# Comportements d'agents en mouvement : une approche cognitive pour la reconnaissance d'intentions

Nicolas Vidal

**THESE DE DOCTORAT DE**
**L'UNIVERSITE PIERRE ET MARIE CURIE**

Spécialité

**INFORMATIQUE**

Présentée par

**M. VIDAL Nicolas**

Pour obtenir le grade de

**DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE**

Sujet de la thèse :

**Comportements d'Agents en Mouvement : Une Approche Cognitive pour la Reconnaissance d'Intentions**

soutenue le --/09/2014

devant le jury composé de : (préciser la qualité de chacun des membres).

| | |
|---|---|
| --- | Présidente du jury |
| **M. AKNINE Samir** | Directeur de thèse |
| **M. TAILLIBERT Patrick** | Encadrant industriel |
| **M. MATHIEU Philippe** | Rapporteur |
| **M. MANDIAU René** | Rapporteur |
| **M. CAILLOU Philippe** | Examinateur |
| **Mme. EL FALLAH SEGROUCHNI Amal** | Examinateur |

Université Pierre & Marie Curie - Paris 6
Bureau d'accueil, inscription des doctorants
Esc G, 2ème étage
15 rue de l'école de médecine
75270-PARIS CEDEX 06

Tél. Secrétariat : 01 44 27 28 10
Fax : 01 44 27 23 95
Tél. pour les étudiants de A à EL : 01 44 27 28 07
Tél. pour les étudiants de EM à MON : 01 44 27 28 05
Tél. pour les étudiants de MOO à Z : 01 44 27 28 02
E-mail : scolarite.doctorat@upmc.fr

# Résumé

Dans un contexte applicatif de surveillance de zone maritime, nous voulons fournir à un opérateur humain des informations sémantiquement riches et dynamiques relatives aux comportements des entités sous surveillance. Réussir à relier les mesures brutes en provenance d'un système de capteurs aux descriptions abstraites de ces comportements est un problème difficile. Ce dernier est d'ailleurs en général traité en deux temps : tout d'abord, réaliser un prétraitement sur les données hétérogènes, multi-dimensionnelles et imprécises pour les transformer en un flux d'évènements symbolique, puis utiliser des techniques de reconnaissance de plans sur ces mêmes évènements. Ceci permet entre autre chose, la possibilité de décrire des étapes de plans symboliques de haut niveau sans avoir à se soucier des spécificités des capteurs bas niveau. Cependant, cette première étape est destructrice d'information et de ce fait génère une ambigüité supplémentaire dans le processus de reconnaissance. De plus, séparer les tâches de reconnaissance de comportements est générateur de calculs redondants et rend l'écriture de la bibliothèque de plans plus hardue. Ainsi, nous proposons d'aborder cette problématique sans séparer en deux le processus de reconnaissance. Pour y parvenir, nous proposons un nouveau modèle hiérarchique, inspiré de la théorie des langages formels, nous permettant de construire un pont au dessus du fossé sémantique séparant les mesures des capteurs des intentions des entités. Grâce à l'aide d'un ensemble d'algorithmes manipulant ce modèle, nous sommes capables, à partir d'observations, de déduire les plausibles futures évolutions de la zone sous surveillance, tout en les justifiant des explications nécessaires.

# Moving Objects Behaviours

## *A Cognitive Approach for Intention Recognition*

In a maritime area supervision context, we seek providing a human operator with dynamic information on the behaviors of the monitored entities. Linking raw measurements, coming from sensors, with the abstract descriptions of those behaviors is a tough challenge. This problem is usually addressed with a two-stepped treatment: filtering the multidimensional, heterogeneous and imprecise measurements into symbolic events and then using efficient plan recognition techniques on those events. This allows, among other things, the possibility of describing high level symbolic plan steps without being overwhelmed by low level sensor specificities. However, the first step is information destructive and generates additional ambiguity in the recognition process. Furthermore, splitting the behavior recognition task leads to unnecessary computations and makes the building of the plan library tougher. Thus, we propose to tackle this problem without dividing the solution into two processes. We present a hierarchical model, inspired by the formal language theory, allowing us to describe behaviors in a continuous way, and build a bridge over the semantic gap between measurements and intents. Thanks to a set of algorithms using this model, we are able, from observations, to deduce the possible future developments of the monitored area while providing the appropriate explanations.

# Mots Clefs

Reconnaissance d’activités
Reconnaissance de comportements
Programmation par contraintes
Grammaires formelles
Reconnaissance de formes

# Keywords

activity recognition
behavior recognition
constraint programming
formal grammars
pattern recognition

# Collaborations

**Laboratoire d'Informatique de Paris 6**

*4 place Jussieu*

*75005 Paris*


**THALES Systèmes Aéroportés**

*2 Avenue Gay Lussac*

*78990 Elancourt*

# Table Des Matières

# Chapitre 1

# Introduction

# 1.1 Problématique Applicative

Nous nous plaçons dans le cadre de la surveillance de vastes zones maritimes. Cela peut concerner des applications civiles telles que :

- La surveillance du respect des règles de trafic maritime.
- La régulation des quotas de pêche.
- Les missions de sauvetages en mer.
- La lutte contre le trafic illicite de marchandises.

Ou militaires :

- Le maintien de situation d'embargo.
- La protection de zones d'entraînement militaire.
- La protection des eaux territoriales.

Pour chacune de ces applications, il s'agit de surveiller des zones sur lesquelles évoluent un ensemble d'entités.

Aujourd'hui, un opérateur humain est pourvu d'un ensemble d'outils lui permettant d'assurer une veille sur un plan d'eau. Ces outils lui permettent d'obtenir une vision synthétique de l'ensemble des mesures prises par l'ensemble des capteurs sur la scène, on parle de fusion de données.

En effet, l'ensemble de ces données provient de multiples sources telles que des avions de patrouille maritime, de l'imagerie satellite, de stations radars ou encore d'observations humaines. Cet ensemble de sources de données est appelé réseau de capteurs. Ce dernier fournit donc des mesures hétérogènes et potentiellement incomplètes ou imprécises.

La tâche d'analyse de ce fouillis d'information est usuellement déléguée à l'opérateur. Cette tâche est difficile. Etant donné le nombre de paramètres devant être pris en compte dans le cadre de la surveillance de zone maritime, nous souhaitons automatiser une partie du processus d'analyse jusqu'à présent effectuée par l'opérateur. Nous voulons lui procurer un nouveau type d'information : des informations sur les comportements des entités sous surveillance. Ces informations sont fondamentales pour pouvoir obtenir une compréhension globale de la situation et pour permettre à l'humain de prendre des décisions appropriées. Nous allons donc concevoir un système de reconnaissance de comportement dédié à cette tâche.

## 1.2 Reconnaissance de Comportement

La *reconnaissance de comportement*, proche de la *reconnaissance de plan* ou encore de la *reconnaissance d'activité*, a pour but d'inférer des informations inaccessibles concernant une entité surveillée, à partir d'une séquence d'observations. De manière classique, une bibliothèque de comportements (souvent définie en intention) est disponible et l'on cherche à savoir si le comportement observé appartient à cette bibliothèque.

La reconnaissance de comportement *en ligne* vise au même but que la reconnaissance de comportement, avec la spécificité suivante : les phases d'observation et de reconnaissance sont confondues. Il est donc nécessaire d'émettre à chaque arrivée d'observation des hypothèses sur l'ensemble des comportements éligibles de la bibliothèque. Les comportements de la bibliothèque étant souvent classés ou annotés, déterminer en avance quel comportement l'entité sous surveillance est en train d'exhiber aide à la compréhension et à l'analyse de la séquence d'observation. D'autre part, les résultats de la reconnaissance de comportement en ligne peuvent être exploités à des fins prédictives.

La grande majorité des systèmes de reconnaissance de comportement est aujourd'hui divisée en deux catégories.

Ceux qui essayent, à partir de données issues de capteurs physiques prenant des mesures sur le monde réel, de détecter des évènements, et ceux qui à partir d'un ensemble d'évènements symboliques essayent de reconnaître des séquences d'actions complexes.

On remarquera que souvent les premiers systèmes se revendiquent naturellement du domaine de la reconnaissance d'activités et les seconds de celui de la reconnaissance de plans.

En effet, bien que la reconnaissance d'activité soit souvent considérée comme l'application de techniques de reconnaissance de plans à la surveillance d'activités humaines, il n'empêche qu'en pratique, les problématiques engendrées par le changement de nature des données d'entrées en font un problème tout autre.

Ainsi, les systèmes ayant pour but de reconnaitre des comportements complexes, et souhaitant travailler avec des observations provenant du monde réel sont souvent dotés de deux sous-systèmes, l'un extrayant des évènements symboliques à partir des capteurs, l'autre inférant des hypothèses sur les comportements possibles pouvant être associés à l'entité surveillée à partir des évènements générés.

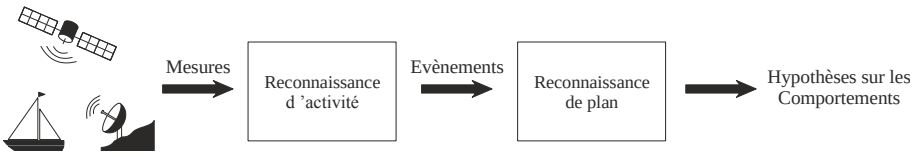


**Figure 1-1 Décomposition d'un système de reconnaissance de comportement**

Nous nous pencherons sur les problématiques engendrées par cette décomposition en deux systèmes par la suite, avant de proposer notre solution en une seule étape.

# 1.3 Définitions Préliminaires

Nous allons dans cette section décrire de manière générale un socle commun à l'ensemble des systèmes de reconnaissance de comportement dans le but de pouvoir tisser des liens entre les différents modèles ainsi que de pouvoir les comparer. Par la suite, nous serons amenés à enrichir ces définitions afin de prendre en compte des mesures hétérogènes imprécises provenant du monde réel.

## 1.3.1 **Observation**

Une observation $o$ est un symbole représentant la valeur de la perception que l'on a de l'objet sous surveillance. L'ensemble des observations sera noté $\Omega$.

Ex : {sortie_du_port, au_large, au_mouillage, … } peut être un ensemble d'observations relatives à la position d'un bateau. Ces observations sont amenées à être de natures différentes, ainsi que de niveau de détail variable. Il peut en effet s'agir de la position approximative d'un bateau comme de la nature des cargaisons qu'il transporte.

## 1.3.2 **Comportement**

Un comportement $b$ est une séquence d'observations: $b = \{o_1 \cdot o_2 \cdot \ldots \cdot o_n\}$. L'ensemble des comportements $B$ est donc le monoïde libre muni de l'opération de concaténation (représentée par le symbole $\cdot$) et ayant pour base $\Omega$.

Ex : La séquence suivante peut être perçue comme un comportement de pêche : sortie_du_port · navigation_vers_zone_de_pêche · en_zone_de_pêche · navigation_vers_port · entrée_au_port.

### Comportement Observé

Le comportement effectivement observé sera noté $b^* = \{o_1^* \cdot o_2^* \cdot \ldots \cdot o_n^*\}$.

## Bibliothèque de comportements

La **bibliothèque de comportements** $B = \{O_1, O_2, \dots, O_n\}$ est l'ensemble des comportements que l'on cherche à reconnaître.

## Reconnaissance de comportement

La **reconnaissance de comportement** consiste à déterminer si le comportement effectivement observé est présent dans la bibliothèque de comportements :

$$b^* \in B \, ?$$

La reconnaissance de comportement **en ligne** consiste à déterminer **à tout moment** le sous-ensemble des comportements de la bibliothèque de comportements dont le début est identique avec le comportement effectivement observé.

$$Soit\ b^* = \{o_1^* \blacktriangleleft o_2^* \blacktriangleleft \dots \blacktriangleleft o_n^*\}$$
$$\exists ?\, b = \{o_1 \blacktriangleleft o_2 \blacktriangleleft \dots \blacktriangleleft o_n \blacktriangleleft \dots \blacktriangleleft o_i\} \in B \mid \forall j \in \{1,2,\dots,\mathrm{n}\}, o_j^* = o_j$$

## Classe de comportements

Une **classe de comportements** $C$ est un label (*i.e.* une valeur symbolique) associé à un sous-ensemble de comportements de la bibliothèque de comportements.

## Modèle génératif de comportement

Un **modèle génératif de comportement** est composé d'une part d'un **formalisme de représentation** de connaissances permettant de spécifier un ensemble de comportements sans avoir à tous les expliciter, et d'autre part d'un **ensemble d'algorithmes** permettant de **générer** ce sous-ensemble de comportements et de **reconnaître** l'appartenance d'un comportement à ce sous-ensemble.

On associe fréquemment à une instance de modèle génératif de comportement le label correspondant à la classe de comportements du sous-ensemble spécifié.

Il est parfois possible que le sous-ensemble des comportements spécifiés ne soit pas fini, auquel cas, l'algorithme génératif prendra un temps infini. Ceci n'est pas forcément problématique car l'on s'intéresse ici à la reconnaissance en ligne de comportement.

# 1.4 Problématiques étudiées dans cette thèse

Les problématiques aujourd'hui identifiées dans la communauté de la reconnaissance de comportements sont multiples.

## 1.4.1 Choix du modèle

Tout d'abord, la première tâche inhérente à tout système de reconnaissance (dynamique ou non) de comportement est de définir la bibliothèque de comportements. Comme nous l'avons précédemment précisé, cette dernière est la plupart du temps spécifiée en intention à l'aide d'un modèle génératif de comportement. Le choix de ce modèle est donc crucial car il détermine le degré d'expressivité avec lequel nous pourrons spécifier les relations entre les observations.

## 1.4.2 Gestion de l'ambigüité

La deuxième problématique est de gérer le nombre souvent élevé d'hypothèses produites par le système de reconnaissance. Dans la plupart des approches, une solution stochastique est retenue, éliminant les hypothèses les moins probables. Pourtant, cette approche présente bien des défauts comme celui d'éliminer des hypothèses certes peu probables, mais pourtant décisives du point de vue de l'application. Dans le cadre de détection de comportements anormaux, ces derniers sont par nature peu probables.

## 1.4.3 Prise en compte de mesures numériques imprécises

Il est évident qu'il sera difficile de faire correspondre les définitions précédentes (notamment celle d'observation) directement avec des ensembles multi-dimensionnés de mesures hétérogènes, pouvant arriver simultanément ou non et provenant d'un système de capteurs imparfaits. Parmi les solutions communément admises se trouvent la discrétisation, le filtrage, ou encore la mise en place d'un système complet de détection d'événements souvent associé à de la reconnaissance d'activité.

### 1.4.4 **Exploitation des résultats de la reconnaissance**

Etant donné notre cadre applicatif, les techniques de reconnaissance de comportement seront utilisées dans un but de production d'informations. Ces dernières seront utilisées par un humain et devront lui fournir des indications nécessaires à une prise de décision critique. L'intelligibilité et la fiabilité des données produites par notre système seront d'une importance capitale. Les résultats devront être justifiés et explicités.

### 1.4.5 **Fiabilité du système**

Nous avons préalablement spécifié que parmi les différents domaines d'application visés se trouvaient être des applications militaires. Dans ce contexte, la prise en compte de l'opérateur humain au centre du processus décisionnel est impérative. Ainsi, associer une classe de comportements à une entité ne sera pas suffisant, surtout s'il ne s'agit comme l'on peut s'y attendre d'hypothèses. Il sera nécessaire de lui fournir des justifications ou faisceaux de preuves lui permettant de corroborer les résultats du système.

Pour pouvoir accorder sa confiance, un opérateur devra être en mesure de confirmer les hypothèses émises par le système en comparant les différentes données produites à sa propre expertise de la situation.

### 1.4.6 **Système idéal**

En résumé, nous cherchons à obtenir des méthodes nous permettant à partir :

- D'un flux de mesures issues de capteurs
  - Hétérogènes
  - Imparfaits (mesures imprécises et manquantes)
  - Nombreux
  - Redondants

- De connaissances humaines provenant d'experts du domaine de l'application ciblée

A obtenir des résultats, automatiquement raffinés au cours du temps, concernant les comportements caractéristiques que les entités sous

surveillance sont susceptibles d'exhiber. Sachant qu'il sera nécessaire que ces résultats puissent :

- Etre compréhensibles par un opérateur humain.

- Exhiber les hypothèses posées par nos méthodes justifiant ces résultats et pouvant aider un opérateur humain à confirmer les résultats.

- Présenter une certaine forme de prédiction quant aux évolutions possibles de la scène et cela dans le but de permettre une intervention adaptée en cas de d'avènement de situation non-conforme aux prédictions.

L'objectif fixé dans cette thèse est la production de telles méthodes, et leur implémentation dans un système opérationnel permettant à des opérateurs humains de mieux comprendre le comportement de la zone sous surveillance.

Nous montrerons que les modèles purement symboliques proposés dans le domaine de la *reconnaissance de plans* sont insuffisants pour traiter directement des mesures issues de capteurs réels et que les actuelles propositions de conversion de ces mesures en ensemble de symboles sont source de destruction d'informations pourtant nécessaires à la levée d'ambigüités entre différents plans.

D'autre part, nous expliciterons en quoi les modèles actuels utilisés pour la *reconnaissance d'activités* et traitant spécifiquement des mesures numériques imprécises sont inadaptés pour proposer des résultats exploitables (phénomène de boite noire) dans un contexte applicatif où l'opérateur humain est au centre du processus décisionnel. En effet, la possibilité de vérification des résultats obtenus par le biais d'un système automatique est obligatoire pour la prise de décision dans un contexte critique.

Nous allons donc devoir proposer un nouveau modèle, permettant de décrire les connaissances humaines détenues par des experts des domaines d'applications cibles sans pour autant définir un niveau d'abstraction engendrant de la destruction d'information, et qui au travers d'un ensemble d'algorithmes, pourra être exploité pour produire les résultats attendus.

## 1.5 Organisation du document

La première section suivant ce chapitre introductif sera consacrée à la présentation de l'état de l'art (Chapitre 2). Un des objectifs premiers de cette thèse étant l'élaboration d'un nouveau modèle de description des connaissances liant les mesures numériques aux connaissances symboliques de haut niveau. Nous présenterons en premier lieu des modèles classiques utilisés pour la reconnaissance de plans (généralement hautement symboliques) (2.1). Nous tisserons notamment des parallèles entre des modèles classiques tels que les Modèles de Markov Cachés et les Grammaires Formelles Stochastiques Nous discuterons ensuite de la pertinence des propositions actuelles pour relier ces deux modèles et des problématiques engendrées par la décomposition du problème en deux sous problèmes (génération d'évènements et reconnaissance de plans) (2.4). Nous nous attacherons dans cette section décrire des modèles hiérarchiques tels que les Hiérarchical ou Cascading Hidden Markov Models ou encore les Hierarchical Tasks Networks.

Le Chapitre 3 s'attachera à décrire la contribution principale de cette thèse, à savoir un nouveau modèle permettant dans le même paradigme de décrire en intention des comportements complexes, représentant les évolutions possibles et connues des entités sous surveillance. Pour y parvenir, nous enrichirons dans la section 3.2 les définitions préalablement énoncées (1.3). Ensuite nous pourrons décrire les différentes extensions proposées au modèle des grammaires formelles sur lequel nous nous basons dans la section 3.4.

Puis, nous proposerons un ensemble de techniques et d'algorithmes dans le Chapitre 4 nous permettant d'utiliser les connaissances décrites par le biais de notre modèle pour permettre de fournir les informations requises à un opérateur humain à partir d'un flux de mesures provenant d'un système de capteurs. Nous nous attacherons à décrire dans la section 4.1 la richesse des informations désirées et les grandes lignes des algorithmes utilisés pour les produire. Nous nous attarderons ensuite en 4.2 sur les points de détails concernant la gestion de l'incertitude et de l'imprécision relatives aux mesures issues de capteurs. Nous terminerons en rappelant les limites des prérequis imposés par l'utilisation de telles techniques en 4.3.

Nous présenterons enfin dans le Chapitre 5 les différentes expérimentations effectuées nous permettant de valider notre approche. Nous décrirons un prototype de système implémentant nos algorithmes, utilisant notre modèle de description de comportements (BOBAC) et travaillant sur des données issues d'un simulateur ad-hoc dans la section 5.1. Une bibliothèque sommaire mais complète de règles et écrite dans notre modèle sera présentée à titre d'exemple dans la section 5.2.

Nous terminerons ce manuscrit dans le Chapitre 6 par une conclusion présentant les différentes problématiques restant à résoudre, notamment

concernant la reconnaissance de comportements entrelacés et évoquerons également plusieurs applications futures et actuelles de ces travaux.

# Chapitre 2
# Etat de l’art

Nous allons donc nous intéresser dans cet état de l'art en priorité à la description de modèles existants, qui utilisés au travers d'une série d'algorithmes forment l'essentiel des techniques de reconnaissance de comportements.

Nous divisons notre présentation en deux catégories : tout d'abord celle des modèles dédiés à la reconnaissance de plans en ligne, et travaillant à partir de données symboliques (ensemble finis) et permettant de relier des concepts 'humains' dits de haut niveaux ; et ensuite celle des modèles dits de 'bas niveaux' se concentrant sur la reconstitution d'actions 'élémentaires' à partir de données numériques, hétérogènes et bruitées.

Nous terminons ensuite par la présentation de systèmes s'intéressant au même titre que nos travaux à la problématique globale de la déduction de comportements possibles de haut niveau en prenant pour source de données d'entrée les mesures quasi brutes fournies par un système de capteurs réels.

# 2.1 Modèles Evènementiels

La plupart des modèles évènementiels dont nous allons parler par la suite ont été utilisés pour faire face à la problématique de la reconnaissance de plans. Cette problématique a été clairement introduite par (Schmidt, Sridharan, & Goodson, 1978). Nous pouvons également citer (Carver, Lesser, & McCue, 1984) qui ont tenté de formaliser des modèles dédiés à la reconnaissance de plans. Par la suite, (Kautz, 1987) s'est attaché à proposer une théorie de la reconnaissance de plan, en utilisant la logique du premier ordre pour représenter sa base de plans, cependant dans ces travaux des hypothèses fortes sont posées :

- La bibliothèque de plans est complète
- Les entités sont supposées infaillibles

Ces deux hypothèses sont évidemment peu compatibles avec une observation d'entités dans le monde réel.

Nous nous proposons dans un premier temps de présenter quelques modèles basiques qui ont servi de base aux systèmes modernes de reconnaissance de plans.

## 2.1.1 Modèles de Markov Cachés

Une chaîne de Markov est un triplet $< E, P, I >$ où $E$ est un ensemble fini d'états, $P$ est une fonction de transition permettant le passage d'un état vers un autre et $I$ est une distribution de probabilités sur l'état initial. Ici, nous représenterons cette fonction de transition par une matrice de probabilités des transitions de chaque état vers un autre.

### Exemple

Soit un robot dans un environnement 2D discrétisé se dirigeant vers une case objectif. Le robot peut se déplacer sur une case adjacente entre chaque pas de temps ou rester sur place (on admettra qu'il ne se déplace pas en diagonale).

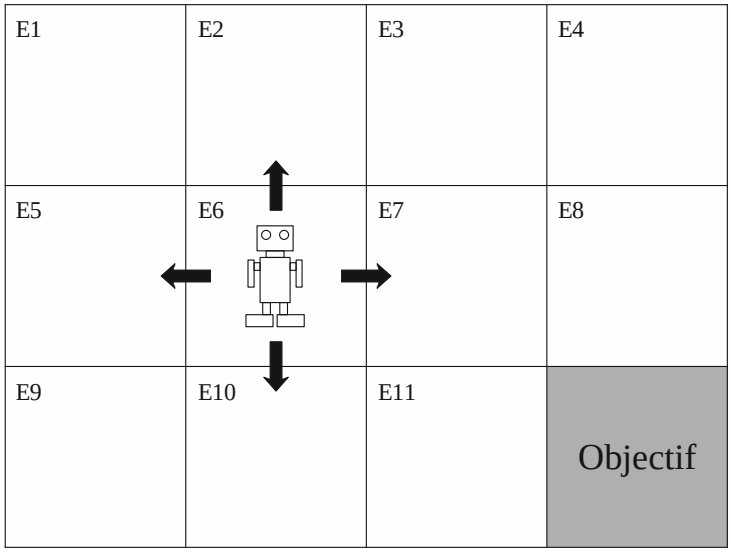


**Figure 2-1 Robot se déplaçant vers une case objectif dans un environnement 2D**

Ainsi, si l'on souhaite représenter le comportement de ce robot par une chaîne de Markov, nous aurons : $E = \{E1, E2, E3, \dots, E11, Objectif\}$, en effet, l'ensemble des états est l'ensemble des positions possibles du robot sur la grille.

