaptées (la notion d'heuristique qui sera utilisée ici est celle correspondant à la définition donnée par D. Lenat [Lenat 1982], simple « règle de jugement informelle »).

Par exemple, un robot libre de se déplacer et d'interagir a tout intérêt à apprendre le plan du lieu où il se trouve et les visages des principales personnes avec qui il interagit. Mais il sera encore plus efficace s'il peut se modifier lui-même pour définir des actions spécifiques pour se rendre dans une pièce précise de ce plan, et s'il définit des méthodes rapides (au prix éventuel de quelques erreurs) permettant d'identifier rapidement les personnes concernées.

## 1.2. Présentation des composants nécessaires

Comment réaliser un système adaptable ? C'est-à-dire ici comment réaliser un système capable de s'autoprogrammer efficacement en fonction de l'environnement ? Plusieurs composants semblent nécessaires pour qu'un système puisse s'adapter efficacement.

### 1.2.1. Induction

Le système doit pouvoir **induire** de nouveaux concepts à partir de son expérience. C'est-à-dire qu'à partir d'un ensemble d'observations, il doit être capable d'extraire un sous-ensemble intéressant ayant des propriétés communes et de former à partir de là un nouveau concept. Le robot baladeur doit ainsi apprendre le concept de porte pour pouvoir l'utiliser. Le joueur de Tetris les différentes pièces et formes de murs intéressantes. L'induction ne porte pas que sur des objets, mais aussi sur des comportements, dans l'environnement ou dans l'agent lui-même : Ainsi l'agent négociateur doit identifier les comportements standards de ses interlocuteurs (offreurs, demandeurs, sites), l'agent joueur de Tetris apprendre comment placer rapidement une pièce après l'avoir réalisé à plusieurs reprises.

### 1.2.2. Déduction

Découvrir ces nouveaux concepts ne suffit pas. Il faut pouvoir les utiliser. Une fois que le système a identifié un nouvel objet, il doit être capable de créer les règles de perception adaptées pour pouvoir l'identifier. S'il découvre une nouvelle règle sous-jacente à l'environnement, il doit pouvoir en déduire comment changer sa façon d'agir en conséquence. Par exemple, identifier une pièce à Tetris, utiliser une porte, réagir en fonction d'un comportement de négociation. Les concepts appris, et donc les déductions, peuvent également porter sur la méthode de réflexion même (et non sur l'environnement).

Le système doit donc être capable de **déduire** à partir de ce qu'il perçoit de l'environnement et des nouveaux concepts qu'il a créé.

### 1.2.3. Intégration / renforcement

Le système ne doit pas faire que créer ces nouveaux objets et règles, il doit les utiliser de façon efficace. C'est-à-dire utiliser les plus « adaptés », ceux qui donneront les meilleurs résultats dans les circonstances actuelles. L'ajout de nouveaux concepts, bien que pertinents, peut en effet ne faire que ralentir le système s'ils sont utilisés en plus, ou après, ceux déjà

présents, et non en priorité ou à leur place lorsque c'est possible. Une façon d'obtenir cette bonne **intégration** des nouveaux concepts avec les anciens est par un **renforcement** des objets et règles utilisés, qu'ils soient anciens ou nouveaux, en fonction des circonstances.

Par exemple, un robot qui doit identifier des personnes peut favoriser les heuristiques qu'il crée en fonction de leur efficacité, mais aussi selon les heures de la journée où ils ont été le plus présents. Le matin, il cherchera intuitivement parmi les personnes qu'il a l'habitude de voir à cette période de la journée.

### 1.2.4. Apprentissage combiné

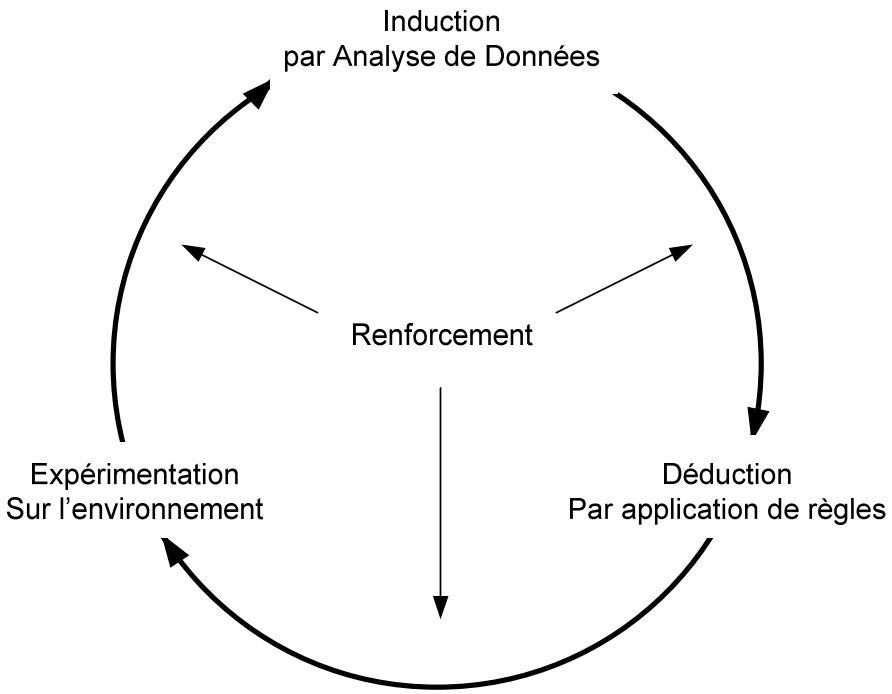
Chacun de ces trois apprentissages est utile, mais c'est un **apprentissage combiné** utilisant les trois méthodes qui est nécessaire pour une bonne adaptabilité, et donc une meilleure efficacité. Ensemble, ces trois méthodes se complètent parfaitement. Au contraire, indépendamment, chacun de ces apprentissages, bien que très efficace dans son domaine d'application, se trouve limité par son objectif même.

L'induction seule est limitée au type de données qu'elle induit. L'analyse d'objets conduit à la découverte de nouveaux types d'objets ; celle de prédicats à de nouveaux types de prédicats. Les règles d'utilisation de ces nouveaux concepts sont elles fixes et donc adaptées au cas général. Des règles mieux adaptées à des cas particuliers ne peuvent être créées (ou très difficilement) avant que ces cas n'apparaissent.

La déduction seule permet de compléter mais a du mal à découvrir. Ainsi, les découvertes réalisées par des programmes déductifs tels qu'AM ou Eurisko proviennent en fait de fonctions inductives basiques (globalement de la recherche aléatoire) intégrées dans des règles déductives.

Le renforcement seul ne permet pas de raisonner mais uniquement d'apprendre des réflexes.

Combinés, les effets des trois apprentissages se complètent : l'induction permet de créer de nouveaux concepts à partir des observations, la déduction de créer les règles et propriétés associées et le renforcement de l'intégrer dans les connaissances pour n'utiliser que les plus efficaces. Ce cycle peut ainsi reprendre : nouveaux concepts, nouvelles déductions, intégration, application et nouvelles observations.




### 1.2.5. Représentation réflexive homogène

Il existe une contrainte majeure pour pouvoir combiner ces types d'apprentissage, qui explique pourquoi les applications actuelles ne le font pas: l'agent doit être capable d'analyser et modifier sa sémantique comme son comportement. Il doit être capable de créer et modifier ses règles de perception et de raisonnement. Il doit être capable d'induire sur les objets perçus dans le monde comme sur les règles qu'il y a observés ou sur ses heuristiques de comportement. Autrement dit, les règles doivent être des concepts comme les autres, elles doivent pouvoir être aussi bien modifiées qu'exécutées. Le système doit avoir une **représentation homogène** des connaissances. Il ne doit pas distinguer les objets des règles (comme dans un système de prédicats), car les règles doivent être modifiées comme les objets. Il ne doit pas non plus distinguer les types de relations des concepts (comme dans les graphes conceptuels ou les logiques de descriptions), car il doit pouvoir modifier sa sémantique en traitant les types de relations comme des objets.

Une fois admise l'utilisation d'une représentation homogène des connaissances, ses nombreux avantages apparaissent :

- *Application possible des règles analysées et construites* : Ces règles s'appliquant sur le graphe, elles peuvent s'appliquer de façon réflexive à elles-mêmes ou à d'autres règles, permettant ainsi un raisonnement réflexif de l'agent sur ses propres connaissances.

- *Traitement uniforme des concepts* : Pouvoir les analyser de façon uniforme ne signifie pas ne créer que des règles générales. Les concepts sont classifiés, mais représentés uniformément. De nombreuses règles peuvent ainsi porter sur comment déduire un sens à partir d'un nouvel objet, ne s'appliquer qu'aux concepts de type « sens » ou « objet », et créer ou modifier des concepts de type « règle ». D'autres règles peuvent porter indifféremment sur un objet ou une règle.

### 1.2.6. Contrôle autonome

Par ailleurs, un agent possédant ce type de raisonnement ne peut pas avoir un objectif fixe et explicitement défini sous forme de connaissances, car il doit pouvoir faire évoluer ses buts en fonction des concepts et règles qu'il apprend. Le contrôle d'un agent raisonnant de façon réflexive doit nécessairement être le plus indépendant possible de sa sémantique [Pitrat 1995], afin d'éviter par exemple la création de règles d'auto-destruction ou d'auto-satisfaction, comme dans Eurisko. Pour que l'agent soit autonome et que l'évaluation des concepts créés ne nécessite pas d'intervention humaine comme dans Eurisko, nous avons généralisé le concept de température utilisé dans CopyCat [Mitchell 1993] pour guider progressivement les règles choisies par l'agent en fonction d'une observation de l'état du système dans son ensemble. Du fait de la similarité fonctionnelle, nous appellerons ces fonctions de monitorage « émotions ». Comme les émotions humaines, les émotions seront ici des fonctions qui, selon l'état de l'environnement et des connaissances de l'agent, modifieront son comportement et influenceront ses choix.

## 1.3. Application au domaine des agents

Quel est l'intérêt de ce domaine par rapport aux autres applications possibles ? Les agents sont par définition situés dans un environnement qu'ils connaissent rarement parfaitement. Ils vont donc le découvrir progressivement et devoir s'adapter, aussi bien au niveau des concepts que des comportements en fonction des différentes situations. La définition à priori par un expert des concepts caractéristiques au domaine est rendue très difficile lorsqu'il est confronté à un environnement ouvert.

Un autre avantage de l'application aux agents par rapport à un domaine comme la résolution de problème est que les erreurs dues à l'utilisation d'heuristiques ont moins d'impact. L'intérêt d'une heuristique est qu'elle arrive au résultat dans un temps beaucoup plus court qu'un algorithme certain. Mais cela peut entraîner une faible probabilité d'erreurs

due aux approximations. Or, l'agent agissant en temps réel cherche rarement la meilleure solution, il cherche à obtenir une bonne solution dans un temps minimal. L'intérêt des heuristiques est alors particulièrement important.

Enfin, le critère d'évaluation utilisé par Eurisko ou par CopyCat était déterminé par des critères internes (fonctions d'évaluation dans Eurisko, activation dans un graphe de concepts fluides dans CopyCat). L'intérêt d'introduire un environnement est de faire réagir l'agent en fonction de cet environnement. Le critère de sélection de règle (une activation dans le graphe, comme dans CopyCat), peut alors dépendre également de l'environnement qui va introduire un facteur extérieur. La perception de la situation actuelle va entraîner l'activation de certains concepts et règles qui vont modifier le comportement de l'agent.

Inversement, quel est l'avantage d'un tel système par rapport aux modèles d'agent actuels ? Principalement l'adaptabilité au domaine, la possibilité de modifier les règles de comportement en fonction d'un environnement qui est inconnu au départ, donc de s'adapter en fonction des règles découvertes.

Ce modèle a l'avantage d'être compatible avec beaucoup de modèles de comportement ou de raisonnement actuels. Par exemple, un modèle comme le modèle BDI [Bratman*, et al.* 1988] définit principalement des concepts relatifs aux objectifs et aux connaissances et des règles pour définir l'ordre dans lequel les traiter. Rien n'empêche d'utiliser un modèle de gestion des objectifs de ce type tout en utilisant un modèle de représentation et de sélection réflexif et heuristique.

**Application aux jeux**

L'agent doit évoluer dans un environnement. Un environnement possible est celui du jeu temps réel, qui présente de nombreux avantages. Les jeux ont toujours été une application privilégiée de l'intelligence artificielle. Ils présentent de nombreux avantages [Schaeffer et Herik 2002] : Un micro-domaine, tout d'abord, ce qui permet d'obtenir des résultats sans avoir à rentrer une masse considérable d'informations pour que le programme fonctionne. Un objectif bien précis (gagner) et des problèmes d'un type constant (les règles sont toujours les mêmes, les conditions d'application stables, et le choix du jeu peut se faire en fonction du type de problème souhaité). Une modélisation claire et une information connue (même si elle n'est pas toujours parfaite). De plus, les solutions trouvées aux problèmes posés par les jeux peuvent bien souvent s'étendre à d'autres domaines. Les problèmes industriels, par exemple, peuvent bien souvent se ramener à un jeu aux règles particulières. Bien gérer une entreprise

peut être assimilé à un grand jeu dont l'objectif est de réaliser un profit maximum (si on se restreint au point de vue financier, bien sûr).

La recherche actuelle dans ce domaine démontre bien l'intérêt de l'utilisation d'heuristiques pour plus d'efficacité. En effet, du fait de la complexité croissante des jeux étudiés, le nombre de coups possibles à chaque tour et le nombre de tours qu'il est nécessaire d'étudier pour arriver à un coup correct sont de plus en plus élevés et rendent une recherche arborescente simple irréalisable. Des heuristiques sont donc utilisées pour limiter la recherche arborescente. Ainsi, au Go, le très grand nombre d'intersections libres pour jouer (19x19 au début du jeu) conduit à réduire la recherche du meilleur coup en calculant à l'avance un grand nombre de résultats de sous-arbres qui apparaissent dans des situations données [Cazenave 1996]. De même à Sokoban, l'utilisation d'heuristiques fondées sur la connaissance du jeu par le programmeur, comme par exemple le fait de savoir que pousser une caisse au début d'un tunnel équivaut au niveau résultat à la pousser jusqu'au bout, permet de réduire considérablement la profondeur de la recherche [Junghanns et Schaeffer 2001]. Ces heuristiques sont le plus souvent données par le programmeur grâce à son expertise (comme pour [Junghanns et Schaeffer 2001]). Mais elles peuvent également être apprises progressivement par le système pour être ensuite intégrées au programme [Cazenave 1996].

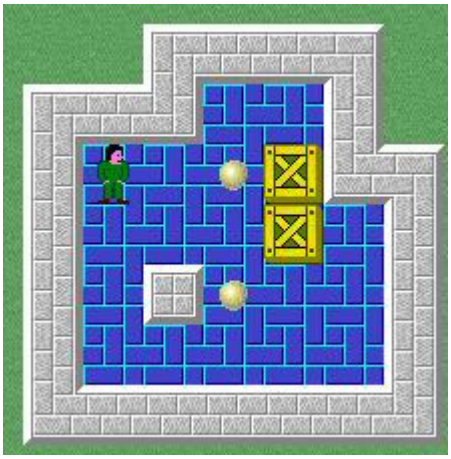


**Figure 1.2 : Exemple simple du jeu de Sokoban : le joueur doit placer les caisses (jaunes) sur les points ronds sachant qu'il ne peut que pousser et qu'il ne peut le faire qu'avec une caisse à la fois**

## 1.4. Problématique et plan

Notre objectif est de proposer un modèle d'agent permettant un apprentissage combiné (induction – déduction – renforcement) et un raisonnement réflexif en utilisant une

représentation homogène des connaissances. Ce système doit permettre à l'agent d'apprendre de nouveaux concepts et comportements en fonction de son environnement et de les utiliser efficacement.

L'étude des systèmes existants, présentée au chapitre 2, montre que les éléments existent en général indépendamment les uns des autres mais ne sont pas compatibles entre eux. En particulier, les différents systèmes d'apprentissage sont fortement dépendants du système de représentation des connaissances utilisé par le système.

Le modèle proposé doit permettre à l'agent de réaliser tous ces apprentissages simultanément. Pour illustrer son fonctionnement, une application dérivée de l'exemple donné de Tetris a été développée. Les éléments essentiels pour tester les différents éléments du modèle ont été conservés. Une présentation globale du système et de l'exemple est donnée au chapitre 3.

La première étape pour réaliser un tel système est de disposer d'un modèle de représentation homogène des connaissances sur lequel les différents types d'apprentissage puissent s'appliquer. Pour cela, nous avons choisi une représentation sous forme de graphe orienté et non typé, dans laquelle on peut simplifier la représentation en utilisant des Frame. Cette représentation est décrite au chapitre 4.

L'architecture globale de l'agent par rapport à son environnement est ensuite décrite au chapitre 5.

Le premier type d'apprentissage nécessaire est l'induction. Celle-ci est réalisée à l'aide d'une analyse de données symboliques sur le graphe de connaissances. La modélisation et la méthode employée est décrite au chapitre 6.

L'analyse de données symboliques nous permet également de suivre de manière dynamique et graphique l'évolution des concepts utilisés et appris par l'agent. Cette méthode de représentation dynamique d'objets symboliques est décrite au chapitre 7.

L'agent doit également être capable de raisonner et déduire, donc d'appliquer des règles. Le fonctionnement de ce raisonnement est décrit au chapitre 8.

Le raisonnement permet à l'agent de déduire et de planifier en fonction d'objectifs explicites. Toutefois, s'agissant d'un agent, il doit disposer d'un contrôle autonome. Et comme la sémantique de l'agent est variable, ce contrôle doit être indépendant de cette sémantique. Les émotions permettent à l'agent de réaliser ce contrôle indépendamment de la sémantique

actuelle. Elles permettent également d'intégrer les nouvelles connaissances grâce au renforcement des liens qu'elles génèrent. Les émotions et le renforcement sont présentés au chapitre 9.

Un système a été développé pour tester le modèle présenté. Les résultats obtenus par rapport aux objectifs décrits ici sont présentés au chapitre 10.

Enfin quelques perspectives sont données après la conclusion au chapitre 11.

# Chapitre 2
# Etat de l'Art

Définir un modèle d'agent qui puisse réaliser un apprentissage multiple et un raisonnement réflexif sur une représentation homogène des connaissances nécessite de combiner principalement trois domaines théoriques particuliers :

- Les systèmes réflexifs pour identifier leur capacité d'adaptation aux différentes méthodes d'apprentissage et au domaine des agents.
- Les systèmes d'apprentissage pour identifier les modèles d'apprentissage inductifs, déductifs et de renforcement existants pour étudier comment ils pourraient s'adapter à un système de représentation homogène des connaissances et donc se combiner entre eux.
- Les modèles d'agent pour comprendre leurs spécificités et l'intégration possible d'une représentation homogène et des divers apprentissages dans les modèles existants.

## 2.1. Systèmes réflexifs

Le principe d'un système réflexif est de pouvoir s'appliquer sur lui-même, permettant ainsi, au moins de façon partielle, de modifier sa façon de raisonner ou son comportement.

Un système réflexif peut être vu comme l'aboutissement logique des systèmes utilisant des métaconnaissances. Les métaconnaissances sont des connaissances portant sur des connaissances [Pitrat 1990]. L'utilisation de telles connaissances dans un système présente de nombreux avantages : elles peuvent permettre de transformer la représentation du problème en fonction des connaissances du domaine, de prendre des décisions à partir d'avis contradictoires, d'expliquer ces décisions et de suggérer des règles (comme dans le système CREDEX, appliqué à l'acceptation de prêts à des PME [Pinson 1987, 1989]). Elles peuvent également servir à surveiller un système comme dans SADE [Kornman 1993b, a]. Ce système montre bien l'évolution logique des systèmes à base de métaconnaissances : celles-ci servant à

contrôler et modifier les connaissances de bases, comment contrôler ces métaconnaissances ? Il est possible d'ajouter d'autres niveaux (des méta-métaconnaissances), et de continuer ainsi à l'infini. Il est aussi possible de ne constituer qu'un niveau et que les métaconnaissances puissent se contrôler elles-mêmes. C'est ce que permet par exemple le système SADE dans lequel le programme peut s'observer en train de s'observer. La distinction métaconnaissance-connaissance est alors purement fonctionnelle car ce qui est considéré comme métaconnaissance à un instant donné peut être pris comme connaissance observé à l'instant suivant. La représentation et le raisonnement sont donc réflexifs. J. Pitrat définit ainsi la conscience réflexive humaine comme « la connaissance directe que nous avons de ce qui se passe en nous » [Pitrat 1995]. Les avantages des métaconnaissances sont conservés et même augmentés par le fait qu'elles peuvent s'appliquer à elles-mêmes. La réflexivité pose toutefois un nouveau problème qui est celui du choix des règles appliquées et des concepts utilisés, dans la mesure où il n'y a plus de niveau supérieur fixe contrôlant le système. Le raisonnement heuristique réflexif propose un début de solution à ce problème en s'inspirant du fonctionnement du cerveau humain.

### 2.1.1. Le raisonnement heuristique réflexif

Comment réaliser un système capable d'un tel fonctionnement ? Il doit pouvoir créer des concepts adaptés au domaine, créer des règles en fonction de ces concepts et appliquer ces règles. Autrement dit, il doit raisonner en utilisant les heuristiques qu'il a lui-même conçues de façon réflexive (en s'auto-programmant).

Les premiers programmes fonctionnant selon ce système sont AM ([Lenat 1978, Davis et Lenat 1982]), suivi d'EURISKO ([Lenat 1983a]). AM est un programme permettant de découvrir de nouveaux concepts mathématiques. Ces concepts sont obtenus par application d'heuristiques de recherche et d'évaluation de concepts. AM a par exemple redécouvert des concepts tels que les nombres premiers ou la multiplication.

EURISKO a étendu le principe de l'utilisation d'heuristiques de recherche et d'évaluation en permettant de les appliquer aux heuristiques elles-mêmes. En plus de chercher des concepts intéressants et adaptés au domaine (dans le cas d'EURISKO, un jeu de société où il faut constituer une flotte de vaisseaux spatiaux), le programme cherche donc des heuristiques adaptées (comme « intégrer un vaisseau rapide et peu armé dans la flotte »). De plus, comme cette recherche se fait à l'aide d'heuristiques, celles-ci peuvent également être modifiées et améliorées. De même, les types d'attributs et de relations entre concepts (intérêt,

est-un, …) ne sont plus fixes comme dans AM, mais également modifiables par des heuristiques.

L'intérêt d'AM et d'EURISKO est de montrer la faisabilité et l'intérêt de l'utilisation d'heuristiques de recherche et d'évaluation de concepts, en particulier lorsque ces heuristiques peuvent s'appliquer à d'autres heuristiques. La notion d'heuristique qui sera utilisée ici est celle correspondant à la définition donnée par Lenat ([Lenat 1982]), simple « règle de jugement informelle ».

Une des limites principale reprochées à AM et EURISKO est que le choix des concepts intéressants se fait en partie par des heuristiques d'évaluation, mais surtout par l'intervention de l'utilisateur.

Ce type de fonctionnement est à la base de la recherche menée par le groupe FARG (Fluid Analogies Research Group [FARG 2004]) fondé à l'université d'Indiana par D. Hofstadter. L'objectif des recherches du groupe est d'étudier et de recréer certaines des compétences centrales les plus complexes de la psychologie humaine, comme l'analogie, la découverte et la créativité. Leur hypothèse de base est que l'activité mentale consiste en un ensemble de micro-évenements et que l'unité apparente de l'esprit humain est une conséquence de la régularité statistique de l'exécution d'un grand nombre de ces micro-évenements exécutés en parallèle [Hofstadter 1985]. Ce modèle a été appliqué successivement à plusieurs micro-domaines :

- Le premier et le plus développé des domaines d'application est celui des analogies lexicales avec CopyCat puis MetaCat, que nous allons détailler plus bas.
- LetterSpirit ([McGraw et Hofstadter 1993, McGraw 1995, Rehling 2001]) est une application au domaine de la création d'alphabets. Le programme simule les comportements d'analogie en recréant des alphabets artistiquement cohérents.
- TableTop ([French 1992, Hofstadter et French 1992, French 1995]) s'intéresse aux problèmes d'analogie du type « Quel est le A de Y ? ». Plus précisément, si une personne d'un coté d'une table montre un élément sur la table et qu'il dit à son voisin d'en face « fait comme moi ! » (« do this ! »), qu'est-ce que le programme doit montrer ? Quel est le meilleur équivalent sachant qu'il se trouve en face ?

- NASR propose une formalisation logique des concepts fluides utilisés pour ces différents modèles. Nous présentons cette logique plus en détail en section 2.1.5.
- D'autres applications sont en développement, comme la géométrie avec la recherche d'analogies entre des figures simples [Foundalis 2004].

Contrairement à AM ou Eurisko, dans CopyCat ([Mitchell 1993, Hofstadter 1995]), son successeur MetaCat ([Marshall 1999, 2002]) et désormais MagnifiCat (en cours de développement), l'utilisateur n'intervient pas car l'objectif est d'obtenir plusieurs résultats différents lors de plusieurs utilisations différentes. Le programme permet de simuler le raisonnement analogique humain sur des problèmes de type abc->abd alors xyz-> ? Pour y arriver, le système exécute un grand nombre d'heuristiques simples (appelées coglets) qui créent, évaluent, confirment et détruisent des liens et des groupes entre les concepts de départ (constitués par les lettres contenues dans l'énoncé du problème). L'efficacité et la qualité du résultat proviennent de l'ordre, et donc du choix, des coglets appliqués. Ce choix est fait grâce à l'activation de concepts tels que longueur, inverse, successeur, 2 ou b présents dans un graphe de concepts fluides. Les coglets liés aux concepts activés ont plus de chance d'être exécutés (directement, ou parce qu'une priorité plus importante leur a été attribuée lors de leur création) ou choisis (lorsqu'il y a besoin de choisir un paramètre à tester).

Ce système ne réalise pas d'apprentissage réflexif (les coglets ne peuvent se modifier entre eux, même s'ils peuvent s'observer) car l'objectif n'est pas l'apprentissage, mais simplement l'application (pour simuler l'analogie). La méthode employée reste toutefois l'application de nombreuses heuristiques sur un graphe de concepts. Un point particulièrement intéressant du système concerne le monitorage global du système, du fait que l'utilisateur n'intervient jamais. Ce monitorage se fait grâce à la température, qui ne décrit pas un objectif précis, mais plutôt un état du système : plus les concepts de départ sont utilisés de façon cohérente, plus la température est élevée. Ce niveau de température va modifier le choix et l'action des coglets, donc le comportement de l'agent.

Ces systèmes montrent l'intérêt et l'efficacité de programmes raisonnant et découvrant de nouveaux concepts adaptés aux domaines à l'aide de nombreuses heuristiques, et permettant de découvrir de nouvelles heuristiques grâce à la réflexivité. Ces systèmes permettent d'une part une certaine créativité dans la mesure où le résultat obtenu n'est pas toujours le même et d'autre part une grande adaptabilité grâce à l'utilisation de concepts et d'heuristiques adaptées au domaine.

Cette utilisation statistique d'heuristiques pour créer et modifier des concepts adaptés au domaine est la caractéristique fondamentale de ce que nous appellerons réflexion heuristique réflexive (RHR). Plus précisément les points communs à ce type de systèmes sont :

- Une utilisation successive de nombreuses heuristiques sur un graphe de concepts
- Le choix du concept et de la règle utilisée se fait de façon probabiliste proportionnellement à un critère d'évaluation.
- Une absence de contrôle centralisé : les heuristiques et concepts sont tous au même niveau, qu'ils soient nouvellement créés ou présents à la création du système

Ce type de système s'inspire comme nous l'avons vu du comportement humain, et a été en particulier décrit en sciences cognitives et en philosophie par D. Dennett ([Dennett 1991, 1996]). En intelligence artificielle, il s'inspire de la « société de l'esprit » décrit par M. Minsky ([Minsky 1985]). Qu'il s'agisse d'homoncules (Dennett), de coglets (Hofstadter) ou de Frames (Minsky, Lenat), chaque élément a le même objectif : une heuristique simple utilisée de façon statistique et portant sur l'analyse et la modification du reste des connaissances.

Comme le montrent Eurisko et CopyCat, l'intérêt principal de ce type de système est l'adaptabilité. Il permet tout d'abord de s'auto-modifier en fonction des règles apprises sur l'environnement. On peut l'interpréter comme une adaptabilité structurelle : le fonctionnement même du raisonnement est changé. Mais le système a également une forte adaptabilité conjoncturelle : en fonction de la situation actuelle, grâce au fonctionnement probabiliste, ce sont les règles et concepts les plus adaptés à la situation qui vont être utilisés.

### 2.1.2. AM et Eurisko

#### 2.1.2.1.AM

L'objectif d'AM ([Lenat 1978, Davis et Lenat 1982], discuté dans [Lenat et Brown 1983, Ritchie et Hanna 1983]) est la découverte de concepts mathématiques par application d'heuristiques de découvertes. Le programme débute avec 115 concepts de base de la théorie des ensembles et 242 heuristiques. Ces heuristiques peuvent accomplir trois types de tâche :

- Suggérer une nouvelle tâche

- Créer un nouveau concept
- Ajouter ou supprimer des informations à un slot d'un concept.

Appliqué à la théorie des nombres, le programme a ainsi découvert des concepts intéressants tels que les nombres premiers (nombre avec deux diviseurs).

La représentation des connaissances se fait sous forme de Frames. Chaque concept a un certain nombre de slots (parmi 25 types de slots disponibles), qui sont remplis soit avec des valeurs soit par des liens vers d'autres concepts.

Les concepts de départ de AM sont représentés figure 2.1

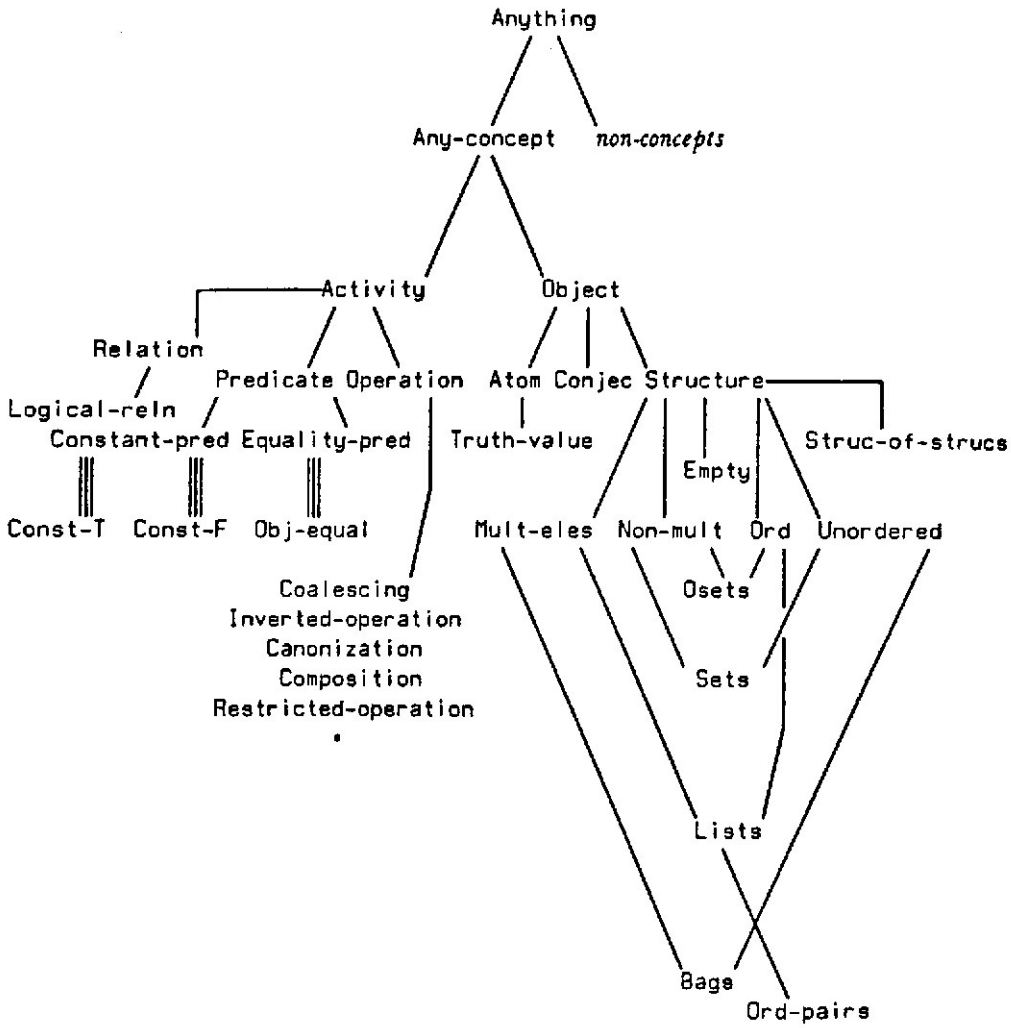



**Figure 2.1 : Graphe des concepts de départ d'AM**

Une tâche du système correspond à un slot à remplir. Le programme dispose d'un agenda de tâches. Quand le temps alloué à une tâche est épuisé, il sélectionne la tâche qui a la valeur la plus élevée dans l'agenda et c'est elle qui devient la nouvelle tâche pour une certaine période. Pendant cette période, le programme cherche les heuristiques qui peuvent aider à remplir la tâche et les exécutent.

Un intérêt d'AM est de proposer un processus de découverte de concepts basé sur des heuristiques et de fournir ces heuristiques de recherche. De plus, c'est un des rares programmes à considérer directement le temps passé dans l'évaluation de l'intérêt des concepts qu'il utilise.

La limite principale réside dans l'importance laissée au programmeur qui doit régulièrement intervenir pour guider le programme ou juger de la pertinence des concepts trouvés. Un autre inconvénient d'AM est qu'il n'apprend pas de nouvelles heuristiques ni de nouveaux types de slots. C'est cette limite qui a été résolue par son successeur Eurisko.

#### 2.1.2.2.Eurisko

Eurisko [Lenat 1982, 1983a, b] a pour objectif de découvrir non seulement de nouveaux concepts, mais également de nouvelles heuristiques et de nouveaux types de slots, et ce toujours en utilisant des heuristiques. Pour cela, aussi bien les heuristiques que les types de slots sont représentés, et donc traités, comme des concepts, c'est-à-dire à l'aide de Frames. De nouvelles heuristiques de découverte d'heuristiques et de slots sont introduites. Par exemple, la règle *Si une règle est utilisée souvent avec succès, en créer une nouvelle en réduisant une de ses préconditions*.

Eurisko a été appliqué à un jeu de société de combat où la difficulté du jeu provient de la constitution de la flotte spatiale à partir d'un ensemble de règles très complexes et précises. L'objectif était qu'Eurisko trouve des concepts et des heuristiques permettant de décrire une flotte efficace. C'est ce qu'il fit puisqu'il remporta plusieurs fois de suite le championnat mondial contre des adversaires humains. Par exemple, il a appris l'heuristique « inclure dans la flotte un petit vaisseau très agile et non armé ». Ce vaisseau, bien qu'inutile au niveau du combat, tirait parti d'une faille dans les règles qui faisait qu'il était inatteignable, et donc indestructible, par les vaisseaux ennemis. De ce fait, le programme était sur d'obtenir au moins un match nul (la victoire ne pouvait être obtenue que par destruction totale de l'ennemi ou par abandon).

La représentation des connaissances est la même que celle d'AM, sous forme de Frames. Par exemple, le concept correspondant au concept d' « Energy Gun » est décrit figure 2.2. La principale différence réside dans le nombre variable de slots possibles, puisqu'Eurisko peut créer de nouveaux types de slots. De plus, les heuristiques et les slots étant également décrits sous forme de Frames, certains nouveaux types de slots font leur apparition au début pour pouvoir les décrire.

```
Name. EnergyGun
Generalizations: (Anything Weapon)
AllIsA: (GameConcept GameObj Anything Category WeaponType
        DefensiveWeaponType OffensiveWeaponType Obj
        AbstractObj PhysGameObj PhysObj)
IsA: (DefensiveWeaponType OffensiveWeaponType PhysGameObj)
MyWorth: 400
MyInitialWorth: 500
Worth: 100
InitialWorth: 500
DamageInfo: (SmallWeaponDamage)
AttackInfo: (EnergyGunAttackInfo)
NumPresent: NEnergyGuns
UspPresent: EnergyGunUSP
DefendsAs: (BeamDefense)
Rarity: (0.11 1 9)
FocusTask: (FocusOnEnergyGun)
MyIsA: (EuriskoUnit)
MyCreator: DLenat
MyTimeOfCreation: "4-JUN-81 16:19:46"
MyModeOfCreation: (EDIT NucMissile)
```

**Figure 2.2 : Frame correspondant au concept EnergyGun d'EURISKO**

La description des heuristiques se fait avec deux slots principaux : un pour les préconditions d'application, et un pour l'action effectuée si les conditions sont remplies. Une version compilée en LISP à chaque fois que ces slots sont modifiés est également présente dans un autre slot pour que l'heuristique puisse être exécutée.

Contrairement à AM qui disposait d'un seul agenda, le contrôle se fait à l'aide de plusieurs agendas. Plusieurs concepts disposent de leur propre agenda de tâches qui se rapportent à eux-même ou aux concepts qui héritent d'eux. Lorsqu'un agenda devient trop important il est divisé et reparti entre les sous-concepts.

Eurisko distingue en fait trois niveaux de contrôle :

- Niveau supérieur : le programme décide quel concept va être analysé ; Une période de temps relativement longue lui sera attribuée.

- Niveau intermédiaire : le programme choisit une tâche parmi celle des agendas du concept ou des concepts dont il hérite s'il ne dispose pas de son propre agenda. Une période de temps (courte) est associée à cette tâche.
- Niveau inférieur : le programme recherche une heuristique permettant d'aider à remplir la tâche en cours et l'exécute.

L'intérêt principal d'Eurisko est bien sûr de fournir une méthode de découverte de concepts et d'heuristiques dans un contexte de règles nombreuses et complexes. Il permet de plus au système de s'auto-améliorer en modifiant ses propres règles de recherche et de fonctionnement pour être toujours plus efficace.

A l'exception de l'apprentissage des heuristiques et des slots, les critiques valables pour AM restent vraies pour Eurisko : celui-ci a besoin d'une forte intervention de l'utilisateur pour juger régulièrement de la pertinence des heuristiques et des concepts trouvés. Il n'est de plus pas adapté à un concept d'agent travaillant en temps réel avec des données imprécises ou erronées.

La méthode utilisée par notre modèle pour le raisonnement déductif est proche de celle d'Eurisko, en l'adaptant à une représentation des concepts sous forme de graphe et non plus en LISP, ce qui permet un apprentissage implicite et un contrôle par transmission d'activation, et en l'intégrant dans un agent, donc en prenant en compte en particulier le fait que les connaissances peuvent être fausses ou incomplètes, ainsi que l'importance du temps dans l'action et la perception.

### 2.1.3. Maciste

MACISTE ([Pitrat 1993, 1998], une présentation est donnée également par [Starynkevitch 2001]) a de nombreux points communs avec les systèmes de RHR. L'objectif est une réflexivité complète du système afin qu'il puisse s'autoprogrammer. Le programme n'utilise effectivement qu'un très petit nombre de commandes C de base, le reste des fonctions étant programmé dans le formalisme logique de MACISTE. Le formalisme a par exemple été appliqué dans MALICE ([Pitrat 1999, 2000]) pour monitorer une résolution de problème.

La sémantique et le comportement du système sont entièrement et dynamiquement modifiables. Le système est divisé entre un grand nombre d'expertises. Cette division est purement fonctionnelle, les connaissances étant représentées au même niveau.

Techniquement, les règles sont décrites et analysés sous forme déclarative, puis une version compilée leur est associée. Il existe donc deux représentations de chaque règle : une pour l'analyse et une compilée à chaque modification.

Une connaissance (comme une métaconnaissance) est un couple « attribut-valeur ». Une règle est constituée d'un ensemble de prémisses et d'un ensemble d'actions. On constate ici que bien que le système en lui-même soit entièrement réflexif, cela n'implique pas que les connaissances et les règles soient représentées de façon homogène. Une différence importante avec notre système, au niveau de l'objectif, est que MACISTE cherche à être un système entièrement réflexif. Nous cherchons à représenter de façon homogène les connaissances et à les traiter uniformément, ce qui implique un système en grande partie réflexif.

Par ailleurs, le choix des règles et des concepts utilisés dans MACISTE est lui aussi réalisé grâce à une expertise et donc à un ensemble de règles. La différence avec un système de RHR vient donc de l'absence de fonctionnement stochastique. Par rapport à notre approche, l'objectif d'autonomie et d'interaction avec l'environnement est de plus absent. Le système à un problème bien défini et il le traite sans soucis d'interaction.

### 2.1.4. CopyCat et MetaCat

L'objectif de CopyCat ([Mitchell et Hofstadter 1990, Mitchell 1993], présenté notamment dans [Pitrat 1998]) est de reproduire le raisonnement analogique humain. Son but n'est pas l'apprentissage. CopyCat, puis MetaCat ([Hofstadter 1995, Marshall 1999, 2002, 2004]) utilisent de nombreuses heuristiques (*coglets*) de façon probabiliste, avec un système de transfert d'activation similaire à celui que nous utilisons pour modifier la probabilité d'activation des coglets. Le rôle des coglets est de créer de nouvelles structures et liens dans l'espace de travail et de modifier les liens entre les connaissances du graphe de concepts fluide (*slipnet*), modifiant par là même ces concepts. L'objectif est de résoudre un problème d'analogies entre chaînes de caractères (du type abc->abd, alors ijk-> ?)

Concrètement, le problème est représenté dans un espace de travail (tel que celui donné en figure 2.3). Le programme effectue successivement un grand nombre de coglets qui vont créer des groupes et ajouter des liens entre les éléments de l'espace de travail. Le choix des coglets se fera en fonction de leur urgence qui est définie lors de leur création en fonction de l'activation des concepts auxquels ils sont liés dans le slipnet du programme (les slipnet initial de CopyCat est donné en figure 2.4). Par exemple, si des groupes de longueurs identiques sont trouvés, le concept de longueur va être activé dans le slipnet et les coglets associés à la

longueur vont être exécutées avec une plus grande probabilité. Les concepts du slipnets sont liés entre eux par des liens typés qui ont une certaine intensité et qui permettent de propager l'activation entre des concepts proches. Le système ne peut créer de nouveaux concepts ou de nouveaux liens dans le slipnet, mais il peut changer l'intensité des liens.

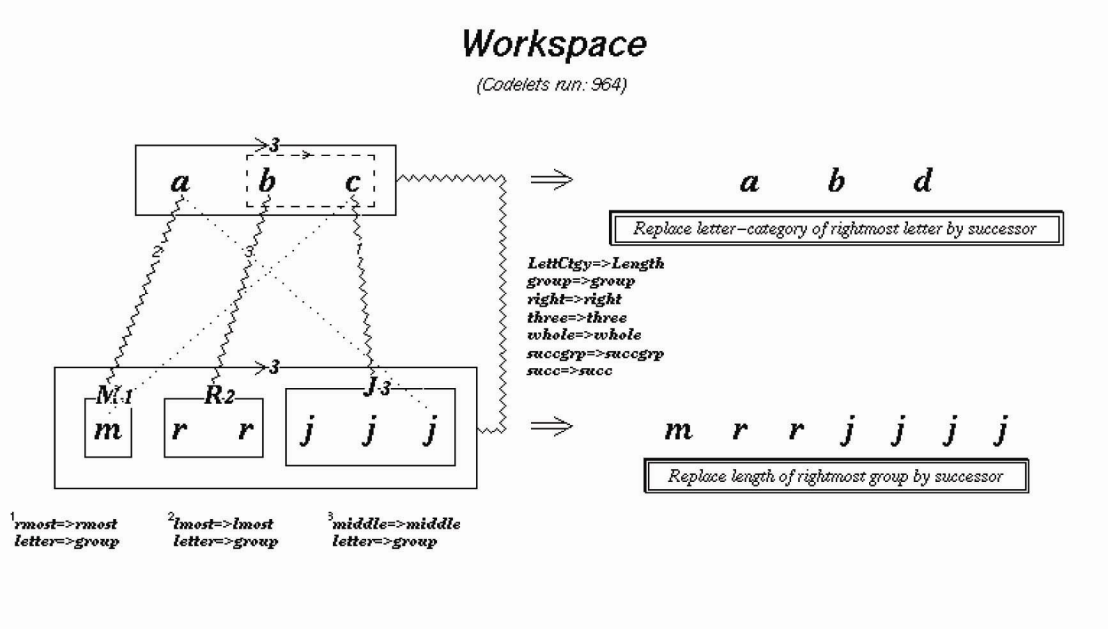



**Figure 2.3 : Espace de travail de CopyCat pour abc->abd alors mrrjjj-> ?**

Le nombre de coglets varie au cours de l'exécution car les coglets peuvent en créer d'autres qu'ils ajoutent à la liste des coglets exécutables (coderack). La construction des structures par les coglets a lieu progressivement. Ils commencent par proposer des groupes ou des ponts entre les structures. Puis ces structures sont renforcées par certains coglets et mis en concurrence avec d'autres structures. Et si la structure résiste, elle est enfin validée. Un groupe ou un lien est donc construit par une série de coglets plutôt que par un coglet isolé.

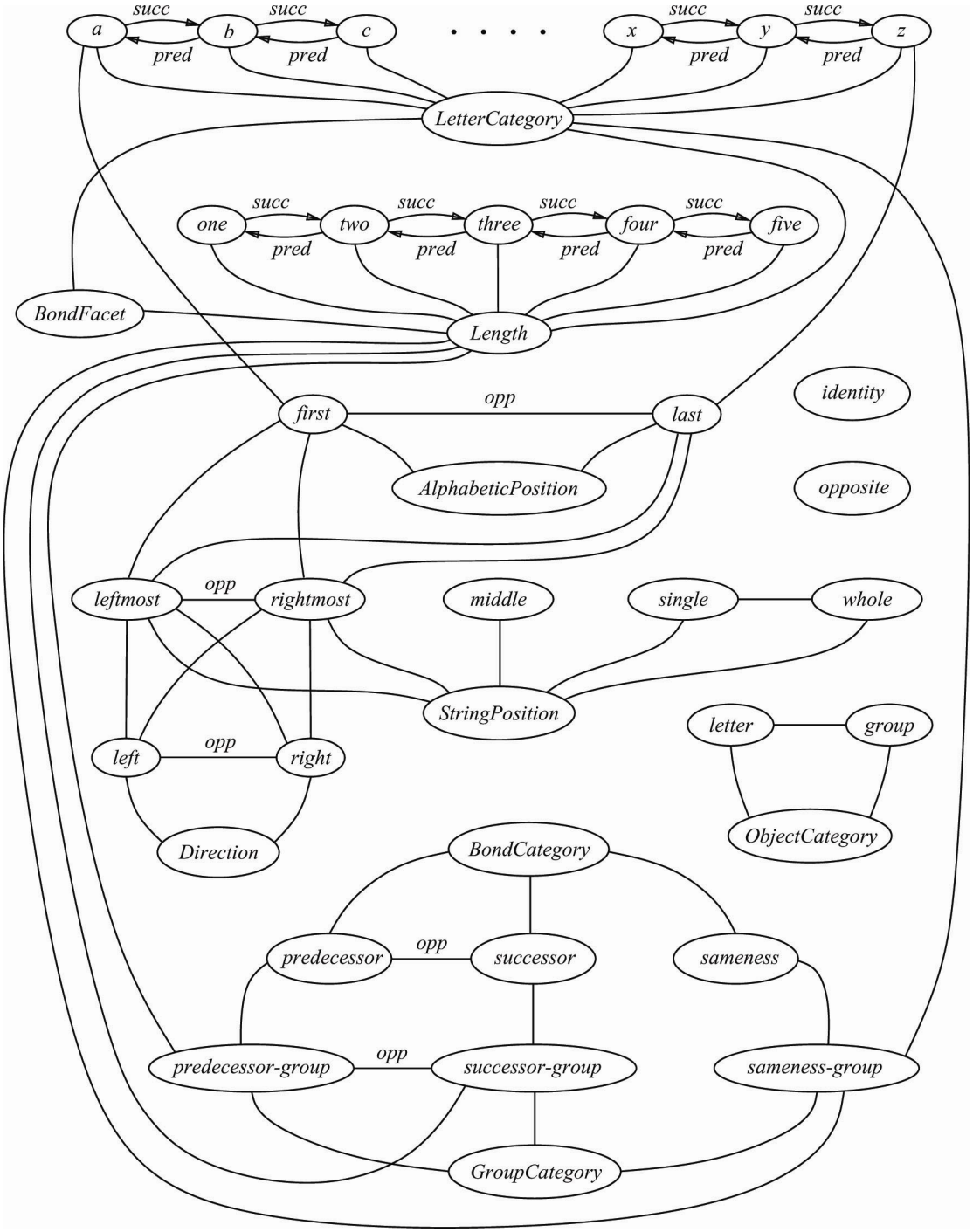



**Figure 2.4 : SlipNet de départ de CopyCat**

L'objectif de ces programmes n'est pas directement l'apprentissage puisqu'ils visent à simuler la constitution d'analogies par le cerveau humain, il conduit toutefois à la création d'une nouvelle règle heuristique qui exécute l'analogie (qui n'est pas explicite mais présente grâce aux liens). L'absence d'apprentissage de nouvelles règles et de nouveaux concepts est la principale différence avec le modèle que nous proposons.

Outre l'utilisation successive d'un grand nombre d'heuristiques sur un graphe de concepts liés entre eux, qui est la base de ce type de système, la recherche de la solution présente des similitudes intéressantes avec AM et EURISKO. En particulier, le fait d'appliquer un grand nombre d'heuristiques simples conduit à différencier la proposition d'une hypothèse, de son test et de sa validation. Les nouvelles structure ou concepts sont construits progressivement par apports successifs et par confrontation à d'autres solutions possibles. Nous reprendrons cette méthodologie pour la construction, l'évaluation puis la validation des hypothèses.

### 2.1.5. NASR

L'objectif de NASR (Non Axiomatic Reasoning System), présenté dans [Wang 1994, 1995, to appear], est de fournir un formalisme logique aux concepts fluides comme ceux utilisés par CopyCat. La sémantique du système devant être évolutive en fonction de l'environnement, cette logique doit être réflexive.

La logique NASR est caractérisée par une relation binaire non typée entre les concepts, disposant de deux poids : la fréquence f et la confiance c.

> Un jugement $S \subset P < f, c >$ signifie que S est dans l'extension de P et que P est dans l'intension de S, avec les valeurs de vérité du jugement définissant leur degré d'appartenance. ([Wang 1995] p.46)

Par exemple, $S \subset P < 0.8, 0.2 >$ signifie que P est un S dans 80% des cas mais que cette information n'est valable qu'avec une confiance de 0,2.

Au contraire, $S \subset P < 1,1 >$ signifie que P est un S de façon certaine.

La réflexivité de la propriété d'héritage permet d'illustrer son fonctionnement. Du fait de cette réflexivité, chaque concept se contient au moins lui-même dans son extension et se définit en intention. Les « types » dont héritent les concepts ne sont donc pas des catégories qui pourraient avoir comme propriété leur nombre d'éléments, mais au contraire l'ensemble des propriétés communes de leurs descendants. Ils définissent ainsi les descendants en intention (par ces propriétés communes), tandis que la présence de ces propriétés définie le fait qu'ils appartiennent à l'extension. D'où la dualité extension-intention et la réflexivité de la relation.

A partir de cette logique, P. Wang définit un ensemble de mécanismes d'inférence logique adaptés à ces concepts fluides.

Cette utilisation pour l'inférence logique illustre la principale différence avec notre approche. La représentation et la sémantique correspondent très bien à la notre avec une différence symbolisée par le facteur fréquence f des liens : la logique NASR définit f comme une fréquence, permettant ainsi un raisonnement probabiliste et des calculs d'inférence automatiques. Notre objectif n'est pas de raisonner par inférence logique mais de raisonner explicitement en utilisant les liens pour transmettre un facteur d'activation. Cette interprétation fonctionnelle des liens nous conduit à utiliser un facteur d'intensité, compris entre -1 et 1 pour permettre l'inhibition, là où NASR utilise une fréquence probabiliste comprise entre 0 et 1.

## 2.2. Systèmes d'apprentissages

Comment apprendre de nouvelles règles et de nouveaux concepts? Les systèmes d'apprentissages peuvent se distinguer en fonction du type de représentation des connaissances qu'ils supposent, du type de connaissance apprise, et surtout de la méthode utilisée. On distingue traditionnellement trois types de méthodes ([Mitchell 1997], [Cornuéjols et Miclet 2002]). Soit on utilise des règles de déduction sur les connaissances actuelles de l'état du monde pour en déduire de nouvelles règles et de nouveaux faits (apprentissage déductif), soit on considère un grand nombre d'observations et on cherche des règles ou des concepts dans ces observations (apprentissage inductif). Une combinaison des deux permet de combiner les avantages des deux systèmes. Soit on réalise un renforcement automatique entre les nœuds d'un graphe en fonction du résultat d'une fonction d'évaluation.

### 2.2.1. Déduction

Le principe de base de la déduction est d'utiliser des connaissances (règles) sur d'autres connaissances (faits) pour pouvoir ainsi les enrichir.

L'avantage des systèmes déductifs est de pouvoir être appliqués à des domaines abstraits, tels que la résolution de problème ou les mathématiques. Au contraire les sytèmes inductifs, se fondant sur un ensemble d'observations, nécessitent des données concrètes. AM et Eurisko, premiers systèmes déductifs et réflexifs permettant de découvrir des heuristiques, portaient ainsi sur les mathématiques et sur des règles de jeu. Les systèmes de déduction développés par la suite se situent principalement dans le cadre de l'Explanation Based Learning, avec en particulier Prodigy, programme appliqué à la résolution de problèmes. Avec la même application, SOAR, dont le but est la modélisation de la cognition humaine, propose une approche différente de l'apprentissage déductif.

#### 2.2.1.1.Explanation Based Learning

Le principe d'Eurisko est de créer des concepts, de les remplir et de voir s'ils sont intéressants ensuite. Il peut très bien créer des concepts qui n'existent pas dans le contexte étudié, des règles fausses ou des heuristiques qui ordonnent de détruire le système.

Une autre façon d'aborder le problème de la découverte de règles est de chercher uniquement des règles vraies, et ce en justifiant ce qui s'est passé sur un exemple, que celui-ci ait été bon ou mauvais. C'est le principe de l'Explanation Based Learning (EBL) : à partir de règles de fonctionnement du domaine, les programmes d'EBL analysent des exemples et en déduisent des règles. C'est pourquoi ce type d'apprentissage est généralement appelé apprentissage déductif (il déduit des règles du domaine une règle qui explique ce qui s'est passé ou comment le reproduire), ou analytique (il analyse une situation pour la justifier et la reproduire ou l'éviter). L'objectif de ce genre d'apprentissage est généralement de fournir uniquement des règles qui soient prouvées (car elles sont fondées sur des observations et des règles du domaine sans erreur) et donc vraies. Un bon exemple de programme d'EBL est Prodigy qui applique le principe de l'apprentissage analytique à la résolution de problème. Un autre système, SOAR, bien que plus général dans son approche de l'intelligence artificielle (il modélise un processus cognitif complet), fonctionne sur le même principe que l'EBL tout en employant une technique particulière pour l'apprentissage de règles ou de concepts.

##### 2.2.1.1.1. Prodigy

L'objectif de Prodigy ([Minton*, et al.* 1989]) est d'apprendre à résoudre des problèmes de plus en plus rapidement.

La représentation des connaissances se fait sous forme de Frames Objets-Attributs. Le système distingue plusieurs types de connaissances :

- Les règles générales d'apprentissage
- Les connaissances du domaine
- Les règle apprises

L'apprentissage se fait par développement progressif d'une explication et des préconditions qui lui sont associées. Une fois l'explication (de l'échec ou du succès obtenu) développée, celle-ci est ajoutée aux règles apprises et applicables pour résoudre un problème. Il faut noter que l'explication est développée directement sous forme généralisée, aucun

retraitement n'est donc nécessaire à la fin de la résolution pour qu'elle soit applicable par la suite à des problèmes similaires.

Un des avantages de Prodigy est que les auteurs se sont particulièrement intéressés à l'efficacité marginale des règles. En effet, ils ont bien remarqué que l'ajout d'une règle au système n'allait pas obligatoirement accélérer la résolution des problèmes étant donné les coûts de traitement que représente l'ajout. Les règles créées sont ainsi évaluées en fonction des gains qu'elles apportent au système.

L'intérêt d'un système comme Prodigy est bien sûr le système de création d'explication à partir du domaine qu'il propose. Sa limite, surtout par rapport à notre objectif, est qu'il ne propose que des règles prouvées, et non des heuristiques permettant d'accélérer fortement un traitement même si leur application comporte un risque d'erreur. Les données traitées par le modèle doivent de plus être précises et sans erreur. Les données du domaine doivent également permettre de trouver une explication, donc être complètes et sans inconsistances. Autant de contraintes inadaptées au monde des agents.

Notre modèle cherche au contraire à proposer des règles efficaces et utiles pour l'agent. Elles peuvent alors parfois être fausses, voir inconsistantes entre elles, la transmission de l'activation permettra d'activer la bonne au bon moment en fonction du problème traité par l'agent. L'essentiel est qu'elles permettent à l'agent de percevoir plus vite des informations plus pertinentes et d'agir similairement de façon plus efficace.

##### 2.2.1.1.2. SOAR

L'objectif de SOAR ([Newell 1990]) est de modéliser le comportement cognitif humain dans son ensemble, et en particulier la méthode de résolution de problèmes. L'idée de base est que chaque fois que le système rencontre une difficulté pour atteindre un but, il crée un nouveau but correspondant à la résolution de cette difficulté. A chaque but est associé un espace de travail qui lui est propre et qui débute complètement vide. Des éléments sont ensuite ajoutés à cet espace de travail (et jamais retirés) jusqu'à ce que l'élément correspondant au but soit ajouté, et donc que le but soit atteint.

L'apprentissage ce fait par « chunking », qui correspond à une généralisation de la méthode utilisée pour atteindre un but. Une fois qu'un but a été atteint, une nouvelle règle est créée qui relie les éléments nécessaires aux règles utilisées pour atteindre ce but au but lui-

même, ce qui signifie que lorsque ces éléments seront présents dans un espace de travail, la nouvelle règle ajoutera automatiquement le but à cet espace de travail.

L'avantage de cette méthode d'apprentissage est qu'elle permet d'apprendre de nombreux types de concepts avec une méthode identique, y compris des règles temporelles ou non.

Par rapport à notre système, l'objectif de Soar, comme pour les systèmes d'EBL est toutefois d'obtenir une règle vraie qui permette de résoudre une généralisation du sous-problème considéré, de façon parfaite et systématique. L'efficacité et la dynamique de l'environnement n'entrent pas en considération.

#### 2.2.1.2.Introspect : apprentissage automatique d'heuristiques par auto-observation

Le domaine du jeu offre une application privilégiée pour appliquer les méthodes d'apprentissage. Le jeu de Go, en particulier, offre un nombre de coup possible tel que les méthodes de calcul brutes sont inapplicables. Introspect ([Cazenave 1996]) propose un système de déduction d'heuristiques admissibles pour accélérer les calculs de coups.

L'objectif est d'améliorer un système de recherche arborescente classique en y incluant automatiquement des heuristiques déduites par généralisation de problèmes étudiés. La modélisation est donc fixe et donnée par le programmeur, le programme construit des heuristiques admissibles sur des sous-problèmes. Ces heuristiques sont compilées en C++ et intégrées directement au programme de jeu. Le programme ne s'auto-observe pas lorsqu'il joue, il se contente d'appliquer la recherche arborescente améliorée par les nombreuses heuristiques déduites. Le Schéma global de fonctionnement de l'apprentissage est présenté figure 2.5.

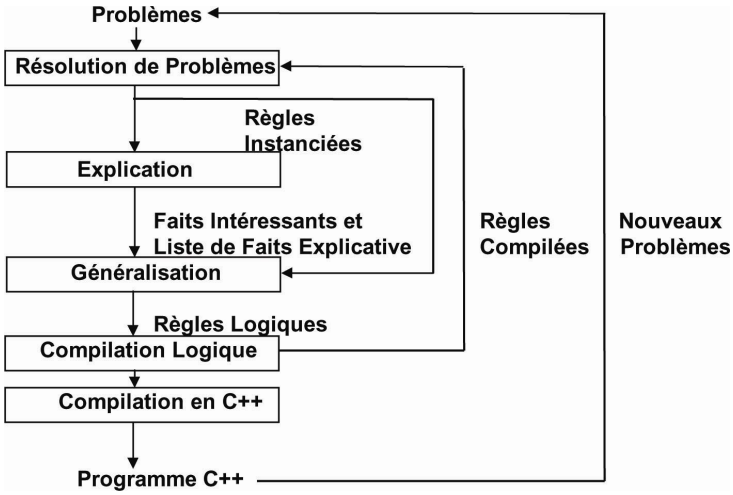


**Figure 2.5 : Schéma de fonctionnement d'Introspect (tiré de [Cazenave 1996])**

Tout comme notre système, Introspect cherche à déduire des heuristiques qui lui permettent de se comporter de manière plus efficace. La recherche se limite toutefois aux heuristiques admissibles (c'est-à-dire dont le résultat est vrai) alors que nous cherchons des heuristiques efficaces, même si cela peut parfois provoquer des erreurs. De plus, l'objectif d'Introspect est une efficacité maximum lors du jeu grâce à la compilation des règles en C++, alors que nous cherchons à pouvoir adapter les concepts et les règles y compris pendant le jeu. Ceci est en particulier dû au fait que l'agent ne fonctionne pas par phase d'analyse/compilation/application, mais au contraire en continu et en interaction avec son environnement. Introspect distingue également un méta-niveau qui réalise l'apprentissage (non modifiable) et un niveau d'application qui est modifié avec les heuristiques. Enfin, la sémantique est fixe et le concept n'apprend pas de nouveaux concepts plus abstraits sur lesquels il pourrait apprendre des heuristiques plus générales.

#### 2.2.1.3.LRTA* : Algorithmes de recherches d'heuristiques admissibles en temps réel

Un de nos objectifs étant de découvrir des heuristiques adaptées au domaine dans un agent temps, il est naturel de citer les algorithmes dont l'objectif est de découvrir des heuristiques admissibles en temps réel (d'autant plus que ceux-ci ont été appliqués aux agents). Bien que la formulation puisse paraître semblable, nous allons voir que notre objectif est différent, voire complémentaire.

Le cadre d'application des algorithmes de recherche d'heuristiques en temps réel tels que LRTA* (Learning Real Time A* [Korf 1990, 2000], extension de l'algorithme de recherche de chemin au meilleur coût A* [Hart*, et al.* 1968]) est le suivant :

- X : un ensemble fini non vide d'états (nœuds)
- A : un ensemble d'actions (liens) $A \subset X \times X - \{(x,x) \mid x \in X\}$
- k : fonction qui associe à chaque lien un coût $k : A \to [0;\infty[$
- s : état initial $s \in X$
- G : ensemble d'états objectifs $G \subset X$

A partir du graphe (X,A), on cherche à atteindre un des états objectifs de G avec un coût minimal. Pour cela, l'algorithme utilise une fonction heuristique h ($h : X \to [0;\infty[$) qui évalue le coût pour aller d'un état x à un état quelconque de G. si h(x) est exactement égal au coût minimal, h est dite *correcte*. Si h est inférieure ou égale au coût minimal, h est *admissible*. L'algorithme va progressivement améliorer la fonction heuristique h en parcourant le graphe et ainsi déterminer les chemins optimaux à partir des différents états du graphe.

L'algorithme LRTA* a été notamment appliqué au domaine des agents par [Ishida 1997], puis amélioré par des mécanismes de contrôle par [Shimbo et Ishida 2003].

La description du fonctionnement de l'algorithme en lui-même n'est pas nécessaire pour comprendre la différence fondamentale avec notre approche. Ce type d'algorithme suppose un environnement connu, et en particulier un nombre fini d'états et de liens pour être appliqués, et si possible des coûts fixes et connus entre les états et l'assurance qu'un lien existe pour atteindre un état objectif à partir de n'importe quel état. D'où la définition utilisée de la fonction heuristique qui constitue la plus grande inconnue du système : le coût pour atteindre un objectif depuis un état x. Notre objectif est différent, il consiste à découvrir l'environnement lui-même, les concepts sous-jacents et ses règles de fonctionnements, et d'adapter le fonctionnements et les connaissances de l'agent en fonction de cet environnement. C'est pourquoi notre définition d'heuristique est celle de D. Lenat, plus large, qui la définie comme « règle de jugement informelle ». Nous recherchons des heuristiques pour l'agent signifie que nous cherchons des règles au comportement incertain et éventuellement erroné.

De la même façon qu'une architecture d'agent de type BDI (voir section 2.3.1), les algorithmes du types LRTA* sont en fait complémentaire de notre approche dans la mesure où ils proposent des règles de raisonnement et de planifications. Nous proposons un modèle de création de concept et de choix des règles appliquées. Pour se replacer dans le formalisme de ces algorithmes, nous cherchons à définir le graphe (X,A) ainsi que les fonctions de coût que l'agent va utiliser. Il serait donc intéressant d'ajouter ce type d'algorithme, en tant que fonction intégrée ou, mieux, de façon explicite, à notre agent pour l'aider à se décider.

### 2.2.2. Induction

L'objectif de l'apprentissage inductif est de déduire des règles ou des concepts à partir d'un grand nombre d'observations. On peut distinguer deux grandes familles de méthodes:

l'apprentissage artificiel et la fouille de données (Machine Learning vs Knowledge Discovery in Databases) ([Kodratoff 2001]). Alors que le but du ML est d'obtenir de l'information cohérente et prouvée, le KDD cherche à obtenir de l'information utile, auparavant inconnue, même si elle est contradictoire. Selon Y. Kodratoff, la différence est épistémologique: le KDD doit être basé sur le monde réel pour modifier le comportement de l'agent (utilisateur des résultats), même si les informations obtenues ne sont pas prouvées, pas précises et adaptées uniquement au cas particulier considéré. C'est précisément notre objectif: obtenir des informations utiles pour l'agent qui puissent guider son comportement pour bien utiliser les concepts et règles découverts grâce à la déduction.

**Différences dans l'approche scientifique**

| **Informatique classique** | **Apprentissage** | **Fouille de Données** |
|---|---|---|
| Simule un raisonnement déductif ( = applique des modèles existants) | Simule un raisonnement inductif ( = invente un modèle) | Simule un raisonnement inductif ( « encore plus inductif ») |
| Résultats théoriques, valides si prouvés | Résultats théoriques, valides si prouvés | Résultats théoriques, valides si prouvés et intelligibles |
| Résultats expérimentaux, validés par leur précision | Résultats expérimentaux, validés par leur précision | Résultats expérimentaux, validés par leur utilité |
| doivent être aussi universels que possible | doivent être aussi universels que possible | relatifs à des cas particuliers |
| élégance = concision | élégance = concision | élégance = l'adéquation avec le modèle de l'utilisateur |
| Tend à rejeter l'IA | Tend à rejeter l'IA (statistiques) ou<br>Prétend appartenir à l'IA (Machine Learning) | Intègre naturellement l'IA |

**Figure 2.6 comparaison des méthodes d'apprentissage inductif (tiré de [Kodratoff 2001])**

La méthode d'apprentissage utilisée doit de plus permettre d'utiliser des données non numériques en entrée, car elle doit porter sur les concepts et règles utilisés par l'agent. La plupart des méthodes inductives utilisent directement les données observées, et donc des données numériques. Une méthode inductive permettant d'utiliser tous les types de données, puisque fondée sur le concept général d'objet symbolique, est l'Analyse de données symboliques ([Diday 2000b], [Bock 2000]). Le choix des exemples pertinents par calcul de

dissimilarités entre graphes de connaissances utilise ainsi des méthodes d'analyse de données symboliques pour permettre la comparaison des arbres de connaissances et obtenir les règles implicites les plus pertinentes possibles.

### 2.2.3. Combinaison

Par rapport aux autres méthodes d'apprentissage, notre apprentissage conciste donc en un apprentissage inductif (utilisant l'analyse de donnée symbolique) guidé par un apprentissage déductif (par application d'heuristiques). Cette vision permet de lier apprentissage déductif et apprentissage inductif d'une façon différente de celle proposée notamment par T. Michell ([Mitchell 1997]) qui utilise l'apprentissage déductif pour apprendre la fonction cible utilisée par l'apprentissage inductif.

Par ailleurs, le renforcement utilisé dans notre modèle par les émotions pour favoriser les liens conduisant aux situations souhaitées est très proche de l'apprentissage par renforcement classique (voir notamment [Sutton et Barto 1998]), en incluant un facteur de stabilité qui rend l'apprentissage (la modification du lien) progressivement moins important pour stabiliser le lien. Ce renforcement permet lui-même de guidée les raisonnements déductifs de l'agent.

## 2.3. Modèles d'agent

Comme nous l'avons vu en introduction, les agents sont une application particulièrement adaptée à notre modèle d'apprentissage combiné avec raisonnement sur des connaissances homogènes. L'agent évolue nécessairement dans un environnement qui peut être changeant ou inconnue, rendant l'adaptabilité de l'agent à celui-ci particulièrement importante pour gagner en efficacité. La possibilité d'expérimenter les nouvelles heuristiques créées sur un environnement toujours présent est également une différence fondamentale et un atout important par rapport à des applications d'apprentissage classiques sur un ensemble fixe de données. Enfin, le domaine des agents présente l'avantage d'avoir une vision globale du processus cognitif. Alors que la majorité des autres branches de l'IA s'intéresse à un sous-domaine particulier, le domaine des agents est contraint, de part son objectif même, à s'intéresser à l'ensemble des composants et à leur interaction. Or l'apport de notre modèle se situe précisément dans cette intégration de trois types d'apprentissage différents et d'un type de raisonnement (heuristique et réflexif) sur des connaissances homogènes. L'intérêt de cette

unification des théorie dans l'approche agent est notamment soulignée par P. Wang ([Wang to appear] p.26).

Les domaines d'application des agents sont également particulièrement adaptés. Les robots, naturellement, du fait de la complexité et de l'évolution de leur environnement. Mais aussi les jeux du fait de l'importance de l'apprentissage de concepts et d'heuristiques adaptées pour s'améliorer. Mais également des domaines comme la négociation automatique ou les formations de coalitions. Ainsi, dans un modèle de formation de coalition où chaque agent cherche à maximiser sa propre utilisé (comme dans [Caillou*, et al.* 2001, 2002b, c, a, 2003] ou [Aknine et Caillou 2004b, a]), l'identification des préférences des autres agents permet d'améliorer grandement sa stratégie au niveau des offres envoyées et surtout de leur destinataire. Apprendre des heuristiques afin de tenter d'identifier des groupes de préférences standard et en déduire un comportement adapté aurait donc un grand intérêt pour chaque agent.

Notre approche est complémentaire et non substitutive aux modèles d'agent actuels et une analyse détaillée des modèles existant ne permettrait pas de mieux la situer. Les différences fondamentales se situent en effet au niveau de la représentation des connaissance, qui n'est pas une clef des modèles d'agents puisque ceux-ci ont généralement pour objectif de décrire comment utiliser les connaissances. Notre approche d'apprentissage et de raisonnement vient également compléter les approches de contrôles actuels qui, comme nous allons le voir, peuvent très bien s'inclure dans notre modèle d'agent.

### 2.3.1. Architecture

Le système de contrôle que nous proposons se rapproche par sa décentralisation de l'architecture de subsomption proposée par R. Brooks ([Brooks 1999]). Notre objectif est toutefois de conserver des règles explicites pour pouvoir les interpréter et les modifier alors que la subsomption ne fonctionne que de façon implicite. De plus l'agent peut disposer de règles de fonctionnements globales tant qu'elles sont intégrées et analysables comme les autres dans le graphe. De telles règles sont utilisées notamment par [Wooldridge et Parsons 1998] qui présentent un modèle d'agent BDI utilisant des métarègles de contrôle de la reconsidération des intentions.

L'objectif de notre modèle est complémentaire des modèles d'agent existant. Les architectures actuelles proposent des règles de raisonnements et de traitements des objectifs (voir [Wooldridge 1999]). Notre modèle propose un mode de sélection des règles et de

création de concepts, perceptions et actions. Il pourrait par exemple très bien fonctionner avec des règles de gestion des objectifs et des plans de type BDI ([Bratman, *et al.* 1988]), a condition de conserver l'ensemble des concepts et des règles dans le même graphe. Les différences entre croyances, plans et intensions n'ont aucune raison d'être physiques et peuvent être purement sémantique dans le graphe.

Notre objectif étant l'adaptabilité de l'agent à l'environnement, notre agent peut également être rapproché des *animats* [Meyer et Wilson 1991]. Ceux-ci peuvent être défini comme des animaux simulés ou des robots chez qui on étudie les mécanismes qui génèrent les comportements permettant une adaptation dans un environnement plus ou moins imprévisible. Notre objectif cognitif d'interaction avec l'environnement et de raisonnement explicite nous situent dans l'approche dynamique des animats, par opposition à l'approche connexionnistes [Guillot 1999]. Par opposition à une modélisation sous forme de réseaux de neurones, l'approche dynamique considère en effet l'animat et son environnement comme deux systèmes dynamiques qui modifient en permanence leurs états respectifs. Les animats sont en particulier un des domaines les plus actifs pour l'intégration des émotions comme facteur de décision en intelligence artificielle [Canamero 1999].

### 2.3.2. Emotions

L'étude des émotions s'est récemment développée en intelligence artificielle ([Petta et Trappl 2001], [Scheutz 2002]). Elles peuvent être utilisées de deux façons.

La première utilisation consiste à essayer de détecter les émotions humaines pour faciliter la communication et la compréhension lors des interactions homme-machine [Pitrat 1997]. L'identification et la restitution d'émotions permettent en effet d'interpréter ce que l'utilisateur désire et de réagir en conséquence. Cette problématique a été en particulier popularisée par le robot Kismet du M.I.T. [Breazeal 2000].