La matrice des probabilités de transitions *P* sera de la forme suivante :

| Arrivée / Départ | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | E10 | E11 | Objectif |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **E1** | 0 | 0,5 | 0 | 0 | 0,5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **E2** | 0 | 0 | 0,5 | 0 | 0 | 0,5 | 0 | 0 | 0 | 0 | 0 | 0 |
| **E3** | 0 | 0 | 0 | 0,5 | 0 | 0 | 0,5 | 0 | 0 | 0 | 0 | 0 |
| **E4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **E5** | 0 | 0 | 0 | 0 | 0 | 0,5 | 0 | 0 | 0,5 | 0 | 0 | 0 |
| **E6** | 0 | 0 | 0 | 0 | 0 | 0 | 0,5 | 0 | 0 | 0,5 | 0 | 0 |
| **E7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0,5 | 0 | 0 | 0,5 | 0 |
| **E8** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **E9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **E10** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **E11** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **Objectif** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Tableau 2-1 Matrice des probabilités de transitions entre les états pour le Robot**

Cette matrice a été conçue en fonction des règles suivantes :

- La case *objectif* se trouve en bas à droite de la grille.

- Le Robot reste sur la case *objectif* une fois qu'il y est arrivé.
- Le robot se déplace le plus directement possible vers la case objectif.

Enfin, la distribution $I$ de probabilités sur l'état initial sera :

| $E1$ | $E2$ | $E3$ | $E4$ | $E5$ | $E6$ | $E7$ | $E8$ | $E9$ | $E10$ | $E11$ | $Objectif$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ |

**Tableau 2-2 Distribution de probabilité sur l'état initial du Robot**

Nous pouvons utiliser ce modèle pour prédire les prochains états du robot en fonction d'un état initial à partir du moment où l'état du robot (sa position dans la grille) est observable.

## Etat non directement observable

Plaçons-nous maintenant dans le cas où l'état du robot n'est pas directement observable. Le robot dispose maintenant de capteurs limités, lui permettant uniquement de savoir s'il y a présence d'une case sur laquelle se déplacer sur chacune des directions cardinales et nous n'avons à notre disposition que la valeur de ces capteurs à chaque instant.

Il est possible de représenter ce phénomène par le biais d'un Modèle de Markov Caché (MMC). Ce dernier étend le modèle des chaînes de Markov en introduisant la notion d'observation.

Un MMC est un quintuplet $< E, P, I, O, \pi >$, avec $E$ et $P$ ayant les mêmes définitions que précédemment, $O$ étant l'ensemble de toutes les observations possibles et $\pi$, la fonction d'observation, ici représentée par la matrice des probabilités des états en fonction de l'observation.

En reprenant l'exemple précédent, nous devons encoder l'ensemble des observations possibles par une série de symboles. Il y a 16 observations différentes possibles pouvant être encodées par le tableau suivant :

| **Présence d'une case libre au …** | | | | |
|---|---|---|---|---|
| **Nord** | **Sud** | **Est** | **Ouest** | **Symbole** |
| non | non | non | non | 0000 |
| non | non | non | oui | 0001 |
| non | non | oui | non | 0010 |
| non | non | oui | oui | 0011 |
| non | oui | non | non | 0100 |
| non | oui | non | oui | 0101 |

| | | | | |
|---|---|---|---|---|
| non | oui | oui | non | 0110 |
| non | oui | oui | oui | 0111 |
| oui | non | non | non | 1000 |
| oui | non | non | oui | 1001 |
| oui | non | oui | non | 1010 |
| oui | non | oui | oui | 1011 |
| oui | oui | non | non | 1100 |
| oui | oui | non | oui | 1101 |
| oui | oui | oui | non | 1110 |
| oui | oui | oui | oui | 1111 |

**Tableau 2-3 Choix des symboles représentant la présence de cases libres autour du Robot**

Si nous reprenons l'environnement précédent, alors $\pi$ peut prendre la forme représentée par le tableau ci-dessous.

| Etat / Observation | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | E10 | E11 | Objectif |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0011 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0101 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0110 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0111 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1011 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1101 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1110 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1111 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

**Tableau 2-4 Matrice de probabilités des observations sachant l'état pour le robot.**

On remarque qu'ici, il y ambiguïté pour déterminer l'état dans lequel le robot se trouve en fonction de l'observation dans 6 cas : 0111 peut correspondre à l'état E2 ou E3, 1011 peut correspondre à l'état E10 ou E11 et enfin 1111 peut correspondre à l'état E6 ou E7.

Nous avons représenté jusqu'alors les symboles $P$ et $\pi$ par des matrices, en réalité, ils peuvent prendre la forme de n'importe quelle fonction de probabilités.

## Imprécision d'un capteur

Il est possible de représenter l'effet sur le comportement d'un capteur défectueux ou imparfait. Supposons par exemple qu'une fois sur quatre, le capteur *nord* n'arrive pas à détecter la présence d'un mur, ainsi, seule la matrice $\pi$ sera impactée et prendra la forme suivante :

| Etat / Observation | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | E10 | E11 | Objectif |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0011 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0101 | ¾ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0110 | 0 | 0 | 0 | ¾ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0111 | 0 | ¾ | ¾ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1011 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1101 | ¼ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1110 | 0 | 0 | 0 | ¼ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1111 | 0 | ¼ | ¼ | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

**Tableau 2-5 Matrice de probabilités des observations sachant l'état pour un robot avec un défaut sur le capteur *nord*.**

Sur cette matrice, on observe ainsi que l'état E1 pourra engendrer deux observations : 0101 avec une probabilité de ¾ soit 0,75 ou 1101 avec une probabilité de 0,25. N'oublions pas que comme nous parlons de probabilités, la somme des probabilités d'observations selon l'état (somme des nombres d'une colonne) doit être égale à 1. Il en va de même pour les états E2, E3 et E4, c'est-à-dire tous les états correspondant à des cases de la première ligne.

Nous pouvons également noter qu'ici, le capteur *nord* ne détecte jamais la présence d'un mur s'il n'y en a pas.

## Limitations du modèle

### Conséquences de l'Hypothèse Markovienne

Il est évident que ce modèle très simple basé sur l'hypothèse Markovienne (l'état suivant ne dépend que de l'état courant) implique un grand nombre de restrictions.

Par exemple, il est impossible de caractériser directement un comportement pourtant apparemment simple comme celui de tourner en rond pour le robot.

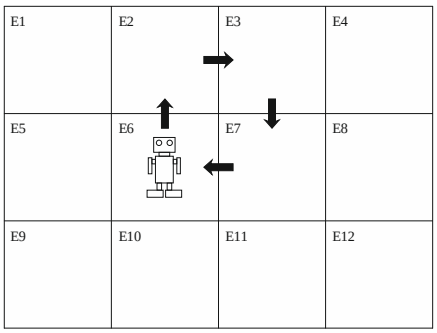


**Figure 2-2 Circuit anti-horaire sur 2 cases de large**

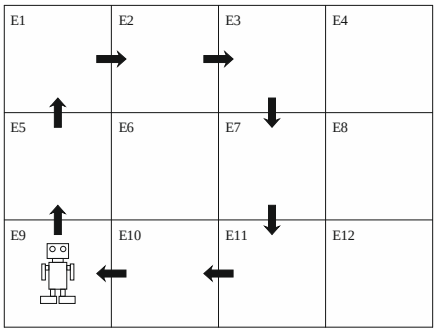


**Figure 2-3 Circuit anti-horaire sur 3 cases de large**

En effet, en conservant comme notion d'états l'identifiant de la case, il est impossible de représenter les comportements des figures ci-dessus car la case suivante dépend de la case courante ainsi que de la case précédente pour la Figure 2-2 et des deux cases précédentes pour la Figure 2-3.

La seule solution pour alors pouvoir représenter ces comportements à l'aide d'un Modèle de Markov Caché est de modifier notre notion d'état, ce dernier devant être un couple ou un triplet, reflétant l'historique des derniers états traversés. Nous sommes donc confrontés au choix suivant : ne posséder qu'une seule notion d'état pour l'ensemble de nos comportements ou avoir une notion d'état propre et optimisée pour chaque modèle de comportement. Le premier choix permet certes une homogénéité plus grande au sein de la base de modèles comportementaux, mais il implique une augmentation du nombre d'états de manière inutile pour les comportements ne nécessitant pas des triplets de cases pour être modélisés. A l'inverse, posséder une notion d'état variable pour chaque modèle de comportement est néfaste quant à la lisibilité de la base de modèles comportementaux. De plus, cela implique un travail d'annotation manuel, pour chaque comportement, de la signification de cette notion d'état.

De plus, quelle que soit la solution retenue, la corrélation entre les états instantanés physiques du robot (numéro de cases) et la notion d'état au sein du Modèle de Markov est perdue. Ceci n'est pas surprenant car le système dynamique du robot exécutant des circuits ne respecte pas l'hypothèse Markovienne.

Dans le cadre de la surveillance de trafic maritime, le nombre de comportements à modéliser est très vaste, et ils sont bien plus complexes que les exemples jouets que nous venons de présenter. L'historique des états physiques d'un bateau (pouvant s'étaler sur plusieurs heures à raison d'environ une observation à la minute) joue un rôle crucial pour déterminer ses états physiques suivants. Les modèles de Markov Cachés sont donc inadaptés pour représenter une grande partie des comportements qui nous intéressent.

### Extensions des Modèles de Markov Cachés

Bien que non directement applicable à la problématique de reconnaissance de comportements dans le monde réel, les MMCs ont été enrichis pour pouvoir représenter des notions de hiérarchies, de décomposition de tâches ou encore des données numériques.

Nous pouvons citer par exemple les Embedded Hidden Markov Models ou

Hierarchical Hidden Markov Models (HHMMs). Ce modèle introduit une notion de hiérarchie dans les HMMs en introduisant deux niveaux de séquences d'états. Les états du niveau supérieur se décomposent eux-mêmes en une séquence d'états de niveau inférieur (Bui, Svetha, & West, Policy recognition in the Abstract Hidden Markov Models, 2002).

Cependant, raisonner de manière exacte dans les HHMMs implique une complexité exponentielle en fonction du nombre d'états, c'est pourquoi il existe des variantes permettant de réduire cette complexité au prix d'une perte possible d'information comme par exemple les Cascading Hidden Markov Models.

Ces derniers sont utilisés dans (Blaylock & Allen, 2006) pour la reconnaissance de plans ayant une forme particulière (Goal Schemas).

Nous pourrions encore énumérer beaucoup d'autres adaptations des HMMs sans pour autant progresser dans notre domaine. En effet, la popularité des HMMs résulte de leur simplicité, autant en terme de représentation que de fonctionnement. Chercher à enrichir ce modèle lui fait bien souvent perdre ce principal atout. Ainsi, si les limites inhérentes aux HMMs représentent un obstacle infranchissable à notre problème, peut-être vaut-il mieux se tourner vers d'autres formalismes de représentation.

## 2.1.2 **Théorie des Langages Formels**

### Alphabet

Un alphabet est un ensemble de symboles (mots).

Ex : {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z, }

## Phrase

Une **phrase** est une succession de symboles.

Ex : zeus a ete a suez

## Langage

Un **langage** est un ensemble de phrases.

Ex : L'ensemble des palindromes.

## Grammaire

Une **grammaire formelle** est un formalisme de représentation de connaissances permettant de spécifier un Langage en intention sans avoir besoin de spécifier chacune de ses phrases.

Ex : $S \rightarrow \varepsilon|a|b|c|\ldots|z|aSa|bSb|cSc|dSd|\ \ldots|zSz.$

Une grammaire formelle est définie par un quadruplet $< N, T, R, S >$ où :

- $N$ : un ensemble de symboles non-terminaux.
- $T$ : un ensemble de symboles terminaux (alphabet).
- $R$ : un ensemble de règles de production.
- $S$ : un ensemble de symboles de départ (axiome) tel que $S \subset N$.

La forme des règles de production varie selon le type de grammaire. Il existe plusieurs classifications des grammaires formelles, la plus connue étant celle proposée par Noam Chomsky (Chomsky, 1965). Un type de grammaire borne l'expressivité de l'ensemble des langages associés, ainsi que la complexité du type d'automate à utiliser durant la phase d'analyse (appartenance d'une phrase à un langage).

Par exemple, si nous souhaitions définir une grammaire $G_{parentheses equilibrees}$ générative de l'ensemble des mots composés d'un ensemble de parenthèses ouvrantes et fermantes équilibrées, mieux connue sous le nom de Langage de Dyck nous ne pourrions y parvenir à l'aide d'une grammaire régulière. En effet, il est nécessaire de recourir à une grammaire hors contexte, nécessitant la mise en place d'un automate à pile a minima pour pouvoir être reconnue.

Exemple de Langage de Dyck :

$$S \rightarrow \varepsilon | TS$$
$$T \rightarrow (S)$$

De manière générale pour reconnaître un langage régulier un automate à états finis déterministe sera suffisant, tandis qu'un automatie à pile sera au minimum nécessaire pour reconnaître un langage hors contexte et enfin un automate linéairement borné.

## Parallèles entre la reconnaissance syntaxique et la reconnaissance de comportement

La reconnaissance syntaxique en théorie des langages formels s'intéresse à répondre à la question suivante :

Etant donné une phrase $p$, appartient-elle à un langage $l$ ?

Ex : Ce livre est-il écrit en français ?

Pour pouvoir répondre à cette question, il est présupposé que l'on possède une grammaire générative du langage $l$ : $G_l$. A l'aide de cette dernière et d'un ensemble d'algorithmes appropriés, il est possible de construire un automate $A_l$ pouvant déterminer l'appartenance ou la non-appartenance de la phrase $p$ au langage $l$.

### Rapprochement des concepts

L'**alphabet** (ensemble des symboles) peut être apparenté à l'**ensemble des observations** possibles pour une scène.

L'**ordre des mots** (séquence spatiale) peut être associé à l'**ordre d'arrivée des observations** (séquence temporelle).

Le **Langage** (ensemble de séquences de mots) peut être mis en parallèle avec l'**ensemble des comportements** que l'on cherche à reconnaître.

Ainsi, une **grammaire formelle** est un formalisme de représentation de connaissances pouvant être utilisé de pair avec un ensemble d'algorithmes pour produire un **modèle génératif de comportements**.

## Grammaire Stochastiques

Nous nous plaçons dans le cas d'une grammaire hors-contexte :

$$< N, T, R, S >$$

Toute grammaire hors-contexte peut s'écrire sous la forme normale de Chomsky. Une grammaire sous forme normale de Chomsky contient uniquement des règles de production de la forme :

$$A \rightarrow BC \text{ ou}$$
$$A \rightarrow \alpha \text{ ou}$$
$$S \rightarrow \varepsilon$$

Avec $A, B, C \in N$, $\alpha \in T$ et $\varepsilon$ le mot vide, tels que S n'apparaît jamais dans la partie droite d'une règle de production.

Dans une grammaire formelle stochastique, la forme des éléments de R diffère. Une *partie droite* est ajoutée à chaque règle de production, séparée de la partie classique par le symbole |. On peut par ce biais associer, à chaque règle de production, une probabilité :

$$A \rightarrow BC \mid 0.5$$
$$A \rightarrow DE \mid 0.5$$

Ainsi, pour qu'une grammaire formelle soit stochastique, il suffit que la somme des probabilités de toutes les règles de production ayant le même symbole en partie gauche soit égale à 1.

De plus, l'élément S représente maintenant une distribution de probabilités indiquant pour chaque élément de N, sa probabilité d'être un axiome (symbole de départ).

### Parallèle avec les Modèles de Markov Cachés

Reprenons l'exemple présenté sur la Figure 2-1. Essayons de représenter une grammaire formelle stochastique traduisant le comportement représenté par le MMC décrit par les Tableau 2-4 et Tableau 2-1. Ainsi, soit une grammaire stochastique :

$$< N, T, R, S >$$

et un MMC :

$$< E, P, I, O, \pi >$$

Tels que l'ensemble des symboles terminaux de la grammaire soit également l'ensemble des observations du modèle de Markov Caché :

$$T = O = \{0001, 0010, \dots, 1111\}$$

Tels que l'ensemble des symboles non-terminaux de la grammaire soit également l'ensemble des états du modèle de Markov Caché :

$$N = E = \{E1, E2, E3, \dots, E11, Objectif\}$$

Tels que l'ensemble des symboles de départ $S$ associé de leur probabilité respective au sein de la grammaire soit également l'ensemble des probabilités $I$ pour chaque état d'être un état de départ :

$$S = I =$$

| $E1$ | $E2$ | $E3$ | $E4$ | $E5$ | $E6$ | $E7$ | $E8$ | $E9$ | $E10$ | $E11$ | $Objectif$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ | $\frac{1}{12}$ |

Alors, l'ensemble des règles de production $R$ de la grammaire sera :

$$R =$$

| Capteurs Parfaits | | Capteur *nord* défectueux | |
|---|---|---|---|
| $E1 \rightarrow 0110\ E2$ | 0.5 | $E1 \rightarrow 0110\ E2$ | 0.375 |
| $E1 \rightarrow 0110\ E5$ | 0.5 | $E1 \rightarrow 0110\ E5$ | 0.375 |
| | | $E1 \rightarrow 1110\ E2$ | 0.125 |
| | | $E1 \rightarrow 1110\ E5$ | 0.125 |
| $E2 \rightarrow 0111\ E3$ | 0.5 | $E2 \rightarrow 0111\ E3$ | 0.375 |
| $E2 \rightarrow 0111\ E6$ | 0.5 | $E2 \rightarrow 0111\ E6$ | 0.375 |
| | | $E2 \rightarrow 1111\ E3$ | 0.125 |
| | | $E2 \rightarrow 1111\ E6$ | 0.125 |
| $E3 \rightarrow 0111\ E4$ | 0.5 | $E3 \rightarrow 0111\ E4$ | 0.375 |
| $E3 \rightarrow 0111\ E7$ | 0.5 | $E3 \rightarrow 0111\ E7$ | 0.375 |
| | | $E3 \rightarrow 1111\ E4$ | 0.125 |
| | | $E3 \rightarrow 1111\ E7$ | 0.125 |
| $E4 \rightarrow 0101\ E8$ | 1 | $E4 \rightarrow 0101\ E8$ | 0.75 |
| | | $E4 \rightarrow 1101\ E8$ | 0.25 |
| $E5 \rightarrow 1110\ E6$ | 0.5 | $E5 \rightarrow 1110\ E6$ | 0.5 |
| $E5 \rightarrow 1110\ E9$ | 0.5 | $E5 \rightarrow 1110\ E9$ | 0.5 |
| $E6 \rightarrow 1111\ E7$ | 0.5 | $E6 \rightarrow 1111\ E7$ | 0.5 |
| $E6 \rightarrow 1111\ E10$ | 0.5 | $E6 \rightarrow 1111\ E10$ | 0.5 |
| $E7 \rightarrow 1111\ E8$ | 0.5 | $E7 \rightarrow 1111\ E8$ | 0.5 |
| $E7 \rightarrow 1111\ E11$ | 0.5 | $E7 \rightarrow 1111\ E11$ | 0.5 |
| $E8 \rightarrow 1101\ Objectif$ | 1 | $E8 \rightarrow 1101\ Objectif$ | 1 |
| $E9 \rightarrow 1010\ E10$ | 1 | $E9 \rightarrow 1010\ E10$ | 1 |
| $E10 \rightarrow 1011\ E11$ | 1 | $E10 \rightarrow 1011\ E11$ | 1 |
| $E11 \rightarrow 1011\ Objectif$ | 1 | $E11 \rightarrow 1011\ Objectif$ | 1 |
| $Objectif \rightarrow 1001\ Objectif$ | 1 | $Objectif \rightarrow 1001\ Objectif$ | 1 |

**Tableau 2-6 Parallèles entre un modèle de Markov Caché et une Grammaire Stochastique**

Il est possible de représenter R par une matrice de taille $Card(N) \times Card(T) \times Card(N)$, indiquant la probabilité d'occurrence d'une règle en fonction du symbole non-terminal de gauche (l'état de départ), du symbole terminal de la partie droite (l'observation émise) et du symbole non terminal de la partie droite (l'état d'arrivée).

NOTATION : La probabilité de la règle $A \to tB$ est notée $R(A, t, B)$, avec $A, B \in N$ et $t \in T$.

Ainsi, il est possible de transformer un Modèle de Markov Caché sous la forme d'une grammaire formelle hors contexte stochastique sous forme normale de Chomsky. Généralisons cette transformation.

On est en droit de se demander ainsi si les deux méta-modèles sont équivalents. A savoir, si pour un ensemble d'états $N$ et un ensemble d'observations $O$ donnés, on peut générer ou reconnaitre les mêmes ensembles de séquences d'observations, et si oui, avec des probabilités identiques.

Soit un MMC tel que $\mathcal{M} = < E, P, I, O, \pi >$, si $\mathrm{C}(\mathcal{M})$ est l'ensemble des comportements pouvant être générés/reconnus par $\mathcal{M}$. Alors, il existe une GFHCS $\mathcal{g} = < N, T, R, S >$ telle que $\mathrm{C}(\mathcal{M}) = \mathrm{C}(\mathcal{g})$ avec :

- $N = E$
- $T = O$
- $S = I$
- $\forall A, B \in N, \forall t \in T,\ R(A, t, B) = P(A, B) \times O(t, A)$

Cependant l'inverse n'est pas vrai, en effet, imaginons une grammaire stochastique hors contexte telle que :

$$\exists t_1, t_2 \in T \ et \ A, B \in N \ tels \ que \ R(A, t_1, B) \neq R(A, t_2, B)$$

Ceci étant en violation de l'hypothèse de Markov (dépendance de l'état suivant en fonction de l'observation effectuée). Nous ne pouvons obtenir un MMC équivalent. Ainsi, pour tout $i,j$, $\{\mathrm{C}(\mathcal{M}_i)\} \subset \{\mathrm{C}(\mathcal{g}_j)\}$ mais $\{\mathrm{C}(\mathcal{g}_j)\} \not\subset \{\mathrm{C}(\mathcal{M}_i)\}$.

## 2.1.3 **Autres modèles non développés**

### Réseaux de tâches hiérarchisés (HTN)

Très utilisés dans des tâches de planification, les Hierarchical Task Network (Erol, Hendler, & Nau, 1994) sont également utilisés dans le cadre de la

reconnaissance de plan. Nous pouvons notamment citer (Adams & Goel, 2007).

### Reconnaissance de chroniques

Utilisé dans le système de reconnaissance d'IxTeT (Dousson & Ghallab, 1994), un modèle de chronique se doit de représenter un schéma de l'évolution d'une partie de la scène observée. La description de ce modèle se fait par le biais de déclarations de contraintes temporelles portant sur des évènements et des assertions sur les différents attributs de la scène. Des systèmes de triggers sont aussi possibles. On peut ainsi formellement définir un modèle de chronique comme un tuple composé :

- D'un ensemble d'instants,
- D'un ensemble de contraintes temporelles entre les instants définissant ainsi le graphe ou treillis temporel de la chronique,
- D'un ensemble de motifs d'évènements qui seront à mettre en correspondance avec les observations issues du monde extérieur,
- D'un ensemble de motifs d'assertions servant à vérifier qu'un certain contexte est vérifié entre certains instants de la chronique,
- D'actions externes que le système déclenchera lors d'une reconnaissance ou lors de la prévision de celle-ci dans un certain délai (focalisation),
- De déductions sous formes d'assertions ou d'évènements permettant au système de reboucler sur lui-même les conséquences d'une reconnaissance.

Le système de reconnaissance de chronique a pour but de faire correspondre les données d'entrées avec les modèles de chroniques décrits par un expert du domaine.

Le modèle permet aussi de décrire une chronique par la composition d'autres chroniques. Ainsi, on peut parvenir à décrire une certaine hiérarchie de chroniques. Cependant une hypothèse relativement forte est faite dans ce modèle : il n'y peut avoir de récursivité au sein de ce principe de description par décomposition. En effet, une chronique ne peut être décrite en partie par elle-même ou bien par d'autres chroniques l'utilisant dans leurs descriptions.

## 2.1.4 **Utilisation de systèmes symboliques avec des capteurs réels**

Comme nous pouvons le pressentir, confronter les techniques ci-dessus au monde réel n'est pas chose aisée. Cette difficulté provient principalement du

fait de l'ensemble de symboles terminaux (dans le cas des Grammaires Formelles) ou d'observations (dans le cas des Modèles de Markov Cachés). En effet, il est difficile de faire le lien entre un ensemble de mesures hétérogènes provenant d'un système de capteurs et un ensemble fini monodimensionnel de symboles.

Ainsi, une étape de transformation est bien souvent utilisée pour générer des évènements ou observations compatibles avec les modèles précédemment cités.

Nous allons ici nous attarder à décrire les travaux proposés dans (Avrahami-Zilberbrand, Kaminka, & Zarosim, Fast and complete plan recognition: Allowing for duration, interleaved execution, and lossy observations, 2005). Nous y retrouvons une classique décomposition en deux sous-systèmes. Un module de reconnaissance de plan et un module de transformation de données multi-dimensionnées en un ensemble de symboles. Le nom de ces derniers doit présenter une sémantique forte pour pouvoir facilement être interprété par un humain. Par exemple, dans le cadre de robots jouant au football comme présenté dans leurs travaux, les symboles peuvent être :

*with ball, without ball, position, kick, ...*

En opposition, des symboles de plus haut niveaux (pouvant être décomposés en sous symboles) sont également utilisés :

*turn, score, attack, defend, ...*

Le premier module ne travaille ainsi qu'à partir d'un ensemble de symboles. Il s'agit d'un modèle hiérarchique pré-déployé comme le présente la figure ci-dessous. Dans cette dernière, est proposé un exemple de bibliothèque de plan en relation avec un joueur robot de foot.

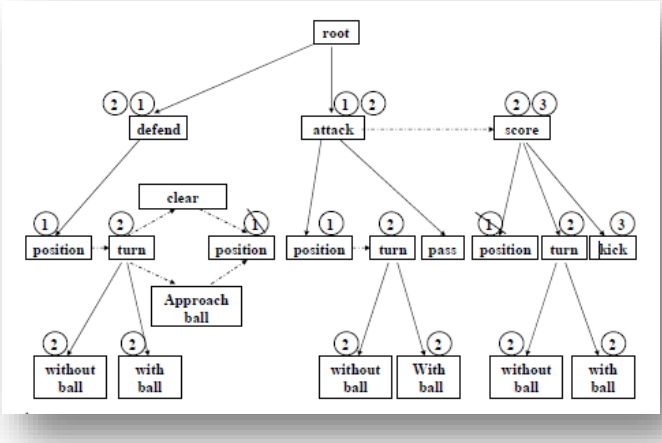


**Figure 2-4: Exemple de bibliothèque de plan utilisée par le module de reconnaissance de plan de (Avrahami-Zilberbrand & Kaminka, Fast and Complete Symbolic Plan Recognition, 2005)**

Les symboles terminaux, représentant les feuilles de l'arbre ci-dessus, doivent donc être inférés à partir des données multi dimensionnées issues des capteurs.

Pour réaliser cette tâche, il est proposé de construire un *Feature Decision Tree* permettant de rapidement transformer un tuple de données mesurées en symbole terminal. Les feuilles/classes de ce FDT sont les symboles terminaux de notre arbre.

De plus la construction automatique du FDT est effectuée à partir de l'arbre représentant la bibliothèque de plans. La traduction des observations en étape de plan est obtenue en traversant de haut en bas le FDT, le fils sélectionné étant déterminé par la vérification des conditions présentes sur les nœuds du FDT.

Le principal avantage lié à l'utilisation d'un FDT construit en fonction de la bibliothèque de plans est l'efficacité du système de conversion et la possibilité de gérer l'imprécision ou l'ambigüité. En effet, plusieurs étapes de plans peuvent être déduites d'une même observation. Il s'en suit alors plusieurs hypothèses de reconnaissances. Le suivi de ces hypothèses est simplement effectué en annotant l'arbre représentant la bibliothèque de plans tel que ci-dessus. En effet, les numéros (1, 2 et 3) visibles sur la figure ci-dessus représentent l'appartenance d'une étape de plan à une hypothèse (numérotée) de reconnaissance.

Par la suite, ces mêmes auteurs ont adressé les problématiques de classements des hypothèses (Avrahami-Zilberbrand & Kaminka, Hybrid Symbolic-Probabilistic Plan Recognizer: Initial steps, 2006), ainsi que la possibilité de reconnaître une mise en pause de plans, ou l'enchevêtrement de ceux-ci (Avrahami-Zilberbrand, Kaminka, & Zarosim, Fast and complete plan recognition: Allowing for duration, interleaved execution, and lossy observations, 2005). Enfin, a été proposé la possibilité d'incorporer à la reconnaissance une notion d'utilité ou pertinence subjective à l'observateur humain pour orienter le processus de reconnaissance (Avrahami-Zilberbrand & Kaminka, Incorporating observer biases in keyhole plan recognition (efficiently!), 2007).

Nous avons trouvé l'ensemble de ces travaux plus que pertinents pour notre problématique et ces derniers ont grandement influencés nos travaux. Cependant, deux principales problématiques auxquelles nous sommes soumis ne sont pas abordés dans les précédentes recherches. En effet, n'est pas abordé le mécanisme permettant de situer le niveau d'abstraction *approprié* du symbole terminal. Hors, il nous semble qu'une des principales difficultés réside dans la définition de cette bibliothèque de symboles terminaux. Nous argumenterons par la suite en faveur de la non-définition de cette couche d'abstraction arbitraire.

Enfin, tous les algorithmes proposés (et notamment concernant la reconnaissance des plans enchevêtrés) reposent sur l'hypothèse d'une bibliothèque de plans complètement déployée en mémoire. Ce qui nous le verrons est utopique dans notre cadre applicatif.

## 2.2 Fusion de données

Le domaine privilégié concernant l'extraction de connaissances à partir de systèmes de capteurs est celui de la Fusion de donnée. Ce domaine ne s'intéresse pas de manière spécifique à la reconnaissance de motifs spatio-temporels comme nous le souhaitons dans notre application. En effet, ce domaine s'attache principalement à distinguer les différentes couches de traitement que l'on peut appliquer à des données en vue d'obtenir une information synthétisée, analysée ou enrichie. C'est ainsi qu'aujourd'hui les différents niveaux de traitements de l'information dans le monde de la fusion de donnée peuvent être distingués en plusieurs niveaux d'abstraction :

- Level 0: Source Preprocessing/subject Assessment
- Level 1: Object Assessment
- Level 2: Situation Assessment
- Level 3: Impact Assessment (or Threat Refinement)
- Level 4: Process Refinement
- Level 5: User Refinement (or Cognitive Refinement)

Cette répartition des tâches a été proposée et adoptée par le Data Fusion Information Group, enrichissement des niveaux 0 et 5 le précédent modèle proposé par le Joint Directors of Laboratories.

Si nous souhaitions situer nos travaux par rapport à cette répartition, nous pouvons signaler que nous allons nous intéresser particulièrement aux niveaux un, deux, quatre et cinq.

Nous pourrions ainsi nous intéresser aux modèles et techniques proposées dans les travaux de recherche proposés dans chacune de ces tâches. Cependant cela nous parait être une erreur de chercher à travailler de manière isolée sur chacune de ces tâches.

Bien qu'il soit annoncé que cette répartition est donnée à titre indicatif uniquement et que la frontière entre certains de ces niveaux peut-être parfois floue, nous pensons que raisonner à partir d'un tel modèle peut être source d'inefficacité et que bien souvent les mêmes algorithmes et modèles se retrouvent d'un niveau à l'autre et que seul la destination du résultat de l'utilisation de ces derniers permet de situer un traitement dans une couche ou l'autre.

De plus, raisonner à partir d'un tel modèle peut donner de fausses idées sur la précédence ou la dépendance de ces tâches, ce qui ne se vérifie pas pour certaines tâches de fusion de donné.

Ainsi, certes nous pouvons affirmer que notre problématique peut se retrouver relever d'une problématique liée à la fusion de données, mais cependant, nous réfutons les grands principes de ce domaine quant à nos hypothèses de travail pour ne pas écarter des pistes dans lesquelles se verraient entrelacées des tâches issus de plusieurs des niveaux traditionnellement segmentés.

En effet, nous mettons en avant dans nos travaux le fait d'éviter de séparer les niveaux de traitement de l'information ***a priori***. Pour nous, à chaque application son niveau de découpe particulier. Ainsi, réaliser une catégorisation des tâches générale ne peut que conduire à dénaturer la particularité de chaque problématique applicative. De plus et nous n'aurons de cesse de le rappeler, la spécification des connaissances doit être indépendante du traitement et de l'utilisation que l'on souhaite en faire. Une classification préétablie des traitements possibles pourrait avoir un impact négatif sur la liberté d'écriture des connaissances par l'imposition d'un *Domain Specific Language* non adapté.

# 2.3 Reconnaissance de comportement multi-entités

De par la nature de notre problématique applicative, nous allons être amenés à devoir reconnaître non pas seulement des comportements possibles qu'une entité peut exhiber, mais également des comportements de groupes d'entités comme par exemple, deux bateaux en mer se rencontrant à un point de rendez-vous, ou encore des manœuvres militaires effectuées par un ensemble de navires ou enfin des activités industrielles ne pouvant être effectuées qu'avec une flotte de bateaux ayant chacun un rôle spécifique comme la prospection sismique.

Les difficultés liées au passage à un modèle multi-entités ont été relevés par (Castelfranchi & Rino, 1995). L'une des principales problématiques soulevées est la reconnaissance de comportements d'agent prenant en compte le comportement d'autres agents, ainsi nous devrions pouvoir disposer d'un modèle permettant de modéliser le mécanisme de reconnaissance d'une entité sous surveillance. Nous ne nous attaquerons pas dans nos travaux à ce passage au niveau méta de la reconnaissance de comportements.

Nous nous sommes ainsi intéressés à des travaux s'attaquant à cette problématique, mais pour rester pragmatique, nous ne chercherons pas à modéliser les processus cognitifs pouvant être mis en œuvre par les agents sous surveillance. Parmi ces derniers peuvent être cités (Sukthankar & Sycara, Automatic recognition of human team behaviors, 2005) et (Joo & Chellappa, 2006). La problématique applicative de ces derniers est l'application de technique de reconnaissance de comportements anormaux dans le cadre de la surveillance des zones critiques d'un aéroport. La première proposition concerne le parking d'un aéroport.

**Figure 2-5 : Surveillance vidéo du parking d'un aéroport et du comportement exhibé par les usagers**

Mais ils s'intéressent également à la surveillance du tarmac et des activités du personnel. En effet, reconnaître des comportements anormaux dans ces zones d'activités est critique et la remontée de ce type d'information à un humain est essentielle.

**Figure 2-6 : Surveillance de l'embarquement de passagers à bord d'un avion et du déchargement d'un avion sur le tarmac d'un aéroport.**

Pour parvenir à cela, un modèle de grammaire formelle est utilisé. Le modèle est enrichi par l'utilisation d'une *probabilité* pour chaque règle ainsi que d'un ensemble d'attributs et de conditions booléennes sur ces derniers. Voici par exemple l'ensemble des règles nécessaires à la reconnaissance de l'activité *se garer*.

| Production rules | P | Attribute rules & Semantic conditions |
|---|---|---|
| $S \rightarrow CASEVEHICLES_N$ | 1.0 | |
| $CASEVEHICLES \rightarrow appear_0\ CASING_1\ disappear_1$ | 0.5 | (isPerson(appear.class) ∧ ~isInside(appear.loc, ParkingLot) ∧ ~isInside(disappear.loc, ParkingLot)) |
| $CASEVEHICLES \rightarrow DRIVEIN_0\ CASING_1\ DRIVEOUT_1$ | 0.3 | |
| $CASEVEHICLES \rightarrow appear_0\ CASING_1\ DRIVEOUT_1$ | 0.1 | (isPerson(appear.class) ∧ ~isInside(appear.loc, ParkingLot)) |
| $CASEVEHICLES \rightarrow DRIVEIN_0\ CASING_1\ disappear_1$ | 0.1 | (~isInside(disappear.loc, ParkingLot)) |
| $DRIVEIN \rightarrow appear_0{}^1\ DSTOP_1\ appear_N{}^2$ | 1.0 | (isVehicle($appear^1$.class) ∧ isNear($appear^2$.loc, DSTOP.loc) ∧ isPerson($appear^2$.class)) |
| $DSTOP^1 \rightarrow start_0\ stop_1\ DSTOP_1{}^2$ | 0.9 | $DSTOP^1$.loc := $DSTOP^2$.loc |
| $DSTOP \rightarrow start_0\ stop_1$ | 1.0 | DSTOP.loc := stop.loc |
| $DRIVEOUT \rightarrow stop_0\ disappear_1\ start_N\ DEXIT_3$ | 1.0 | (isNear(disappear.loc, start.loc)) |
| $DEXIT \rightarrow stop_0\ start_1\ DEXIT_1$ | 0.9 | |
| $DEXIT \rightarrow disappear_0$ | 1.0 | |
| $CASING \rightarrow start_0\ stop_1\ start_1\ CASING2_1$ | 1.0 | CASING.mindist:=min(stop.dist,CASING2.mindist) (isSmall(CASING.mindist)) |
| $CASING2^1 \rightarrow stop_0\ start_1\ CASING2_1{}^2$ | 1.0 | $CASING2^1$.mindist := min(stop.dist, $CASING2^2$.mindist) |
| $CASING2 \rightarrow stop_0\ start_1$ | 1.0 | CASING2.mindist := stop.dist |

**Figure 2-7 : Bibliothèque de règles nécessaires pour décrire l'activité *garer un véhicule.***

Nous pouvons observer que le modèle de règle, en plus de s'être vu associé un ensemble de conditions et une probabilité d'occurrence voit ses symboles annotés de chiffres à l'exposant ou à l'indice.

Les chiffres à l'exposant sont présents pour discriminer plusieurs symboles identiques situés de part et d'autre de la règle de production, dans le but de pouvoir correctement identifier l'appartenance de chaque attribut utilisé dans les conditions de cette règle.

Les chiffres en indice, eux permettent d'indiquer le numéro d'observation attendu à partir du premier symbole de la règle. Ainsi, deux symboles successifs mais indiquant le même chiffre en indice signifient que ces derniers doivent être observés simultanément.

Concernant l'aspect multi-entités, on peut observer qu'un prétraitement est réalisé, leur permettant d'isoler au sein du flux vidéo des entités séparées comme les voitures, humains, bagages, avions, … Dans un second temps est associé à chacune de ces entités un ensemble d'attributs comme la position, la taille, la vitesse, … On peut évaluer ici l'influence d'une modélisation objet qui se retrouve également dans l'écriture des conditions (appear.loc, appear.class …).

On peut donc noter ici que la distinction entre entités est réalisée par un deuxième système non décrit. Par contre au sein de la grammaire, aucune différence n'est réalisée entre une règle concernant plusieurs entités ou une règle n'en concernant qu'une seule.

Ce principe d'avoir en amont un système permettant d'identifier des entités distinctes est un prérequis que nous adopterons dans une certaine mesure et qui a été repris dans d'autres systèmes proposés du suivi multi-entités tels que (Kaminka, Pynadath, & Tambe, 2002).

Bien qu'intéressant, nous ne sommes pas en accord avec ce modèle pour plusieurs raisons.

Tout d'abord, ce dernier fait la distinction entre les attributs synthétisés et les attributs hérités. De notre point de vue, une règle et les attributs associés à ses symboles doit être indépendante de son usage, et la valeur de ces derniers doit pouvoir être déterminée dans les deux sens, voire pouvoir être raffinée au cours du temps.

Par exemple, nous pourrions souhaiter écrire une règle $Distance$ dotée de trois attributs $entité1$, $entité2$, et $distanceEuclidienne$ dont l'utilisation pourrait être double, à savoir de calculer la distance entre les deux entités, auquel cas l'attribut synthétisé serait $distanceEuclidienne$, et les attributs synthétisés seraient $entité1$ et $entité2$.

Mais nous pourrions également vouloir utiliser cette même règle pour trouver deux entités parmi un ensemble qui soient à une distance spécifique l'une de l'autre. Les attributs synthétisés et hérités seraient ici inversés. Du fait de cette possible double utilisation, il nous semble important de ne pas avoir à préciser dans une règle si un attribut est synthétisé ou hérité, car selon l'utilisation de la règle, il pourrait être l'un ou l'autre.

Enfin, le modèle repose sur un ensemble de fonction booléenne telles que :

*isInside(), isNear(), isPerson(), ...*

Ces dernières ne sont pas décrites dans la grammaire et leur définition est réalisée en utilisant donc un autre modèle. Il ne fait pas de doute ici qu'il s'agisse de fonctions écrites dans un langage de programmation externe au modèle. Ceci pourrait être tolérable à condition qu'il ne s'agisse que de fonctions utilitaires, n'ayant pas de rapport direct avec le domaine d'activité, mais tel n'est pas le cas. Ici, nous avons bien une partie de la sémantique du comportement qui se trouve écrite dans un langage externe à la bibliothèque de comportement.

Cet aveu d'impuissance remet en question la pertinence de l'utilisation d'un modèle formel issu de la théorie des langages. En effet, pourquoi ne pas représenter ce système de règles comme un ensemble de classes et de méthodes dans un langage objet traditionnel ? Cette question est au cœur de tout *Domain Specific Langage* commençant à devenir trop généraliste. En effet, à partir de quel moment le DSL cesse d'en être un et devient un langage de programmation généraliste ? La critique ici adressée ici vise le fait que le langage de règle a besoin d'un modèle de programmation généraliste pour pouvoir exister (fonction appelées en partie droite de règles). On aurait aimé pouvoir détenir la totalité de la sémantique du domaine dans le seul modèle de règle énoncé, ce que nous proposerons par la suite. En effet, selon nous, si un langage est proposé pour représenter la connaissance liée à un domaine, il est nécessaire que toute la connaissance puisse s'y retrouver représentée.

On peut noter également que la problématique applicative de la surveillance automatique ou semi-automatique des activités menées sur le tarmac d'un aéroport peut se retrouver chez d'autres travaux, tels que (Carter, Young, & Ferryman, 2006). Ces derniers cependant utilisent une approche hybride utilisant Modèles de Markov Cachés et Réseaux Bayésiens comme bibliothèque de plans. Nous tenons à rappeler ici une des contraintes nous empêchant d'avoir recours à ce type d'approche souvent lié à l'apprentissage artificiel et étant le manque de base d'exemple ou encore l'importante nécessité de devoir proposer une information sémantique riche en résultat de reconnaissance à un opérateur humain.

# 2.4 Problématiques liées à la division en deux sous-systèmes

Comme précédemment énoncé, la grande majorité des systèmes de reconnaissance de comportements travaillant avec des données issues du monde réel se décomposent de la manière suivante :

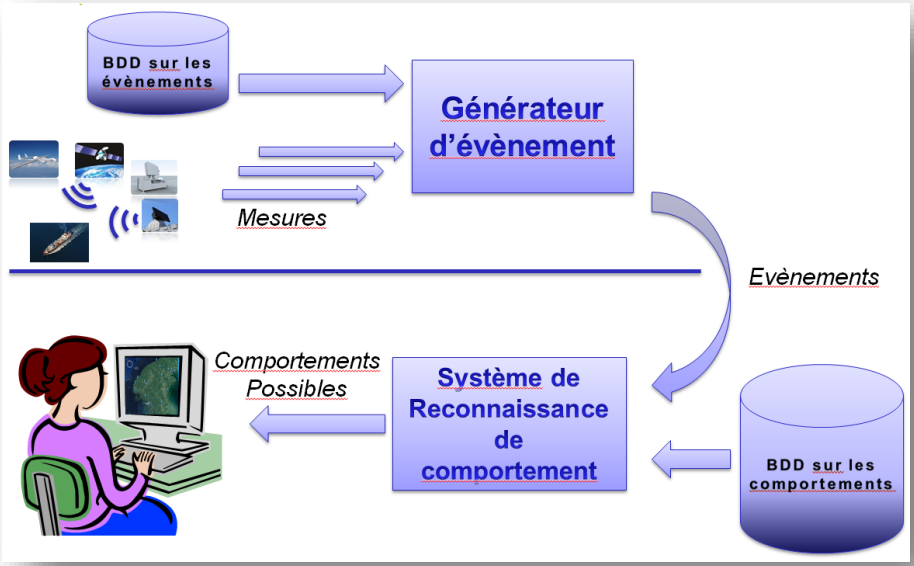


**Figure 2-8 Vue Classique d'un système de reconnaissance de comportement traitant des données réelles**

Deux étapes sont ainsi opérées pour fournir des informations à un opérateur humain, une première consistant à transformer le flux continu de mesures arrivant en entrée en un flux d'évènements symboliques, et une deuxième transformant le flux d'évènements généré en un ensemble d'hypothèses sur l'ensemble des comportements possibles reconnus.

## 2.4.1 Destruction arbitraire d'information

Nous pouvons constater sur la Figure 2-8 que des données multi-dimensionnées et hétérogènes (issues des mesures) sont projetées dans un espace mono dimensionné symbolique (les évènements). Cette transformation est bien entendu rarement une simple linéarisation (la taille de l'espace de

description des évènements serait bien trop grande et inintelligible). Et ainsi, une destruction ou compression des informations brutes provenant des capteurs est opérée.

Sachant que les capteurs nous fournissant des mesures brutes sur le monde réel sont imparfaits (mesures imprécises et bruitées), et que les comportements que l'on peut être amené à reconnaître sont complexes et ne se distinguent les uns des autres la plupart du temps que sur des détails, il est peu recommandé d'utiliser un modèle de système impliquant une destruction systématique d'information.

D'autre part, nous pouvons remarquer également que cette destruction ayant été prévue à la conception du système, elle sera ainsi commune à l'ensemble des comportements reconnus. Or, Identifier le comportement de transport de marchandises d'un cargo ne nécessite pas le même niveau de détail sur les données d'entrée que de prédire la rencontre de deux voiliers en pleine mer.

Il pourrait être tentant alors de proposer deux systèmes, chacun répondant à une problématique de reconnaissance de comportement particulière. Cependant, la définition du niveau de détail nécessaire à la reconnaissance d'une catégorie de comportement ne peut être définie que par un expert du domaine. De plus ce niveau intermédiaire est à même de changer durant le raffinement et l'exploitation du système. Enfin, il serait dommage de chercher à traiter la reconnaissance de chaque comportement avec un modèle et une méthode différente dans la mesure où une base commune de connaissances les relie.

Ainsi, on remarque que cette notion de *niveau évènementiel* est variable et soumise à modification. Nous montrerons dans les sections suivantes de ce document qu'il est nécessaire de s'affranchir de cette méthodologie classique de décomposition en deux sous-systèmes.

# Chapitre 3
# Un modèle unique reliant les buts aux mesures

# 3.1 Intentions et comportements

Jusqu'ici passé sous silence, nous allons nous intéresser brièvement à la notion de détection d'intentions. Par détection d'intentions, est souvent sous-entendue la problématique de la détermination de ce que souhaite faire l'entité sous surveillance.

En admettant que l'intention soit le but poursuivi par l'entité sous surveillance à un instant $t$ tel que nous le propose le modèle Belief Desire Intention (Bratman, 1987) et sachant que pour réaliser un but, l'entité met en place un plan, nous serions tentés de faire correspondre le plan (succession d'actions exécutées par l'entité) avec la notion de comportement préalablement énoncée. Ainsi reconnaitre une intention s'apparenterait à reconnaître une classe de comportements.