Une deuxième façon d'aborder le problème est de s'intéresser aux fonctions internes des émotions et à leur fonctionnement, c'est-à-dire d'étudier et de reproduire leur influence sur la cognition et le raisonnement humain [Canamero 1998, 1999]. Moins développée, cette approche s'appuie en particulier sur les travaux réalisés en neurobiologie par A. Damasio ([Damasio 1995], [Damasio 1999]). Cette approche a notamment été utilisée par [Canamero 1997, Velasquez 1997] et [Velasquez 1998]. Dans ce dernier système par exemple, l'utilisation d'émotions permet d'orienter en biaisant le comportement d'agents logiciels ou de robots alors même que le contrôle se fait de façon décentralisée dans une architecture proche

de la subsomption. C'est cette approche que nous allons adopter, en utilisant les émotions comme guide, à la fois pour le comportement, et pour l'apprentissage.

### 2.3.3. Utilisation d'agents et jeux en temps réel

Les jeux ont toujours été une application privilégiée aussi bien des agents que de l'intelligence artificielle en générale. L'utilisation d'agent a actuellement tendance à s'accroître du fait de l'évolution dans les jeux étudiés ([Schaeffer et Herik 2002],[Herik*, et al.* 2002]). Alors que les jeux à information parfaite (Dames, Echec [Campbell*, et al.* 2002], Go [Bouzy et Cazenave 2001], Sokoban [Junghanns et Schaeffer 2001]) sont toujours les plus étudiés, les jeux à informations imparfaites (Poker [Billings*, et al.* 2002]) le sont de plus en plus et on commence à s'intéresser aux jeux avec interactions et en temps réel. Ceux-ci, pourtant fortement présents au niveau industriel (ils représentent la majorité d'un marché du jeu vidéo en forte expansion), ont été très peu étudiés et représentent un secteur de recherche potentiel important ([Laird et Lent 2000]). Le faible intérêt qu'ils ont suscité jusqu'à présent provient de l'approche différente qu'ils nécessitent. Pour les jeux traditionnels, le programme dispose pour choisir le coup suivant d'une information fixe (complète ou non) et d'un certain temps pour se décider, ce qui lui permet éventuellement de développer un grand nombre de possibilités, d'appliquer une fonction d'évaluation et ainsi de choisir le meilleur coup. Au contraire, dans un jeu en temps réel, le programme doit en permanence percevoir, choisir et agir. Il doit donc s'adapter en permanence aux changements de l'environnement et réagir à un flux continu d'informations pour réaliser un flux continu d'actions. Cette interaction accrue avec l'environnement peut être combinée à une interaction accrue avec d'autres acteurs, humains ou artificiels. Alors que les jeux traditionnels réduisent l'interaction au coup joué sur le plateau de jeu, de nombreux jeux commerciaux nécessitent une interaction constante avec d'autres caractères, que ce soit pour s'allier, commercer ou même négocier. Ces interactions importantes, que ce soit avec l'environnement ou avec d'autres acteurs, rendent l'utilisation d'agents particulièrement adaptée. Un agent peut être défini comme un système informatique situé dans un environnement et qui est capable d'agir de façon autonome et flexible sur cet environnement pour atteindre ses objectifs prédéfinis ([Wooldridge 2001]). Par rapport à un programme classique, les caractéristiques principales d'un agent peuvent être reprises de [Ferber 1995] :

- Il est capable d'agir dans un environnement.
- Il peut communiquer directement avec d'autres agents.

- Il est mû par un ensemble de tendances.
- Il possède des ressources propres.
- Il est capable de percevoir.
- Il possède une représentation partielle de l'environnement.
- Il possède des compétences.
- Son comportement tend à satisfaire ses objectifs en tenant compte des ressources et des compétences dont il dispose, en fonction de ses perceptions, de sa représentation et des communications qu'il reçoit.

Tous ces éléments répondent aux objectifs et aux contraintes des jeux en temps réels avec interactions : perception et action dans un environnement qu'il ne connaît que partiellement, des tendances pour le guider et une capacité à communiquer avec les autres agents.

La principale application des agents dans le domaine du jeu reste la Robocup et ses extensions ([Stone et Sutton 2001]), qui permettent de confronter aussi bien des robots que des agents logiciels. En dehors de cette application spécifique, leur utilisation reste toutefois marginale.

Le jeu mono agent de Tetris, par exemple, a été très peu étudié du fait de son aspect temps réel. Seules quelques analyses ont montré sa complexité ([Bertsekas et Tsilsiklis 1996],[Harada et Russel 1999]), en particulier en démontrant qu'indépendamment de toute considération temporelle, le jeu est NP-complet ([Demaine*, et al.* 2002]). La façon de jouer des joueurs humains, et en particulier l'utilisation épistémique des actions (et non dans un but pragmatique pour se rapprocher d'un objectif) a par ailleurs été étudiée en sciences cognitives par [Kirsh et Maglio 1994].

# Chapitre 3
# Présentation du modèle et exemple

## 3.1. Introduction

### 3.1.1. Objectif

Notre objectif est de réaliser un agent capable de s'adapter à l'environnement. Pour cela, il doit pouvoir découvrir et utiliser efficacement de nouveaux concepts et heuristiques. L'étude de l'état de l'art nous a montré que de nombreux systèmes existent pour induire, déduire et renforcer, mais sur des représentations hétérogènes et incompatibles. Nous cherchons donc ici à développer un modèle d'agent permettant ces trois types d'apprentissage grâce à une représentation homogène des connaissances. En analysant son graphe de connaissance, l'agent induit de nouveaux concepts, à partir desquels il déduit comment les utiliser et enfin le renforcement lui permet d'utiliser les concepts et les règles les plus pertinents en fonction des circonstances. Il peut ensuite raisonner à partir de sa nouvelle sémantique avec ses nouvelles règles, tout en continuant ses différents apprentissages, construisant ainsi des règles et des concepts de plus en plus utiles et adaptés.

### 3.1.2. Exemple

Nous avons déjà vu l'intérêt de l'apprentissage de concepts et heuristiques adaptés au domaine dans des jeux tels que Tetris. Pour illustrer notre modèle, nous l'avons appliqué à une version simplifiée de Tetris reprenant ses caractéristiques principales permettant de montrer l'efficacité de notre modèle. Pour faciliter la compréhension de celui-ci, de nombreuses applications à l'exemple seront données comme illustration au cours de ce mémoire. Pour bien les différencier de la présentation théorique, dans ce chapitre elles apparaîtront dans un cadre et pourront être cités explicitement dans le texte : voir cadre 3.1.

**Cadre 3.1 : exemple**

Les cadres-exemples ont pour but d'illustrer la présentation théorique au travers d'une application pratique du modèle présenté.

### 3.1.3. Plan du chapitre

Pour avoir une vision globale du modèle, nous allons présenter ici tous ses éléments. Le contexte est tout d'abord décrit au travers de l'environnement. Puis nous présentons la structure de l'agent dans cet environnement. Une bonne compréhension des règles et concepts utilisés nécessite ensuite de préciser la sémantique de base utilisée par l'agent.

Sans apprentissage, l'agent doit pouvoir raisonner. Ce fonctionnement de base est présenté en section 3.4. Pour s'adapter, l'agent doit induire à partir de ses connaissances, et en particulier à partir des connaissances acquises par interaction avec l'environnement. La méthode d'induction par analyse de données symboliques est donc présentée. L'induction permet de créer les concepts pertinents. La déduction lui permet ensuite de construire les règles associées. Les émotions présentées ensuite permettent à l'agent d'intégrer ces règles et de le guider indépendamment de sa sémantique en constante évolution.

Enfin, une vision globale de l'apprentissage combiné est donnée pour justifier la complémentarité et l'interdépendance des différents modules.

## 3.2. Environnement

Par définition, un agent se situe dans un environnement avec lequel il interagit. C'est une des principales différences avec les programmes classiques, qui peuvent exister pour eux même. Par exemple, un programme de résolution de problème n'a pas besoin d'environnement, il peut se contenter de raisonner sur ses connaissances et le problème qu'il doit traiter. Au contraire, un agent évolue au sein d'un environnement qu'il perçoit et sur lequel il peut (généralement) agir.

Disposer d'un environnement présente de nombreux avantages pour l'agent. En particulier, il peut y expérimenter les règles qu'il découvre ou imagine. Il évite ainsi le problème du surapprentissage classique en apprentissage inductif : quand les données sur lesquelles on réalise l'apprentissage sont en nombre limité, on court toujours le risque de trop

apprendre, c'est-à-dire d'apprendre des règles spécifiques à l'échantillon étudié. L'agent, lui, peut toujours collecter de nouvelles données dans l'environnement pour tester ses théories.

En contrepartie, l'agent ne dispose pas directement d'une connaissance complète et symbolique du monde environnant : il le perçoit. Cette perception peut être plus ou moins de haut niveau. Une perception de bas niveau signifie que l'information dont dispose l'agent pour raisonner est constituée de signaux bruts : la position des pierres pour un jeu de Go ou de Dames, la valeur du pixel observé pour la vision (comme pour Tetris, voir notre exemple en cadre 3.2). Au contraire, une perception de haut niveau fournit des informations symboliques directement utilisables pour le raisonnement : chaînes ou yeux au Go, types de Pièces et formes de mur à Tetris. De nombreux intermédiaires sont possibles. L'essentiel est que plus une perception est de haut niveau, plus elle est dépendante de l'environnement et donc de la connaissance qu'en a son concepteur. Percevoir les concepts utiles pour le raisonnement nécessite de connaître l'environnement à l'avance. De plus, une perception symbolique de haut niveau limite souvent les concepts utilisés par l'agent à ceux perçus, l'intérêt de l'apprentissage de concepts encore plus adaptés étant considéré comme marginal.

Un agent a une connaissance partielle de son environnement avant de le découvrir. Même son concepteur peut être incapable de dire quels types d'objets un agent va percevoir et donc ce qu'il lui faudra percevoir. D'où l'intérêt de laisser l'agent lui-même construire ses perceptions haut-niveau.

Par exemple, à Tetris, l'agent ne perçoit qu'un pixel à la fois (voir cadre 3.2). Apprendre à percevoir des pièces et des formes de mur permet à l'agent de construire des raisonnements plus abstraits tout en gardant sa capacité d'adaptation si l'environnement évolue.

**Cadre 3.1 : Objectif et environnement de l'exemple**

**Objectif de l'exemple**

L'objectif de cet exemple est de fournir une application illustrant la méthode d'apprentissage et de raisonnement proposée. Pour obtenir un problème suffisamment simple pour être décrit rapidement, nous avons choisi une simplification du jeu de Tetris. La plupart des éléments de base ont été conservés, afin que le joueur ait toujours besoin d'adapter sa sémantique et son comportement pour apprendre à jouer efficacement.

**Description du jeu**

L'environnement du jeu est représenté figure 3.1

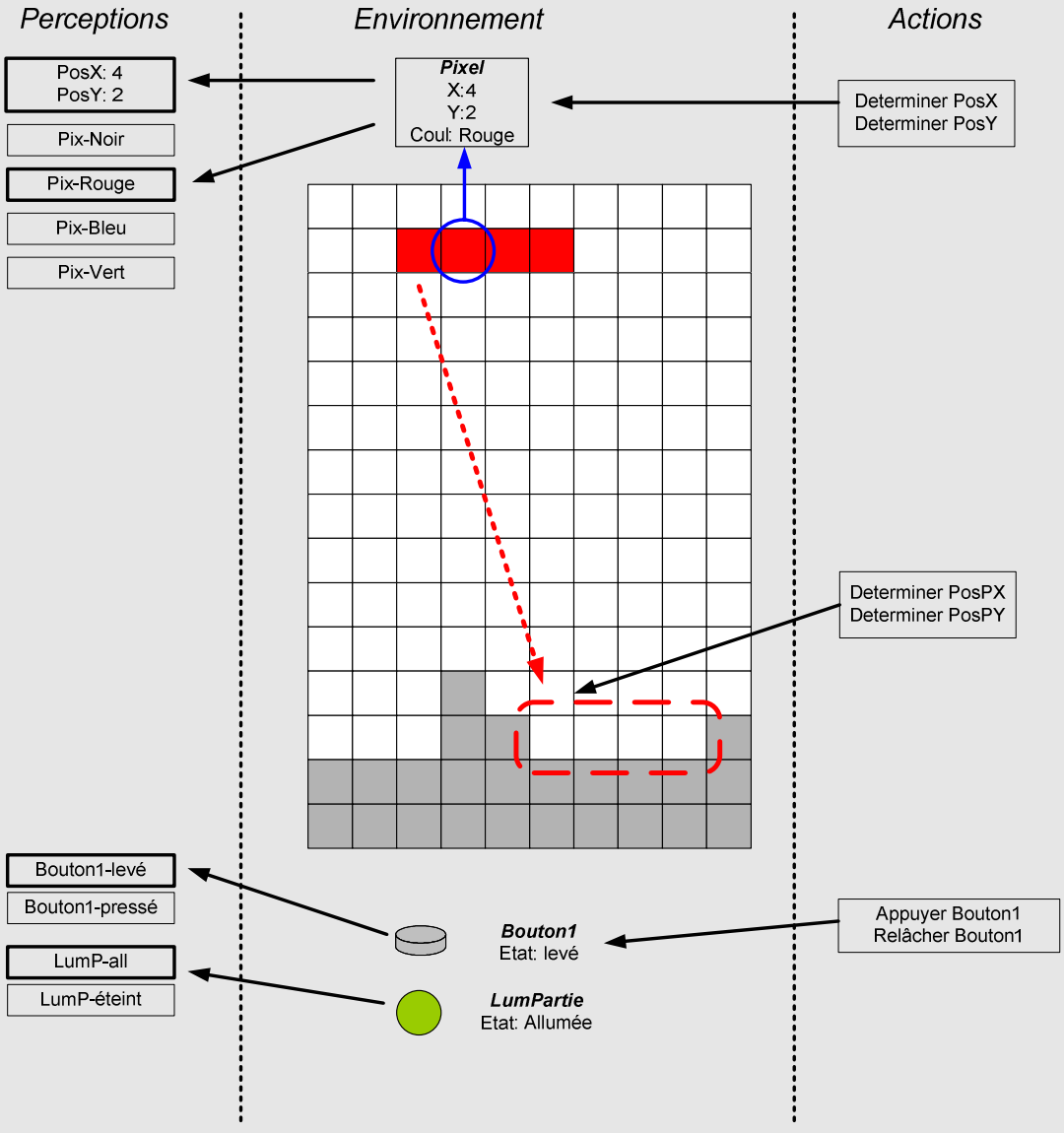



**Figure 3.1 : Environnement exemple**

Globalement, le jeu se déroule ainsi : le joueur initialise la partie. Ensuite, une pièce apparaît en haut de l'écran et un mur en bas. Le joueur doit trouver le plus rapidement possible une zone du mur correspondant à la forme de la pièce. Dès qu'il a placé la pièce à un endroit satisfaisant du mur, la partie se termine et le joueur a d'autant plus de points qu'il a été rapide.

Nous présentons ici la version de base de notre Tetris simplifié, qui sera modifiée par la suite pour étudier les influences de ces modifications sur le comportement de l'agent. Plus précisément le jeu de base peut se définir ainsi:

Le jeu se déroule dans un cadre de 10*20 briques.

Le joueur perçoit une brique (coordonnées *PosX*, *PosY*)

Il peut également réaliser une action (l'équivalent de placer son doigt ou un curseur) sur une autre case (coordonnées *PosPX*, *PosPY*). Cette action est réalisée à ces coordonnées en appuyant sur le *Bouton1*, ce qui lui permet soit de débuter la partie (s'il est aux coordonnées 0,0), soit d'essayer de placer la pièce (si la partie est déjà en cours).

Pour débuter la partie, l'agent doit appuyer sur le bouton en 0,0.

Lorsque la partie commence, une des 7 pièces possibles de Tetris apparaît aux coordonnées 3,1 et un mur (parmi 5 possibles) apparaît en bas du cadre.

Les 7 pièces ont toutes une couleur différente, le mur est gris.

Une lumière (*LumPartie*) est allumée si la partie est en cours, éteinte sinon.

Il faut bien différencier une brique, qui correspond à une case de l'environnement, d'une pièce qui correspond à un ensemble de briques adjacentes.

**Comment l'agent doit apprendre à jouer**

L'agent ignore quels sont les types de pièces, les murs et leurs positions respectives. Par contre il connaît les règles du jeu. Au départ, il doit donc réaliser un balayage systématique pour trouver la pièce et le mur, faire leur description complète et en déduire où placer la pièce pour finir la partie. Par la suite, pour s'adapter, il doit apprendre en particulier les différents types de pièces et de mur et trouver des heuristiques rapides pour les trouver et les identifier. Ayant modifié sa sémantique (il doit raisonner avec ces nouvelles pièces et non avec les notions génériques de « pièce » et « brique ») et son comportement (nouvelles perceptions utilisées à la place des anciennes), il doit pouvoir raisonner à partir de là et en déduire directement où placer

## 3.3. Structure de l'agent

### 3.3.1. Présentation globale

La structure de l'agent est pensée pour se situer dans un environnement et pour utiliser une représentation homogène des connaissances : Il est composé de connaissances (nœuds du graphe) reliées entre elles par des liens. Il communique avec l'environnement au travers des actions et des perceptions. Les émotions agissent sur son graphe de connaissances et sont perçues par ses sens (qui perçoivent les variations des niveaux des émotions).

Le schéma global de l'agent est le suivant:

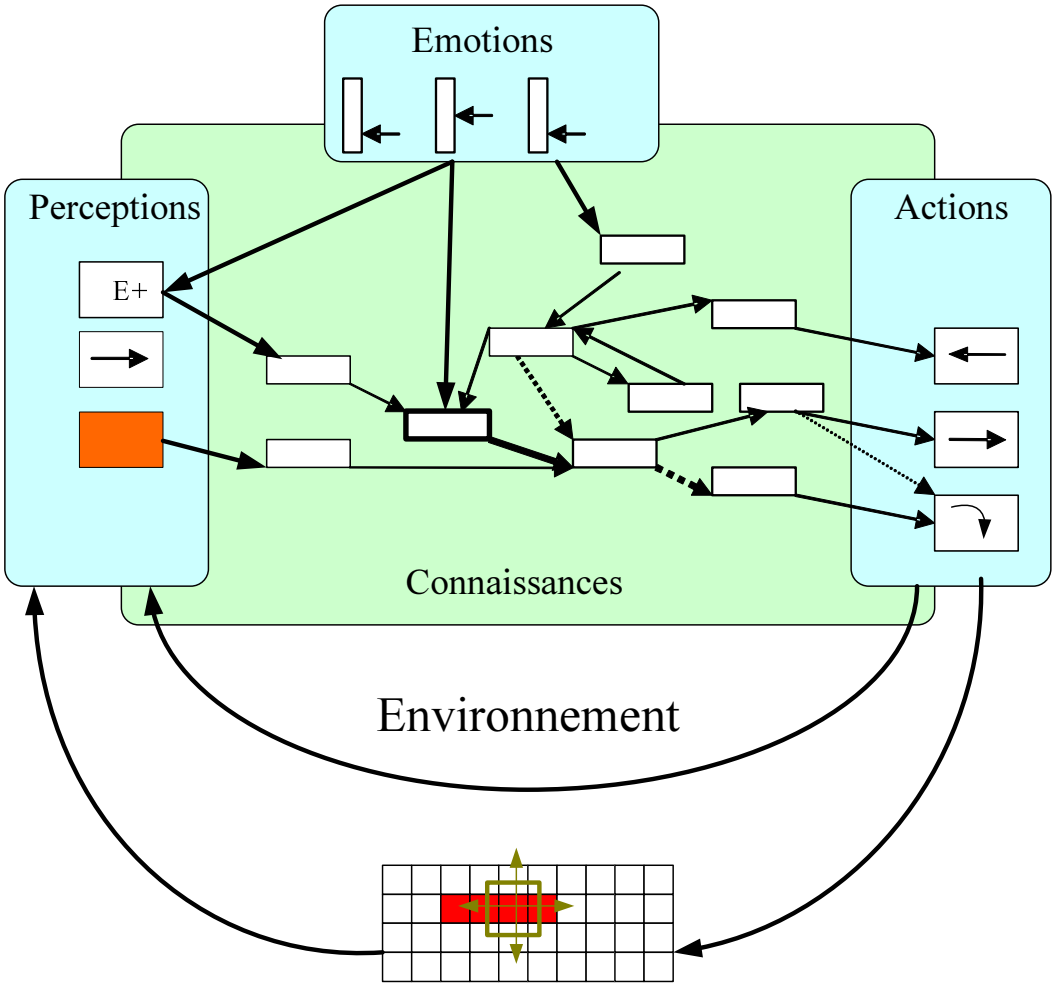



**Figure 3.2 : Schéma global de l'agent**

### 3.3.2. Interaction avec l'environnement

Les contacts avec l'environnement se font grâce aux perceptions et aux actions de l'agent. Un sens (perception) va activer une connaissance spécifique de l'agent de façon d'autant plus importante que l'évènement correspondant au sens est présent dans l'environnement. Par exemple, un sens sensible à la couleur rouge activera d'autant plus la connaissance correspondant à la vision de la couleur rouge que celle-ci est présente dans le domaine observé. Un sens dispose en plus d'une règle explicite qui lui permet d'interpréter sous forme symbolique les différentes activations.

Inversement, une action sera effectuée d'autant plus que la connaissance correspondant à cette action sera activée. Pour une action binaire comme appuyer sur un bouton, l'action sera effectuée si la connaissance correspondante dépasse un certain seuil. Pour plus de détails sur les perceptions et action, voir la section 8.3.

**Cadre 3.2 Agent et Environnement Exemple**

**Perceptions et actions de l'agent**

Il est important de réaliser ce que perçoit l'agent sur l'environnement :

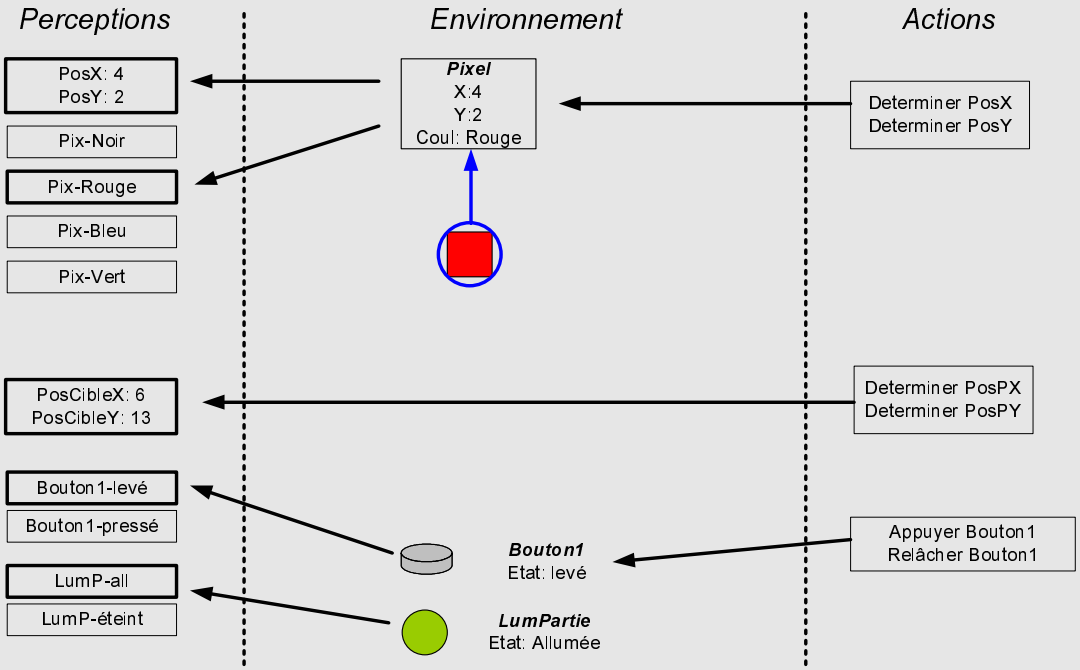



**Figure 3.3 Perception et actions de l'agent sur l'environnement**

La figure 3.3, par opposition à la figure 3.1 donnant une vision externe de l'environnement, permet de bien se rendre compte que l'agent ne perçoit qu'une brique à la fois. La recherche ou la description systématique d'une forme est donc particulièrement fastidieuse.

La perception des différents éléments peut se faire aussi bien de façon proactive que réactive : De façon réactive, la lumière va introduire une activation dans le graphe au niveau d'une connaissance particulière, *CLumPAllumée*. De façon proactive, une règle va regarder l'état de la connaissance *CLumPAllumée* et définir explicitement l'état du concept « *LumP* » comme étant « *Allumé* » (le fonctionnement des perceptions et action est présenté en détail au chapitre 0).

### 3.3.3. Cycle de fonctionnement

L'agent exécute un cycle d'activité de façon régulière. La périodicité de ce cycle est fixe. Les différentes activités réalisées lors de chaque période sont représentées sur le graphique suivant :

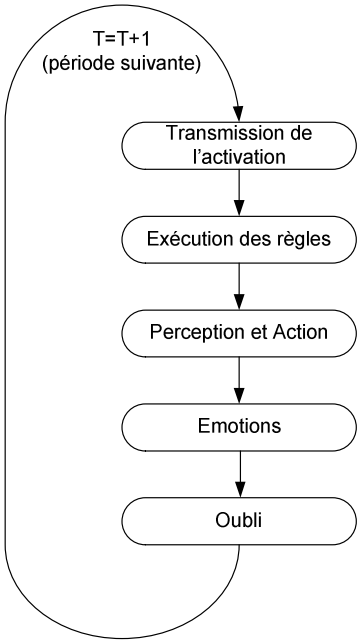



**Figure 3.4 : Cycle de fonctionnement de l'agent**

Le principe clef du contrôle est la transmission de l'activation : à chaque période, une connaissance transmet une partie de son degré d'activation aux connaissances auxquelles elle est liée. Ensuite son propre niveau d'activation diminue (sans cette actualisation, le graphe serait rapidement saturé et toutes les connaissances activées au maximum). Pour plus de détails sur la transmission de l'activation, voir la section 5.2.

De l'activation est introduite dans le graphe de deux façons:

- Par les émotions
- Par les perceptions

### 3.3.4. Graphe de connaissance

L'utilisation d'un graphe de connaissance orienté non typé permet d'obtenir une représentation homogène quel que soit le type de connaissance.

L'agent est composé d'un grand nombre de connaissances liées entre elles par des liens d'activation. L'intensité des liens permet de définir des concepts fluides autour des connaissances contenant la connaissance elle-même ainsi que les connaissances auxquelles elle est le plus liée, en particulier ses attributs. Alors qu'une connaissance est représentée par un rectangle à bords carrés, on utilise une Frame (à bords ronds) pour un concept centré sur une connaissance. Les deux notations suivantes sont donc équivalentes :

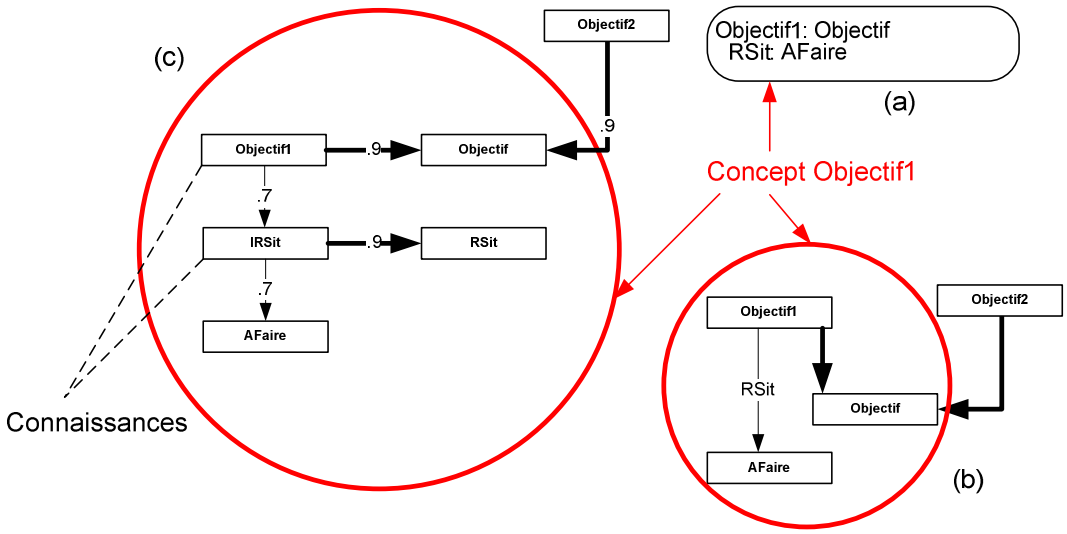



**Figure 3.5 : Trois représentations du concept *Objectif1*, de la plus simple (a), à la plus précise (c)**

Les liens épais, correspondant à une intensité élevée, sont des liens de type implicite *Est-Un* (est une instance de). Dans une Frame, *Objectif1* : *Objectif* signifie que *Objectif1* est un *Objectif*. *Objectif1* va donc bénéficier de tous les attributs de *Objectif*, et en particulier des règles qui lui sont associées. Les liens d'un autre type que l'identité, tels que la situation *AFaire* d'un *Objectif*, sont représentés grâce à des connaissances intermédiaires liées à la fois à la connaissance correspondant au type de l'attribut (*RSit*) et à la connaissance cible de l'attribut (*AFaire*). Cette connaissance, désignée par *IRSit*, est comme toute autre connaissance le centre d'un concept. Celui-ci peut être vu comme le concept correspondant à une liaison de type correspondant vers la cible désignée, ici une liaison de type *RSit* ayant pour cible *AFaire*.

Une notation *Objectif1->RSit* signifie qu'on utilise l'attribut *RSit* associé à la connaissance *Objectif1*. On obtient donc la connaissance correspondant à la situation actuelle de *Objectif1*.

La lettre I au début d'un nom signifie « instance de ». Par exemple *IRSit* est une connaissance Instance de *RSit* (il s'agit d'une relation de type Situation).

Pour plus de détails sur les connaissances, voir le chapitre 0.

**Cadre 3.3 Graphe de connaissances de l'agent**

Un exemple de concepts issus du graphe de connaissance de l'agent est présenté figure 3.6.

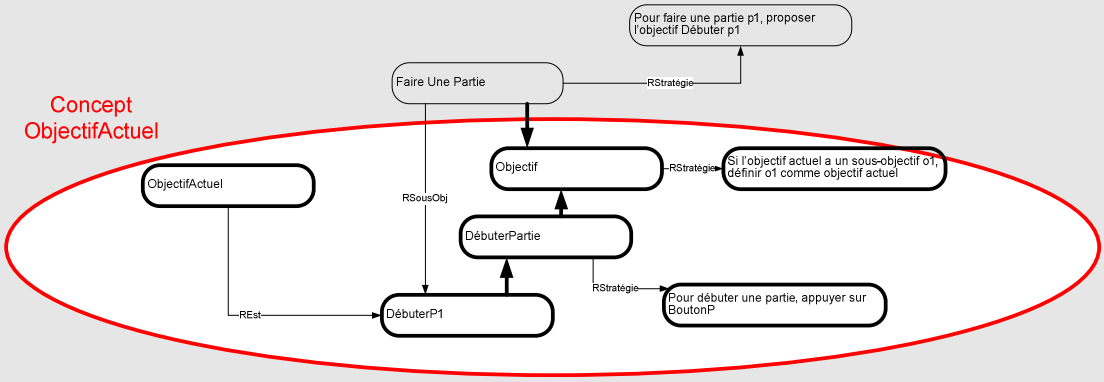



**Figure 3.6 : Exemple de sous-graphe correspondant à la gestion d'un objectif**

Dans cet exemple, l'objectif actuel est défini explicitement comme étant *DébuterP1* (par la relation *REst*), qui est une instance de *DébuterPartie*, lui-même instance de *Objectif*. Le concept général d'*Objectif* est lié à la règle « *Si l'objectif actuel a un sous-objectif o1, définir o1 comme objectif actuel* », tandis que celui plus précis de *DébuterPartie* a comme stratégie « *Pour débuter une partie, appuyer sur BoutonP* ». Le fait de définir *DébuterP1* comme objectif actuel permet d'inclure l'ensemble de ses concepts et règles dans le concept fluide « Objectif Actuel ». L'évolution et l'utilisation des règles de cet exemple sont décrites avec la gestion des objectifs en section 9.2.2.

Pour pouvoir être utilisées et analysées, les règles sont décrites de façon explicite dans le graphe. Un exemple de règle (très simple) est donné en figure 3.7.

```
Regle1: RegleSiAlors
  Si: Test1: TestEstDansConcept
    Sujet: varLitParam1: VarLitParam
      Sujet: varUn1: VarUn
        Sujet :  Pièce
        Param: Couleur
      Quoi: Rouge
  Alors: defInstance1: DefInstance
    Sujet: varUn1
    Quoi: Barre
```

**Figure 3.7 : Représentation explicite de la règle « Si la couleur d'une pièce est rouge, alors c'est une barre »**

## 3.4. Sémantique de base

### 3.4.1. Intérêt de la sémantique

Pour réaliser des raisonnements complexes, l'agent dispose d'une sémantique de départ. Il peut ensuite la faire évoluer lui-même, mais celle-ci permet de définir les concepts de base qu'il va utiliser. Ceux-ci sont définis le plus généralement possible, l'objectif étant toujours de lui laisser les spécialiser lui-même par la suite pour s'adapter à l'environnement. D'autres concepts peuvent également être donnés, liés spécifiquement à l'application de l'agent, comme c'est le cas pour notre application (voir cadre 3.4).

Pour bien comprendre la suite de la description de l'agent, il est nécessaire de décrire les concepts de base qui sont utilisés :

- Concept : type le plus général
- Situation : une situation correspond à une condition. Par exemple, « *La lumièreP est allumée* » ou « *La pièce est une Barre* ». Cette connaissance permet de mettre en relation rapidement des expressions ayant un élément en commun. Elle permet également de transmettre l'activation aux connaissances qui sont susceptibles d'être utilisées si cette situation se présente.
- Perception : Une perception (ou sens) permet d'identifier l'existence d'un élément. Elle est définie par ce qu'elle recherche (une Pièce, une Barre, la situation d'une partie,…).
- Action : Une action permet d'agir sur l'environnement ou les connaissances. Elle est définie par ce qu'elle fait (appuyer sur un bouton, réaliser un déplacement aléatoire,…).
- Objectif : Un objectif correspond à un but. Il est défini par une condition d'atteinte (débuter une partie, déduire une perception, …).
- Relation : Une relation permet de définir un attribut d'un certain type.
- Exécutable : Règles et méthodes. Les exécutables peuvent être interprétés par le système pour être exécutés. Les règles peuvent être sélectionnées et appliquées automatiquement lors de la phase d'exécution des règles de l'agent. Les méthodes, au contraire, doivent être appelées par une autre règle pour être

exécutée. Les exécutables sont définies par ce qui est interprété (la règle en elle-même).

- Descriptions : les descriptions correspondent à des règles de descriptions de l'environnement ou de l'agent. La définition est écrite sous forme interprétable, elle n'est pas exécutée. Elle a pour seul objectif d'être analysée pour en déduire des règles ou méthodes exécutables (par exemple les règles du jeu sont données sous forme de Description à l'agent).

**Cadre 3.4 Connaissances de départ spécifique à l'application**

Le but de cette application est de montrer comment l'agent s'adapte, et non de faire un agent complètement générique pour toutes les situations. C'est pourquoi de nombreuses connaissances spécifiques au domaine lui sont fournies, y compris les règles du jeu. En particulier :

**Objets**

L'agent connaît en particulier les concepts de *Pièce*, *Brique*, *Forme* et les différentes couleurs

**Règles et Méthodes**

L'agent a une méthode spécifique lui permettant de calculer automatiquement la forme complémentaire d'une autre forme. Cela lui permet à partir d'une pièce de déduire automatiquement la forme de mur correspondant qu'il doit rechercher pour que cela corresponde.

**Description**

L'agent dispose de deux règles de description du monde au départ :

Une description qui lui indique la règle du jeu : Pour gagner, placer la pièce sur une forme complémentaire du mur.

Une description qui lui indique comment débuter : Pour débuter la partie, appuyer sur le bouton en 0,0.

## 3.5. Raisonnement

### 3.5.1. Raisonnement implicite

Un réflexe "Si la lumière est allumée, appuyer sur le bouton" peut être simplement représenté par le lien suivant:


**Figure 3.8 : Réflexe « Si la lumière est allumée, appuyer sur le bouton »**

A chaque fois que la lumière sera allumée, on aura alors activation de la connaissance *Lumière*, puis de *Situation1*, puis de *AppBouton*, ce qui entraînera l'action souhaitée.

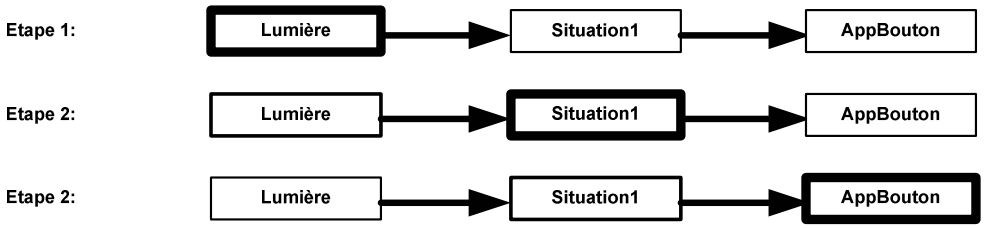



**Figure 3.9 : Transmission de l'activation dans la règle réflexe**

La perception de la lumière commence par activer *Lumière,* à l'étape suivante *Lumière* transmet son activation à *Situation1*, qui finit par transmettre à *AppBouton*. Si *AppBouton* est suffisamment activé, l'agent va effectivement appuyer sur le bouton.

Les émotions vont elles aussi introduire de l'activation dans le graphe pour favoriser certaines connaissances, à la fois directement et par l'intermédiaire de connaissances correspondant au souvenir d'émotions perçues.

En plus des réactions induites par les sens, l'agent peut agir de façon proactive grâce à la transmission de l'activation déjà présente dans le graphe. Les liens entre les connaissances ne sont pas tous orientés des sens vers les actions, avec quelques connaissances intermédiaires. De nombreuses boucles peuvent être présentes au milieu, agissant tout à fait indépendamment des sens et des actions. Ainsi, un raisonnement consistera juste en une transmission d'activation dans le graphe, avec éventuellement création de nouvelles connaissances. L'agent ne réagit alors plus à un événement extérieur, mais à ses propres connaissances.

### 3.5.2. Raisonnement explicite

Pour permettre un raisonnement cognitif, à chaque période le programme choisit aléatoirement une règle qui est décrite de façon explicite dans le graphe et l'interprète dynamiquement. Le choix se fait avec une probabilité proportionnelle au degré d'activation de la règle. Ainsi, les règles les plus utiles à un moment donné en fonction de l'état mental de l'agent ont plus de chance d'être activées. La succession des règles explicites exécutées par l'agent forme son raisonnement.

### 3.5.3. Raisonnement construit

Les règles permettent ensuite de réaliser toutes les fonctions cognitives nécessaires au bon fonctionnement de l'agent. En particulier :

Les règles de guidage : le suivi des objectifs explicites permet à l'agent de réaliser des raisonnements construits et des actions complexes, comme initialiser un sens à partir d'un nouveau concept.

Les règles de monitorage : ces règles permettent à l'agent de contrôler ce qu'il fait, par exemple pour vérifier qu'il n'est pas parti en boucle

Les règles d'analyse : ces règles permettent de déduire et de compléter de nouveaux concepts. Cela peut se faire par déduction (comme déduire une règle de perception ou une hypothèse à partir d'une définition) ou par induction. Les fonctions d'induction, et en particulier l'induction par analyse de données symboliques, sont incorporées dans des connaissances interprétables. Elles sont donc utilisées explicitement par l'agent quand il le veut et portent sur les concepts qu'il définit.

**Cadre 3.5 Exemple de raisonnement**

**Vision haut niveau (raisonnement construit)**

Pour réaliser sa première partie, l'agent va se fixer successivement deux objectifs : débuter la partie, puis la gagner. Pour débuter la partie, l'agent a déduit, à partir de la règle du jeu, qu'il devait appuyer en 0,0, ce qu'il fait. Il vérifie que la partie est bien en cours et considère cet objectif atteint.

Pour gagner la partie, il commence par rechercher de façon systématique la pièce et le mur. Une fois qu'il a trouvé la pièce (une brique non noire et non grise), il se fixe pour objectif de la décrire entièrement pour pouvoir ensuite en déduire sa forme complémentaire. Il fait de même avec le mur, recherche une forme correspondante et place la pièce à cet endroit.

Une fois la partie finie, il va l'analyser. Pour cela il va notamment se fixer comme objectif d'induire un nouveau concept issu de cette partie puis d'en déduire les règles et des hypothèses associées.

**Vision règle explicite**

La réalisation de ce raisonnement se fait par application d'un grand nombre de règles. Par exemple, pour débuter la partie, l'agent applique les règles de suivi d'objectif, d'action sur le bouton, de sélection de l'emplacement, de modification de l'objectif, …

**Vision bas niveau (activation transmise et raisonnement implicite)**

Le fonctionnement de base fait que l'agent exécute un grand nombre de règles en les sélectionnant en fonction de leur degré d'activation, donc de l'attention qu'il leur porte. Une fois les éventuels paramètres choisis de façon probabiliste, des règles sont également sélectionnés en fonction du degré d'activation. La transmission de l'activation entre les connaissances permet d'appliquer avec une plus forte probabilité les règles utiles dans la situation actuelle.

Par exemple, lorsque l'objectif est « Débuter la partie », les règles appartenant à la stratégie de cette objectif, de même que les règles liées au concept de « Débuter » ou « Objectif » seront fortement activées. C'est le cas en particulier de la règle lui indiquant de se placer en 0,0 et d'appuyer sur le bouton, puis de la règle définissant l'objectif comme atteint si la partie est en cours, … Une fois que l'objectif a changé, ce sont d'autres règles qui sont activées et donc probablement utilisées.

## 3.6. Induction

Pour s'adapter à son environnement, l'agent doit créer de nouveaux concepts à partir de son expérience. Cette recherche est réalisée explicitement en appliquant une règle réalisant l'induction en fonction de ses objectifs. Par exemple, rechercher un nouvel objet pertinent, un objectif, …La recherche doit se faire sur le graphe de connaissances, indépendamment de toute sémantique (celle-ci pouvant évoluer).

L'Analyse de Données symboliques (ADS) permet d'induire sur n'importe quel type de données et est donc particulièrement adaptée dans notre cas – très éloigné des tableaux numériques généralement analysés.

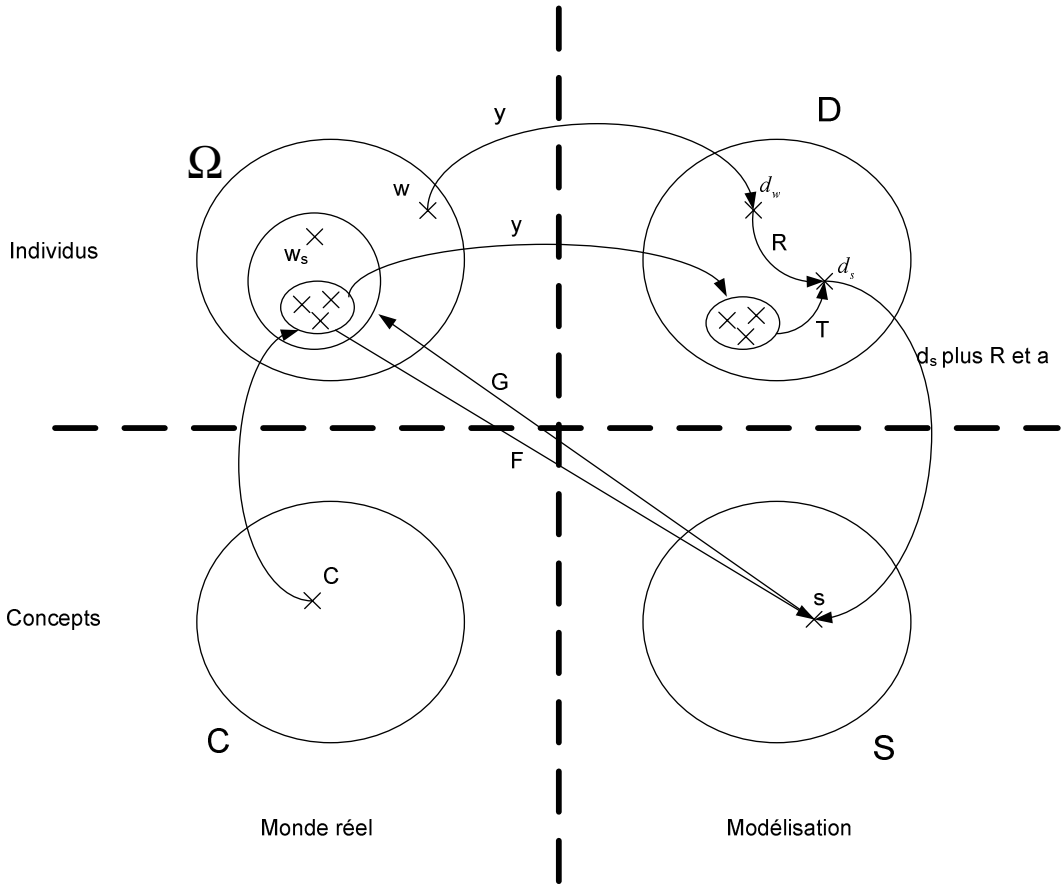



**Figure 3.10 Ensembles utilisés par l'analyse de données symboliques : l'ensemble des descriptions D permet de décrire les individus du monde réel éléments de Ω, tandis que l'ensemble des objets symbolique S permet de modéliser les concepts du monde réel éléments de C.**

Le cadre fourni par l'ADS permet de décrire la méthode utilisée (voir figure 3.10).

Le monde réel de l'ADS correspond au graphe de connaissance, la modélisation à sa modélisation.

Les individus étudiés sont donc les sous-graphes correspondant aux concepts.

En décrivant ces sous-graphes à partir de leur liens et en définissant une mesure de distance sur ce type de description, il est alors possible d'utiliser tous les outils offerts par l'ADS pour étudier les données non numériques.

On peut en particulier l'utiliser pour extraire un sous-ensemble pertinent des individus. La fusion des descriptions de ce sous-ensemble nous permet d'obtenir un nouvel objet symbolique correspondant au nouveau concept appris.

Une différence notable par rapport aux données utilisées traditionnellement en ADS provient du fait que le concept appris (le nouvel objet symbolique) est du même type que ceux étudiés à l'origine. Il est donc possible d'insérer un nouvel individu dans le monde réel (le graphe) correspondant à ce nouveau concept. Cela se fait au travers d'un nouvel opérateur, que nous définirons ici comme opérateur d' « individualisation ». Son rôle est en particulier ici d'introduire la définition explicite du concept qui n'est décrit que par ses liens dans l'objet fusionné initial.

Le nouveau concept construit reste toutefois un concept « brut » ne disposant que de sa définition et de quelques attributs. Pour être utilisé par l'agent, celui-ci doit déduire les règles nécessaires à son utilisation.

L'induction par ADS est décrite en détail au chapitre 0.

En plus de ses méthodes d'induction, la modélisation réalisée grâce à l'ADS nous permet également de visualiser les concepts actuellement utilisés par l'agent sous forme visuelle. Cette visualisation est particulièrement utile du fait de la difficulté de visualiser de façon claire des concepts utilisés alors que ceux-ci sont très nombreux, représentés sous forme relationnelle et selon une sémantique par définition changeante. Nous avons donc défini une méthode de visualisation dynamique d'objets symboliques basée sur une pyramide symbolique permettant de représenter une sélection de concepts et leur évolution (voir chapitre 0 pour plus de détails).

**Cadre 3.6 Exemple d'induction**

Après plusieurs parties, l'agent peut réaliser une induction sur le concept de « *Pièce* » pour en déduire un nouveau concept pertinent à partir des exemples dont il dispose. Cette induction se fait en plusieurs étapes :

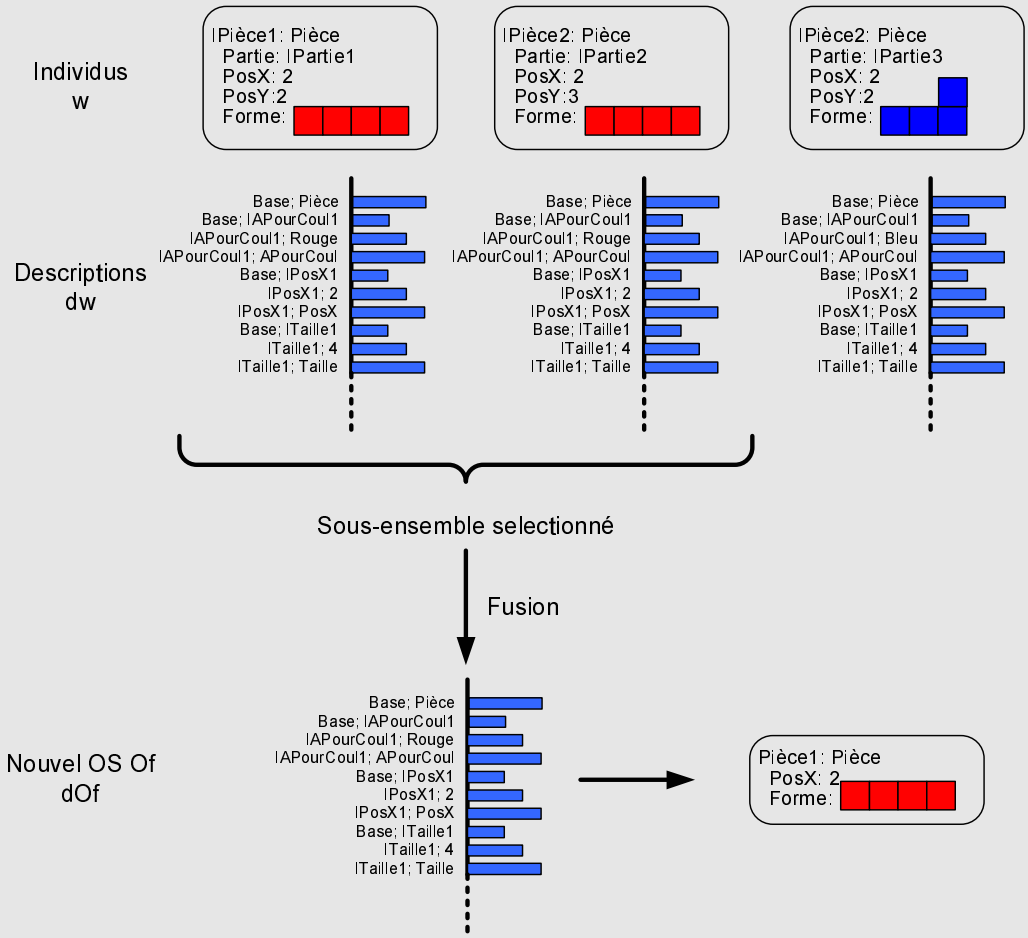



**Figure 3.11 Présentation globale de la méthode d'induction par Analyse de Données Symboliques**

Les graphes correspondant aux différentes pièces observées jusqu'à présent correspondent aux individus analysés du monde réel. Leur modélisation est issue de l'ensemble des liens composant les sous-graphes (modifiés pour éviter les redondances d'informations).

Un sous-ensemble d'individus similaires est sélectionné à partir de ces descriptions en maximisant à la fois la similarité et la taille du groupe. Ici, on sélectionne toutes les barres rouges observées.

L'ensemble des descriptions des pièces sélectionnées est fusionné pour obtenir la description du nouveau concept. Lors de cette fusion, seuls les liens communs aux différentes pièces sont conservés. Ici, principalement la forme (*barre*), la couleur (*rouge*) et le type (*Piece*). Les éléments particuliers (différentes positions, conditions d'observations, parties correspondantes), sont écartés.

La nouvelle description correspond à un ensemble de liens. Pour obtenir un nouvel individu, on réalise une individualisation : à partir des exemples et de la description, la fonction déduit les connaissances intermédiaires à ajouter et la définition du nouveau concept. Ici : « *Une Pièce est une Barre si elle est Rouge et comporte quatre briques alignées* ». Les connaissances intermédiaires sont nécessaires pour définir explicitement les attributs. Ici, la couleur rouge et la forme.

Le nouveau concept et sa définition sont introduits dans le graphe de connaissances et l'agent va ensuite pouvoir déduire les règles associées.

## 3.7. Déduction

### 3.7.1. Objectif

Seul, le nouveau concept est inutile pour l'agent. Ce dernier doit pouvoir construire des règles pour l'utiliser, construire des hypothèses qui lui permettent de profiter de cette nouvelle sémantique pour être plus efficace.

Ces règles de déduction doivent souvent être spécifiques au type de concept analysé ou construit. Ainsi, il y aura une règle spécifique pour déduire les situations impliquées par une perception nouvellement créée. Pourquoi des règles si spécifiques et donc dépendantes de la sémantique, alors que tous les autres éléments de l'agent (induction, émotion, structure sont fait pour être indépendants de la sémantique ? A l'opposé de tous ces autres composants, les règles d'analyse sont d'autant plus efficace qu'elles sont spécifiques et peuvent évoluer avec la sémantique. Si les règles d'origine sont crées par le concepteur, l'agent peut les faire évoluer ou en créer de nouvelles pour s'adapter aux changements de sa sémantique. L'intérêt de la représentation homogène reste entier celle-ci est toujours nécessaire pour que l'agent puisse à la fois interpréter et analyser les règles.

### 3.7.2. Règles spécifiques

Des règles spécifiques ont donc été définies ici pour déduire les règles de perception à partir d'un concept appris. Elles permettent également d'émettre quelques heuristiques sous forme d'hypothèses à tester. Si celles-ci sont valables suffisamment souvent, elles seront validées et utilisées en plus des règles strictes. Ce sont encore d'autres règles d'analyse qui exécutent cette vérification et la validation des hypothèses.

Les règles de déduction permettent également d'extraire les situations correspondant aux différentes perceptions et actions.

**Cadre 3.7 Exemple de déduction**

A partir du concept *Piece1* correspondant à une Barre rouge, l'agent peut déduire de nombreux autres concepts pour pouvoir l'utiliser efficacement.

Sa définition lui permet de déterminer une règle stricte de perception qui va vérifier si une *Pièce* à toutes les caractéristiques requises, et qui va alors la définir comme étant une *Barre*. Des situations correspondant aux différentes conditions unitaires sont créées. Par exemple, une situation correspondant à la situation « *Une pièce est rouge* » est créée.

Certaines hypothèses destinées à être testées vont également être générées. En particulier des hypothèses comme quoi une seule condition suffit à définir le concept. Par exemple, *si une Piece est rouge alors c'est une Barre*. Des règles correspondant à ces hypothèses sont créées et intégrées à la perception. Celles-ci ne vont toutefois pas modifier les connaissances mais uniquement comparer leur résultat à la définition stricte pour valider ou infirmer l'hypothèse. Une fois testées sur plusieurs exemples, les hypothèses peuvent être intégrées aux méthodes valides et utilisées directement.

## 3.8. Emotions et renforcement

### 3.8.1. Définition et objectif

L'agent a besoin d'un contrôle qui soit au maximum indépendant de la sémantique, car celle-ci va évoluer. Il a donc besoin de fonctions qui vont modifier sa façon de raisonner en fonction de l'état du graphe ou de l'environnement. Nous appellerons ces fonctions « émotions » du fait de la similarité fonctionnelle avec les émotions humaines.

Ces émotions vont fournir un critère externe pour guider l'agent et intégrer les nouvelles connaissances. Les concepts et règles utiles doivent en effet être utilisés plus souvent, éventuellement plus que ceux présents lors du lancement de l'agent. Inversement, les concepts découverts par les inductions et les règles créées par l'agent ne sont pas toujours utiles. Un des principaux problèmes des systèmes déductifs est de ne conserver que les concepts qui font gagner du temps au programme sous peine de se retrouver avec une surabondance de concepts qui va ralentir le raisonnement. Comment sélectionner automatiquement les règles et concepts utiles au bon moment ? C'est le rôle des émotions.

### 3.8.2. Premier avantage : intégrer

L'intégration des concepts utiles se fait par renforcement des liens entre les concepts utilisés lorsqu'une émotion est ressentie positivement et par inhibition (donc renforcement négatif) lorsqu'elle est ressentie négativement. Les connaissances affectées et le calcul du niveau de l'émotion dépendent de l'émotion elle-même.

Par exemple, l'émotion Concentration varie positivement lorsqu'un objectif est atteint. Un lien est créé (ou renforcé) entre l'objectif atteint et les règles et concepts qui ont été utilisés avec succès. Les règles et concepts utiles dans des circonstances particulières auront alors tendance à être réutilisés si des circonstances similaires (mais par obligatoirement identiques) se reproduisent.

### 3.8.3. Deuxième avantage : guider

L'intégration des règles est une première méthode indirecte pour guider l'agent avec les émotions. En effet, les règles et concepts provoquant des émotions positives sont renforcés et plus souvent utilisés, conduisant l'agent à reproduire des situations similaires.

De façon plus directe, les émotions peuvent introduire de l'activation directement dans le graphe, permettant de favoriser certains concepts ou règles. La Concentration active par exemple les règles de suivi d'objectif, ce qui permet à l'agent de contrôler ce qu'il fait et de suivre son plan.

### 3.8.4. Exemples d'émotions

Certaines émotions peuvent dépendre de l'environnement (comme gagner à un jeu), certaines peuvent être utilisées quelle que soit l'application, en fonction du comportement

souhaité de l'agent. Par exemple, la curiosité incite l'agent à découvrir de nouveaux concepts, l'ennui lui évite de rester inactif, …

**Cadre 3.8 Exemple d'émotions**

Les émotions permettent à la fois de guider l'agent et d'intégrer les connaissances, ces deux objectifs se déroulent de façon complémentaire :

**Intégration**

Une fois le nouveau concept de *Barre* induit et ses concepts associés créés, il faut les intégrer au graphe afin que ceux qui sont utiles soient utilisés et que les autres soient progressivement oubliés. Une règle utile pour atteindre un objectif, par exemple la règle correspondant à l'hypothèse « *Si la pièce est rouge, c'est une barre* », sera ainsi notamment associée à l'objectif « *Identifier la Pièce* ». Au contraire, d'autres règles à priori activées car très générales mais ne concernant en rien l'identification de la pièce, sont progressivement inhibées. De même, des règles spécifiques mais inutiles car moins efficaces, comme « *Si la pièce est composée de quatre briques alignées, alors c'est une barre* », sont progressivement inhibées (toujours par rapport à l'objectif « *Identifier la Piece* ») car l'identification à partir de la couleur est toujours plus rapide.

**Contrôle**

Les émotions permettent également de guider l'agent. C'est notamment une conséquence indirecte de l'intégration dû au renforcement émotionnel. Ainsi, les règles appliquées lorsque l'agent va réaliser un bon score (comme l'identification rapide de pièce grâce à « *Si la pièce est rouge, c'est une Barre* ») vont être renforcées et appliquées plus souvent.

## 3.9. Adaptabilité à l'environnement

L'objectif de base du système est d'obtenir une certaine adaptabilité de l'agent. C'est ce que permet la combinaison de la méthode de raisonnement et du cycle d'apprentissage proposé.

L'expérimentation dans l'environnement permet à l'agent d'obtenir des informations. A partir de là, l'induction crée de nouveaux concepts et modifie donc sa sémantique. Puis la déduction modifie son comportement en créant les règles et hypothèses correspondantes. L'intégration lui permet de sélectionner et d'utiliser celles qui lui sont utiles. Une fois ce cycle

fait, il peut recommencer : L'agent expérimente à nouveau sur l'environnement, mais cette fois en utilisant ses nouvelles perceptions et sa nouvelle sémantique, il induit à partir de là et il recommence sa sélection. Cela lui permet de réaliser un raisonnement de plus en plus adapté à l'environnement, tout en conservant la possibilité de se réadapter si celui-ci change, car il peut toujours réutiliser des fonctions et concepts précédents pour représenter un concept qui ne correspondrait pas à sa nouvelle sémantique.

**Cadre 3.9 Exemple d'adaptabilité**

Avec sa sémantique et son comportement de départ, l'agent identifie une Pièce, la décrit avec sa forme complémentaire, identifie un mur, le décrit, compare les deux et place la pièce.

Après quelques parties, il crée le concept *Pièce1* (correspondant à une *Barre*) et apprend que s'il perçoit du rouge, c'est qu'il y a une *Pièce1*. Il perçoit donc directement une *Pièce1*, avec toutes ses caractéristiques propres, et non plus une *Pièce*.

Il fait de même avec le mur, et identifie *Mur1*. A partir de cette nouvelle sémantique, il est à même de raisonner sur ces concepts et de créer par exemple la règle « *Si la pièce actuelle est une Piece1 et le mur un Mur1, placer la pièce actuelle en (3,7)* ».

Même une fois qu'il a adapté sa sémantique avec toutes les pièces possibles, il reste capable de se réadapter si une nouvelle huitième pièce apparaît. De même, si une pièce change de couleur, il peut progressivement inhiber le concept utilisé précédemment et en créer un nouveau.

## 3.10. Synthèse

Les différentes étapes peuvent se résumer à l'aide de la figure 3.12.

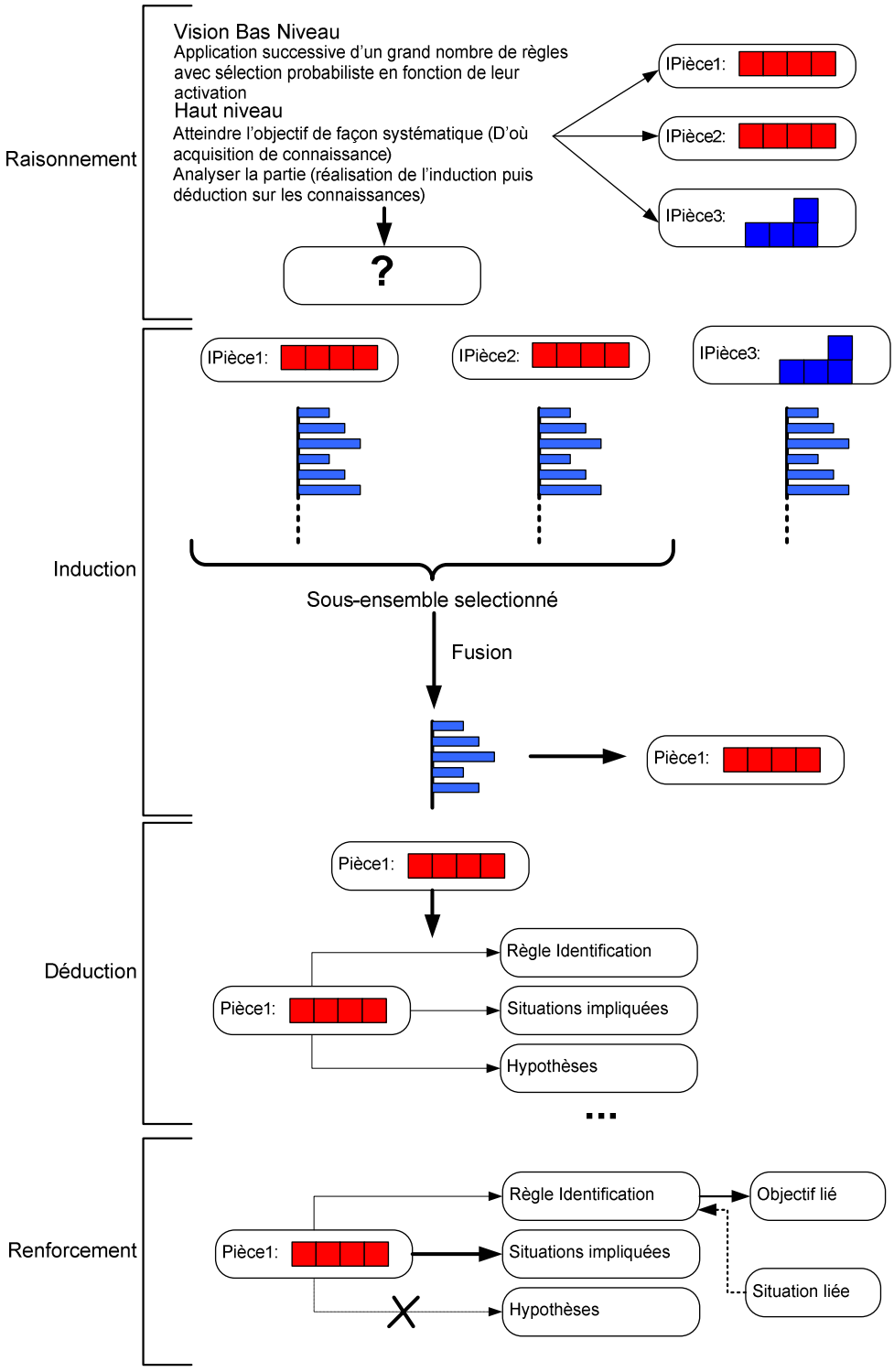



**Figure 3.12 Synthèse du fonctionnement du cycle d'apprentissage**

# Chapitre 4
# Représentation des connaissances

Comment sont représentés les règles et les concepts ? Que désignent les termes de concept, de connaissance et d'activation dans le graphe ?

Le modèle de graphe orienté d'activation permet de répondre au double problème représentationnel (permettre la réflexivité) et fonctionnel (permettre une utilisation efficace aussi bien des anciennes que des nouvelles connaissances). Pour rendre possible un traitement et une modification des règles, celles-ci sont représentées de façon explicite comme plusieurs connaissances liées entre elles (Une connaissance représente un nœud du graphe). Chaque partie ou fonction d'une règle peut ainsi être analysée et modifiée indépendamment ou par rapport aux autres.

Une justification du choix du modèle et une présentation globale sont données en section 4.1. Les définitions des termes utilisés sont données en section 4.2, et les connaissances sont analysées en détail section 4.3. Le rôle des liens et la notion de concept fluide sont discutés en section 4.4. En particulier, la représentation des règles est discutée en section 4.5. L'objectif du modèle est de pouvoir exécuter les règles décrites de façon explicite dans le graphe Leur interprétation dynamique est décrite en section 4.6.

## 4.1. Introduction

### 4.1.1. Choix du modèle de représentation

Le modèle choisi pour la représentation des connaissances doit être adapté aux différents objectifs fixés : il doit permettre une certaine réflexivité des connaissances, c'est-à-dire que l'agent doit pouvoir analyser, modifier et créer ses propres règles et concepts. Les concepts, règles et les métarègles doivent être modélisés de la même façon afin que les règles puissent s'appliquer à elles-mêmes et qu'elles puissent servir de paramètres pour d'autres règles. De plus, le type de représentation choisi doit permettre un contrôle simple et efficace de l'agent.

Une représentation sous forme de prédicats ne permet pas la représentation au même niveau des règles et des variables, et donc leur traitement homogène. Une représentation sous forme de graphe, au contraire, permet cette uniformité de tous les composants.

Une fois choisie la représentation sous forme de graphe, un deuxième problème se pose alors pour la définition du modèle : quel niveau choisir pour le grain du graphe ? C'est-à-dire quel doit être le degré de décomposition des règles ?

- Le niveau le plus haut, la Frame [Minsky 1975], comme dans EURISKO [Lenat 1983a]? Ses problèmes (par rapport à notre objectif) sont la nécessité d'analyser le contenu de fonctions écrites dans le langage de programmation pour pouvoir les modifier et surtout la restriction des types de « slots » des Frames en fonctions de modèles pré-établis et difficilement modifiables.
- Le niveau le plus bas, le neurone ? Bien que donnant de très bons résultats liés à l'apprentissage par renforcement (notamment dans son application au backgammon [Tesauro 2002]), il n'est pas interprétable et des règles complexes ne sont pas ingérables explicitement.

Nous choisissons ici un niveau intermédiaire : des liens non typés (qui n'ont qu'une signification fonctionnelle) et des règles représentées de façon explicite ou implicite avec tous leurs composants séparés dans des connaissances distinctes. L'avantage de conserver un niveau implicite moins interprétable est qu'il permet à l'agent d'apprendre des « habitudes » qui enrichissent progressivement le niveau explicite. Le niveau explicite permet lui de conserver une adaptabilité et une logique pour le raisonnement. La représentation des connaissances choisie est très proche des graphes de concepts fluides (slipnets) développé par D. Hofstadter ([Hofstadter 1995]) dans CopyCat ([Mitchell 1993]) pour simuler le raisonnement analogique. Une orientation des liens ainsi que des facteurs de stabilité sur les liens (en plus des valeurs d'intensité) ont étés ajoutés pour permettre l'apprentissage. De plus, les liens du slipnet peuvent être typés par d'autres concepts. La volonté de traiter tous les liens de manière identique nous a conduit à ne pas le faire. L'information sur le type de relation est conservée en créant une connaissance intermédiaire liée à la fois à la connaissance du type et au paramètre-cible.

Une formalisation logique des concepts fluides a été réalisée par P. Wang à travers le système NASR ([Wang 1995]). Le lien d'héritage de type unique avec deux facteurs d'intensité et de confiance est identique dans notre système. La différence fondamentale est

une conséquence de l'objectif de chaque représentation. Alors que NASR a un objectif d'inférence logique, ce qui le conduit à utiliser une fréquence comprise entre 0 et 1 pour l'intensité d'un lien, notre représentation a un objectif fonctionnel de transmission d'activation, ce qui nous conduit à utiliser une intensité comprise entre -1 et 1, permettant ainsi l'inhibition. Notre raisonnement ne se fait pas par inférence logique comme dans NASR, mais de façon implicite par transmission d'activation, et de manière explicite par application de règles (interprétées) sur le graphe.

Par rapport aux graphes conceptuels ([Sowa 2000]) ou aux logiques de description ([Napoli 1997]), l'objectif est différent. Les liens entre les nœuds d'un graphe conceptuel ont une signification uniquement sémantique, d'où l'intérêt de les typer. Le raisonnement au niveau des types et au niveau des concepts est séparé. Dans notre modèle, l'interprétation des liens est avant tout fonctionnelle (transmission de l'activation). De plus, les liens ne sont pas typés. Il est nécessaire de considérer les types de liens comme des connaissances standards afin de pouvoir raisonner dessus de la même façon que sur les autres concepts.

### 4.1.2. Présentation

D'un point de vue fonctionnel, le choix des connaissances utilisées aussi bien pour l'application de règles que pour le choix d'un paramètre se fait par transmission d'un certain degré d'activation entre les connaissances en fonction des liens orientés qu'elles ont entre elles. Une connaissance aura d'autant plus de chance d'être activée ou choisie que son degré d'activation est important. Une connaissance fortement activée va ainsi transmettre son activation aux connaissances avec lesquelles elle est liée, en fonction de l'intensité du lien. En particulier, un concept sous-type d'un autre concept va activer fortement son concept-père, ainsi que tous les éléments de définitions ou les propriétés qui lui sont liés en propre.

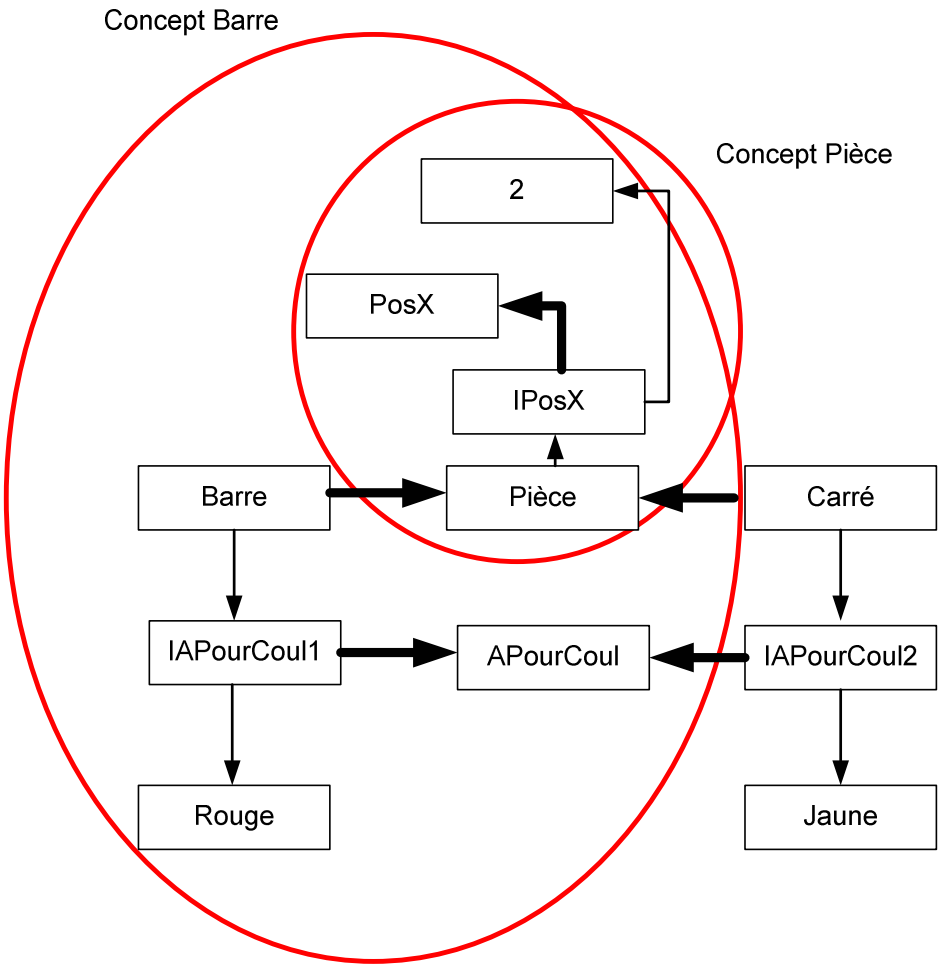



**Figure 4.1 Extrait du Graphe autour des concepts Barre et Pièce**

Ainsi, l'activation de la connaissance *Barre* va activer tout ce qui est lié au concept de barre : *Piece* (une barre est une pièce), *Rouge* (une barre est rouge) et *2* (une barre est une pièce, donc sa position est PosX=2), mais pas *Carré* ou *Jaune* (une barre n'est ni un carré ni jaune). L'activation de *Pièce* seule, c'est-à-dire du concept général de pièce, va activer uniquement *2*.

Les liens entre les connaissances sont ainsi fonctionnels, contrairement par exemple aux liens des graphes conceptuels qui ont un but sémantique. L'objectif ici est d'activer la bonne connaissance au bon moment. Le contenu sémantique est, lui, contenu dans les connaissances d'un même concept. Par exemple, le fait que *2* décrive la position de la pièce est contenu dans la connaissance intermédiaire *IPosX* (pour Instance de *PosX*) qui est de type *PosX*, indiquant ainsi la signification de la relation. La représentation explicite d'une règle ou d'un concept se fait ainsi par des connaissances intermédiaires qui sont toutes activées lors de l'activation du concept auquel elles appartiennent. Les connaissances implicites et l'identité sont elles modélisées par les liaisons fonctionnelles entre les connaissances.