Cependant, non seulement cela nous parait réducteur, mais il est nécessaire de réaliser pourquoi ceci ne fonctionne tout simplement pas.

## 3.1.1 Reconnaissance et exécution

Nous serions ainsi tentés de vouloir apparenter la reconnaissance d'une classe de comportements comme fonction inverse de l'exécution d'une intention. Pourtant ceci n'est valable que dans une seule situation : lorsque l'entité qui reconnait possède les mêmes informations que l'entité qui exécute. Formalisons cette pensée :

Le choix du plan ou sa construction est déterminé au sein d'un agent BDI par ses croyances et les évènements détectés. Les croyances sont par définition propres à l'agent et ne seront jamais accessibles directement à un système de reconnaissance externe. Peut-on y accéder indirectement ?

Dans notre cadre applicatif, l'ensemble des données à disposition de l'entité sous surveillance n'a que peu de rapport avec celles du système de reconnaissance de plan.

Du fait de cette inéquation entre les deux ensembles, les bibliothèques de plans permettant de réaliser certains buts ne peuvent être utilisées telles quelles pour reconnaître ces derniers.

Pour illustrer ceci, nous pourrions prendre le cas d'un changement de bord pour un voilier.

La séquence d'actions à effectuer de la part de l'équipage d'un voilier pour virer de bord en pleine mer pourrait s'illustrer comme ceci :

Ex : $VirerDeBord \rightarrow AmenerLeBateauAuPres \triangleleft VirerJusquacequelaVoileFaceille \triangleleft LarguerEcouteBordee \triangleleft EmbraquerEcouteOpposee.$

Du côté d'un observateur situé dans un avion de patrouille de maritime, la description d'un changement de bord pourrait être décrite telle que :

Ex : $VoilierVireDeBord \rightarrow VoilierAuPrèsBabord \triangleleft VoilierFaceAuVent \triangleleft VoilierAuPrèsTribord.$

Ceci peut s'expliquer par deux raisons. Tout d'abord la différence de point de vue entre l'observateur et l'exécutant induit une différence de granularité dans la décomposition des actions. D'autre part, les marqueurs clefs indiquant la bonne exécution d'une action sont de nature différente selon le point de vue.

Ainsi, ici nous recommandons de ne pas nous attacher à chercher à décrire les comportements du point de vue de l'exécutant en espérant pouvoir ensuite réutiliser cette bibliothèque pour la reconnaissance de ces mêmes actions. Nous pensons à l'inverse qu'une bibliothèque de comportement utilisable pour des tâches de reconnaissance doit avoir été définie avec cette finalité à l'esprit et non de penser pouvoir l'utiliser de manière réversible.

La décomposition temporelle d'un comportement ne doit pas être dictée par l'enchainement d'actions permettant sa réalisation, mais bien par l'ensemble des états observables successifs le caractérisant.

### 3.1.2 **Déterminisme dans le choix de l'intention**

Un autre argument que nous pouvons avancer dans le choix de ne pas adopter un modèle agent-centré pour l'élaboration de notre bibliothèque de comportements est la problématique de la rationalité de l'entité sous surveillance.

En effet, pour pouvoir décider de la pertinence de construire une bibliothèque de comportement se basant sur le modèle de fonctionnement des entités sous surveillance, il peut être intéressant de se pencher sur le concept de choix d'action, semblant au cœur du mécanisme de planification que nous prêtons aux agents sous surveillance.

Une hypothèse communément admise pour représenter le mécanisme décisionnel d'un agent est celui de l'agent rationnel doté d'une perception limitée. Ainsi pour pouvoir espérer prédire les futures actions d'une telle entité, il est nécessaire de représenter les croyances de l'agent sous surveillance.

Cependant, l'intérêt de représenter ces croyances n'a de sens que si ces dernières sont exclusivement à l'origine de l'adoption d'un but par l'agent.

Dans notre cadre applicatif, nous surveillons des humains. Les comportements de ces derniers ne sont pas réputés pour être rationnels, et ce d'autant plus dans le cadre d'une activité de loisir telle que la navigation de plaisance.

Nombre d'actions sont effectuées sous l'effet de pulsions et d'envies, dont l'être humain n'a souvent pas conscience et qui sont loin de contribuer à la réalisation d'un but quelconque. Bien entendu il sera toujours envisageable de produire une explication après coup, de rationnaliser l'adoption de telle ou telle attitude, mais aucun raisonnement rationnel n'est à l'origine de bien des comportements.

Notons ici qu'il ne s'agit pas de l'adoption de plans sous-optimaux en lieu et place de plans plus efficaces, mais bel et bien de plans dénués d'objectif quelconque. De même nous ne prétendons pas qu'aucun des comportements exhibés par un humain ne soit rationnel, issu d'un processus cognitif déterministe, mais simplement que nombre d'entre eux ne le sont pas.

De ce constat, nous proposons ainsi un autre argument allant à l'encontre du choix de construire la bibliothèque de comportement comme étant un calque des processus décisionnels intervenant au sein de nos agents sous surveillance.

### 3.1.3 **Comportements multi-entités**

Nous sommes également confrontés à la problématique de représenter et de reconnaitre des comportements mettant en scène plusieurs entités. En effet, dans un modèle BDI, chaque but, plan ou croyance est défini relativement à l'agent auquel il se rapporte. Ainsi, un comportement mettant en jeu plusieurs entités ne peut jamais explicitement être défini. Il est juste le résultat de l'interaction entre plusieurs agents exécutant chacun des plans locaux.

Or, dans notre problématique application, nous sommes amenés à devoir décrire explicitement de telles interactions (ex : rencontre entre deux navires au large, …).

De plus, la granularité choisie pour la décomposition des entités est relative à l'application. Par exemple, les entités d'une zone maritime peuvent être les bateaux, tandis que les entités d'un corps humain sous surveillance médicale peuvent être les différents organes. Enfin, au sein de la même application, nous pouvons imaginer que le concept d'entité peut varier selon le comportement exhibé.

Encore une fois, cela nous invite à penser que nous ne pouvons ainsi pas être agent-centré dans la définition de notre bibliothèque ou encore dans la définition d'un langage génératif permettant de spécifier un ensemble de comportements en intention.

### 3.1.4 **Intention et reconnaissance de comportements**

D'autre part, lorsque nous sommes amenés à décrire une hiérarchie de comportements se décomposant en sous-comportements, nous pouvons nous rendre compte qu'en fin de compte tous les comportements dérivent d'une seule et même abstraction. Il en est ainsi lorsque nous utilisons par exemple une grammaire formelle comme modèle génératif de notre ensemble de comportements.

Ainsi, si le but poursuivi correspond au symbole de départ de notre grammaire, nous ne serions qu'en mesure de spécifier que nos entités exhibent l'intention $Behaviour$ à tout instant. A cela peut être répondu qu'il s'agit en réalité des symboles découlant directement du symbole de départ de notre grammaire.

Ex : $Behaviour \rightarrow FaireUneRegate \mid Pêcher \mid FaireUneCroisière \mid TesterLaNouvelleVoile \mid AdmirerLaCote \mid FaireLeTourDUneIle \mid BattreSonRecordDeVitesse$

La règle de grammaire ci-dessus est volontairement mal construite. En effet, nous nous sommes placés dans le cas où naïvement nous construirions la grammaire en fonction des buts possibles pouvant être poursuivis par les occupants d'un voilier. Or, rien n'empêche notre skippeur de tester la nouvelle voile pendant qu'il admire la côte. De même, il est possible qu'au cours d'une croisière, l'on puisse souhaiter battre son record de vitesse à la voile.

La problématique posée ici n'est pas tant celle du changement d'intention ou de comportements entrelacés, mais plutôt le fait que chacun de ces buts soient placés au même niveau dans un modèle hiérarchique.

Or, lors de l'utilisation d'un modèle hiérarchique dans le cadre de la reconnaissance de comportements il est nécessaire de s'attacher à décrire la décomposition de concepts en sous-concepts en respectant au possible une cohérence entre les niveaux d'abstractions choisis.

Ainsi, si pour réaliser le but tester sa voile, il est nécessaire d'effectuer ce test au cours d'une sortie en mer, alors il est nécessaire de mettre le concept de sortir en mer à un niveau plus élevé que celui du test de matériel. Nous pourrions proposer une règle de la sorte :

Ex : $SortirEnMer \rightarrow SortirDuPort \leftarrow AllerAuLarge \leftarrow TesterLaNouvelleVoile \leftarrow RentrerAuPort$

Nous pouvons ainsi observer que la réalisation d'une intention peut intervenir n'importe où dans la hiérarchie des règles de notre grammaire.

Sortir en mer n'est pas ici un but en soi, mais le comportement général observé au cours duquel peut se réaliser le but principal de tester la nouvelle voile. En effet, tester la nouvelle voile ne peut s'opérer que dans un certain contexte, autrement dit, il ne s'agit que d'une étape d'une action plus générale.

Nous observons ainsi qu'une étape de comportement peut être le but principal d'un agent observé. Nous sommes ainsi en droit de nous poser la question suivante : Comment pouvons-nous faire la différence, en tant qu'observateur n'ayant pas accès aux états mentaux internes à notre entité sous surveillance, entre une étape de comportement utilitaire et le but principal que l'agent cherche à accomplir.

Autrement dit, en reprenant l'exemple précédent, comment déterminer laquelle des étapes *AllerAuLarge* ou *TesterLaNouvelleVoile* représente le but de l'agent ?

La réponse est relativement simple. Sans autre information externe à notre bibliothèque, nous ne le pouvons simplement pas, car les deux peuvent être le but effectif poursuivi par notre agent.

D'autre part, cette information ne nous est que d'une utilité relative. Dans le cadre applicatif de la tenue de situation, peu importe le fait qu'une situation indésirable soit accidentelle ou résultant de la volonté d'un agent sous surveillance, nous souhaitons être en mesure de la prédire pour l'éviter quel que soit le cas.

# 3.2 Retour sur les concepts de base

Avant de décrire notre modèle de représentation des connaissances, les définitions suivantes doivent être explicitées.

## 3.2.1 **Mesure**

Une *mesure* est un quadruplet composé des champs nommés suivants :

- Capteur
  - L'identifiant du capteur ayant effectué la mesure.
- Valeur
  - La valeur obtenue, cette dernière peut être numérique ou symbolique.
- Entité
  - L'identifiant de l'entité sur laquelle est effectuée la mesure.
- Propriété
  - La propriété mesurée.

*EXEMPLE : (capteur : **radar01**, valeur : **44**, entité : **Le Belem**, propriété : **cap en degrés**).*
Ceci représente une mesure du cap, effectuée par le capteur nommé radar01, concernant le bateau nommé Le Belem, la valeur de cette mesure étant égale à 44 degrés.

Par ailleurs, et dans la suite de ce document, lorsque l'on souhaitera parler de la valeur d'un des champs d'une mesure $m$, on utilisera la notation suivante $m : < nom\ champ >$
(ex : *m : capteur*, *m : valeur*, *m : entité* ou *m : propriété*).

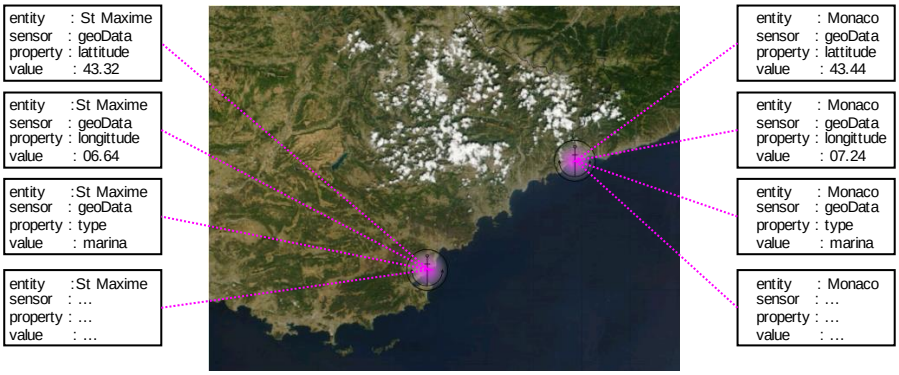


**Figure 3-1 : Exemple de mesures pouvant être récupérées sur les entités d'une zone maritime sous surveillance**

## 3.2.2 **Redéfinition d'une Observation**

Jusqu'à présent, les observations étaient définies comme des symboles. Or, ici une observation, représentant un état de la scène sous surveillance à un instant donné, ne peut être ramenée à un symbole. En effet, à une même entité peuvent être associées plusieurs propriétés mesurables. De même, plusieurs entités sont surveillées simultanément. Ainsi, nous définissons une **observation de la scène** comme l'**ensemble des mesures** prises par tous les capteurs sur toutes les propriétés de toutes les entités à un instant donné.

EXEMPLE : $o_t^* = \{$(capteur : **radar01**, valeur : **44**, entité : **Le Belem**, propriété : **cap en degrés**),
(capteur : **radar01**, valeur : **-1.29**, entité : **Le Belem**, propriété : **longitude**),
(capteur : **radar01**, valeur : **50.23**, entité : **Le Belem**, propriété : **latitude**),
(capteur : **radar01**, valeur : **318**, entité : **Le Vaillant**, propriété : **cap en degrés**),
(capteur : **radar01**, valeur : **-1.6**, entité : **Le Vaillant**, propriété : **longitude**),
(capteur : **radar01**, valeur : **49.6**, entité : **Le Vaillant**, propriété : **latitude**)$\}$.

L'exemple précédent nous montre une observation d'une scène à une date arbitraire, sous la surveillance d'un radar nommé **radar01**, dans laquelle se trouvent deux voiliers. Les radars nous permettent d'obtenir les informations concernant la position (longitude, latitude) et le cap de ces deux entités.


**Figure 3-2 : Scène maritime contenant deux navires de plaisance.**

### 3.2.3 **Observation Partielle**

Etant donné la quantité d'informations contenues dans une observation de la scène, nous définissons une observation partielle de la manière suivante.

Une observation $o_t$ est **partielle** par rapport à une observation $o_t^*$ de la scène $ssi\ o_t \subset o_t^*$.

### 3.2.4 **Séquence d'observations**

Nous sommes dans un cadre de reconnaissance dynamique de comportement. Il s'agit donc de surveiller une scène en temps réel. Au fur et à mesure que le temps s'écoule, le système de capteurs assurant la surveillance de la scène fournit une succession d'ensembles de mesures : **la séquence d'observations**.

L'ordre d'arrivée des observations est supposé cohérent avec la date de chacune des observations.

Par définition, il est supposé que chaque observation de la séquence est **complète**, c'est-à-dire qu'elle contient l'intégralité de toutes les propriétés de toutes les entités mesurables à une date précise. On discutera par la suite de la pertinence et de la faisabilité de cette hypothèse.

### 3.2.5 **Comportement(s)**

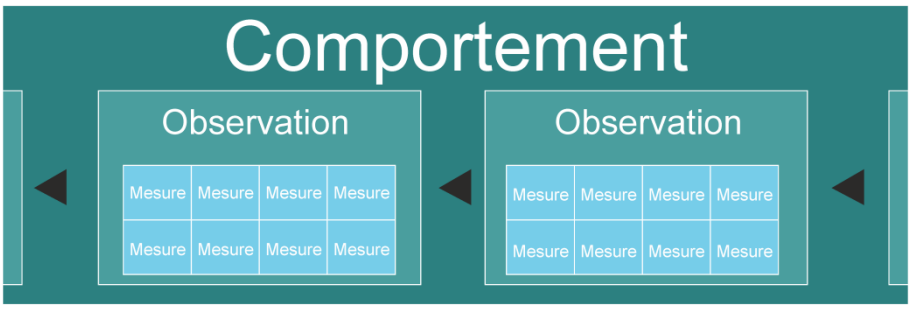


**Figure 3-3 : Nouvelle hiérarchie entre les concepts de comportement, observation et mesure**

Nous avons vu préalablement qu'un comportement $O$ est une séquence observations: $O = \{o_1, o_2, \dots, o_n\}$, cela est toujours le cas, bien qu'il soit nécessaire d'apporter les précisions suivantes :

#### Comportement de la scène

Le comportement global de la scène (il s'agit d'un système dynamique), est la séquence d'observations observée: $O^* = \{o_1^* \cdot o_2^* \cdot \dots \cdot o_n^*\}$.

#### Comportement d'une entité

Nous nous intéressons à des systèmes ouverts. Ainsi, le nombre de dimensions nécessaires, ou nombre de variables, pour décrire l'état du système à un moment donné est susceptible d'évoluer dans le temps (ex : arrivée d'un nouveau bateau dans la zone surveillée). De plus dans les applications concernées, certaines de ces variables peuvent être regroupées pour former les caractéristiques d'une entité appartenant à l'environnement sous surveillance, formalisons ceci.

Un comportement d'une entité nommée $e$ est une séquence d'observations $O^e = \{o_1 \cdot o_2 \cdot \dots \cdot o_n\}$ telle que $\exists m \in o_i \mid m : entite = e$ avec $i \in \{1,2, \dots, n\}$.

En effet, une séquence d'observation contenant au moins une observation comprenant une mesure relative à une entité sera considérée comme un comportement pour cette entité.

L'ensemble des comportements possibles pour une entité $e$ sera noté $\Omega_e = \{O_1^e, O_2^e, \dots, O_n^e\}$

Il peut être remarqué que si l'on considère deux entités $e_1$ et $e_2$ il n'est pas impossible que $\Omega_{e_1} \cap \Omega_{e_2} \neq \{\emptyset\}$. On parlera alors dans ce cas de comportements multi-entités.

## Comportement Cohérent

Un comportement $O$ est **cohérent** vis-à-vis du comportement de la scène $O^*$ si et seulement si :

$$\forall\, o_i \in O \; et \; o_i^* \in O^*, o_i \subset o_i^*$$

Ainsi, un comportement est considéré comme cohérent si toutes les mesures constituant les observations de sa séquence d'observation ont été effectivement mesurées.

Nous pouvons également noter qu'un comportement est considéré comme reconnu s'il est cohérent.

## Comportement partiellement cohérent

Un comportement $O = \{o_1 \cdot o_2 \cdot \ldots \cdot o_k \cdot \ldots \cdot o_n\}$ est considéré comme partiellement cohérent vis-à-vis du comportement de la scène $O^* = \{o_1^* \cdot o_2^* \cdot \ldots \cdot o_k^*\}$ ssi :

$$\forall o_i \in \{o_1, o_2, \ldots, o_k\}, \forall\, m_j \in o_i, m_j \in o_i^*$$

Ainsi, un comportement est considéré comme partiellement cohérent si les $k$ observations du comportement de la scène contiennent les mesures des $k$ premières observations de la séquence d'observation du comportement.

D'autre part, un comportement est considéré comme en cours de reconnaissance s'il est partiellement cohérent.

## 3.3 Choix d'un existant

Plutôt que d'ajouter un énième formalisme de représentation à l'état de l'art, nous avons décidé de partir d'un modèle adapté et de l'étendre selon nos besoins.

### 3.3.1 **Critères**

#### Modèle Génératif

Comme précédemment énoncé, spécifier exhaustivement l'ensemble des comportements possibles vis-à-vis d'une classe de comportements donnée serait irréalisable s'il fallait lister toutes les valeurs de toutes les mesures à tous les pas de temps correspondant à un type de comportement particulier. C'est pourquoi il est nécessaire d'avoir à disposition un modèle génératif que l'on peut paramétrer et qui par le biais d'un ensemble d'algorithmes peut générer/reconnaître tous les comportements appartenant à une classe de comportements que l'on souhaite identifier.

#### Expressivité

Bien que de manière générale plus d'expressivité entraîne une augmentation de la complexité algorithmique autant spatialement que temporellement, il reste essentiel à nos yeux de pouvoir exprimer avec aisance des modèles de comportements qui peuvent être complexes.

#### Hiérarchie

Les entités sous surveillance sont susceptibles d'exhiber des plans complexes. Ainsi, l'interprétation d'une observation à la date $t$ est susceptible d'être dépendante de précédentes observations estampillées à des dates bien antérieures. L'hypothèse Markovienne n'est pas admissible. Une bonne manière de représenter cette dépendance est d'introduire une notion de hiérarchie.

Ainsi, après avoir dressé une liste de comportements types et essayé de les décrire informellement, il apparaît impératif que l'on dispose de relations hiérarchiques entre nos évènements de type « est déclenché par » ou « est composé de ». En effet, à un modèle comportemental donné est associé un évènement père qui est déclenché lorsque la reconnaissance totale du comportement est effectuée. Ainsi la description d'un modèle comportemental

revient à énoncer l'ensemble des règles qui peuvent permettre à l'évènement père associé d'être déclenché.

Nous justifions ce besoin de hiérarchie entre les concepts par deux arguments. Tout d'abord, il apparaît naturel de pouvoir décrire un concept par ce qui le compose. On peut imaginer l'exemple informel suivant :

L'évènement « embarcation suspecte » est déclenché si trois évènements «embarcation évite patrouille de garde-côtes » sont déclenchés consécutivement, l'évènement « embarcation évite patrouille de garde-côtes » est déclenché si les évènements « patrouille de garde-côte se dirige vers embarcation » et « embarcation change de cap » sont déclenchés successivement, et ainsi de suite.

De plus, on peut remarquer qu'ainsi la description de la règle de déclenchement d'un évènement est indépendante de celle des autres évènements, et de ce fait, la réutilisabilité des connaissances ainsi que leur maintenance en est grandement simplifiée.

Les connaissances humaines sont par essence hiérarchisées, il est normal de souhaiter retranscrire cette notion dans nos bases de connaissances. De plus, il serait chimérique d'espérer décrire des comportements complexes concernant directement des mesures multi dimensionnées dans un modèle 'à plat' tout en préservant l'intelligibilité de notre base de connaissance sur les comportements. Ainsi, nous souhaitons opter pour un modèle dans lequel nous pourrons successivement décrire des relations de décompositions (temporelles pour la plupart) nous permettant de lier de manière continue et progressive, des notions de mesures à des concepts tels que le fait d'aller d'un port à un autre.

## Récursivité

D'autre part, nous avons déjà évoqué la notion de récursivité qui fait défaut à certains formalismes. Pourtant il apparaît essentiel d'offrir cette liberté pour définir les comportements qui nous intéressent. Effectivement, de par notre domaine d'étude (la surveillance maritime), nous allons être amenés à décrire des comportements s'étalant temporellement sur des périodes relativement longues.

Durant ces périodes, un navire suivant un comportement donné peut être amené à effectuer plusieurs séquences d'actions répétitives au sein d'un seul et même comportement. On peut prendre l'exemple d'un bateau de transport effectuant un certain nombre d'aller-retour entre deux ports avant d'aller finalement s'ancrer à son port d'attache.

On pourrait opposer à cela qu'il suffit de connaître le nombre d'occurrences de la répétition de la séquence d'actions pour pouvoir décomposer ce comportement sans l'aide d'une description récursive. Mais c'est exactement ici que se situe le problème auquel nous sommes confrontés à savoir que bien souvent cette information est indisponible.

Prenons le cas d'un chalutier qui part pécher jusqu'à ce que ses cales soient remplies ou que sa réserve de carburant soit épuisée. Pour pouvoir déterminer le nombre d'occurrences de manœuvres qu'il doit effectuer avant de rentrer au port, l'observateur devrait avoir connaissance de ces paramètres, ce qui n'est pas notre cas.

## Apprentissage

Nous sommes, dans le cas de notre problème, confrontés à une absence relative de données d'exemples ainsi qu'à une difficulté élevée pour nous en procurer, par contre, nous avons à notre disposition une expertise dont nous pouvons nous servir pour construire notre système.

De plus, construire un simulateur dans le but d'apprendre les règles de simulation elles-mêmes ne présente que peu d'intérêt. En effet, nous pouvons directement nous servir de ces mêmes règles de simulation pour construire notre système de reconnaissance. Ainsi, l'argument de l'apprentissage ne rentrera que très peu en compte dans le choix d'un modèle de représentation.

## Aisance de représentation

Il s'agit ici d'un point crucial de notre approche, en effet, comme nous venons de le définir, les modèles comportementaux de notre système ne seront pas construits par apprentissage, mais en retranscrivant manuellement l'expertise relative à notre domaine d'application. Ainsi, notre système se devra de posséder un formalisme de représentation interne favorisant une saisie facilitée des informations. Il est aussi envisageable de créer une interface permettant de les collecter sans avoir à confronter l'expert du domaine directement au formalisme interne de notre base de données de modèles de comportements que nous aurons choisi d'adopter.

Nous avons déjà notifié la récursivité comme permettant de décrire avec grande facilité des comportements de manière générale dans le cas où nous n'avions pas accès à certaines informations. Mais d'autres mécanismes tels que la séquentialité des évènements sont attendus comme intégrés au système et de ce fait à la modélisation elle-même. En effet, définir qu'un évènement doit être reconnu avant un autre est la base de notre système dynamique, car nous observons une scène évoluant au cours du temps et nous ne devons pas considérer le temps comme un paramètre comme un autre. En effet, nous devons nous servir des spécificités dynamiques de notre problème non seulement pour en réduire la complexité, mais aussi dans le but d'en extraire une représentation adaptée.

# 3.4 Extensions des Grammaires Formelles

Nous avons préalablement présenté un modèle de grammaire formelle stochastique. Nous avons pu notamment voir les concepts de règles, indiquant la décomposition (dans notre cas temporelle) d'un concept en sous-concepts. Nous allons maintenant proposer des extensions à ce modèle pour permettre de prendre en compte directement les notions de mesures préalablement définies au sein des règles de production, et cela sans provoquer une perte d'intelligibilité de notre grammaire. Certaines de ces extensions sembleront déjà préexistantes, mais quelques particularités sont divergentes avec des extensions voisines.

Le modèle de base que nous nous proposons d'enrichir est celui des grammaires formelles hors contexte telles que décrites par (Chomsky, 1965).

Nous allons dans un premier temps énoncer de manière informelle les différents ajouts syntaxiques et fonctionnels qui ont été nécessaires. Nous nous pencherons ensuite sur leurs formalisations et les différentes problématiques théoriques qu'ils engendrent.

Rappelons les éléments définissant une telle grammaire $G$ :

- Un ensemble de symboles terminaux $T$
- Un ensemble de symboles non-terminaux $N$
- Un ensemble de règles $R$ de la forme $A \rightarrow B \cdot a$ avec $A, B \in N$ et $a \in T$
- Un ensemble de symboles de départ $S$ avec $S \subset N$

*EXEMPLE : Soit G une grammaire hors contexte telle que :*

$T$ : $\{auPort, enMer\}$

$N$ : $\{Comportement, SortieEnMer, ResterAuPort, Naviguer\}$

$R$ : $\{Comportement \rightarrow SortieEnMer,$
$Comportement \rightarrow ResterAuPort,$
$SortieEnMer \rightarrow auPort \cdot Naviguer \cdot auPort,$
$ResterAuPort \rightarrow auPort \cdot ResterAuPort,$
$ResterAuPort \rightarrow auPort,$
$Naviguer \rightarrow enMer \cdot Naviguer,$
$Naviguer \rightarrow enMer\}$

S : $\{Comportement\}$

La grammaire ci-dessus nous permet de décrire très succinctement ce que pourrait être un comportement basique de bateau à partir de deux symboles terminaux décrivant son état.

### 3.4.1 **Symboles dotés d'attributs**

Comme l'exemple ci-dessus nous le montre, nous avons besoin pour pouvoir écrire des comportements cohérents, sans pour autant faire croître de manière démesurée nos ensembles de symboles, d'annoter nos symboles par exemple pour spécifier que le port d'arrivée et de départ seraient identiques.

Un problème similaire survient lorsque l'on s'intéresse à la problématique du typage d'un langage de programmation lorsque l'on souhaite écrire un compilateur. C'est pour répondre à cette dernière problématique qu'ont été proposées les grammaires attribuées (Knuth, 1968). Cette extension permet de mêler des informations syntaxiques à des informations sémantiques, ce qui correspond précisément à ce que l'on souhaite pour écrire le comportement ci-dessus. Réécrivons ainsi l'exemple ci-dessus à l'aide d'une grammaire mêlant attributs synthétisés et attributs hérités :

*EXEMPLE : Soit G une grammaire hors contexte attribuée telle que :*

$T$ : $\{auPort, enMer\}$

$N$ : $\{Comportement, SortieEnMer, ResterAuPort, Naviguer\}$

$R$ : $\{Comportement \rightarrow SortieEnMer,$
$Comportement \rightarrow ResterAuPort,$
$SortieEnMer \rightarrow$
$\quad auPort(\boldsymbol{port}: \boldsymbol{Port1}) \cdot Naviguer \cdot auPort(\boldsymbol{port}: \boldsymbol{Port1}),$
$ResterAuPort(\boldsymbol{port}: \boldsymbol{Port1}) \rightarrow$
$\quad auPort(\boldsymbol{port}: \boldsymbol{Port1}) \cdot ResterAuPort(\boldsymbol{port}: \boldsymbol{Port1}),$
$ResterAuPort(\boldsymbol{port}: \boldsymbol{Port1}) \rightarrow auPort(\boldsymbol{port}: \boldsymbol{Port1}),$
$Naviguer \rightarrow enMer \cdot Naviguer,$
$Naviguer \rightarrow enMer\}$

S : $\{Comportement\}$

La modification effectuée (en gras ci-dessus) nous permet de bien spécifier que les ports de départ et d'arrivée doivent être les mêmes. Ici *port* représente le nom de la propriété et *Port1* la variable contenant l'identifiant du port de départ/destination.

A partir de là, nous sommes tentés de lier ce principe avec notre notion de *mesure* préalablement énoncée. Ainsi, une mesure ne pourrait-elle pas être représentée par un symbole terminal auquel on associe des attributs comme :

$$mesure(capteur : Capteur1, valeur : Valeur1, entité : Entité1,$$
$$propriété: Propriété1)$$

Pourtant, comme précisé en 2.1.2, un symbole terminal est supposé être une observation de l'entité observée et comme stipulé précédemment, une observation est un ensemble de mesures et non une seule et simple mesure. Ainsi le symbole terminal de notre grammaire serait un ensemble de mesures :

$$observation($$
$$mesure(capteur : Capteur1, valeur : Valeur1,$$
$$entité : Entité1, propriété: Propriété1),$$
$$mesure(capteur: Capteur2, valeur: Valeur2,$$
$$entité: Entité2, propriété: Propriété2),$$
$$\dots,$$
$$mesure(capteur: CapteurN, valeur: ValeurN,$$
$$entité: EntitéN, propriété: PropriétéN))$$

Ne pouvant pas prétendre à spécifier une grammaire en utilisant de tels symboles terminaux, nous proposons la notation suivante :

$$[capteur: C, valeur: V, entite: E, propriete: P]$$

Ceci désigne une ***observation*** comprenant une mesure dotée de ses quatre paramètres.

La décomposition ci-dessous désigne donc deux observations successives qui contiennent deux mesures aux caractéristiques identiques.

$$[capteur: C, valeur: V, entite: E, propriete: P]$$
$$[capteur: C, valeur: V, entite: E, propriete: P]$$

Nous souhaitons préciser également qu'ici, nous avons privilégié l'utilisation d'attributs nommés à celle d'attributs positionnels pour une raison simple, parfois certains symboles non terminaux sont dotés de tellement d'attributs qu'il est fastidieux de devoir tous les spécifier. Imposer l'utilisation d'attributs nommés peut amener à négliger les attributs inutilisés dans la règle de production en cours d'écriture.

## 3.4.2 **Relations entre attributs**

Nous avons vu précédemment une première manière de tisser des relations entre les attributs des mesures des différentes observations. En effet, réutiliser le même nom de variable tisse une relation d'égalité entre les membres des différentes mesures.

Sachant que nous souhaitons représenter des comportements spatio-temporels. Nous allons être amenés à représenter des relations géométriques et ainsi à devoir exprimer des relations arithmétiques. Pour ce faire nous nous proposons, comme usuellement présenté dans les grammaires attribuées, d'ajouter aux règles de production une partie supplémentaire permettant de définir des relations sur les variables représentant les valeurs des attributs.

Ainsi, nous pourrons écrire des règles de cette forme :

$$Changement\ de\ couleur \rightarrow$$
$$[capteur: C1, valeur: V1, entite: E, propriete: couleur] \blacktriangleleft$$
$$[capteur: C2, valeur: V2, entite: E, propriete: couleur]$$
$$| V1 \neq V2.$$

Cette règle ne pourra donc être validée que si dans deux clichés successifs de notre scène se trouve au moins une mesure concernant la propriété couleur d'une même entité, et que la valeur de cette propriété varie de l'un à l'autre.

Ou encore :

$$Acceleration \rightarrow$$
$$[capteur: C1, valeur: V1, entite: E, propriete: vitesse] \blacktriangleleft$$
$$[capteur: C2, valeur: V2, entite: E, propriete: vitesse]$$
$$| V1 < V2.$$

Cette règle ne pourra être validée que si dans deux clichés successifs de notre scène se trouve au moins une mesure concernant la propriété vitesse d'une même entité, et que la valeur de cette propriété augmente strictement d'un cliché à l'autre.

Cependant un problème évident se pose, comment peut-on poser des relations arithmétiques sur deux mesures appartenant au même cliché ? Nous allons nous attacher à répondre à cette question dans la section 3.4.3.

## 3.4.3 **Séquence et Simultanéité**

Nous avons vu précédemment que le caractère ◂ nous permet de séparer des symboles terminaux et non terminaux en indiquant leur succession temporelle. Dans les formalismes usuels des grammaires formelles, aucun symbole n'est nécessaire pour indiquer cette succession, et ce n'est pas sans raison que nous avons choisi d'en utiliser un.

Comme nous venons de le mettre en évidence, il est nécessaire de trouver un moyen de spécifier des relations entre les attributs des mesures d'une même observation. Nous avons donc choisi d'introduire un nouveau symbole indiquant la simultanéité entre des observations, à savoir &.

Grâce à ce dernier, nous pouvons ainsi écrire :

$$Distance(distance: Dist, entite1: E1, entite2: E2) \rightarrow$$
$$[capteur: C1, valeur: Lon1, entite: E1, propriete: longitude] \&$$
$$[capteur: C2, valeur: Lat1, entite: E1, propriete: lattitude] \&$$
$$[capteur: C3, valeur: Lon2, entite: E2, propriete: longitude] \&$$
$$[capteur: C4, valeur: Lat2, entite: E2, propriete: lattitude]$$
$$| Dist = sqrt((Lon1 - Lon2)^2 + (Lat1 - Lat2)^2),$$

$E1 \neq E2.$

Cette règle ne pourra être validée que s'il existe au sein d'un même cliché (simultanéité exprimée grâce au symbole &) au moins quatre mesures concernant les propriétés de longitude et latitude de deux entités différentes, et que la valeur *Dist* corresponde à la distance euclidienne entre ces dernières. Nous pourrions ici renommer cette règle *DistanceEuclidienne* et spécifier d'autres règles comme *DistanceDeManhattan*, etc. Ou encore décrire plusieurs règles *Distance* et utiliser un attribut pour spécifier le type de distance.

D'autre part, nous étendons ce symbole à la relation de simultanéité entre tous types de symboles, terminaux et non-terminaux. Par exemple nous pourrons écrire :

$$ApprocherDe(distancePrecedente: PDist, entite: E1, cible: C) \rightarrow$$
$$Distance(distance: CurrentDist, entite1: E1, entite2: C)\&$$
$$OrientéVers(entite: E1, cible: C)\triangleleft$$
$$ApprocherDe(distancePrecedente: CurrentDist, entite: E, cible: C)$$
$$| PDist < CurrentDist,$$
$$E1 \neq C$$

De la même manière que dans une expression arithmétique certains opérateurs ont des priorités plus importantes que d'autres, comme dans l'expression suivante par exemple où la multiplication a une priorité plus importante que l'addition :

$$5 + 3 \times 4 \rightarrow 5 + (3 \times 4)$$

Nous remarquerons dans notre cas que le symbole/opérateur & est prioritaire sur $\triangleleft$. L'exemple ci-dessus propose un comportement récursif décrivant le fait pour une entité de se rapprocher d'une cible. Ceci suppose donc que $Distance$ et $OrientéVers$ se déroulent en même temps, c.à.d. qu'ils concernent la même séquence d'observations.

Intéressons-nous plus en détail à la règle ci-dessus. Tout d'abord, remarquons qu'il s'agit d'une règle récursive, ainsi, potentiellement elle pourra concerner un nombre arbitraire de clichés de notre scène. Pour que cette dernière règle puisse être validée à chaque étape, il est nécessaire que pour chacun des clichés l'on puisse valider la règle $Distance$ entre deux entités $E1$ et $C$ ainsi que la règle $OrientéVers$, tout en vérifiant qu'il ne s'agit pas de la même entité et que la distance entre ces deux dernières ne cesse de diminuer d'itération en itération. Ici les règles $Distance$ et $OrientéVers$ piocherons dans les mesures d'un même cliché.

### 3.4.4 **Impact de l'ajout du symbole indiquant la simultanéité : &**

Comme précédemment énoncé nous traitons des modèles génératifs de comportements, c'est-à-dire qu'à partir d'une instance de notre modèle, nous pouvons générer/reconnaitre par le biais d'algorithmes tout un ensemble de comportements. Cet ensemble de comportement est communément appelé le *langage*.

DEFINITION : Soit G une grammaire, alors nous noterons $L_G$ l'ensemble de comportements pouvant être reconnus/générés par G.

Par extension, nous pouvons parler de langage associé à un symbole non-terminal au sein d'une grammaire.

DEFINITION : Soit G une grammaire et $N_G$ son ensemble des symboles non terminaux, alors nous noterons $l_A$ avec $a \in N_G$ l'ensemble de comportements pouvant être reconnus/générés en prenant A comme unique symbole de départ de G.

Rappelons de plus qu'au vu de nos précédentes définitions, le langage généré par la grammaire composée de l'unique règle

$$S \rightarrow [valeur: VAL, type: TYPE]$$
$$| VAL = \text{"chalutier"}, TYPE = \text{"}classification\text{"}$$

est composé de toutes les séquences d'une observation contenant au moins une mesure concernant un chalutier.

Le symbole & tel que nous le proposons permet ainsi de spécifier la grammaire composée de l'unique règle

$$S \rightarrow [valeur: VAL1, type: TYPE1] \& [valeur: VAL2, type: TYPE2]$$
$$| VAL1 = \text{"chalutier"}, TYPE1 = \text{"}classification\text{"}$$
$$| VAL2 = \text{"voilier"}, TYPE2 = \text{"}classification\text{"}$$

Et dont le langage associé est l'ensemble des comportements d'une observation contenant au moins deux mesures concernant respectivement un chalutier et un voilier.

Ce qui nous amène naturellement à la conclusion suivante : le symbole & utilisé pour décrire une relation de simultanéité peut être perçu comme l'intersection des langages pouvant être générés/reconnus par deux symboles au sein d'une grammaire.

NOTATION : Soit $G =< N, T, R, S >$ avec $A \in N$, nous notons $R_A \in R$ l'ensemble des règles de production de notre grammaire dont le symbole de départ est $A$.

Ainsi, nous caractérisons le symbole & de la manière suivante :

Soit une grammaire $G =< N, T, R, S >$ avec $A, B, C \in N$. Si $R_A = \{A \rightarrow B\&C\}$ alors $l_A = l_B \cap l_C$.

### Discussion sur l'ajout de l'intersection dans une grammaire hors contexte

Ajouter le symbole & a pour effet indirect de permettre de proposer une mécanique similaire à l'intersection de langages. Hors, il est prouvé que l'intersection des langages générés par deux grammaires hors contexte peut ne pas pouvoir être généré/reconnu par une grammaire hors contexte. Ainsi, sans même parler d'attributs et de leur traitement, nous pouvons déjà affirmer que déterminer si une séquence d'observation appartient à notre modèle est à minima un problème de décision PSPACE complet.

Nous pouvons ainsi nous demander si cet ajout est réellement justifié tant son impact sur le système futur de reconnaissance est important. En effet, nous aurions peut-être pu nous contenter de limiter l'utilisation du symbole & entre deux symboles terminaux et ainsi résoudre notre problématique de pouvoir poser des relations sur les valeurs d'attributs de deux mesures au sein d'une même règle de production. Cependant, une de nos priorités était la possibilité de pouvoir réutiliser des règles en les combinant avec d'autres sans pour autant devoir écrire manuellement une nouvelle règle combinant les caractéristiques des deux autres.

## 3.4.5 **Méta-Symboles**

Nous avons également introduit plusieurs symboles inspirés des expressions rationnelles (Thompson, 1968) et nous permettant de factoriser considérablement l'écriture des règles de production de notre grammaire.

### La répétition

Bien souvent nous sommes tentés de spécifier qu'une partie d'un comportement peut se répéter au moins une fois. Ainsi plutôt que d'écrire deux règles telles que les suivantes :

$$ResterSurPlace(posX{:}X, posY{:}Y, entite{:}E) \rightarrow$$
$$Position(X{:}X, Y{:}Y, entite{:}E)\triangleleft$$
$$ResterSurPlace(posX{:}X, posY{:}Y, entite{:}E).$$

$$ResterSurPlace(posX{:}X, posY{:}Y, entite{:}E) \rightarrow$$
$$Position(X{:}X, Y{:}Y, entite{:}E).$$

Nous pouvons nous contenter d'utiliser le symbole $+$ :

$$ResterSurPlace(posX{:}X, posY{:}Y, entite{:}E) \rightarrow$$
$$Position(X{:}X, Y{:}Y, entite{:}E) +.$$

## L'observation arbitraire

Nous sommes également amenés pour des besoins de 'recalage' à devoir consommer des observations sans pour autant utiliser les informations des mesures qu'elles contiennent, pour ce faire, nous utilisons le symbole $*$ pouvant prendre la place de n'importe quelle mesure. Ainsi, nous sommes capables d'écrire :

$$ConsommeTroisObservations \rightarrow$$
$$[*]\triangleleft[*]\triangleleft[*].$$

Ceci nous permet ainsi de prédéfinir le nombre d'observations devant être consommées par une règle déjà écrite :

$$ResterSurPlacePendantTroisObservations \rightarrow$$
$$ResterSurPlace\ \&$$
$$ConsommeTroisObservations.$$

Nous pouvons également nous en servir pour pouvoir compter le nombre d'observations consommées par un autre symbole :

$$NombreObservationsConsommees(Compteur{:}Cnt, Nombre{:}Nb) \rightarrow$$
$$[*]$$
$$\mid Nb = Cnt + 1$$
$$NombreObservationsConsommees(Compteur{:}Cnt, Nombre{:}Nb) \rightarrow$$
$$[*]\triangleleft$$
$$NombreObservationsConsommees(Compteur{:}CntPlusOne,$$
$$\qquad Nombre{:}Nb)$$
$$\mid CntPlusOne = Cnt + 1$$

Nous avons donc ici deux règles (la condition de terminaison et la règle récursive) nous permettant de compter un nombre arbitraire d'observations. Ainsi à chaque observation deux cas se produisent : la règle est validée si la valeur du compteur incrémentée de un peut être associée à la valeur $Nb$, d'autre part, la règle récursive produit une attente de validation remise à la prochaine observation. Voici ainsi un usage que l'on pourrait en faire pour limiter le nombre d'observations que peut consommer un comportement :

$$ResterSurPlacePendantTroisObservations \rightarrow$$

$$ResterSurPlace\ \&\ NombreObservationsConsommees(Nombre: Nb) \mid Nb = 3.$$

Ainsi le seul moyen pour que cette règle soit validée est que la séquence d'observation comporte une entité restant sur plance pendant les trois prochaines observations.

Le symbole & est d'autant plus important qu'il est nécessaire pour pouvoir créer des règles de production dites *de calcul*. Comme par exemple :

$$ChangementDeReferentiel(OriginX: Ox, OriginY: Oy, OriginAngleX: OAx, OriginAngleY: OAy, X: X, Y: Y, NewX: NX, NewY: NY) \rightarrow [*] \mid NX = (X - Ox) \times OAx + (Y - Oy) \times OAy, NY = (Y - Oy) \times OAx - (X - Ox) \times OAy, X = NX \times OAx - NY \times OAy + Ox, Y = NX \times OAy + NY \times OAx + Oy.$$

La règle suivante est validée lorsque les coordonnées $X, Y$ d'un point correspondent aux coordonnées $NX, NY$ après que l'on ait fait effectuer un changement de repère vers un nouveau d'origine $Ox, Oy$ et dont le vecteur indiquant l'axe des abscisses soit de coordonnées $OAx, OAy$ dans l'ancien repère.

Par exemple, Le point de coordonnées (3,4) a pour coordonnées (4,0) dans nouveau repère dont l'origine serait située en (3,0) et ayant subi une rotation de 90° dans le sens trigonométrique. La règle est ainsi validée car :

$$4 = (3 - 3) \times 0 + (4 - 0) \times 1$$
$$0 = (4 - 0) \times 0 - (3 - 3) \times 1$$
$$3 = 4 \times 0 - 0 \times 1 + 3$$
$$4 = 4 \times 1 + 0 \times 0 + 0$$

Cette règle peut être utilisée dans le but de calculer les nouvelles coordonnées d'un point dans un nouveau repère, à l'inverse de calculer les coordonnées d'origine d'un point dont on connait les coordonnées dans un ancien repère, ou encore de calculer les paramètres d'un repère en fonction des coordonnées d'un point dans l'ancien et le nouveau repère. La sémantique est donc découplée de l'usage, ce que nous recherchons en priorité.

Ainsi, nous pouvons abstraire des relations complexes entre attributs sous la forme de règles de production et les combiner avec d'autres symboles.

## La négation

Enfin, nous proposons également le symbole \ pour indiquer la négation d'un symbole terminal ou non terminal. Nous pouvons donc écrire :

$$NotAtPosition(posX: X, posY: Y, entite: E) \rightarrow \backslash AtPosition(poxX: X, posY: Y, entite: E)$$

Cependant, la négation est à manipuler avec précaution, surtout lorsqu'on l'utilise avec des symboles non terminaux.

$$NePasResterSurPlace(entite: E) \rightarrow \backslash ResterSurPlace(posX: X, posY: Y, entite: E).$$

Il est nécessaire de s'interroger sur la signification de la négation d'un symbole non terminal, et les conditions de validation de cette négation.

En effet, dans notre premier exemple, la négation du non-terminal $Position$ est immédiate et facilement testable car cette relation ne concerne qu'un seul cliché de notre scène. Pourtant, il serait naïf de croire que seul $AtPosition(poxX: X, posY: Y, entite: E)$ ou $\backslash AtPosition(poxX: X, posY: Y, entite: E)$ peuvent être validés. En effet, il suffit de supposer la présence de deux capteurs sur la scène pour que les deux règles puissent être validées simultanément. De plus, cela peut également être le cas si les valeurs mesurées sont suffisamment imprécises comme le montre la figure ci-dessous.

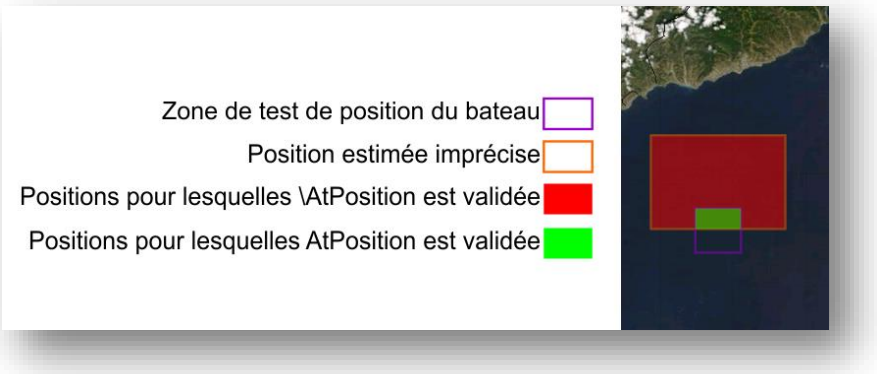


**Figure 3-4 : Selon les hypothèses faites sur l'imprécision de la mesure, le voilier peut être considéré comme étant ou non dans la zone testée.**

Ainsi, ici, nous pouvons avoir une hypothèse supposant que l'entité se trouve dans la zone verte et une seconde hypothèse tirant parti du fait que l'entité se trouve dans la zone rouge. L'hypothèse du monde clos n'est pas tenable du fait de l'imprécision des mesures, et de leur représentation par des intervalles de valeurs flottantes.