## 4.2. Définitions

L'utilisation de termes largement employés pour désigner des concepts parfois très différents nécessite de préciser quelques définitions sur le sens qui sera appliqué dans notre modèle :

**Connaissance** : Un nœud du graphe. Une connaissance est liée à d'autres connaissances à l'aide de liens. A chaque instant elle est caractérisée par un degré d'activation. Elle peut contenir une fonction intégrée. La description complète d'une connaissance est donnée en section 4.3. Une **instance** d'une connaissance est une connaissance qui a un lien (ou une suite de liens consécutifs) d'intensité maximum pointant vers la connaissance (par exemple dans la figure 4.1, Pièce est une instance de Barre). Une connaissance est représentée dans un rectangle à bords droits.

**Lien** : Un lien orienté entre deux nœuds du graphe. Il est caractérisé par deux valeurs : une valeur d'intensité décrivant sa valeur de vérité et une valeur de stabilité décrivant la confiance accordée à l'intensité actuelle.

**Concept** : Un concept est représenté par un sous-graphe centré sur une connaissance. Le nom du concept est identique au nom de la connaissance sur laquelle il est centré (par exemple le concept de Barre centré sur la connaissance Barre). En plus de cette connaissance, il contient l'ensemble des connaissances qui lui sont liées, directement ou indirectement, dans la limite d'une certaine valeur d'activation des liens intermédiaires (voir section 4.4 pour plus de détails). Un concept est représenté dans un rectangle à bords arrondis.

**Degré d'activation** d'une connaissance ou du concept centré sur cette connaissance : Valeur liée à la connaissance qui est proportionnelle à sa probabilité d'être activée. Elle peut être interprétée comme le degré d'attention porté par l'agent au concept considéré.

**Fonction intégrée** d'une connaissance: Lors de chaque période, un certain nombre de concepts décrivant des règles explicites sont sélectionnés avec une probabilité proportionnelle à leur degré d'activation. Celles-ci sont interprétées dynamiquement en appelant les fonctions intégrées des connaissances qui les composent.

**Règle** : type de concept qui peut être représentée de façon explicite, implicite ou intégrée.

## 4.3. Description d'une connaissance

Une connaissance est un élément de savoir qui contient plusieurs informations :

### 4.3.1. Fonction intégrée

Chaque connaissance C peut contenir une fonction intégrée qui est exécutée lorsqu'une règle explicite est interprétée et qu'une sous-partie de la règle est une instance de C. Les fonctions intégrées des connaissances correspondent aux fonctions élémentaires de l'agent, au niveau du traitement de l'information du graphe (réalisation de test, modification de lien, lecture de paramètre) ou de tout autre domaine pour lequel on a préféré utiliser une fonction intégrée à une connaissance plutôt qu'une règle implicite ou explicite. L'inconvénient de ce choix est que la fonction ne pourra pas être analysée ou décomposée. L'avantage est qu'elle est exécutée en une fois, elle peut être aussi complexe qu'on le souhaite et elle peut utiliser le graphe de connaissances aussi bien comme donnée que pour le modifier. On peut par exemple introduire des règles intégrées permettant des calculs élémentaires (addition, division), des fonctions de parcours de graphes pour les jeux, des fonctions de traitement des règles pour le raisonnement explicite.

### 4.3.2. Degré d'activation

Une connaissance i possède à tout instant t un degré d'activation $a_i^t$ compris entre 0 et 1. Plus ce niveau sera élevé, plus la connaissance aura de chance d'être sélectionnée pour une utilisation ou une analyse éventuelle. Cette valeur est modifiée de quatre façons :

- par la transmission d'activation au travers des liens pointant vers la connaissance
- par amplification de l'activation lors de l'interprétation d'une règle explicite (si une règle a pour conséquence de favoriser ou d'inhiber une connaissance)
- directement par les émotions
- par l'actualisation due au temps

### 4.3.3. Liens

Une connaissance est liée à d'autres connaissances à l'aide de liens orientés qui ont une utilité fonctionnelle (ils servent à transmettre une activation). Chaque lien lie une

connaissance-source a à une connaissance-cible b et possède deux valeurs mesurant l'intensité et la stabilité de la liaison entre deux connaissances.

L'**intensité** ($v_{a,b}$) indique la force du lien entre a et b et est normalisée entre -1 et 1. Cette valeur servira à déterminer la part d'activation transmise à la connaissance-cible.

Une valeur de 1 signifie que la totalité de l'activation sera transmise à la cible. Cela peut être interprété comme une « identité orientée » : toutes les informations et fonctions liées à la cible sont comprises dans le concept de la source (car elles seront activées de la même façon), mais pas obligatoirement l'inverse. Par exemple, une Barre est une Pièce mais une pièce n'est pas toujours une Barre.

Une valeur négative signifie un effet inhibiteur sur l'activation de la connaissance-cible par opposition à l'effet positif précédent.

Une valeur de 0 signifie une absence de lien entre la source et la cible. La différence entre une intensité 0 et une absence réelle de lien réside dans la modification future de l'intensité due à la stabilité actuelle du lien.

Il existe un effet mémoire. Plus le lien est renforcé, moins il va changer. D'où l'utilisation d'une deuxième valeur qui correspond à la vitesse de modification du lien.

La **stabilité** du lien ($p_{a,b}$) entre a et b est normalisée entre 0 et 1. Elle augmente avec les activations et les modifications du lien. Lorsqu'elle est égale à 1, il n'y a plus de modification de l'intensité du lien, sa valeur est définitive. Si elle est égale à 0, il y a oubli immédiat et destruction du lien qui devient inutile.

### 4.3.4. Représentation d'une connaissance

Une connaissance sera représentée comme un rectangle au bord plus ou moins épais en fonction de son degré d'activation actuel.

Les liens entre connaissances seront représentés par des flèches plus ou moins épaisses en fonction de leur intensité (en pointillés dans le cas d'intensité négative). La stabilité du lien sera éventuellement représentée par le gris plus ou moins foncé de la flèche.

Par exemple :

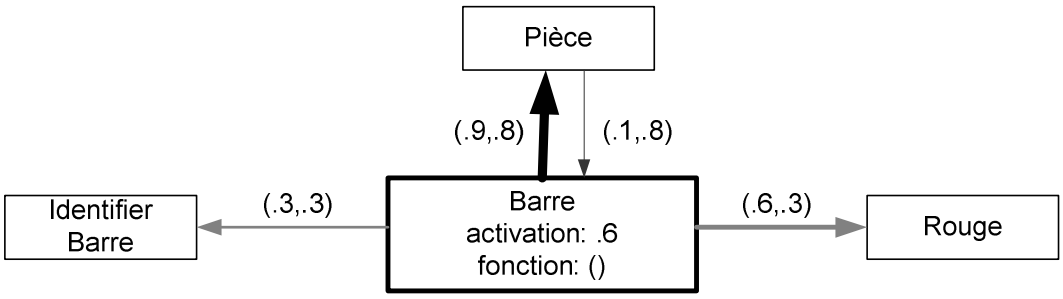



**Figure 4.2 Représentation complète de la connaissance Barre**

La connaissance *Barre* a un degré d'activation de 0,6, d'où un bord épais, les trois autres connaissances ayant une activation moindre.

Le lien de *Barre* vers *Pièce* a une intensité de 0,9 (trais épais) et une stabilité de 0,8 (trais noir). L'intensité indique que *Barre* est « très liée » à *Pièce*, ce qui a pour conséquence que tout ce qui est lié à *Pièce* sera activé aussi par *Barre*. Cela peut être interprété comme le fait qu'une *Barre* est une *Pièce*. La stabilité signifie que l'intensité du lien ne va plus beaucoup changer, la valeur de la liaison est fixée de façon quasi définitive.

Le lien de *Pièce* vers *Barre* a une intensité de 0,1. Cela signifie que le lien existe, mais est faible. Une *Pièce* peut effectivement être une *Barre*, mais cela peut également être autre chose. Tout ce qui est lié à *Barre* ne sera pas activé si *Pièce* est activée. La stabilité élevée (0,8) signifie que cette faible liaison est stable, donc certaine.

Le lien entre *Barre* et *Identifier Barre* signifie que la connaissance *Barre* est liée à la fonction de *Identifier Barre*. Cette liaison est positive (0,3), même si faible, la fonction aura donc quelques chances d'être sélectionnée si *Barre* est activée. La liaison est par ailleurs incertaine (stabilité de 0,3), donc susceptible d'être modifiée facilement par la suite, par exemple à la suite d'une utilisation réussie et utile de la fonction *Identifier Barre* après l'activation de *Barre*.

Le lien entre *Barre* et *Rouge* signifie que la notion de *Barre* est liée de façon assez importante à *Rouge* (0,6), cette liaison étant susceptible d'être modifiée facilement (stabilité de 0,3)

## 4.4. Rôle des liens d'activation et concepts

L'intensité des liens a une double interprétation.

- Fonctionnelle : elle détermine l'activation transmise entre deux connaissances.
- Sémantique : les connaissances fortement liées sont activées simultanément, et peuvent donc être assimilées à un même concept. On peut alors parler de concept fluide dans la mesure où sa description, ses fonctions et définitions peuvent, d'une part, varier dans le temps en fonction des liens, et, d'autre part, sont plus ou moins étendues à un instant donné en fonction du seuil qu'on se pose.

L'interprétation de l'intensité et le sens des liens entre les connaissances nous permettent de distinguer certains cas-types :

### 4.4.1. Liens et Identité

Deux connaissances liées de façon maximale et bilatérale transmettront la totalité de leur activation à l'autre. Elles seront donc en permanence (à une légère différence temporelle près) activées de la même façon, et les connaissances qui leur sont liées seront activées indifféremment par l'une ou par l'autre. Du point de vue de l'activation ou de leurs liens avec l'extérieur, elles sont identiques. On considère en fait qu'elles forment un même concept.

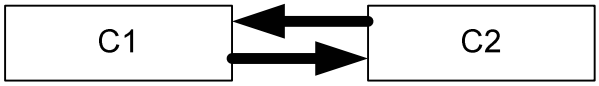



**Figure 4.3 Exemple d'identité de concepts entre C1 et C2**

### 4.4.2. Concepts flous et fluides

Un concept C1 sous-type ou instance d'un autre concept C2 (par exemple « barre » est un sous-type de « pièce ») sera lié fortement au père mais pas inversement. Les propriétés du père appartiendront alors au concept fils, alors que les propriétés du fils n'appartiendront pas au concept père.

Un concept centré sur une connaissance peut alors être vu comme l'ensemble des connaissances qui ont une liaison avec la connaissance centrale avec un certain seuil d'intensité. Ainsi, figure 4.1, le concept de *Barre* contient toutes les connaissances qui le caractérisent plus celles du concept de *Pièce*.

On peut parler de concepts flous car les limites du concept dépendent de la limite qu'on se fixe au niveau de l'intensité des liens. Par exemple, figure 4.4, si on considère une limite forte, le concept de *Barre* ne se compose que des connaissances *Barre* et *Piece*. Au contraire,

si on accepte des intensités plus faibles, le concept englobe également les attributs liés aux deux connaissances : la position et la couleur.

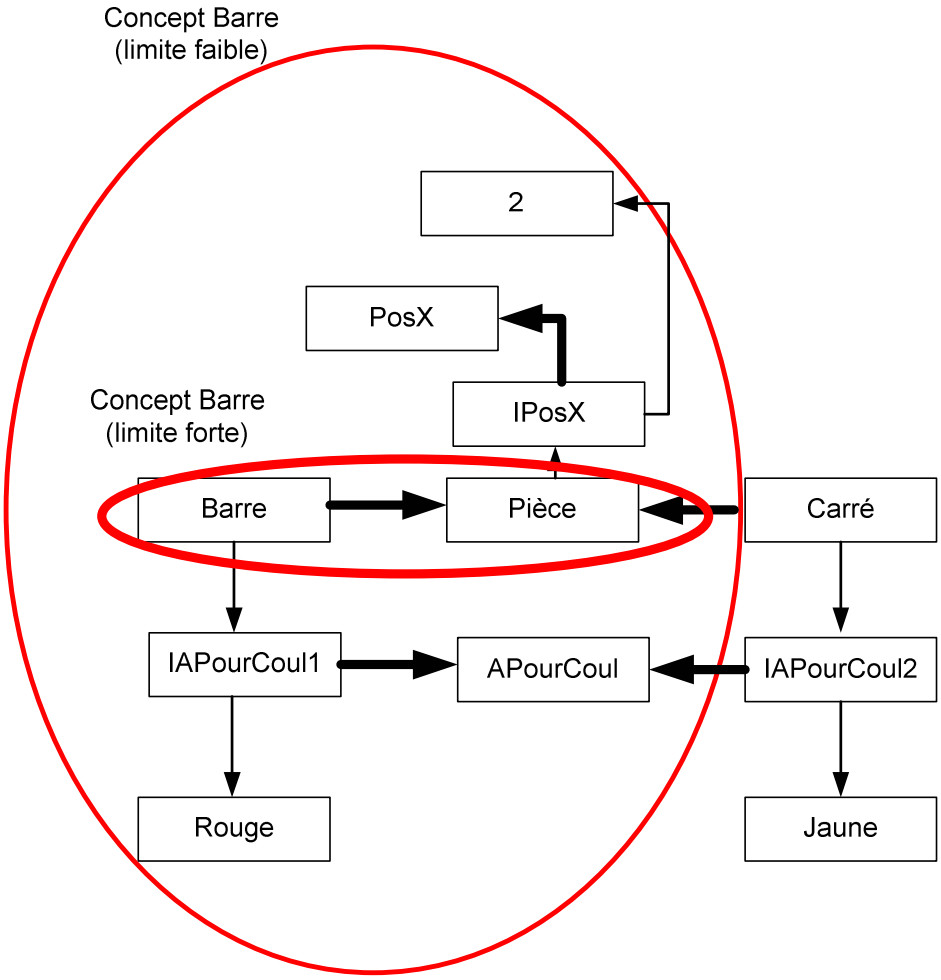



**Figure 4.4 Composants du concept de *Barre* selon la limite d'intensité choisie**

La fluidité des concepts (tirés de [Mitchell 1993]) provient des variations des liens entre les connaissances. Entre deux périodes, le renforcement d'un lien peut intégrer un grand nombre de connaissances dans un concept.

### 4.4.3. Attributs

Les liens d'un autre type que l'identité, tels que la couleur rouge d'une barre, sont représentés grâce à des connaissances intermédiaires liées à la fois à la connaissance correspondant au type de lien (*APourCoul*) et à la connaissance cible de l'attribut (*Rouge*). Cette connaissance, désignée par *IAPourCoul1* (pour Instance de A Pour Couleur numéro 1), est comme toute autre connaissance le centre d'un concept. Celui-ci peut être vu comme le

concept correspondant à une liaison de type correspondant vers la cible désignée, ici une liaison de type *APourCoul* ayant pour cible *Rouge*.

Cette modélisation permet d'activer uniquement les connaissances liées au concept activé. Par exemple si une Barre est une Pièce qui a pour couleur Rouge et qu'un Carré est une Pièce qui a pour couleur Jaune, la modélisation sera la suivante :

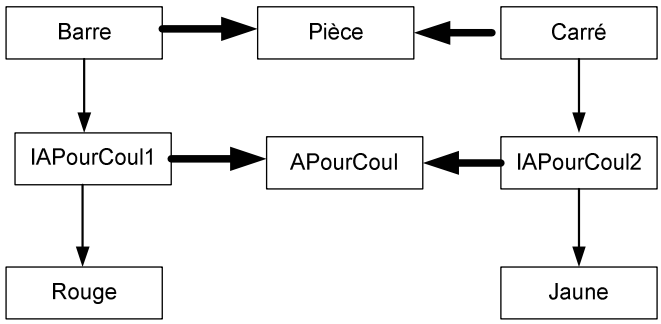



**Figure 4.5 Représentation des attributs couleur de Barre et de Carré**

Lorsque le concept de *Barre* est activé (en T=1), seules les connaissances liées à la Barre sont activées : d'abord *Pièce* et *IAPourCoul1*, (en T=2), puis *APourCoul* et *Rouge* (en T=3), mais jamais le *Jaune* ou le *Carré*, qui, bien qu'étant une *Pièce*, n'est pas lié à *Barre* :

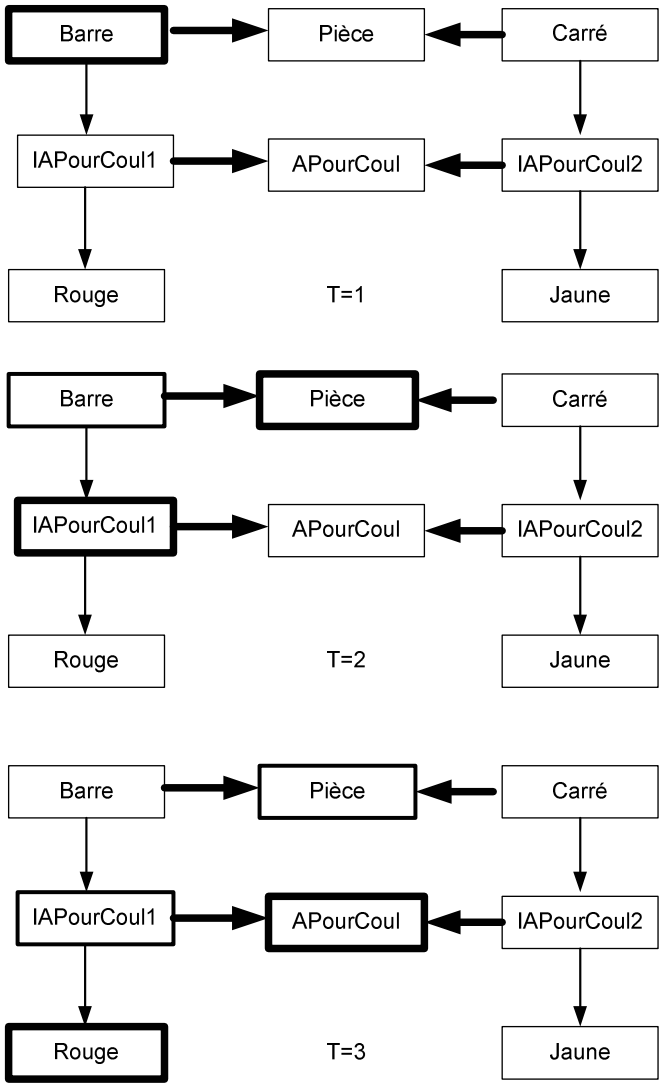



**Figure 4.6 Transmission de l'activation vers les attributs de Barre**

Pour simplifier la lecture, les liens correspondant à un attribut (donc avec connaissance intermédiaire d'un certain type) pourront être représentés avec le nom du type de relation sur le lien à la place de la connaissance intermédiaire. Par exemple la représentation précédente devient :

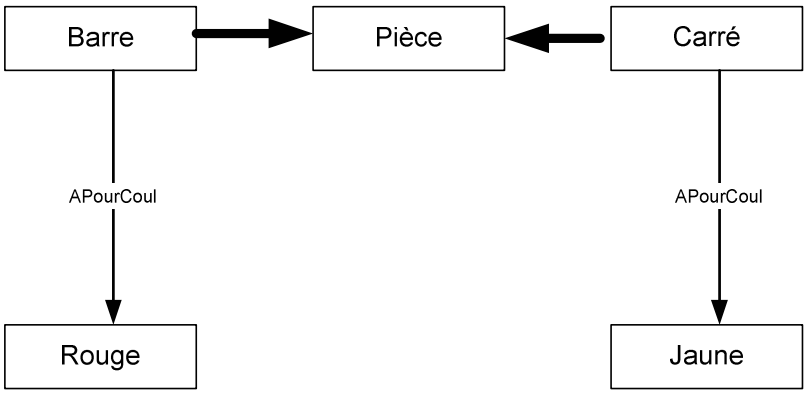



**Figure 4.7 Représentation simplifiée des types des attributs**

### 4.4.4. Représentation d'un concept

Ces interprétations en terme d'attributs permettent de représenter les concepts de façon abstraite sous forme de Frame. Cette représentation, plus lisible que celle du graphe donne une vision centrée sur une connaissance (la connaissance correspondant au concept) avec ses attributs liés.

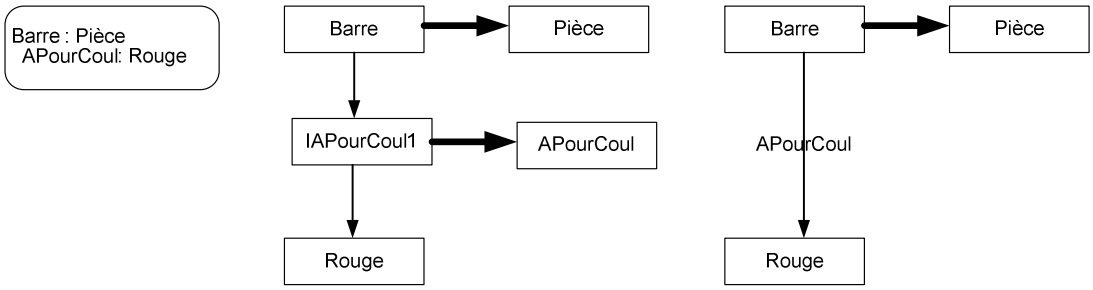



**Figure 4.8 Représentation du concept de Barre sous la forme de Frame, Graphe et Graphe simplifié**

Pour distinguer les connaissances et les concepts, on représente un concept sous la forme de Frame avec un bord arrondi. Le « : » (Barre : Pièce) sous-entend un lien de type est-un implicite (Une Barre est une Pièce). Les autres attributs, décalés vers la droite, sont décrits en fonction de leur type, donc du type de la connaissance intermédiaire dans la relation (*IAPourCoul1*).

Si on veut représenter des attributs de l'attribut, on décale les sous-attributs vers la droite. Par exemple la représentation explicite de la règle en section 4.5.3.

## 4.5. Représentation d'une règle

Il peut exister plusieurs types de règles dans le graphe : des règles intégrées aux connaissances, des règles implicites et des règles explicites.

### 4.5.1. Représentation intégrée

Un concept, règle ou fait, peut être inclus dans la fonction intégrée ou dans les paramètres d'une connaissance. Le but est une grande rapidité d'exécution et une totale liberté de conception. L'inconvénient est que le concept ne peut être analysé, décomposé ou transformé par les autres connaissances. L'agent ne peut faire de rapport sur son contenu, et ne sera ainsi jamais conscient de la façon dont il obtient le résultat produit par cette connaissance.

Les règles utilisant la représentation intégrée sont les règles de base, qui n'ont pas intérêt à être analysées ou décomposées, telles les fonctions arithmétiques ou certaines fonctions de gestion des règles explicites. Cette représentation est aussi utilisée pour des règles complexes comme l'analyse inductive automatique utilisant l'analyse de données symboliques présentée au chapitre 0.

### 4.5.2. Représentation implicite

Le but d'une représentation implicite est, grâce aux liens, d'activer les connaissances les plus adaptées à la situation. Une représentation implicite de la règle « une barre est rouge » est ainsi une simple liaison entre la connaissance *Barre* et la connaissance *Rouge*. Cette règle n'est pas explicite en ce sens qu'une analyse extérieure ne peut pas déterminer la nature du lien entre les connaissances et aucune interprétation n'est nécessaire pour appliquer la règle. Elle correspond bien à l'objectif de règle implicite car lorsque la connaissance *Barre* est activée, cela active automatiquement la connaissance *Rouge*, ce qui implique que si l'agent a besoin de choisir une couleur, par exemple pour tester une règle explicite qui associerait une couleur à ce type de pièce, le rouge sera favorisé et sera choisi en priorité.

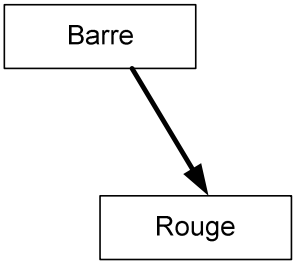


**Figure 4.9 Représentation implicite de la règle « Une Barre est Rouge »**

### 4.5.3. Représentation explicite

Les règles doivent pouvoir être utilisées, analysées et modifiées, c'est le cœur même du modèle. Pour cela, les différents composants d'un concept doivent être décrits de façon explicite. Les liens n'ayant pas de contenu sémantique, la description des relations entre les composants d'un concept se fait à l'aide d'attributs.

La sémantique des règles explicites, c'est-à-dire les différents types de règles, de tests, d'actions et de variables n'est pas fixe. L'agent peut en ajouter de nouveaux. Des types de bases sont fournis à l'agent pour qu'il puisse par la suite les compléter en les combinant.

Par exemple, les règles doivent pouvoir utiliser des variables. Plusieurs types de variables simples existent au départ, tels que *VarFixe* (qui désigne une connaissance particulière de façon constante), *VarUn* (qui, chaque fois qu'elle est évaluée, renvoie une connaissance du type désigné par son sujet, ainsi « Une Pièce »), *VarLitParam* (qui renvoie la valeur de l'attribut de type *Param* de son Sujet, par exemple « La couleur de la pièce »). La règle peut utiliser des paramètres variables, tels que « un concept ». Pour pouvoir différencier les variables d'une même règle, il n'existe qu'une seule connaissance par variable dans la représentation sous forme de graphe, même si elle est utilisée plusieurs fois dans la règle.

La règle « Si la couleur d'une pièce est rouge, alors c'est une barre » est ainsi décrite par le concept suivant :

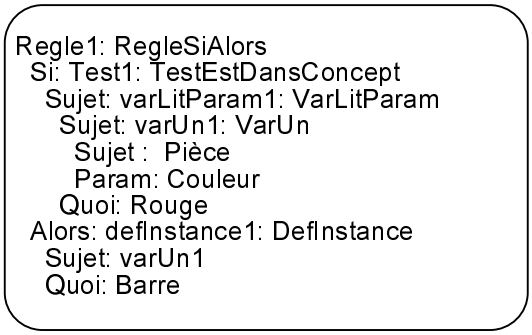



**Figure 4.10 Représentation explicite de la règle « Si une Pièce est Rouge, alors c'est une Barre »**

*varUn1* désigne « une pièce », *varLitParam1* « la couleur d'une pièce », *Test1* le test « Si la couleur d'une pièce est Rouge », *defInstance1* l'action « la pièce varUn1 est une

Barre ». De façon globale, *Règle1* correspond donc bien à « Si la couleur d'une pièce est rouge, c'est une barre ».

Les règles ainsi modélisées sont donc interprétables, mais surtout facilement utilisables comme données et donc modifiables par d'autres règles.

Le fonctionnement concret et l'utilisation des règles explicites sont décrits dans la section 5.5.

Sauf mention contraire, la notion de règle se rapportera par la suite à cette représentation sous la forme explicite des règles.

## 4.6. Interprétation dynamique des règles explicites

Les règles doivent pouvoir être utilisées, analysées et modifiées. Pour cela, les différents composants d'un concept doivent être décrits de façon explicite.

Comme une règle peut être modifiée constamment par d'autres règles, une compilation à la création de la règle est impossible, il faudrait réaliser une reprogrammation dynamique à chacun de ses changements. Pour éviter cette tâche complexe et lente, l'exécution de la règle se fait en appelant de façon dynamique chacun de ses sous-composants, chaque sous-composant se chargeant d'appeler ses sous-composants et ainsi de suite.

Par exemple, pour la règle: « Si l'activation de *CoLumP-all* est supérieure à l'activation de *CoLumP-et*, alors la situation de la *LumP* est *Allumé* » :

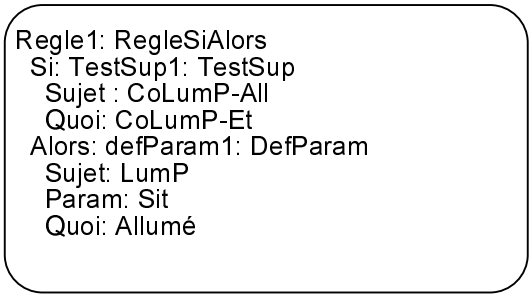



**Figure 4.11 Représentation explicite de la règle « Si l'activation de *CoLumP-all* est supérieure à l'activation de *CoLumP-et*, alors la situation de la *LumP* est *Allumé* »**

Lorsque la règle est exécutée, la fonction d'interprétation de règle conditionnelle (*RegleSi.interprete()*) commence par identifier la condition (*ITestSup1* : une instance de *TestSup*, la connaissance correspondant à un test de supériorité), trouve la fonction d'interprétation correspondante (qui correspond à la fonction de la connaissance interprétable

la plus proche de la condition, ici *TestSup*) et demande le résultat de cette fonction en lui passant en paramètre la connaissance correspondant à la condition (exécution de *TestSup.interprete(ITestSup1)* ). La fonction *interprete()* de la condition va elle-même demander les résultats de ses composants (ici deux instances de *LitActivation*, connaissance qui lit le degré d'activation d'une autre connaissance), réaliser le test et retourner le résultat à la fonction d'interprétation de règle (*RegleSi.interprete()*).

Une description plus détaillée du fonctionnement de l'interprétation est donnée en section 10.1.1.2.

Les connaissances directement exécutables constituent les briques de bases des règles explicites. Ce sont les fonctions de base intégrées à l'agent qui effectuent les opérations élémentaires telles que lire un attribut, créer une connaissance, choisir une variable d'un certain type, …

Ces fonctions intégrées à l'agent peuvent également correspondre à des fonctions implémentées directement en java pour être plus efficaces. Ce type de fonction a l'avantage d'être plus rapide et plus libre qu'une fonction décrite de façon explicite. L'agent ne pourra bien sûr ni l'analyser, ni la modifier, mais il pourra l'utiliser et analyser ses résultats. La réflexivité complète n'étant pas l'objectif du système (qui est de trouver et d'utiliser efficacement des concepts et heuristiques adaptés au domaine), l'utilisation de telles fonctions permet de doter l'agent de fonctions efficaces dans des domaines que l'utilisateur ne souhaite pas obligatoirement le voir explorer en profondeur. Par exemple, les fonctions d'addition, multiplication ou comparaison d'entiers sont théoriquement découvrables et exprimables dans ce type de système puisqu'AM les a découvertes. Mais comme il ne s'agit pas du tout du domaine de travail de l'agent et que ces fonctions sont très utiles pour traiter les ensembles et les données numériques comme les scores, elles ont été directement intégrées à l'agent. Cela ne l'empêche pas de les redécouvrir s'il en a besoin, cela lui permet juste de réaliser ces opérations d'instinct (sans qu'il sache comment il y est parvenu).

L'agent peut bien sûr créer ses propres méthodes explicites (test, action, …). L'interprétation d'une règle explicite qui n'est pas composée de fonctions intégrées à l'agent fonctionne alors de façon similaire au cas précédent. L'exécution de la méthode explicite est réalisée en exécutant les composants de sa définition.

Ce traitement peut sembler lourd mais il a de nombreux avantages :

- Il permet de traiter des concepts représentées sous forme explicite (c'est le but)

- La sémantique de représentation n'est pas fermée. L'agent peut très bien créer de nouveaux types de règles, de tests ou de variables. Il lui suffit de décrire la définition correspondante et il peut instancier sa nouvelle méthode explicite exactement comme une méthode de base.
- Il est entièrement décentralisé. L'activation des règles d'interprétation se fait à partir de la règle traitée elle-même. Par exemple, si la règle utilise une sémantique complètement différente de la sémantique de base, seules les règles de traitements correspondantes seront activées.
- Il permet une grande modularité des représentations. Les branches peuvent ainsi être analysées indépendamment mais aussi modifiées sans que le reste ne soit affecté. Modifier le sujet de la règle précédente (par exemple pour essayer une règle plus générale) se fait simplement en remplaçant ou modifiant la branche *ISujet*, le reste peut rester identique.

## 4.7. Conclusion

Dans ce chapitre, nous avons présenté une méthode de représentation homogène des connaissances. Les règles comme les objets observés ou les concepts appris sont représentés de façon uniforme dans un graphe orienté non typé. Les liens ont une signification purement fonctionnelle. Les deux valeurs d'intensité et de stabilité qui caractérisent les liens modélisent les informations concernant la force (actuelle) du lien et son évolution possible (donc la certitude de l'information actuelle). Les structures peuvent être interprétées pour modéliser des attributs typés ou des liens d'instanciation. Cette interprétation permet à la fois un raisonnement explicite sur le graphe et une représentation simple des concepts sous forme de Frames.

Chaque connaissance (nœud du graphe) est au centre d'un concept fluide constitué de toutes les autres connaissances potentiellement activées par cette connaissance-racine. Ce concept fluide va évoluer avec les changements du graphe, que ceux-ci soient réalisés explicitement par l'agent (par application de règles), ou qu'ils soient réalisés implicitement (par renforcement ou par oubli). La représentation homogène des connaissances permet en particulier de représenter les règles utilisées par l'agent de façon explicite. Elles peuvent ainsi être aussi bien créées que modifiées, exécutées ou analysées par l'agent.

# Chapitre 5
# Fonctionnement global de l'agent

## 5.1. Présentation

Comment l'agent fonctionne-t-il ? Comment utilise-t-il ses concepts et ses règles ? Le principe de base est l'exécution successive d'un grand nombre de règles heuristiques. L'avantage de cette exécution massive est que l'impact d'un échec possible de plusieurs heuristiques est moins grand. Le choix des règles appliquées et des concepts analysés demeure toutefois fondamental, car le nombre de combinaisons rend impossible une application systématique de toutes les règles, surtout dans le cadre d'un agent temps réel. La liaison des concepts par des liens fonctionnels nous permet de modéliser l'attention de l'agent au travers d'un degré d'activation du concept. Chaque concept transmet son activation à ses voisins et permet ainsi aux règles et aux objets les plus adaptés à la situation actuelle d'être utilisés et analysés. La transmission d'activation entre les connaissances du graphe est décrite en section 5.2 et le choix des règles et des concepts qui en résultent en section 5.3. L'objectif de ce modèle étant d'intégrer un système de raisonnement heuristique réflexif dans un agent, les perceptions et les actions nous permettent respectivement d'introduire et d'utiliser l'activation dans le système (voir section 5.4). La nécessaire autonomie de l'agent conduit à ajouter un contrôle émotionnel qui va également introduire de l'activation dans le graphe et modifier sa structure (voir section 5.5). Enfin, les multiples créations de connaissances rendent nécessaire l'intégration d'une fonction d'oubli, afin que la taille du graphe n'explose pas (voir section 5.6).

Ces différentes sections correspondent également aux différentes étapes de fonctionnement de l'agent (présentées également dans [Caillou 2003a, b]). Celui-ci fonctionne sur le principe d'un faux parallélisme : les différentes étapes s'enchaînent rapidement afin de simuler une action parallèle. L'ordre de réalisation a donc peu d'importance sur le résultat final.

A chaque période, l'agent effectue donc successivement les étapes suivantes :

- Transmission de l'activation
- Exécution des fonctions intégrées
- Perception et actions
- Régulation par les émotions
- Oubli

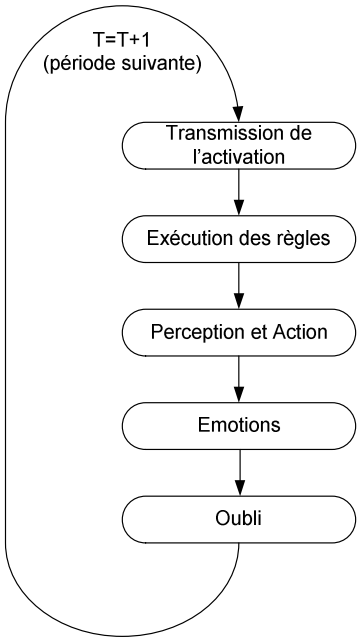



**Figure 5.1 Cycle de fonctionnement de l'agent**

## 5.2. Transmission de l'activation

Le principe de la transmission de l'activation est de faire en sorte que les connaissances très liées soient activées simultanément. Ainsi plus une connaissance est activée, plus les connaissances qui lui sont liées, donc d'abord ses propriétés et ses composants, sont activées à leur tour.

Lorsqu'une connaissance $A$ en active une autre $B$, le degré d'activation de $B$ est augmenté en fonction du degré d'activation de $A$ ($a^t_a$), de l'intensité du lien de $A$ à $B$ ($v_{a,b}$ ) et du degré d'activation actuel de $B$ ($a^t_b$). Par ailleurs, à chaque période, l'activation d'une connaissance est actualisée selon un taux β.

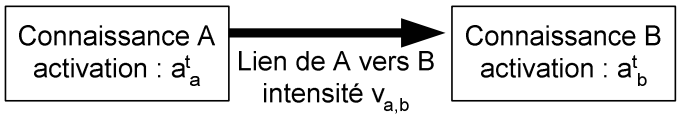



**Figure 5.2 Présentation des notations d'activation et d'intensité pour deux connaissances A et B**

La fonction de calcul de l'activation d'une connaissance i en t+1 est :

$$a_i^{t+1} = 1 - \left(1 - a_i^t\right)^{1-\beta} \prod_j \left(1 - a_j^t\right)^{\alpha \frac{v_{ji}^t}{\sum_k v_{jk}^t}}$$

Une illustration chiffrée de cette fonction est donnée plus bas (en figure 5.5)

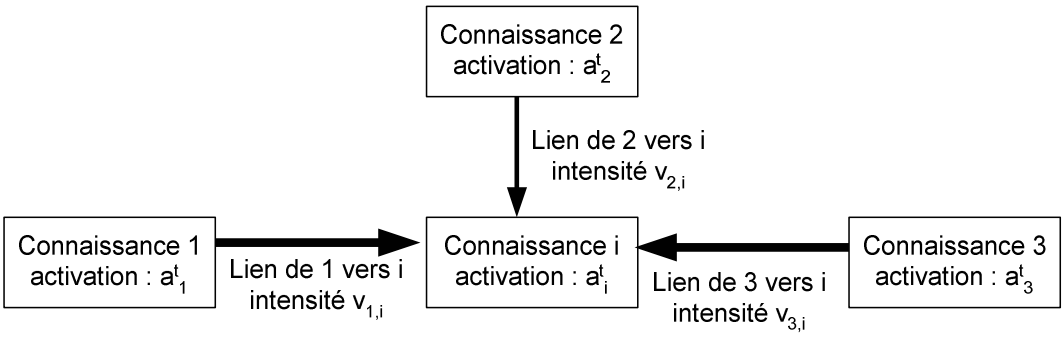



**Figure 5.3 Présentation des notations d'activation et d'intensité pour une connaissance i recevant l'activation de trois connaissances 1, 2 et 3**

Le fond de cette équation peut s'interpréter de la façon suivante : pour avoir l'activation $a^{t+1}{}_i$ d'une connaissance i en t+1,

- Actualiser la valeur de $a^t_i$ par un facteur β pour prendre en compte le temps : $\left(1 - a_i^t\right)^{1-\beta}$
- Pour chaque connaissance j qui a un lien allant de j vers i avec une intensité $v_{i,j}$, diviser cette intensité par le total des intensité des liens partant de j ($\sum_k v_{jk}^t$) (pour pouvoir contrôler l'activation totale du graphe). Transmettre une part de l'activation de j $a^t_j$ pondéré par cette intensité et par un facteur α : $\left(1 - a_j^t\right)^{\alpha \frac{v_{ji}^t}{\sum_k v_{jk}^t}}$

La forme de cette équation s'explique simplement par les avantages qu'elle procure.

En effet, l'activation doit être conservée entre 0 et 1, mais le nombre de liens pointant vers la connaissance, et donc le nombre d'augmentation qu'elle peut subir à chaque période, doit pouvoir être libre. Une façon simple de modéliser la transmission mathématiquement serait une simple somme pondérée d'une activation comprise entre 0 et +∞. Pour pouvoir effectuer cette somme, on effectue un changement de variable permettant d'obtenir une valeur $l^t_i$ variant toujours dans le même sens que $a^t_i$ mais entre 0 et +∞.

$$l_i^t = -\ln\left(1 - a_i^t\right)$$

Cette transformation, et sa transformation inverse permettant de retrouver $a^t_i$ nous permettent de passer d'un domaine [0 ;1[ à un domaine [0 ;+ ∞ [

$$a_i^t = 1 - e^{-l_i^t}$$

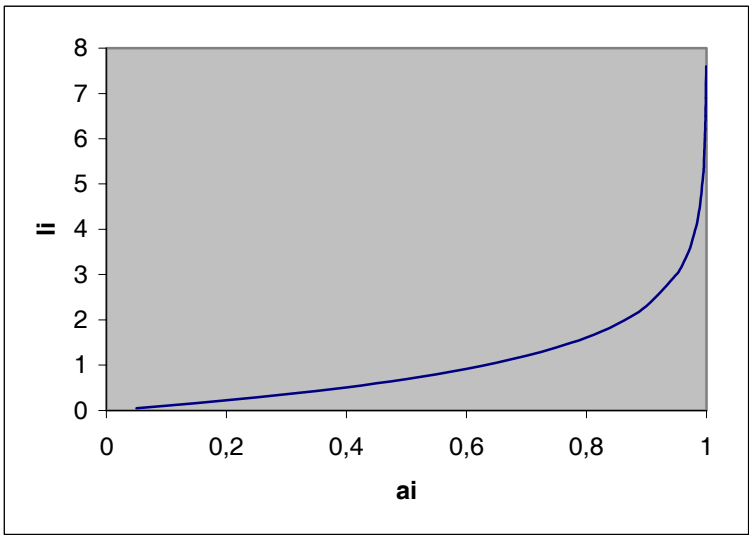



**Figure 5.4 Représentation de la fonction li permettant de changer le domaine de définition de l'activation**

Une fois passé sur le domaine [0 ;+ ∞ [, il est possible d'ajouter simplement les activations transmises, toujours pondérées par α et par l'intensité des liens divisées par la somme des intensités des liens partant de chaque connaissance j :

$$l_i^{t+1} = \alpha \sum_j \frac{v_{ji}^t}{\sum_k v_{jk}^t} l_j^t$$

L'actualisation se transforme également simplement :

$$l_i^{t+1} = (1-\beta) l_i^t$$

L'équation globale de transmission de l'activation est donc :

$$l_i^{t+1} = (1-\beta) l_i^t + \alpha \sum_j \frac{v_{ji}^t}{\sum_k v_{jk}^t} l_j^t$$

Un autre avantage de cette fonction est que la variation de la valeur de l'activation totale contenue dans le graphe de connaissance peut être connue. Son évolution est la suivante :

$L^{t+1} = (1-\beta+\alpha) L^t$ en posant

$L^t = \sum_i l_i^t$ (total des activations entre 0 et +∞)

La variation de l'activation totale ne dépend donc que du degré d'actualisation (β) et du taux de transmission de l'activation entre les connaissances (α).

Par exemple, si on considère le graphe de connaissances réduit aux cinq connaissances suivantes sur lequel on a précisé la valeur de l'intensité des liens et les degrés d'activation des connaissances, en fixant α=0,8 et β=0,6:

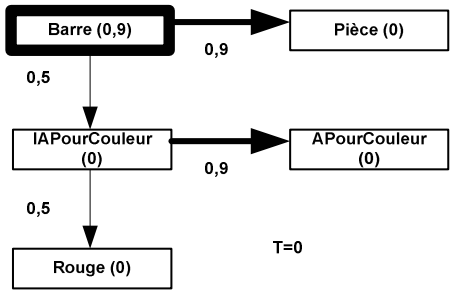



**Figure 5.5 Exemple de transmission d'activation : situation en t=0**

En T=0, la connaissance *Barre* est activée alors qu'aucune des autres ne l'est.

En T=1, elle va transmettre cette activation aux deux connaissances qui lui sont liées : *Pièce*, qui est fortement liée, va être activée à 0,694, et *IAPourCouleur*, qui est liée de façon moins importante, va être activée à 0,482.

Par exemple :

$$a_{ICouleur}^{t=2} = 1 - (1-0)^{1-0,6} (1-0,9)^{0,8\frac{0,5}{0,5+0,9}} = 0,482$$

$$a_{Barre}^{t=2} = 1 - (1-0,9)^{1-0,6} = 0,601$$

L'activation de *Barre* va ensuite être actualisée et tombe à 0,601.

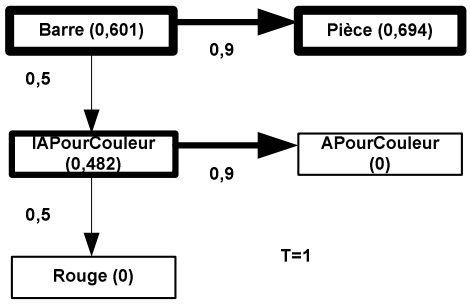



**Figure 5.6 Exemple de transmission d'activation : situation en t=1**

En T=2, *IAPourCouleur* va transmettre à *APourCouleur* et à *Rouge*, *Barre* va à nouveau transmettre à *Pièce* et *IAPourCouleur*, et ces trois connaissances vont être actualisées. Le graphe final sera :

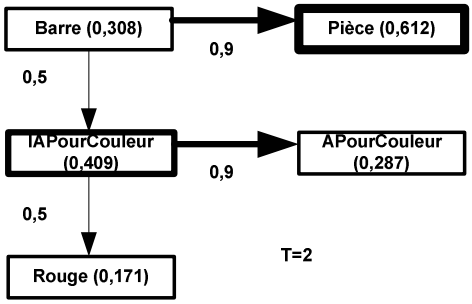



**Figure 5.7 Exemple de transmission d'activation : situation en t=2**

## 5.3. Exécution des règles explicites

Certains concepts du graphe représentent des règles explicites. Un certain nombre de ces concepts sont sélectionnés aléatoirement avec une probabilité proportionnelle à leur degré d'activation. Les règles qu'ils représentent sont alors interprétées dynamiquement.

La sélection aléatoire proportionnellement à l'activation permet d'utiliser toutes les règles, même si c'est avec une fréquence très faible pour celles qui sont peu activées. L'utilisation systématique des règles les plus activées sans choix probabiliste conduirait à créer des effets de seuils ne correspondant pas à l'objectif de contrôle décentralisé du formalisme. Ce n'est pas parce qu'une connaissance A est très légèrement moins activée qu'une autre B qu'elle ne doit pas être utilisée et prise en compte. Elle doit juste avoir moins de chances de l'être.

Le même mécanisme de sélection aléatoire proportionné à l'activation est également utilisé pour les variables dans les fonctions exécutées. Par exemple, si une règle cherche une action à analyser, elle choisira au hasard parmi les actions explicites du graphe proportionnellement à leur activation. Cela favorise les concepts qui sont a priori les plus utiles, car le plus liés à la situation actuelle, mais aussi ceux qui ont été utiles dans des situations semblables grâce au renforcement (voir chapitre 0).

## 5.4. Perceptions et actions

### 5.4.1. Perceptions (ou Sens)

#### 5.4.1.1. Principe

L'objectif du système perceptif est de permettre à tout type de sens d'introduire de l'information dans le graphe et à l'agent d'utiliser cette information. Pour être efficace, l'agent doit être capable d'agir de façon réactive (la perception modifie les connaissances de l'agent automatiquement) et proactive (l'agent consulte l'état de ses perceptions).

Pour cela, les perceptions (ou sens) activent des connaissances spécifiques dans le graphe et y introduisent un certain degré d'activation. Cette activation peut être utilisée de deux façons. De façon réactive, l'activation introduite dans le graphe est transmise à d'autres connaissances, permettant un raisonnement implicite à partir de ces perceptions et activant éventuellement des concepts plus abstraits que la perception brute qui pourront ensuite être utilisés explicitement. De façon proactive, une règle explicite peut lire la valeur de l'activation de la connaissance perçue (ou d'une autre connaissance décrivant un concept plus abstrait activé) pour pouvoir l'interpréter et l'utiliser.

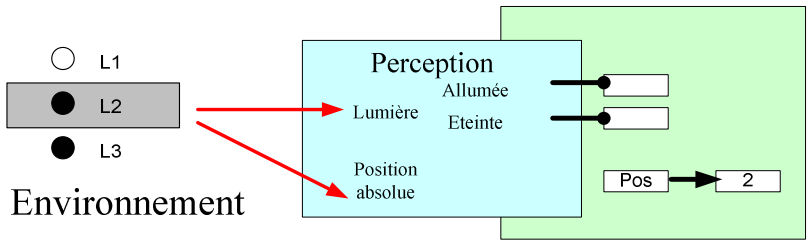



**Figure 5.8 Perception de l'état d'une lumière et de la position actuelle**

Par exemple, une perception binaire, comme l'état d'une lampe, sera liée à deux connaissances, *CoAllumée* et *CoEteinte*. Lorsque la lumière observée est allumée, *CoAllumée* est activée mais pas *CoEteinte*, et inversement si la lumière est éteinte. Cette information, présente dans le graphe de connaissances puisque *CoAllumée* et *CoEteinte* sont présentes dans ce graphe, peuvent être utilisée de façon proactive ou réactive :

De façon proactive, une règle peut comparer l'état des deux connaissances pour connaître l'état actuel de la lampe. Ainsi, la règle de mise à jour de l'état de la lampe, donnée à l'agent lors de sa création pour qu'il puisse utiliser explicitement l'information perçue, est définie par :

*Si Activation(CoAllumée)>Activation(CoEteinte) alors Lumière -> REtat = Allumée sinon Lumière -> REtat = Eteinte*

De façon réactive, une des connaissances peut être liée à une règle ou à n'importe quel autre concept qu'elle activera automatiquement lorsqu'elle sera elle-même activée.

Pour une perception plus complexe, où un grand nombre, voire un nombre infini, de stimuli différents, doivent pouvoir être perçus, il peut être impossible d'associer une connaissance à chaque cas possible. Le système perceptif peut alors soit créer une nouvelle connaissance correspondant à la valeur perçue et l'inclure directement en paramètre explicite de la connaissance correspondant à la perception. Ou bien il peut lier explicitement le concept décrivant la situation de la perception au concept correspondant à ce qui est perçu si celui-ci existe déjà dans le graphe.

Par exemple, pour percevoir la position actuelle, on peut ne pas souhaiter rentrer une connaissance par position possible dans le graphe. Dans ce cas, il y a deux possibilités :

Soit, à chaque fois que la position change, le système perceptif crée une nouvelle connaissance correspondant à la nouvelle position et l'ajoute en attribut explicite du concept de position actuelle.

Soit, si le nombre correspondant à la nouvelle position existe déjà dans le graphe, on ajoute cette connaissance-nombre en attribut explicite du concept de position actuelle.

Des règles peuvent également être données au départ pour décrire des perceptions plus complexes, et l'agent peut bien entendu créer lui-même des concepts plus abstraits et les règles d'identification associées. Une description du traitement explicite et de la création de nouvelles perceptions est donnée en section 8.3.1.

#### 5.4.1.2.Exemple particulier de perception : l'introspection

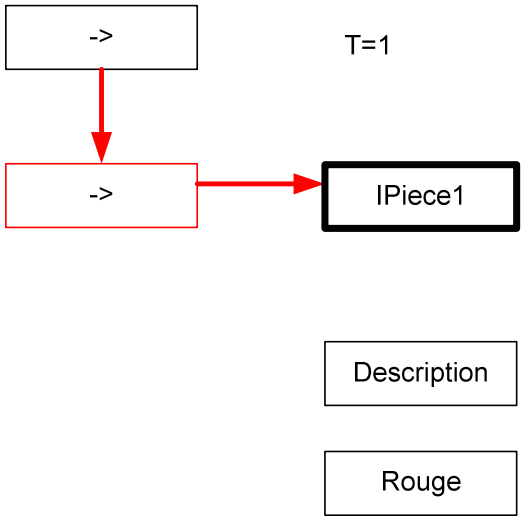
Les perceptions de l'agent sont naturellement fonction du domaine d'application de l'agent. Certaines perceptions, comme l'introspection, peuvent toutefois s'appliquer quelque soit le domaine.

L'introspection est un sens particulier permettant à l'agent d'observer les concepts qui ont suivi la plus forte variation d'activation au cours de la période précédente. Il crée une nouvelle connaissance qui est liée à tous les concepts ayant été fortement activés, lui permettant de savoir ce qu'il vient de faire et de penser. Par ce sens, il perçoit également les émotions qu'il vient de ressentir, pour pouvoir chercher à les reproduire plus tard, consciemment (par des règles explicites) ou inconsciemment (par réactivation automatique). Ce sens permet enfin à l'agent de garder une trace implicite des changements dans ces connaissances et donc indirectement dans le monde. Il peut utiliser ce savoir pour reproduire un évènement temporel ou simplement pour l'analyser et le comprendre. Le choix des connaissances observées se fait en fonction de règles implicites dirigeant l'attention de l'agent. Ces règles définissent les concepts à observer. Plus une connaissance est éloignée d'un concept observé, plus il faudra qu'elle subisse une forte variation d'activation pour que celle-ci soit rapportée. Ainsi, un agent s'intéressant au suivi de ses objectifs va centrer son attention sur le concept « objectif ». Les connaissances fortement liées à ce concept et qui subissent une faible variation seront rapportées. Par contre un changement dans la perception visuelle, par exemple, devra être très important pour être rapporté.

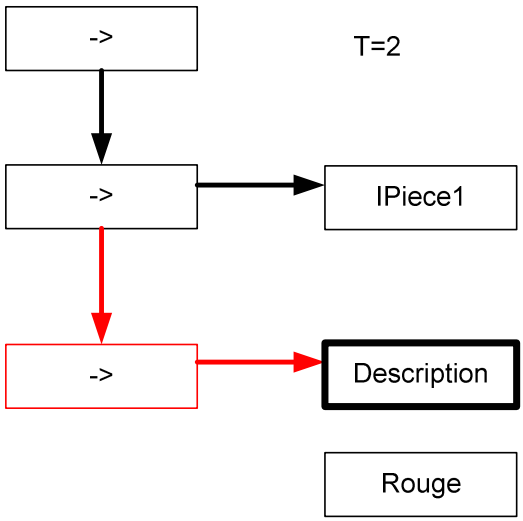
A chaque période, l'agent crée donc une connaissance « suivi de » (ou « -> »), il lie la connaissance « suivi de » précédente à la nouvelle et lie toutes les connaissances dont une

modification a été rapportée à la nouvelle (avec une intensité fonction de la variation de l'activation).

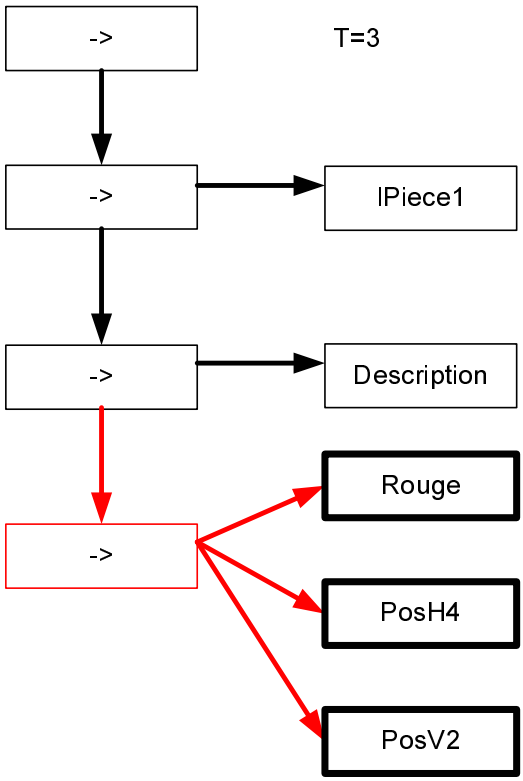
Concrètement, le résultat obtenu est un arbre de connaissances dont le tronc, ou fil conducteur temporel, est constitué de connaissances « suivi » :




Par exemple, si en T=1 la connaissance IPiece1 est très activée (alors qu'elle ne l'était pas auparavant), l'introspection crée une nouvelle connaissance *suivi* (en rouge) et crée deux liens (en rouge) : un depuis l'ancienne connaissance *suivi* vers la nouvelle et un depuis cette nouvelle connaissance vers IPiece1



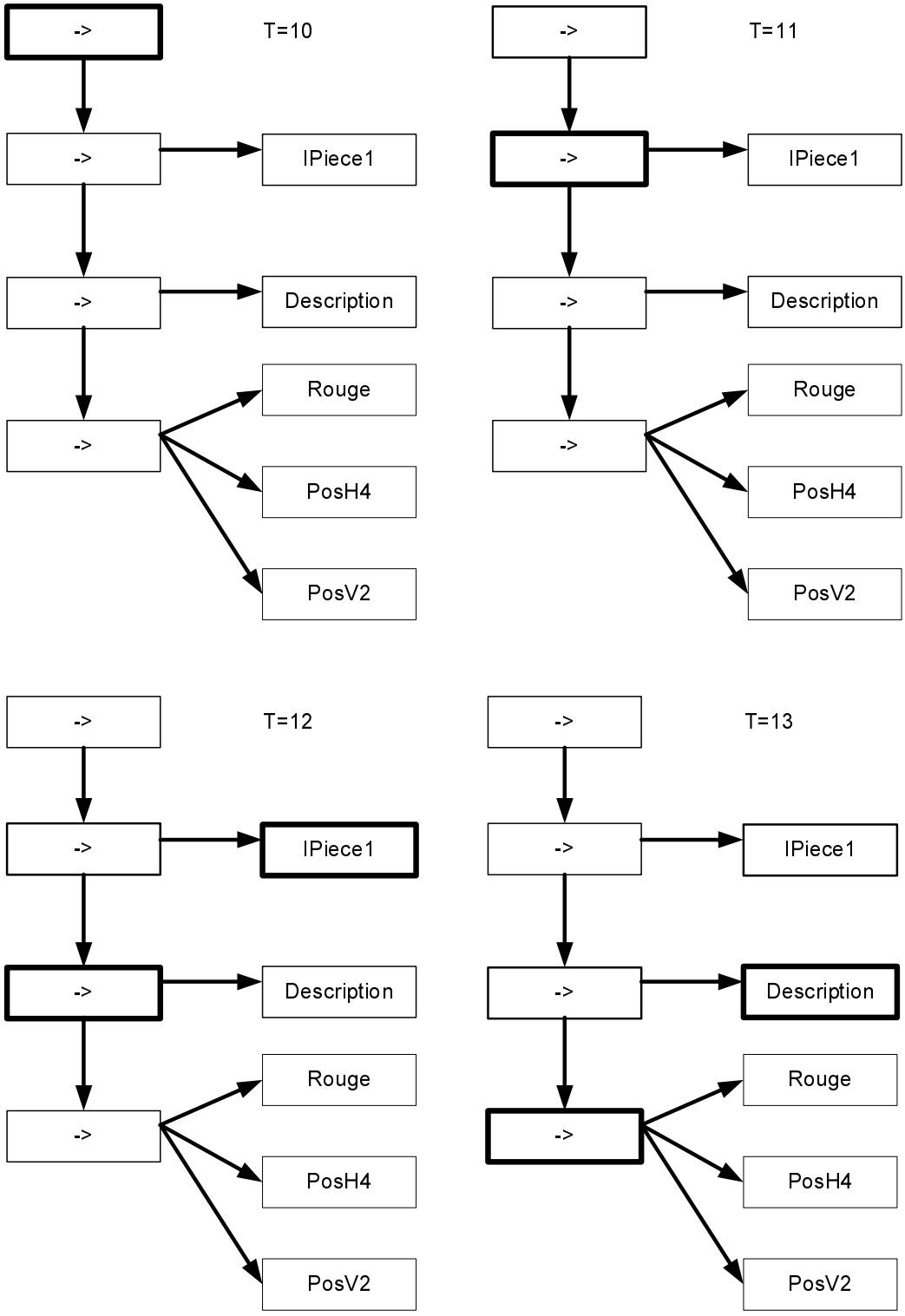
De même, si en T=2 la connaissance description connaît une forte augmentation de son niveau d'activation, une nouvelle connaissance est créée, un lien avec la connaissance *suivi* est ajouté ainsi qu'un lien vers la connaissance Description.




Enfin, si en T=3, à la fois la connaissance *Rouge* est fortement activée et les connaissances *PosH4* et *PosV2* sont créées et activées, ces trois connaissances sont liées à la nouvelle connaissance suivi créée.

Cette méthode présente plusieurs avantages :

- Une activation d'une connaissance « suivi » provoque ainsi une répétition des événements observés dans les instants qui ont suivi en conservant la chronologie. Ainsi, dans l'exemple précédent, si en T=10 la première connaissance *suivi* est activée, en T=11 la deuxième suit, en T=12, *IPiece1* et la troisième, en T=13, Description et la quatrième connaissance suivi sont activées. L'ordre d'activation correspond à l'ordre d'activation d'origine.




- La dimension temporelle est modélisée et peut être analysée grâce aux connaissances « suivi »
- La perception et l'analyse d'un changement dans les connaissances peuvent être effectuée.

- De façon inverse, on peut réaliser des règles d'anticipation ou d'imagination qui appliquent des changements observés précédemment à la situation actuelle, tout en conservant la possibilité de les supprimer par la suite

### 5.4.2. Actions

Comme pour les perceptions, une action effectue sa fonction si l'activation de la connaissance particulière impliquant cette action est supérieure à celle l'inhibant. Cela permet à l'agent d'agir sur l'environnement. L'agent peut avoir connaissance de la réalisation d'une action soit par introspection en analysant la variation des connaissances liées à l'action, soit en percevant les changements consécutifs à son action dans l'environnement, soit enfin grâce à un sens spécifique à cette action.

Par exemple, l'action de presser/relâcher un bouton a deux connaissances associées : *CoPresser* et *CoRelacher*. Il existe de plus un sens associé au bouton qui lui permet de connaître sa situation actuelle au travers de deux connaissances *CoSePressé* et *CoSeRelaché*.

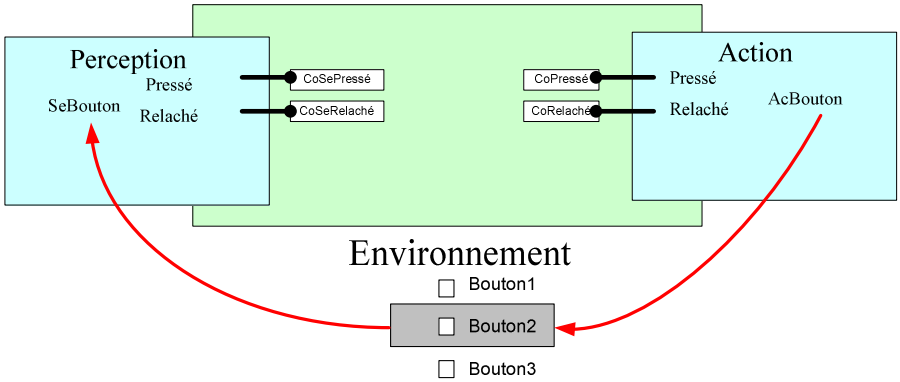



**Figure 5.9. Action de presser sur un bouton et perception directe de cette action**

Une règle est associée à l'action pour que l'agent puisse l'utiliser explicitement dans son raisonnement. Elle est définie de la façon suivante:

*PresserBouton : Activer CoPressé jusqu'à ce que Bouton->etat=pressé*

Cette règle permet d'activer la connaissance provoquant l'action. L'attente jusqu'à ce que l'état du bouton soit perçu comme pressé implique que la règle est placée en attente jusqu'à ce que la condition soit remplie, ou qu'elle soit retirée de l'ensemble des règles en cours. On peut remarquer ici qu'une autre règle peut simultanément inhiber la connaissance activée *CoPressé* ou au contraire activer *CoRelaché*, provoquant dans les deux cas l'effet

inverse de *PresserBouton*. L'action sera alors le résultat de plusieurs règles concurrentes. Celles qui seront le plus activées décideront indirectement de l'action menée car c'est la connaissance qu'elles stimuleront qui sera la plus activée.

Des règles d'action plus complexes peuvent être fournies à l'agent et il peut en concevoir lui-même.

Une description du fonctionnement explicite des actions et de leur création est donné en section 8.3.2.

## 5.5. Régulation par les émotions

Comment savoir si l'agent va activer les bonnes connaissances ? On veut que l'agent favorise certains comportements dans certaines circonstances. Comme on ne peut pas décrire les connaissances qu'on veut favoriser (les connaissances variant constamment), on définit des facteurs d'évaluation portant soit sur l'extérieur (l'environnement) soit sur le fonctionnement de l'agent. Ces facteurs vont influencer le comportement de l'agent de deux façons : directement par modification de ses paramètres (par introduction d'activation dans le graphe et par renforcement des liens) et par la perception qu'il a de ces facteurs. Pour simplifier et pour leur similitude fonctionnelle, on appelle ces fonctions émotions. Ainsi, si on veut un agent curieux, on va ajouter une émotion (positive) curiosité. L'activation de cette émotion augmentera lorsque l'agent créera de nombreuses connaissances (il acquiert ainsi de nouvelles informations). Les liaisons entre les connaissances activées précédemment (y compris celles créées par introspection) seront alors renforcées ce qui conduira l'agent à reproduire son comportement de façon implicite. De façon explicite, l'agent percevra cette augmentation de l'activation de l'émotion et fera un rapport. Il pourra alors apprendre que la méthode qu'il a utilisée conduit à une émotion positive et il cherchera à la reproduire.

De même, la recherche des points lors d'une partie de Tetris ou d'un autre jeu se fait, bien sûr, de façon explicite (réponse à des objectifs du type « Comment gagner des points ? »). Cependant, le choix (implicite) de ces objectifs et l'influence implicite pour le choix des règles implicites et explicites utilisées pour l'atteindre sont guidés par l'émotion de jeu qui est activée (provoquant ainsi un renforcement) et perçue lorsque des points sont marqués. Il faut noter que ce sont les variations des émotions qui sont perçues et non leur valeur.

Une étude détaillée du contrôle et des émotions est donnée au chapitre 0.

## 5.6. Oubli

L'introspection et l'interprétation des règles explicites conduisent à une importante création de connaissances. L'oubli est donc une fonction fondamentale pour que le système n'explose pas rapidement. On peut distinguer deux formes d'oublis.

La première est l'oubli des connaissances qui ne pourront de toute façon plus servir. Cette suppression se fait en recherchant les connaissances qui n'ont aucun lien qui pointe sur elles, et qui ne sont donc plus susceptibles d'être activées. Ceci est valable non seulement pour les connaissances isolées, mais aussi pour les groupes de connaissances qui n'ont aucun lien pointant vers le groupe (et dont aucune connaissance n'est plus activée). Ces connaissances ou groupes de connaissances peuvent alors être retirés du graphe.

Un deuxième type d'oubli est un oubli dû à la dégradation des liens afin de permettre l'effacement de souvenirs, même faiblement liés à d'autres connaissances. Pour cela, les liens avec une stabilité très faible (comme ceux issus de l'introspection s'ils ne sont pas renforcés ou utilisés) voient leur stabilité encore diminuer jusqu'à atteindre 0. Si la stabilité atteint 0, le lien peut être supprimé, et la connaissance pourra alors être supprimée du fait de la première forme d'oubli.

## 5.7. Conclusion

Dans ce chapitre, nous avons présenté un modèle d'agent capable d'utiliser efficacement des concepts et des règles apprises. Pour avoir un comportement adaptable, ce fonctionnement est indépendant des règles et de la sémantique de l'agent.

Le principe de base du fonctionnement est la transmission de l'activation à travers le graphe de connaissances de l'agent. Les concepts et les règles utilisés à un moment donné sont choisi de façon probabiliste parmi ceux qui sont les plus activées. Ce fonctionnement présente l'avantage de ne considérer pour les calculs qu'une partie du graphe, de taille fixe. La vitesse de fonctionnement des différentes phases n'est donc pas fonction de la taille totale du graphe.

Les concepts les plus activés à un instant donnés constituent le champ d'attention, c'est-à-dire l' « état mental » actuel de l'agent. L'activation provient de trois origines. En premier lieu, des concepts activés eux-mêmes, ils vont transmettre leur activation à leurs voisins. L'état mental actuel va donc déterminer l'état mental suivant. En second lieu, les perceptions vont également introduire de l'activation dans le graphe, elles fonctionnent donc aussi bien de façon réactive (par introduction d'activation) que de façon proactive (par exécution de règles

déterminées par l'état mental actuel de l'agent). Enfin, les émotions introduisent également de l'activation et renforcent les liens entre les connaissances, permettant de guider l'agent indépendamment de sa sémantique (en fonction de l'état de l'agent et de son environnement).

# Chapitre 6
# Analyse de données symboliques et graphes de connaissances

## 6.1. Introduction

Comment l'agent peut-il apprendre de nouveaux objets et règles à partir de concepts représentés sous la forme de graphes ? Comment comparer et généraliser ces concepts ?

Lorsque les connaissances d'un système sont représentées par des données numériques, l'induction et la visualisation des données peuvent se faire grâce à des systèmes d'analyse de données classiques. On peut ainsi utiliser une Analyse en Composantes Principales, une classification hiérarchique ou une méthode descendante pour visualiser les grands groupes de données et éventuellement en déduire des classes abstraites pertinentes pour le programme.

Dans un système où les données sont représentées sous forme de graphes, l'utilisation de ces outils n'est plus possible. Comment transformer numériquement un graphe de connaissances sans perdre d'informations importantes ? Le principe de la représentation et de l'induction reste toutefois utile. A partir d'un graphe de connaissances très complexe, il serait intéressant de visualiser les grands groupes de concepts utilisés et leur évolution. Il serait également utile de pouvoir trouver des concepts présentant de fortes similarités pour pouvoir créer des concepts plus abstraits.

L'analyse de données symboliques (ADS) offre un cadre et des outils pour modéliser et utiliser des données symboliques. L'objet symbolique (OS), permet de modéliser les concepts et ainsi de pouvoir les utiliser de façon autonome au cours de l'analyse.

Notre objectif est de réaliser une analyse sur des données représentées sous forme de nœuds et de liens puis d'utiliser les concepts découverts pour pouvoir compléter les connaissances de l'agent [Caillou et Diday 2005]. La modélisation est décrite en section 6.2. Une mesure de similarité entre graphes adaptée a donc été développée (voir section 6.3) pour

permettre l'application des différents outils de l'ADS (voir section 6.5). Notre objectif étant d'utiliser les nouveaux concepts appris, un algorithme permettant d'introduire un objet symbolique appris dans le graphe comme un nouvel individu a été développé (voir section 6.6). Quelques résultats obtenus grâce à cette méthode sont présentés en section 6.7.

## 6.2. Modélisation

L'analyse de données symboliques offre un cadre d'analyse pour traiter tout type de données, mais aussi pour modéliser et utiliser les concepts sous-jacents. La modélisation et les notations utilisées peuvent se résumer à l'aide de la figure 6.1.

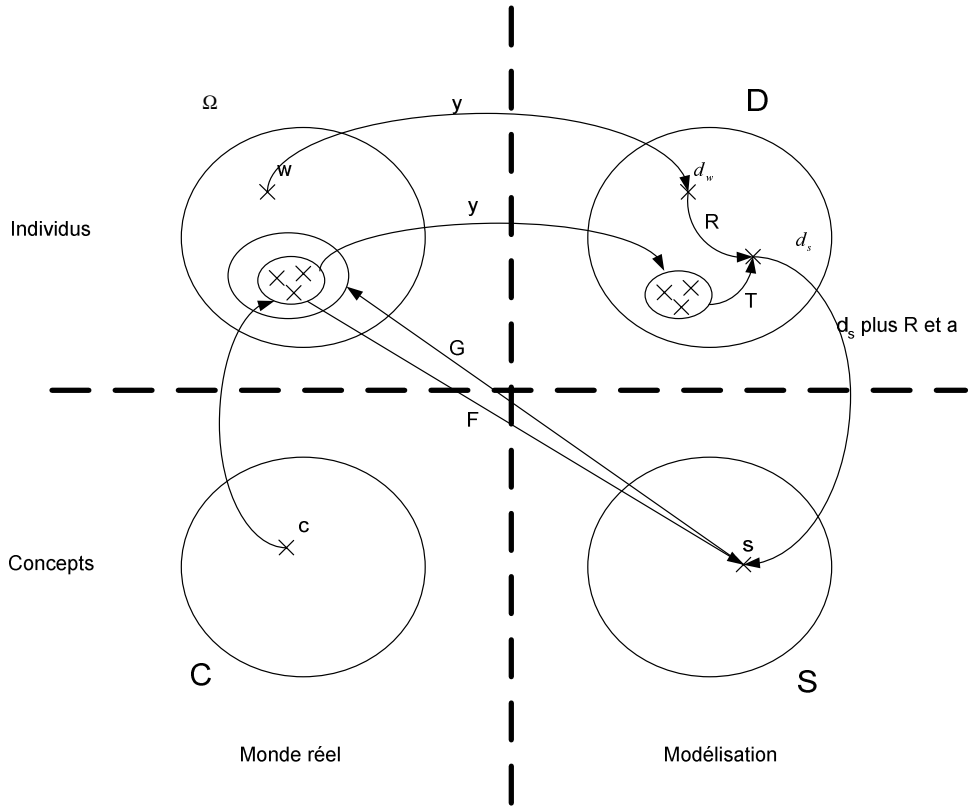



**Figure 6.1 Ensembles utilisés par l'analyse de données symboliques : l'ensemble des descriptions D permet de décrire les individus du monde réel éléments de Ω, tandis que l'ensemble des objets symboliques S permet de modéliser les concepts du monde réel éléments de C [Diday 2000a].**

Les données traitées lors de l'analyse (les $d_w$) correspondent à la modélisation des individus du monde réel ($w$). A partir des descriptions de ces individus, les outils d'analyse de données symboliques permettent d'extraire des objets symboliques ($s$) correspondant à une modélisation de concepts du monde réel ($c$). Ces objets symboliques sont définis par un triplet

(*a,R,d*) : la fonction *a* permet de déterminer si un individu *w* fait partie du concept décrit par *s* à partir de la description de *s* (i.e. *d*) et des relations utilisées (*R*). Une description générale de l'analyse de données symboliques peut être trouvée dans [Diday 2000b, a]. Nous allons détailler certains de ces éléments et les appliquer à notre modèle.

## 6.2.1. L'espace des individus Ω et leur modélisation

### 6.2.1.1.Présentation

L'ensemble Ω désigne l'ensemble des individus *w* du monde réel. Les individus sont modélisés dans un espace de descriptions *D* qui exprime leurs propriétés à l'aide de variables qui les caractérisent. Une description est constituée d'un ou plusieurs produits cartésiens exprimant ces propriétés.