Enfin, quelle est la signification de la négation d'une étape de plan traitant de plusieurs clichés ? Formellement il devrait pouvoir s'agir de l'existence d'une succession de clichés tels qu'ils ne valident pas l'étape de plan niée.

Que se passe-t-il maintenant dans le cas de négation de règles récursives ? Ici se pose un problème évident, si le nombre de répétition n'est pas borné, alors il ne sera jamais possible de prouver que la négation du symbole est impossible, ceci présente un réel problème pratique et conduit à une explosion du nombre d'hypothèses qui ne sera jamais réduit.

Prenons par exemple notre précédente règle nous permettant de compter le nombre d'observations consommées, y associer le symbole \ provoquera une validation subtile. Prenons le cas suivant :

$$NeConsommePasTroisObservations \rightarrow$$
$$\backslash NombreObservationsConsommees(Nombre: Nb)$$
$$|Nb = 3.$$

Cette règle se verra validée pour une succession de une, deux, quatre, cinq, …, une infinité d'observations (sauf trois). Ce qui peut être acceptable d'un point de vue sémantique. Cependant, intéressons-nous à la règle :

$$ConsommeMoinsDeTroisObservations \rightarrow$$
$$\backslash NombreObservationsConsommees(Nombre: Nb)$$
$$|Nb \leq 3.$$

Certes cette règle ne pourra être validée que sur des séquences de une, deux ou trois observations, pourtant si la séquence contient deux mille observations, cette règle restera active jusqu'à tenter de consommer la dernière observation. En effet, le test sur la valeur de Nb étant effectué dans la règle de terminaison, la règle récursive est toujours temporairement valide. Pour pouvoir élaguer plus vite ce test, il aurait fallu rajouter dans les relations entre attribut dans la règle récursive :

$$NombreObservationsConsommees(Compteur: Cnt, Nombre: Nb) \rightarrow$$
$$[*]$$
$$| Nb = Cnt + 1$$
$$NombreObservationsConsommees(Compteur: Cnt, Nombre: Nb) \rightarrow$$
$$[*]\blacktriangleleft$$
$$NombreObservationsConsommees(Compteur: CntPlusOne,$$
$$Nombre: Nb)$$
$$| CntPlusOne = Cnt + 1,$$
$$\boldsymbol{Nb > CntPlusOne}$$

Pour toutes ces raisons et ambiguïtés, la négation n'a été implémentée dans notre système que pour des cas très particuliers et nous conseillons généralement la réécriture d'une nouvelle règle plutôt que l'utilisation de notre symbole car souvent, son usage n'est pas approprié.

### 3.4.6 **Evolution successives à partir des modèles classiques**

Les travaux présentés dans ce manuscrit ont été principalement dirigés par un ensemble d'exemples inspirés de cas pratiques et réels que ne nous nous sommes efforcés de pouvoir traiter. Parmi les exemples que nous avons jugés pertinents figurent l'anticipation du comportement d'un voilier, ou encore le suivi du trafic maritime de navire de transports de marchandises. Nous nous

sommes rapidement rendu compte que chercher à expliciter ce genre de comportements (sachant que les données d'entrée sont des mesures issues de capteurs hétérogènes et imprécis) à l'aide de modèles classiques dit efficaces n'était pas pertinent comme nous avons pu l'observer dans le Chapitre 2.

Les extensions apportées à notre modèle ont été la conséquence de deux phénomènes.

## Expressivité

Certains comportements ne pouvaient tout simplement pas être retranscrits par des modèles trop pauvres. Le choix d'une grammaire hors contexte par rapport à une grammaire rationnelle découle de cette problématique.

## Déclarativité

Certaines décisions ont été dictées par l'aisance de représentation des règles, facilitant la lecture et leur modification, ou encore leur proximité avec leur description humaine naturelle. Le choix d'attributs plutôt que l'augmentation du nombre de symboles terminaux a en partie découlé de cette problématique.

## Efficacité de la reconnaissance mise de côté

Les contraintes liées à l'efficacité de la reconnaissance n'étaient que secondaires durant ce travail de recherche du bon modèle, représenter explicitement et correctement les comportements exemples était déjà un problème en soi.

# Chapitre 4
# Utilisation du modèle

# 4.1 Production d'arbres de reconnaissance

Nous nous sommes interrogés sur la nature et la manière de présenter des résultats d'un système de reconnaissance à un opérateur humain. En effet, que peut-on attendre comme résultats fournis par un système de reconnaissance de comportements.

Nous aurions pu envisager comme beaucoup de ne fournir que les étiquettes des comportements reconnus comme par exemple : $sortir\_en\_mer$. De plus pour plus de précisions, nous aurions pu proposer les valeurs des attributs associés à cette étiquette, comme par exemple : $sortir\_en\_mer(Bateau{:}'kalee', Port{:}'Fréjus')$. Cependant, nous avons été confrontés à un double problème.

En effet, si nous reprenons notre modèle de comportement, l'étiquette 'racine' est de manière constante $Behavior$, ce qui est relativement pauvre en information, de plus, comme nous l'avons cité en section 3.1, l'intention ou le but poursuivi par l'agent sous surveillance peut très bien n'être qu'une étape du comportement qu'il exhibe. Ainsi, la pertinence de la remontée d'information ne se situe pas dans la racine ou les fils directs de la racine de notre bibliothèque de comportements.

D'autre part, remonter un résultat sans justification est inacceptable dans notre cadre applicatif. Il est en effet nécessaire de pouvoir exhiber des preuves ou faisceaux d'évidence permettant rapidement à un opérateur humain de reproduire brièvement ou du moins de suivre les étapes de raisonnements appliquées, lui permettant de valider une hypothèse remontée par le système. Choisir de ne remonter qu'un résultat pauvre sans le faisceau d'évidence ayant permis de l'obtenir n'est pas envisageable.

Bien entendu, nous ne souhaitons pas nous borner à notre domaine applicatif, ainsi, la présentation et la nature de ces résultats doit être indépendante de l'application ciblée, quitte à ensuite proposer une manière spécifique de les présenter sous forme de surcouche. Nous ne nous attarderons pas dans ce manuscrit à cela.

### 4.1.1 L'arbre comme hypothèse de reconnaissance

Nous pouvons remarquer que notre modèle est issu de la théorie de la compilation et des langages formels. Hors, dans ces domaines, lors de l'étape de reconnaissance (Parsing), la grande majorité des algorithmes construisent un arbre de reconnaissance, parfois appelé arbre de syntaxe abstraite comme le montre la figure ci-dessous, illustrant la décomposition en unité syntaxique d'un programme écrit dans un langage de programmation simpliste.

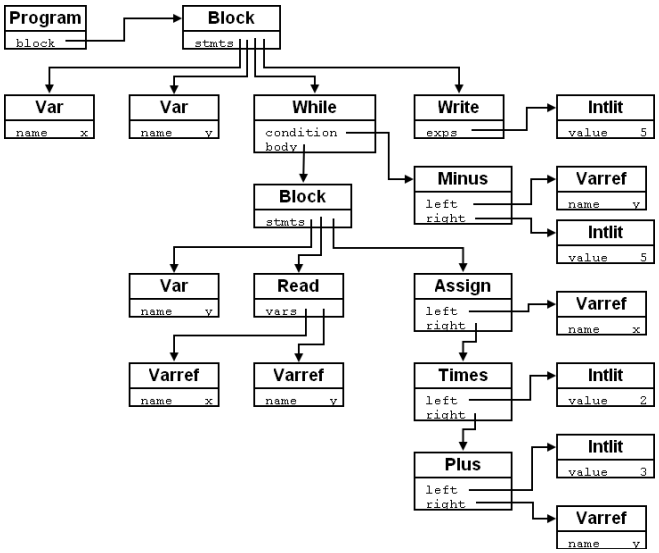


**Figure 4-1: Exemple d'arbre de syntaxe abstraite obtenu sur l'analyse du code d'un programme.**

Ainsi, dans le cas présent, les nœuds de cet arbre seraient les symboles non terminaux de nos règles de production, et un seul type de feuille serait possible : le symbole *measurement,* tel que le montre l'arbre ci-dessous.

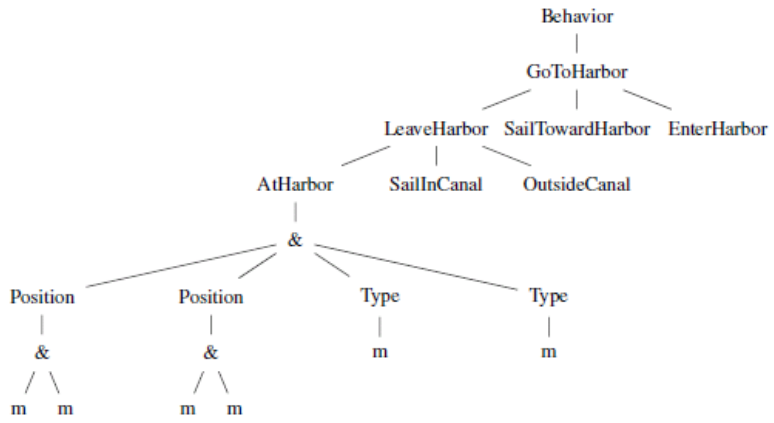


**Figure 4-2: Exemple d'arbre de reconnaissance pouvant être produit par notre système.**

Bien évidemment et comme montré dans la figure ci-dessus, à chaque nœud de l'arbre est associé son ensemble d'attributs instanciés ou non encore instanciés.

L'intérêt de présenter un tel arbre est multiple. Tout d'abord, il permet de représenter naturellement la granularité de l'information de par sa structure hiérarchique. Ainsi, si l'interface de visualisation proposée le permet, on peut choisir de n'afficher l'arbre que pour les $n$ premiers niveaux de détails, ce qui permet d'obtenir une vue condensée et de ne la détailler que si nécessaire.

D'autre part, en affichant toutes les étapes et sous étapes reconnues, il permet de mettre en évidence d'éventuels buts ponctuels poursuivis par l'agent qui impliqueraient la mise en œuvre d'un comportement plus général.

De plus, n'oublions pas qu'un opérateur humain désire avoir à sa disposition le faisceau d'évidence ayant permis au système d'inférer son résultat. Présenter l'arbre de reconnaissance permet à l'humain de pouvoir vérifier et comprendre rapidement les étapes critiques ayant été effectuées par le système pour en arriver au résultat proposé.

Enfin, n'oublions pas que nous sommes dans un contexte de reconnaissance de comportements $en\ ligne$, ainsi, rien ne nous empêche de proposer entre chaque arrivée d'ensemble de mesures le résultat intermédiaire de la reconnaissance, à savoir un arbre de reconnaissance partiel et ainsi proposer un certain degré de prédiction quant aux possibles évolutions de la scène sous surveillance.

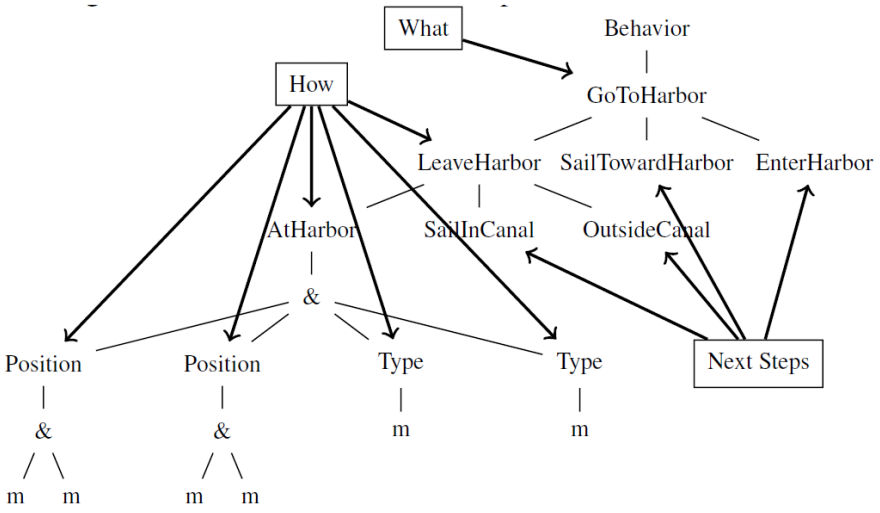


**Figure 4-3: Le triple résultat proposé par l'arbre de reconnaissance de comportements.**

Ainsi, nous pouvons observer sur l'exemple ci-dessus le triple résultat proposé par un arbre de reconnaissance de comportement. Le *quoi ?* étant le fils de la racine en cours de reconnaissance, le *comment ?* représentant la justification attendue par l'opérateur humain, c'est-à-dire les étapes validées jusqu'à présent et le *vers où ?* indiquant les étapes suivantes que l'on s'attend à observer pour être en cohérence avec l'hypothèse émise.

### 4.1.2 **Algorithmes**

Nous venons, dans la section précédente, de définir ce que nous souhaitions obtenir en résultat de nos algorithmes. La figure suivante récapitule brièvement les entrées/sorties du système qui utilisera les algorithmes que nous allons présenter.

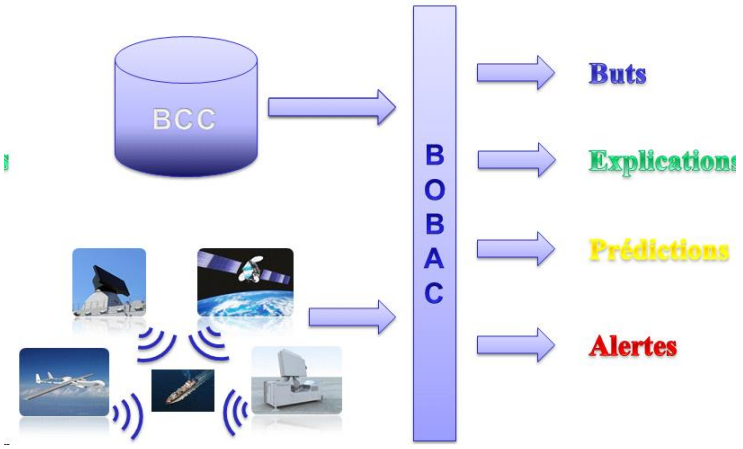


**Figure 4-4: Schéma représentant les entrées/sorties de notre système de reconnaissance de comportements.**

Nous supposons donc maintenant avoir à disposition une grammaire écrite selon le modèle proposé. Nous souhaitons savoir si le comportement de la scène sous surveillance (l'ensemble des clichés disponibles depuis une date t jusqu'à la date courante) correspond à un comportement ayant pu être généré par la grammaire. Une méthode naïve consisterait à générer l'ensemble des comportements possibles et à vérifier que le comportement témoin appartient bien à cet ensemble. Ceci est bien évidemment impossible dans notre cas au vu du grand nombre (voire de l'infinité dans le cas de certaines règles de production récursives) de comportements pouvant être produits à partir d'une même grammaire.

D'autre part, souvent nous ne possédons pas le comportement complet (on peut observer indéfiniment une scène) et devons cependant fournir des résultats à un opérateur humain. Nous souhaitons présenter les interprétations possibles des mesures récupérées en fonction de la grammaire à disposition.

Nous fournissons l'état des arbres de syntaxe abstraite en cours de construction comme résultat. Chaque interprétation est présentée sous la forme d'un arbre exhibant une dérivation possible de notre grammaire compatible avec les mesures observées. Les nœuds de ces arbres, recensent plusieurs champs d'informations (nom, variables indiquant son état, etc.) et ne possèdent chacun qu'un seul parent et deux listes de nœuds fils (les nœuds fils validés et les nœuds fils restants à valider). Ils sont construits à partir des règles de production de notre grammaire.

Le principe général de notre système est le suivant. La liste d'arbres de syntaxe abstraite L est initialisée au début de la reconnaissance à l'aide des règles concernant le symbole de départ de la grammaire (Behavior). Imaginons ainsi la règle de grammaire suivante :

$Behavior \rightarrow MissionIdleState \, Behavior \, Behavior$

Cette dernière sera transformée en l'arbre suivant :

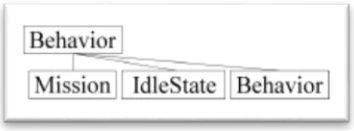


**Figure 4-5 : Exemple d'initialisation d'un arbre partiel de reconnaissance.**

Puis, à chaque arrivée de cliché, nous effectuons l'itération suivante, mettant à jour la liste des arbres de syntaxe abstraite.

- Développer chaque arbre à gauche en profondeur d'abord pour un nœud classique, ou en largeur d'abord pour chaque nœud & (simultané) :

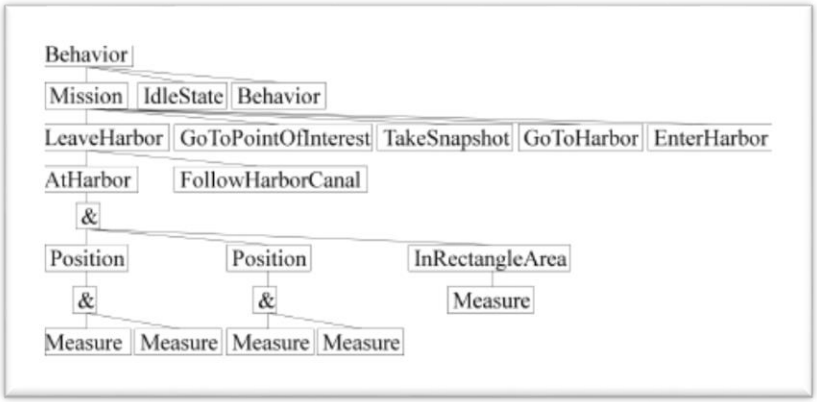


**Figure 4-6 : Etat de l'arbre de reconnaissance juste avant de le confronter aux mesures issues des capteurs.**

- Associer à chaque feuille/measurement une mesure du cliché courant, si plusieurs combinaisons sont possibles, dupliquer les arbres.
- Valider ou invalider les nœuds développés précédemment et repositionner le nœud en cours pour chaque arbre sur le prochain nœud à développer.

Ainsi à chaque fin d'itération, il suffit d'afficher une représentation graphique des arbres pour avoir immédiatement pour chaque interprétation/arbre :

- Quoi
    - Les fils directs du nœud racine Behavior.
- Comment
    - L'ensemble des nœuds validés de l'arbre.
- Vers Où
    - L'ensemble des nœuds n'étant pas encore validés.

La gestion de la cohérence des relations entre attributs est réalisée à partir de techniques de programmation par contraintes. Nous associons à chacun de nos arbres un store de contraintes qui lui est propre.

À chaque développement de nœud à partir d'une règle, les relations entre attributs ainsi que leurs liens sont transmis dans le store correspondant. Ainsi, dès lors que l'ensemble de contraintes contenu dans le store n'est pas satisfiable, l'arbre est retiré de la liste des interprétations L.

## 4.2 Traitement de l'imprécision et de l'incertitude

Jusqu'ici passé sous silence, nous allons traiter de la gestion de l'imprécision et de l'incertitude dans notre système de reconnaissance. Nos données sont issues de capteurs réels. Ces derniers sont ainsi nécessairement imparfaits. De même, ils ne peuvent nous fournir des informations qu'à intervalles plus ou moins réguliers.

Nous sommes donc ainsi en présence d'observations manquantes et imprécises. Nous présentons dans les sections suivantes les différentes pistes écartées et celle retenue.

### 4.2.1 Usage des probabilités

Il est fréquent de voir la théorie des probabilités utilisée pour représenter des phénomènes incertains. Ainsi, nous pourrions choisir de représenter la valeur numérique renvoyée par un capteur, par une loi normale ou autre densité de probabilité.

Cependant, les probabilités, dont le but initial est de pouvoir modéliser des phénomènes naturels issus de plusieurs évènements aléatoires ne nous semblent pas adaptées dans notre problématique.

En effet, l'usage des probabilités implique l'utilisation de ces probabilités par la suite, que ce soit pour classer les hypothèses ou pour en éliminer certaines.

Or, dans un cadre militaire, il n'est pas réaliste de laisser un système prendre la décision d'écarter une situation possible avec une probabilité faible sans intervention humaine. En effet, une situation dangereuse devra toujours être remontée, et ce quelle que soit sa probabilité d'apparition.

D'autre part, l'usage de probabilité ne rend pas possible la distinction entre incertitude et imprécision.

Enfin, présupposer d'un modèle de densité de probabilité particulier pour nos capteurs numériques nous apparait réducteur. En effet, nos sources de données étant multiples et hétérogènes, il est fort probable que le modèle choisi ne puisse convenir à l'ensemble de ces dernières.

Ainsi nous avons pour l'instant choisi d'écarter l'usage de la théorie des probabilités dans notre système.

### 4.2.2 **Arithmétique des Intervalles**

Pour l'ensemble des capteurs avec lesquels nous sommes amenés à travailler peut être définie une marge d'erreur. Cette dernière peut être écrite dans les spécifications d'un de ces derniers ou être inférée grâce aux caractéristiques proposées telle que sa précision ou l'étendue de sa mesure.

Cette caractéristique commune à chacun de nos capteurs nous incite à représenter toutes nos mesures numériques internes non pas par un nombre, mais par un encadrement de la valeur de ce nombre.

Ainsi, à toutes nos variables numériques seront associées un intervalle de nombre flottants, bornant de manière certaine la valeur représentée.

### 4.2.3 **Utilisation d'un solveur de contraintes sur les intervalles**

Ce choix de travailler avec des intervalles de nombre flottants n'est pas sans conséquence. Comme nous avons pu le voir précédemment dans les spécifications de notre modèle génératif de comportement, nous sommes amenés à spécifier des relations sur nos variables représentant des valeurs réelles. Ainsi, comme nous avons choisi de représenter ces valeurs par des intervalles de nombre flottants les encadrant, il est nécessaire d'avoir à disposition une arithmétique nous permettant de garantir le respect de ces relations.

Notre système repose ainsi en partie sur l'arithmétique des intervalles, et notamment sur la propagation de contraintes sur intervalles (Older & Vellino, 1993).

Ainsi notre système repose sur l'utilisation d'un solveur de contraintes sur intervalles développé et utilisé en interne chez Thales Systèmes Aéroportés, à savoir Interlog (Botella & Taillibert, 1993). Ce dernier présente de multiples avantages.

Nous rappelons que les valeurs de nos mesures physiques imprécises sont justement représentées soit par des symboles, soit par des intervalles de réels. Interlog (Botella & Taillibert, 1993), nous permet de traiter les contraintes sur les intervalles de réels. Une extension de cet outil multi contexte aux contraintes sur les symboles est envisagée.

Ce dernier est en effet capable de gérer de multiples stores de contraintes en même temps, ce qui correspond à notre choix d'avoir un store de contraintes par arbre de syntaxe abstraite.

Du surcroît et par l'utilisation combinée des contraintes sur les intervalles et de notre représentation, non seulement nous pouvons réduire au plus tôt la

liste des interprétations (dès que les contraintes ne sont plus satisfiables), mais un attribut représentant un réel et dont la valeur est un intervalle peut voir sa valeur précisée bien après sa mesure effective. La reconnaissance présente permet ainsi d'affiner les observations passées. Ainsi, une hypothèse validée préalablement à tort en raison de l'imprécision d'une mesure (intervalle trop large) pourra être remise en cause dans le futur, si de nouvelles contraintes sur cette même mesure surviennent, ce qui pourra ainsi invalider l'hypothèse courante.

D'autre part, une problématique vis-à-vis du fait de conserver ces variables dans le but de les réviser par la suite est une potentielle explosion mémoire.

Ainsi, comment savoir quels ensembles de variables et contraintes l'on doit garder et de quels ensembles peut-on se passer pour le reste de la reconnaissance ?

Comme nous avons pu le voir précédemment, nous ne pouvons utiliser de variables globales ou externes à nos règles lors de l'écriture de l'une d'entre elles.

En effet, la transmission de données entre nos règles et dans l'arbre de reconnaissance se réalise par unification de variables déclarées dans la tête de la règle et de variables utilisées dans les symboles de la partie droite de la règle.

Il est donc ainsi possible de construire un graphe de dépendance entre les variables. Ces dépendances sont construites au fur et à mesure que des relations font intervenir des variables les unes avec les autres.

Prenons la règle suivante :

$$
\begin{aligned}
&Acceleration \rightarrow \\
&[capteur: C1, valeur: V1, entite: E, propriete: vitesse] \\
&[capteur: C2, valeur: V2, entite: E, propriete: vitesse] \\
&\mid V1 < V2.
\end{aligned}
$$

Ici, les variables $V1$ et $V2$ sont liées par le biais de la relation $V1 < V2$. Dans ce cas précis, lorsque le nœud $Acceleration$ aura été reconnu et où qu'il se trouve dans l'arbre de reconnaissance, nous pourrons supprimer du store de contraintes les variables $V1$ et $V2$ ainsi que la contrainte $V1 < V2$.

Prenons une règle un peu plus étoffée :

$$
\begin{aligned}
&ChangementDeReferentiel(OriginX: Ox, OriginY: Oy, \\
&\qquad\qquad OriginAngleX: OAx, OriginAngleY: OAy, \\
&\qquad\qquad X: X, Y: Y, NewX: NX, NewY: NY) \rightarrow \\
&[*] \\
&\mid NX = (X - Ox) \times OAx + (Y - Oy) \times OAy, \\
&\quad NY = (Y - Oy) * OAx - (X - Ox) * OAy, \\
&\quad X = NX * OAx - NY * OAy + Ox, \\
&\quad Y = NX * OAy + NY * OAx + Oy.
\end{aligned}
$$

Cette dernière nous permet de tisser les dépendances suivantes entre variables déduites des différentes relations dans la règle :

$$(NX, X, Ox, OAx, Y, Oy, OAy)$$

Si jamais une règle de niveau supérieur lie l'une de ces variables à une autre variable et que cette règle reste *en cours de validation* sur plusieurs observations, aucune des variables ni contraintes précédentes ne seraient retirées du solveur.

Conserver certaines contraintes au sein de notre solveur est certes coûteux, mais nous pouvons ainsi élaguer des hypothèses qui ne l'auraient pas été si jamais nous avions choisi de considérer nos relations entre variables comme des tests plutôt que des contraintes perdurant au cours du temps.

Prenons l'exemple de règle récursive suivant :

$$Immobile(Entite{:}E, X{:}X, Y{:}Y) \rightarrow$$
$$Position(Entité{:}E, X{:}X, Y{:}Y)$$
$$Immobile(Entite{:}E, X{:}X, Y{:}Y).$$

Ainsi, au bout de trois observations, la position estimée pourrait être celle présentée dans la figure ci-dessous :

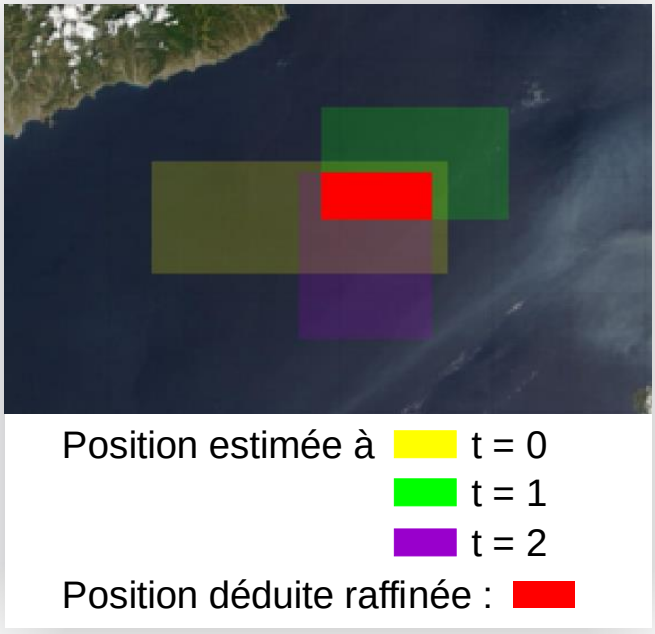


**Figure 4-7 : Raffinement de l'estimation de la position d'une entité au cours du temps.**

De même, prenons la règle ci-dessous :

$$ImmobileDansLaZone(Entite{:}E, Zone{:}Z, X{:}X, Y{:}Y) \rightarrow$$
$$Position(Entité{:}E, X{:}X, Y{:}Y)\&$$

$PointALinterieurDeLaZoneZone(X:X,Y:Y,Zone:Z)$ ◂
$ImmobileDansLaZone(Entite:E,Zone:Z,X:X,Y:Y).$

Dans la situation suivante, cette dernière sera invalidée lors du traitement de la deuxième observation :

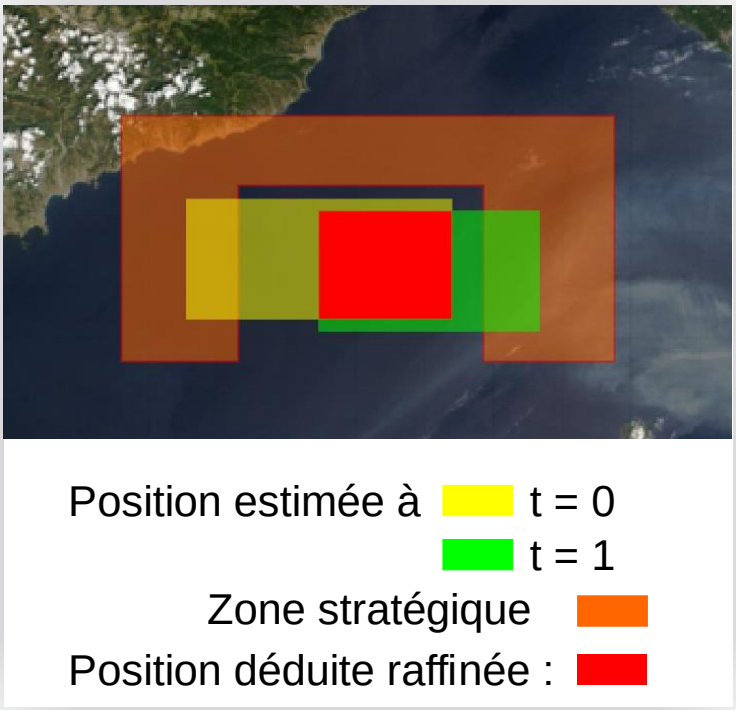


**Figure 4-8 : La prise en compte des observations précédente seule permet de déduire que l'entité n'est pas dans la zone stratégique.**

Ainsi, dans la situation ci-dessus, se contenter de réaliser un test d'appartenance à la zone à chaque arrivée d'observation n'est pas suffisant pour savoir si l'entité est immobile à l'intérieur ou pas. Notre solveur de contrainte, détectant qu'il n'est pas possible pour l'entité d'appartenir aux trois zones élimine l'hypothèse.

Au fur et à mesure où nous déplions notre arbre de reconnaissance, nous ajoutons les contraintes présentes dans les règles utilisées au sein de l'hypothèse courante. Si jamais deux règles existent pour un même symbole, nous dupliquons le store de contraintes et l'associons au nouvel arbre de reconnaissance généré.

Déterminer à quels moment l'on peut retirer une variable ainsi que ses contraintes associées du store de contraintes est un mécanisme très proche d'un garbage collector, à cela près que les dépendances entre les variables

forment un arbre et non un graphe quelconque, ainsi lors de la validation d'une règle il est très aisé de savoir si l'on peut retirer certaines contraintes du store de contraintes.

Le test de satisfiabilité de l'ensemble de contraintes peut être effectué à chaque dépliement (pour un élagage au plus tôt) ou bien à chaque confrontation du symbole terminal à l'observation courante (auquel cas certains cas auront été dépliés à tort).

Souvent un bon compromis est d'effectuer un test de satisfiabilité simple comme l'arc-consistance durant le dépliage, et un test de 2b consistance ou 3b consistance au moment de la confrontation avec l'observation courante.

Un grand avantage lié à l'utilisation d'Interlog est le fait de pouvoir spécifier des contraintes non linéaires et de conserver des temps de calculs raisonnables par rapport à des solveurs non linéaires génériques.

L'inconvénient majeur est que la réussite d'un test de satisfiabilité ne garantit pas l'existence d'une solution au système de contraintes, mais seulement le fait que si une solution existe, alors les valeurs des variables la constituant seront comprises dans les intervalles qui leur sont associés.

Ainsi, il est probable que de temps à autre, une hypothèse soit conservée alors qu'elle aurait pu être éliminée en utilisant un autre type de solveur (au prix d'un temps de calcul bien supérieur bien entendu).

Actuellement l'utilisation d'Interlog nous offre un grand panel d'opérateurs arithmétiques utilisables au sein de nos contraintes (équations, inéquations, multiplication, additions, puissance,…).

Nous avons également pu interfacer notre système avec d'autres solveurs généralistes comme ceux implémentés à l'intérieur de la suite de développement Mathematica[1]. Dans les résultats obtenus, les hypothèses étaient certes élaguées au plus tôt, cependant le coût engendré par le test de consistance était prohibitif pour un usage online tel que voulu dans notre cadre applicatif.

## Prise de recul par rapport à notre système

Dans l'ensemble, nous pourrions tourner un autre regard vers notre système. En effet, ce dernier ne se contente au final que de créer de multiples stores de contraintes et de les renseigner au fur et à mesure de la reconnaissance. Ainsi, il serait légitime de s'interroger sur la pertinence de créer et de renseigner ces derniers dynamiquement et pourquoi ne pas se contenter de déployer l'ensemble des contraintes dans l'ensemble des stores possibles en amont.

[1] Wolfram Research Mathematica : http://www.wolfram.com/mathematica

A y regarder de près, il se trouve que l'ensemble de toutes les contraintes présentes dans toutes les règles représente un modèle génératif de comportement.

En effet, considérons la règle suivante :

$$Acceleration \rightarrow$$
$$[capteur: C1, valeur: V1, entite: E, propriete: vitesse] \cdot$$
$$[capteur: C2, valeur: V2, entite: E, propriete: vitesse]$$
$$| V1 < V2.$$

Nous pourrions la traduire par :

$$\exists m_1 \in o_n^*, m_2 \in o_{n+1}^* \; tq$$
$$m_1: property = m_2: property = vitesse$$
$$\wedge \; m_1: entity = m_2: entity$$
$$\wedge \; m_1: value < m_2: value$$

C'est-à-dire qu'il existe deux mesures, l'une appartenant à une observation à la date courante et l'autre appartenant à une observation à la date suivante, telles que le nom de leur champ propriété soit égal à $vitesse$ et qu'elles concernent la même entité et que la valeur mesurée de la première soit inférieure à la deuxième.

Nous pourrions ainsi déployer l'ensemble des règles possibles pour en extraire une formule booléenne que nous fournirions directement à notre solveur de contraintes.

Quelles raisons nous poussent donc ainsi à ne pas déployer l'intégralité de nos contraintes dans notre solveur dès le démarrage ?

## La nature dynamique de la reconnaissance en ligne

Comme nous l'avons précédemment cité, nous nous intéressons à des applications où la reconnaissance de comportement a lieu de manière simultanée et synchronisée avec la prise de mesures de la part de notre système de capteurs. Ainsi, bon nombre de variables concernant des observations futures non encore mesurées ont pour valeur $[-\infty, +\infty]$ pour le cas de variables numériques ou $undefined$ dans le cas de variables symboliques. Il est donc inutile et de surcroit couteux de fournir au solveur des contraintes qui ne peuvent être résolues.

## L'impossibilité du déploiement total de l'ensemble des contraintes arithmétiques

Nous avons précédemment évoqué le fait qu'il était utopique de penser pouvoir générer à priori l'ensemble des comportements de la scène (suite de

clichés d'observations) que l'on souhaite reconnaitre avant de chercher à les confronter aux observations.

Rappelons que les arguments évoqués étaient notamment les domaines continus ou pseudo continus des variables représentant les valeurs des propriétés de nos entités, ainsi que leur grand nombre.

L'ensemble des contraintes étant lui-même un modèle générateur des comportements de la scène que l'on souhaite reconnaitre, on pourrait estimer que ce dernier peut être complètement déployé. Cependant, cela serait sans compter les classes de comportements à durée indéterminée (concernant un nombre variable de clichés).

Par exemple, prenons l'ensemble de règles récursives suivant :

$$AccelerationContinue(previousSpeed{:}\,PS, speed{:}\,S, entite{:}\,E) \rightarrow [capteur{:}\,C1, valeur{:}\,S, entite{:}\,E, propriete{:}\,vitesse]\text{◂}\ AccelerationContinue(previousSpeed{:}\,S, entite{:}\,E).$$

$$\mid PS < S$$

$$AccelerationContinue(previousSpeed{:}\,PS, speed{:}\,S, entite{:}\,E) \rightarrow [capteur{:}\,C1, valeur{:}\,S, entite{:}\,E, propriete{:}\,vitesse].$$

$$\mid PS < S$$

$$ResterSurPlace(posX{:}\,X, posY{:}\,Y, entite{:}\,E) \rightarrow Position(X{:}\,X, Y{:}\,Y, entite{:}\,E)\text{◂}\ ResterSurPlace(posX{:}\,X, posY{:}\,Y, entite{:}\,E).$$

$$ResterSurPlace(posX{:}\,X, posY{:}\,Y, entite{:}\,E) \rightarrow Position(X{:}\,X, Y{:}\,Y, entite{:}\,E).$$

Profitons-en ici pour remarquer que les paramètres/arguments de nos règles ne sont pas des paramètres positionnels, mais nommées et optionnels, tels que l'on peut en voir dans des langages comme le C#. L'intérêt de posséder des arguments nommés et optionnels plutôt que positionnels réside principalement dans le fait que de nombreuses règles peuvent intervenir pour un même symbole non terminal. Et à chacune de ces règles correspond des arguments qui ne peuvent intervenir que dans cette dernière. Il serait donc dommage de devoir les repréciser dans la définition des autres règles. De plus cela permet de pouvoir utiliser un symbole sans connaitre l'intégralité des paramètres qui lui sont associés. Ainsi, seul la sémantique concernant la règle apparait dans cette dernière et la sémantique des autres règles ne s'y retrouve pas.

Et imaginons le symbole génératif de notre grammaire tel que :

$$Comportement(entite: E) \rightarrow ResterSurPlace(entite: E) \cdot AccelerationContinue(entite: E).$$

La grammaire ci-dessus peut être validée pour n'importe quelle entité qui reste sur place pendant un certain temps, puis qui se met à accélérer pendant un certain temps.

En admettant que trois entités soient sous surveillance, l'une accélérant tout le long des 5 clichés, et l'autre restant sur place durant les 5 clichés, et enfin une dernière ne commençant à accélérer qu'au bout de 3 clichés. Les trois arbres de reconnaissance produits seront les suivants :

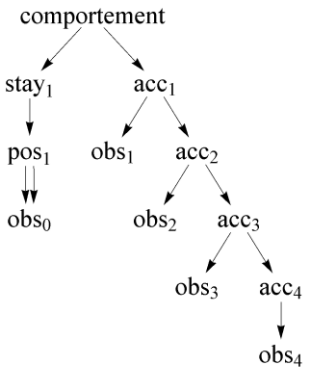


**Figure 4-9 : Arbre de reconnaissance de l'entité accélérant depuis le début.**

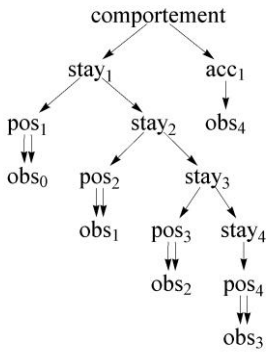


**Figure 4-10 : Arbre de reconnaissance de l'entité restant sur place jusqu'à la fin.**

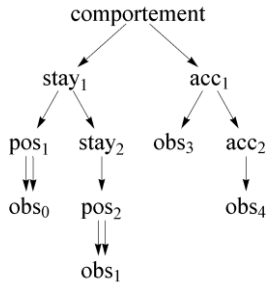


**Figure 4-11 : Arbre de reconnaissance de l'entité n'accélérant qu'à partir du troisième cliché.**

Nous pouvons constater que d'autres comportements également pouvaient être envisagés comme accélérer à partir du deuxième ou quatrième cliché.

Imaginons maintenant devoir écrire directement un ensemble de contraintes valides pour le comportement de nos trois entités avec $numobs$ le nombre de clichés :

$$\forall i,j \in \{1,\dots,numobs\}, \exists m_i, n\text{-}_i \in o_i^*, k\epsilon\{1,\dots,numobs-1\}tq$$
$$\forall i,j \in \{1,\dots,numobs\}, m_i{:}\,entity = m_j{:}\,entity = n_i{:}\,entity$$
$$= n_j{:}\,entity \wedge$$
$$\forall a \in \{1,\dots,k\}, \forall b \in \{1,\dots,k\}, m_a{:}\,property = x \wedge n_a{:}\,property$$
$$= y \wedge m_a{:}\,value = m_b{:}\,value \wedge, n_a{:}\,value = n_b{:}\,value \wedge$$
$$((k \leq numbobs - 2 \wedge \forall c \in \{k+1, numobs-1\}, m_c{:}\,property =$$
$$= vitesse \wedge m_c{:}\,value < m_{c+1}{:}\,value) \vee k$$
$$= numobs - 1)$$

La génération de toutes les contraintes exemptes de quantificateurs à partir de la formule ci-dessus engendre un nombre de tests d'inégalités et égalités égal à :

$$4 * numobs^2 + (\sum_{k=1}^{numobs-1} (4*k)^2) + numobs = 6numobs^2 - numobs$$

Ainsi, pour 5 clichés, 145 contraintes d'inégalités ou d'égalités devraient être insérées dans le solveur, et pour 100 clichés, 59900. Ceci est évidemment trop important pour pouvoir espérer un quelconque traitement temps réel. Extrapoler cette constatation à une véritable base de connaissances constituée de plusieurs centaines de règles rend l'hypothèse de représenter des comportements par un ensemble de conjonctions ou de disjonctions de contraintes arithmétiques irréalisable dans la pratique.

Dans le cas présent, notre système assurerait le maintien d'un maximum de 4 contraintes dans le store par entité à chaque pas de temps, peu importe le

nombre total de clichés, les contraintes concernant les observations passées étant retirées de chaque store de contraintes.

## Expressivité, usage

Enfin, si nous abordons des critères de facilité de conception tels que la modularité, la lisibilité et bien que ces derniers soient relativement subjectifs, il nous apparait évident qu'un modèle hiérarchique par nature, imposant une liaison explicite des concepts sémantiques d'un niveau à ceux du niveau supérieur ne peut qu'être favorable en comparaison d'un ensemble de contraintes 'à plat'.

## La vraie nature de notre système

Lorsque l'on regarde notre ensemble modèle/algorithme, nous nous apercevons qu'il permet donc d'assurer le maintien de stores de contraintes, d'éviter de le surcharger avec des variables et relations qui ne peuvent encore être réduites, et en éliminant au fur et à mesure celles qui ne peuvent plus évoluer.

Ainsi, si un ensemble de contraintes peut être considéré comme un modèle génératif de comportements, notre modèle peut donc être vu comme un méta-modèle génératif de comportements.

# 4.3 Prérequis au système

## 4.3.1 Système d'interpolation/extrapolation

Nous avons insistés depuis le début du manuscrit sur l'importance de devoir effectuer le moins de prétraitements systématiques possibles sur les données brutes en provenance des capteurs. Cependant, nous devons cependant remarquer qu'une des hypothèses nécessaires pour pouvoir écrire nos règles et notre grammaire de comportement est que l'on puisse supposer que les observations arrivent sous forme d'un flux continu certes, mais surtout ***régulier***.

En effet, il est très difficile de décrire des règles bas niveau sans présupposer de cette régularité. Les données numériques imprécises et manquantes sont gérées grâce à la présence du solveur de contraintes sur intervalles Interlog, nous allons maintenant proposer un système nous permettant de compenser la non régularité de l'arrivée des données en provenance du système de capteurs.

En réalité, nous allons nous appuyer sur le fait que nous gérons déjà les données imprécises et/ou manquantes pour pouvoir traiter le cas de flux de données non régulier plus simplement.

Nous ne souhaitons évidemment pas que cette non régularité soit à prendre en compte dans l'écriture de notre modèle génératif de comportement. Nous allons donc accepter ici d'atténuer notre volonté de ne pas prétraiter systématiquement nos données d'entrée et interposer entre notre système de capteur et notre système de reconnaissance de comportement un système d'interpolation/extrapolation de données comme l'on peut le voir sur le schéma ci-dessous.

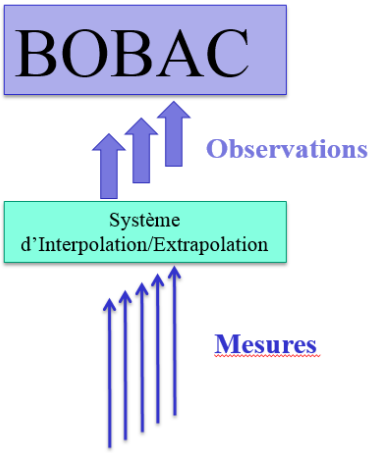


**Figure 4-12 : Un système d'interpolation/extrapolation systématique des mesures est nécessaire**

Nous allons  nous servir de ce système pour réaliser plusieurs tâches.

## Former les observations à partir des mesures

La première tâche de notre système d'interpolation/extrapolation est de former à partir des mesures brutes issues de notre système de capteurs nos observations contenant l'ensemble des valeurs des propriétés mesurées des entités sous surveillance à partir de l'ensemble des capteurs. On peut donc imaginer que le but de ce système est de fournir à intervalle régulier une matrice de taille $N_c \times N_e \times N_p$ et dans laquelle sont stockés les intervalles bornant les valeurs mesurées par notre système de capteurs ou les valeurs symboliques de ces dernières s'il y a lieu.

Il est donc attendu que le système d'interpolation/extrapolation génère à intervalle régulier une telle matrice, laquelle constituera une observation qui sera ensuite transmise à notre système.

Le delta temporel séparant ces émissions d'observations doit pouvoir être modifié selon le domaine applicatif concerné.

## Interpolation/Extrapolation des intervalles

Bien entendu, il est fort peu probable qu'en entrée de ce système les mesures provenant du système de capteurs arrivent à intervalle régulier. Il est même fortement probable que chaque capteur fournira des données à un rythme différent des autres, et pas forcément de manière régulière.

Par exemple il est envisageable qu'une partie de la zone sous surveillance ne soit pas couverte à tout moment par un capteur, on parlera de zones d'ombre. Il faut pourtant pouvoir continuer à prendre en compte les autres mesures arrivant sur d'autres entités et envoyer des observations complètes de notre scène pour que notre système de reconnaissance de comportement puisse continuer de déployer ses hypothèses.

La tâche revient donc à notre système intermédiaire d'extrapoler à partir des $n$ dernières mesures récupérées une nouvelle valeur cohérente à renseigner dans l'observation en cours de génération.

Dans le cas d'un capteur estimant la position d'une entité, le système, à partir d'un modèle de déplacement défini agrandira d'un delta approprié l'intervalle de la position précédente (très certainement en tenant compte de la vitesse maximale pouvant être adoptée par l'entité).