*y* est une application de Ω dans *D* dite de « description des individus » car elle associe une description à chaque élément de Ω : $y(w)=d_w$

### 6.2.1.2.Application

#### 6.2.1.2.1. Ensemble Ω des individus

Notre objectif est d'analyser les connaissances de l'agent à partir de son graphe pour extraire de nouveaux concepts pertinents par rapport à l'environnement. Le monde réel (pour notre analyse) correspond alors au graphe de l'agent lui-même, et non à l'environnement de l'agent.

Pour bien comprendre notre modélisation, il est possible de faire une analogie avec le web mining et l'analyse de pages web. Le monde réel lors d'une telle analyse est l'ensemble des pages web, et non le monde qu'elles décrivent. Ces pages vont ensuite être décrites (passage de Ω à D) pour être analysées. De nombreuses descriptions sont possibles, notamment à cause des liens hypertextes entre pages. Une page web n'est souvent rien sans les pages auxquelles elle accède. C'est en particulier vrai pour les pages de menu, qui peuvent pourtant être les plus intéressantes car croisant de nombreuses informations que peut rechercher l'utilisateur. Il faut alors choisir ce qui la décrit le mieux : son contenu seul, son contenu ainsi que le contenu des pages accessibles directement, le contenu ainsi que les pages accessibles en trois clics, le contenu ainsi que les pages du même site, etc. De nombreuses descriptions du même monde réel sont possibles et vont déterminer la qualité de l'analyse et la

nature des résultats. A chaque page web peut donc être associé un individu w. Bien que cet individu soit clairement identifié, sa définition est floue et plusieurs descriptions sont possibles.

De façon analogue, à chaque connaissance (nœud) du graphe de l'agent peut être associé un « concept fluide » (que nous appellerons *Frame* pour le distinguer des *concepts* de l'analyse de données symboliques). Cet individu est clairement identifié (le nœud), mais sa définition est floue. En effet, une Frame se définit par sa connaissance racine et par toutes les autres connaissances (le sous-graphe) qu'elle peut activer, directement ou indirectement, à travers les liens. Il est donc possible de décrire une Frame avec des sous-graphes plus ou moins complexes en fonction des critères choisis (intensité minimale des liens, nombre de liens intermédiaires,…). Pour plus de détails sur les concepts fluides, les Frames et la définition floue des concepts, voir section 4.4.

Il y a donc autant d'individus possibles que de connaissances (nœuds) dans le graphe. Le grand nombre d'individus existant ne signifie pas qu'ils sont tous analysés. Le choix des individus analysé est laissé à l'agent (de même que pour une analyse de web mining l'utilisateur peut choisir d'analyser uniquement les pages d'un domaine ou les pages racines des sites).

Nous utiliserons les notations suivantes pour désigner le graphe :

- $C$ désigne l'ensemble des connaissances (nœuds) du graphe. Une connaissance $c \in C$ est liée à d'autres connaissances au travers de liens.
- $L$ désigne l'ensemble des liens du graphe. Un lien $l \in L$ est orienté et possède une connaissance origine (*origine(l)*), une connaissance cible (*cible(l)*) et une intensité comprise entre -1 et 1 (*intensité(l)*). Pour simplifier l'analyse, on ne considère qu'une seule des valeurs des liens décrites au Chapitre 4 : l'intensité, le facteur de stabilité n'ayant pas de valeur sémantique. Pour être significatif, on ne prend en compte que les liens dont le facteur de stabilité est supérieur à un certain seuil (0,3), la valeur d'intensité pouvant alors être considérée comme relativement établie.
- $\Omega$ désigne l'ensemble des Frames, ou sous-graphes, centré autour de chacune des connaissances du graphe. Chaque individu $w \in \Omega$ est un sous-graphe identifié à partir d'une connaissance racine $c$.

Pour illustrer notre modèle, nous allons considérer le graphe représenté figure 6.2. Ce graphe contient 15 connaissances. Il contient donc potentiellement 15 individus analysables. Nous choisissons ici d'analyser les individus qui sont des instances de *Pièce* (c'est-à-dire les connaissances possédant un lien d'intensité élevée avec la connaissance *Pièce* ; pour plus de détail sur l'instanciation, voir section 4.4). Nous allons donc considérer trois individus : les trois sous-graphes centrés sur les connaissances *IPièce1*, *IPièce2* et *IPièce3*. Ces trois sous-graphes peuvent être représentés sous forme de Frame pour plus de lisibilité (voir figure 6.3). *IPièce1* et *IPièce2* sont rouges, *IPièce3* est bleue et *IPièce1* est de taille 2 (les autres sont de taille inconnue).

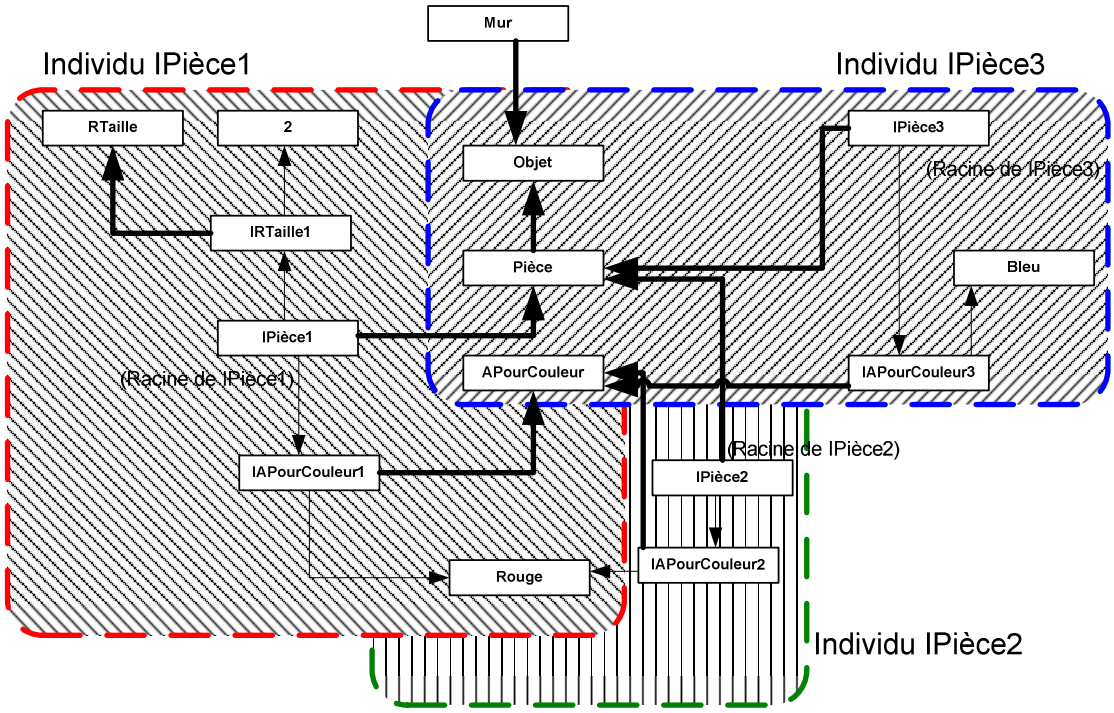



**Figure 6.2 Exemple de graphe contenant trois instances de *Pièce* correspondant aux individus *IPièce1*, *IPièce2* et *IPièce3***


**Figure 6.3 Représentation sous forme de Frame des individus *IPièce1*, *IPièce2* et *IPièce3***

#### 6.2.1.2.2. Description des individus

Un individu est identifié par sa connaissance racine.

D'après la sémantique fonctionnelle de nos liens, une bonne description d'un individu est de considérer l'ensemble des connaissances activables au travers des liens depuis la connaissance racine. Un individu est donc décrit par tous les liens de son sous-graphe.

Chaque lien est décrit par sa connaissance origine, sa connaissance cible et son intensité. On peut donc définir son domaine de définition $\Lambda$ :

$$\Lambda = \left(C \times C; IR\right)$$

Un individu est décrit par un ensemble de liens. La variable $y_l$ correspondante prend donc ses valeurs dans P($\Lambda$). Il s'agit d'une variable symbolique multi-valuée telle que définie par [Bock 2000].

Le sous-graphe considéré est défini par l'ensemble des connaissances activable par la connaissance racine de l'individu. C'est-à-dire toutes les connaissances $c \in C$ pour lesquelles il est possible de trouver une chaîne de liens partant de la connaissance racine et arrivant à *c*.

Les liens étant définis par leurs connaissances origine et cible, l'utilisation d'une variable pour les connaissances du sous-graphe donnerait une information redondante par rapport aux informations contenues dans les liens.

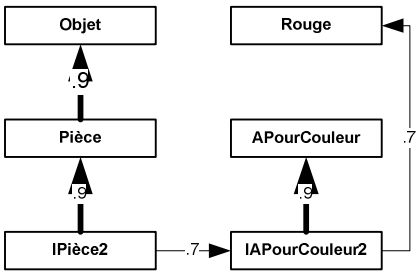
Les liens décrivant un individu sont les liens du graphe qui sont atteignables depuis la racine de l'individu. Il s'agit donc de tous les liens partant de la racine et de tous ceux partant d'une connaissance pour reliée par une chaîne de liens à la racine.

$$y_l\left(w\right) = \left\{ lien \in L \,/\, \begin{array}{l} \text{soit} : \exists l_1 ... l_k \in L^k \; k \in \mathbb{N}^* \; avec \\ origine(l_1) = racine(w), \\ cible(l_k) = origine(lien), \\ \forall t \in [\![1; k-1]\!], cible(l_t) = origine(l_{t+1}) \\ \\ \text{soit} : origine(lien) = racine(w) \end{array} \right\}$$

De façon équivalente, il est possible de définir cette variable récursivement.

$$y_l\left(w\right) = \left\{ lien \in L \,/\, \begin{array}{l} \text{soit} : origine(lien) = racine(w) \\ \\ \text{soit} : \exists lien' \in y_l(w) \; avec \\ cible(lien') = origine(lien) \end{array} \right\}$$

Par exemple, la description de l'individu *IPièce2* est donnée figure 6.4. L'individu est décrit par ses cinq liens. Chaque lien est lui-même décrit par son origine, sa cible et son intensité.




$$d_{IPièce2} = \left\{ \begin{array}{l} \{Racine, Pièce; 0,9\}; \\ \{Pièce, Objet; 0,9\}; \\ \{Racine, IAPourCouleur2; 0,7\}; \\ \{IAPourCouleur2, APourCouleur; 0,9\}; \\ \{IAPourCouleur2, Rouge; 0,7\} \end{array} \right\}$$

**Figure 6.4 Individu *IPièce2* et sa modélisation**

L'intensité des liens variant entre -1 et 1, on peut représenter cette variable comme une suite de couples associés à une valeur réelle, donc sous une forme d'histogramme :

$$d_{IPièce2} = \left\{ \begin{array}{l} \{Racine, Pièce; 0,9\}; \\ \{Pièce, Objet; 0,9\}; \\ \{Racine, IAPourCouleur2; 0,7\}; \\ \{IAPourCouleur2, APourCouleur; 0,9\}; \\ \{IAPourCouleur2, Rouge; 0,7\} \end{array} \right\}$$

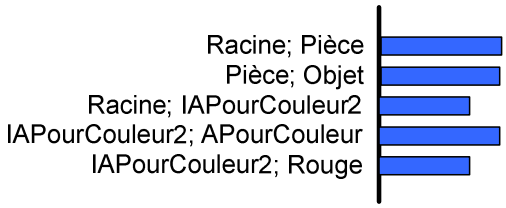



**Figure 6.5 Représentations de la description de l'individu *IPièce2***

Dans la description, la connaissance racine de l'individu constitue un cas particulier. Chaque individu se définit en effet par rapport à un nœud central (racine) qui va activer les autres nœuds à travers les liens. Or, pour comparer les individus entre eux, seules comptent les connaissances qui sont activées depuis la racine, et non la racine elle-même. Lors de la description d'un individu les liens dont l'origine est la racine de l'individu sont donc décrit comme partant d'une connaissance générique *Racine*.

Un tableau de données correspond ainsi à un ensemble d'individus décrits par leurs liens. Par exemple, si on considère les individus présentés en figure 6.3, on obtient le tableau de la figure 6.6

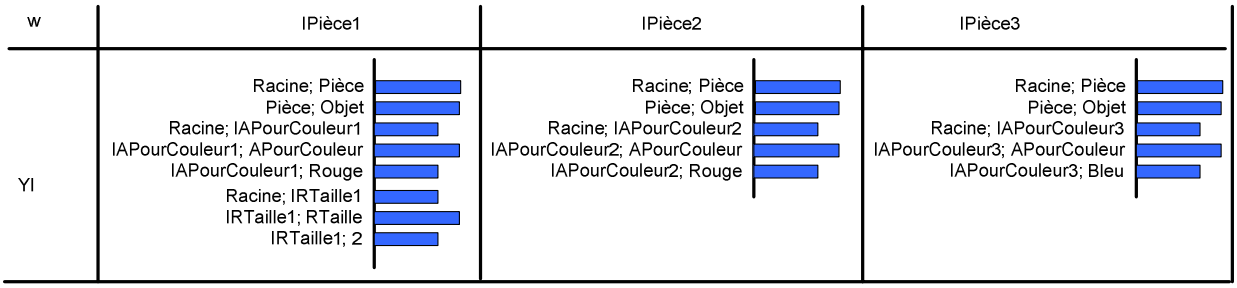



**Figure 6.6 Exemple de tableau de données avec 3 pièces**

Pour un individu complexe, comme une pièce avec sa description, le nombre de liens peut vite devenir important. Ainsi, pour une description très simplifiée d'une pièce composée de deux briques (ou pixels) avec sa position en abscisse, sa taille et sa couleur, on obtient le graphe représenté figure 6.8 (en omettant des nœuds tels que *Nombre*, *Couleur*, …) correspondant à la Frame de la figure 6.7. La description sous forme de suite de liens est présentée figure 6.9.

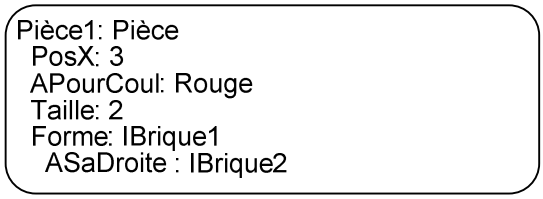



**Figure 6.7 Représentation sous forme de Frame de l'individu Pièce1**

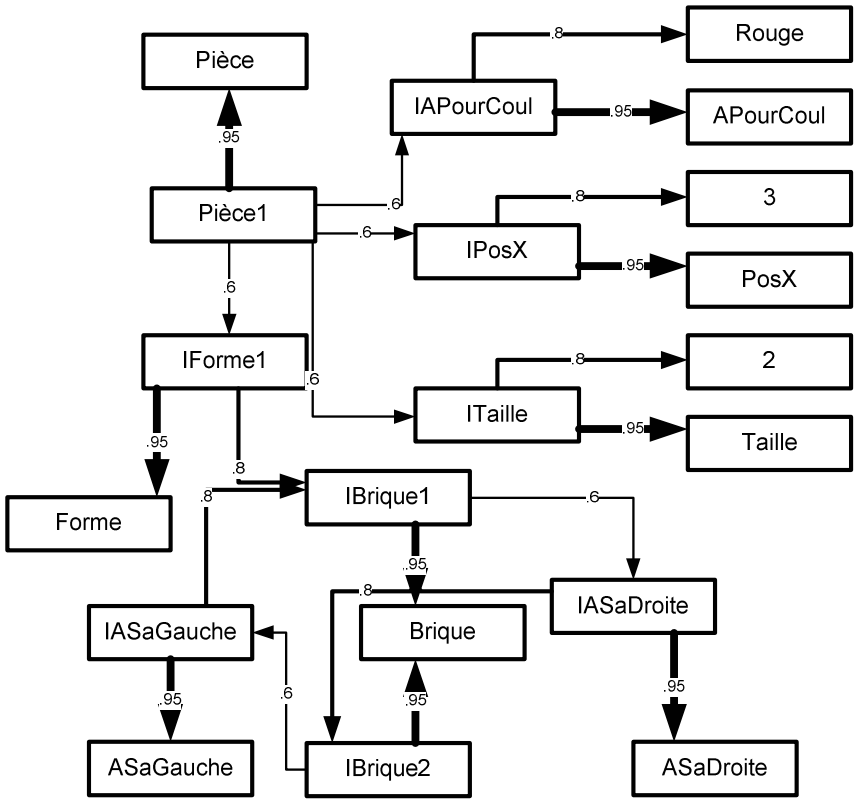



**Figure 6.8 Représentation sous forme de Graphe de l'individu Pièce1**

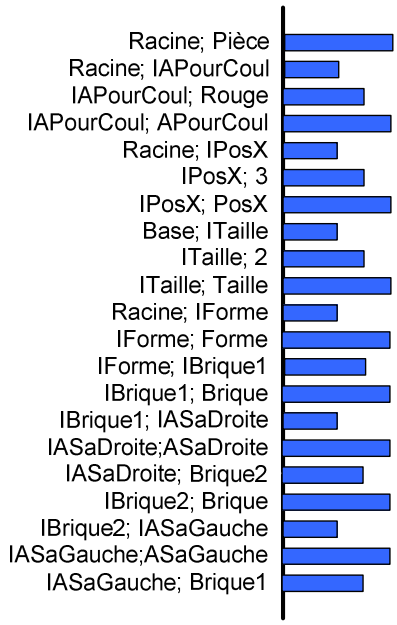



**Figure 6.9 Description de l'individu Pièce1**

## 6.2.2. Concepts et objets symboliques

### 6.2.2.1.Présentation

Les objets symboliques (OS) permettent de modéliser les concepts du monde, qu'ils soient issus directement d'un individu, d'une classe d'individus ou d'un outil d'analyse. On rappelle que l'ensemble des objets symboliques est noté $S$ et ses éléments $s \in S$. C'est l'élément clef de l'analyse qui correspond à une modélisation d'un concept abstrait. C'est lui que l'on cherche à décrire, à comprendre et à découvrir. Si un concept est bien modélisé, l'extension de l'objet symbolique contient un individu si et seulement si cet individu appartient à l'extension du concept.

Un objet symbolique est défini par :

- Une description notée $d \in D$
- Une relation binaire $R$ sur $D$ permettant de comparer $d$ à une autre description de $D$
- Une fonction $a$ permettant d'évaluer le résultat de la comparaison (à l'aide de $R$) de la description d'un individu de $\Omega$ par rapport à la description donnée $d$.

Un OS peut être obtenu en particulier à partir d'une fusion de plusieurs individus. L'OS obtenu modélise alors la classe qui contient les individus fusionnés. L'avantage des OS est qu'une fois créés, ils constituent une entité autonome, qui peut être elle-même fusionnée (pour créer des classes plus générales), décrite ou encore comparée à l'aide de sa description.

### 6.2.2.2.Application

Dans notre application, un concept, ou une classe d'individus, peut être décrit au travers de liens minimaux qu'un individu doit posséder pour appartenir au concept. La description de l'OS sera donc un « squelette » minimum de sous-graphe.

La description $d$ d'un OS sera constituée de la même variable multivaluée $y_l$ (décrivant un ensemble de liens) que les individus.

Par exemple, le concept d' « instance de Pièce » peut se décrire avec un unique lien de *Racine* vers *Pièce* avec une intensité élevée de 0,9. Tout individu qui active suffisamment la connaissance *Pièce*, directement ou indirectement, à partir de sa connaissance racine devant logiquement être une instance de *Pièce*.

La description de l'OS InstPiece est :

$$d_{InstPiece} = \{\{Racine, Pièce; 0, 9\}\}$$

La relation $R$ permet de comparer la description $d$ de l'OS et une autre description $d' \in D$ (généralement issue d'un individu). Intuitivement, on aura *d'Rd* si tous les liens de $d$ existent dans *d'* avec une intensité supérieure et éventuellement des liens intermédiaires. L'existence possible de liens intermédiaires dans *d'* nous conduit à définir de façon plus précise la fonction caractéristique de $R$, $h_R$ (c'est-à-dire $h_R(d', d) = 1$ si et seulement si *d'Rd* et 0 sinon).

$$\begin{array}{rccl} h_R : & D \times D & \rightarrow & \{0;1\} \\ & (d';d) & \mapsto & \left| \begin{array}{l} 1 \; si \; \forall lien \in d, \exists \left(l'_1 ... l'_k\right) \in d'^k, k \in \mathbb{N}^* \; avec \\ \quad origine(l'_1) = origine(lien) \\ \quad cible(l'_n) = cible(l) \; avec \\ \quad \forall t \in [\![1; k-1]\!], cible(l'_t) = origine(l'_{t+1}) \; et \\ \quad \underset{j=1..k}{Min}\left(intensité(l'_j)\right) \geq intensité(l) \\ 0 \; sinon \end{array} \right. \end{array}$$

Par exemple, la description *d'* d'un d'individu sera reliée à la description *d* de *InstPièce* si *d'* contient une chaîne allant de *Racine* vers *Pièce* avec une intensité minimum pour la chaîne de 0,9.

Comme on ne peut avoir *d'Rd* et *dRd'* que si *d=d'*, on peut remarquer que R définit ici un ordre partiel sur D. Toutefois, cet ordre n'est pas défini sur l'ensemble des individus Ω car deux individus différents peuvent avoir une description identique (car la connaissance centrale, qui est différente pour tous les individus, est toujours remplacée par la même connaissance Racine dans la description).

La fonction a de reconnaissance est ici une fonction binaire qui vérifie si d'Rd. Si les liens d'un individu sont au moins aussi fort que ceux de l'OS, *a(w)=vrai*. Si au moins un lien est manquant, même en considérant les chaînes de liens intermédiaires, ou si l'intensité est inférieure à celle de l'OS, *a(w)=faux*.

$$\begin{array}{rccl} a : & \Omega & \rightarrow & \{0;1\} \\ & w & \mapsto & \left| \begin{array}{l} 1 \; si \; d_w R d \\ 0 \; sinon \end{array} \right. \end{array}$$

Ainsi

$$a_{s_{InstPiece}}(IPièce1) = vrai$$

La définition de l'objet symbolique InstPièce peut s'écrire :

$$a_{InstPiece} = \left[ y_l \supseteq \left\{ (Racine, Piece; 0,9) \right\} \right]$$

On distinguera donc ici le concept (abstrait, correspondant à l'objet symbolique), du graphe, qui lui est observé. Un concept se définit en intention par les connaissances et les liens qui le composent. Il décrit les connaissances qu'un individu doit activer (et dans quelles proportions) pour faire partie de son extension. Un individu (graphe) appartiendra à l'extension d'un objet symbolique s'il dispose des connaissances nécessaires dans le graphe et que ces connaissances sont activées avec autant ou plus de force que dans le concept, c'est-à-dire si les liens vers ces connaissances sont au moins aussi forts que ceux de l'OS.

Un exemple de pièce de couleur rouge est :

$$a_{PièceRouge} = \left[ y_l \supseteq \left\{ (Racine, Piece; 0,9) \right\} \right] \wedge \left[ y_l \supseteq \left\{ (Racine, Rouge; 0,7) \right\} \right]$$

Un individu appartiendra à l'extension de ce concept si son graphe contient les connaissances *Rouge* et *Pièce*, et que sa racine possède un lien vers *Pièce* (avec éventuellement des connaissances intermédiaires) dont l'intensité minimum est 0,9 et vers *Rouge* au minimum 0,7. *IPièce1* et *IPièce2* appartiennent à l'extension de *PièceRouge*, *IPièce3* ne lui appartient pas.

## 6.3. Calcul de le similarité

L'analyse de données symboliques permet d'utiliser des données quelconques et non pas seulement des données numériques. Pour cela, il faut définir une mesure de dissimilarité ou de similarité entre ces données. Le choix de cette mesure détermine la qualité du résultat final. Il se fait en fonction de la sémantique associée à la forme des données. Ainsi, il existe de nombreuses similarités entre des intervalles ou des variables multivaluées en fonction de l'interprétation faite de la représentation.

Pour plus d'efficacité, nous réalisons un prétraitement des données utilisées.

### 6.3.1. Prétraitement des données

Notre objectif est que deux individus soient d'autant plus similaires qu'ils peuvent activer un grand nombre de connaissances communes. Une modification des valeurs associées aux liens et l'ajout de liens virtuels facilitent cette comparaison.

#### 6.3.1.1.Modification des intensités des liens

La valeur d'intensité associée à la description du lien est modifiée de sorte qu'elle ne corresponde pas directement à l'intensité du lien dans le graphe. On considère en effet que l'intensité décrite correspond à l'activation maximale qui peut passer par ce lien en provenant de la racine. La valeur est donc la combinaison (le produit) des intensités des liens réels depuis la racine. Par exemple, pour le graphe de la figure 6.10, la valeur associée au lien entre *IAPourCouleur2* et *Rouge* sera le produit entre 0,7 et 0,7. Dans la description de l'individu, cela signifie qu'il y a au plus une activation de 0,49 qui peut atteindre le nœud *Rouge* en provenant de *IAPourCouleur2*.

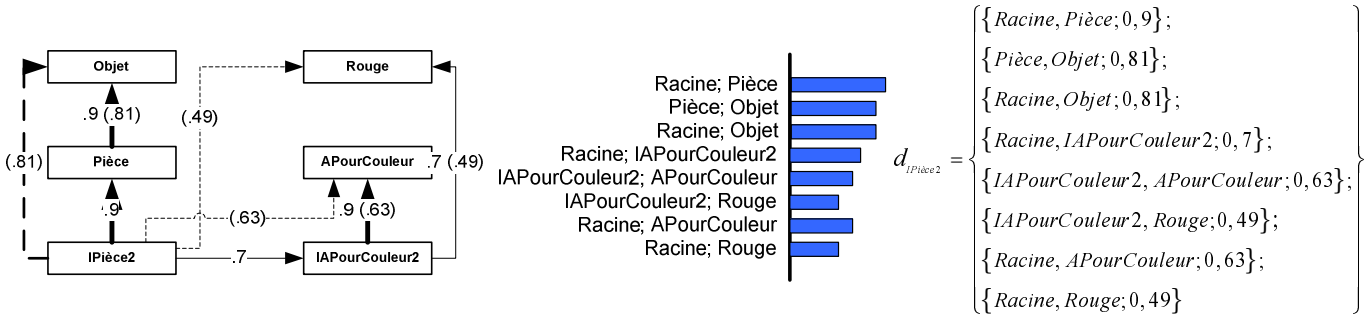



**Figure 6.10 Individu *IPièce2* et sa modélisation après traitement**

#### 6.3.1.2.Ajout de liens virtuels

Les liens ayant une fonction de transmission d'activation, on est obligé de compléter la description avec des liens virtuels entre les connaissances liées par une chaîne de liens. L'intensité du nouveau lien est alors égale à la combinaison des intensités des liens intermédiaires. Par exemple, un lien virtuel est ajouté entre *Racine* et *Objet* et entre *Racine* et *Rouge*.

#### 6.3.1.3.Discussion sur le facteur de combinaison

Pour combiner deux liens et créer les liens virtuels ou calculer l'intensité traitée, nous avons choisi ici d'utiliser le produit des intensités. Ce n'est pas le seul choix possible.

Toujours conformément à la sémantique du modèle, il est par exemple possible de choisir le minimum. Chaque opérateur possède ses avantages propres : le produit correspond plus exactement à la mesure de l'activation maximum qui peut arriver au bout d'une chaîne de deux liens : il prend également en compte la distance réelle avec la racine. Le minimum permet lui de bien conserver l'interprétation d'instanciation des liens forts.

### 6.3.2. Mesure de similarité

Du fait de la modification des valeurs d'intensité en fonction de la connaissance racine, les intensités modifiées des liens entre deux connaissances seront différentes pour des individus ou des objets symboliques différents.

Pour la suite, nous utiliserons la fonction *lien* qui permet d'obtenir l'intensité modifiée du lien en fonction de son origine, sa destination et de la description considérée :

$$\begin{array}{rccl} lien: & C\times C\times D & \rightarrow & [-1;1] \\ & (c_1;c_2;d) & \mapsto & lien(c_1;c_2;d) \end{array}$$

Avec $lien(c_1,c_2,d)$ correspond à l'intensité modifiée du lien de $c_1$ vers $c_2$ pour la description $d$. Il faut noter que $lien(c_1,c_2,d)$ vaut 0 si la description $d$ ne contient aucun lien de $c_1$ vers $c_2$.

Le principe de la similarité est de mesurer la somme des minima d'activation vers chaque connaissance commune depuis la racine. Une fois les prétraitements utilisés, elle peut s'écrire simplement :

$$\begin{array}{rccl} Sim: & D\times D & \rightarrow & \mathbb{R}^+ \\ & (d_1;d_2) & \mapsto & \displaystyle\sum_{c_2\in C} Min\{lien(Racine,c_2,d_1);lien(Racine,c_2,d_2)\} - \\ & & & \displaystyle\sum_{c_2\in C} \underset{c_1\in C\setminus\{Racine\}}{Max} \left(Min\{lien(c_1,c_2,d_1);lien(c_1,c_2,d_2)\}\right) \end{array}$$

Chaque partie peut s'expliquer intuitivement, et on peut représenter le calcul à l'aide d'un tableau récapitulatif des liens (figure 6.11)

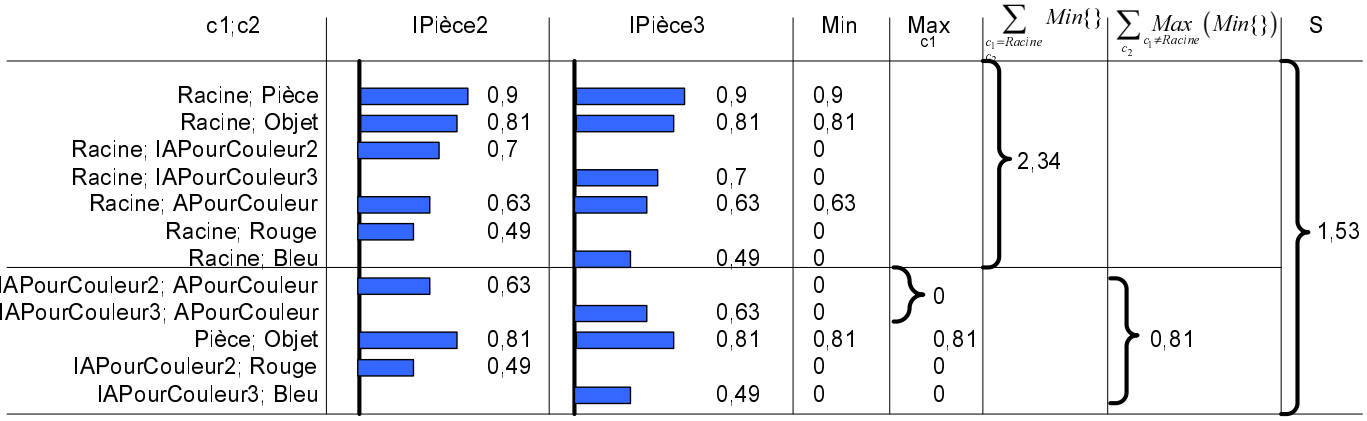



**Figure 6.11 Calcul de la similarité entre les individus IPièce2 et IPièce3**

**Somme des intensités communes des liens depuis la racine**

$$Base = \sum_{c_2 \in C} Min\{lien(Racine, c_2, d_1); lien(Racine, c_2, d_2)\}$$

Du fait de la nature fonctionnelle des liens, l'élément important est la proportion des connaissances communes qui sera activée dans le cas d'une activation de la connaissance centrale (Racine) qui définit l'OS. Autrement dit, nous comparons non pas tous les liens entre les connaissances, mais uniquement les liens virtuels entre la racine et les autres connaissances communes. La base de la similarité entre deux OS sera donc égale à la somme des minimums d'intensité des liens virtuels de la racine vers chaque connaissance commune. Par exemple, figure 6.12, on obtient une similarité de base de 2,34.

**Retrait de l'activation redondante**

$$Redondance = \sum_{c_2 \in C} \underset{c_1 \in C \setminus \{Racine\}}{Max} \left( Min\{lien(c_1, c_2, d_1); lien(c_1, c_2, d_2)\} \right)$$

La mesure de base précédente entraîne l'apparition de redondance au delà des connaissances communes. Par exemple, entre *IPièce2* et *IPièce3*, si une règle ou un attribut est ajouté au concept de *Pièce*, toutes les connaissances correspondant à cet ajout augmenteront mécaniquement la similarité entre les deux individus. Ainsi, dans l'exemple présenté figure 6.12, le fait qu'une *Pièce* soit un *Objet* augmente la similarité de 0,81 (somme des minimums des liens virtuels vers *Objet*). Ce qui nous intéresse est ce qui est commun aux individus (ce sont des *Pièce*) et non les conséquences de ces éléments communs (ce sont des *Objet*). La

suppression de ces éléments redondants permet en particulier de favoriser la découverte de nouveaux concepts caractérisés par des attributs qui n'ont pas encore été regroupés (voir les résultats obtenus section 6.7)

Pour compenser cette redondance, nous calculons une mesure de redondance pour chaque Connaissance (pour chaque c2). Ce facteur est calculé en considérant tous les liens communs aux individus qui aboutissent à c2 et qui ne débutent pas par *Racine*. Pour chaque lien, on prend le minimum des intensités (Min (intensité(i) ;intensité(j)). Le facteur de redondance pour c2 est égal au maximum de ces minima. Ici, il n'y a qu'une connaissance ayant des liens communs ne provenant pas de *Racine* : *Objet*. On calcule le facteur de redondance à partir de l'unique lien : *Pièce/Objet* (Max{{Min{0,81 ;0,81}}}=0,81) et on retire cette valeur de la similarité globale.

Cette mesure est fortement liée au modèle de représentation des connaissances utilisé. Cela limite son applicabilité à d'autres modèles de graphes comme les graphes conceptuels (dans ce cas, il est nécessaire de typer les liens virtuels et de calculer des dissimilarités entre ces types). Mais le but d'une mesure de similarité est avant tout de fournir une analyse de données fiable. Sa dépendance à la sémantique peut alors être vue non comme un problème mais au contraire comme un avantage, car cela signifie que les résultats obtenus sont particulièrement significatifs pour l'application considérée.

## 6.4. Opérateur de fusion

L'opérateur de fusion (opérateur T dans le schéma global de la figure 6.1) est une application permettant d'associer une description à un ensemble de descriptions (il s'agit ici d'une extension de la définition d'origine de l'opérateur de fusion qui associe une description à un couple de descriptions). Cet opérateur permet en particulier d'obtenir les objets symboliques correspondant aux paliers d'une pyramide ou d'une classification, chaque palier étant issu de la fusion des paliers inférieurs. L'extension de l'OS produit de la fusion doit alors contenir (et être le plus proche possible) l'ensemble des extensions des OS de base.

Pour réaliser la fusion, nous prenons le minimum des intensités des liens communs (réels ou virtuels) aux objets fusionnés. Pour fusionner un ensemble de n objets symboliques $G=\{d_1,\dots,d_n\}$ :

$$T: \quad \boldsymbol{P}(D) \rightarrow D$$
$$G \mapsto \left\{ l \in \Lambda \,/\, \begin{array}{l} \forall c_1, c_2 \in C \times C \,/\, \forall k \in [\![1;n]\!], \exists l_k \in d_k / [origine(l_k) = c_1] \wedge [cible(l_k) = c_2] \\ l = \left\{ c_1; c_2; \underset{k}{Min}\left(\text{intensite}(l_k)\right) \right\} \end{array} \right\}$$

Par exemple, la fusion de *IPièce2* et *IPièce3* peut se décrire à l'aide de la figure 6.12

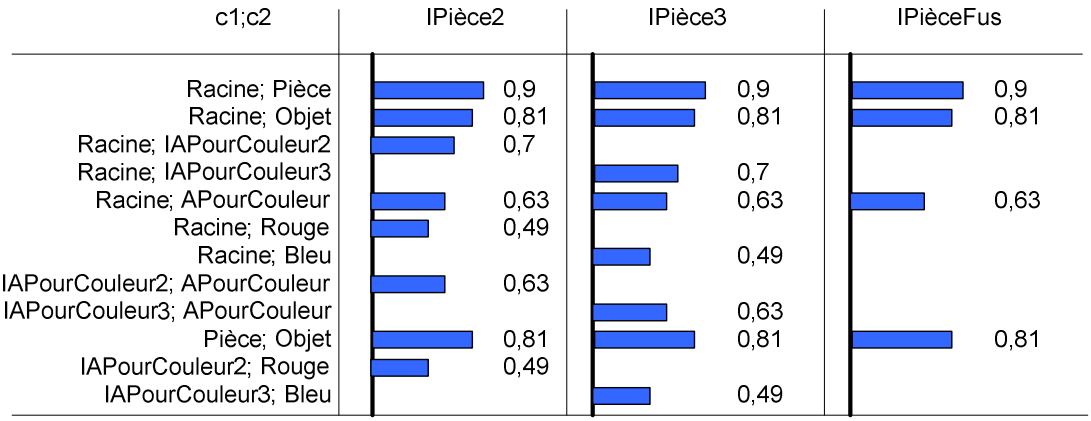



**Figure 6.12 Calcul de la fusion entre les individus IPièce2 et IPièce3**

La description de l'objet issu de la fusion est alors :

$$d_{IPièceFus} = \left\{ \begin{array}{l} \{Racine, Pièce; 0,9\}; \\ \{Racine, Objet; 0,81\}; \\ \{Racine, APourCouleur; 0,63\}; \\ \{Pièce, Objet; 0,81\} \end{array} \right\}$$

On peut en déduire la définition complète de l'objet symbolique :

$$a_{IPièceFus} = [y_l \supseteq \{(Racine, Piece; 0,9)\}] \wedge [y_l \supseteq \{(Racine, Objet; 0,81)\}]$$
$$\wedge [y_l \supseteq \{(Racine, APourCouleur; 0,63)\}] \wedge [y_l \supseteq \{(Pièce, Objet; 0,81)\}]$$

## 6.5. Utilisation des outils d'analyse de données symboliques

L'application des méthodes classiques d'analyse de données symboliques permet d'analyser le graphe de connaissances. Par exemple, l'application d'un algorithme de nuées dynamique permettrait d'isoler deux sous-classes dans un ensemble. En l'appliquant aux exemples d'un concept (comme les exemples du concept de pièces qui ont été perçus par l'agent au cours des différents parties) on peut ainsi obtenir deux sous-ensembles correspondant à des concepts intéressant à analyser (et représentant dans notre exemple les pièces d'une même forme).

Ces algorithmes ne peuvent toutefois s'appliquer que de façon statique (pour analyser l'agent à un moment donné). D'où l'intérêt de développer l'outil de visualisation dynamique des OS issus d'une pyramide présenté au Chapitre 7 .

Pour l'induction, nous utilisons des algorithmes existants pour extraire un concept pertinent dans notre graphe de connaissances Pour réaliser cette sélection, nous nous servons, ici, de deux méthodes différentes en fonction du nombre d'individus à analyser (du fait de l'importance du temps d'exécution dans un agent temps réel).

### 6.5.1. Classification systématique

Lorsque le nombre d'individus est suffisamment faible, il est possible de calculer toutes les combinaisons possibles et de choisir celle qui présente le plus de similarités. La fonction d'évaluation de chaque combinaison correspond alors à la mesure de la similarité globale multipliée par une puissance de la taille du groupe :

$$eval(G) = Sim(G) \times \left(Card(G)\right)^{p}$$

avec comme similarité celle décrite précédemment généralisée à n individus G={s1,… ,sn}

$$\begin{array}{rccl} Sim: & \boldsymbol{P}(S) & \rightarrow & \mathbb{R}^{+} \\ & G & \mapsto & \displaystyle\sum_{c_2 \in C} \underset{s_k \in G}{Min}\left(lien(Racine, c_2, s_k)\right) - \sum_{c_2 \in C} \underset{c_1 \in C \setminus \{Racine\}}{Max}\left(\underset{s_k \in G}{Min}\left(lien(c_1, c_2, s_k)\right)\right) \end{array}$$

Si p=0, le choix se fera toujours vers un groupe de taille 2. Au contraire, si p est élevé, le choix se portera vers un groupe de taille plus importante.

### 6.5.2. Classification par pyramide

Lorsque le nombre d'individus est plus important, il est trop long de calculer toutes les combinaisons. Nous avons donc choisi de réaliser une pyramide [Diday 1984] et de sélectionner un palier de cette pyramide comme nouveau concept. La méthode de création et de sélection est décrite avec précision au Chapitre 7 .

Des exemples de pyramides obtenues sont donnés en section 6.7.

### 6.5.3. Fusion

Une fois le groupe d'individus sélectionné, on utilise l'opérateur de fusion pour obtenir un nouvel Objet Symbolique correspondant au concept appris (voir exemple figure 6.15).

## 6.6. Individualisation

Afin que l'agent puisse utiliser le nouvel objet pour raisonner, il faut l'introduire dans le graphe comme un nouvel individu. Cette opération d' « individualisation », création d'un individu à partir d'un objet symbolique, n'est habituellement pas traitée par les modèles d'analyse de données car ceux-ci ont pour objectif d'extraire des concepts et non de les réinsérer par la suite. Il nous faut donc compléter le modèle global comme présenté figure 6.13 :

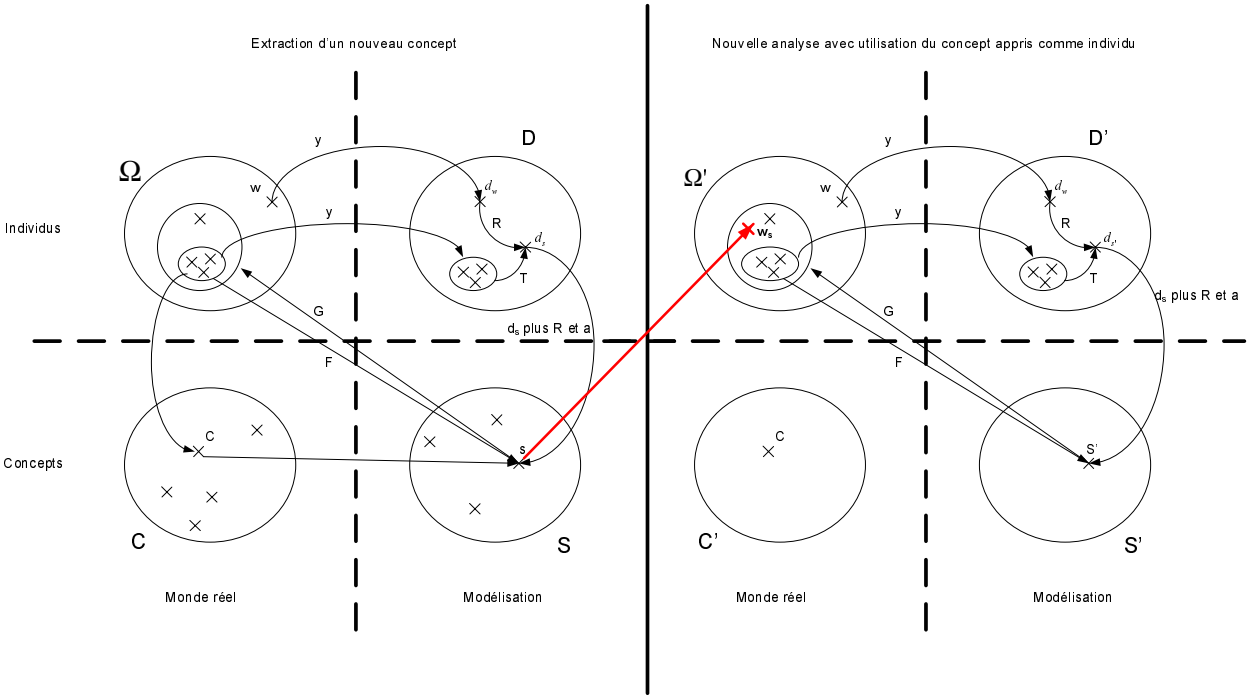



**Figure 6.13 Ajout de l'opérateur d'individualisation dans le cadre de l'Analyse de Données Symboliques**

Le nouveau concept est ajouté aux connaissances de l'agent. Comme il n'y a pas de différence de représentation en fonction de l'abstraction des concepts, le nouveau concept pourra être considéré comme un individu lors des analyses suivantes. Le nouvel ensemble des individus (Ω') peut alors être vu comme l'ensemble des concepts précédents (C). Celui-ci

contient à la fois le nouveau concept (c) et tous les concepts dont l'extension correspond à un seul individu de Ω. Card(Ω')=Card(Ω)+1 (le nouveau concept).

Comme ce nouvel individu appartient à l'extension du concept dont il est issu, si l'on considère le nouvel objet symbolique (s') issu de la fusion des mêmes individus que lors de l'analyse précédente, l'extension de cet objet contiendra l'individu correspondant au nouveau concept ($w_s$). C'est-à-dire qu'on aura : $a_{s'}(w_s)$=vrai.

Pour introduire le nouveau concept, le graphe ne peut être modifié directement à partir de la description de l'objet symbolique. Pour être utilisable pour le raisonnement, il faut reconstruire les nœuds intermédiaires qui ont été perdus au cours de l'analyse car spécifiques aux différents individus.

Par exemple, si on considère deux individus qui correspondent à des Pièces contenant deux Briques ayant chacune une position relative :

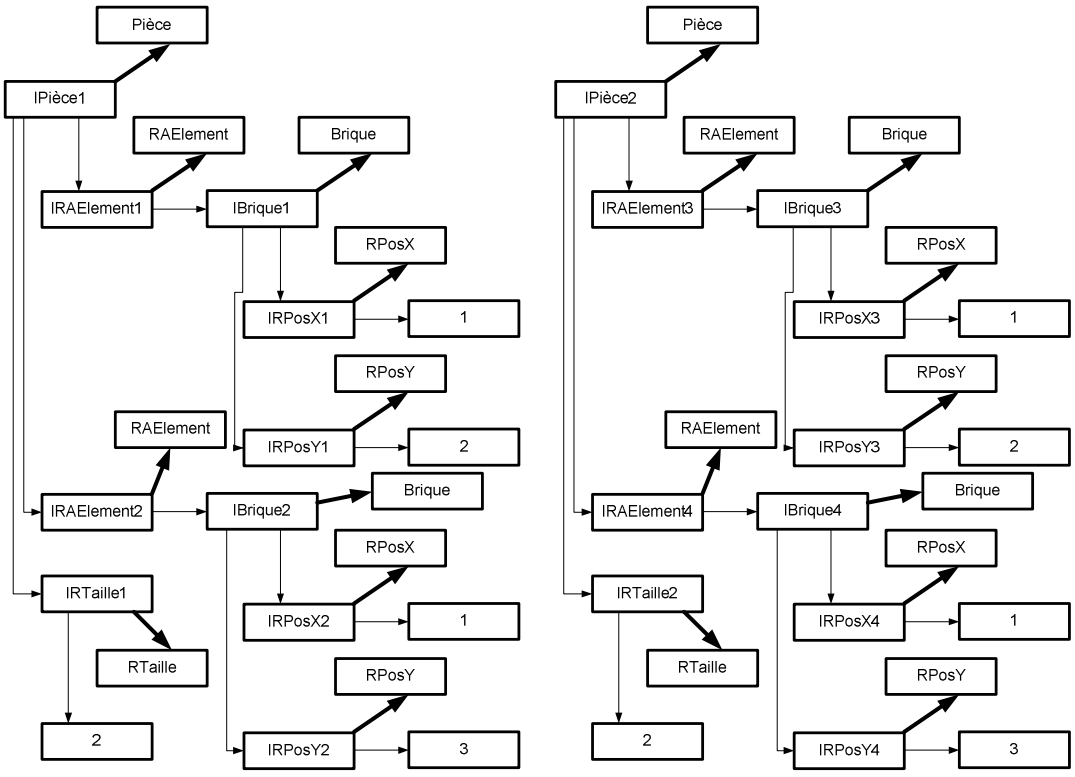



**Figure 6.14 Individus correspondant à deux Pièces qui ont chacune deux Briques caractérisées par leur position relative**

La fusion de ces individus permet d'obtenir la modélisation suivante pour le nouveau concept :

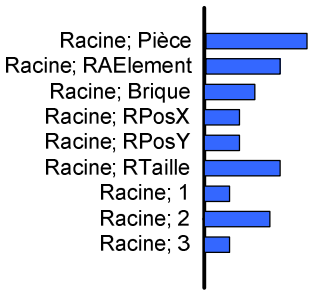



**Figure 6.15 Résultat de la fusion des deux individus IPièce1 et IPièce2**

Les Connaissances correspondant aux positions relatives en X des Briques sont différentes dans les deux pièces, même quand la position est la même (par exemple *IRPosX1* et *IRPosX3*). Dans la construction du nouvel individu, il faut reconstruire une nouvelle connaissance *IRPosX5* qui va reprendre les caractéristiques des deux connaissances intermédiaires remplacées.

Le principe de l'algorithme d'individualisation est le suivant : il va chercher à compléter le graphe progressivement tout en plaçant tous les liens issus de la fusion. Chaque connaissance analysée du concept créé (*TabNouvConn*[]) correspond à une connaissance de chaque individu (*TabCorresp*[][]).

L'algorithme peut se diviser en plusieurs étapes (une version plus détaillée est donnée dans le cadre 1 en fin de section):

1. Initialisation, créer une nouvelle connaissance à analyser $N_0$ et l'associer aux racines des individus
2. Pour la connaissance à analyser en cours $N_{noact}$ associée pour les n individus aux connaissances $I^i_{noact}$
   a. Recherche de liens communs à tous les $I^i_{noact}$ vers une connaissance commune C et ajouter ces liens à $N_{noact}$
   b. Recherche d'attributs communs à tous les $I^i_{noact}$
   c. Pour chaque type T d'attribut commun trouvé :
      i. S'il existe des cibles communes, ajouter l'attribut à $N_{noact}$
      ii. Si les cibles sont différentes, créer une nouvelle connaissance à analyser $N_{t+1}$ et l'associer pour chaque individu à une connaissance $I^i_{t+1}$ en utilisant la similarité entre OS pour déterminer la connaissance la plus adaptée
3. Passer à la connaissance à analyser suivante et revenir en 2. S'il n'y en a pas, terminer.

### 6.6.1. Phase 1 : Initialisation

Lors de l'initialisation, la connaissance de base correspond à la *Racine* de chaque individu et l'algorithme commence avec cette unique base à analyser.

## 6.6.2. Phase 2 : Recherche de liens et d'attributs communs

### 6.6.2.1. Phase 2a : Recherche de liens vers une connaissance commune

Pour la connaissance à analyser en cours, l'algorithme va chercher des liens communs dans les connaissances correspondantes des individus. S'il en trouve, il ajoute un lien identique à la connaissance à analyser et le retire de l'ensemble des liens à trouver.

### 6.6.2.2. Phase 2b : Recherche d'attributs communs

Il va ensuite chercher les attributs communs aux différentes connaissances correspondantes pour recréer des attributs similaires.

S'il trouve des types d'attributs communs, on peut distinguer plusieurs cas type :

Si les attributs pointent vers une connaissance identique (dans notre exemple, l'attribut *Taille* est toujours de valeur 2), il suffit d'ajouter un attribut avec la même valeur à la connaissance analysée.

Si les attributs pointent vers des connaissances différentes, il faut alors créer une nouvelle connaissance à analyser qui correspondra a chacune d'entre elles.

Le problème devient intéressant lorsqu'il y a plusieurs valeurs d'attributs dans chaque individu et qu'elles sont toutes différentes. Dans notre cas, chaque *Pièce* a plusieurs attributs *RAElement* (*IBrique1* et *IBrique2* pour *IPiece1*, *IBrique3* et *IBrique4* pour *IPiece2*), tous différents. Comment savoir que *IBrique1* correspond à *IBrique3* et *IBrique2* à *IBrique4* ? On peut utiliser ici un des avantages de notre représentation homogène et de notre mesure de similarité en générant un objet symbolique à partir de chacune des connaissances intermédiaires possibles et en choisissant les correspondances qui maximisent la similarité. Par exemple, on va calculer les similarités *IBrique1*-*IBrique3* et *IBrique1*-*IBrique4* et déterminer ainsi que *IBrique1* correspond à *IBrique3* et *IBrique2* à *IBrique4*.

La correspondance avec les valeurs des intensités des liens obtenus par fusion se fait lors de la création des attributs. Ainsi, lors de l'ajout de l'attribut *RPosX* à *IBrique5* (la nouvelle brique créée), il y a ajout de la connaissance intermédiaire *IRPosX5*. Le choix de l'intensité de *IBrique5* vers *IRPosX5* se fait de telle sorte que le lien virtuel résultant de la *Racine* vers *RPosX* soit égal ou inférieur à la valeur obtenue par fusion (la valeur est inférieure si le lien par défaut des attributs est insuffisant).

Le graphe obtenu a partir de l'exemple présenté est donc :

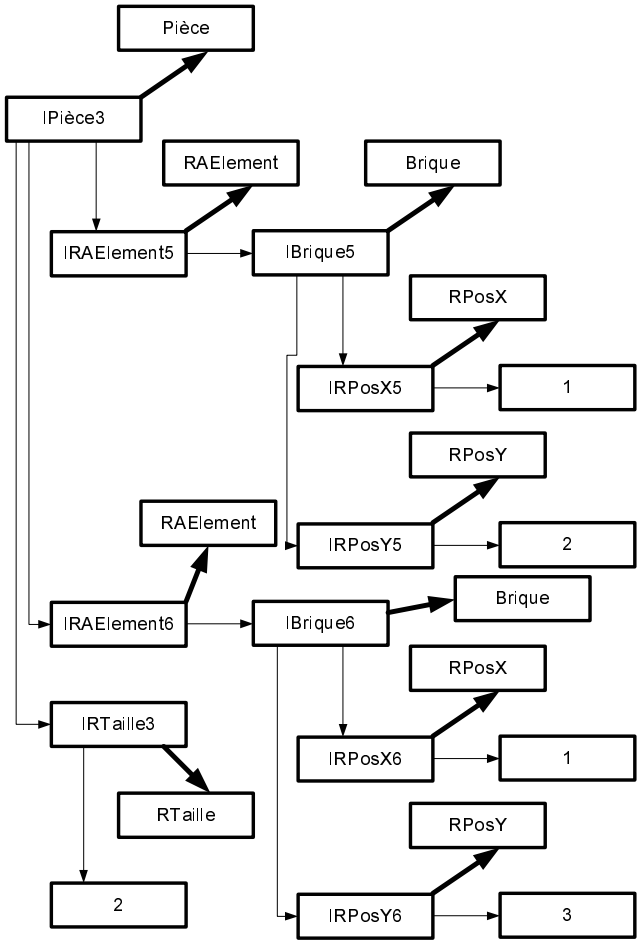



**Figure 6.16 Résultat de l'individualisation**

Ce nouvel individu est directement utilisable par l'agent pour raisonner, et éventuellement pour induire à nouveau en l'utilisant comme individu.

**Cadre 6.1 : algorithme d'individualisation**

**1.**

```
noanal=-1
NombreDeConnIntermediaire=0
TabNouvConn[0]=BaseNouvConcept
TabCorresp[0,i]=BaseIndividu(i)
```

**2.**

```
TantQue noanal<NombreDeConnIntermediaire
Noanal=noanal+1
```

**2.a.**

```
Si Pour tout i=1..NbIndiv il existe un lien commun vers C
  TabNouvConn[noanal].ajLien(C,Min des intensité des liens)
  Si LiensATrouver[] contient un lien vers C,
    Si ce lien aune intensité inférieure,
      retirer ce lien de LiensATrouver
```

**2.b.**

```
  NbTypeAttributsMax=TabCorresp[noanal,1].nbTypeAttributs
  Pour tout a=1.. NbTypeAttributsMax
    TypeAttribut[a]= TabCorresp[noanal,1].TypeAttibut[a]
NbAttributsMax[a]= TabCorresp[noanal,1].NbAttribut[a]
NoMinAtt[a]=1
    Pour tout c=1.. NbAttributsMax[a]
      CorrespAttrib[a,1,c]= TabCorresp[noanal,1].Attrib[a,c]
Pour tout i=2..nbindiv
    Pour tout a=1.. NbTypeAttributsMax
      Si TabCorresp[noanal,i].TypeAttribut.contient(TypeAttibut[a])
        Si TabCorresp[noanal,i].NbAttribut[a]< NbAttributsMax[a]
          NoMinAtt[a]=i
          NbAttributsMax[a]= TabCorresp[noanal,i].NbAttribut[a]
        Pour tout c=1.. TabCorresp[noanal,i].NbAttribut[a]
          CorrespAttrib[a,i,c]= TabCorresp[noanal,i].Attrib[a,c]
      Sinon NbAttributsMax[a]=0
```

**2.c.**

```
Pour tout a=1.. NbTypeAttributsMax Si NbAttributsMax[a]>0
  Pour Tout j=1..NbAttributsMax[a]
```

**2.c.i.**

```
    Si Pour Tout i=1..nbindiv
      CorrespAttirb[a,i] contient CorrespAttrib[a,NoMinAtt[a],j]
    Alors
      TabNouvConn[noanal].ajAttribut(TypeAttribut[a],
        CorrespAttrib[a,NoMinAtt[a],j])
```

**2.c.ii.**

```
    Sinon
      NombreDeConnIntermediaire= NombreDeConnIntermediaire+1
      NouvConn=Nouvelle Connaissance
      TabNouvConn[NombreDeConnIntermediaire]=NouvConn
      TabNouvConn[noelem].ajAttribut(TypeAttribut[a],NouvConn)
      Pour Tout i=1..nbindiv
        TabCorresp[NombreDeConnIntermediaire]=
          CorrespAttrib[a,i,k] avec k qui Maximise
            Similarité avec CorrespAttrib[a,NoMinAtt[a],j]
```

## 6.7. Résultats

Le bon fonctionnement de notre système d'induction par analyse de données symboliques peut se mesurer de deux façons :

- Trouve-t-il des concepts intéressants ? L'objectif étant de fournir un système découvrant des concepts dans le graphe de connaissances, la première étape consiste bien sûr à s'intéresser à ce qu'il découvre.
- S'intègre-t-il bien dans le système ? D'un point de vue technique, les différentes étapes de l'induction sont illustrées par des résultats extraits de l'application pour montrer l'intégration au système global.

### 6.7.1. Concepts extraits

L'avantage de cette méthode d'analyse est qu'elle peut porter sur n'importe quel type de concept dans le graphe. Quel que soit le concept analysé, on peut toutefois constater une première tendance, qui consiste à extraire un concept très général qui pourrait s'interpréter comme « instance du type général ». Par exemple, lors d'une analyse des objectifs, il va extraire le concept d' « objectif exécutable », par opposition à « catégorie d'objectif ». Lorsqu'il analyse le concept de *Pièce*, il va extraire « Pièce observée », par opposition à la catégorie générale de *Pièce*. Cette distinction n'existe pas à l'origine dans le graphe, un sur-type n'étant pas une catégorie mais un ensemble de propriétés communes. Le fait que la distinction n'existe pas n'implique pas qu'elle ne soit pas utile. Au contraire, lorsque l'agent crée le concept de « Pièce observée », cela signifie juste qu'il rassemble toutes les propriétés communes de toutes les pièces qu'il a observées, ce qui permet à toute nouvelle pièce observée de les acquérir instantanément.

On obtient par exemple, en analysant les objectifs au début d'une exécution la pyramide de la figure 6.17.

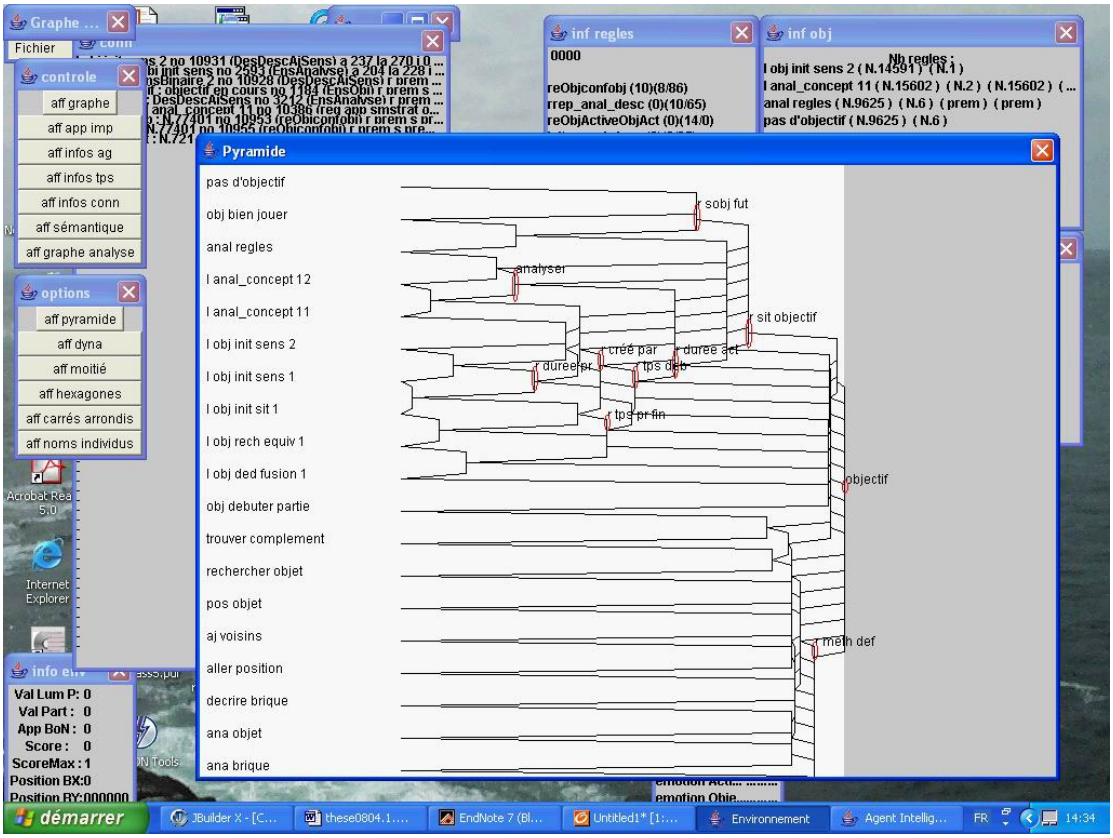



**Figure 6.17 Exemple de pyramide issue de l'analyse du concept Objectif**

Le premier concept extrait est celui correspondant aux dix objectifs supérieurs, et qui correspond bien aux « objectifs exécutables » par opposition aux catégories d'objectifs, situés au dessous. Ceux-ci doivent être instanciés avant d'être exécutés car ils nécessitent des paramètres supplémentaires pour fonctionner. Le deuxième concept extrait par l'agent (par ordre d'évaluation) est celui d' « objectif avec sous-objectif en attente » (les trois premiers en haut – *r_sousobj_fut*), catégorie elle aussi pertinente.

Techniquement, chaque individu de ce type d'analyse comporte entre 100 et 1200 Connaissances (nœuds du sous-graphe), et peut donc disposer de près de 1000x1000 liens virtuels. Le renforcement (voir Chapitre 9 ) est naturellement un grand générateur de liens, mais ceux-ci permettent d'affiner l'analyse en extrayant des concepts issus justement de ces liens renforcés.

Les regroupements réalisés par le système ne se font que rarement en fonction de la catégorie la plus visible qu'est le type d'objectif. Par exemple, dans la pyramide de la figure 6.18, bien qu'il y ait onze objectifs du même type *ana_brique*, on remarque que le palier correspondant à leur regroupement n'est même pas sélectionné dans les dix meilleurs.

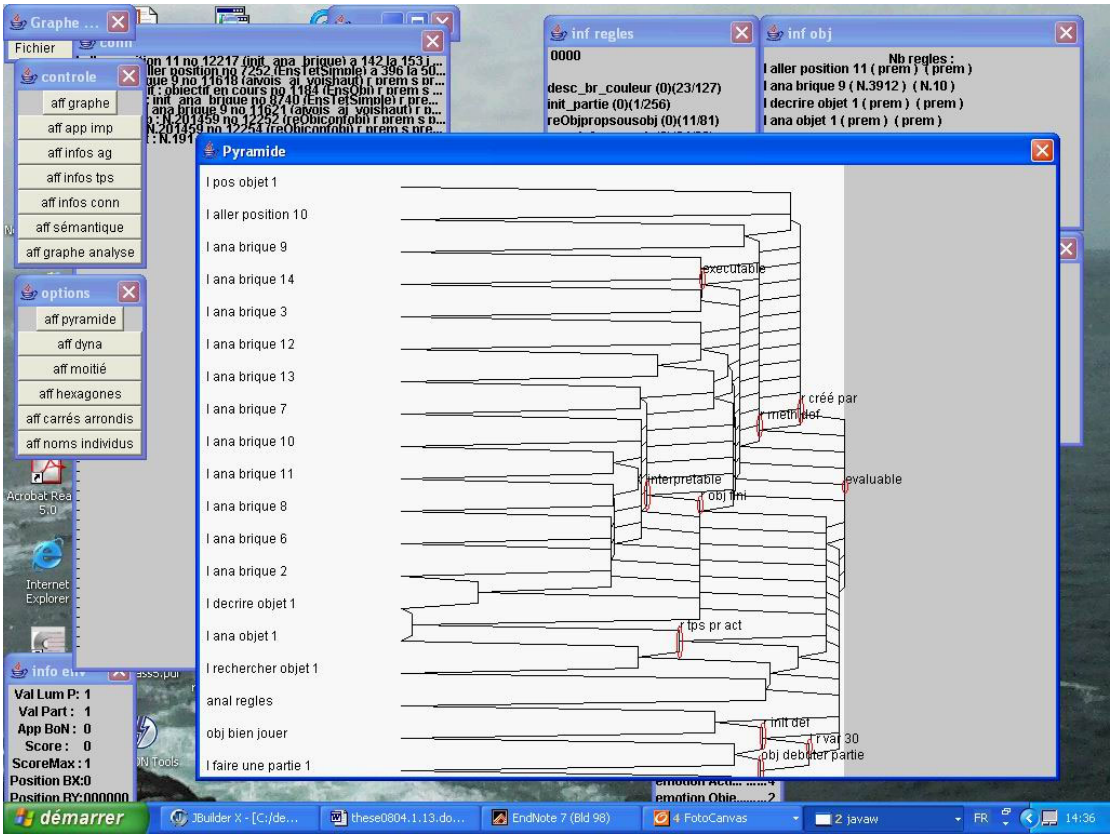



**Figure 6.18 Exemple de pyramide issue de l'analyse du concept Objectif**

L'intérêt de la suppression des informations redondantes de la fonction de similarité apparaît ici : à partir du moment où deux individus sont du même type, leur similarité augmente (de la valeur du lien : 0,9) mais tout ce qui est caractéristique du type (définition, fonctions liées, …) et qui conduirait automatiquement à la sélection du groupe comme meilleure solution, est considéré comme redondant, donc retiré. Des concepts plus intéressants peuvent alors apparaître, tels que ceux sélectionnés ici d'Objectif en cours (*r_tps_pr_act*) et d'Objectif créé par une règle (*r_créé_par*).

Un deuxième avantage de la suppression de la redondance provient du fait qu'une fois qu'un concept a été sélectionné et créé (comme *PièceObservée*), il peut être lui-même analysé. Même s'il n'est pas analysé directement, il va influencer les résultats en supprimant toutes les valeurs communes aux *PièceObservée* de l'analyse suivante.

Ainsi, l'analyse induisant le concept de *PièceObservée* (*nPiece_p*) à partir de *Pièce*, puis celui de *Barre* (*nnPiece_p_p*) à partir de celui de *PièceObservée* donnent les résultats de fusion de la figure 6.19. Pour un OS donné, ces fenêtres indiquent son extension (colonne de gauche) et les liens composant sa description et ayant pour origine la racine (colonne de droite, avec la cible et l'intensité du lien).

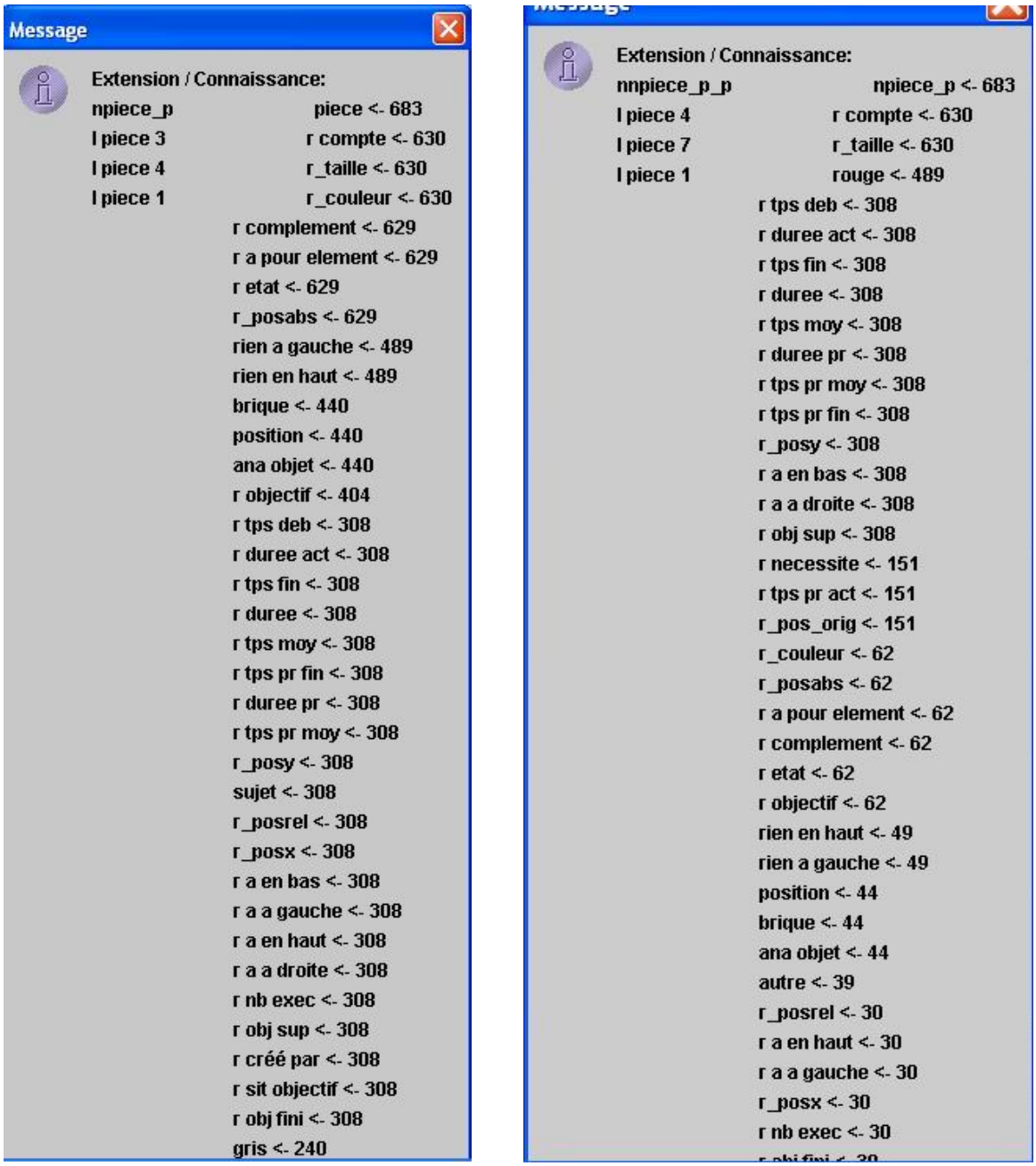



**Figure 6.19 Résultats issus de la fusion du concept de PièceObservée (nPièce_p) issu de Pièce, et de celui de Barre (nnPièce_p_p) issu de PièceObservée**

On voit dans le résultat de la fusion que ce qui caractérise une *Barre* (*nnPièce_p_p*), concept extrait, ne prend pas en compte tous les points communs aux Pièces en général qui apparaissent dans la fusion de *PièceObservée* (le fait qu'il y ait un complément *r_complément*, une position absolue *r_pos_abs*, …). Le fait que ce soit une *Pièce* n'apparaît même pas dans la fusion. Ce qui caractérise une *Barre*, c'est bien principalement le fait que ce soit une *PièceObservée* (*nPièce_p*) et notamment qu'elle soit *Rouge*.

L'analyse pouvant porter sur tout les types de concepts, il est également possible d'analyser directement les règles explicites :

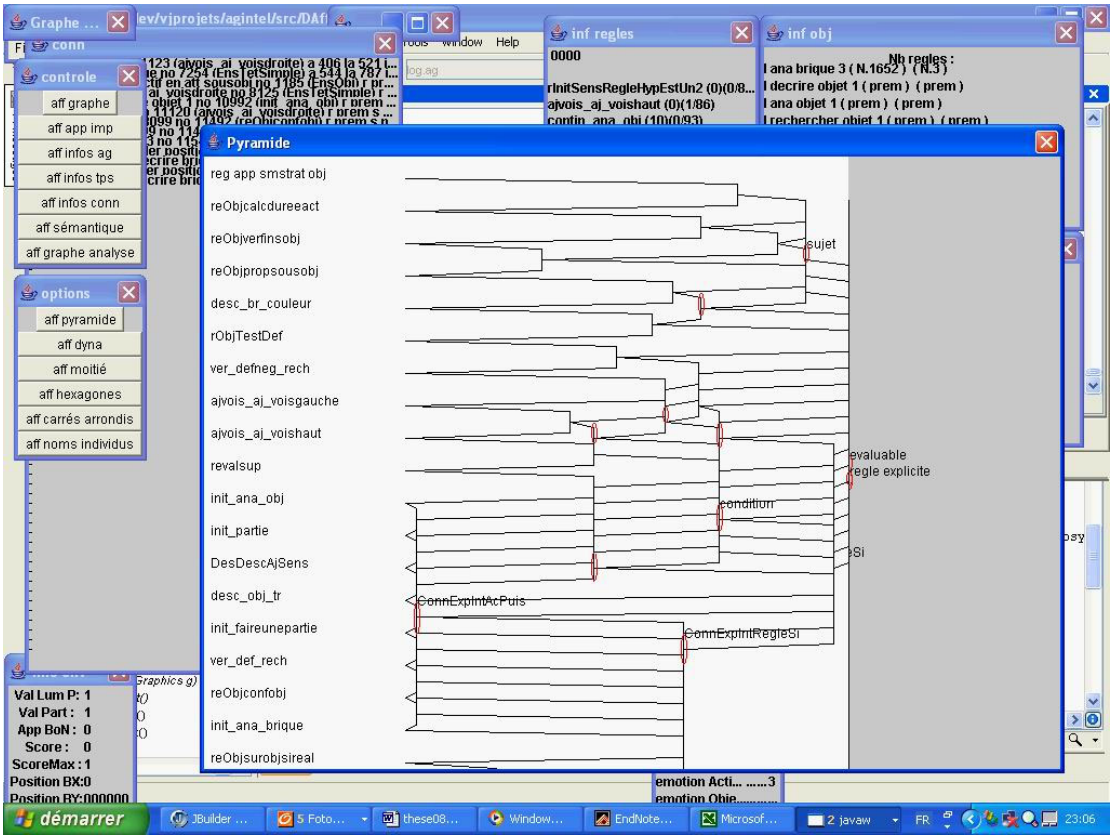



**Figure 6.20 Exemple de pyramide issue de l'analyse du concept Regle**

Dans ce cas, l'analyse porte aussi bien sur la structure (type de sous-composants utilisés, comparateurs, tests, …) que sur le domaine (description, action, …). Par exemple le premier concept issu dans la pyramide figure 6.20 est celui comprenant les 8 règles en bas de la pyramide. Ces règles ne portent pas du tout sur le même domaine, mais ont une structure très similaire, ce qui les conduit à avoir une mesure de similarité élevée. Un tel concept serait très utile à l'agent s'il pouvait en déduire des règles d'analyse de règles. Par exemple, constater que les règles ayant une structure lourde (beaucoup de préconditions ou d'actions) sont plus lentes à exécuter ou s'exécutent rarement. Ce type d'analyse ne poserait aucun problème technique puisque le concept est créé : il faudrait juste disposer des règles d'analyse correspondantes (nous verrons au Chapitre 8 pourquoi nous nous limitions dans cette présentation aux règles d'identification).

### 6.7.2. Intégration

Le bon fonctionnement de la fusion et de l'individualisation peut être vu au travers du résultat de l'analyse du concept de *PièceObservée* (*nPièce_p*) conduisant au concept de *Barre* (*nnPièce_p_p*). La pyramide obtenue lors de l'analyse de cinq pièces dont les trois premières sont des barres est la suivante :

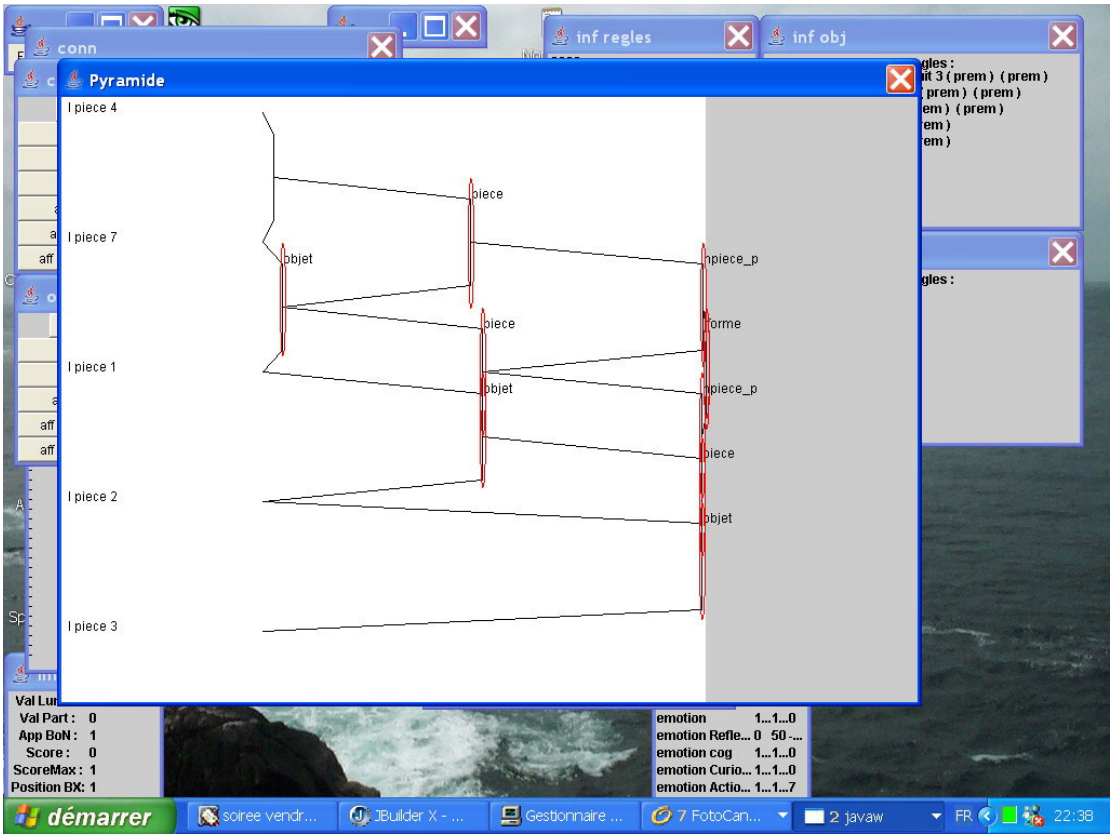



**Figure 6.21 Exemple de pyramide issue de l'analyse du concept nPiece_p (Pièce observée)**

Le programme choisit logiquement de fusionner les trois premiers individus et construit un nouvel individu *nnPièce_p_p* à partir de cette fusion. On constate figure 6.22 en particulier l'attribut *Rouge* qui sera ainsi affecté automatiquement à toutes les instances de *Barre* (voir section 10.1.3.4 pour une explication des éléments de ce type de fenêtre).


**Figure 6.22 Concept Barre (nnpiece_p_p) créé à partir de l'analyse du concept PièceObservée (npièce_p)**

Le fonctionnement de la fusion et de l'individualisation peut être vérifié en comparant le résultat de la fusion des individus et le résultat de la fusion des individus plus le nouveau concept créé (voir figure 6.23). On constate que les valeurs restent les mêmes, ce qui signifie que le nouvel individu a des liens virtuels au moins aussi forts que les liens fusionnés. Le nouveau concept appartient donc à l'extension de l'OS fusionné, donc à sa propre extension. Ceci est une conséquence logique de notre type de représentation. Nos concepts globaux étant des ensembles de propriétés communes et non des catégories, ils appartiennent à leur propre extension.

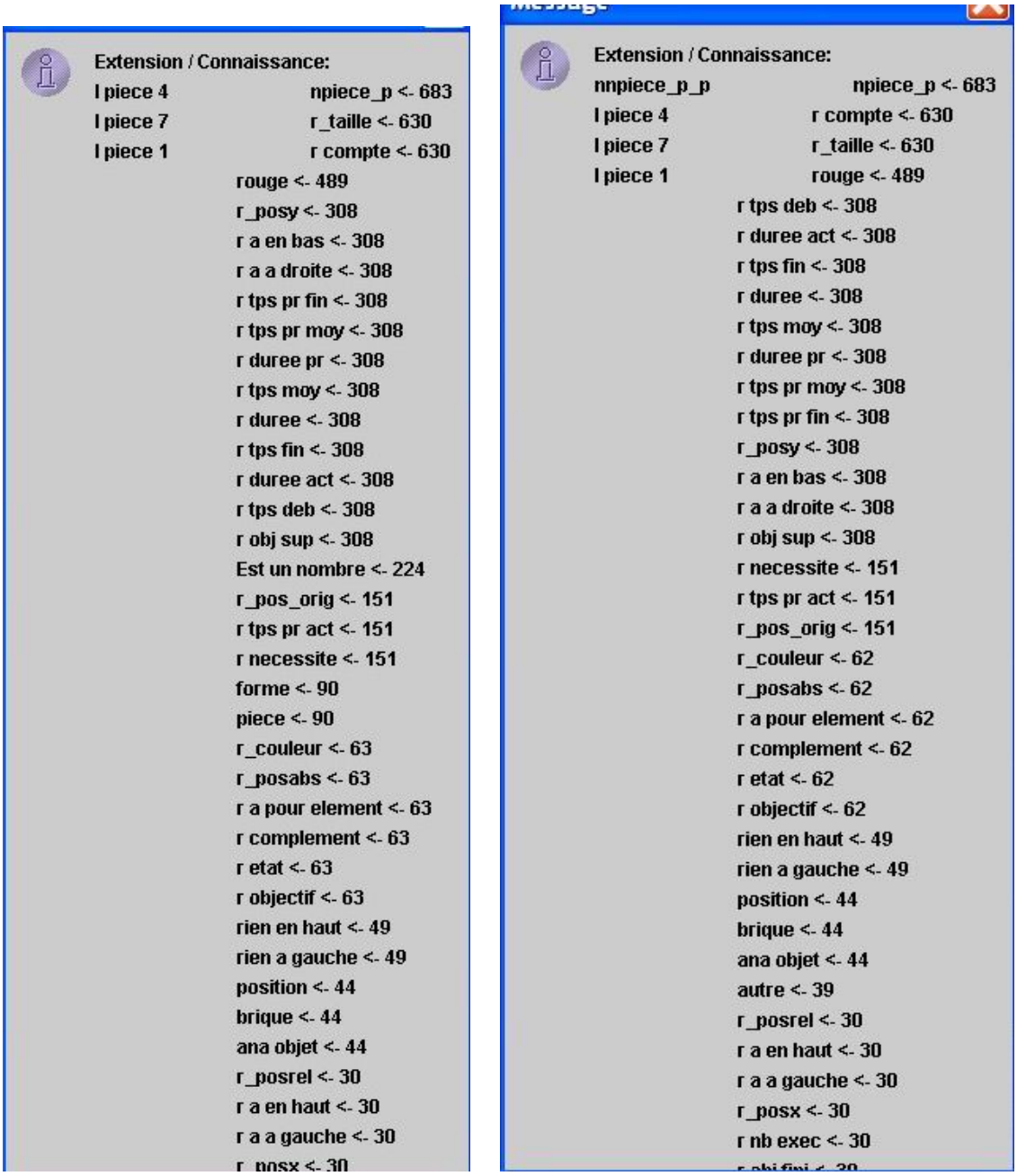



**Figure 6.23 Exemple de résultat de la fusion des objets de l'extension du nouveau concept de Barre (nnPiece_p_p) et de l'extension plus le concept lui même**

## 6.8. Conclusion

Dans ce chapitre, nous avons proposé une méthode d'induction utilisant l'analyse de données symboliques pour à la fois extraire des concepts pertinents du graphe de connaissances de l'agent et les réintégrer dans le graphe. Cette méthode est indépendante de la sémantique de l'agent et fondée sur une mesure de similarité entre sous-graphes permettant de déterminer un groupe d'individus à fusionner pour créer le nouveau concept.

Du fait de la nature progressive de notre analyse (l'agent apprend petit à petit et utilise les concepts déjà appris pour en créer de nouveaux), les nouveaux concepts sont réintroduits dans le graphe par une procédure d'invidualisation et peuvent ainsi être réutilisés pour être analysés au cours des inductions suivantes. La mesure de similarité développée est particulièrement adaptée à cette nature progressive de l'analyse. En effet, la redondance qu'elle mesure, et qui est soustraite à la similarité de base, permet d'extraire les caractéristiques des concepts déjà appris afin de favoriser l'émergence de nouveaux concepts intéressants. Les résultats ont montré que les concepts extraits sont pertinents par rapport au contexte, qu'ils sont réintroduits dans le graphe en correspondant au concept issu de la fusion et qu'ils sont bien utilisés par l'agent au cours des analyse suivantes. Les différentes étapes de l'induction sont résumées sur la figure 6.24.

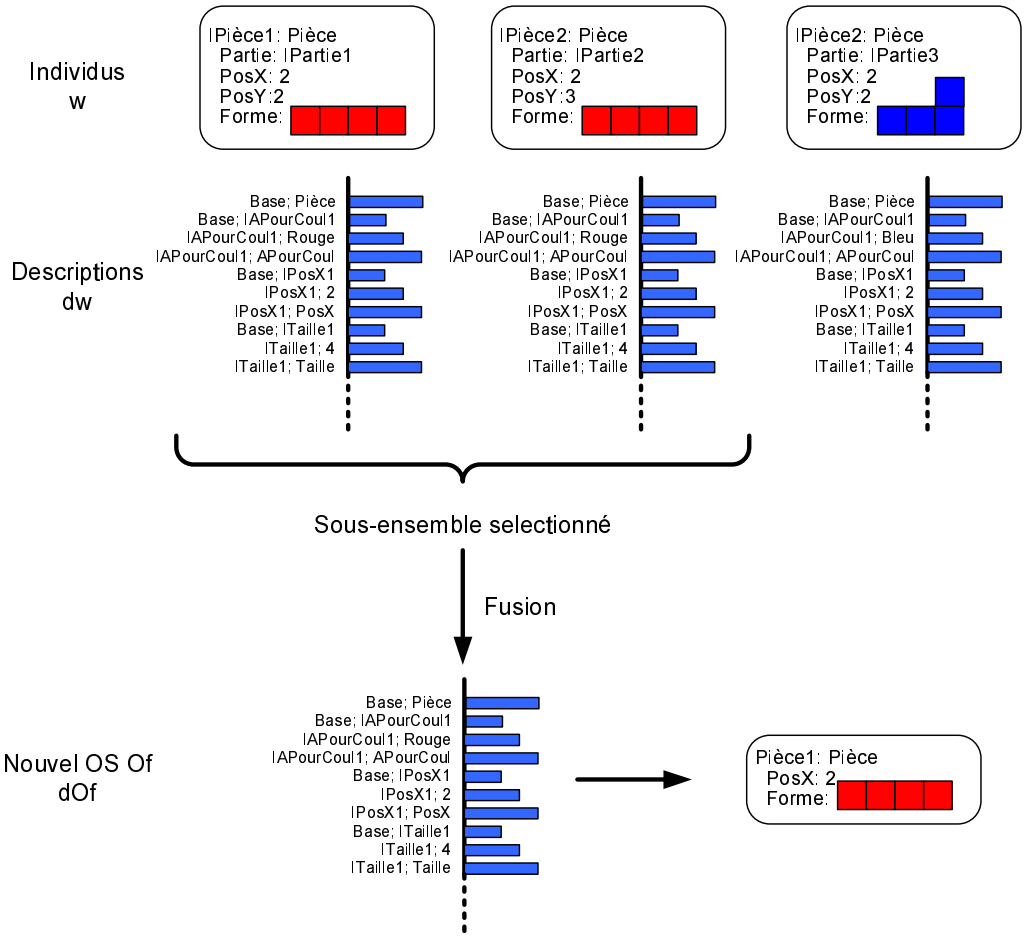



**Figure 6.24 différentes étapes de l'apprentissage inductif**

# Chapitre 7
# Méthode de représentation dynamique d'objets symboliques

## 7.1. Présentation

Comment représenter les concepts utilisés par l'agent ? Comment visualiser l'évolution de ces connaissances ?

Un outil comme la constitution de pyramides en analyse de données symboliques (ADS) permet non seulement d'obtenir une représentation des paliers de distances entre individus, mais également d'obtenir un Objet Symbolique par palier, avec sa définition, et donc la possibilité de l'utiliser en calculant ses propriétés et son extension. Les paliers les plus pertinents de la pyramide peuvent ainsi être interprétés et analysés à leur tour. Toutefois, le grand nombre de paliers rend vite leur analyse ou leur visualisation ardue.

Comment représenter des objets symboliques identifiés à l'aide de la pyramide? Une possibilité serait de réaliser une analyse en composante principale (ACP) symbolique et de représenter les OS par rapport à des axes factoriels. Cette solution a l'inconvénient de nécessiter de représenter un OS, qui a une définition symbolique, sur des axes numériques. C'est assez simple pour des données intervalles, faisable pour des variables multivaluées numériques, mais cela devient très difficile sans perdre beaucoup d'informations avec des descriptions sous forme de graphes telles que celles utilisées ici, ou avec des variables de type textuel ou taxonomique.

C'est pourquoi nous choisissons ici de ne pas utiliser d'axe factoriel ou tout autre représentation en fonction d'une variable. L'objectif de la méthode de représentation proposée est de visualiser les propriétés importantes des objets symboliques ainsi que leurs liens. Individuellement, on cherche en particulier à visualiser la population, la variance et l'évolution des objets symboliques. Il est par ailleurs intéressant de visualiser les objets proches, inclusifs

ou chevauchants. La représentation de l'évolution des objets nous permet de visualiser, en plus de leurs changements propres, le rapprochement de deux objets ou de leur part commune.

Les outils actuels ne donnent pas de représentation de l'évolution des objets, les données utilisées lors des analyses étant généralement statiques (ou éventuellement considérées comme temporelles mais uniquement dans un rôle de prévision). Or l'analyse de données symboliques peut très bien, comme le montre l'application actuelle, être utilisée pour traiter des données dynamiques. Le cas actuel n'a aucune raison d'être isolé. Une analyse dynamique de données pourrait ainsi être appliquée à des données boursières, ou à tout autre ensemble de données dont on veut visualiser les changements.

Après avoir rappelé les notations utilisées en section 7.2, la réalisation du graphique nécessite de calculer la pyramide saturée associée (section 7.3). On choisit ensuite les OS parmi les paliers obtenu (section 7.4). Les OS choisi sont enfin affiché (section 7.5). L'étude de l'évolution dynamique est donnée en section 7.6, quelques exemples d'application sont donnés en section 7.7 et nous concluons en section 7.8.

## 7.2. Notations

Nous reprenons les notations traditionnelles de l'analyse de données symboliques (voir par exemple [Bock 2000]), introduites au Chapitre 6 , et en particulier :

- $\Omega$ l'ensemble des individus

- $|\Omega| = n$

- $O_\Omega$ l'objet symbolique dont l'extension contient l'ensemble des individus

- d(Oi,Oj) une mesure de dissimilarité entre deux objets symboliques
- d(wi,wj) une mesure de dissimilarité entre deux individus
- fus(Oi,Oj) un opérateur de fusion d'objet symboliques
- V(Oi) la variance interne a l'objet Oi, soit la somme des dissimilarités entre les individus de l'extension de Oi :

$$V(O) = \sum_{\substack{w_1 \in ext(O) \\ w_2 \in ext(O)}} d(w_1, w_2)$$

## 7.3. Calcul de la pyramide sous-jacente

La première étape pour réaliser l'analyse consiste à calculer la pyramide à partir des individus étudiés. Cette pyramide a deux objectifs : D'abord, elle fournit un ordre pour l'affichage des individus, et donc des OS, dans le graphe final. Ensuite, elle permet d'obtenir les paliers qui correspondent aux OS affichés.

Pour afficher dynamiquement l'état des connaissances d'un agent fonctionnant en temps réel, l'algorithme de construction de cette pyramide sous-jacente doit être le plus rapide et le moins gourmand en mémoire possible. L'algorithme suivant permet ainsi de construire la pyramide avec une complexité et une utilisation limitée de la mémoire.

Le principe de base de l'algorithme est très simple : en partant de paliers ne contenant qu'un individu chacun, on cherche à chaque étape à fusionner les deux paliers qui sont les plus proches (cette fusion pouvant se faire entre deux paliers issus soit d'un groupe de paliers déjà liés, soit entre deux groupes indépendants). L'algorithme se termine lorsqu'un palier contient tous les individus.

Plus précisément :

1. Initialiser tous les individus en les plaçant dans des groupes séparés.
2. Recherche du palier suivant, soit entre les deux groupes les plus proches, qui sont alors fusionnés, soit entre deux paliers d'un groupe existant. Il y a donc deux types de paliers qui peuvent être liés :
    a. Les paliers qui sont en bas à droite des groupes – si on considère une vision verticale de la pyramide – qui constituent l'ensemble *DispD*. Ils vont chercher à se lier à ceux en bas à gauche des autres groupes (ensemble *DispG*)
    b. Les paliers intérieurs aux groupes. Ceux-ci, s'ils n'ont pas encore de palier supérieur droit (*SupD(Oi)*) et que leur voisin de droite existe (*SupD(InfD(Oi)*), n'ont qu'une possibilité de liaison, ce voisin *VoisD(Oi)=SupD(InfD(Oi))* et ils appartiennent alors à l'ensemble des paliers avec liaison possible mais imposé, l'ensemble *Imp*.

### 7.3.1. Initialisation

On initialise l'ensemble des nœuds terminaux :

```
O[i]={Wi}
```

Chacun de ces nœuds n'a ni objet supérieur, ni objet inférieur, ni voisin imposé :

```
O[i].SupG=vide
O[i].SupD=vide
O[i].InfG=vide
O[i].InfD=vide
O[i].VoisD=vide
```

Chacun de ces nœuds constitue un groupe indépendant :

```
Groupe[i]={O[i]}
```

Chaque nœud constitue à la fois l'extrême droite et l'extrême gauche du groupe, et peut donc être relié à un autre groupe par la droite ou par la gauche :

```
DispD={O[1],…,O[n]}
DispG={O[1],…,O[n]}
```

Aucun nœud n'a un voisin imposé :

```
Imp={}
```

La pyramide n'est alors composée que des nœuds terminaux :

**Figure 7.1 Pyramide initiale**

### 7.3.2. Création d'un nouveau palier

Tant que la pyramide n'est pas achevée (*Imp* non vide ou plusieurs groupes), on répète l'étape.

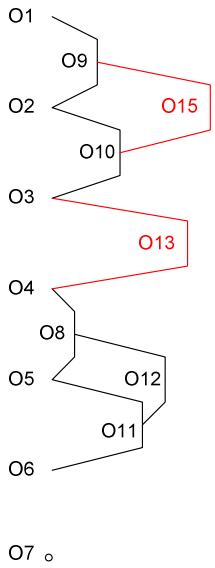



**Figure 7.2 Pyramide après l'ajout de 5 paliers**

Le prochain palier peut venir soit de la fusion de deux groupes (O13), soit de l'intérieur d'un groupe (O15)

On considère les objets extrêmes de chaque groupe et on considère le groupe qui a la dissimilarité associée la plus faible.

```
Chercher OGg∈ DispD et ODg∈ DispG tel que diss(OGg,ODg) soit minimale
```

Puis le couple intérieur qui a la dissimilarité la plus faible :

```
Chercher OGv∈ Imp et ODv=OGv.Vois tel que diss(OGv,ODv) soit minimale
```

On prend le meilleur de ces deux couples :

```
(OG,OD)=(OGg,ODg) si diss(OGg,ODg)< diss(OGv,ODv)
et  (OG,OD)=(OGv,ODv) sinon
```

Un nouveau palier est créé :

```
np=np+1
O[np]=fus(OG,OD)
O[np].InfG=OG
O[np].InfD=OD
OD.SupG= O[np]
OG.SupD= O[np]
```

On vérifie si le futur voisin du nouveau palier existe déjà :

```
Si O[np].InfD.SupD non vide,
Imp=Imp+{O[np]}
O[np].Vois= O[np].InfD.SupD
```

Inversement, on vérifie s'il constitue le nouveau voisin d'un palier existant:

```
Si O[np].InfG.SupG non vide,
Imp=Imp+{O[np].InfG.SupG}
O[np].InfG.SupG.Vois=O[np]
```

Si les paliers fusionnés proviennent de deux groupes différents, on fusionne les deux groupes :

```
Si (OG,OD)= (OGg,ODg)
DispD=DispD-OG
DispG=DispG-OD
Ajouter à Groupe(OG) tous les éléments de Groupe(OD), supprimer Groupe(OD)
```

Si les paliers fusionnés proviennent d'un même groupe, le palier de gauche fusionné n'appartient plus à l'ensemble des paliers à voisin imposé :

```
Si (OG,OD)= (OGv,ODv)
Imp=Imp-OG
```

L'algorithme complet est décrit cadre 7.1

**Cadre 7.1 : algorithme de construction de la pyramide**

```
                              Initialisation
O[i]={Wi}
O[i].SupG=vide
O[i].SupD=vide
O[i].InfG=vide
O[i].InfD=vide
O[i].VoisD=vide
Groupe[i]={O[i]}
DispD={O[1],…,O[n]}
DispG={O[1],…,O[n]}
Imp={}
                                  Boucle
Tant que Imp non vide ou qu'il reste plusieurs groupes :
                    Recherche des paliers à fusionner
Chercher OGg∈ DispD et ODg∈ DispG tel que diss(OGg,ODg) soit minimale
Chercher OGv∈ Imp et ODv=OGv.Vois tel que diss(OGv,ODv) soit minimale

(OG,OD)=(OGg,ODg) si diss(OGg,ODg)< diss(OGv,ODv)
      et (OG,OD)=(OGv,ODv) sinon
                        Création du nouveau palier
np=np+1
O[np]=fus(OG,OD)
O[np].InfG=OG
O[np].InfD=OD
OD.SupG= O[np]
OG.SupD= O[np]

Si O[np].InfD.SupD non vide,
      Imp=Imp+{O[np]}
      O[np].Vois= O[np].InfD.SupD

Si O[np].InfG.SupG non vide,
      Imp=Imp+{O[np].InfG.SupG}
      O[np].InfG.SupG.Vois=O[np]
                              Fusion de groupes
Si (OG,OD)= (OGg,ODg)
      DispD=DispD-OG
      DispG=DispG-OD
      Ajouter à Groupe(OG) tous les éléments de Groupe(OD)
      supprimer Groupe(OD)
        Suppression de la liste des paliers internes à fusionner
Si (OG,OD)= (OGv,ODv)
     Imp=Imp-OG
```

### 7.3.3. Modélisation issue de la pyramide

Une fois la pyramide obtenue on peut définir l'ensemble des individus ordonnés dans l'ordre de la base de la pyramide, l'ensemble des paliers ainsi que la fonction qui calcul la somme des dissimilarités des individus adjacents :

- On ordonne les éléments de $\Omega$ (w1..wn) en utilisant la relation d'ordre issue de la base de la pyramide

- P est l'ensemble des paliers de la pyramide. Pour chaque palier $O \in P$, on défini les fonctions renvoyant le premier et le dernier individu du palier :

$$\begin{aligned} indmin: \quad P &\rightarrow \mathbb{N} \\ O &\mapsto \underset{w_i \in O}{Min}(i) \end{aligned}$$

$$\begin{aligned} indmax: \quad P &\rightarrow \mathbb{N} \\ O &\mapsto \underset{w_i \in O}{Max}(i) \end{aligned}$$

-D(O) la somme des dissimilarités entre les individus *successifs* de l'extension de Oi :

$$\begin{aligned} indmax: \quad P &\rightarrow \mathbb{R} \\ O &\mapsto D(O) = \sum_{i=indmin(O)}^{indmax(O)-1} d(w_i, w_{i+1}) \end{aligned}$$

## 7.4. Sélection des objets symboliques affichés

La pyramide nous permet d'obtenir un ordre pour l'affichage des individus et des OS. Elle peut également nous fournir les OS affichés. Le choix de ces OS détermine naturellement la pertinence de la représentation finale. Notre méthode de représentation présente l'avantage de laisser le choix entre plusieurs méthodes de sélection, en fonction de l'objectif recherché.

### 7.4.1. Recherche du saut maximum

La lecture intuitive d'une pyramide permet de faire apparaître des OS intéressants. Repérer les sauts au niveau de la fonction d'indice (ou à défaut au niveau du nombre de paliers lorsqu'il n'y a pas de fonction d'indice) permet d'identifier des concepts représentant une bonne cohérence interne (valeur de la fonction d'indice proche pour tous les individus du paliers) relativement aux autres OS.

Rechercher les sauts maximaux dans la fonction d'indice constitue donc une méthode automatique de sélection de paliers pertinents:

$$Omax = \arg\max_{O \in P} \left\{ \min \left\{ f\left(SupG(O)\right) - f(O); f\left(SupD(O)\right) - f(O) \right\} \right\}$$

Avec SupG(O) qui désigne le palier supérieur gauche de O, SupD(O) le palier supérieur droit, et f(O) une fonction d'indice.

Cette opération peut être répétée un certain nombre de fois si on souhaite un nombre fixe d'OS affichés ou jusqu'à une certaine valeur seuil du saut si on veut contraindre la pertinence des OS.

### 7.4.2. Méthodes alternatives de sélection

Une autre méthode de sélection consiste à choisir en fonction de la sémantique du modèle. Ainsi, l'utilisateur peut s'intéresser aux OS représentant une caractéristique particulière. Dans notre cas, rechercher une activation moyenne importante ou une forte liaison avec des émotions permet d'obtenir des graphes pertinents par rapport au modèle.

Il existe d'autres méthodes de sélections possibles. Par exemple, une sélection interactive directe de l'utilisateur sur la pyramide, laissant ainsi une grande liberté. Une sélection en fonction des variables taxonomiques permet de visualiser la décomposition progressive des concepts correspondants à ces variables et les propriétés des différentes branches (il peut alors être intéressant d'imposer l'ordre de départ de la pyramide en fonction de la variable taxonomique étudiée afin de favoriser la visualisation).

De plus, les critères peuvent être combinés. En particulier, ils peuvent être utilisés simultanément en affichant les OS correspondants aux critères dans différentes couleurs.

## 7.5. Représentation des objets symboliques

Comme nous l'avons précisé en introduction, l'objectif de cette méthode de représentation est de visualiser les propriétés importantes des objets symboliques ainsi que leurs liens. Individuellement, on cherche en particulier à visualiser la population, la variance et l'évolution des objets symboliques.

Un des principaux objectifs etant de visualiser la proximité d'objets symboliques pour pouvoir identifier des regroupements ou des évolutions intéressantes, la distance entre deux objets symboliques adjacents correspond à la distance entre leurs deux individus les plus proches. L'ordonnée du plan correspond ainsi aux différents individus, dans l'ordre obtenu par la pyramide et séparés par leurs distances respectives :

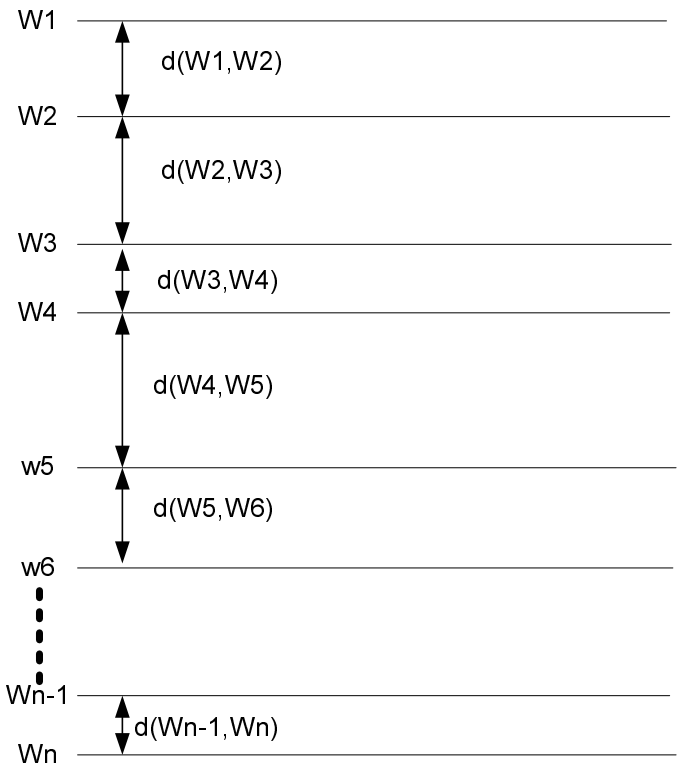



**Figure 7.3 Mesure de l'écart entre individus**

Pour visualiser rapidement l'inclusion et le chevauchement, les objets symboliques sont représentés par des surfaces distinctes, superposées ou chevauchantes en fonction des objets :

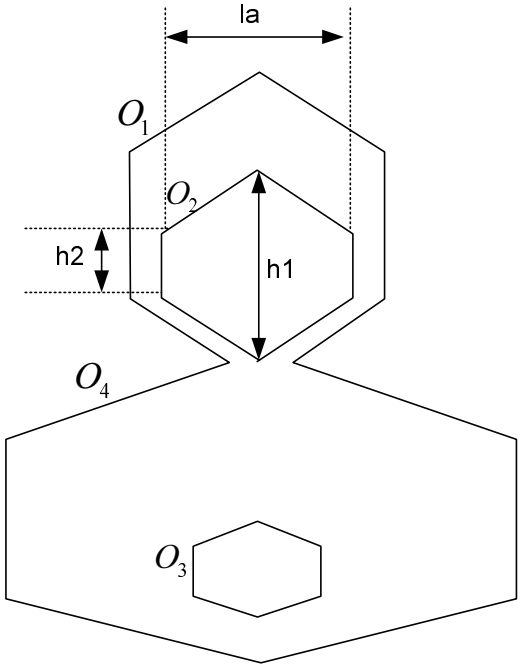



**Figure 7.4 Trapèzes décrivant un objet symbolique**

Plus précisément, pour visualiser simplement la part de la population correspondant à chaque objet, la largeur de l'OS correspond à la cardinalité de son extension.

$$la = |O| = card\left(ext\left(O\right)\right)$$

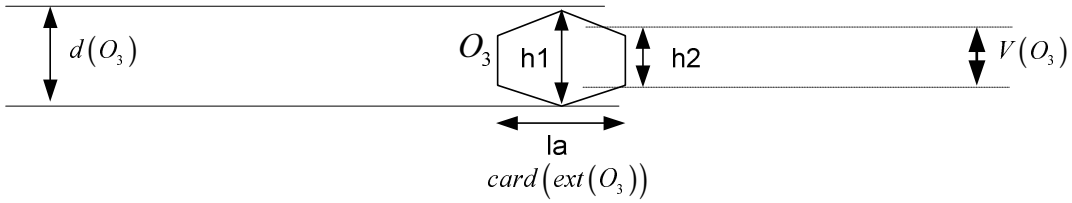



**Figure 7.5 Différentes mesures composant chaque trapèze**

Chaque OS correspond à un hexagone composé de deux trapèzes. La grande base commune des trapèzes, correspondant à sa position sur l'axe des individus (ordonnée), débute donc à l'ordonnée définie par son premier individu et termine à celle définie par son dernier. Sa longueur, donc la hauteur de l'hexagone, est donc de :

$$h1 = D\left(O\right) = \sum_{i=indmin(O)}^{indmax(O)-1} d(w_i, w_{i+1})$$

Cette hauteur, et plus précisément le rapport de la hauteur à la largeur, et donc de la somme des distances à la population, donne une bonne idée de la dispersion interne à l'OS. Une forme plate dans le sens de la largeur correspond à un OS particulièrement dense, c'est à dire contenant des individus proches les uns des autres (relativement au reste de la population). Au contraire, une forme plate dans le sens de la hauteur dénote un écart moyen plus élevé que pour les autres individus.

L'écart visualisé ainsi ne tient toutefois compte que des individus *adjacents* ($\sum_{i=indmin(O)}^{indmax(O)-1} d(w_i, w_{i+1})$). Une mesure de la dispersion plus significative est donnée par la variance, qui considère l'ensemble des distances entre les individus de l'OS ($\sum_{\substack{w_1 \in ext(O) \\ w_2 \in ext(O)}} d\left(w_2, w_2\right)$). Alors que la grande base du trapèze correspond à D(O) (ce choix est obligatoire du fait de la volonté de visualiser rapidement les distances entre les OS), les petites bases des trapèzes correspondent à la variance de l'OS (l'échelle n'est alors plus la même, la hauteur totale du plan correspondant à la variance globale).

$$h2 = V(O) = \sum_{\substack{w_1 \in ext(O) \\ w_2 \in ext(O)}} d(w_1, w_2)$$

Le plan correspond à $O_\Omega$ qui contient tous les autres OS et qui définit les bornes des trois axes :

En abscisse : la population totale $|O_\Omega|$

En ordonnée : Pour la grande base des trapèzes : $D(O_\Omega) = \sum_{i=1}^{n-1} d(w_i, w_{i+1})$

Pour la petite base des trapèzes : $V(O_\Omega) = \sum_{w_i \in \Omega, w_j \in \Omega} d(w_i, w_j)$

$O_\Omega$ est donc représenté par un rectangle qui couvre l'ensemble du plan.

En résumé, chaque OS représenté correspond à un hexagone composé de deux trapèzes symétriques dont les caractéristiques sont :

$$la = |O| = card(ext(O))$$

$$h1 = D(O) = \sum_{i=indmin(O)}^{indmax(O)-1} d(w_i, w_{i+1})$$

$$h2 = V(O) = \sum_{\substack{w_1 \in ext(O) \\ w_2 \in ext(O)}} d(w_1, w_2)$$

Par exemple avec 11 individus et 4 OS composés respectivement de :

Ext(O1)={w1,w2,w3,w4}

Ext(O2)={w2,w3,w4}

Ext(O3)={w5,w6}

Ext($O_\Omega$)={w1,w2,…,w11}

On obtient une représentation de ce type :

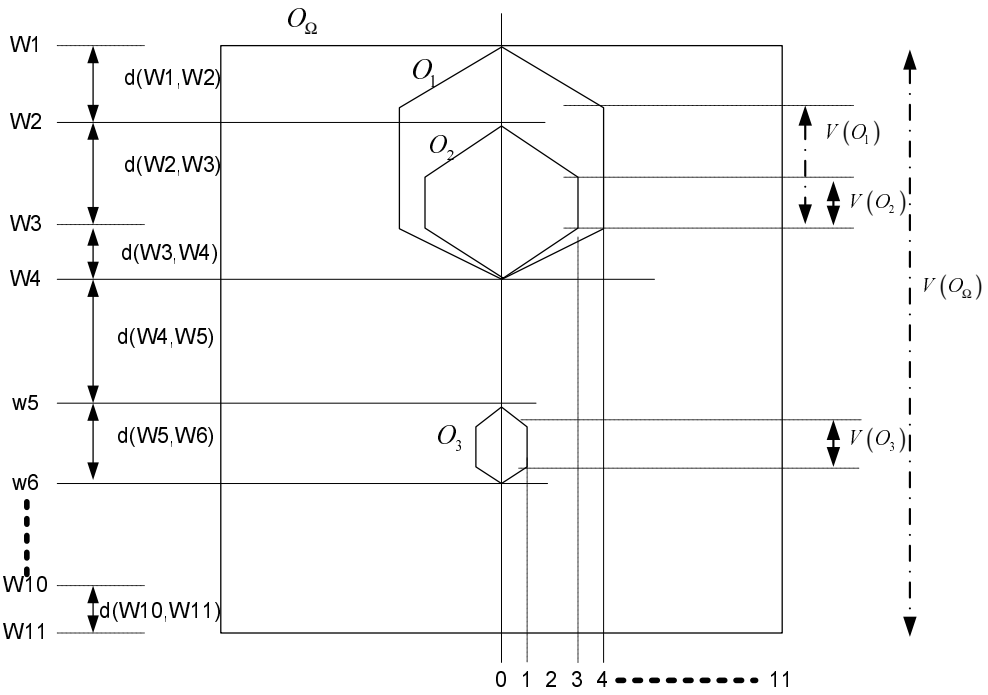



**Figure 7.6 Vision globale de la représentation**

## 7.6. Calcul et affichage des variations

Comment visualiser l'évolution des OS ? L'utilisation de données dynamiques incite à représenter non seulement l'état actuel, mais également l'évolution par rapport aux analyses précédentes. Après avoir trouvé les correspondances entre les nouveaux OS et les anciens, le modèle proposé permet de visualiser ces évolutions.

### 7.6.1. Mise en correspondance des OS

Représenter l'évolution des OS pose un problème d'identification de deux OS à deux périodes différentes. Pour étudier l'évolution d'un OS depuis l'analyse précédente, il faut savoir à quel OS il correspondait. Plusieurs solutions sont possibles pour cette mise en correspondance en fonction de l'objectif de représentation qu'on se fixe.

#### 7.6.1.1.Identification à partir de l'intention

Les OS sont considérés comme identiques si leurs intentions (définitions) sont identiques. Cette solution a l'avantage de conserver une définition fixe, donc de représenter l'évolution d'un concept stable au niveau de ce qu'il représente. Le risque est toutefois important qu'aucun OS de l'analyse précédente ne corresponde exactement à la définition de

l'OS étudié. Le principe même de l'évolution des données va en effet très probablement faire évoluer les définitions des OS regroupant exactement les mêmes individus.

Une solution alternative et proche est de conserver les mêmes OS au fur et à mesure de l'analyse. Les OS sont donc déterminés lors de la première période. Lors des périodes suivantes, le programme se contente de rechercher les individus appartenant à l'extension de ces OS. Cette approche présente l'avantage de conserver l'intention de l'OS et donc de visualiser l'évolution de concepts à la définition stable. De nouveaux problèmes peuvent toutefois apparaître :

- Des OS peuvent se retrouver avec une extension vide. C'est particulièrement probable si les OS sont issus directement des paliers de la première pyramide et que les connaissances sous-jacentes de l'agent ont évoluées. Une solution dans ce cas est de remplacer les OS vides par de nouveaux concepts sélectionnés automatiquement.
- Des OS peuvent être disjoints, c'est à dire constitués d'individus qui ne sont plus adjacents dans la représentation. En effet, les groupes ne sont plus issus d'un palier de la pyramide *courante* et l'ordre peut donc avoir changé. Une solution peut être de représenter tous les groupes (hexagones) appartenant à l'extension d'un même OS d'une même couleur.
- Un OS peut perdre sa pertinence. Les individus sous-jacents étant en constante évolution, un OS pertinent au départ peut ne plus l'être à la période suivante. Or, les données ne sont pas conservées d'une période à l'autre. Pour pouvoir représenter une évolution, la définition se base donc nécessairement sur une intention dont la pertinence est déjà ancienne. Une solution peut être d'évaluer la pertinence des OS à chaque période et de remplacer les moins pertinents par de nouveaux OS sélectionnés automatiquement. Ceux-ci seront pertinents car sélectionnés en fonction de la situation actuelle, mais on ne pourra pas connaître leur évolution avant la période suivante.

#### 7.6.1.2.Identification à partir de la distance

On cherche à assouplir le critère précédent en recherchant l'OS de l'analyse précédente qui est le plus proche (dissimilarité la plus faible). L'avantage est que chaque nouvel OS étudié trouvera un correspondant. De plus, la définition des OS évoluera progressivement et

restera donc pertinente au fur et à mesure des analyses. L'inconvénient est qu'une correspondance sera trouvée même si l'algorithme n'aurait intuitivement pas du en trouver. Par exemple, si un nouvel OS apparaît constitué uniquement de nouveaux individus très différents de ceux présents lors de l'analyse précédente, il serait souhaitable que cet OS ne trouve aucun correspondant.

#### 7.6.1.3.Identification à partir de l'extension

Il est possible de recherche les OS contenant les mêmes individus (moins ceux qui ont disparus et sans tenir compte des individus apparus). Cette solution a l'avantage d'être rapide à calculer (en particulier par rapport à la recherche de la dissimilarité minimum qui nécessite un grand nombre de calculs de dissimilarités).

L'évolution de la cardinalité de l'extension affichée représente alors le nombre de nouveaux individus inclus dans le concept moins le nombre d'individus qui étaient dans les concepts et qui ont disparu de la population.

### 7.6.2. Représentation de l'évolution

Le système de représentation proposé permet de représenter l'évolution des objets symboliques extraits à deux niveaux.

#### 7.6.2.1.Evolution de la cardinalité de l'extension

Pour l'évolution de la cardinalité de l'extension de l'OS, un hexagone plus petit (si la cardinalité a augmenté) ou plus grand (si elle a diminué) est superposé à l'OS représenté.

Pour que cette visualisation soit facilement interprétable, l'échelle de cet « ancien » hexagone est la nouvelle échelle. Ainsi, si la taille de la population a évolué entre les deux analyses, la largeur de l'hexagone correspondant à un OS changera.

Cette méthode peut être généralisée pour visualiser l'évolution sur plusieurs périodes avec un dégradé de couleurs des hexagones correspondants aux différentes périodes.

#### 7.6.2.2.Evolution des distances

Pour visualiser l'évolution des distances entre OS (particulièrement intéressante si on souhaite étudier l'évolution des rapports entre OS, par exemple pour trouver des concepts susceptibles d'être généralisés par fusion), on affiche la variation de distance entre chaque

couple d'individus adjacents par une flèche (de couleur et de sens des pointes différents en fonction du type d'évolution). L'épaisseur de la flèche et la longueur de ses pointes sont proportionnelles à la variation pour permettre une interprétation rapide.

## 7.7. Exemples d'application

Pour appliquer et illustrer notre modèle, nous l'avons utilisé dans le but de fournir une représentation claire des concepts créés et utilisés par notre agent.

### 7.7.1. Modélisation

Dans ce cadre, la modélisation et les choix reprennent ceux décrits au Chapitre 6 :

- Individu : chaque individu correspond à une portion du graphe de connaissances centré autour d'une connaissance racine.
- Similarité : On calcule la somme des minimums des intensités des liens communs entre les OS (voir en section 6.3)
- Dissimilarité : A partir de la similarité, on déduit :

$$d(O_1,O_2)=\frac{100}{1+sim(O_1,O_2)}$$ si sim(O1,O2)>0 et d(O1,O2)=0 sinon

- Indice : l'indice d'un palier est égal à la distance maximum entre les individus de l'extension de l'OS correspondant.

$$f(P)=\max_{O_1\in P,O_2\in P}\{d(O_1,O_2)\}$$

- Sélection des OS : pour sélectionner les OS pertinent parmi les paliers de la pyramide, on choisit la méthode automatique de recherche des sauts d'indice maximum décrite en section 7.4.
- Identification des OS : pour rechercher les OS correspondant dans les analyses précédentes, on utilise la méthode des extensions décrite en 7.6.1.3.
- Nom des OS : pour déduire le nom d'un OS correspondant à un palier, on utilise la principale caractéristique de l'OS (le lien dont l'intensité est la plus élevée) qui n'est pas utilisée par les paliers au dessus de lui. Cela permet d'obtenir comme nom une caractéristique particulière à cet OS et facilite ainsi la lecture et l'interprétation.

### 7.7.2. Résultats

En appliquant le modèle de représentation aux individus correspondants aux objectifs contenus dans le graphe de l'agent, on a par exemple obtenu la représentation de la figure 7.7 (avec la pyramide associée):

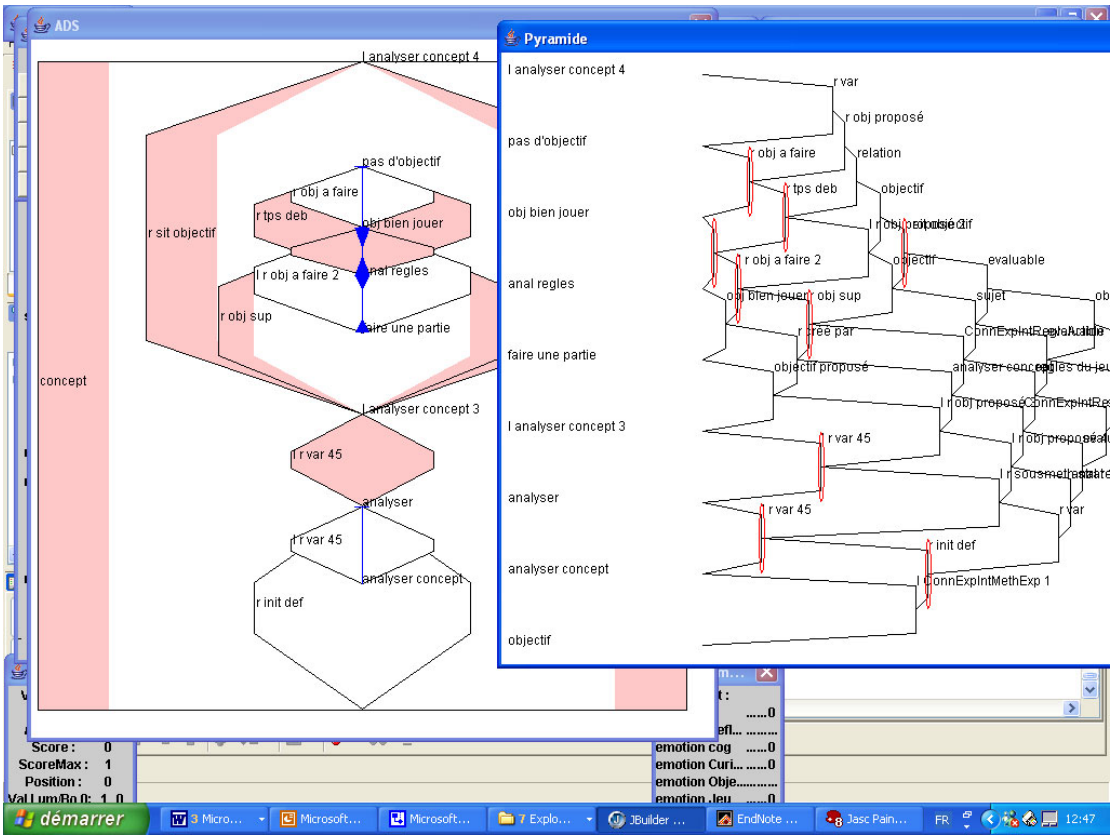



**Figure 7.7 Exemple de représentation appliqué au concept d'Objectif**

De nombreux OS issus de paliers de la pyramide correspondent à des choix pertinents par rapport à l'application (les paliers entourés en rouge sur la pyramide correspondent aux paliers sélectionnés pour être affichés):

On constate tout d'abord que trois individus, regroupés sous l'OS *r_init_def* sont particulièrement éloignés des autres. Ceci est d'autant plus vrai pour l'individu le plus bas, *objectif*. Pour comprendre cette classification, il faut savoir que dans la modélisation utilisée ici, les types d'objets sont représentés dans le même graphe que les instances. Les types d'objectifs sont représentés au même niveau que les objectifs eux-même. Or, *objectif* correspond au type d'objectif le plus général, et *analyser* et *analyser_concept* correspondent aux deux autres types d'objectifs utilisés par l'agent à ce moment là. Le regroupement sous

*r_init_def* est donc pertinent, il correspond au regroupement des types d'objectifs (qui ont entre autre la caractéristique d'avoir une méthode les décrivant dans l'attribut *r_init_def*) en opposition aux objectifs.

Symétriquement, les objectifs sont regroupés dans l'OS *r_sit_objectif*. Les objectifs sont en effet caractérisés par le fait qu'ils ont une propriété qui décrit la situation actuelle de l'objectif dans *r_sit_objectif* (par opposition aux types d'objectifs, qui n'en ont pas besoin).

L'OS *r_tps_deb* correspond lui aussi à un concept particulièrement utile regroupant tous les objectifs débutés (contenant une propriété *r_tps_deb*), donc soit en cours soit terminés.

L'OS *r_obj_sup* correspond au concept de sous-objectif. En effet, les concepts disposant d'une propriété *r_obj_sup* sont appliqués en sous objectif de l'objectif en décrit par ce paramètre. Concrètement, tous les objectifs sont des sous-objectifs à l'exception de *pas_d'objectif* qui est l'objectif racine.

La situation particulière de l'individu *I_analyser_concept_4* s'explique de la façon suivante : on pourrait s'attendre à ce qu'il se situe plus prêt d'*I_analyser_concept_3* ou même d'*analyser* dont il descends. La raison pour laquelle il est si loin du plus proche individu (*pas_d'objectif*) est que l'analyse a été faite alors que l'agent était en train de construire l'objectif *I_analyser_concept_4*. C'est pourquoi il rentre bien dans le concept des objectifs (*r_sit_obj*) mais qu'il est si loin des autres et qu'il ne rentre pas dans le concept des sous-objectifs (*r_obj_sup*).

D'un point de vue dynamique, par rapport à l'analyse précédente, on peut déduire que l'individu *I_analyser_concept_3* a été ajouté, entraînant une augmentation de la population de tous les OS le contenant (*concept*, *r_sit_objectif*, *r_obj_sup,…*). *I_analyser_concept_4* a également été ajouté (il vient d'être créé) et augmente donc la population de *concept* et *r_sit_objectif*.

Au point de vue des distances, on constate que *obj_bien_jouer*, *anal_regle* et *faire_une_partie* se rapprochent, probablement à cause du fait qu'ils ont plus de propriétés communes (notamment le fait qu'ils sont maintenant tous en attente d'être effectué, d'où leur appartenance à l'OS *obj_a_faire*).

A l'analyse suivante, on obtient la représentation de la figure 7.8.

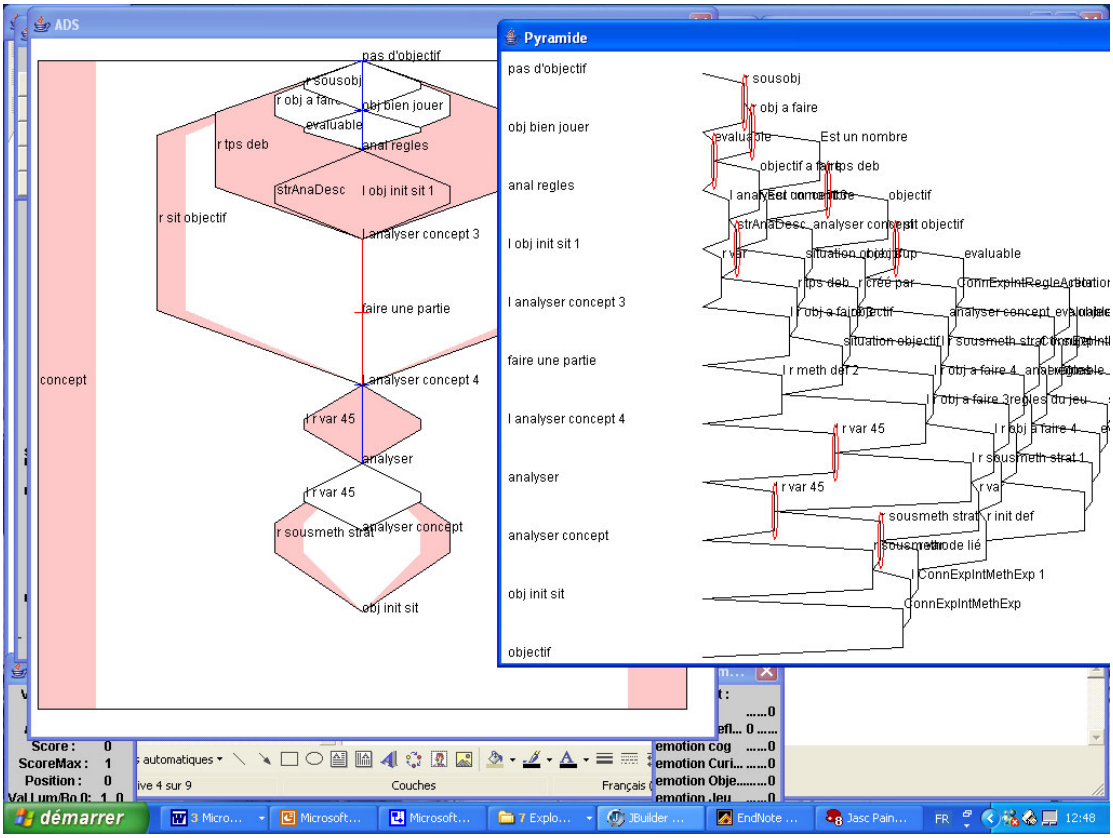



**Figure 7.8 Représentation dynamique du concept d'Objectif**

Un nouveau type d'objectif a été utilisé et est donc analysé : *obj_init_sit*. Il appartient à la catégorie des types d'objectifs qui ont une sous-méthode associée pour les réaliser (*r_sousmeth_strat*), dont la population augmente.

On constate que *I_analyser_concept_4*, qui est désormais fini, est mieux intégré aux autres objectifs (entre *faire_une_partie* et *analyser*). Un nouvel objectif (*I_obj_init_sit_1*) conduit a l'augmentation de la population des concepts d'objectif (*r_sit_objectif)* et à la création d'un nouveau concept d'OS correspondant aux objectifs dont le but est d'analyser une description (*strAnaDesc*).

Enfin, la représentation des OS étant symétrique, il faut noter qu'on peut se contenter d'afficher une moitié de l'hexagone, ce qui permet de conserver plus de place, et donc de clarté, pour les noms des individus. Enfin, pour chaque OS, on peut détailler son intention (qui correspond ici à toutes les connaissances communes et aux intensités minimales des liens correspondants) pour savoir exactement à quoi il correspond et quelle est son extension.

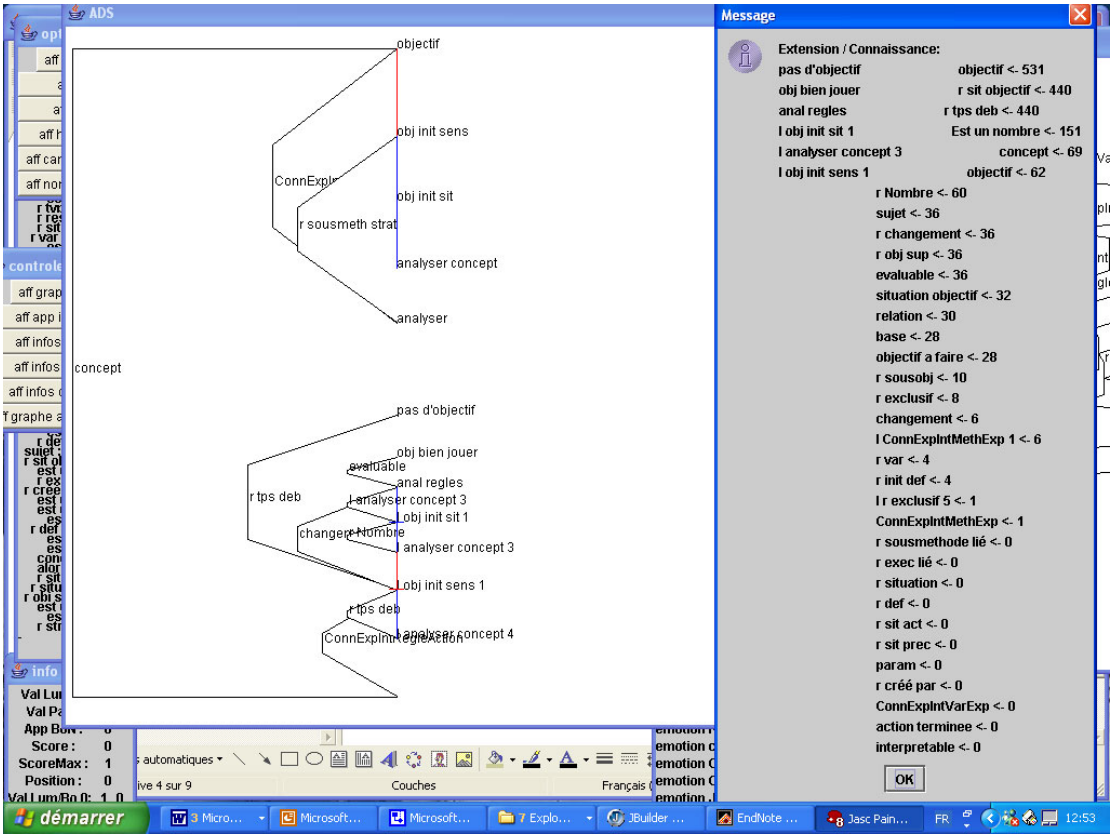



**Figure 7.9 Vision du concept d'Objectif avec détail d'un objet représenté**

## 7.8. Conclusion

L'objectif de cette méthode de représentions d'objets symboliques est de visualiser de façon intuitive dans le plan les caractéristiques et les liens entre les principaux concepts de la population analysée ainsi que leur évolution. La méthode proposée permet d'observer simplement la population, les distances, les variances internes de ces objets ainsi que leurs évolutions. Elle présente l'avantage de laisser de nombreux choix à l'utilisateur (en particulier au niveau du choix de la méthode de sélection des objets, de la méthode d'identification dynamique, de la mesure de similarité).

# Chapitre 8
# Raisonnement : Perception, action et déduction

## 8.1. Présentation

Quelles sont les règles et concepts nécessaires pour obtenir un raisonnement construit et adaptatif dans un agent ? Comment sont-elles créées, insérées dans le graphe et utilisées ?

**Raisonnement construit**

Le raisonnement complexe passe en grande partie par l'application de règles explicites relatives aux différents domaines. L'agent dispose ainsi au départ de règles permettant de définir, suivre et vérifier la fin des objectifs, d'autres règles permettant de créer une perception à partir d'une définition d'un concept, de règles permettant de générer et de vérifier des hypothèses… On constate que ces règles sont spécifiques à certaines fonctionnalités (déduire, vérifier, évaluer, …) et à certains concepts (perception, hypothèse, objectif, …). Elles sont donc dépendantes de la sémantique de l'agent.

**Règles spécifiques**

Pourquoi des règles si spécifiques et donc dépendantes de la sémantique, alors que tous les autres éléments de l'agent (induction, émotion, structure) sont faits pour être indépendants de la sémantique ? A l'opposé de tous ces autres composants, les règles sont d'autant plus efficaces qu'elles sont spécifiques et peuvent évoluer avec la sémantique. Si les règles d'origine sont créées par le concepteur, l'agent peut les faire évoluer ou en créer de nouvelles pour s'adapter aux changements de sa sémantique. L'intérêt de la représentation homogène reste entier, celle-ci est toujours nécessaire pour que l'agent puisse à la fois interpréter, créer et analyser les règles. L'agent va donc disposer de nouvelles règles en plus des anciennes et il doit pouvoir raisonner avec l'ensemble de ces règles de façon efficace.

**Principe d'utilisation des règles**

Le principe de base pour le choix des règles appliquées, et donc de la méthode de raisonnement choisie, est celui de tout système de raisonnement heuristique réflexif : les règles sont exécutées avec une probabilité proportionnelle à un certain facteur, ici leur activation. Savoir quelles règles l'agent utilise pour raisonner dépend donc des concepts qui sont activés à l'instant considéré.

Par exemple, lorsque l'objectif actuel de l'agent est de démarrer une partie, au moins deux types de règles sont particulièrement activées et définissent le raisonnement de l'agent : Tout d'abord celles associées au concept « démarrer une partie », donc l'action d'appuyer sur le bouton correspondant ainsi que la règle vérifiant l'état de la partie et si l'objectif a été atteint. Mais aussi celles associées au concept d' « objectif » en général, c'est-à-dire les règles vérifiant si l'objectif actuel à un sous-objectif à réaliser, celles qui vont replacer l'objectif supérieur en objectif actuel si l'objectif actuel est atteint, …

**Perceptions et actions dans un agent**

Le fonctionnement est donc aussi bien réactif (transmission d'activation à partir des concepts) que proactif (appel explicite de méthodes ou de règles). L'intérêt de ce double fonctionnement et de l'adaptabilité aux changements de sémantique apparaît particulièrement avec les perceptions et actions de l'agent.

**Règles de déductions**

L'objectif global du système est d'obtenir un agent adaptable, c'est-à-dire qui crée et utilise efficacement des règles et concepts adaptés à l'environnement. Pour cela, il doit réaliser un triple apprentissage (induction – déduction - renforcement) sur ses connaissances. Si la découverte de nouveau concept par induction est en grande partie indépendante de la sémantique, les règles de déduction liées à ce nouveau concept seront-elles, comme on l'a vu, dépendante de la sémantique. La déduction des règles d'utilisation se fera différemment si on a créé un nouvel objectif, une nouvelle règle ou une nouvelle relation. Différentes règles devront également exister en fonction de ce qu'il faut obtenir : des hypothèses, des règles d'identification, … Les règles de déduction sont donc dépendantes de ce sur quoi porte l'induction et de ce qui doit être déduit.

L'avantage de cette spécificité est encore une fois l'adéquation à l'objectif. Plus une règle a une application restreinte, plus elle pourra être efficace lorsqu'elle s'appliquera. Et les

règles de déduction étant représentées explicitement dans le graphe, l'agent peut les analyser et en déduire de nouvelles en fonction de l'évolution de sa sémantique.

L'inconvénient est le nombre de règles nécessaires en fonction de la sémantique de départ de l'agent. En d'autres termes, plus l'agent a une sémantique complexe au départ, donc plus il sait faire de choses, plus il aura besoin de règles pour être capable d'adapter toutes ses fonctions.

**Objectif**

Notre objectif dans ce chapitre sera principalement de montrer comment fonctionne un tel raisonnement déductif. Non pas de présenter un raisonnement déductif général adapté à tous les cas, mais de présenter un raisonnement déductif spécifiquement adapté à un domaine utile à un agent : la perception de nouveaux concepts, que ce soit dans l'environnement ou dans son propre raisonnement. Le modèle déductif proposé ici est semblable à un modèle tel que celui d'EURISKO quant à la méthode (application successive d'un grand nombre d'heuristique), mais appliqué ici dans ce domaine (domaine perceptif d'un agent).

Réaliser des raisonnements dans des domaines différentes (planification, résolution de problèmes, …) ou déduire sur des types différents (règles, relations, …) ne change pas le modèle global proposé ici. Cela nécessite toutefois d'ajouter les règles de traitement et de déduction correspondantes à l'agent.

**Plan**

Pour bien comprendre les concepts et fonctionnalités du raisonnement de l'agent, nous allons tout d'abord présenter la sémantique de base utilisée ici.

Puis à la fois pour illustrer la déduction et pour expliquer comment notre système de représentation homogène et le système de raisonnement peuvent s'intégrer dans un agent, nous présentons le fonctionnement des perceptions et actions de l'agent.

Enfin nous décrivons le système déductif qui permet à la fois de guider l'induction et de traiter les concepts induits pour construire les concepts associés.

## 8.2. Sémantique de base

L'agent utilise un grand nombre de concepts dès le départ qu'il va ensuite enrichir par apprentissage. Les concepts présentés ici sont les plus généraux et permettent de comprendre la sémantique des règles présentées par la suite.

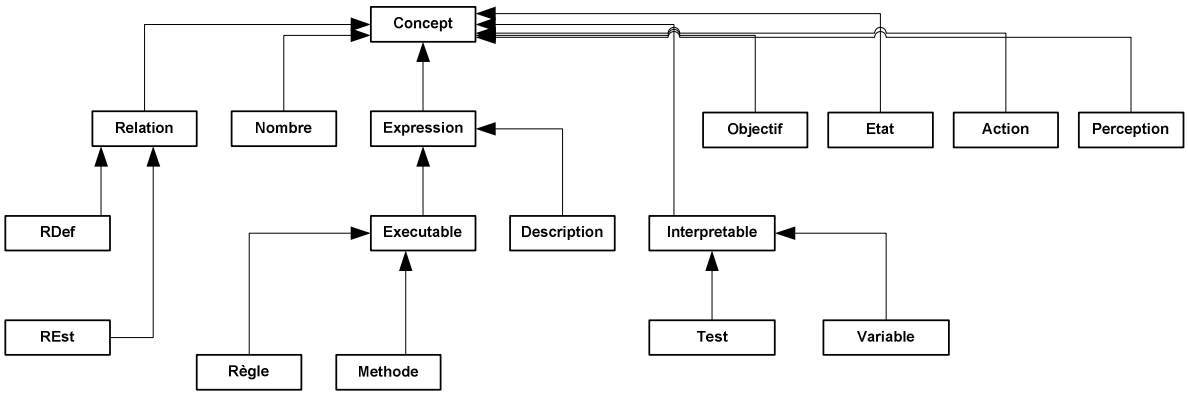



**Figure 8.1 Extrait de la sémantique initiale de l'agent**

### 8.2.1. Concept

Le *Concept* est la notion la plus générale qui n'a aucun sens en soi. Son principal intérêt est de pouvoir faire des règles générales portant sur tout *Concept* et englobant ainsi tout autre notion sous-type de *Concept*.

### 8.2.2. Relation

Les relations servent à définir les types des attributs. Par convention, nous mettrons un « R » au début du nom des Relations pour ne pas les confondre avec les autres concepts. Par exemple, *RObjectif* est le concept définissant un attribut pointant vers un objectif, alors que *Objectif* désigne un objectif. Par soucis de lisibilité, cette convention n'a pas toujours été utilisée dans les représentations données ici sous forme de Frames.

Certains types de relations sont particulièrement utiles :

- *REst* permet à un Concept d'en désigner un autre. Par exemple, l'objectif actuel est défini par *ObjectifActuel->REst*. Encore par soucis de lisibilité, cet intermédiaire a généralement été omis dans les notations données dans ce rapport. Ainsi, la situation de l'objectif actuel, qui correspond à *ObjectifActuel->REst->RSit* a été notée *ObjectifActuel->Sit*
- *RDef* désigne la définition d'un concept
- *RSujet* est la relation la plus utilisée pour désigner les variables d'une méthode

### 8.2.3. Objectif

Un *Objectif* est caractérisé par le fait qu'il peut être atteint ou non. Il dispose d'une règle (dans *Objectif->RDef*) qui permet de vérifier s'il est actuellement atteint. Son état actuel (à faire, en cours, atteint, …) est décrit dans *Objectif->RSit*.

Par exemple, l'objectif *ObjAllerPosition* est atteint lorsque la position actuelle est égale à celle du sujet de l'objectif.

### 8.2.4. Action

Une action est caractérisée par ce qu'elle fait (par opposition à l'objectif qui est définie par ce qu'il cherche à obtenir). Plus de détails sur les actions sont donnés en section 8.3.2.

### 8.2.5. Perception

Une perception est caractérisée par ce qu'elle perçoit. Une description détaillée des perceptions est donnée en section 8.3.1.

### 8.2.6. Expression

Les Expressions désignent les concepts décrivant des règles de façon explicite. La définition explicite de la règle est donnée dans (*Expression->RDef*). On distingue les expressions exécutables des descriptions qui sont de simples descriptions du monde.

#### 8.2.6.1.Exécutable

Les expressions exécutables disposent d'une fonction interprétable lors de l'interprétation dynamique des règles dans *RDef*. Cette fonction peut être exécutée indépendamment si c'est une Règle, et elle doit être appelée par une autre règle ou méthode si c'est une Méthode.

##### 8.2.6.1.1. Règle

Lors de la phase d'exécution des règles de l'agent, chaque concept Règle a une probabilité de voir sa définition interprétée par le système. Cette probabilité est fonction de l'activation du concept.

##### 8.2.6.1.2. Méthode

Une méthode doit être appelée par une autre règle ou méthode pour être interprétée. Elle peut éventuellement utiliser des variables lors de son exécution. Par exemple, *AjEtat(RSujet,RParam,RQuoi)* est une méthode décrite explicitement qui réalise deux opérations : elle ajoute l'état donné dans *RQuoi* à l'attribut du type donné par *RParam* du concept donné par *RSujet*. Par exemple AjEtat*(RSujet=ObjectifActuel, RParam=Sit, RQuoi=EnCours)* définit l'attribut *Sit* de *ObjectifActuel* à *EnCours*. Ensuite elle vérifie qu'il n'y a pas de relation de type *RExclusif* entre deux valeurs de l'attribut. Par exemple, un objectif ne peut être à la fois *EnCours* et *Terminé*. Dans ce cas, la méthode supprime les valeurs précédentes incompatibles avec l'état ajouté.

#### 8.2.6.2.Description

Une Description, contrairement à un *Executable*, n'est jamais interprétée. Il s'agit d'une simple description d'un fait ou d'une règle du monde. On distingue en particulier les descriptions de type implication logique (de type *Implique*), qui permettent de construire de nouvelles perceptions, des implications temporelles (de type *Entraine*) qui permettent de déduire de nouvelles actions.

### 8.2.7. Interprétable

Un concept interprétable peut être utilisé par l'interpréteur dynamique de règle pour exécuter une règle ou un morceau de règle. Les interprétables de départ de l'agent constituent les briques de base pour construire les règles. Par exemple :

- *AcSiAlors* : Ce concept détermine une action de type si..alors. Elle prend deux paramètres : une condition (dans *RCond*) et une action (dans *RAlors*).
- Test : l'agent dispose d'un grand nombre de sous-type de tests au départ, par exemple pour vérifier qu'un attribut contient une valeur, qu'une connaissance est plus activée qu'une autre, …
- Variables : plusieurs sous-types d'interprétables correspondent à des fonctions de lecture de variable. En particulier : *VarFixe*, qui désigne de façon fixe une connaissance ; *VarLitParam*, qui lit un attribut d'un concept (si cet attribut a plusieurs valeurs, il choisit aléatoirement proportionnellement à l'activation) ; *VarUn* qui renvoie un concept du type demandé en choisissant aléatoirement en fonction de l'activation.

- *AcActive* : une action peut constituer dans l'activation d'une certaine connaissance. Cela permet de favoriser explicitement certains concepts dans le graphe.

### 8.2.8. Nombre

Pour simplifier la gestion des nombres, un concept Nombre existe. Chaque instance de Nombre correspond à un nombre et peut ainsi être facilement comparé ou utilisé par les fonctions numériques intégrées à l'agent (somme, soustraction, …).

## 8.3. Perceptions et actions

Quelles sont les règles utilisées par l'agent pour percevoir et agir ? Les règles décrivant la gestion des perceptions sont décrits en 8.3.1 et les actions en 8.3.2.

### 8.3.1. Perceptions

L'agent doit être capable d'agir de façon aussi bien réactive que proactive. Il doit pouvoir percevoir et réagir à l'état de l'environnement, mais aussi à des concepts plus abstraits qu'il aura créé. Pour permettre cette intégration de concepts abstraits dans le système perceptif, une perception se composent de différents éléments décrits en section 8.3.1.1 et dont le fonctionnement est donné en 8.3.1.2. La création de nouvelles perceptions est présentée en 8.4.

#### 8.3.1.1.Composants d'une perception

On peut catégoriser les perceptions en fonction de leurs caractéristiques :

- Perception directe/perception abstraite. Une perception directe étudie directement l'environnement, sans concept intermédiaire (par exemple l'état d'une lumière ou d'un bouton). Au contraire une perception abstraite utilise des critères ne portant pas directement sur l'environnement (l'état de la partie fonction de l'état d'une lumière, ou le type de pièce fonction de la couleur de la pièce), et peut également n'avoir aucun lien avec l'environnement.
- Perception binaire/perception valeur : une perception-binaire modifie une valeur si son test est vérifié, et ne la modifie pas sinon (par exemple l'état allumée d'une lumière ou l'état en-cours d'une partie). Une perception-valeur peut

affecter plusieurs valeurs différentes à l'objet modifié (par exemple la perception d'une couleur).

- Perception-param/perception-EstUn : Une perception-EstUn détermine si un concept (l'objet) est un sous-type d'un autre concept (la valeur). Par exemple, *SePièce* va vérifier si un concept est une pièce. Une perception-Param va déterminer la valeur d'un attribut (le *param*) d'un concept (l'objet).

Une perception est toujours composée de 3 éléments principaux. Nous présentons tout d'abord le cas le plus simple de la perception directe binaire avant de donner les modifications dans les autres cas :

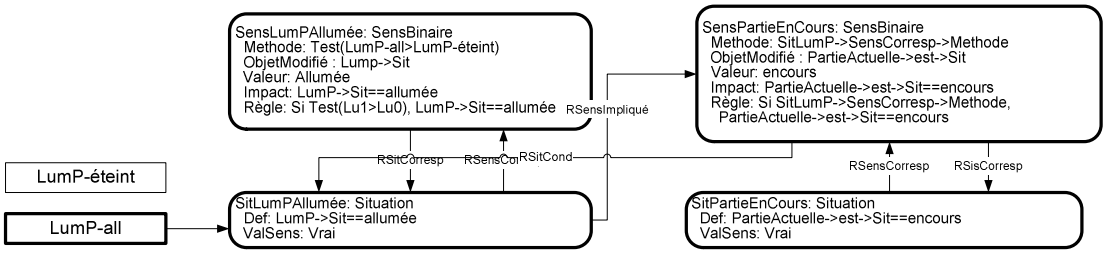



**Figure 8.2 : Exemple de deux perceptions binaires successives, état de la lumière *LumP* et état de la partie**

- Les connaissances perceptives sont les points d'entrée de l'activation dans le sous-graphe correspondant à la perception. Par exemple, pour la perception de l'état de la *LumièreP*, deux connaissances perceptives sont activées en fonction de cet état : *LumP-éteint* et *LumP-all*. L'activation de ces connaissances varie directement en fonction de l'environnement. L'intérêt principal des connaissances perceptives est de permettre la réactivité de l'agent (voir section suivante).

- Un concept Sens explicite regroupant les différentes informations relatives au sens. En particulier : Une méthode renvoyant vrai si la perception est vérifiée, l'objet modifié, la valeur affectée, l'impact de la perception (correspondant a objet=valeur), une règle réalisant le test et affectant la valeur si celui-ci est positif (correspondant a Si *Methode=Vrai* alors *Impact*)

- Une Situation correspondant au fait que l'événement observé est perçu. Par exemple, la situation *SitLumPAllumée* correspond au fait que la *LumièreP* est *allumée*. Le fait correspondant est décrit dans la définition de la Situation. La

présence d'une situation n'est pas obligatoire. Son intérêt, en particulier pour le fonctionnement réactif et la déduction, est décrit plus bas.

La différence fondamentale pour une perception abstraite se situe au niveau des connaissances perceptives. Pour une perception directe, les connaissances perceptives sur lesquelles se base la perception sont influencées directement par l'environnement. Dans une perception abstraite, les connaissances perceptives correspondent à des situations issues d'autres perceptions.

Ainsi, la perception de l'état de la partie est une perception abstraite qui ne porte pas directement sur l'état de l'environnement mais sur le fait que la lumière soit allumée. Dans ce cas la connaissance perceptive sera la situation SitLumPAllumée. C'est l'activation de cette situation qui entraînera l'activation su Sens et l'exécution réactive de la règle associée

Pour les perceptions-valeurs, la méthode du sens explicite ne renvoie plus un résultat vrai/faux mais une valeur qui correspond à la valeur perçue. Dans ce cas, il peut y avoir plusieurs Situation correspondant à un même sens. Par exemple, pour le sens correspondant à la position en abscisse (*PosX*), il peut y avoir autant de Situation que de positions possibles. Concrètement, aucune de ces situations n'existe tant qu'elle n'a pas de raison d'exister. Dans notre cas, cela signifie qu'une situation correspondant à la situation *PosX*=2 ne sera créée que lorsque l'agent analysera l'impact du fait observé « toutes les pièces sont en position *PosX*=2 ».

Les situations, qui peuvent sembler à priori superflues pour le fonctionnement perceptif, sont utiles à de nombreux niveaux : elles servent de connaissances perceptives pour les perceptions abstraites, c'est-à-dire que ce sont elles qui vont faire réagir l'agent à la présence de certaines situations. De plus elles sont utiles lors des raisonnements déductifs de l'agent. Par exemple, pour faire le lien entre la règle « Si *LumP->sit=allumée*, alors *PartieActuelle->Sit=EnCours* » et « *Si PartieActuelle->Sit==EnCours alors ProposerObjectif(TrouverPiece)* ». Dans le premier cas, il s'agit d'une affectation (affectation de la valeur *EnCours* à l'attribut *Sit* de *PartieActuelle*, en réalité représenté par une méthode *DefParam(PartieActuelle, Sit, Encours)* ) alors que dans le deuxième il s'agit d'une lecture suivi d'un test (l'attribut *Sit* de *PartieActuelle* contient-il *EnCours* ?, en réalité représenté par une méthode *TestExisteValParam(PartieActuelle, Sit, Encours)*). Le lien entre ces deux règles se fait au travers de la Situation définie par le fait que *PartieActuelle->Est==EnCours*.

L'analyse de la première règle va conduire à créer une perception associée à la situation alors que l'analyse de la seconde va créer une action fondée sur la même situation.

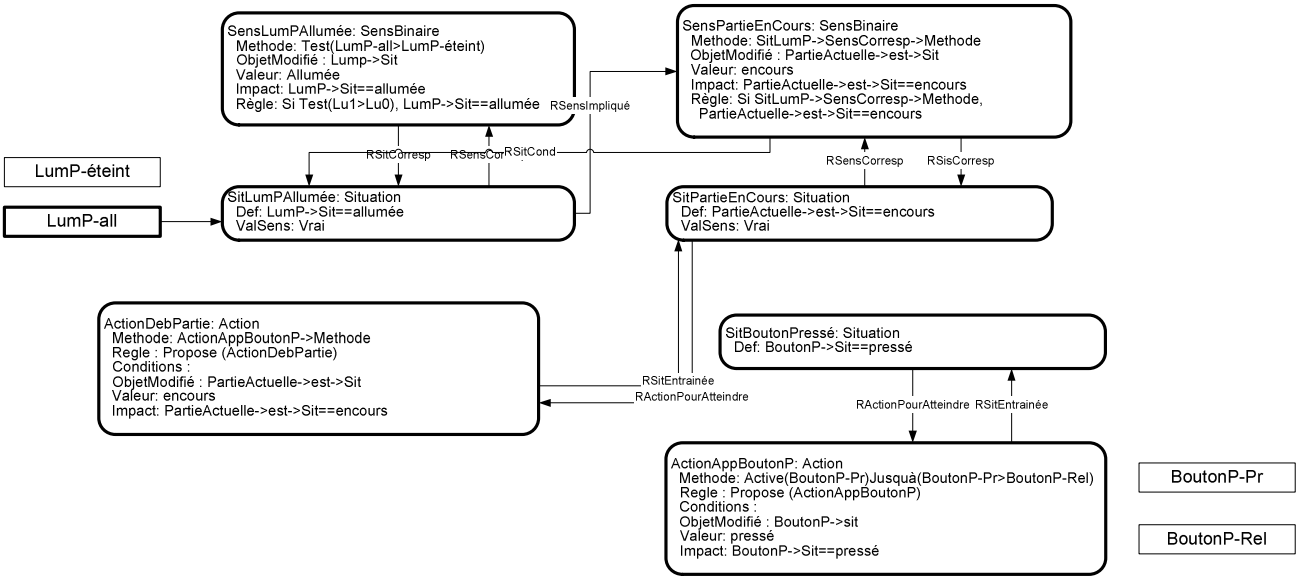



**Figure 8.3 Exemple de liaison perception-action au travers de la Situation *SitPartieActuelleEnCours***

#### 8.3.1.2.Fonctionnement

Le fonctionnement des perceptions est aussi bien proactif que réactif :

D'un point de vue proactif, une règle peut utiliser la méthode associée au sens pour connaître sa valeur. Par exemple le suivi de l'objectif *DébuterPartie* peut appeler explicitement la méthode du sens correspondant pour savoir si l'objectif est atteint ou non.

D'un point de vue réactif, les connaissances perceptives vont activer la ou les situations et le sens explicite, ce qui va entraîner l'exécution probable de la règle associée. Par exemple, l'activation de *LumP-all* va activer la situation correspondant à l'état *Allumée* de la lumière (*SitLumPAllumée*) qui va activer le sens de perception de l'état de la partie (*SensPartieEnCours*) et donc la règle associée, qui va probablement être exécutée et ainsi mettre à jour l'état actuel.

Sans même réaliser une perception explicite, l'agent peut réagir à un changement dans le graphe ou dans l'environnement. L'activation de la situation, indépendamment de la règle associée au sens explicite, peut en effet activer une autre règle qui sera ainsi exécutée.

### 8.3.2. Actions

La composition et le fonctionnement des actions sont similaires à ceux des perceptions :

#### 8.3.2.1.Composants

On peut classifier les actions entre actions directes et actions abstraites : les actions directes agissent directement sur l'environnement (comme presser le *BoutonP* ou se positionner en abscisse), alors que les actions abstraites agissent soit indirectement sur l'environnement (comme Débuter une partie, au travers la pression sur le *BoutonP*) soit uniquement sur les autres concepts.

Une action a trois composants principaux :

- L'action explicite contient toutes les informations explicites concernant l'action :
  - Les conditions d'application décrivent quand il est possible d'appliquer l'action (l'agent peut décider de l'appliquer même si la condition n'est pas remplie, mais il est alors moins sûr de son fonctionnement). Par exemple, la partie n'est pas commencée pour l'action « Débuter la partie »
  - L'impact indique un éventuel changement dans l'état du monde provoqué par l'action. Par exemple « *PartieActuelle->Sit=Encours* », conséquence de l'action.
  - La méthode permet d'effectuer l'action (il s'agit d'une méthode, elle doit donc être invoquée par une règle pour être exécutée). Par exemple « *AcAppBoP->Methode* » qui correspond à l'invocation de la méthode de l'action « *Appuyer sur le BoutonP* ».
  - La règle (qui peut être exécutée sans être invoquée) propose l'action si les conditions d'application sont remplies. Par exemple *« Si la partie n'est pas commencée, proposer l'action DébuterPartie* ».
- Les connaissances actives : équivalents des connaissances perceptives, ce sont elles qui propagent l'activation pour agir sur l'environnement (pour une action directe) ou pour entraîner une action implicite au travers de l'activation d'autres concepts (comme une autre action). Pour une action abstraite comme

« *DébuterPartie* », il s'agit d'une situation correspondant à la description « *PartieActuelle->Sit=EnCours* ».

- Les situations initiales permettent de décrire tout ou partie des conditions d'application sous forme de situations. Elles permettent à la fois à l'agent de faire des déductions à partir des définitions de ces situations, mais aussi de transmettre de l'activation afin que l'action elle-même soit activée quand ces situations sont activées.

#### 8.3.2.2.Fonctionnement

Tout comme les perceptions, les actions peuvent fonctionner de façon proactive ou réactive.

De façon proactive, si un agent a un objectif qui peut être atteint à travers une action, il peut simplement invoquer la méthode de l'action. N'importe quelle règle peut invoquer l'action, que la condition d'application soit remplie ou non, et ainsi l'exécuter, même si elle n'était pas du tout activée auparavant.

De façon réactive, si une action est activée, par exemple si une ou plusieurs situations correspondant aux conditions d'application de l'agent sont activées, la règle intégrée à l'action va s'appliquer. Dans ce cas, si la condition d'application de l'action est remplie, l'action va être proposée à l'agent. Cela signifie que l'action va être ajoutée à l'ensemble des actions actuellement proposées. L'agent vérifie de façon régulière les actions proposées et exécute les plus activées si elles ne vont pas à l'encontre de ses objectifs.

## 8.4. Déduction

L'apprentissage inductif permet à l'agent d'identifier et créer de nouveaux concepts. L'apprentissage déductif doit lui permettre de créer leurs conditions d'utilisations nécessaires. La déduction fonctionne comme un ensemble de règles décrites explicitement dans le graphe et analysant différents composants pour en créer d'autres. Elles peuvent être spécialisées en fonction du type de donnée analysée (objet, objectif, règle,…) ou du type de concept à créer (situation, règle, hypothèse,…). Le fonctionnement global (application de règles) restant le même, le bon fonctionnement du modèle sera illustré ici à travers des règles souvent spécifiques à la perception et à l'intégration du nouveau concept induit. Nous allons ainsi montrer comment, par l'application des règles de déduction, il est possible à la fois de

modifier le système perceptif de l'agent pour inclure ce nouveau concept et de développer certaines hypothèses pour développer et intégrer des heuristiques de perception plus efficaces.

On peut distinguer plusieurs catégories de règles principales pour atteindre cet objectif. Comme nous le verrons, ces différentes étapes ne nécessitent pas les précédentes et ne sont pas obligatoirement réalisées immédiatement à la suite l'une de l'autre. La première étape pour obtenir un nouveau concept intégré consiste à créer ce nouveau concept par induction, cette création étant réalisée au cours d'un processus de raisonnement explicite. A partir de ce nouveau concept et de sa définition, l'agent peut en déduire une nouvelle perception, les règles et les situations associées, ce qui rend ce concept utilisable. En analysant les situations créées, il peut les fusionner avec des situations existantes et ainsi créer des liens d'activation et déduire des méthodes de perception plus efficaces. A partir de ce nouveau concept, comme de tout autre concept, il peut créer des hypothèses permettant de l'identifier plus rapidement. A partir de nouvelles hypothèses, il va ensuite modifier les perceptions associées pour lui permettre de tester leur validité. Enfin, une fois une hypothèse validée, il peut l'intégrer comme méthode de perception permanente. Ces différentes étapes sont décrites figure 8.4

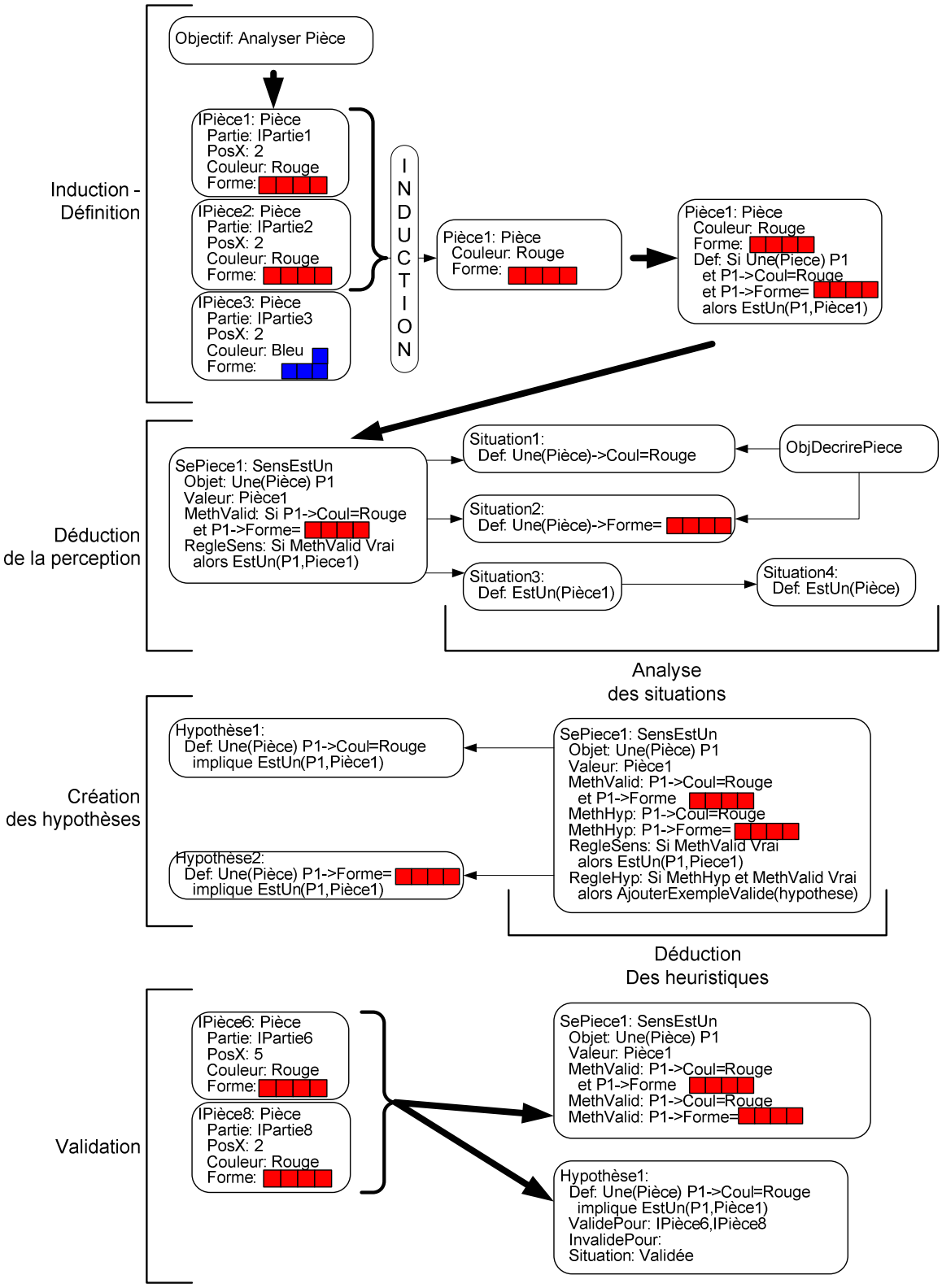



**Figure 8.4 Récapitulation des étapes de la déduction**

### 8.4.1. Création d'un nouveau concept par induction

La première étape consiste à induire un nouveau concept. L'agent a pour objectif d'analyser soit un concept précis, soit un concept aléatoire. Si l'objectif ne précise pas le concept, une règle choisit un concept avec une probabilité proportionnelle à son activation, et à condition que le temps total d'analyse de ce concept jusqu'à présent ne dépasse pas 10% du temps total. La méthode d'induction est appelée explicitement pour le concept sélectionné et l'analyse de données symboliques décrite au chapitre 0 nous permet d'extraire un nouveau concept avec tous ses attributs.

Une définition du concept est alors automatiquement déduite pour ce concept qui correspond à l'intersection de tous les attributs du concept. Il s'agit d'une définition logique étant donné que ces attributs ont été ajoutés car communs à tous les exemples qui définissent le nouveau concept.

Par exemple, une Barre peut être définie à partir des points communs à toutes les barres rencontrées, donc ayant comme forme quatre briques alignées et étant de couleur rouge.

### 8.4.2. Création d'une nouvelle perception

A partir d'une description comme la définition du concept, l'agent peut déduire une perception adaptée et les situations associées.

La méthode de perception utilisée (*MethValid*) correspond à l'ensemble des conditions de la définition.

La règle de perception appelle la méthode valide actuelle et, si celle-ci est vérifiée, affecte le type à l'objet étudié (par exemple, affecte le type *Piece1* à la pièce analysée).

Les conditions et le résultat de la perception permettent de déterminer les situations associées. Pour chaque condition unitaire (Test, Affectation, …), l'agent crée une situation définie comme la description de cette condition. Par exemple, « *Une piece est rouge* » ou « *une pièce est une Piece1* ». Ces situations vont permettre à la fois de lier les concepts entre eux et de raisonner plus facilement.

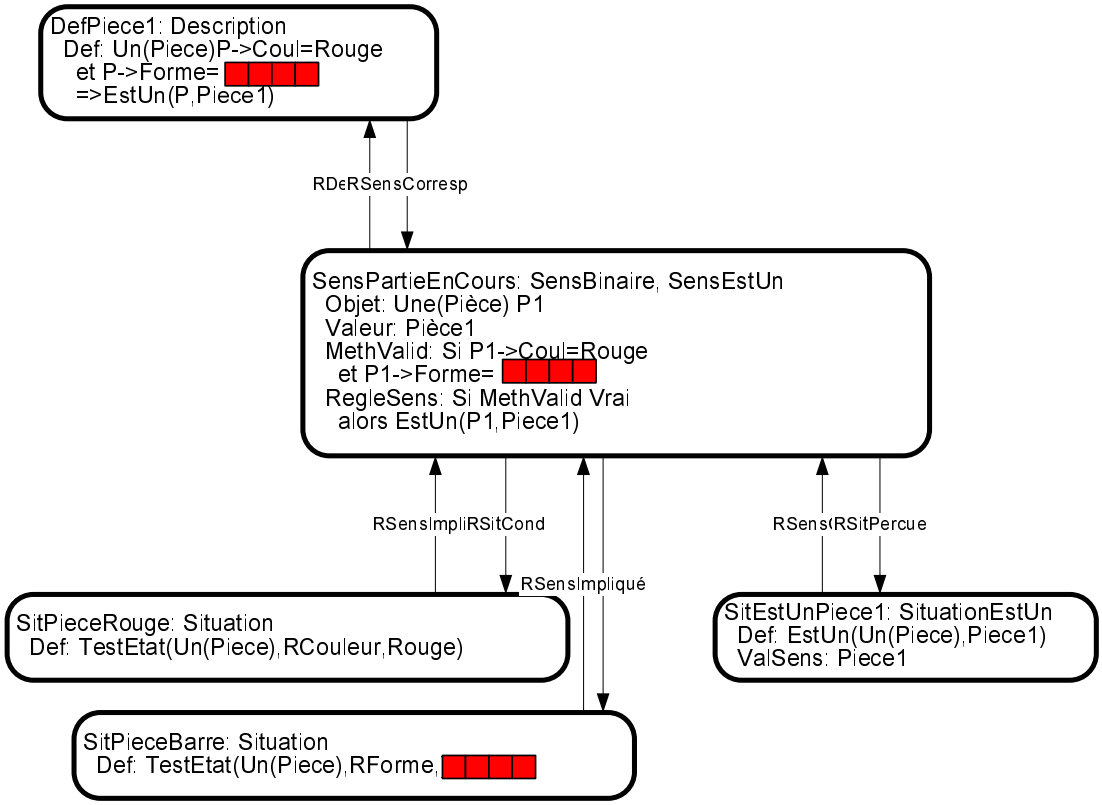



**Figure 8.5 Exemple de perception déduite de la définition de Pièce1**

La déduction peut se faire à partir de toute description, par exemple à partir des règles du jeu qui sont données sous cette forme à l'agent. Ainsi, à partir de la règle « *LumièreP est allumée implique PartieActuelle est EnCours* », il déduit la perception et les situations suivantes :

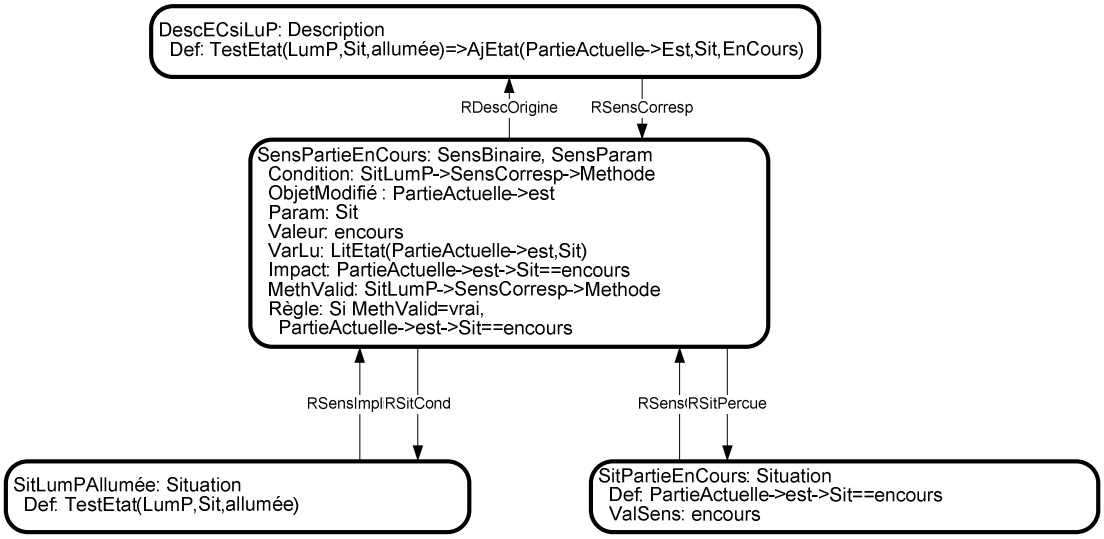



**Figure 8.6 Exemple de perception déduite de la description du jeu**

### 8.4.3. Analyse des situations associées

Les situations associées à la nouvelle perception peuvent être comparées aux situations déjà présentes dans le graphe pour être liées ou fusionnées.

Par exemple, les situations décrivant la couleur et la forme de *Pièce1* sont des spécialisations des situations associées à l'objectif de description d'un objet. Ces objectifs vont être liés afin à la fois que l'activation de l'objectif active la perception, et pour que l'agent puisse proposer des hypothèses sur la méthode d'atteindre l'objectif à partir du nouveau concept.

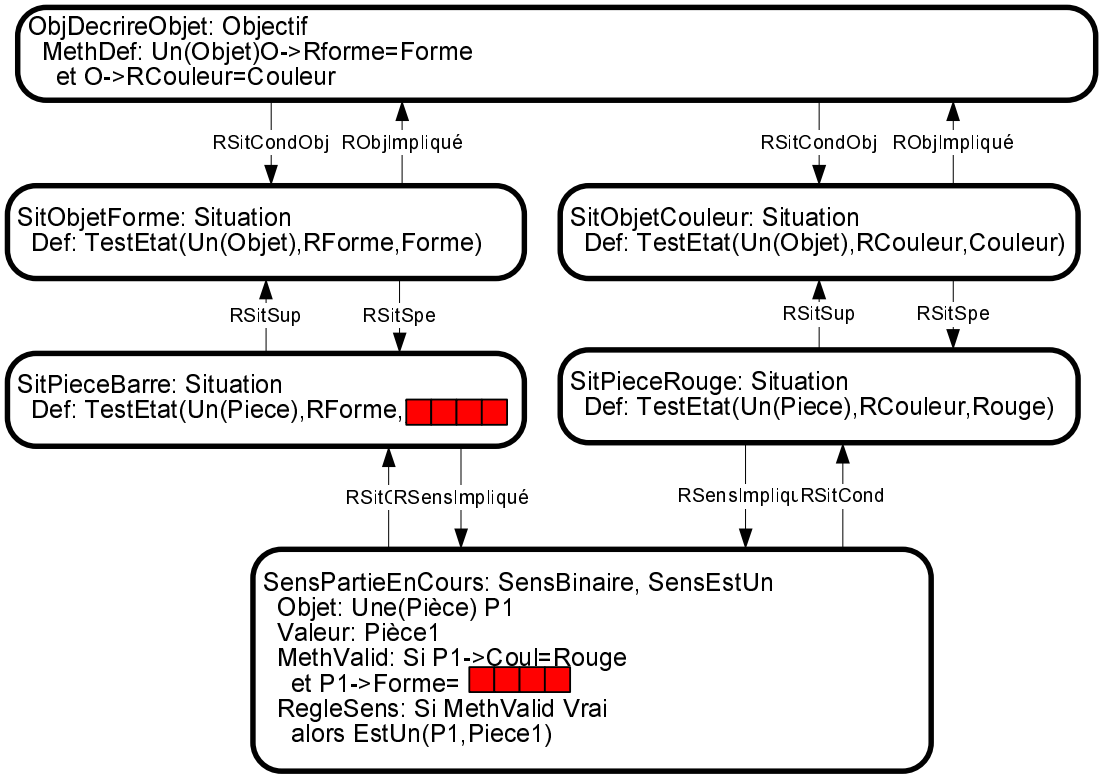



**Figure 8.7 Analyse des situations issues de SensPiece1**

Dans le cas de la règle qui décrit la partie en cours, la situation créée (« *LumP->Sit=Allumée* ») a une définition exactement identique à la situation correspondant à la perception de l'état de *LumP*. Les deux situations sont donc fusionnées (c'est-à-dire que tous leurs attributs et leurs liens sont mis en commun) et une nouvelle méthode est proposée pour la perception de *PartieActuelle->Sit* qui utilisera directement la méthode de la perception correspondant à la situation fusionnée (*SeLumP*). Ainsi, lorsque l'agent désirera connaître l'état de la partie, il utilisera la méthode de *SePartieActuelle* qui utilisera directement le résultat de *SeLumP* pour retourner un résultat correct le plus rapidement possible (comme présenté en figure 8.7)

### 8.4.4. Création de nouvelles hypothèses

A partir de la définition stricte, certaines règles vont chercher à relâcher certaines conditions pour générer des hypothèses sur la façon de percevoir un concept. Ainsi, une hypothèse va être créée correspondant à chaque condition unitaire de la définition. A partir de la définition « *Une pièce P est une Pièce1 si la forme de P est une Barre et la couleur de P est Rouge* », il va donc créer deux hypothèses : « *Une pièce P est une Pièce1 si la forme de P est une Barre* » et « *Une pièce P est une Pièce1 si la couleur de P est Rouge* ».

D'autres méthodes de création d'hypothèses sont possibles en plus de celle-ci qui est appliquée systématiquement :

- Relâcher la condition sur le type de l'objet : « *Un objet P est une Piece1 si…* » à la place de *Une Piece P*.
- Réaliser des conjonctions d'hypothèses invalidées. Si toutes les hypothèses associées à un concept ont été rejetées, l'agent va associer les deux qui ont eu le taux de succès le plus important pour en créer une nouvelle.
- Recherche des attributs ayant une valeur majoritairement commune dans les exemples. Par exemple, si la plupart des *Pièce1* sont en position *PosX*=2, il va créer une hypothèse « *EstUn(Un(Piece)P,Piece1) implique P->PosX=2* ». Cette hypothèse va générer une nouvelle perception associée à *Piece1->PosX* et être lié à *SensPiece1* notamment à travers la situation *SitEstUnPiece1* qui va être fusionnée.

### 8.4.5. Création des heuristiques associées aux hypothèses

De manière identique à la déduction à partir des définitions, l'agent va déduire des méthodes associées aux hypothèses qu'il va placer dans les perceptions correspondantes. Ces méthodes seront placées dans un attribut *MethHyp* qui sera utilisé par la règle de test des hypothèses.

Une nouvelle règle de test des hypothèses sera ajoutée qui va comparer le résultat d'une méthode validée à celui d'une méthode hypothétique. Si les résultats sont identiques, l'hypothèse sera validée pour cet objet. S'ils sont différents, elle sera invalidée.

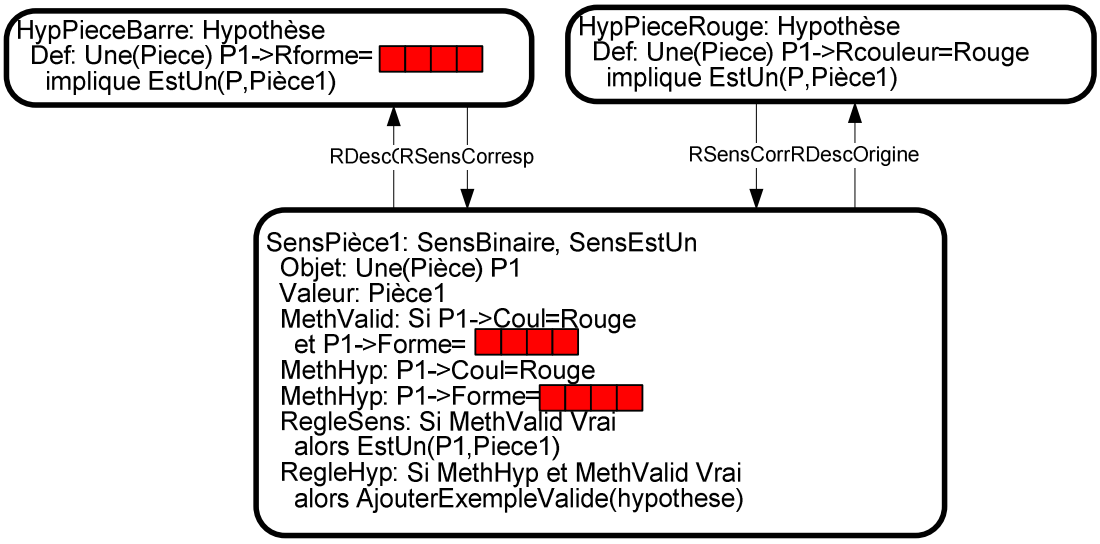



**Figure 8.8 Déduction des méthodes heuristiques et des tests des hypothèses**

## 8.4.6. Validation des hypothèses

Une fois qu'une hypothèse a un nombre total d'exemples suffisant, il va pouvoir la valider ou l'invalider. Si elle est validée, la méthode passe de l'attribut *MethHyp* à *MethValid* et peut alors être utilisée par la règle de perception à la place de la méthode stricte.

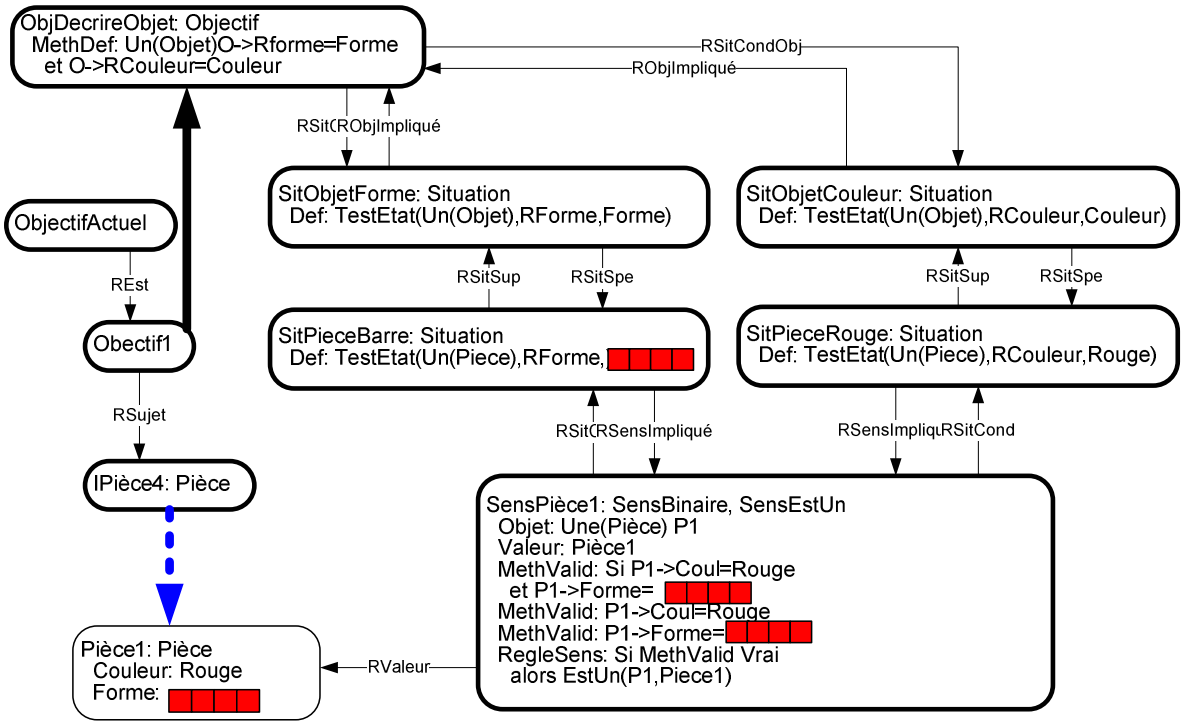


**Figure 8.9 Exemple de fonctionnement d'une nouvelle perception avec heuristiques de perceptions validées**

Dans l'exemple représenté figure 8.9, l'hypothèse « *Si une pièce est rouge, c'est une Pièce1* » a par exemple été validé. A partir de là, on peut illustrer l'utilisation et l'apport de cette nouvelle heuristique. Si l'objectif actuel (*Objectif1*) est par exemple de décrire *IPièce4*. L'objectif sera atteint lorsque *IPièce4* aura une *Forme* et une *Couleur*. Les règles de description associées à *ObjDecrireObjet* permettent de trouver rapidement la couleur alors que la forme est beaucoup plus longue à décrire.

L'activation de l'objectif actuel *Objectif1* va activer *ObjDecrireObjet*, qui va activer les situations associées qui vont elles-mêmes activer *SensPièce1* dont les règles auront ainsi de bonnes chances d'être exécutées.

Dès que la couleur d'*IPièce4* est déterminée, la règle de *SensPièce1* pourra alors s'exécuter : si la couleur de *IPièce4* est *rouge*, *IPièce4* est un *Pièce1*.

Dès que *IPièce4* est associé à *Pièce1* (flèche bleue), il en acquière les attributs, et en particulier la forme. *IPièce4* a alors une couleur et une forme, et *Objectif1* est réalisée sans que l'agent ait eu à décrire la forme de la pièce.

Au début, le choix de la méthode parmi toutes les méthodes valides est aléatoire, les liens avec la perception étant identique. Le renforcement va permettre de renforcer l'utilisation de la meilleure méthode car elle sera utilisée avec succès avant la méthode stricte et donc renforcée plus souvent (voir section 9.3).

## 8.5. Résultats

Comment mesurer l'efficacité de notre système déductif ? Son objectif est de construire des concepts utilisables par l'agent. Nous allons donc illustrer des concepts construits en fonction des types de construction réalisés. Il faut rappeler que nous nous sommes limités ici à construire de nouvelles méthodes et hypothèses d'identification. Il serait intéressant par la suite de compléter ce système par des règles permettant par exemple la déduction de nouvelles actions. Cela ne nécessitera en aucun cas de changer le fonctionnement du système. Le bon fonctionnement de celui-ci peut donc être illustré au travers du fonctionnement des règles d'identification proposées ici.

### 8.5.1. Nouvelle perception

L'agent doit être capable de créer de nouvelles perceptions pour apprendre à identifier un nouveau concept. Par exemple, à partir de la définition du concept de *nPiece_p*, qui correspond au concept de *PièceObservée*, la perception obtenue est la suivante :

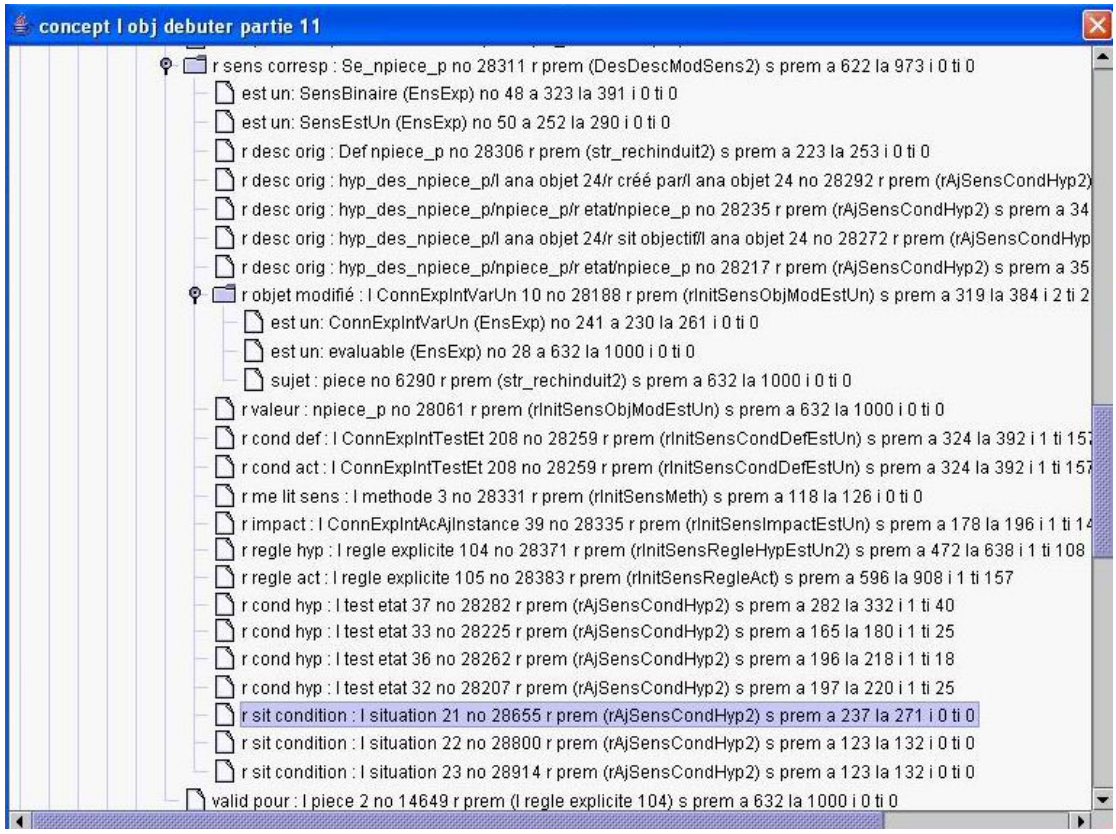



**Figure 8.10 Exemple perception issue du nouveau concept de Pièce observée (*nPiece_p*)**

On distingue en particulier les attributs suivants :

- Objet modifié (une *Pièce*) : correspond à ce qui peut être analysé par le sens
- Type de sens (*SensBinaire*, *SensEstUn*) : indique qu'il s'agit d'un sens binaire étudiant l'appartenance à un type.
- Valeur (*npièce_p*) : indique le type étudié.
- Règle actuelle (*I règle explicite 105*) : règle créée par déduction qui va vérifier si la condition valable actuelle (*cond_act*) est vérifiée pour l'objet (*Une Pièce p1*) et si c'est le cas affecter le type (*npièce_p*) à l'objet (*p1*).

- Règle Hypothèse (I *règle explicite 104*) : règle créée par déduction qui va vérifier si une condition hypothèse (*cond_hyp*) atteint un résultat identique à la condition actuelle, auquel cas l'hypothèse source est validée pour cet objet.
- Situations liées (*I situation 21* à *23*) : correspondent aux différents tests individuels de la définition du concept.

### 8.5.2. Fusion de situations

Les situations sont des concepts spécifiques servant à simplifier l'analyse par l'agent et d'intermédiaire pour la transmission de l'activation.

Par exemple, les situations peuvent être utilisées pour comparer des conditions d'une perception aux conséquences d'une autre. Ainsi, l'agent a au départ une perception qui lui indique l'état de la lumière *LumP*. Il a ainsi deux perceptions, une qui lui indique si *LumP* est allumée, une qui lui indique si *LumP* est éteinte.

L'agent dispose également au départ de la description du jeu « *Si LumP est allumée, alors la partie actuelle est en cours* ». Il en déduit une perception adaptée à la situation *PartieActuelle* est *EnCours*. Il va également chercher si des situations n'existent pas déjà qui correspondent aux situations conditions de son application (*LumP* allumée). Si c'est le cas il va les fusionner et modifier sa méthode pour utiliser la méthode correspondant à la situation-condition.

La figure 8.11 représente les deux situations correspondant aux deux perceptions (les différents composants de la fenêtre sont décrits en section 10.1.3.4):

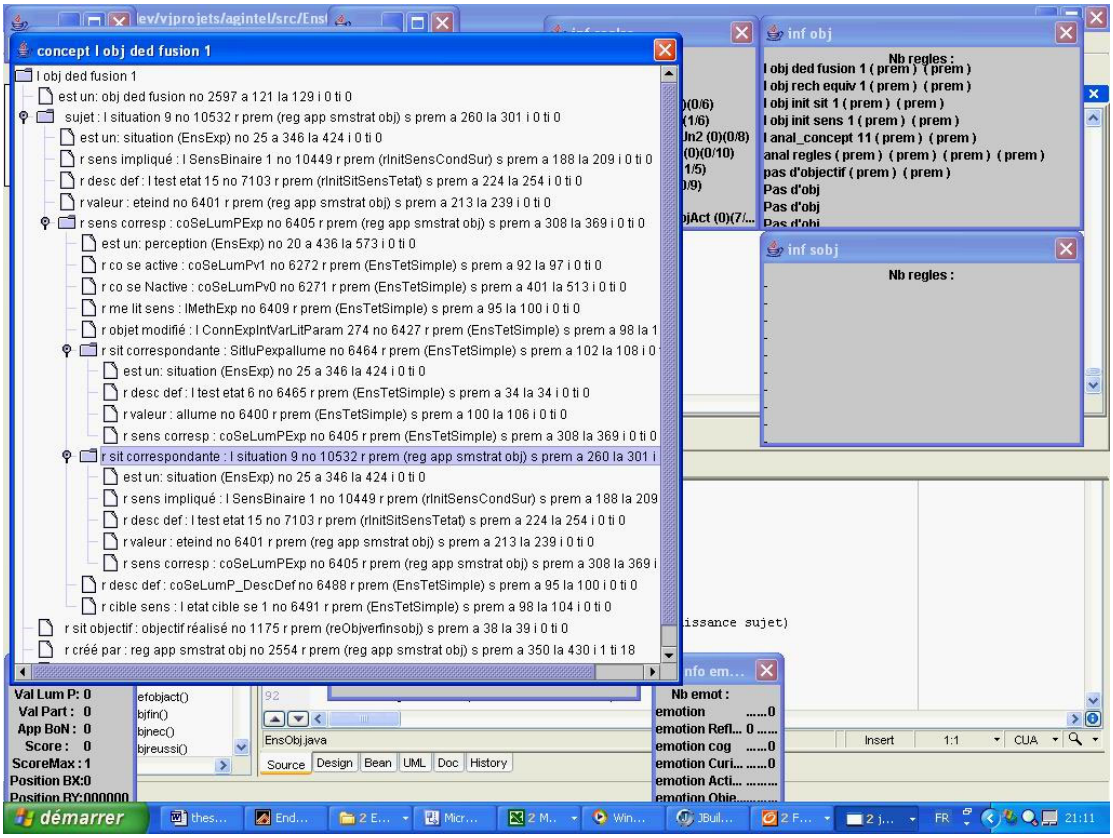



**Figure 8.11 Exemple de situations en cours de fusion : *SitLupexpallume* issue du sens *coSeLumPExp* et *Isituation9* issue du nouveau sens *ISensBinaire1***

La figure 8.12 représente la perception de la partie actuelle avec sa nouvelle condition actuelle et la situation fusionnée.

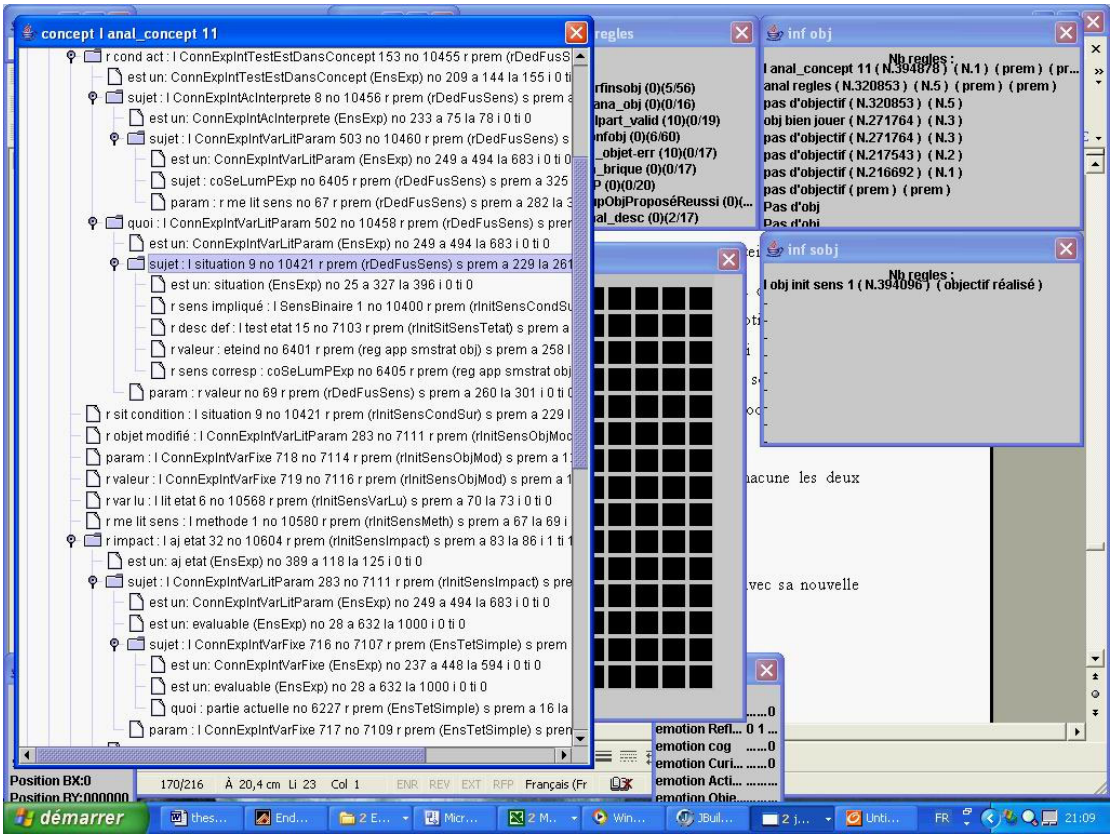



**Figure 8.12 Exemple de méthode issue de situations fusionnées : la condition actuelle (*r cond act*) du nouveau sens interroge (par *IConnExpIntAcInterprete8*) directement la méthode (*r me lit sens*) du sens *coSeLumPExp***

### 8.5.3. Nouvelles hypothèses et validation

A partir de la définition du concept, le système construit automatiquement des hypothèses correspondant aux différents éléments indépendants de la définition, dans le but de les tester pour les valider.

Par exemple, à partir du concept de *nPiece_p*, qui correspond au concept de « Pièce observée » en général, il construit le concept de *nnPièce_p_p* qui correspond au concept de *Barre*, et en extrait des hypothèses. Par exemple, l'hypothèse « *Une npiece_p est rouge implique que c'est une Barre* » représentée figure 8.13 :

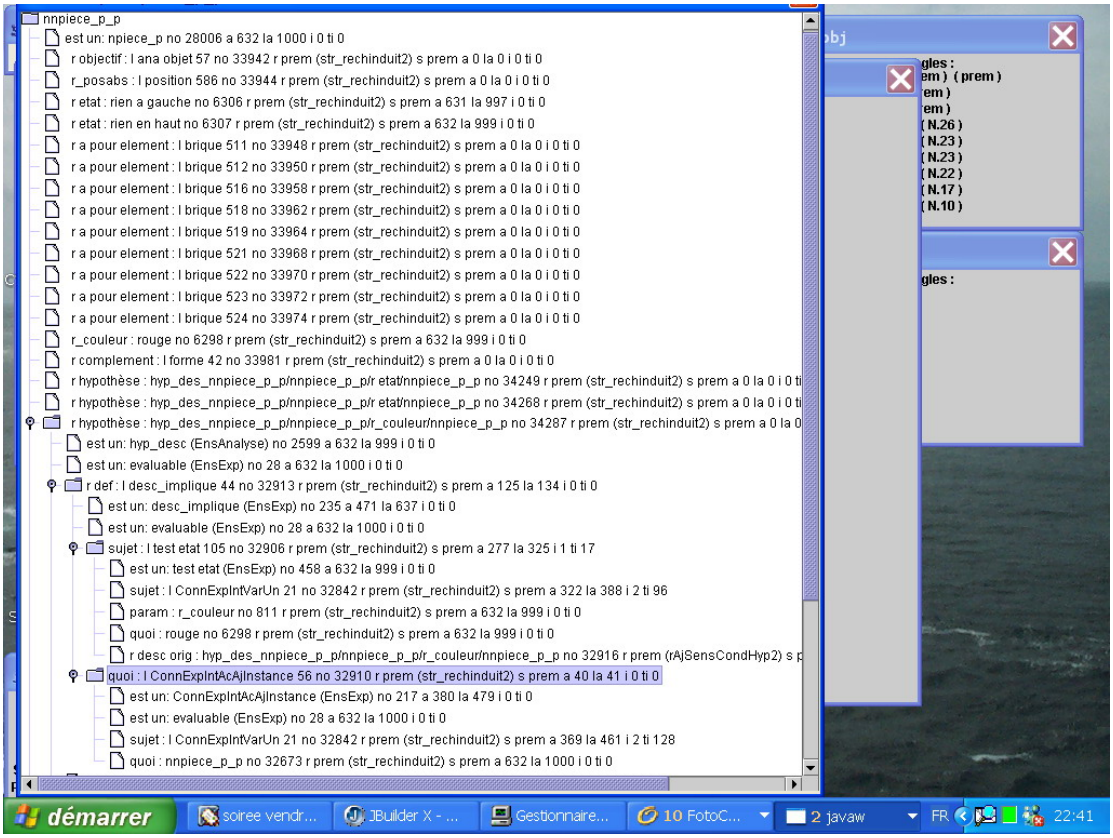



**Figure 8.13 Exemple d'hypothèse issue du nouveau concept de *Barre* (*nnPiece_pp*)**

Pour les tester, l'agent intègre une nouvelle règle dans la perception qui va tester les différentes hypothèses (*CondHyp*) et les comparer au résultat des règles validées. A partir de l'observation de plusieurs nouvelles pièces, il peut valider une hypothèse dont la condition d'application passe du statut de *CondHyp* (à tester) au statut de *CondAct* (utilisée directement).

## 8.6. Conclusion

Les connaissances données à l'agent déterminent en grande partie ce qu'il fait et comment il le fait. Nous avons présenté dans ce chapitre les connaissances que nous avons donné à l'agent afin de lui permettre de raisonner, percevoir et déduire de nouvelles perceptions de façon efficace. Du fait de sa nature même (description de la sémantique et des connaissances données à l'agent), cette partie est la seule qui soit dépendante de la sémantique de l'agent.

Les concepts liés aux perceptions sont destinés à fonctionner de façon aussi bien proactive que réactive. Ils ont de plus été choisis afin de permettre une évolution de la

sémantique de l'agent, et donc des concepts qu'il cherche à percevoir ou des actions qu'il cherche à effectuer.

Les règles et concepts liés à la déduction permettent d'intégrer les nouveaux concepts appris, notamment par induction, dans le raisonnement de l'agent. Pour que les concepts puissent être utilisés, des règles de perceptions sont déduites. Pour qu'elles soient utilisées efficacement, des concepts correspondants aux différentes situations liées à la perception sont créés. La transmission de l'activation, modifiée grâce au renforcement dû aux émotions, permet ainsi d'activer les règles utiles en fonctions des circonstances actuelles (donc des situations activées). Enfin, des hypothèses sont générées automatiquement afin d'être testées et éventuellement validées, permettant d'utiliser des heuristiques plus efficaces que les règles strictes issues de la définition même des concepts perçus.

# Chapitre 9
# Contrôle de l'agent

## 9.1. Présentation : Emotions et Objectifs

Que va faire l'agent ? Dans quel but ? Qu'est-ce qui va l'inciter à utiliser les bonnes connaissances au bon moment ?

Dans des systèmes tels qu'EURISKO ([Lenat 1983a]) ou MACISTE ([Pitrat 1996]), l'agent est guidé par des objectifs explicites, influencé par des évaluations des concepts réalisées elles-mêmes par des règles explicites. Par exemple, lorsqu'EURISKO crée un concept, il lui attribue un intérêt fonction de la règle qui l'a créé et du concept d'où il est tiré.

Lorsqu'un système n'utilise pas de méta-niveau, en particulier si le système est réflexif et qu'il peut créer et modifier ses objectifs, le risque est fort qu'il crée des objectifs stériles. Par exemple, EURISKO a créé une règle d'évaluation s'attribuant l'intérêt de toutes les autres règles et une autre cherchant à supprimer tous les autres concepts. Ces objectifs ne correspondent pas au fonctionnement souhaité. L'intervention de l'utilisateur est alors nécessaire pour aider le système à évaluer les concepts et le guider dans le choix des objectifs.

Or, notre objectif est de concevoir un modèle d'agent. Un agent étant par définition autonome, l'intervention de l'utilisateur doit être minimale après la mise en service de l'agent, si possible absente. Pour réaliser le contrôle à long terme de l'agent, nous allons donc nous inspirer de la méthode utilisée par CopyCat pour diriger progressivement son programme vers une solution : CopyCat n'utilise pas directement d'objectif ou de planification, mais une fonction de température qui dépend de la structure du système et qui va influencer le raisonnement en favorisant certaines règles plutôt que d'autres.

Nous généralisons ce système sous le terme d'émotions, des fonctions qui influencent le raisonnement de l'agent en fonction de l'état du système et de l'environnement. L'avantage de ce type de fonctions est qu'elles sont implicites et non modifiables par l'agent. Comme elles

ne dépendent que de la structure et de l'environnement, elles peuvent s'adapter aux nouveaux concepts qui seront créés.

L'influence sur le raisonnement se fait à plusieurs niveaux :

- Comme dans CopyCat, par introduction d'activation dans le graphe pour modifier les règles et concepts choisis
- Par perception explicite des émotions pour permettre à l'agent de les rechercher explicitement par la suite
- Par renforcement ou destruction de liens lorsque de fortes variations des émotions ont lieu

Ce dernier avantage permet de plus l'intégration des nouveaux concepts dans le graphe. Alors que l'augmentation du nombre de concepts et de règle pourrait provoquer une baisse de l'efficacité de l'agent, car plus de règles doivent être appliquées, le renforcement par les émotions permet d'associer les bons concepts aux bonnes situations.

L'inconvénient de ce type de guidage provient du fait que l'agent ne connaît pas explicitement ses fonctions-émotions et qu'il ne peut en créer de nouvelles. Il ne peut donc pas planifier et faire des raisonnements structurés uniquement à l'aide des émotions.

Les deux systèmes sont complémentaires : une gestion des objectifs, une évaluation explicite des concepts et une planification permettent de réaliser des raisonnements structurés créés par l'agent. Les émotions permettent de guider à long terme, de l'inciter à choisir les bons concepts et les bons objectifs qu'il va ensuite traiter rationnellement.

## 9.2. Contrôle explicite

### 9.2.1. Principe

Pour raisonner de façon cohérente, l'agent a besoin de règles gérant un système de suivi d'objectifs et de sous-objectifs. Cette tâche est assurée dans AM et CopyCat par un agenda global, dans Eurisko par 3 niveaux d'objectifs (global-tache-concept).

Rien ne nous empêche d'utiliser un système complexe de gestion d'objectifs ou de planification tels qu'ils sont développés en intelligence artificielle. Il est souhaitable qu'il soit représenté de façon explicite afin que l'agent puisse lui-même l'améliorer, ne serait-ce que par la création de liens par renforcement entre les modules les plus utiles selon les circonstances.

En particulier, il est possible d'utiliser les modèles issues du monde des agents, tels qu'un modèle BDI ou un modèle de niveaux hiérarchiques (layers).

Pour simplifier le fonctionnement et la modélisation nous avons donc décidé de nous inspirer de l'agenda d'EURISKO. L'avantage principal est la simplicité du modèle, qui nous permet de décrire explicitement les règles de traitement des objectifs et donc de les rendre manipulables par l'agent.

Pour ne pas avoir à identifier des niveaux tout en gardant l'avantage de la hiérarchie, le système utilisé est une généralisation des niveaux d'EURISKO. Les règles peuvent proposer des objectifs au niveau global ou en sous-objectif d'un objectif en cours de réalisation, créant implicitement un sous-niveau. Par exemple :

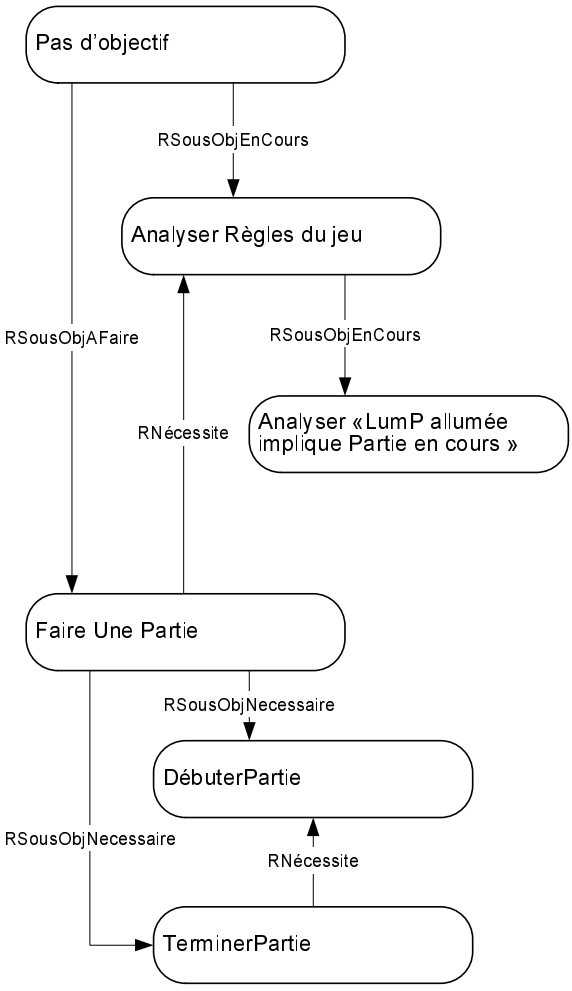



**Figure 9.1 : exemple d'arbre de sous objectifs à partir de l'objectif racine « Pas d'objectif »**

### 9.2.2. Règles de suivi des objectifs

Le suivi des objectifs suit le même principe que les autres règles : les règles de suivi sont appliquées aléatoirement en fonction de leur activation et permettent en particulier d'étudier l'objectif actuel (pour vérifier s'il est encore pertinent, s'il a été rempli), de vérifier si des sous-objectifs sont disponibles, si des sous-objectifs doivent nécessairement être exécutés et dans quel ordre, si il existe un autre objectif plus intéressant (plus activé),…

Par exemple, la figure 9.2 illustre l'évolution du concept fluide d' *ObjectifActuel* et des règles qui lui sont associées.

L'objectif de départ est de *FaireUnePartie*, ce qui active la règle « *Pour faire une partie, proposer un objectif Débuter P1* ». Cette règle étant fortement activée, elle aura de grande chance d'être exécutée, et quand elle l'est un nouveau sous-objectif est ajouté à *FaireUnePartie*, qui est *DébuterP1*. La règle de contrôle général qui vérifie la présence de sous-objectifs non réalisés, est elle tout le temps activée car lié au concept le plus général *Objectif*. Cette règle va changer l'objectif actuel qui va devenir *DébuterP1*.

Quand l'objectif actuel devient *DébuterP1*, de nouvelles règles sont activées alors que celles qui deviennent inutiles ne le sont plus. Ainsi, la règle « *Pour débuter une partie, appuyer sur le boutonP* » s'active, permettant d'atteindre l'objectif. Les liens avec les règles peuvent être créés explicitement, mais ils peuvent également être créés automatiquement à l'aide du renforcement des émotions pour atteindre une configuration plus efficace.

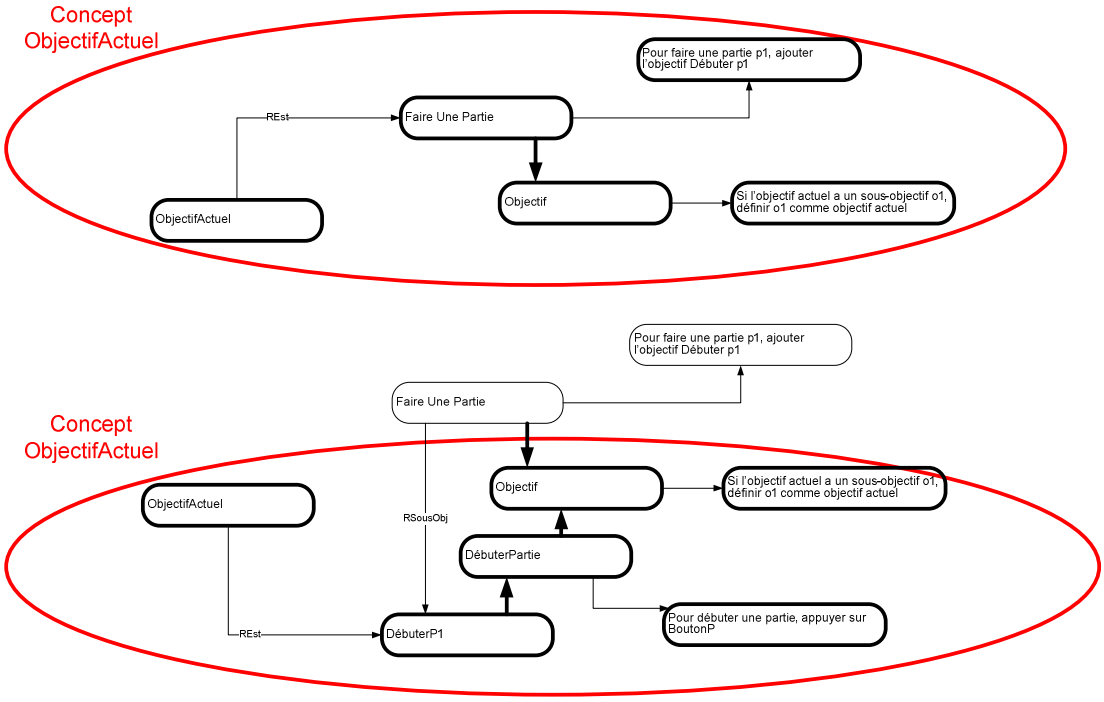



**Figure 9.2 : exemple de concept fluide associé au concept d'Objectif Actuel et ses règles associées**

## 9.3. Emotions : contrôle implicite et intégration

### 9.3.1. Emotions et apprentissage implicite

#### 9.3.1.1.Objectif

Fondamentalement, une émotion est juste une valeur, fonction de l'état de l'environnement et de la situation structurelle de l'agent. Cela lui permet de varier indépendamment des changements sémantiques opérés par l'agent dans son graphe de connaissances et donc de garder sa généralité malgré l'évolution de l'agent. Une émotion peut être positive (comme la curiosité ou le gain au jeu) et dans ce cas elle aura des conséquences positives (renforcement) et l'agent cherchera à renouveler les situations où il l'a perçue. Elle peut également être négative (ennui) et elle aura alors des conséquences négatives (destruction de liens, voire construction de liens inhibiteurs), et l'agent cherchera à éviter les situations qui les ont provoquées.

Les émotions modifient le raisonnement de l'agent de trois façons : en introduisant de l'activation dans le graphe, en renforçant ou en détruisant des liens et au travers de perceptions explicites.

Certaines émotions introduisent directement de l'activation dans le graphe. Par exemple, l'émotion Objectif, correspondant à la concentration, active légèrement le concept d'objectif à chaque période afin que l'agent soit incité à exécuter les règles de gestion d'objectif (vérifier que l'objectif est terminé, s'il y a un sous-objectif,…) et pour favoriser les concepts liés à l'objectif actuel. De même, l'émotion spécifique au jeu favorise certains objectifs précis, comme celui incitant à étudier le jeu et donc à essayer de faire des parties et à les analyser.

De façon plus générale et avec une influence à plus long terme, une forte variation d'une émotion va provoquer un renforcement ou un affaiblissement des liens entre les connaissances qui ont connu la plus forte augmentation de leur activation. Plus précisément, s'il s'agit d'une émotion positive (Curiosité, Objectif, Jeu, …), le lien entre ces connaissances sera renforcé. S'il s'agit d'une émotion négative (Ennui), l'intensité du lien sera réduite ou un lien inhibiteur sera renforcé. Ce renforcement permet de créer un apprentissage implicite de réactions réflexes de l'agent.

La modification due au renforcement varie en fonction de la stabilité du lien. Celle-ci correspond à la confiance dans la valeur d'intensité actuelle du lien. Au fur et à mesure des renforcements, la valeur de vérité du lien (l'intensité) tend ainsi vers sa valeur finale. Une stabilité très faible correspond à un lien très malléable. Une stabilité proche de 0, donc un lien créé par renforcement et rarement renforcé, disparaît. Cela permet d'éviter une explosion du nombre de liens inutiles dans le graphe en ne conservant que ceux qui sont régulièrement renforcés.

L'agent perçoit également explicitement les variations des émotions. Cela lui permet de savoir dans quelles circonstances elles ont eu lieu et ainsi il peut chercher explicitement des moyens de les reproduire ou de les éviter.

#### 9.3.1.2.Renforcement

La formule générale du renforcement est définie en utilisant les variables suivantes :

- $i_t$ : facteur d'intensité du lien au temps t
- $s_t$ : facteur de stabilité du lien au temps t

- Δe : variation du niveau de l'émotion déterminant le renforcement
- n : intensité de l'émotion, constant pour une émotion donnée. Plus n est élevé, plus l'émotion aura d'impact sur les liens.
- α : constante servant à déterminer la vitesse de variation de l'intensité. Plus α est élevé, moins l'intensité variera rapidement.
- β : constante servant à déterminer la vitesse de variation de la stabilité. Plus β est élevé, plus le lien se stabilisera rapidement.
- λ : constante servant à déterminer la vitesse de détérioration du lien. Plus λ est élevé, plus la détérioration sera rapide.

$$i_{t+1} = \frac{\alpha s_t i_t + n\Delta e}{\alpha s_t + n}$$

$$s_{t+1} = s_t + (1 - s_t)\alpha\beta$$

La nouvelle intensité du lien ($i_{t+1}$) est une moyenne pondérée de l'ancienne ($i_t$) et de l'évolution de l'émotion (Δe). L'intensité varie d'autant moins que le facteur de stabilité ($s_t$) est important (pondération en faveur de $i_t$) et d'autant plus que l'émotion a une intensité importante (pondération en faveur de Δe).

Le facteur de stabilité ($s_t$) augmente au fur et à mesure des renforcements. Il augmente d'autant plus qu'il était faible jusqu'à présent.

Ces fonctions ont l'avantage de permettre une influence importante sur l'intensité dès les premiers renforcements tout en autorisant un ajustement rapide et peu d'effet mémoire. Un renforcement alterné aboutira par exemple à une stabilisation autour d'une intensité nulle. Une intensité nulle avec un facteur de stabilité important signifie qu'il n'y a significativement aucun lien entre les deux connaissances.

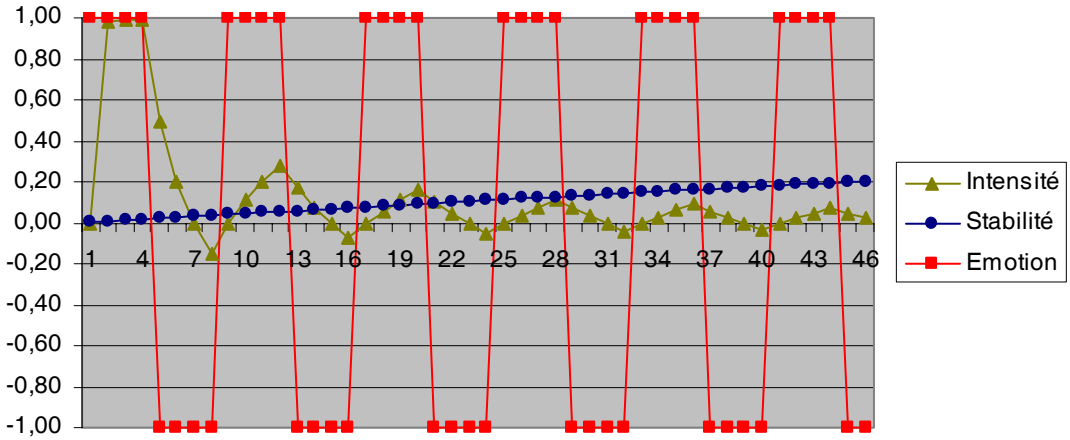



**Figure 9.3 : Evolution de l'intensité d'un lien subissant un renforcement alterné (n=0,5 ; α=100 ; β=0,0001 ; λ=0)**

Il faut ajouter à ces fonctions la formule d'actualisation de la stabilité du lien. En effet, si un lien a un facteur de stabilité faible, donc s'il a été peu renforcé, le lien aura tendance à se détériorer. Le facteur aura donc tendance à diminuer d'autant plus vite qu'il est déjà faible. Au dessous d'une certaine limite, le lien est détruit.

$$s_{t+1} = s_t - \frac{(1 - s_t)}{\lambda}$$

Après la fonction de renforcement, le second critère important est le choix des liens à renforcer. Ce choix dépend bien sûr de l'émotion considérée. Certains critères généraux peuvent toutefois se dégager quant aux concepts choisis :

- Les concepts les plus activés dans les périodes précédentes ont en grande partie déterminé le comportement de l'agent. Ils ont en effet transmis leur activation aux règles qui ont été exécutées et aux concepts qui ont été choisis. Si une émotion positive est survenue, cet « état mental » composé de l'ensemble de ces concepts a donc intérêt à être renforcé. Au contraire, si c'est une émotion négative, il a intérêt à être inhibé.
- De façon plus particulière, les règles et plus précisément encore les règles qui ont pu s'exécuter, constituent une cible intéressante à inhiber ou à renforcer.
- De même, les situations apparaissent utiles à renforcer. Elles correspondent en effet à la description d'un élément du monde actuel (par exemple « Une pièce est de couleur rouge » ou « L'objectif actuel est de décrire un objet »).

- Les objectifs, et particulièrement les objectifs en cours, par leur rôle dans le contrôle explicite, sont également à considérer.
- Sur-Types : Quel que soit le concept, il est souvent intéressant de renforcer non seulement le concept C lui-même, mais aussi les concepts-types dont C est une instance. Par exemple, si on renforce l'objectif particulier *IObjDecrireObjet1*, il peut également être intéressant de renforcer les concepts plus généraux *ObjDécrireObjet* et *Objectif*. L'intérêt du renforcement est de réactiver les concepts dans une situation *similaire*. On ne retrouvera alors probablement pas spécifiquement *IPièce1* ou *IObjDecrireObjet1*, mais une *Pièce* ou un *ObjDecrirePiece*. Prendre des catégories trop large (comme *Objectif* ou *Objet*) n'est pas important car les renforcement positifs et négatifs vont alors se compenser (voir exemple de l'émotion Concentration).

## 9.3.2. Exemples d'émotions

### 9.3.2.1. Concentration (suivi d'objectifs)

L'émotion de suivi d'objectif permet à l'agent de rester concentré sur son objectif et son plan actuel. Cette émotion introduit une faible activation dans le graphe au niveau d'une connaissance spéciale qui est liée à la fois aux règles de suivi d'objectif et à l'objectif actuel. Ainsi, les règles permettant la gestion des objectifs sont plus souvent appliquées, ce qui permet à l'agent de gérer l'évolution et la validité de son plan, alors que les règles associées à l'objectif actuel permettent de l'atteindre.

L'intensité de l'émotion augmente lorsqu'un objectif est atteint. Cela permet de renforcer les liens entre les concepts qui ont étés utilisés pour la résolution de l'objectif. L'intensité retourne ensuite lentement vers sa valeur moyenne.

Les concepts origines des liens renforcés sont les suivants :

- Objectif achevé et sur-types : le but de l'émotion étant d'activer les bons concepts lorsqu'un objectif se présente, on va renforcer les liens de l'objectif réussi vers tout ce qui a été utilisé (donc activé) durant les périodes précédentes :

Les concepts cibles des liens renforcés sont :

- Les situations les plus activées au moment du renforcement

- Les concepts – de façon générale – les plus activés au moment du renforcement
- Les règles ayant réussi à s'exécuter au cours des périodes précédentes

Ainsi, on peut distinguer deux effets pour l'objectif qui vient de s'achever et sa catégorie :

- Les règles ayant contribué à son succès sont renforcées
- Pour les catégories supérieures très générales, seules les règles utiles pour toute la catégorie d'objectif sont renforcées positivement en moyenne. Ce renforcement est adapté à la catégorie, car celle-ci est activée dès qu'une de ses instances est utilisée. Les règles plus particulières sont elles renforcées beaucoup plus souvent négativement (à chaque fois qu'elles sont appliquées pour un objectif où elles sont inutiles, et donc où l'objectif n'est pas atteint, conduisant à une diminution du niveau de l'émotion).

Par exemple, le lien le plus rapide et le plus stable créé par l'agent est le lien de la catégorie la plus générale (*Objectif*) aux règles générales de vérification de la définition de l'objectif actuel (*RObjTestDef*) et de l'objectif supérieur (*RObjTestDefSup*). Ces règles sont logiquement les plus utiles, quelque soit l'objectif, pour sa résolution, puisque ce sont généralement elles qui déterminent s'il est ou non atteint.

#### 9.3.2.2.Jeu

L'émotion de jeu a deux objectifs : D'abord inciter l'agent à analyser le jeu. Pour cela, l'émotion active directement l'objectif correspondant à l'analyse du jeu.

Ensuite, réaliser un apprentissage implicite entre les concepts utiles pour gagner au jeu. Pour cela, l'intensité augmente lorsque le score augmente. Elle augmente également fortement à la fin de la partie, d'autant plus fortement que le score est faible par rapport au meilleur score actuel. Elle revient ensuite lentement à sa valeur moyenne.

#### 9.3.2.3.Curiosité

La curiosité a pour objectif d'apprendre à l'agent à analyser les concepts de façon implicite. Pour cela, l'émotion augmente lorsqu'un concept est créé, qu'un attribut est ajouté ou qu'un lien d'instanciation est ajouté. Cela permet de favoriser les circonstances qui conduisent à la construction de nouveaux concepts.

#### 9.3.2.4.Ennui/impatience

L'ennui est une émotion négative qui a pour but d'éviter qu'il ne se passe rien. C'est pour cela que c'est elle qui active légèrement les règles de suivi des actions proposées. Ces règles permettent à l'agent d'analyser les actions proposées automatiquement en fonction de la situation actuelle (voir section 8.3.2).

L'ennui augmente lorsque les règles échouent et qu'aucun attribut n'est modifié. Il diminue légèrement lorsque des règles réussissent. Une forte augmentation de l'ennui survient lorsqu'il ne se passe rien. Cela a deux conséquences : D'abord la forte intensité de l'émotion fait que les règles de suivi des actions proposées sont plus activées, ce qui incite l'agent à agir. Ensuite l'augmentation d'une émotion négative crée un renforcement négatif des liens entre les concepts utilisés, donc crée des liens inhibiteurs. Cela incitera l'agent à ne plus réunir les mêmes circonstances qui le conduisent à ne pas savoir quoi faire dans l'avenir.

#### 9.3.2.5.Efficacité

L'efficacité a pour but de favoriser des circonstances qui permettent une plus grande efficacité des règles et des stratégies d'objectif. L'émotion efficacité augmentera (entraînant un renforcement) lorsque le temps d'exécution d'une règle réussie ou le temps mis pour atteindre un objectif (les deux sont obtenus automatiquement) est inférieur au temps moyen (lui aussi enregistré en attribut automatiquement).

## 9.4. Résultats

L'objectif des émotions est de guider l'agent et d'intégrer les connaissances afin que les concepts utiles soient activés au bon moment. On peut illustrer le bon fonctionnement de ce système de deux façons :

- De façon explicite et qualitative au travers de la description des liens créés, afin d'étudier leur pertinence.
- De façon quantitative, en mesurant l'évolution du temps mis par l'agent pour atteindre un objectif.

Nous allons ici nous intéresser plus particulièrement à l'émotion Concentration, qui renforce les liens à la suite de l'atteinte d'un objectif, pour étudier certains de ses résultats.

### 9.4.1. Concepts liés

A travers notre interface, il est possible d'afficher l'ensemble des liens créés par renforcement depuis une connaissance. Les valeurs affichées sont :

- Le nom de la connaissance cible (par exemple *reObjCondObj*)
- L'intensité actuelle du lien
- La stabilité actuelle du lien
- Le nombre de renforcements subis
- Le nombre de renforcements avant de dépasser une stabilité de 0,9 (0 sinon)

L'avantage de l'exécution aléatoire de règles est que le renforcement va très vite sélectionner les règles utiles pour réitérer l'émotion. Par exemple, après trois objectifs de type *DécrireBrique*, on obtient les résultats suivants :

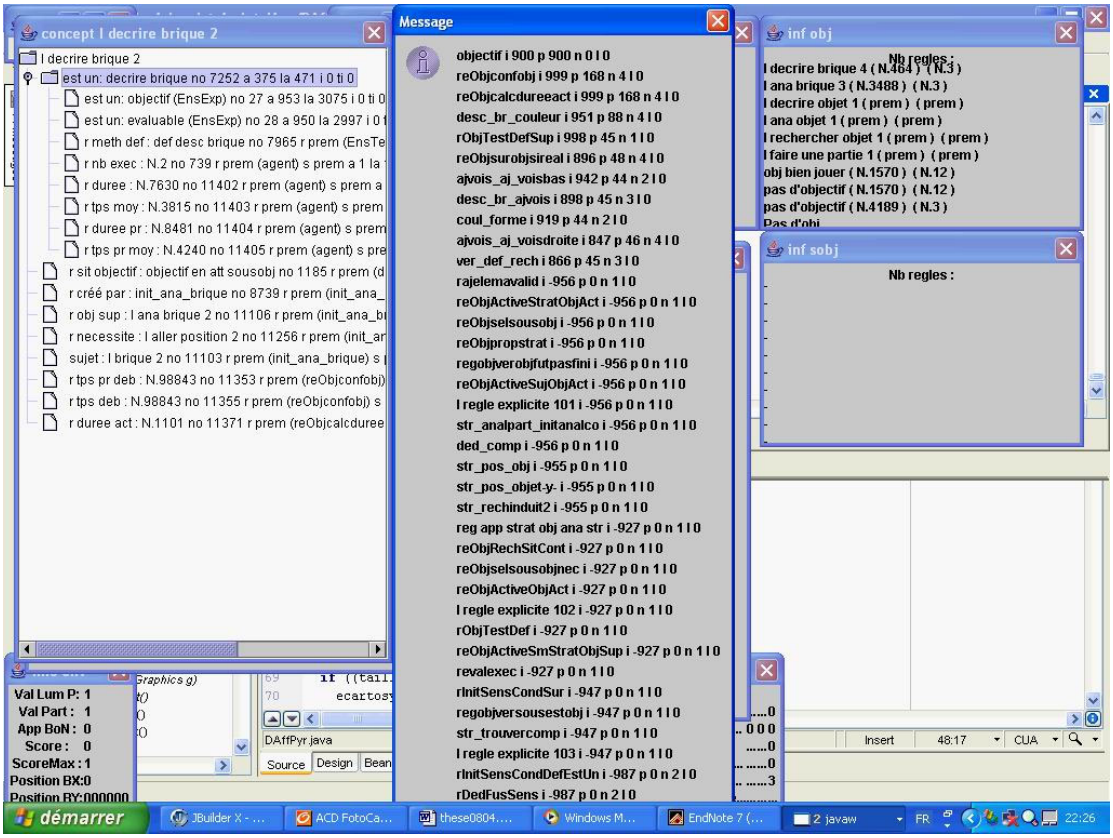



**Figure 9.4 : liens créés par renforcement après trois objectifs *DécrireBrique***

Les dix règles renforcées de façon relativement stable (une stabilité de 45 nécessite environ 500 cycles pour disparaître s'il n'est pas renforcé) peuvent être classés en trois catégories :

- Les règles utiles spécifiquement pour cet objectif (*desc_br_couleur* qui sert à décrire la couleur et *desc_br_ajvois* qui sert à ajouter un sous-objectif pour décrire les briques voisines)
- Les règles utiles aux sous-objectifs éventuels : *ajvois_aj_voisbas* et *ajvois_aj_voisdroite* servent à ajouter les voisins pour le sous-objectif *ObjAjVois*.
- Les règles générales, qui servent à confirmer un sous-objectif (*reObjconfobj*), mettre à jour les informations sur le temps passé sur l'objectif actuel (*reObjcalcdureeact*), vérifier si l'objectif supérieur est atteint (*reObjTestDefSup*) et l'objectif actuel (*reObjTestDef*)

Le fait de renforcer les sur-types aussi bien que l'objectif lui-même permet de renforcer les catégories les plus générales, comme *Objectif*. Dans ce cas, seules les règles générales sont utiles en moyenne et restent donc renforcées :


**Figure 9.5 : liens créés par renforcement sur le concept général *Objectif***

*reObjcalcdureeact*, *reObjconfobj*, *revalsup* (vérifie l'intérêt de passer à l'objectif supérieur comme objectif actuel), *reObjsurobjsireal* et *reObjverfinsobj* (vérifie si les sous-objectifs sont atteints) sont utiles pour une majorité d'objectifs et il est donc utile qu'ils soient activés en permanence.

## 9.4.2. Efficacité

### 9.4.2.1.Nombre d'exécutions

L'influence du renforcement se fait par une plus grande activation des concepts utiles. Les règles utiles doivent donc être exécutées plus souvent après avoir été renforcées lorsqu'on se retrouve dans une situation similaire. Ce phénomène peut être illustré au travers du pourcentage de sélection de la règle *« str_aller_pos »* qui est la seule règle nécessaire à l'objectif *AllerPosition* (ce que l'agent ignore). Les deux traits verticaux sur la figure 9.7 indiquent le passage à un objectif du type *AllerPosition*. Chaque période (en abscisse correspond à 300 exécutions de règles.

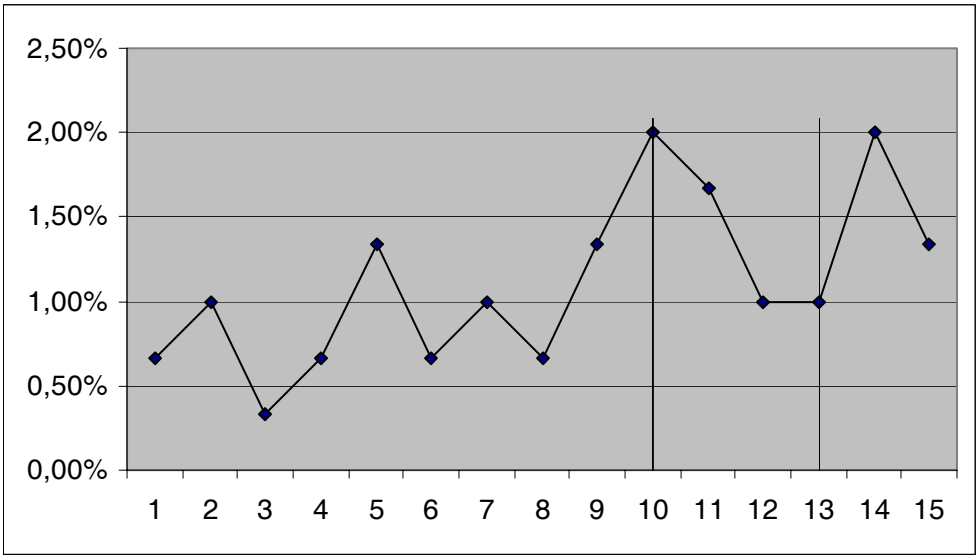



**Figure 9.6 : taux d'utilisation de la règle *str_aller_pos* lors de l'exécution de deux objectifs *AllerPosition* (aux temps t=10 et t=13), chaque période correspondant à 300 exécutions de règles**

On constate bien que le pourcentage de sélection de la règle augmente de moins de 1% à plus de 2% lorsque l'objectif correspond à *AllerPosition*. Dès la première exécution de l'objectif, la règle est en effet renforcée car utile et activée immédiatement. Lors de la deuxième apparition de l'objectif, la règle est automatiquement activée à travers le nouveau lien et son nombre d'exécutions augmente donc de la même façon.

#### 9.4.2.2.Temps par objectif

La mesure la plus adaptée pour mesurer l'efficacité globale du renforcement est le temps d'exécution pour atteindre un objectif. Ce critère est plus adapté que le nombre de cycles ou de règles appliquées (ce qui est équivalent car ces derniers ne prennent pas en compte l'augmentation du nombre de liens total du graphe dû au renforcement. Or, le nombre de liens peut diminuer l'efficacité car la transmission de l'activation entre les connaissances du graphe est alors plus lente. La mesure du temps pour atteindre l'objectif permet d'inclure ce type d'effets secondaires au renforcement.

Par exemple, le temps pour atteindre des objectifs du type *AllerPosition* est donner figure 9.7. Ce graphique correspond au temps moyen sur 5 exécutions pour les 27 premiers objectifs de ce type rencontré par l'agent.

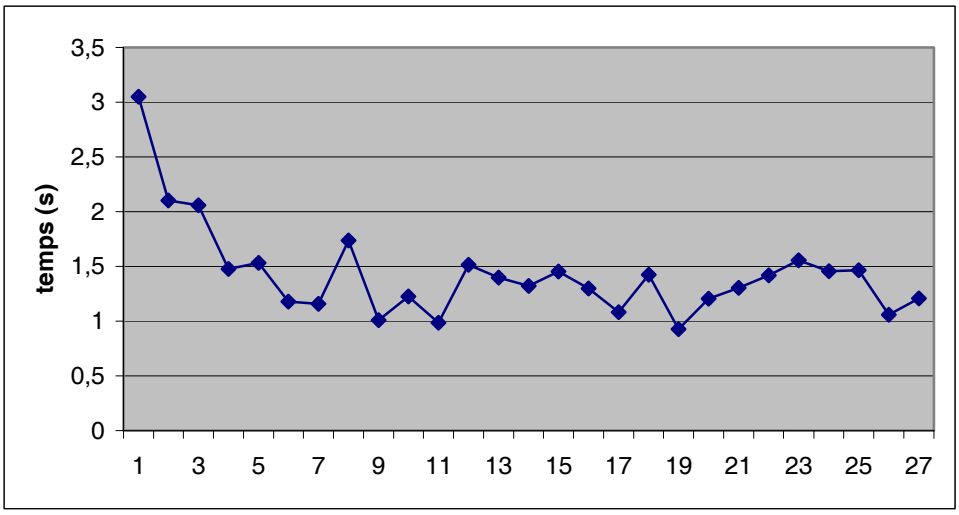



**Figure 9.7 : temps moyen pour atteindre les objectif de type *AllerPosition* (sur 5 exécutions pour les 27 premiers objectifs de ce type rencontrés)**

On constate que dès le deuxième objectif, l'agent gagne 50% d'efficacité. Cela provient du fait que le renforcement a un effet immédiat, même si il n'est pas encore complet car certaines règles peuvent être renforcées inutilement, ou inversement. Le temps d'exécution moyen continue donc à décroître et se stabilise autour d'une moyenne plus de deux fois inférieure au temps initial.

On obtient un résultat similaire sur un objectif un peu plus complexe nécessitant notamment l'exécution de sous-objectifs : *AnalyserBrique*, qui comprend le placement au niveau de la brique puis sa description, en ajoutant entre autre ses voisins éventuels.

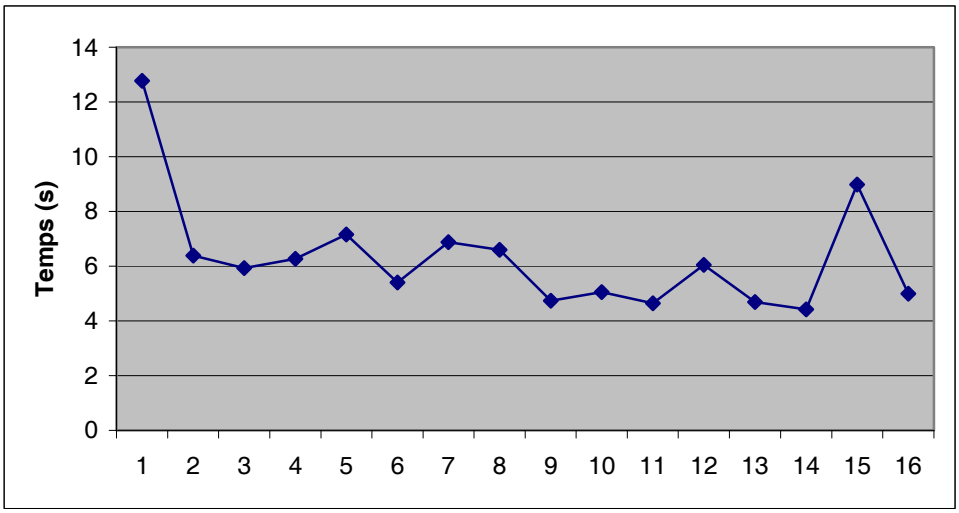



**Figure 9.8 : temps moyen pour atteindre les objectif de type *AnalyserBrique* (sur 5 exécutions pour les 16 premiers objectifs de ce type rencontrés)**

On constate là encore que le temps moyen baisse de moitié par rapport à la première exécution et se stabilise en continuant à décroître légèrement.

## 9.5. Conclusion

Le contrôle d'un agent est un point crucial du fait de son autonomie, qui limite au maximum une intervention directe de l'utilisateur dans le fonctionnement et les choix de l'agent. La combinaison choisie et présentée dans ce chapitre d'un contrôle explicite par des règles de gestion des objectifs et d'un contrôle implicite par des émotions indépendantes de la sémantique de l'agent, autorise des raisonnements complexes et planifiés tout en évitant que ces raisonnements ne conduisent l'agent vers des impasses stériles de façon prolongée. Le renforcement dû aux émotions permet de favoriser les concepts utiles en fonction de critères structurels ou environnementaux. En plus de son intérêt pour guider l'agent, ce renforcement permet également d'intégrer les nouveaux concepts appris, en favorisant les concepts utiles, éventuellement au détriment des concepts originaux de l'agent, et en conduisant à l'oubli des concepts inutiles (vis-à-vis des différentes émotions définies).

# Chapitre 10
# Implémentation du modèle d'agent

## 10.1. Présentation du programme

Pour tester notre modèle, un programme d'agent générique a été développé en Java. Le programme comporte actuellement 335 classes différentes..

La représentation des connaissances et le mode de contrôle décentralisé ont permis de concevoir une architecture entièrement modulaire. Des modules, contenant des ensembles de connaissances (règles explicites, implicites ou intégrées, faits, …) déjà structurées, des actions, des perceptions et des émotions peuvent ainsi être définis séparément et ajoutés très simplement. L'Agent de base ne contient que quelques connaissances, aucune action, aucun sens et aucune émotion. Les différents ensembles (permettant le raisonnement, la déduction, la planification, la vision, …) sont ajoutés en fonction des besoins.

Cette modularité va nous permettre de présenter tout d'abord les classes globales : *Agent*, *Connaissance* et l'interface. Puis chaque module sera présenté avec ses caractéristiques.

### 10.1.1. Agent

L'Agent (*Agent.java*) et l'Environnement (*Environnement.java*) sont deux programmes indépendants. Lorsqu'il est créé, l'agent prend un environnement en paramètre pour pouvoir communiquer avec lui.

L'agent dispose de Vecteurs dans lesquels les différentes connaissances (vecteur conn), actions (vecteur actions), perceptions (vecteur sens) et émotions (vecteur emot) sont progressivement ajoutées par les ensembles ou durant l'exécution de l'agent.

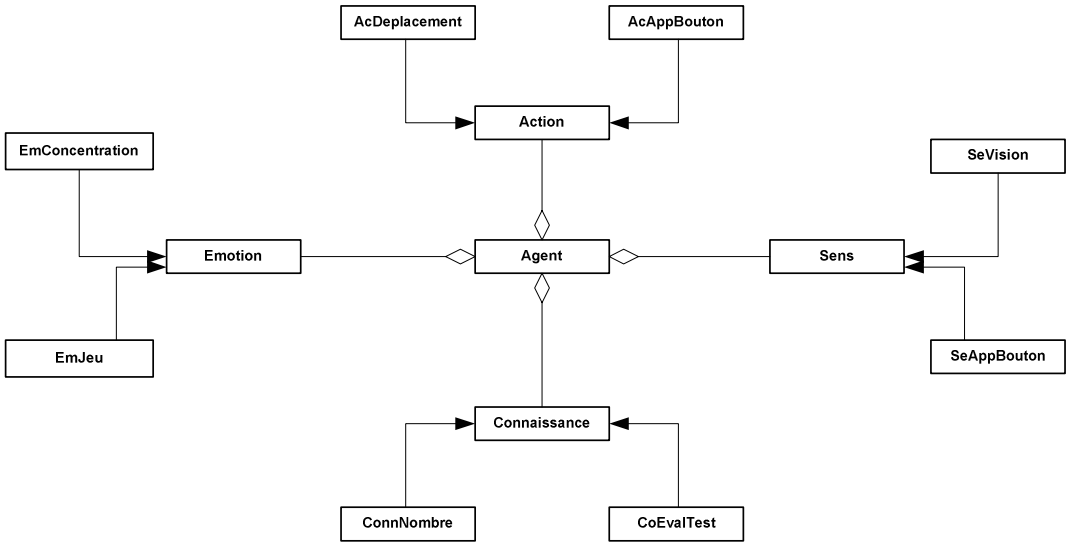



**Figure 10.1 Diagramme des classes de base de l'agent**

Lors de son initialisation, l'agent crée un Timer, un évènement temporel qui le lui permet de réaliser son cycle temporel de façon régulière. A chaque cycle (d'une durée d'environ 50ms dans nos expérimentations), l'agent exécute sa fonction *tpssuiv*() qui lui permet d'exécuter les différentes fonctions du cycle décrites au chapitre 4.7 et résumées figure 10.2.

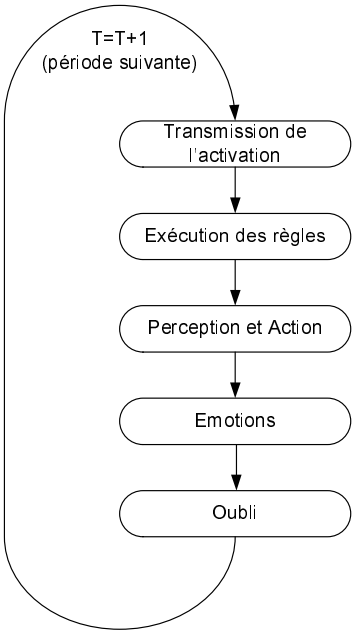



**Figure 10.2 Cycle de fonctionnement de l'agent**

#### 10.1.1.1. Transmission de l'activation

Chaque connaissance dont le degré d'activation dépasse un certain seuil (l'activation maximale actuel/10000) transmet son activation à ses voisines selon la formule décrite en section 5.2. L'agent modifie en réalité une valeur temporaire et toutes les valeurs temporaires sont utilisées simultanément pour déterminer la valeur effective (pour éviter qu'une connaissance transmette l'activation qu'il vient de recevoir).

#### 10.1.1.2. Exécution des règles

A chaque période l'agent choisit aléatoirement une règle avec une probabilité directement proportionnelle à son activation. La règle est ensuite interprétée dynamiquement.

L'interprétation dynamique se fait techniquement de la manière suivante : L'agent choisit parmi les concepts instance de *Regle*, qui sont des *Executable*. Cela signifie qu'ils ont un attribut *RDef* qui est un Interprétable. Par exemple, après avoir sélectionné la règle « *Si une pièce est une barre, alors sa couleur est rouge* », il lit son attribut *RDef* qui pointe vers la connaissance *Règle1* (centre du concept *Règle1* représenté figure 10.3).

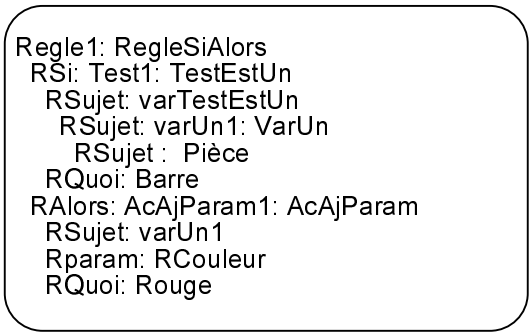



**Figure 10.3 Définition de la règle « Si une pièce est une barre, alors sa couleur est rouge »**

L'agent exécute alors la fonction *litres(Règle1)* qui lit le résultat de l'interprétation d'une connaissance interprétable. Cette fonction recherche la connaissance dont *Règle1* est instance la plus proche et qui contient une fonction *interprete*(). Seules les connaissances ayant une fonction intégrée dispose de cette fonction. *RegleSiAlors* est une connaissance possédant une fonction *interprète*(). Cette fonction consiste à lire les attributs *RSi* et *RAlors* de la connaissance concernée et d'exécuter l'interprétation de la conséquence si le résultat de la condition est une instance de *ResTestEstVrai*.

*RegleSiAlors*.*interprete*(*Règle1*) lit donc l'attribut sujet de *Regle1* (*Test1*) et lit son résultat *litres(Test1).* La fonction *litres*() va à nouveau chercher la connaissance la plus proche

ayant une fonction *interprete*() (*TestEstUn*) et exécuter cette fonction. *TestEstUn.interprete*(*Test1*) va ensuite vérifier si son *RSujet* est une instance de son *RQuoi* et ainsi de suite.

Par exemple, la fonction *interprete*() de *AcAjParam*, qui ajoute une valeur à un attribut est :

```
public void interprete(Connaissance cib,int nores)
  {
    agent.dinftps.debper("ac ConnExpIntAcAjParam");
    super.interprete(cib,nores);
    Connaissance expact=cib;
    Connaissance sujet=expact.litparam(ensexp.cosujet);
    Connaissance param=expact.litparam(ensexp.coparam);
    Connaissance quoi=expact.litparam(ensexp.coquoi);
    Connaissance vsujet=litres(sujet,nores);
    Connaissance vparam=litres(param,nores);
    Connaissance vquoi=litres(quoi,nores);
    if ((vsujet.nonprem()==true)&(vparam.nonprem()==true)&(vquoi.nonprem()==true))
    {
      vsujet.ajparam(vparam,vquoi);
      majres(expact,ensexp.corsituation,ensexp.cositactterm,nores);
    }
    majdesc(nores," le "+desc(param)+" ("+vparam.nom+") de "+desc(sujet)+" ("+vsujet.nom+")
est "+desc(quoi)+" ("+vquoi.nom+") ",expact);
    agent.dinftps.finper("ac ConnExpIntAcAjParam");
  }
```

Les fonctions *debper*() et *finper*() servent à suivre le temps passé par l'agent dans chacun de ses composants. Cela permet entre autre à l'agent de connaître explicitement le temps d'exécution de ses règles et de chacun des éléments de ces règles.

#### 10.1.1.3. Perceptions, actions et émotions

L'agent appelle successivement les fonctions *percoit*(), *sens*() et *agit*() de chaque perception, émotion et action. Pour cette raison, les actions, perceptions et émotions qui sont ajoutées à l'agent ne sont que des classes héritant respectivement de *Action*, *Sens* et *Emotion*. En effet, ces trois classes de base n'ont aucun effet lorsque leurs fonctions sont exécutées. Au contraire de nombreuses connaissances peuvent être du type *Connaissance*, seules les connaissances contenant une règle intégrée sont d'un type héritant de *Connaissance*.

#### 10.1.1.4. Oubli

La fonction d'oubli teste pour une série de connaissances si elles sont encore potentiellement activables en exécutant leur fonction de test et de suppression éventuelle. Une fois rendu en bas de la pile des connaissances, l'agent reprend depuis la dernière connaissance ajoutée et ainsi de suite.

#### 10.1.1.5. Mise à jour de l'affichage

Du fait du grand nombre d'informations affichées à l'écran, la mise à jour n'est effectuée qu'à la fin de chaque cycle.

### 10.1.2. Connaissances

Les connaissances disposent de deux vecteurs de liens : *Lien* contient tous les liens en provenance de la connaissance et *Lieninv* tous les liens arrivant vers la connaissance.

En plus des fonctions de transfert et d'oubli déjà citées, les connaissances disposent de nombreuses fonctions de tests et de recherche dans le graphe de connaissances afin de pouvoir exécuter les règles d'analyse du graphe.

Par exemple :

- *estDansConcept*(*Connaissance co*) permet de savoir si la connaissance actuelle est dans le concept correspondant à la connaissance *co*
- *maxActConnCommPr*(*Connaissance co*) permet de sélectionner de façon proportionnelle à leur activation une connaissance parmi celle qui sont à la fois instances de la connaissance actuelle et de *co*. Par exemple, *Objectif.maxActConnCommPr*(*EnCours*) permet de sélectionner un objectif en cours de façon aléatoire proportionnellement à son activation .

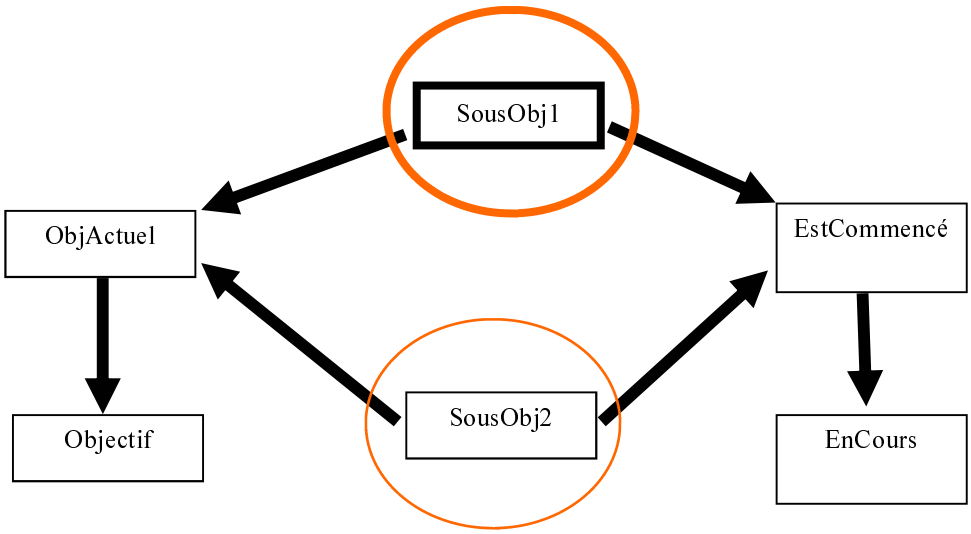



**Figure 10.4 Exemple de fonction de selection probabiliste de connaissance**

### 10.1.3. Interface globale

L'interface de base ne comprend que quelques fenêtres, d'autres venant s'ajouter en fonction des ensembles. L'interface globale est variable en fonction des choix d'affichage de l'utilisateur. On peut par exemple obtenir ceci :

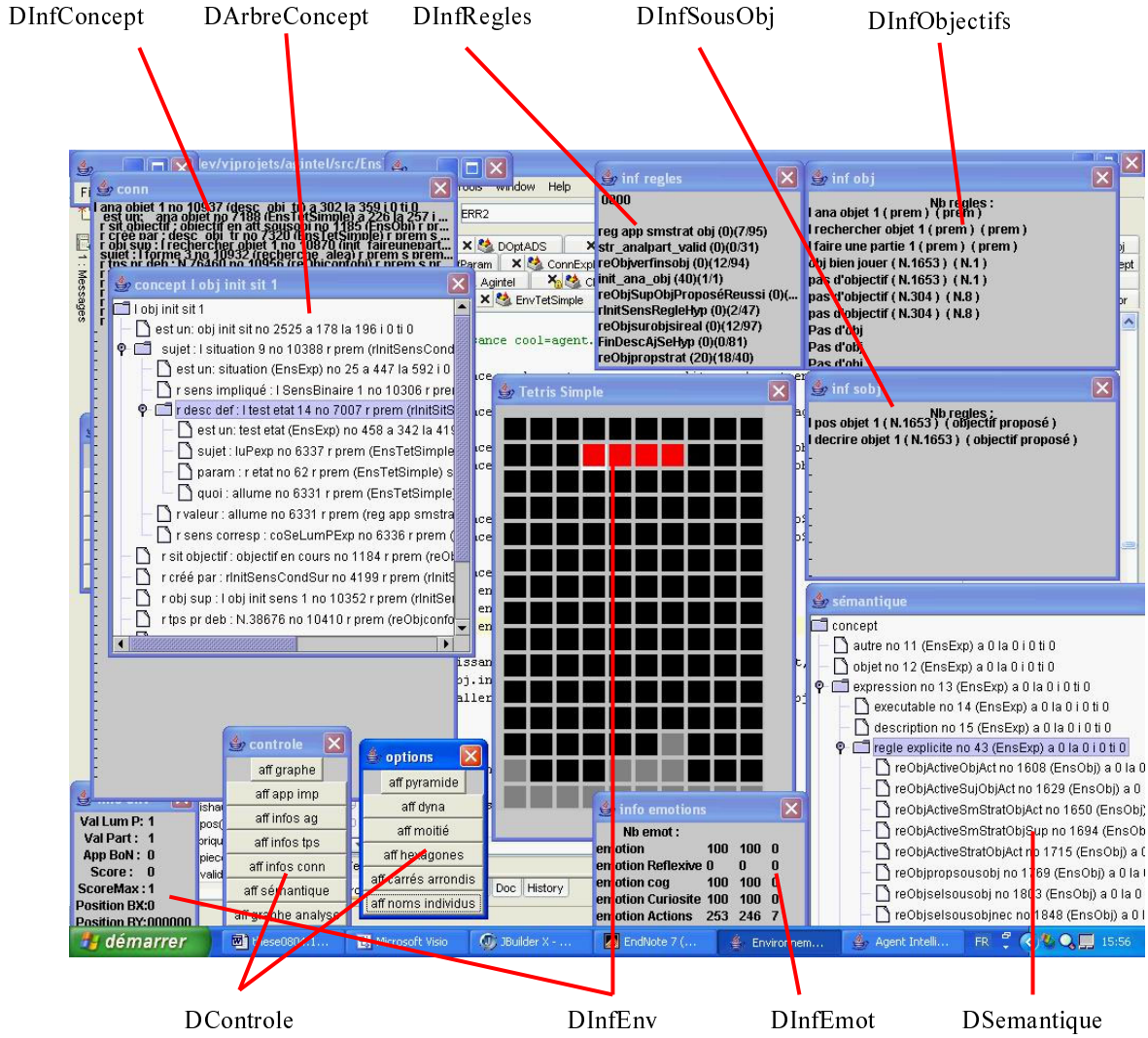



**Figure 10.5 Exemple d'interface globale obtenue**

#### 10.1.3.1. Fenêtre de description globale : *DInfAgent*

Cette fenêtre fournit des informations sur le nombre de connaissances, l'activation totale du graphe et les connaissances les plus activées.

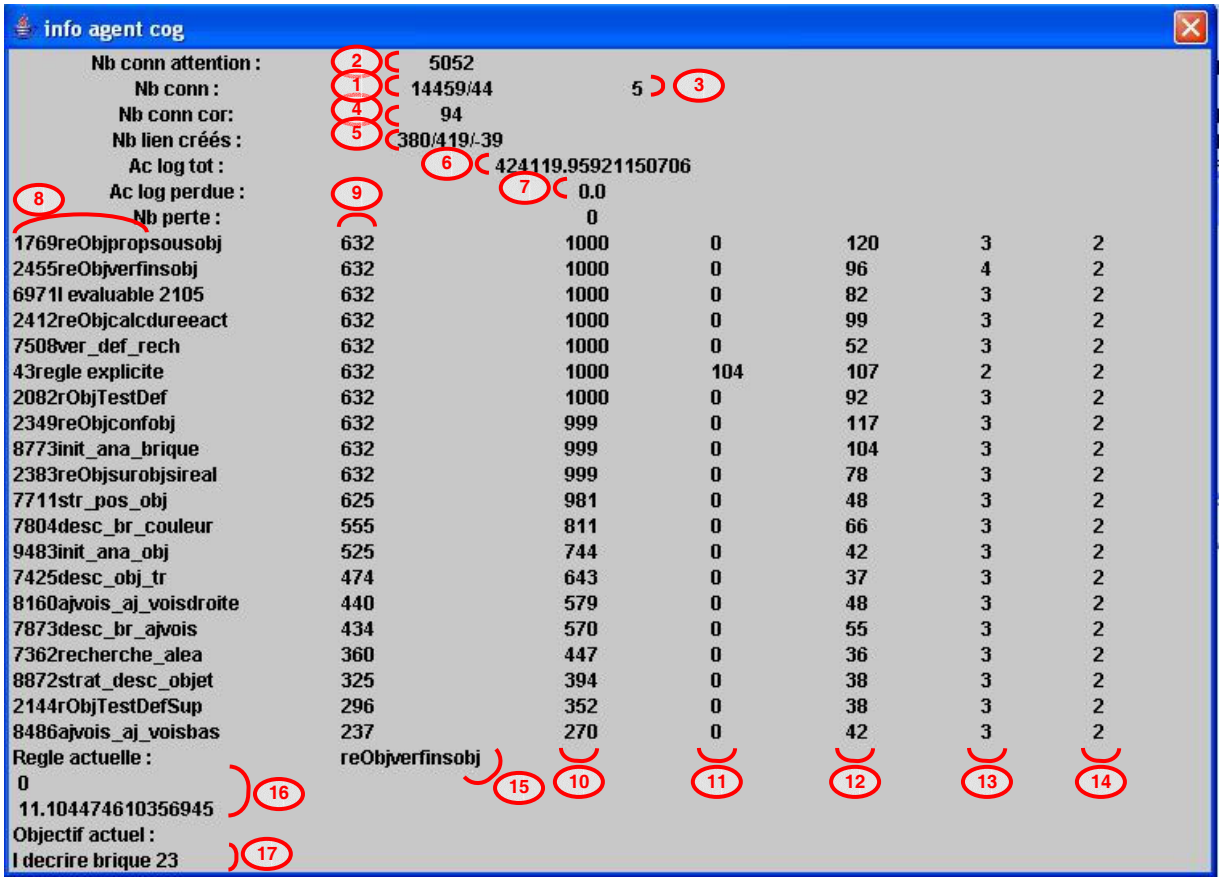



**Figure 10.6 Fenêtre d'information sur les connaissances les plus activées du graphe**

**1 : Nombre de connaissances dans le graphe**

L'indicateur affiche le nombre total de connaissances du graphe et l'évolution de ce nombre depuis le dernier rafraîchissement de la fenêtre (ici tous les 100 cycles de l'agent, donc toutes les 500ms). Le nombre augmente avec l'apprentissage ou le raisonnement, et diminue avec l'oubli.

**2 : Nombre de connaissances dans le champ d'attention de l'agent**

Seule une portion du graphe (les connaissances les plus activées) est utilisée aussi bien pour les choix probabilistes que pour la transmission de l'activation. Le nombre de ces connaissances est gardé à peu près constant (ici environ 5000) afin qu'une augmentation de la taille du graphe n'ait quasiment aucun impact sur le fonctionnement de l'agent. La seule partie du fonctionnement de l'agent dont la complexité est fonction de la taille totale est justement cette sélection des connaissances les plus activées, qui consiste simplement en un test de supériorité à une activation limite. Le reste du fonctionnement est donc indépendant de la taille totale du graphe.

**3 : Seuil limite pour faire partie du champ d'attention de l'agent**

Cette valeur correspond à l'activation (en valeur logarithmique, colonne 10) nécessaire pour qu'une connaissance fasse partie du champ d'attention de l'agent. Ce seuil augmente si le nombre de connaissances dans le champ d'attention de l'agent est supérieur à l'objectif (5000) et diminue s'il est supérieur. Cela permet de maintenir le nombre de connaissances proche de cet objectif tout en limitant l'opération nécessaire pour chaque connaissance du graphe à un simple test de supériorité.

**4 : Nombre de connaissance dans le champ d'attention répondant au critère d'affichage**

La fenêtre permet de réaliser une sélection parmi les connaissances affichées. Ici, par exemple, on affiche uniquement les connaissances instances de *RègleExplicite*. Le chiffre indique le nombre de connaissance répondant à la fois au critère d'affichage (règles explicites) et se trouvant dans le champ d'attention (activation supérieure au seuil).

**5 : Nombre de liens créés et supprimés**

Indique le nombre de liens créés, supprimés et l'évolution totale résultante depuis le dernier rafraîchissement. Les liens sont créés à chaque fois qu'un attribut ou une instanciation est ajoutée ou qu'un renforcement est effectué. Ils sont supprimés par oubli.

**6 : Activation logarithmique totale dans le graphe**

Cet indicateur mesure la somme des activations logarithmiques des connaissances du graphe. On utilise l'activation logarithmique (activation standard passée sur 0 à $+\infty$, voir section 5.2) car elle est définie pour être additive. De plus, les fonctions d'actualisation et de transmission sont définies de telle sorte que l'évolution de ce total puisse être prédit.

**7 : Activation logarithmique perdue**

Un des facteurs imprévisible concernant l'activation logarithmique totale présente dans le graphe est l'activation qui ne peut être transmise car une connaissance ne dispose pas de liens sortants. Dans ce cas, l'activation de la connaissance diminue par actualisation mais aucune autre connaissance ne voit son activation augmentée en contrepartie. L'indicateur mesure donc le total de l'activation perdue à cause des connaissances situées à une extrémité d'une branche du graphe. Le chiffre en dessous indique le nombre de connaissances qui ont contribué à cette perte.

**8 : liste des connaissances les plus activées**

Les vingt connaissances les plus activées du graphe et répondant au critère de sélection choisi (ici les règles explicites) sont affichées avec les informations correspondantes. La première colonne correspond au nom de la connaissance (par exemple reObjpropsousobj) précédé du numéro de la connaissance (1769), numéro unique affecté de façon incrémentale.

**9 : Activation de la connaissance**

Valeur entière du degré d'activation de la connaissance multiplié par 1000.

**10 : Activation logarithmique de la connaissance**

Valeur entière du degré d'activation logarithmique de la connaissance multiplié par 1000 (voir section 5.2 pour le calcul de l'activation logarithmique).

**11 : Nombre d'instances de la connaissances**

Mesure le nombre de connaissances qui sont des instances directes de la connaissance décrite (et non les connaissances qui sont des instances d'une instance). Par exemple, on voit ici qu'il y a 104 instances directes de la connaissance *regle explicite*. Cela ne prend pas en compte toutes les règles qui sont des instances de *RegleSiAlors*, qui est elle-même une instance de *regle explicite.*

**12 : Nombre de liens arrivant à la connaissance**

**13 : Nombre de liens partant de la connaissance**

**14 : Nombre de types de la connaissance**

Mesure le nombre de connaissances dont la connaissance décrite est une instance directe. La plupart des règles décrites ici sont instances de deux connaissances : *regle explicite* et *Evaluable*.

**15 : Dernière règle interprétée**

**16 : Temps moyen d'interprétation des règles (en ms)**

**17 : Objectif actuel**

### 10.1.3.2. Fenêtre d'information sur le temps de calcul : *DInfTps*

Cette fenêtre analyse le temps utilisé dans les différents composants de l'agent. Elle permet ainsi notamment de vérifier que le temps total par cycle n'excède pas le temps théorique.

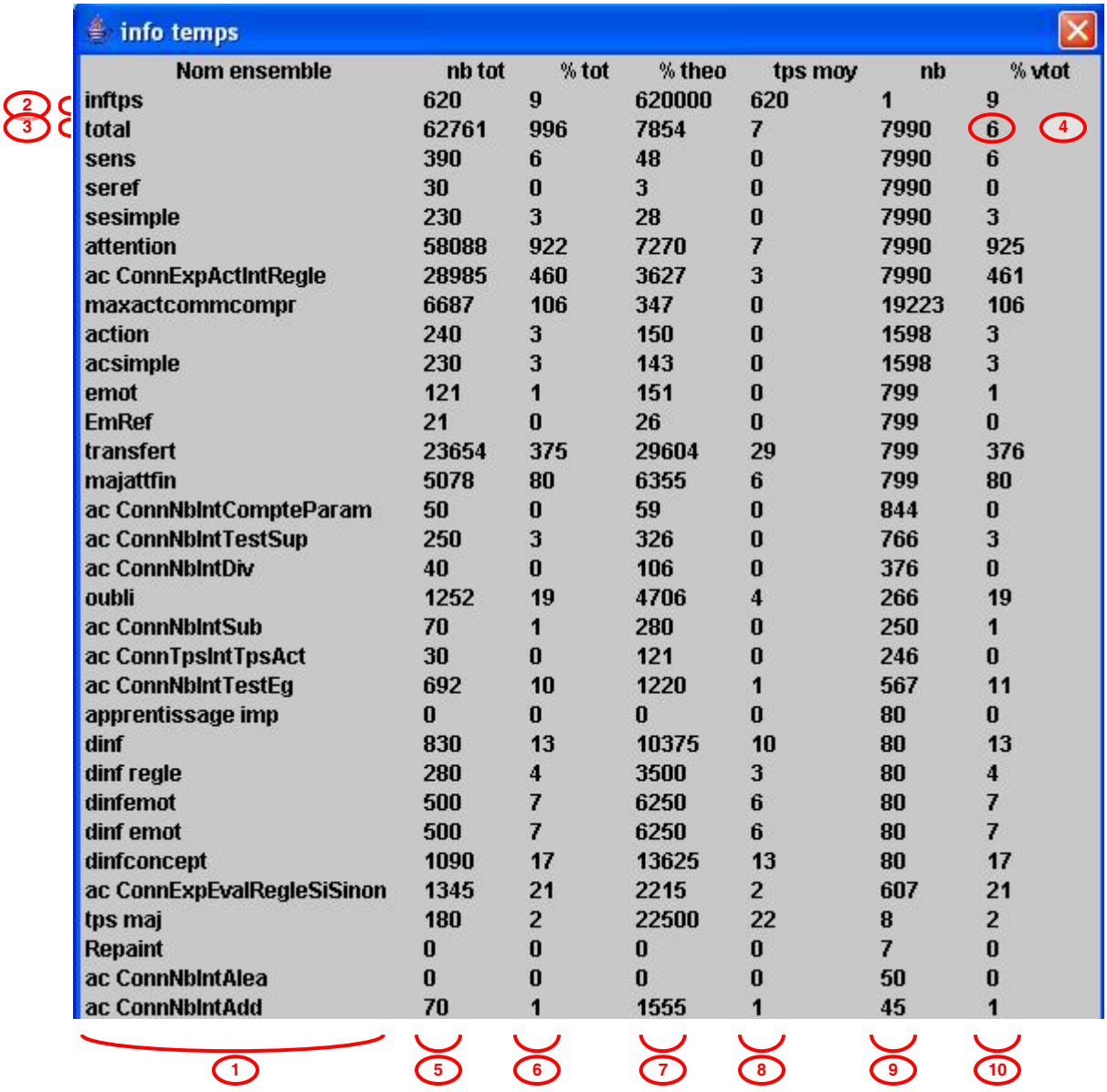



**Figure 10.7 Fenêtre d'information sur le temps passé et le nombre d'exécution de certaines fonctions de l'agent**

**1 : Fonctionnalités de l'agent**

Pour pouvoir analyser le temps passé à exécuter les différentes parties de l'agent, le code a été divisé en sections (non exclusives) correspondant à certaines fonctions de l'agent. Ainsi, on peut distinguer le temps passer pour l'attention (*attention*) de l'agent, qui se subdivise entre

le temps passé pour la transmission de l'activation (*transfert*), le temps passé pour l'interprétation des règles (*ConnExpActIntRegle*) et la mise à jour de l'appartenance des connaissances au champ d'attention de l'agent (*majattfin*).

Les deux premières lignes d'informations, du fait respectivement de leur reflexivité (*inftps*) et de leur globalité (*total*), ne respectent pas les colonnes des lignes suivantes.

**2 : Temps passé à mettre à jour les informations temporelles**

Seule la première et la dernière colonne de cette lignent peuvent s'interpréter de la même façon que les autres lignes. Les autres colonnes ne s'interprètent pas.

**3 : Temps total**

Cette ligne donne une vision global du temps pris par le système. La valeur de la dernière colonne indique le temps moyen par cycle (ici 6ms).

**4 : Temps moyen par cycle**

**5 : Temps total passé pour cette fonctionnalité (en ms)**

**6 : Part du temps passé dans cette fonctionnalité (en millièmes)**

**7 : Sans signification**

**8 : Temps moyen passé pour réaliser la fonctionnalité (en ms)**

**9 : Nombre d'exécution de la fonctionnalité**

**10 : Temps total divisé par le temps passé (en millièmes)**

#### 10.1.3.3. Affichage brut d'une partie du graphe de connaissances : *DAffGraphe*

Cette fenêtre affiche une partie du graphe de connaissances à partir d'une connaissance cible. Le très grand nombre de connaissances du graphe complet et le nombre de connaissances nécessaires pour représenter un concept rend cette vision brute peu utile.

- Plus une connaissance est activée, plus elle est entourée par un trait épais.
- Plus un lien a une intensité (en valeur absolue) élevée, plus il est épais.
- Si l'intensité est positive, le lien est rouge, sinon, il est bleu.
- Plus le lien a une stabilité élevée, plus il est foncé.
- Le chiffre avant le nom de la connaissance désigne son numéro et les deux chiffres après le nombre de types et le nombre d'instances.

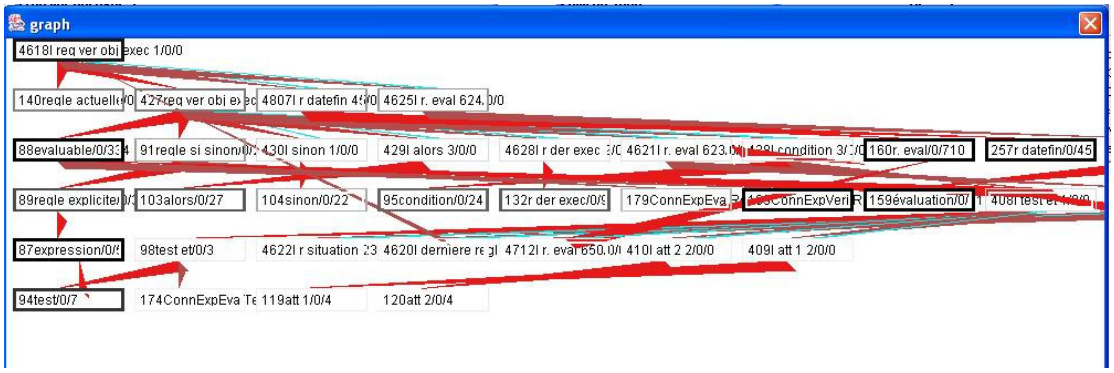



**Figure 10.8 Fenêtre de visualisation d'une partie du graphe de l'agent**

#### 10.1.3.4. Représentation d'un concept sous forme de Frame : *DInfConcept* et *DArbreConcept*

Ces fenêtres affichent le concept correspondant à une connaissance cible sous forme de Frame. *DArbreConcept* permet à l'utilisateur de développer les branches qu'il souhaite afin d'avoir des informations plus précises concernant un attribut du concept initial en particulier.

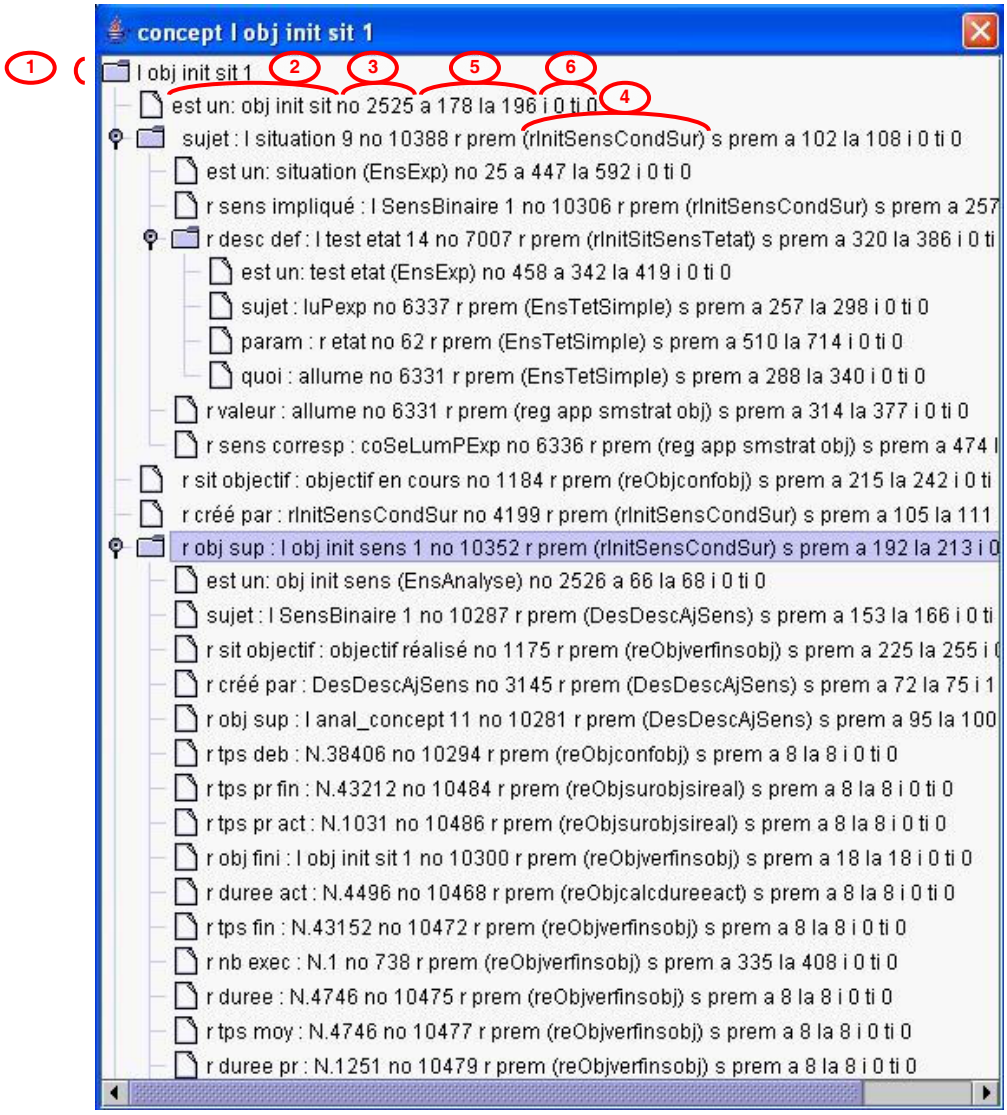



**Figure 10.9 Arbre décrivant un concept du graphe de l'agent**

**1 : Connaissance de départ**

Toute connaissance est au centre d'un concept. L'utilisateur peut donc choisir la connaissance qu'il souhaite (objectif actuel, règle, dernière hypothèse créée, concept créé par induction, …) et afficher le concept correspondant sous forme de Frame. Par exemple, on affiche ici le concept correspondant à l'objectif actuel qui est *I obj init sit 1* (un objectif d'initialisation d'une situation).

**2 : Type d'attribut, « : », Valeur d'attribut**

L'affichage sous forme de Frame permet d'afficher les relations entre les connaissances en les interprétant comme des relations typées, alors qu'on a vu au chapitre 4 que la modélisation n'utilise que des liens non typés et des connaissances intermédiaires pour introduire les types. Par exemple, ici, le sujet (attribut de type *sujet*) de *IObjInitSit1*

(connaissance racine) est *ISituation9*. Les attributs apparaissant décalés sous *ISituation9* sont les attributs d'*ISituation9* (la valeur correspondante, *r_valeur*, le sens impliqué, *r_sens_impliqué*, …). Le lien « est-un » est un type implicite, qui correspond à un lien direct mais d'intensité très élevée (>0,95) entre deux connaissances. *IObjInitSit1* est un *ObjInitSit*, *ISituation9* est un *Situation*, …

**3 : Numéro de la connaissance cible**

La connaissance cible est la connaissance correspondant à la valeur de l'attribut affiché (par exemple, *ISituation9* pour la ligne 3, donc l'attribut *sujet* de *IObjInitSit1*)

**4 : Origine de la connaissance intermédiaire (donc de l'attribution de la valeur)**

Le nom entre parenthèses désigne l'ensemble ou la règle qui a créé la connaissance intermédiaire correspondant à la relation. C'est-â-dire l'ensemble ou la règle qui a assigné cette valeur à cet attribut. Si c'est un ensemble, cela signifie que cette attribution a été réalisée à la création de l'agent lors de l'initialisation de cet ensemble (par exemple, le fait que le *sujet* de *ITestEtat14* soit *luPexp* (ligne 8) a été fixé dès la création de l'agent par l'ensemble *EnsTetSimple*). La plupart des relations sont créées par l'agent lui-même, et donc par une règle. Par exemple, le sujet de *IObjInitSit1* a été attribué lors de la création de l'objectif par la règle *rInitSensCondSur*.

**5 : Activation (précédée de « a ») et activation logarithmique (précédée de « la ») de la connaissance cible**

**6 : Nombre d'interprétations totale (précédé de « ti ») et nombre d'interprétations depuis le dernier rafraîchissement (précédé de « i ») de la connaissance cible**

Certaines connaissances, en particulier les règles et les méthodes, peuvent être interprétées par l'agent pour être exécutées. Ces compteurs permettent de savoir si une règle a été utilisée, et en particulier si elle a été utilisée récemment.

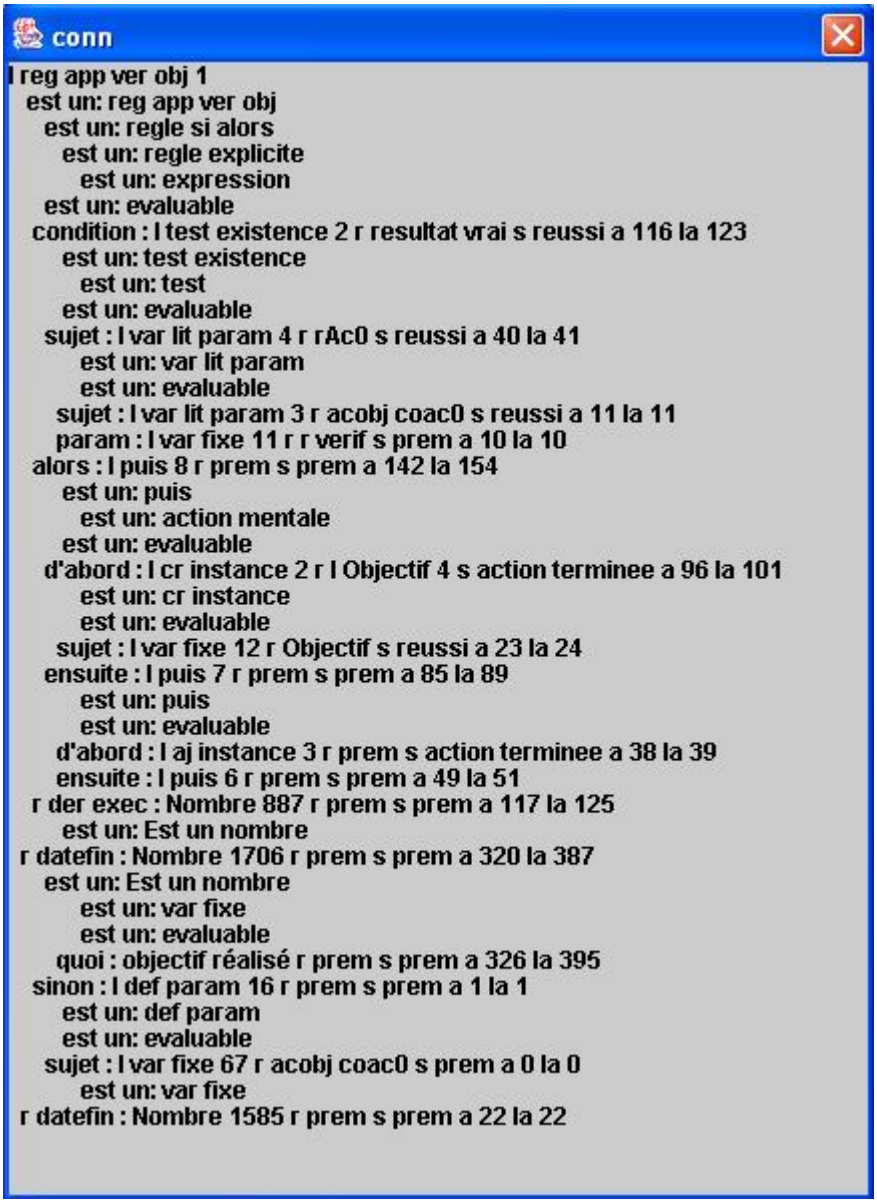



**Figure 10.10 Fenêtre de description d'un concept du graphe de l'agent**

Le fonctionnement de la fenêtre est le même que celui de l'arbre sans que l'utilisateur puisse intervenir pour développer certaines branches.

#### 10.1.3.5. Affichage de la sémantique actuelle de l'agent : *DSémantique*

Cette fenêtre permet de visualiser la sémantique actuelle du système en partant du concept le plus général (Concept)

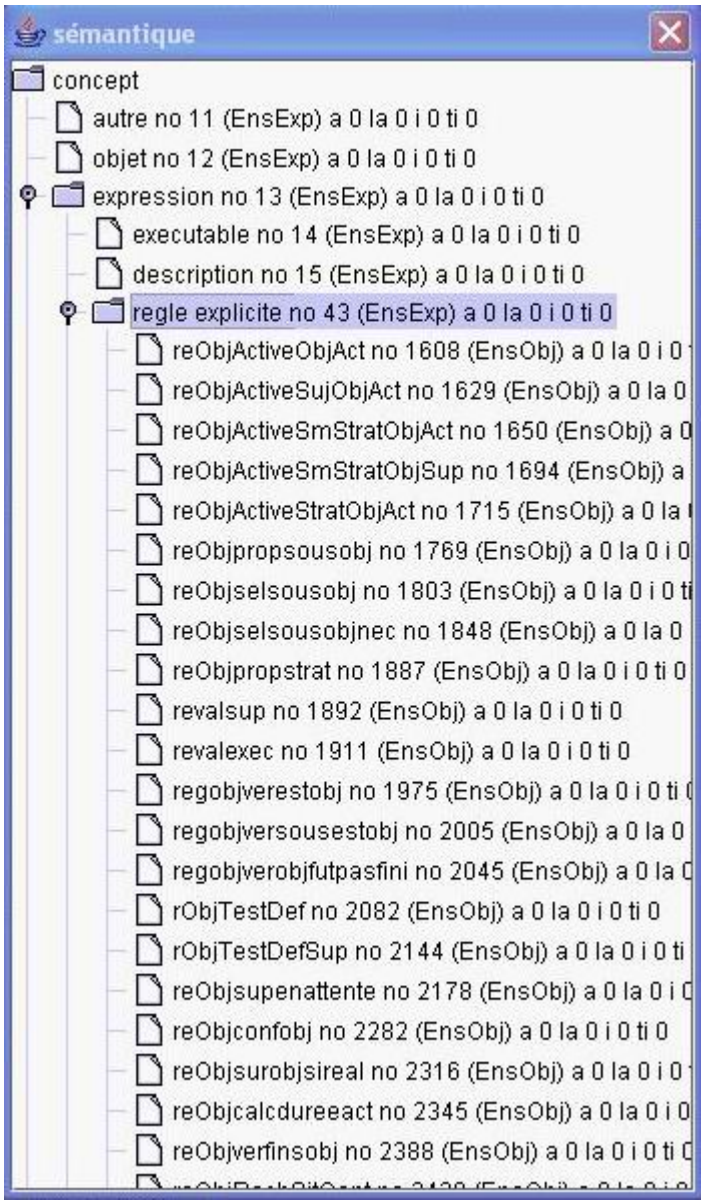



**Figure 10.11 Arbre décrivant la sémantique actuelle de l'agent**

Chaque ligne contient :

- le nom de la connaissance
- son numéro (précédé de no)
- L'ensemble ou la règle qui l'a créé (entre parenthèses)
- Son activation (précédée de « a »)

- Son activation logarithmique (précédée de « la »)
- Le nombre de fois où elle a été interprétée depuis le dernier rafraîchissement (précédé de « i »)
- Le nombre total de fois où elle a été interprétée (précédé de « ti »)

#### 10.1.3.6. Visualisation d'une pyramide : *DPyr*

Cette fenêtre permet de visualiser certains concepts utilisés sous forme de pyramide symbolique. Cette pyramide peut être obtenue suite à une induction (voir Chapitre 6 ) ou pour aider l'utilisateur à visualiser l'évolution des concepts utilisés (voir Chapitre 7 ).

Les paliers entourés de rouge désignent les paliers sélectionnés soient pour l'induction, soit pour l'affichage dynamique.

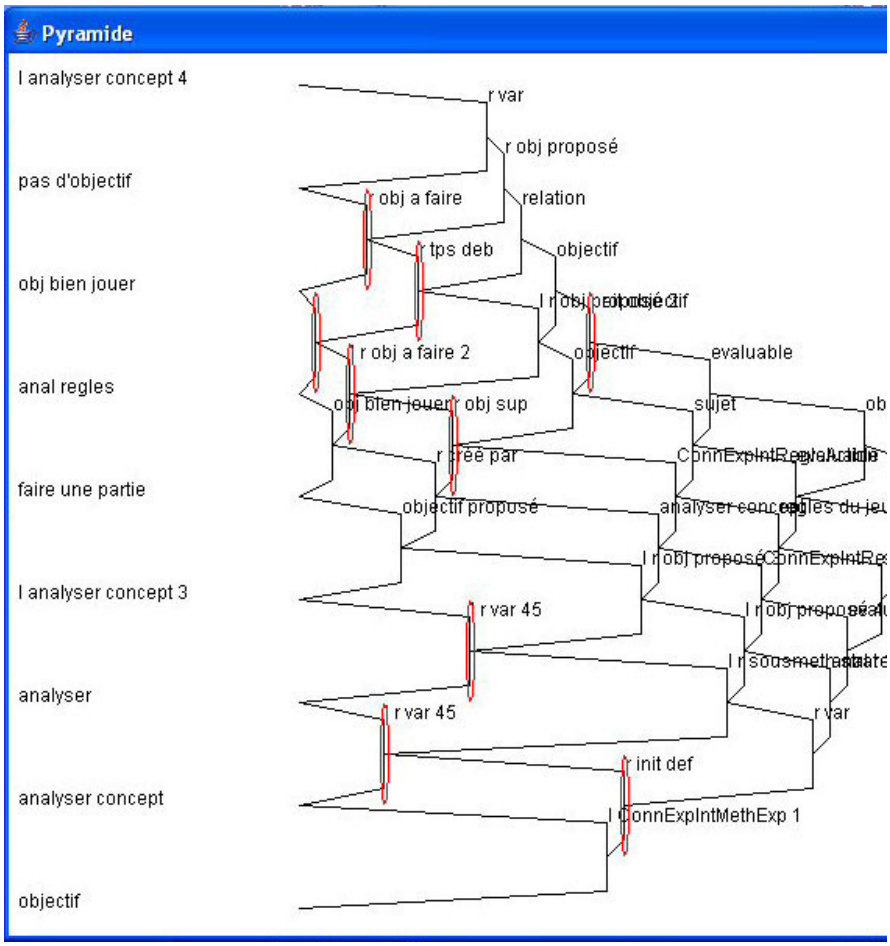



**Figure 10.12 Fenêtre de visualisation d'une pyramide générée à partir d'une analyse des concepts de l'agent**

#### 10.1.3.7. Visualisation de l'évolution des concepts : *DAffADS*

Cette fenêtre permet de visualiser l'évolution des OS sélectionnés. Voir le Chapitre 7 pour plus de détails sur ses composants.

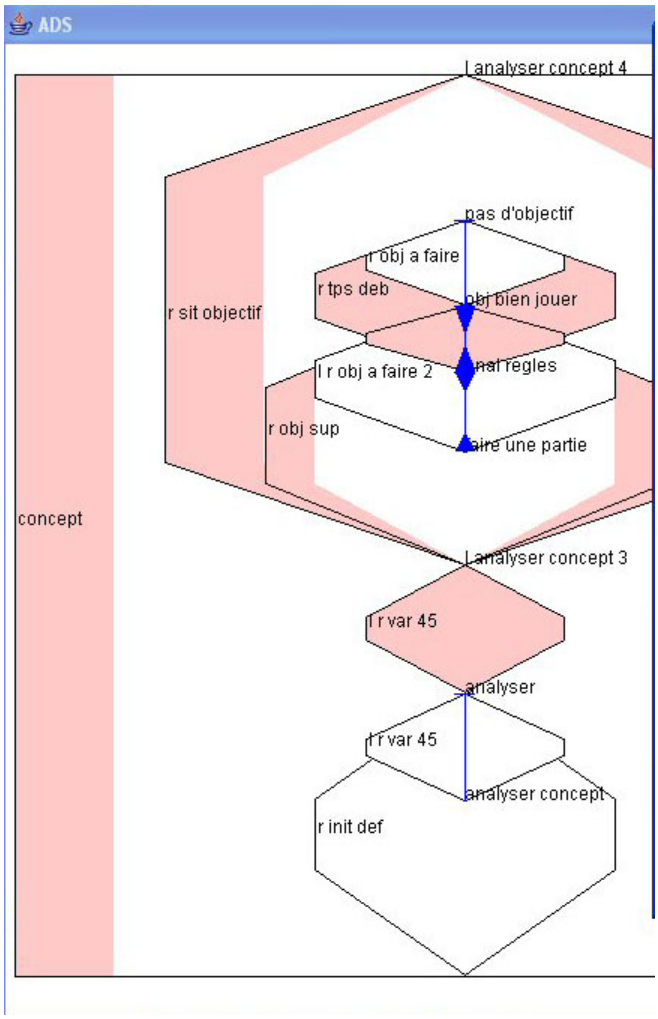



**Figure 10.13 Fenêtre de visualisation dynamique de concepts utilisés par l'agent**

#### 10.1.3.8. Affichage des dernières règles exécutées : *DInfRegles*

Cette fenêtre donne la liste des dix dernières règles interprétées.


**Figure 10.15 Fenêtre d'information sur les dernières règles interprétées par l'agent**

Certaines règles sont créées par l'agent, c'est pourquoi elles ont un nom standard très peu explicite (comme *Ievaluable2084* ou *IRegleExplicite105*).

Pour chaque règle, la fenêtre affiche également le temps d'exécution de la règle (premières parenthèses), le nombre d'exécutions réussies et le nombre d'exécutions manquées (parenthèses suivantes).

#### 10.1.3.9. Informations sur la situation des émotions : *DInfEmotions*

Cette fenêtre indique l'état des différentes émotions : niveau actuel, moyenne mobile et variation.


**Figure 10.16 Fenêtre d'information sur le niveau actuel des émotions**

#### 10.1.3.10. Information sur l'objectif actuel : *DInfObjectifs*

Cette fenêtre indique la situation de l'objectif en cours et de ses supérieurs. Par exemple, ici, l'objectif actuel est *IAnalyserConcept3*. Cet objectif est un sous-objectif de *AnalRègle*, lui-même un sous-objectif de *ObjBienJouer*, qui est un sous-objectif de la racine *PasDObjectif*.

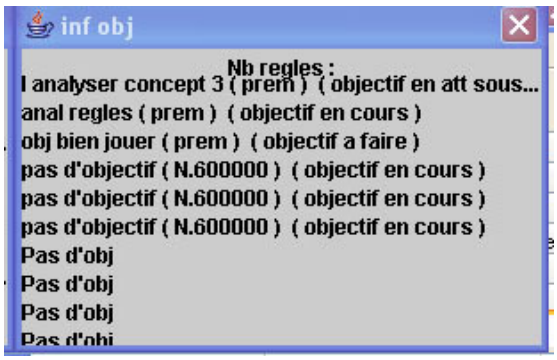



**Figure 10.17 Fenêtre d'information sur l'objectif actuel**

#### 10.1.3.11. Informations sur les sous-objectifs de l'objectif actuel : *DInfSousObjectifs*

Cette fenêtre indique les sous-objectifs de l'objectif en cours et leur situation.

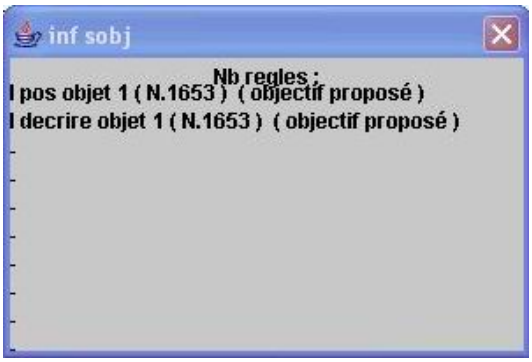



**Figure 10.18 Fenêtre d'information sur l'objectif actuel et ses sous-objectifs**

10.1.3.12. Informations concernant l'environnement actuel : *DInfEnv*

Cette fenêtre fournit des informations sur l'environnement. Elle dépend donc de l'application en cours. Ici, il s'agit de l'état du jeu.

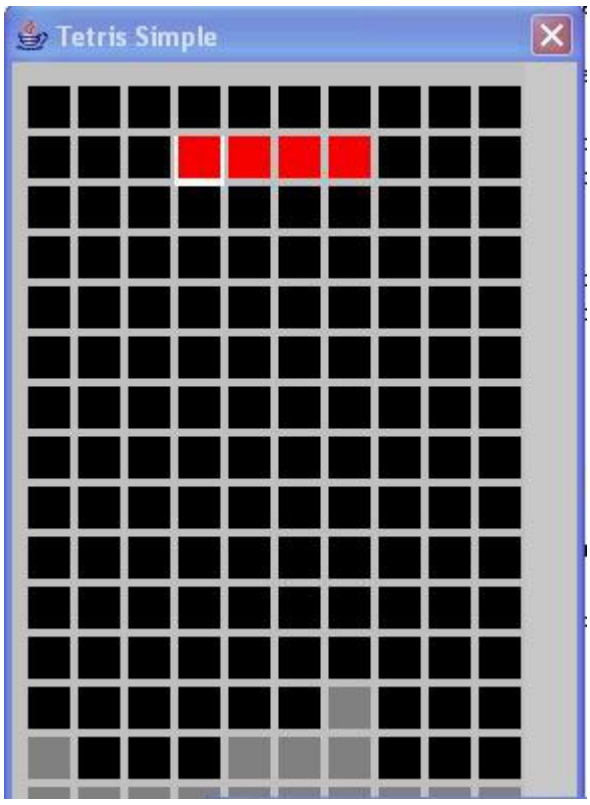



**Figure 10.14 Fenêtre d'information sur l'environnement actuel**

### 10.1.4. Ensembles

En plus de ces composants de base, plusieurs ensembles de composants sont ajoutés pour que l'agent ait un comportement intéressant.

L'agent *AgCog*, qui hérite d'*Agent*, ajoute les principaux ensembles utiles à un agent cognitif répondant à l'objectif du modèle : notions de base (*EnsCog*), raisonnement implicite (*EnsImp*), raisonnement explicite (*EnsExp*), planification (*EnsObj*), gestion des nombres (*EnsNombres*), gestion du temps (*EnsTps*).

#### 10.1.4.1. Introspection et réflexivité (*EnsRef*)

Cet ensemble contient principalement le sens qui donne la perception des modifications dans le graphe de l'agent. Ce sens crée à chaque période un rapport sur les modifications du graphes qui ont été observées à la période précédente (il observe les connaissances définies par les règles gérant l'attention).

#### 10.1.4.2. Raisonnement explicite (*EnsExp*)

Cet ensemble contient les concepts liés à la gestion et l'interprétation des règles explicites par l'agent. Il crée la sémantique de base et lie les règles d'interprétation de règles aux concepts correspondants. Il fournit de plus des fonctions permettant aux autres ensembles de créer plus facilement des règles explicites.

#### 10.1.4.3. Gestion des objectifs (*EnsObj*)

L'objectif de cet ensemble est de gérer le suivi des objectifs. Par exemple la règle « Si *l'objectif actuel a un sous objectif non réalisé, alors le proposer* » est définie de la façon suivante dans *EnsObj* :

```
public void initrepropsousobj()
{
  // si l'obj act o1 a un rsousobj non réalisé o2 alors prop o2

  Connaissance coo1=agent.ensexp.nouvvarlitparam(
      agent.ensexp.nouvvarfixe(coobjact),
      agent.ensexp.nouvvarfixe(agent.ensexp.corest));

  Connaissance coo2=agent.ensexp.nouvvarlitparam(
      coo1,
      agent.ensexp.nouvvarfixe(corsousobj));

  Connaissance cocond=agent.ensexp.nouvtestet(
     agent.ensexp.nouvtestex(coo2),
     agent.ensexp.nouvtestfaux(
            agent.ensexp.nouvtestexistevalparam(coo2,
                   agent.ensexp.nouvvarfixe(corsitobj),
                   agent.ensexp.nouvvarfixe(cositobjatteind))));

  Connaissance coalors=(
```

```
        nouvpropobj(coo2,coo1)
        );
    Connaissance nregle=agent.ensexp.nouvreglesi(cocond,coalors,"reObjpropsousobj");
    coensobj.ajlien(nregle,.4,.8);
}
```

*nouvpropobj(coo2,coo1)* renvoie à une méthode définie explicitement dans *EnsObj* et ajoutant l'objectif *coo2* dans les sous-objectifs proposés de *coo1*.

#### 10.1.4.4. Analyse de règles (*EnsAnalyse*)

Cet ensemble fournit les règles d'analyse de concepts. En particulier les règles de construction et de déduction d'hypothèses, de situations, de perceptions et d'actions.

#### 10.1.4.5. Cognition (*EnsCog*)

Cet ensemble fournit à l'agent des règles implicites sur des concepts de base nécessaires pour faire des raisonnements explicites intéressants. En particulier les règles de traitement des ensembles.

#### 10.1.4.6. Gestion des nombres (*EnsNombre*)

Cet ensemble permet à l'agent de connaître la notion de nombre, de pouvoir comparer des nombres entre eux de réaliser de façon intégrée des opérations simples.

#### 10.1.4.7. Gestion du temps (*EnsTps*)

Cet ensemble contient des connaissances et des règles liées aux notions temporelles telles que le début et la fin (particulièrement utile pour la notion de *Partie* dans un jeu qui est une instance de *Durée*).

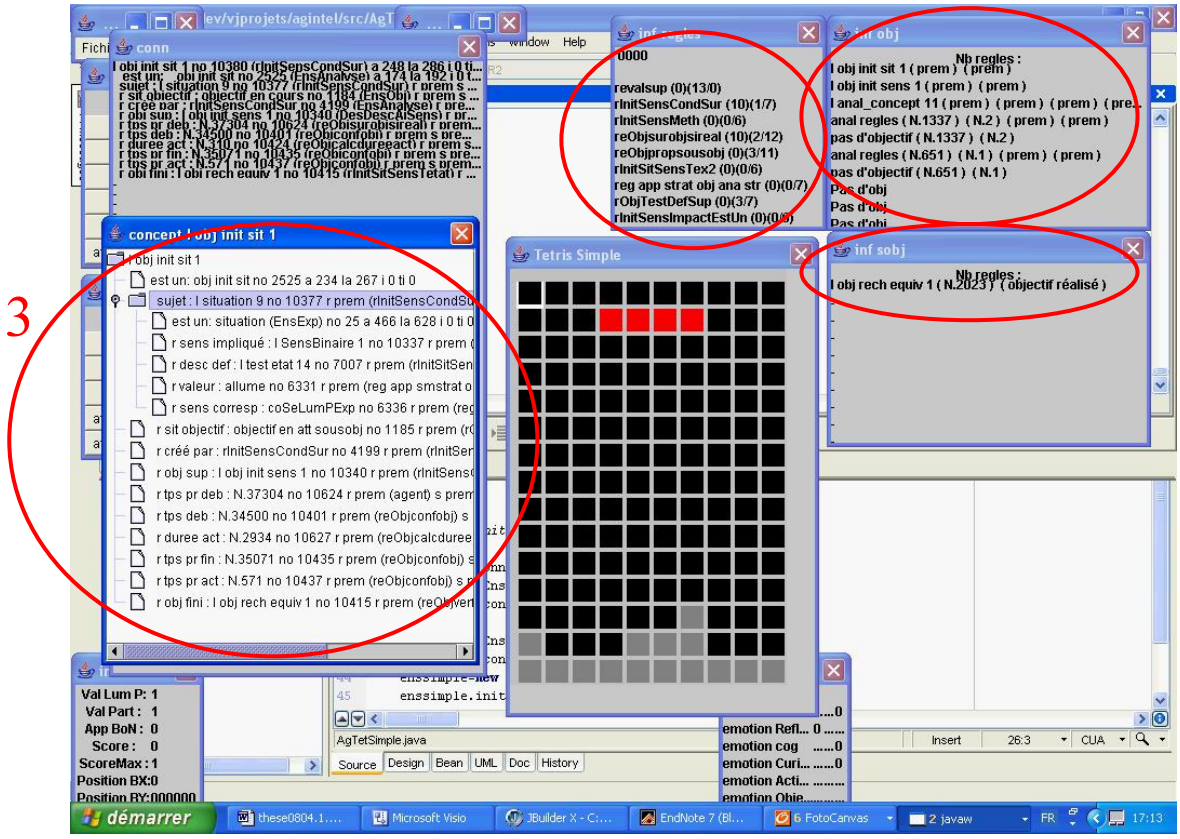
## 10.2. Exemple d'exécution

L'objectif du modèle est de proposer un agent capable de s'adapter à son environnement en découvrant et utilisant de façon efficace de nouveaux concepts. Pour cela, il utilise trois types d'apprentissage complémentaires sur une représentation des connaissances homogène. Chacun de ces éléments (représentation des connaissances – raisonnement – induction-déduction – émotion) a dû être adapté pour fonctionner sur une représentation des connaissances homogène et de façon complémentaire.

La cohérence et le bon fonctionnement de l'ensemble sont illustrés au travers d'un exemple de fonctionnement de l'application en plusieurs étapes.

Le bon fonctionnement individuel de chaque module et certains exemples de résultats obtenus sont donnés dans les différents chapitres décrivant chaque module.

### 10.2.1. Initialisation



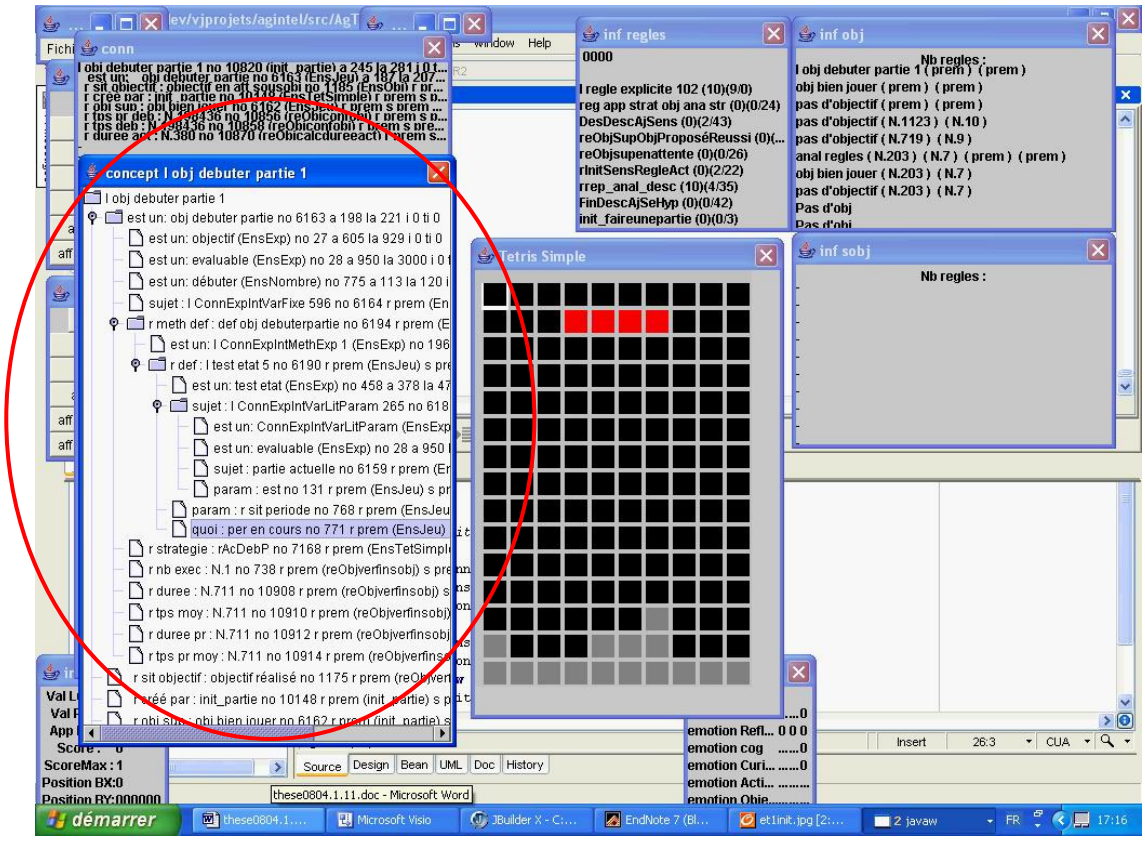

1. L'agent commence par analyser les informations – les règles du jeu – qui lui sont données sous forme de descriptions. L'ordre des objectifs apparaît dans la fenêtre *info_obj* (voir description de la fenêtre section 10.1.3.10). L'objectif actuel, *IObjInitSit1*, qui a pour objectif supérieur *IObjInitSens1*, et ainsi de suite jusqu'à *anal_règle* qui a pour objectif supérieur l'objectif racine *pas_d'objectif*.

2. La fenêtre *inf_sobj* nous permet de voir que l'objectif actuel a un sous-objectif *IObjREchEquiv1* qui est réalisé (voir description de la fenêtre section 10.1.3.11).

3. Afficher les détails d'un concept, en l'occurrence l'objectif actuel *IObjInitSit1*, nous permet de voir que l'agent initialise la *Situation9* qui correspond à l'état allumé de la lampe (voir description de la fenêtre section 10.1.3.4).

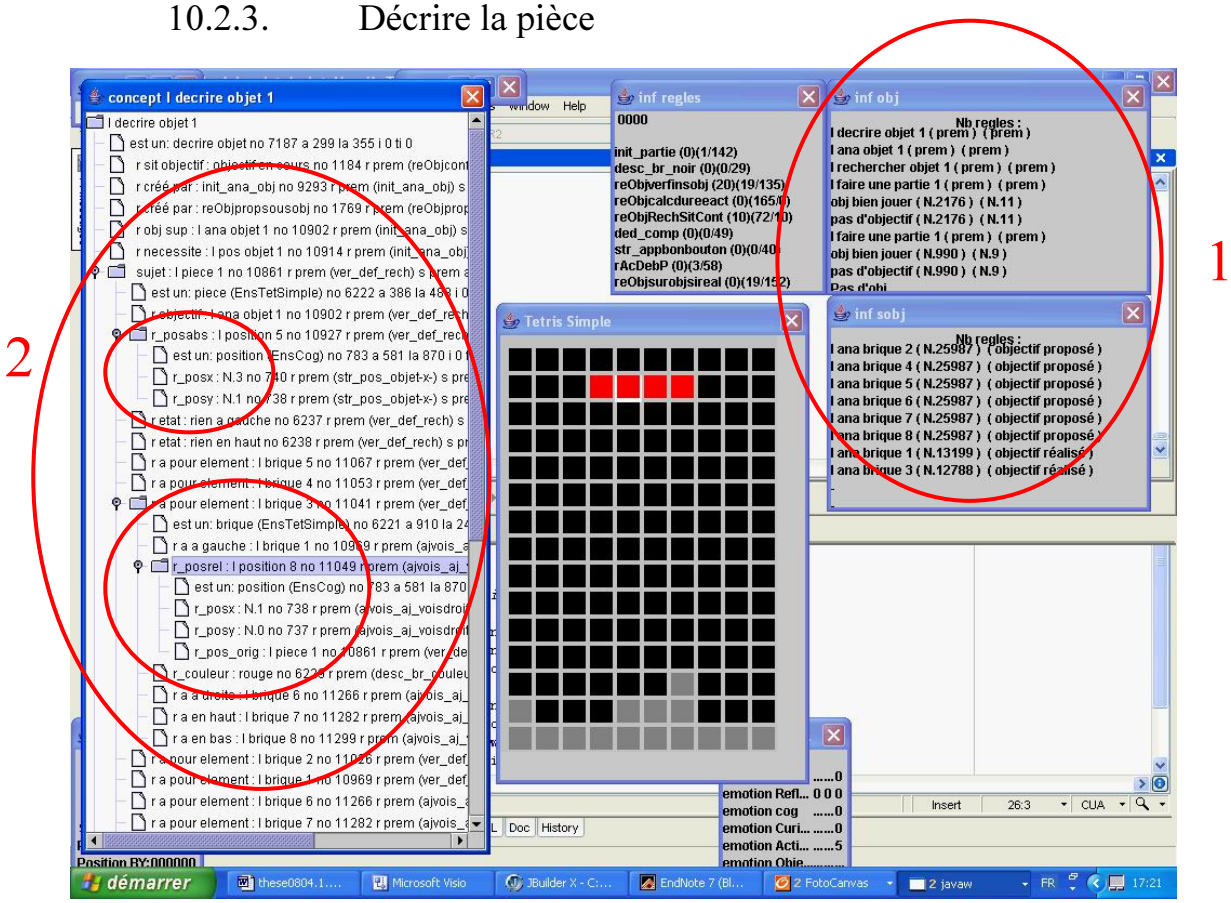
4. Le fonctionnement global de l'agent se fait de façon implicite par transmission d'activation dans le graphe et de façon explicite par interprétation successive d'un grand nombre de règles. La fenêtre *inf_regle* indique les dix dernières règles appliquées, leur temps d'exécution (en ms), le nombre d'exécutions réussies et le nombre d'exécutions manquées.

### 10.2.2. Débuter une partie




1. Une fois les règles analysées, l'agent cherche à débuter une partie. La plupart des objectifs ont une méthode permettant de déterminer s'ils sont atteints. Ici l'objectif *IObjDébuterPartie1* est une instance de *ObjDébuterPartie* et récupère ainsi tous ses attributs. En particulier, sa méthode-définition qui indique que l'objectif est atteint si la partie actuelle est en cours.

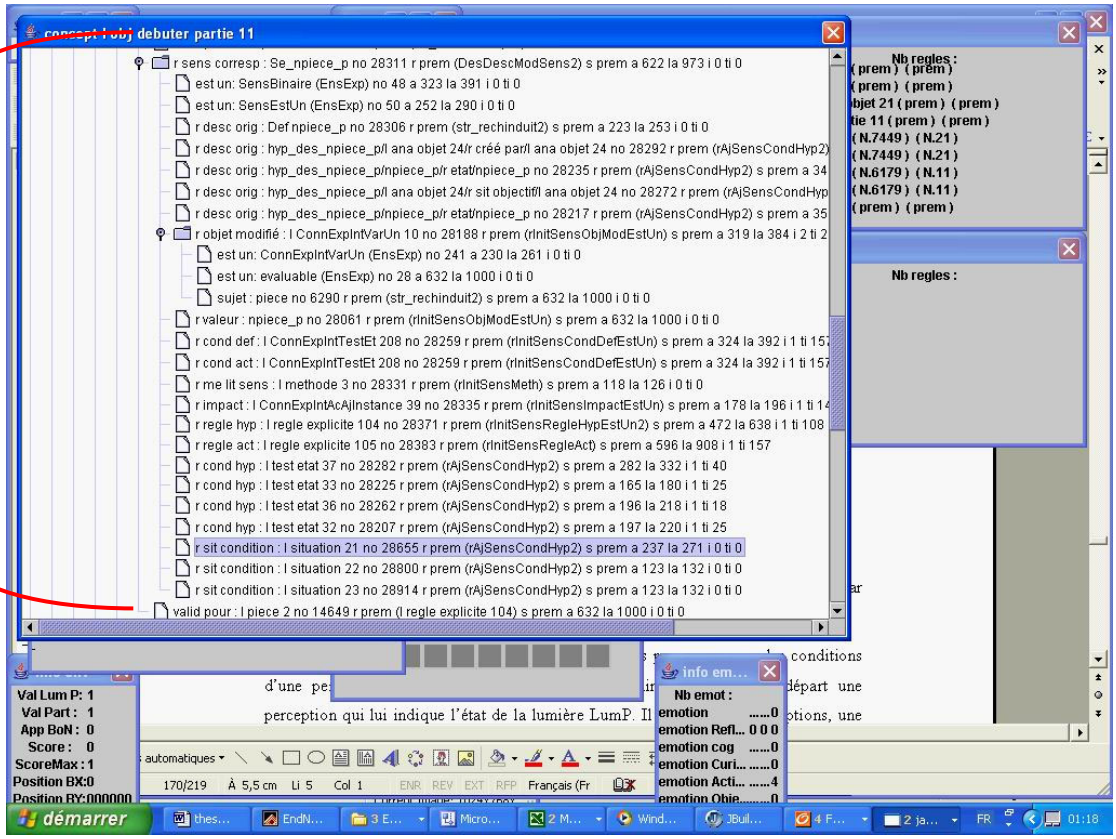
### 10.2.3. Décrire la pièce




1. L'agent n'ayant pas encore identifié les différentes pièces, doit d'abord rechercher la pièce de façon aléatoire dans l'environnement puis décrire de façon systématique l'ensemble de ses éléments.

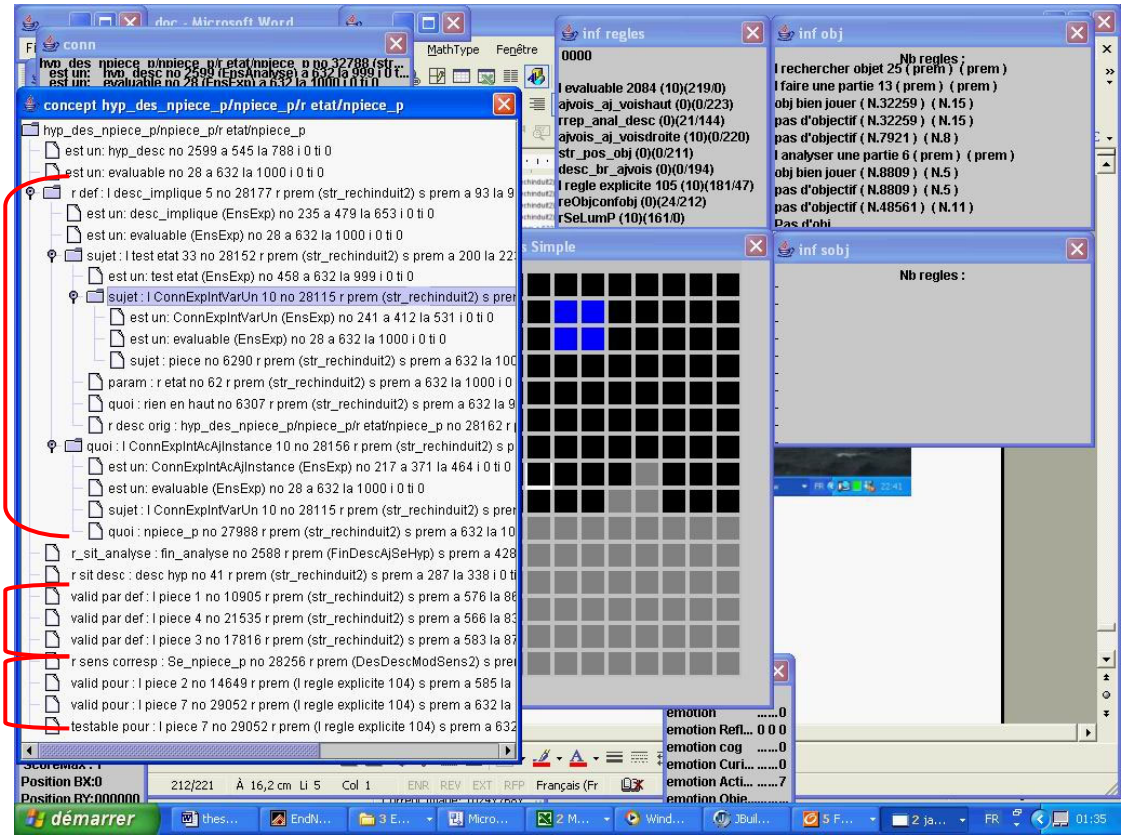
2. Une *Forme* (*Pièce*, *Mur*) a une position absolue (*r_posabs*) dans l'environnement (ici 3,1) et des éléments éventuels qui sont d'autres *Formes* (*Briques*) ayant une position relative (1,0) par rapport au contenant (*IPièce1*).

### 10.2.4. Déduire une nouvelle perception




1. Après chaque partie (trouver pièce – en déduire son complémentaire – trouver le mur et une place correspondant au complémentaire – en déduire la position adaptée pour finir la partie), l'agent analyse un concept et en déduit un nouveau. Par exemple, à partir de l'analyse du concept *Pièce*, il déduit le concept de « *Pièce Observée* » correspondant à *npièce_p*. La description détaillée de l'extraction est donnée en section 6.7 et la perception déduite en section 8.5.

### 10.2.5. Valider les hypothèses

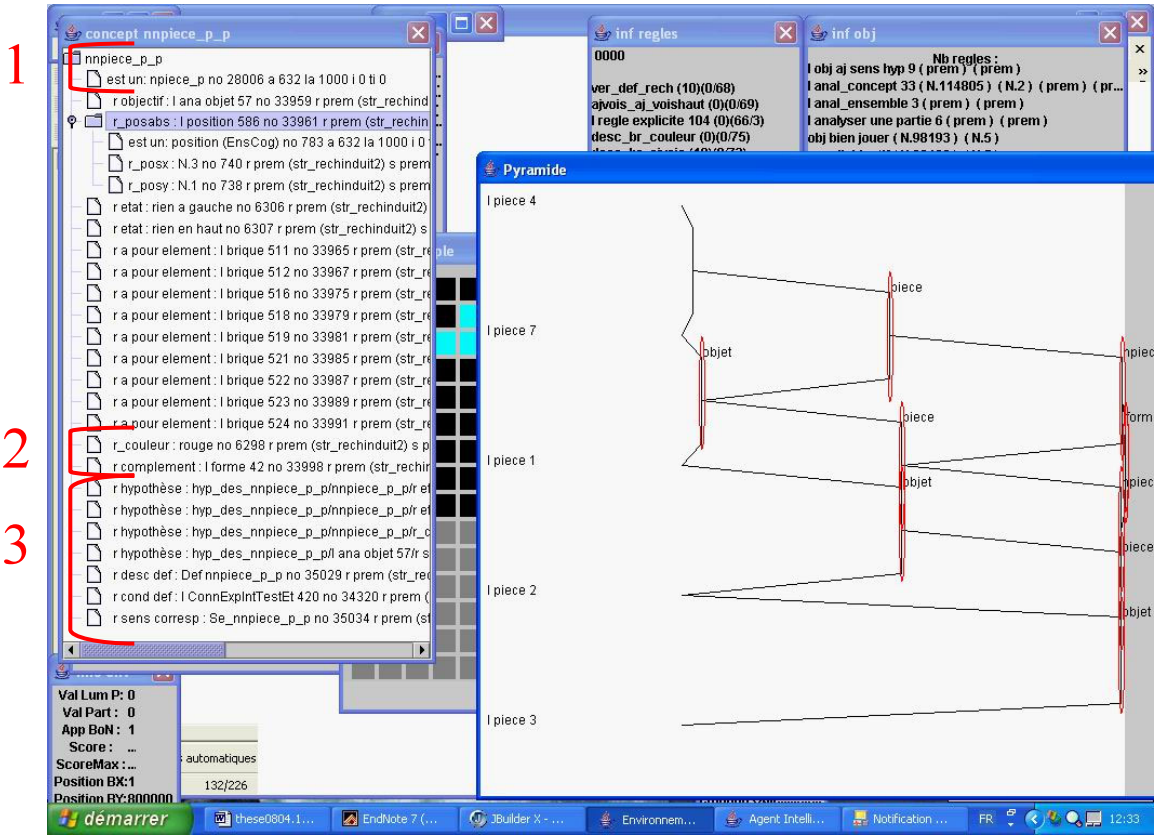



1. L'agent construit également des hypothèses qu'il va pouvoir valider. Par exemple ici, l'hypothèse « *Si une Pièce n'a rien à gauche, alors c'est une PièceObservée* » qui est une hypothèse pertinente étant donné que le fait de n'avoir rien à gauche est propre aux *PiècesObservées* par opposition au type plus général de *Pièce*.

2. L'hypothèse est valide par définition sur les individus qui ont participé à la création du concept. C'est-à-dire qu'il est inutile pour l'agent de tester cette hypothèse sur eux, il faut de nouveaux individus pour que l'agent puisse savoir s'il peut confirmer l'hypothèse ou non.

3. L'agent teste l'hypothèse sur les nouveaux individus observés. Lorsqu'il obtient un nombre et un taux suffisant de tests valides, il peut confirmer l'hypothèse.

### 10.2.6. Utiliser les concepts appris pour en créer de nouveaux




1. L'agent peut utiliser ses nouveaux concepts pour raisonner. En particulier il peut les analyser pour extraire d'autres concepts. Ainsi, à partir du concept de *PièceObservée* (*npiece_p*), il extrait le concept de *Barre* (*nnpiece_p_p*). Une *Barre* est une *PièceObservée* qui est elle-même une *Pièce*. Toute *Barre* récupère donc automatiquement tous les attributs de ces deux concepts père (règles associées, …).
2. La *Barre* a ses attributs propres, comme sa couleur (*Rouge*) ou sa position absolue (toujours 3,1).
3. L'agent en déduit des hypothèses, définitions et perceptions adaptées, de façon similaire au cas précédent (perception du concept de *PièceObservée*)

## 10.3. Conclusion

Pour illustrer et prouver l'applicabilité de notre modèle, nous avons présenté dans ce chapitre un agent correspondant au modèle présenté précédemment que nous avons développé. Cette application nous a permis de vérifier le bon fonctionnement et la cohérence du modèle proposé. La constante évolution de l'agent et en particulier de sa sémantique a nécessité le développement d'une interface particulièrement complexe. Cette interface nous permet

d'explorer les différents concepts utilisés par l'agent et leurs relations, ainsi que de suivre l'état actuel de son raisonnement et de ses émotions.

L'application de l'agent sur un exemple simple d'application nous a permis de vérifier qu'il s'adaptait bien progressivement au domaine et que ses nouveaux concepts étaient bien intégrés et utilisés, en particulier pour les analyses ultérieures.

L'architecture de l'agent est facilement adaptable à une autre application. Les composants (perceptions – actions – émotions – connaissances) de l'agent sont en effets regroupés sous forme d'ensembles fonctionnels (Raisonnement – Planification – Analyse – Nombre - …) permettant une grande modularité et une intégration très simple de nouveaux ensembles. En particulier, l'ensemble spécifique à l'exemple présenté peut être aisément remplacé, les autres composants étant directement réutilisables pour une autre application.

# Chapitre 11
# Conclusion

## 11.1. Contributions

### 11.1.1. Une combinaison de trois types d'apprentissage sur une représentation homogène des connaissances

L'objectif de ce travail était de proposer un modèle d'agent qui puisse s'adapter à son environnement. Pour cela, il devait être capable de créer et d'utiliser efficacement de nouveaux concepts. La découverte de ces nouveaux concepts est réalisée grâce à un apprentissage inductif utilisant l'analyse de données symboliques. Leur utilisation nécessite un apprentissage déductif pour construire les règles, situations et autres concepts associés. Enfin, l'utilisation efficace des connaissances passe par leur intégration au moyen d'un renforcement guidé par les émotions. Ces trois types d'apprentissage sont rarement combinés car basés sur des représentations des connaissances très différentes. Chacun d'entre eux a donc nécessité une adaptation particulière pour fonctionner avec le modèle de représentation des connaissances utilisé. Ce modèle et le fonctionnement global de l'agent ont étés développés pour permettre ces apprentissages ainsi qu'un raisonnement réflexif et efficace de l'agent.

### 11.1.2. Une représentation homogène des connaissances

Tout modèle de raisonnement ou d'apprentissage dépend fortement de la représentation des connaissances qu'il utilise. L'utilisation de trois types d'apprentissages a donc nécessité le développement d'une représentation des connaissances adaptée à chacun d'entre eux. Pour permettre à l'agent d'analyser, exécuter, modifier et renforcer ses propres règles, nous avons proposé au chapitre 4 une méthode de représentation homogène des connaissances. Cette représentation, fondée sur un graphe de concepts fluides inspiré de [Mitchell 1993, Hofstadter 1995] et [Wang 1994] permet une définition floue (plus ou moins large) et fluide (variable avec le temps) des concepts utilisés. Particulièrement adaptée pour une sémantique variable

avec l'environnement, cette représentation permet également de représenter de façon explicite les règles utilisées par l'agent. Celui-ci peut ainsi à la fois les créer, les modifier et les exécuter.

### 11.1.3. Un fonctionnement indépendant de la sémantique fondé sur l'état mental de l'agent

L'utilisation de concepts fluides, d'une sémantique et de règles variables nécessite un fonctionnement particulier. Nous avons adapté le modèle de fonctionnement heuristique probabiliste des systèmes tels que CopyCat pour qu'il soit utilisable par un agent, et qu'il permette l'apprentissage et la représentation homogène des connaissances. Ce modèle de fonctionnement, présenté au chapitre 5, est fondé sur deux principes : D'abord l'interprétation successive d'un grand nombre de règles, représentées explicitement dans le graphe et sélectionnées de façon probabiliste. Ensuite, la transmission d'un flux d'activation dans le graphe, qui permet à l'agent de ne considérer à un instant donné que les concepts et les règles les plus pertinents. Grâce à la sélection probabiliste des règles et des concepts en fonction de l'activation transmise, les anciens et les nouveaux concepts sont utilisés de la même façon. Cette sélection se fait en favorisant les connaissances les plus adaptées en fonction de l' « état mental » de l'agent déterminé par ses concepts les plus activés. L'activation provient de trois origines : tout d'abord des concepts activés eux-mêmes ; ils vont transmettre leur activation à leurs voisins, l'état mental actuel déterminant alors l'état mental suivant. Les perceptions vont également introduire de l'activation dans le graphe ; elles fonctionnent aussi bien de façon réactive (par introduction d'activation) que de façon proactive (par exécution de règles déterminées par l'état mental actuel). Enfin, les émotions introduisent de l'activation et renforcent les liens entre les connaissances ; elles permettent de guider l'agent indépendamment de sa sémantique (en fonction de l'état de l'agent et de son environnement). Ce fonctionnement présente l'avantage de ne considérer pour les calculs qu'une partie du graphe, de taille fixe. La vitesse de fonctionnement des différentes phases n'est donc pas fonction de la taille totale du graphe.

### 11.1.4. Une utilisation dynamique de l'analyse de données symboliques sur des graphes de connaissances

L'apprentissage inductif, présenté au chapitre 6, utilise l'analyse de données symboliques pour extraire de nouveaux concepts du graphe de connaissances

indépendamment de sa sémantique. Ce type d'apprentissage est particulièrement adapté aussi bien au type de représentation des connaissances utilisé (symbolique car sous forme de graphes), qu'à l'objectif de l'apprentissage. En effet, l'objectif de l'induction est de trouver des concepts utiles, d'où le choix d'un outil issu du Data Mining. Pour utiliser l'analyse de données symboliques, nous avons défini une mesure de similarité entre graphes. Celle-ci est bien adaptée à la dynamique de l'apprentissage : les modifications apportées au graphe par les apprentissages précédents sont pris en compte par la mesure de la redondance afin de favoriser les nouveaux concepts originaux.

L'objectif des méthodes de Data Mining est généralement d'extraire des concepts pour qu'un analyste extérieur les utilise. Notre objectif est que l'agent lui-même les utilise de façon identique à ses concepts originaux. Nous avons donc également défini un algorithme d'individualisation afin d'introduire le nouveau concept induit en tant que nouvel individu dans le graphe de connaissances.

Enfin, l'évolution constante et la complexité des concepts utilisés nous ont conduit à développer une méthode de visualisation dynamique d'objets symboliques.

### 11.1.5. Une application réflexive de règles déductives

L'apprentissage déductif, présenté au chapitre 8, fournit des règles pour créer les règles d'identification et certaines hypothèses associées au nouveau concept induit. En particulier, nous présentons les règles de création et d'analyse de règles de perception. De même, nous décrivons la création et la validation des hypothèses conduisant à la création d'heuristiques adaptées au domaine d'application. Ce modèle permet aux perceptions utilisées et créées par l'agent de fonctionner de façon aussi bien réactive que proactive.

### 11.1.6. Un contrôle indépendant à l'aide d'émotions

Le contrôle de l'agent, décrit au chapitre 9, montre comment il est possible, en combinant contrôle explicite par les objectifs et contrôle implicite par les émotions, de guider l'agent malgré sa sémantique variable et un environnement inconnu. Les émotions sont des fonctions définies le plus indépendamment possible de la sémantique, principalement en fonction de l'environnement et de l'état structurel de l'agent. Le contrôle émotionnel se fait via une intégration par renforcement des anciens et des nouveaux concepts dans le graphe. Ce renforcement permet de réactiver les concepts utiles au moment où une combinaison de circonstances similaires se reproduit. En plus de l'intérêt du point de vue de l'utilité des

concepts considérés, le renforcement permet de guider l'agent : les concepts (y compris les règles) conduisant à des émotions positives seront plus souvent renforcés, et donc favorisés par l'agent.

### 11.1.7. Une application du modèle facilement adaptable

Comme l'illustre notre application, présentée au chapitre 10, le système global remplit son rôle de découverte, d'adaptation et de raisonnement en fonction de l'environnement. Les différents résultats présentés aux chapitres 6, 7, 8 et 9 montrent le bon fonctionnement des différents systèmes de raisonnement et d'apprentissage. Une interface adaptée à la sémantique variable et aux concepts fluides a été développée et est également présentée au chapitre 10.

## 11.2. Perspectives

Le modèle proposé dans cette thèse est un outil de base. Il offre un modèle cohérent et global qui présente l'avantage de pouvoir être étendu dans de nombreuses directions.

Le mode de raisonnement de l'agent peut être modifié et amélioré en changeant ou complétant les règles explicites dont il dispose au départ. Il est ainsi possible de lui donner des concepts et des règles relatifs à la planification, la réflexion tactique ou stratégique, ou tout autre domaine qui peuvent se révéler utile.

Il est également possible d'inclure d'autres méthodes intégrées utiles pour l'agent, comme celle réalisant l'induction par analyse de données symboliques. D'autres outils d'ADS, utilisant la mesure de similarité proposée, peuvent notamment être utilisés. Inversement, les outils d'ADS proposés dans cette thèse peuvent être appliqués à d'autres domaines, indépendamment de notre modèle d'agent.

Enfin, notre modèle d'agent peut être appliqué à de nombreux domaines où ses caractéristiques peuvent se révéler utiles. C'est particulièrement vrai des environnements complexes et à priori inconnus, comme l'aide à la conception.

### 11.2.1. Raisonnement de l'agent

L'adaptabilité de l'agent par rapport à l'environnement est limitée par les règles de déduction qui lui sont données au départ. Par exemple, avec les règles décrites dans cette thèse, l'agent ne pourrait apprendre comment déplacer une pièce à un endroit précis dans un jeu de Tetris complet. Cela ne provient pas du mode de représentation des connaissances, du

fonctionnement, de l'induction ou du renforcement, tous indépendants du type de connaissance traité. Cela provient des règles de départ de l'agent, et donc des systèmes déductifs et de contrôle explicite, représentés sous forme de règles explicites. Ceux-ci nécessitent des règles de planification et de déduction d'actions composées supplémentaires pour réaliser un tel apprentissage. Comme nous l'avons vu aux chapitres 8 et 9, cela ne remettrait pas en cause le modèle. Au contraire, il s'agit d'un de ses atouts que de permettre l'inclusion de systèmes aussi divers que des règles de contrôle (type BDI), de planification, de résolution de problème, de gestion explicite des émotions ou de gestion temporelle. Cette inclusion peut se faire par simple ajout d'un ensemble de règles explicites dans le système, sans devoir changer aucun des autres éléments. Comme nous l'avons vu au chapitre 10, c'est déjà de cette façon que sont intégrés les concepts actuels, comme les règles de contrôle et de déduction.

Après avoir présenté dans cette thèse les règles de traitement spécifiques aux perceptions, l'évolution logique dans l'étude du raisonnement consiste donc à s'intéresser aux actions complexes, au traitement explicite des émotions, à la stratégie et à la tactique de jeu. La première étape est d'inclure des règles de planification simples qui permettraient à l'agent de bien gérer et de créer de nouvelles actions complexes. Par exemple, pour réaliser une action comme placer une pièce à une position donnée avec la rotation souhaitée, il pourrait, à partir des résultats des actions individuelles (changement de rotation, changement d'abscisse et d'ordonnée), construire un plan pour y parvenir. Il pourrait ensuite généraliser les plans semblables par induction à l'aide de l'analyse de données symboliques. Des règles de déduction relatives à la construction d'actions lui permettraient alors de construire des actions plus complexes (placer une pièce) qu'il pourrait ensuite utiliser directement pour d'autres plans.

Des règles plus approfondies de monitorage et d'auto-analyse des règles conduiraient à une réflexivité plus approfondie qui lui permettrait de s'auto-améliorer quelle que soit la règle concernée. Plus précisément, l'utilisation de règles explicites d'analyse de la tactique de jeu (choix du placement) et d'analyse des règles d'analyse elles-mêmes permettraient d'utiliser toute la puissance de représentation et d'analyse réflexive du modèle.

De même, il serait probablement intéressant d'étudier et d'ajouter des règles d'analyse des erreurs. Les heuristiques construites et utilisées par l'agent peuvent en effet le conduire à des actions ou des connaissances erronées. Ceci est dû à deux facteurs : d'abord au fait que l'environnement est a priori inconnu, il est donc tout a fait possible qu'une règle qui y est

observée se révèle finalement fausse, soit pour certains cas qui n'étaient pas encore survenus, soit tout simplement parce que cette règle est le résultat d'un hasard persistant. Ensuite, l'objectif des heuristiques n'est pas d'obtenir un résultat toujours juste, mais d'obtenir un résultat efficace dans un temps réduit. Ceci est d'autant plus vrai pour des domaines d'application très complexes où les données sont nombreuses et longues à traiter pour obtenir un résultat certain. Que ce soit pour guider un robot en mouvement ou pour décider du placement de la prochaine pièce de Tetris qui est en train de tomber, il est plus utile d'avoir une réponse rapide, même si elle peut se révéler parfois fausse, que d'obtenir une réponse juste (ce qui n'est pas toujours possible) en un temps trop long. Toutes ces raisons rendent le raisonnement et la surveillance des erreurs particulièrement intéressants à développer.

### 11.2.2. Induction et analyse de données symboliques

Pour améliorer la méthode d'apprentissage inductif à l'aide de l'analyse de données symboliques, il pourrait être intéressant d'utiliser d'autres critères et méthodes pour sélectionner les classes d'individus. En particulier, l'utilisation de l'activation des connaissances permettrait de ne considérer que les objets les plus intéressants pour l'agent. Une méthode de classification telle que celle développée actuellement par [Limam 2004] permet de prendre en compte simultanément la similarité et une autre variable cible, telle que l'activation, pour constituer les classes, et pourrait donc être une bonne extension de notre méthode d'apprentissage.

Pour améliorer la visualisation dynamique des objets symboliques utilisés, il pourrait également être intéressant de partir d'une pyramide 3D, et non 2D, telle que celle présentée par [Diday 2004]. Cela permettrait de représenter les objets dans le plan et non plus à partir d'un axe. Il serait également intéressant d'appliquer les cartes auto-organisatrices de Kohonen [Kohonen 1997] étendues aux données symboliques [El Golli*, et al.* 2004] comme alternative de visualisation des concepts utilisés.

L'extension de la similarité et de l'individualisation à d'autres types de représentation sous forme de graphes, comme les graphes conceptuels [Sowa 2000], permettrait d'étendre leurs domaines d'applicabilités.

La représentation des connaissances, ainsi que l'induction utilisant la mesure de similarité que nous proposons, peuvent également être utilisées dans un domaine comme le Web Mining. La représentation généralement utilisée dans ce domaine est purement sémantique ; l'ajout d'une information fonctionnelle pourrait être utile. Par exemple, un lien

fort entre deux termes opposés est naturel dans notre modélisation et apporte une information intéressante pour sélectionner des pages web pertinentes. De même, les relations d'inhibition (liens négatifs) enrichissent la modélisation utilisée. Comme nous l'avons vu en section 6.2.1.2.1, il existe une analogie très forte entre nos concepts fluides et les pages web. Les liens hypertextes rendent flou la définition de la page web (combien de pages liées faut il inclure dans sa descriptions ? Quels liens suivre ?). Il pourrait être intéressant de modéliser une page par une connaissance (nœud) liée à d'autres connaissances qui représenteraient aussi bien d'autres pages web que des concepts extraits du contenu de la page. Il serait alors possible d'enrichir la base de concepts en analysant les pages et en induisant de nouveaux concepts pertinents, puis en construisant les règles qui permettraient de repérer ces concepts dans d'autres pages, afin de mieux les décrire et donc de mieux les analyser.

De même, la méthode de visualisation dynamique d'objets symboliques peut être appliquée à d'autres domaines, comme les données boursières. Nous avons montré, dans [Caillou et Diday 2001] comment modéliser ces données sous forme d'objets symboliques et l'intérêt, dans ce cadre, de l'utilisation des méthodes d'ADS. Nous avons notamment utilisé la représentation par pyramide du logiciel SODAS[Diday 2000b] pour obtenir des analyses telles que celle de la figure 11.1.

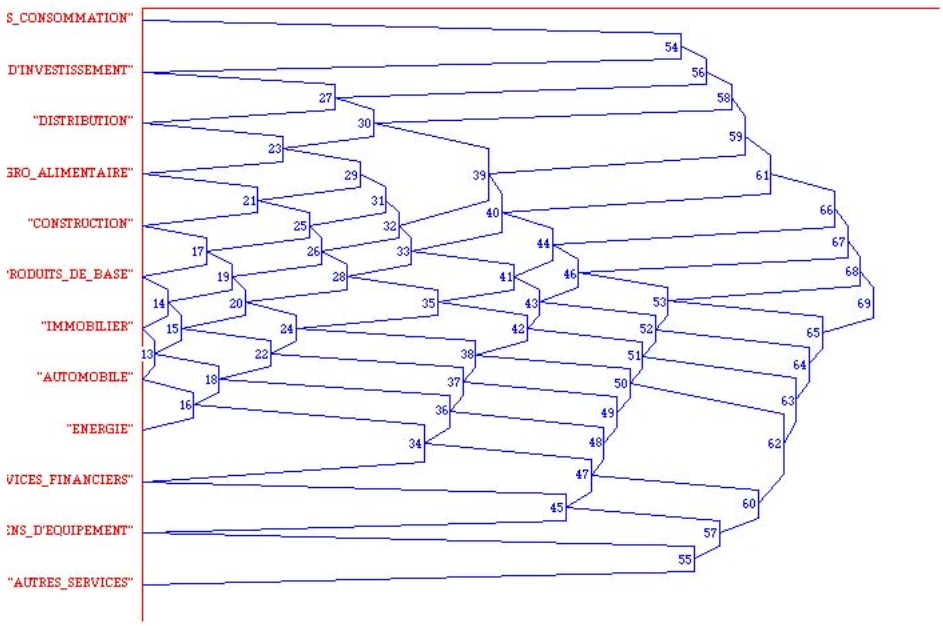



**Figure 11.1 Résultat de l'application de la méthode PYR sur 12 secteurs du SBF 120 (tiré de [Caillou et Diday 2001])**

Cette pyramide est le résultat de l'analyse des différents secteurs boursier sur le SBF 120 entre le 1er novembre 1999 et le 27 mars 2000. Nous avons identifié et interprété certains

paliers pertinents, en particulier le palier 53. Celui-ci isole les neuf secteurs centraux des trois secteurs extrêmes, ce qui constitue une classification pertinente compte tenu des comportements des titres sur cette période. L'application de notre méthode de représentation dynamique conduirait à sélectionner automatiquement certains paliers (et notamment ici le numéro 53, qui apparaît comme correspondant au saut maximum). Leurs propriétés (cardinalité de l'extension, variance interne) et leurs relations (inclusions, distances) pourraient alors être représentées de façon lisible. De plus, l'étude dynamique nous permettrait de faire apparaître l'évolution des ensembles ainsi représentés. Les données boursières, en constante modification et facilement disponibles, constituent donc une application logique de notre représentation.

### 11.2.3. Applications du modèle d'agent

Le modèle proposé est particulièrement adapté pour les domaines complexes nécessitant des raisonnements et où les concepts à utiliser sont à priori inconnus. Nous proposons ici plusieurs applications possibles correspondant à ces critères.

Le domaine le plus proche de l'exemple proposé et le plus naturel est celui des jeux, et en particulier, des jeux en temps réel. Comme nous l'avons vu à la section 11.2.1, cette application bénéficierait de règles spécifiques aux actions et d'un modèle de planification supplémentaire ajoutés à l'agent.

#### 11.2.3.1. Application à la conception assistée par ordinateur

Le domaine de l'aide à la conception constitue également une application particulièrement intéressante. Le principe des programmes de Conception Assistée par Ordinateur (CAO) est d'aider l'utilisateur à créer un nouveau concept (de voiture, d'avion, de programme, …). Cette application est adaptée à notre modèle dans la mesure où le concepteur du programme de CAO ne sait pas (ou très peu) quels vont être les concepts manipulés. Même l'utilisateur peut être incapable d'expliciter sa démarche de conception : les règles de conception, que le programme va chercher à découvrir, font alors parti du savoir tacite de l'utilisateur, qu'il sait appliquer mais non expliquer. De même certains concepts sous-jacents vont être utilisés par l'utilisateur sans qu'il en soit conscient. La modélisation sous forme de concepts fluides est particulièrement adaptée pour suivre la conception d'un nouvel objet par l'utilisateur [Kazakci 2004, Kazakci et Tsoukias 2004]. De plus, dans les programmes de CAO actuels, des formes d'apprentissage existent, mais ces techniques sont appliquées pour

apprendre le résultat final du processus de conception, et non pas la manière par laquelle le processus est conduit. Par conséquent, ces outils restent inchangés après leur utilisation [Gero 1996, Kazakci 2004, Kazakci et Gero 2005]. Au contraire, notre agent peut modifier son fonctionnement en fonction de son expérience, et parallèlement à la construction de nouveaux concepts.

L'aide apportée à l'utilisateur par un agent utilisant notre modèle pourrait être le suivant [Kazakci et Caillou 2005, soumis] : L'utilisateur manipule tout d'abord seul le programme de CAO pour définir divers objets. L'agent l'observe, enrichissant ses connaissances avec les nouveaux concepts créés, et essaie d'extraire des règles et des concepts sous-jacents à partir de ces expériences. Progressivement, il va pouvoir proposer des composants à l'utilisateur, en particulier dans deux cas : d'abord lorsqu'une règle explicite apprise lui indique que la situation actuelle implique généralement l'ajout d'un certain composant (par exemple dans une voiture, l'ajout d'une roue en implique trois autres). Ensuite, lorsqu'un composant est particulièrement activé dans son graphe. Ainsi, lors de la conception d'une voiture, lorsque l'utilisateur ajoute la carrosserie, l'agent peut lui proposer son matériel en fonction de l'activation du concept correspondant. Si l'utilisateur a déjà ajouté de très nombreux matériaux particulièrement légers, les concepts de *matériel*, de *carrosserie* et de *légèreté* vont activer conjointement celui d'*aluminium*, qui aura plus de chance que celui de *tôle* d'être proposé en priorité. Au contraire, si les composants étaient bon marché, le concept de *plastique* sera proposé, et ainsi de suite.

Les deux types de propositions sont utiles et complémentaires : le premier correspond à une explicitation du savoir tacite sous forme de règles ou de concepts. Cela permet à l'utilisateur d'accélérer son travail en lui évitant les combinaisons et implications habituelles. C'est également utile lors d'un changement d'utilisateur, afin de pouvoir réutiliser le savoir d'un ou plusieurs experts et de pouvoir le décrire explicitement. Le second type de proposition (à partir de l'activation) correspond plus à de la créativité qu'à une aide de la part de l'agent, dans la mesure où elle provient du savoir tacite de l'agent lui-même. La conjonction de plusieurs composants ajoutés par l'utilisateur le conduit à proposer spontanément un ajout.

Au delà de simples propositions faites dans un programme de CAO, l'aide peut également être proposée sous forme d'un agent virtuel interagissant avec des avatars d'agents humains dans un monde virtuel de conception [Kazakci 2004]. Le cadre d'une telle interaction a notamment été décrit par [Maher*, et al.* 2001] et permettrait à l'agent de s'enrichir à partir de nombreuses interactions, et aux utilisateurs de choisir entre différents agents aux spécialités ou

aux caractéristiques éventuellement différentes. On peut en particulier imaginer des agents plus ou moins innovants ou excentriques en fonction des émotions de base qui leur sont attribuées, et qui les conduisent à des résultats d'apprentissages différents.

#### 11.2.3.2. Applications multi-agents

Cette application dans un monde virtuel nous permet d'introduire les domaines multi-agents qui sont une suite logique à notre travail dans la mesure où nous proposons un modèle d'agent.

Une application multi-agents possible est très proche de la CAO dans la technique d'application, bien que le domaine semble particulièrement éloigné : il s'agit de la négociation. Un intérêt de notre modèle d'agent dans un processus de négociation consiste en effet à découvrir les stratégies sous-jacentes aux comportements de ses interlocuteurs. Or, découvrir ces stratégies consiste à chercher les concepts et règles tacites derrière le comportement observé. Ce savoir est tacite non plus parce qu'il ne peut être exprimé (comme en CAO), mais parce que l'interlocuteur n'a aucun intérêt à l'exprimer. Une fois certaines règles identifiées, l'agent pourrait modifier dynamiquement son comportement pour pouvoir négocier plus efficacement (notons ici l'intérêt et la nécessité pour l'agent de pouvoir modifier son propre fonctionnement grâce à la représentation homogène des connaissances).

D'autres applications multi-agents sont envisageables. La première est d'utiliser ce modèle pour un robot, réel ou virtuel. La complexité et le manque d'information sur l'environnement rendent particulièrement utile la découverte de concepts et d'heuristiques adaptés à cet environnement. La découverte de concepts en parallèle par plusieurs agents qui interagissent peut également permettre d'étudier de façon intéressante le partage et l'émergence de concepts à travers le langage. Enfin, une application à la vie artificielle avec mutations génétiques est également envisagée. La représentation homogène des connaissances permet en effet une modification du comportement de l'agent aussi bien pendant sa vie qu'entre les générations. Cela pourrait être utilisé pour simuler l'interaction entre des amibes et des virus. Les amibes ont un comportement défini génétiquement qui peut évoluer par *crossing-over* entre les générations. Une modification aléatoire des liens et des émotions nous permettrait de modifier aussi bien la conduite que les règles et les données utilisées. Les virus ont quant à eux la possibilité d'essayer de modifier l'ADN des amibes, et donc de modifier leur comportement au cours de leur vie. Notre représentation permet simplement cette modification en agissant sur le graphe de l'amibe.

De façon similaire, l'influence des émotions sur le comportement peut aider à simuler des situations de prises de décisions humaines en introduisant le facteur émotionnel. En particulier, il est possible d'étendre un modèle tel que celui de la « poubelle » [Cohen*, et al.* 1972] qui simule les prises de décisions, en sélectionnant de façon très simple des compromis acceptables (ce type de modèle restant particulièrement adapté pour la simulation sociale comme le montre notamment [Romelaer et Huault 2002]).

# Chapitre 12

# Bibliographie


S. Aknine et P. Caillou. (2004a), "Agreements without Disagreements: A Coalition Formation Method," in *ECAI 04*, Valencia: IOS Press, pp. 3-7.

S. Aknine et P. Caillou. (2004b), "Méthode Consensuelle De Formation De Coalitions," in *RFIA 04*, Toulouse.

D. Bertsekas et J. Tsilsiklis. (1996), "Neuro-Dynamic Programming."

D. Billings, A. Davidson, J. Schaeffer et D. Szafron. (2002), "The Challenge of Poker," *Artificial Intelligence*, 34, 201-240.

H.-H. Bock. (2000), "Symbolic Data," in *Analysis of Symbolic Data*, eds. H.-H. Bock and E. Diday, Springer, pp. 39-53.

B. Bouzy et T. Cazenave. (2001), "Computer Go: An Ai Oriented Survey," *Artificial Intelligence*, 132, 39-103.

M. E. Bratman, D. J. Israel et M. E. Pollack. (1988), "Plans and Resource-Bounded Practical Reasoning," *Computational Intelligence*, 4, 349-355.

C. Breazeal. (2000), "*Sociable Machines: Expressive Social Exchange between Humans and Robots*," Sc.D. Dissertation, M.I.T. Dept. of EECS.

R. A. Brooks (1999), *Cambrian Intelligence* (MIT Press ed.),

P. Caillou. (2003a), "Découverte Et Utilisation Efficace De Nouveaux Concepts Et Heuristiques : Proposition D'un Modèle Agent," in *RJCIA 03*, Laval, France: Presses Universitaires de Grenoble, pp. 223-236.

P. Caillou. (2003b), "Raisonnement Heuristique Réflexif Dans Un Agent Temps Réel," in *JNMR 03*, Laval, France: INRIA, pp. 51-66.

P. Caillou, S. Aknine et S. Pinson. (2001), "Méthode De Formation Et De Restructuration Dynamique De Coalitions D'agents," in *JFIADSMA 01*, Montreal, Canada: Hermes, pp. 55-70.

P. Caillou, S. Aknine et S. Pinson. (2002a), "How to Form and Restructure Multi-Agent Coalitions," in *AAAI 2002 Workshop on Coalition Formation*, Edmonton, Canada: AAAI Press, pp. 32-37.

P. Caillou, S. Aknine et S. Pinson. (2002b), "A Multi-Agent Method for Forming and Dynamic Restructuring of Pareto Optimal Coalitions," in *AAMAS 02*, Bologna, Italy: ACM Press, pp. 1074-1081.

P. Caillou, S. Aknine et S. Pinson. (2002c), "Multi-Agent Models for Searching Pareto Optimal Solutions to the Problem of Forming and Dynamic Restructuring of Coalitions," in *ECAI 2002*, Lyon, France: IOS Press, pp. 13-17.

P. Caillou, S. Aknine et S. Pinson. (2003), "Méthode Pareto-Optimale De Formation Et Da Restructuration Dynamique De Coalitions D'Agents," *Revue d'Intelligence Artificielle*, 17(4), 655-685.

P. Caillou et E. Diday. (2001), "Modélisation De Données Boursière À L'aide De L'analyse De Données Symbolique," in *EGC 2001*, Nantes: Hermes.

P. Caillou et E. Diday. (2005), "Analyse De Données Symboliques Et Graphe De Connaissances D'un Agent," in *EGC 2005*, Paris: Hermes.

M. Campbell, A. J. J. Hoane et F.-h. Hsu. (2002), "Deep Blue," *Artificial Intelligence*, 134, 57-83.

D. Canamero. (1997), "Modeling Motivations and Emotions as a Basis for Intelligent Behavior," in *First International Conference on Autonomous Agents*, New York: ACM Press, pp. 148-155.

D. Canamero. (1998), "Issues in the Design of Emotional Agents: The Tangled Knot of Cognition," in *1998 AAAI Fall Symposium*, Menlo Park, CA: AAAI Press, pp. 49-54.

D. Canamero. (1999), "Emotions Pour Les Agents Situés," in *Intelligence Artificielle Située*, eds. A. Drogoul and J.-A. Meyer, Paris: Hermes, pp. 59-70.

T. Cazenave. (1996), "*Système D'apprentissage Par Auto-Observation, Application Au Jeu De Go*," rapport de thèse, Université Paris 6, LIP6.

M. D. Cohen, J. G. March et O. J.P. (1972), "A Garbage-Can Model of Organizational Choice," *Administrative Science Quarterly*, 17, 1-25.

A. Cornuéjols et L. Miclet (2002), *Apprentissage Artificiel*, Paris: Eyrolles.

A. R. Damasio (1995), *L'erreur De Descartes: La Raison Des Émotions*, Paris: Odile Jacob.

A. R. Damasio (1999), *Le Sentiment Même De Soi*, Paris: Odile Jacob.

R. Davis et D. Lenat (1982), *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill.

E. D. Demaine, S. Hohenberger et D. Liben-Nowell. (2002), "Tetris Is Hard, Even to Approximate," Technical Report MIT-LCS-TR-865, MIT.

D. Dennett (1991), *La Conscience Expliquée*, Paris: Odile Jacob.

D. Dennett (1996), *La Diversité Des Esprits*, Paris: Hachette.

E. Diday. (1984), "Une Représentation Visuelle Des Classes Empietantes," in *Rapport Inria N.291*, Rocquencourt, France.

E. Diday. (2000a), "L'analyse Des Données Symboliques : Un Cadre Théorique Et Des Outils Pour Le Data Mining," in *Induction Symbolique Numérique À Partir De Données*, Toulouse: Cépaduès-Editions, pp. 12-101.

E. Diday. (2000b), "Symbolic Data Analysis and the Sodas Project: Purpose, History, Perspective," in *Analysis of Symbolic Data*, eds. H.-H. Bock and E. Diday, Springer, pp. 1-23.

E. Diday. (2004), "Spatial Pyramid Clustering Based on a Tessellation," in *International Federation of Classification Societies (IFCS 04)*, Chicago: Springer Verlag, pp. 105-122.

A. El Golli, B. Conan-Guez et F. Rossi. (2004), "A Self Organizing Map for Dissimilarity Data," in *IFCS 2004*, to appear.

FARG. (2004), "Farg Home Page", http://www.cogsci.indiana.edu.

J. Ferber (1995), *Les Systèmes Multi-Agents*, Paris: Interéditions.

H. Foundalis. (2004), "Harry Foudalis Research Page", http://www.cs.indiana.edu/~hfoundal/research.html.

R. M. French. (1992), "*Tabletop: An Emergent, Stochastic Computermodel of Analogy-Making*," PhD thesis, University of Michigan.

R. M. French (1995), *The Subtlety of Sameness*, Cambridge, MA: The MIT Press.

J. S. Gero. (1996), "Design Tools That Learn: A Possible Cad Future," in *Information Processing in Civil and Structural Design*, pp. 17-22.

A. Guillot. (1999), "Pour Une Approche Dynamique Des Animats," in *Intelligence Artificielle Située*, eds. A. Drogoul and J.-A. Meyer, Paris: Hermes, pp. 33-58.

D. Harada et S. Russel. (1999), "Learning Search Strategies".

P. E. Hart, N. J. Nilsson et B. Raphael. (1968), "A Formal Basis for the Heuristic Determintation of Minimum Cost Path," *IEEE Trans. Systems Sci. Cybernet.*, 4(2), 100-107.

H. J. v. d. Herik, J. W. H. M. Uiterwijk et J. v. Rijswijck. (2002), "Games Solved: Now and in the Future," *Artificial Intelligence*, 134, 277-311.

D. Hofstadter (1985), *Gödel, Escher, Bach, Les Brins D'une Guirlande Éternelle*, Paris: Interéditions.

D. Hofstadter (1995), *Fluid Concepts and Creative Analogies*, New York: BasicBooks.

D. Hofstadter et R. M. French. (1992), "Probing the Emergent Bahavior of Tabletop, an Architecture Uniting High-Level Perception with Analogy Making," in *Fourteenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, pp. 528-533.

T. Ishida (1997), *Real-Time Search for Learning Autonomous Agents*, Dordrecht: Kluwer Academic.

A. Junghanns et J. Schaeffer. (2001), "Sokoban: Enhancing General Single-Agent Search Methods Using Domain Knowledge," *Artificial Intelligence*, 129, 219-251.

A. O. Kazakci. (2004), "*Assistant Personnel De Conception: Operationnaliser La Théorie C-K Par Des Agents Situés De Conception*," Mémoire de Pré-soutenance, Université Paris Dauphine, LAMSADE.

A. O. Kazakci et P. Caillou. (2005, soumis), "Fluid Concepts and the C-K Design Theory," in *Sixth International Roundtable Conference Computational and Cognitive Models of Creative Design*, Heron Island, Australia.

A. O. Kazakci et J. S. Gero. (2005), "Situated Design Agents as Personal Design Assistants," *in progress*.

A. O. Kazakci et A. Tsoukias. (2004), "Extending the C-K Design Theory to Provide Theoretical Background for Situated Design Agents," in *8th International Design Conference*, Dubrovnik, Croatia.

D. Kirsh et P. Maglio. (1994), "On Distinguishing Epistemic from Pragmatic Action," *Cognitive Science*, 18, 513-549.

Y. Kodratoff. (2001), "Comparing Machine Learning and Knowledge Discovery in Databases: An Application to Knowledge Discovery in Texts," in *Machine Learning and Its Applications* (Vol. 1), Berlin: Springer-Verlag, pp. 1-21.

T. Kohonen (1997), *Self-Organisation Maps*, New-York: Springer Verlag.

R. E. Korf. (1990), "Real-Time Heuristic Search," *Artificial Intelligence*, 42(2-3), 189-211.

R. E. Korf. (2000), "Recent Progress in the Design and Analysis of Admissible Heuristic Functions," *AAAI 2000*, 1165-1170.

S. Kornman. (1993a), "*Sade : Un Système Réflexif De Surveillance À Base De Connaissances*," Thèse de Doctorat, Université Paris 6, LAFORIA.

S. Kornman. (1993b), "Surveillance Et Réflexivité. Présentation Du Système Sade," *Revue d'Intelligence Artificielle*, 7, 295-327.

J. Laird et M. v. Lent. (2000), "Human-Level Ai's Killer Application: Interactive Computer Game," in *AAAI 2000*, Austin, pp. 1171-1178.

D. Lenat. (1978), "The Ubiquity of Discovery," *Artificial Intelligence*, 9(3), 257-285.

D. Lenat. (1982), "The Nature of Heuristics," *Artificial Intelligence*, 19, 189-249.

D. Lenat. (1983a), "Eurisko: A Program That Learns New Heuristics and Domain Concepts," *Artificial Intelligence*, 21, 61-98.

D. Lenat. (1983b), "Theory Formation by Heuristic Search," *Artificial Intelligence*, 21, 31-59.

D. Lenat et J. S. Brown. (1983), "Why Am and Eurisko Appear to Work," *Artificial Intelligence*, 23(3), 269-294.

M. M. Limam. (2004), "*Classification Descendante Hiérarchique Avec Critères De Discrimination*," Mémoire de présoutenance, Université Paris Dauphine, CEREMADE.

M. L. Maher, S. Simoff et S. Clark. (2001), "Learner-Centred Open Virtual Environments as Places," in *European Perspectives on Computer-Supported Colaborative Learning Conference*, pp. 437-444.

J. B. Marshall. (1999), "*Metacat: A Self-Watching Cognitive Architecture for Analogy-Making and High-Level Perception*," PhD thesis, Indiana University, Department of Computer Science.

J. B. Marshall. (2002), "Metacat: A Program That Judges Creative Analogies in a Microworld," in *ECAI 02 workshop on creative systems*, Lyon, pp. 77-84.

J. B. Marshall. (2004), "Metacat Home Page", http://www.cs.pomona.edu/~marshall/metacat/.

G. E. McGraw. (1995), "*Letter Spirit (Part One): Emergent High-Level Perception of Letters Using Fluid Concepts*," PhD Thesis, Indiana University.

G. E. McGraw et D. Hofstadter. (1993), "Letter Spirit: An Architecture for Creativity in a Micro-Domain," in *Third Congress of the Italian Association for Artificial Intelligence*, Torino: Springer-Verlag.

J.-A. Meyer et S. W. Wilson (1991), *From Animals to Animat: Proceedings of the First Intenational Conference on Simulation of Adaptative Behavior*, Cambridge, MA: MIT Press.

M. Minsky. (1975), "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, ed. P. Winston, New York: McGraw-Hill, pp. 211-280.

M. Minsky (1985), *The Society of Mind*, New York: Simon and Schuster.

S. Minton, J. G. Carbonell, C. A. Knoblock, D. R. Kuokka, O. Etzioni et Y. Gil. (1989), "Explanation-Based Learning: A Problem Solving Perspective," *Artificial Intelligence*, 40, 63-118.

M. Mitchell (1993), *Analogy-Making as Perception : A Computer Model*, Cambridge, MA: MIT Press.

M. Mitchell et D. Hofstadter. (1990), "The Emergence of Understanding in a Computer Moduel of Concepts and Analogy-Making," *Nonlinear Phenomena*, 42(1-3), 322-334.

T. Mitchell (1997), *Machine Learning*, McGraw-Hill.

A. Napoli. (1997), "*Une Introduction Aux Logiques De Descriptions*," Rapport de recherche 3314, INRIA Lorraine.

A. Newell (1990), *Unified Theories of Cognition*, Cambridge: Harvard University Press.

A. Nicolle. (2002), "Sciences De L'artificiel, Modélisation Et Rationalité," *Revue d'Intelligence Artificielle*, 16(1-2), 63-86.

P. Petta et R. Trappl. (2001), "Emotions and Agents," in *ACAI 2001*, Prague: Springer-Verlag, pp. 301-316.

S. Pinson. (1987), "*Méta-Modèle Et Heuristique De Jugement : Le Système Credex. Application À L'évaluation Du Risque Crédit D'une Entreprise*," rapport de thèse, Université Paris 6.

S. Pinson. (1989), "Une Évaluation Multi-Expert Du Risque Entreprise: Le Système Credex," *Technique et Sciences Informatiques*, 8, 127-143.

J. Pitrat (1990), *Métaconnaissance, Futur De L'intelligence Artificielle* (Hermès ed.), Paris:

J. Pitrat (1993), *Penser Autrement L'informatique*, Paris: Hermes.

J. Pitrat (1995), *De La Machine À L'intelligence* (Hermes ed.), Paris:

J. Pitrat. (1996), "Implementation of a Reflective System," *Future Generation Computer Systems*, 12(2-3), 235-242.

J. Pitrat. (1997), "Reconnaitre Les Émotions Des Êtres Humains," in *Colloque Intelligence Artificielle Berder 1997*, pp. 33-46.

J. Pitrat. (1998), "Douglas Hofstadter Et Les Concepts Fluides," in *Colloque intelligence artificielle Berder*, Berder, pp. 20-28.

J. Pitrat. (1999), "Monitorer La Recherche D'une Solution," in *Colloque Intelligence Artificielle Berder 1999*, Berder, pp. 3-15.

J. Pitrat. (2000), "La Mise En Place Du Monitoring Dans Malice," in *Colloque Metaconnaissance de Berder*, Berder, pp. 115-131.

J. A. Rehling. (2001), "*Letter Spirit (Part Two): Modeling Creativity in a Visual Domain*," PhD Thesis, Indiana University.

G. D. Ritchie et F. K. Hanna. (1983), "Am: A Case Study in Ai Methodology," *Artificial Intelligence*, 23(3), 249-268.

P. Romelaer et I. Huault. (2002), "International Career Management: The Relevance of the Garbage-Can Model," Technical, CREPA - Université Paris Dauphine.

J. Schaeffer et H. J. v. d. Herik. (2002), "Games, Computers, and Artificial Intelligence," *Artificial Intelligence*, 134(january 2002), 1-8.

M. Scheutz. (2002), "Agents with or without Emotions?," in *FLAIRS 02*, Pensacola Beach: AAAI Press, pp. 89-94.

M. Shimbo et T. Ishida. (2003), "Controlling the Learning Process of Real-Time Heuristic Search," *Artificial Intelligence*, 146(1), 1-41.

H. A. Simon. (1976), "From Substantive to Procedurl Rationality," in *Method and Appraisal in Economics*, ed. S. J. Latsis, Cambridge.

J. F. Sowa (2000), *Knowledge Representation, Logical, Philosophical and Computational Fundations*, Brooks Cole.

B. Starynkevitch. (2001), "Un Regard Exterieur Sur Le Système Maciste De Jacques Pitrat," in *Colloque Metaconnaissance de Berder*, Berder, pp. 20-32.

P. Stone et R. S. Sutton. (2001), "Keepaway Soccer: A Marchine Learning Testbed," in *Robocup 2001*, Seattle: Springer, pp. 214-223.

R. S. Sutton et A. G. Barto (1998), *Reinforcement Learning*, Cambridge: MIT Press.

G. Tesauro. (2002), "Programming Backgammon Using Self-Teaching Neural Nets," *Artificial Intelligence*, 134, 181-199.

J. D. Velasquez. (1997), "Modeling Emotions and Other Motivations in Synthetic Agents," in *AAAI 97*, MIT Press.

J. D. Velasquez. (1998), "When Robots Weep: Emotional Memories and Decision-Making," in *AAAI 98*, Madison, Wisconsin: MIT Press, pp. 70-75.

P. Wang. (1994), "From Inheritance Relation to Non-Axiomatic Logic," *International Journal of Approximate Reasoning*, 11(4), 281-319.

P. Wang. (1995), "*Non Axiomatic Reasoning System - Exploring the Essence of Intelligence*," PhD Thesis, Indiana University, Department of Computer Science.

P. Wang (to appear), *Rigid Flexibility- the Logic of Intelligence*,

M. J. Wooldridge. (1999), "Intelligent Agents," in *Multiagent Systems*, ed. G. Weiss, Cambridge, MA: MIT Press, pp. 27-78.

M. J. Wooldridge (2001), *An Introduction to Multiagent Systems*, Wiley & Sons.

M. J. Wooldridge et S. Parsons. (1998), "Intention Reconsideration Reconsidered," in *5th International Workshop on Intelligent Agents (ATAL 98)*, Paris, France: Springer, pp. 63-79.

## Modèle ARCO : Apprentissages multiples et Raisonnement réflexif sur des Connaissances hOmogènes

**Philippe CAILLOU**

L'objectif de cette thèse est de proposer un modèle d'agent qui s'adapte à son environnement. Pour cela, il découvre et utilise efficacement de nouveaux concepts, objets et heuristiques, à partir de son expérience.

Les connaissances de l'agent sont représentées de façon homogène dans un graphe orienté afin qu'il puisse créer, exécuter et analyser ses propres règles. L'utilisation homogène et efficace de ces connaissances se fait grâce à la transmission d'un flux d'activation dans le graphe. L'activation des connaissance détermine l'état mental courant de l'agent, qui détermine lui-même de façon probabiliste les concepts et règles utilisées. La découverte de nouveaux concepts se fait par induction à l'aide de l'analyse de données symboliques. Des règles de déduction permettent de créer les règles et liens associés à ces nouveaux concepts. Enfin, le contrôle de l'agent via les émotions permet à la fois d'intégrer les nouveaux concepts en renforçant les liens utiles, et de guider l'agent indépendamment de sa sémantique.

Un agent correspondant au modèle décrit a été implémenté afin de vérifier sa cohérence et son fonctionnement sur un exemple simple. Les résultats montrent que les différents éléments s'intègrent bien et que les différents apprentissages successifs fonctionnent correctement.

**Mots clés** : agent, raisonnement réflexif, induction, déduction, renforcement, émotion, analyse de données symboliques.

**Laboratoire** : LAMSADE – Université Paris Dauphine

## ARCO Model: Combined learning and reflexive reasoning with an homogeneous knowledge representation

**Philippe CAILLOU**

In this thesis, we propose the model of an agent able to adapt to his environment. To achieve this, he discovers and efficiently uses new concepts from his experience.

Knowledge is homogeneously represented in an oriented graph, so the agent can create, execute and analyse his own rules. Transmission of an activation flux in the graph allows the homogeneous and efficient use of this knowledge. The activation level defines the actual mental state of the agent, which determines the rules and the concepts that will be considered for reasoning. We present a symbolic data analysis method to induce new concepts from the graph. To create useful concepts associated with this new ones, we present although deduction rules used by the agent. Finally, control via emotions integrates new concepts by reinforcing the links and guides the agent.

An agent based on this model has been developed and applied on a simple example. We were thus able to check the integrity and functionality of the model.

**Keywords**: agent, reflexive reasoning, induction, deduction, reinforcement, emotion, symbolic data analysis

**Research Unit** : LAMSADE – Université Paris Dauphine