Illustrons cet exemple grâce à la figure suivante :

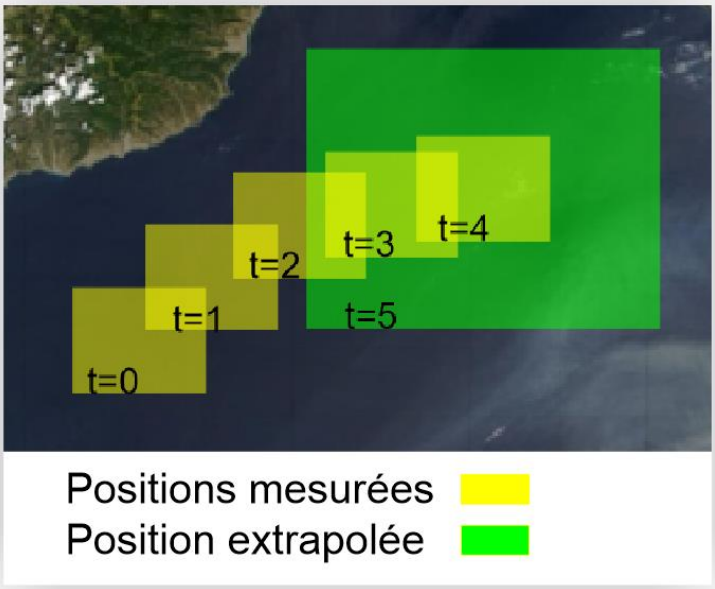


**Figure 4-13 : Illustration de l'extrapolation d'une mesure en vue de renseigner l'observation numéro 5 même si le système de capteurs ne fournit pas de mesure à cet instant précis.**

Dans la figure ci-dessus l'on suppose avoir reçu une mesure nous permettant de construire les quatre premières observations, et ensuite ne pas avoir de mesure pour la cinquième, à ce moment-là le système doit extrapoler la position courante à partir d'ici la dernière position et de la vitesse maximale de l'entité sous surveillance.

Dans un contexte moins schématique et plus réel, le système d'extrapolation/interpolation réalise en permanence l'extrapolation car la probabilité d'obtenir une mesure correspondant exactement au pas de temps fixé par le système est très faible. Cela rajoute évidemment de l'imprécision artificielle à nos données d'entrées, mais dans la pratique n'est pas préjudiciable à l'efficacité de la reconnaissance des comportements.

Enfin, si le modèle d'extrapolation n'est pas disponible ou qu'aucune mesure ne peut être utilisée pour la réaliser, le système d'extrapolation se contentera de renvoyer un intervalle $[-\infty; +\infty]$ en lieu et place de la mesure comme nous l'avons précédemment cité.

Il ne faut pas oublier non plus que selon les règles et les contraintes présentes dans la base de connaissance sur les comportements peuvent également contribuer à déterminer la valeur d'une mesure qui n'a pas été fournie par le système d'interpolation/extrapolation.

En effet, notre système de reconnaissance peut également être vu comme un système de génération de comportement lorsque les mesures en entrées sont trop imprécises.

Ainsi, pour pouvoir générer tous les comportements possibles pouvant être générés à partir d'une instance de notre modèle de grammaire, il suffirait de ne fournir que des observations contenant des mesures extrapolées à $[-\infty, +\infty]$ et toutes les hypothèses (tous les comportements) seraient ainsi produits. Bien entendu pour la plupart des bases de connaissance, cela provoquerait une explosion mémoire, mais cet usage renversé peut être intéressant dans certains cas.

# Chapitre 5
# Implémentation

# 5.1 Le système BOBAC : application à la surveillance maritime

Nous nous sommes donc attachés à produire le système BOat Behaviour Analysis and Characterization, mettant en œuvre les algorithmes permettant de prendre en compte une base de données sur les comportements écrites sous la forme d'une grammaire telle que précédemment définie.

Ce système se décompose en plusieurs parties :

- Un simulateur
- Un capteur simulant des prises de mesures imparfaites régulières sur nos entités
- Un outil de reconnaissance de comportement calqué sur nos propositions et une interface de visualisation des résultats de ce dernier.

D'autre part, et pour faciliter la production de situation différentes a été développé au sein du simulateur un outil d'édition de scénario.

Enfin, pour faciliter l'écriture de bases de connaissances sur les comportements a été développé un outil d'aide à l'écriture des règles, notamment pour s'assurer de vérifier la cohérence de la grammaire produite.

Nous nommons ainsi l'ensemble de ces outils : le système BOBAC.

### 5.1.1 Simulateur de scénario de navigation maritime

Dans les premiers mois de nos travaux, notre première action a été de concevoir un simulateur basique de navigation maritime. En effet, du fait de l'absence d'ensembles de comportements étiquetés, nous nous sommes fixés comme objectif d'en générer nous-mêmes dans le but de pouvoir mieux les comprendre et les analyser.

N'oublions pas que *savoir générer c'est savoir reconnaître*. En effet, les modèles génératifs de comportements dont nous avons parlé peuvent servir aussi bien à la génération de comportement qu'à la reconnaissance de comportement.

Nous nous sommes attachés à reproduire dans un premier temps des comportements de voiliers.

Ces derniers étant fortement dépendants du vent pour déterminer leur trajectoire, parfois se rendre d'un point à l'autre n'est pas forcément une ligne droite comme le montre la figure ci-dessous.

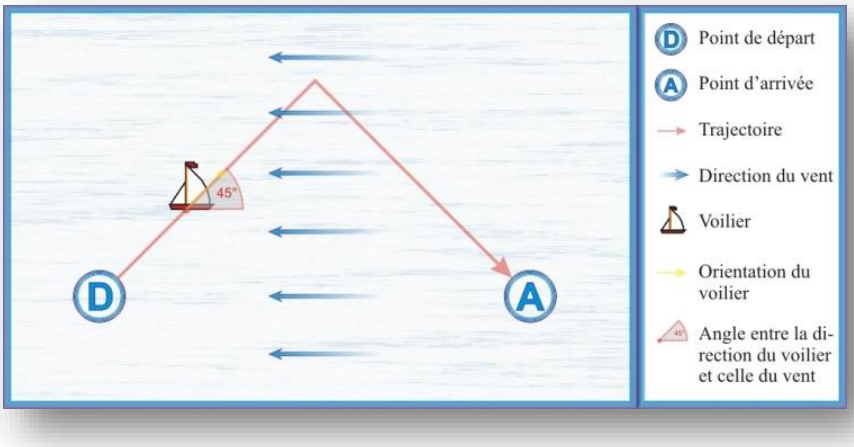


**Figure 5-1 : Schéma représentant un voilier souhaitant *remonter au vent*.**

De plus, ces derniers doivent éviter des obstacles comme des îles ou îlots. Sachant que les opérations de changements de caps peuvent être coûteuses, il est souvent préféré de minimiser ces dernières au prix d'une distance plus grande parcourue comme le montre la figure ci-dessous.

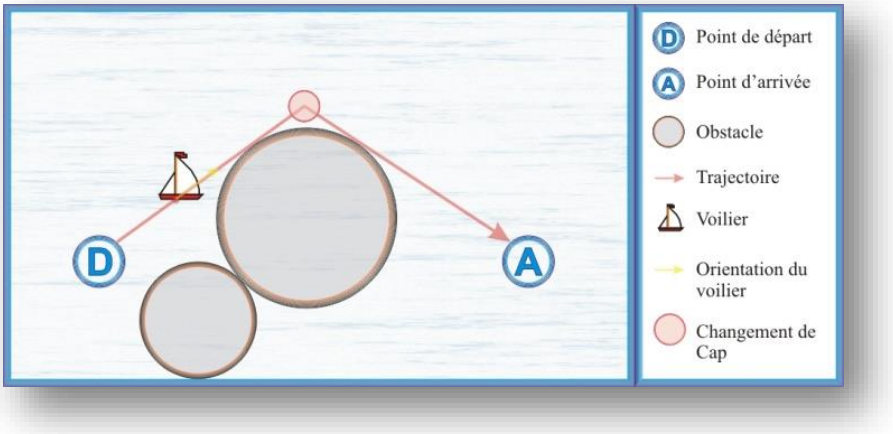


**Figure 5-2 : Voilier contournant un obstacle en n'effectuant qu'un seul changement de cap.**

Ainsi, au sein d'un environnement chargé, un voilier pourrait devoir prendre en compte ces trois critères (minimiser les changements de caps, tenir compte du vent et éviter les obstacles) pour déterminer sa trajectoire. Ceci est traduit par la figure ci-dessous où le vent est considéré comme un obstacle virtuel.

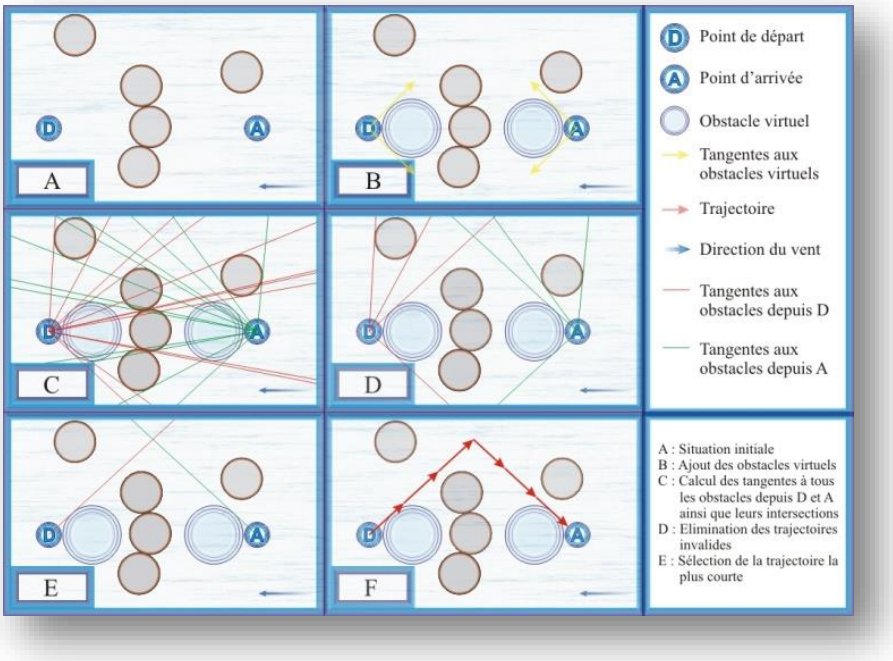


**Figure 5-3 : Exemple de construction de la trajectoire d'un voilier prenant en compte le vent, les obstacles et la minimisation du nombre de changements de cap.**

Nous nous sommes ainsi servis de ce type d'entité dans notre simulateur pour générer des données/mesures à partir desquelles nous pouvant tenter de reconnaitre des comportements typiques.

### Le simulateur du système BOBAC

Fort de cette expérience, nous avons commencé la réalisation du système BOBAC en commençant par un nouveau simulateur associé à un éditeur de scenario.

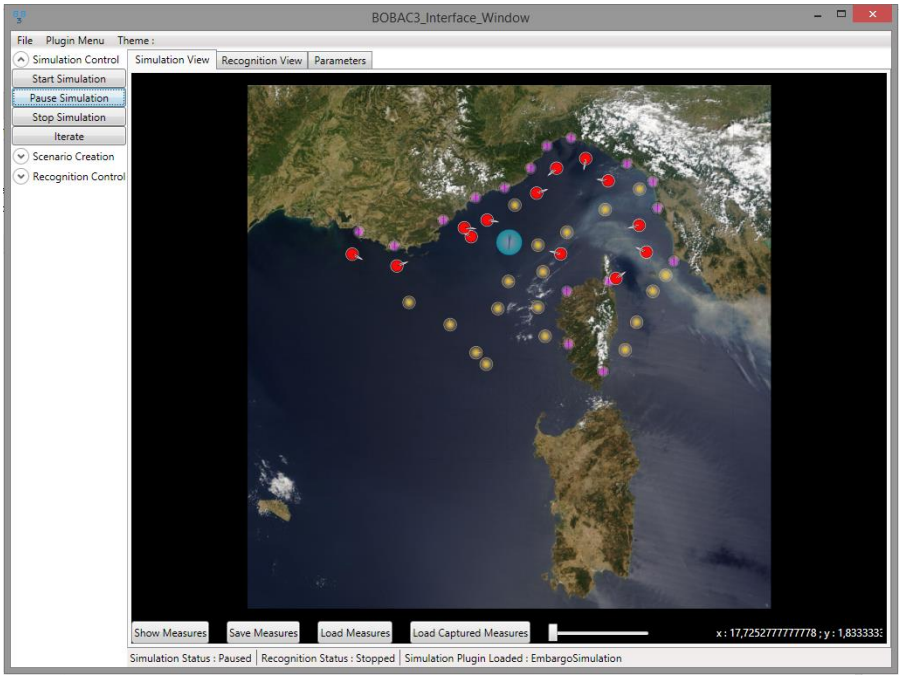


**Figure 5-4 : Capture d'écran de la fenêtre de simulation du logiciel BOBAC.**

Nous pouvons ainsi charger des fichiers décrivant des simulations diverses, comme celle ci-dessus où une vingtaine de bateaux (de type voiliers et cargo) évoluent sur le plan d'eau. Seuls un petit nombre de ports y sont ici représentés.

## 5.1.2 **Editeur de scénario**

Les scenarii utilisés dans le système BOBAC ont tous été produits à l'aide d'un outil de création de scenario. Ce dernier à partir d'un fichier de configuration contenant l'ensemble des types d'entités et de leurs propriétés associées génère automatiquement les outils et interfaces contextuelles permettant d'ajouter des entités à la zone sous surveillance et de leur assigner une trajectoire de manière intuitive.

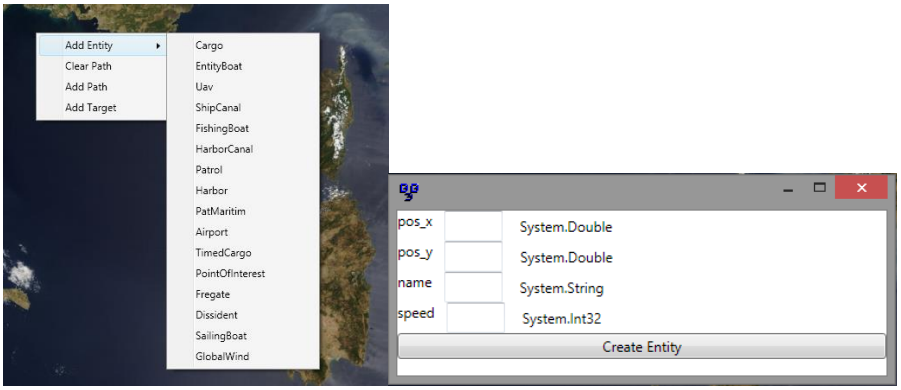


**Figure 5-5 : Interfaces automatiquement générées à partir d'une description de type de simulation permettant d'ajouter et configurer différentes entités aisément.**

Comme le montre la Figure ci-dessus, il est très aisé d'ajouter et de configurer les différents paramètres relatifs au type d'entité en cours d'ajout. De plus ces paramètres seront automatiquement prises en compte par le système global de capteurs virtuels que nous nous apprétons à décrire et ces sur les mesures de ces dernières que porteront les contraintes décrites dans nos règles.

### 5.1.3 **Capteur simulé**

Lorsque nous lançons la simulation, les bateaux commencent à suivre la trajectoire qui leur a été assignée. En simultané, un système de capteur imparfait simulé commence à enregistrer les différentes valeurs des propriétés de chacune de nos entités (comme la position des voiliers, cargos, leur vitesse, leur orientation, leur vitesse instantanée, etc.) et transforme chaque valeur numérique en intervalle contenant cette valeur, intervalle n'étant pas obligatoirement centré sur cette dernière.

Nous disposons également d'une interface pour pouvoir au besoin visualiser l'ensemble des données capturées au cours de la simulation comme indiqué sur la figure suivante :


**Figure 5-6 : Capture d'écran de la fenêtre de visualisation des données mesurées et artificiellement bruitées au cours de la simulation.**

## 5.1.4 **Outil d'aide à l'écriture d'une grammaire**

Comme précédemment énoncé, nous disposons également d'un IDE sommaire pour nous aider à construire un ensemble de règles formant une grammaire cohérente et utilisable dans le système BOBAC. Cet outil est composé d'une fenêtre permettant de lister l'ensemble des règles produites et d'en importer/exporter.Ceci est présenté sur la capture d'écran ci-dessous :

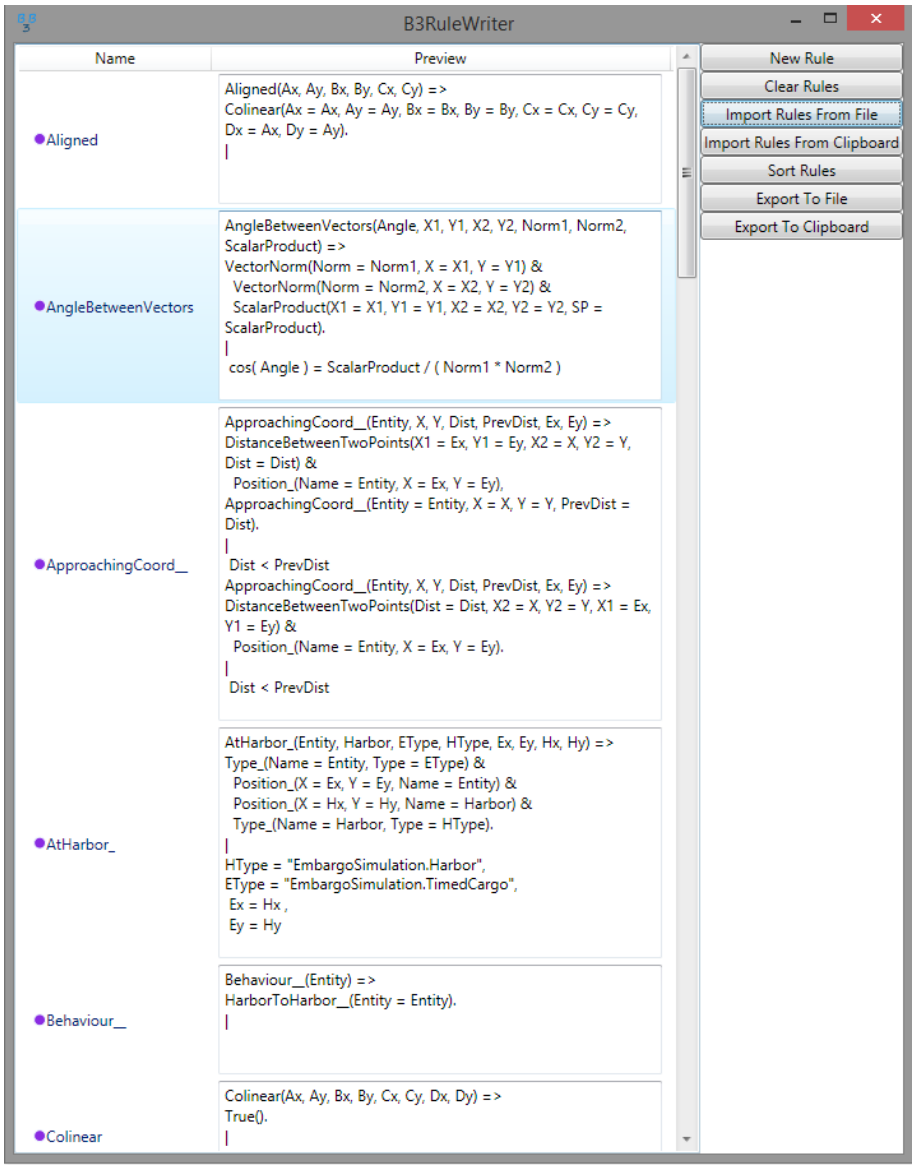


**Figure 5-7 : Capture d'écran de la fenêtre de manipulation de l'ensemble des règles d'une grammaire représentant les comportements possibles des entités sous surveillance.**

De même, nous proposons un outil d'aide à l'écriture de règle. Voici ci-dessous une capture d'écran présentant l'interface de cet outil :


**Figure 5-8 : Capture d'écran de l'outil d'aide à l'écriture de règle.**

Cet outil est essentiel pour permettre de détecter au plus tôt des erreurs dans l'écriture des règles comme par exemple l'utilisation dans une règle d'un symbole non défini. Nous pourrions le voir comme un IDE nous aidant à écrire les règles d'une grammaire, en lieu et place de l'habituelle complétion de l'éditeur de texte se trouvent ici des menus déroulants et autres composants graphiques.

### 5.1.5 **Visualisation du résultat de la reconnaissance**

Enfin, nous proposons une fenêtre dans laquelle est affichée en temps réel arbres de reconnaissance syntaxiques correspondant à autant d'hypothèses en cours de reconnaissance. Ci-dessous se trouve un exemple de cette interface.

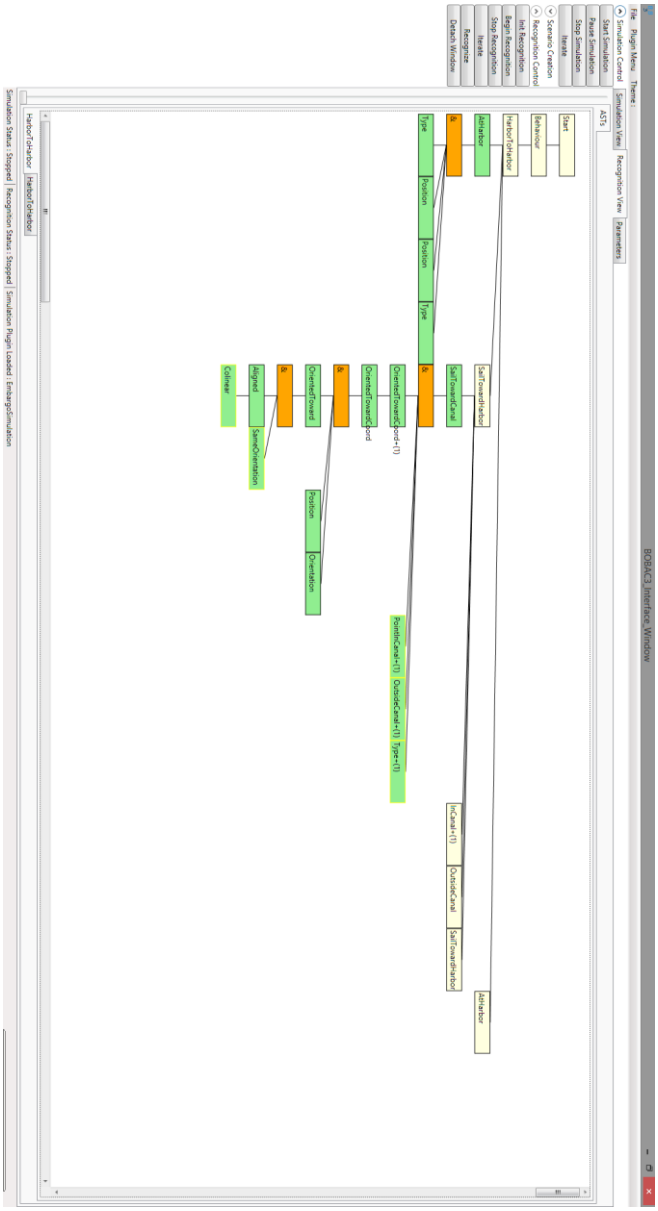

**Figure 5-9 : Capture d'écran de l'interface de visualisation des hypothèses sur les comportements en cours de reconnaissance.**

Comme nous l'avons déjà mentionné, nous pensons que fournir une visualisation de l'arbre de syntaxe abstraite en cours de construction est très pertinent pour un opérateur humain. En effet, sur ce dernier peut être observé les étapes ayant été complètement reconnues (en vert sur la capture d'écran) et celles en cours de reconnaissance ou que l'on s'attend selon l'hypothèse choisie à reconnaître dans futur.

De même le niveau de granularité peut être modifié en temps réel et il est possible de développer les nœuds en cliquant simplement dessus jusqu'à parvenir aux symboles directement en relations avec les mesures.

De même nous pouvons voir au bas de l'interface plusieurs onglets, chacun de ces onglets correspond à une hypothèse correspondant à un autre comportement possible et différent, mais pouvant également être validé sur le même flux d'observation.

Ainsi à partir de cet arbre de syntaxe abstraite, nous proposons à l'opérateur humain à la fois :

- L'ensemble des hypothèses sur les comportements en cours de reconnaissance.
- Une décomposition hiérarchique des étapes reconnues pouvant tenir lieu d'explication permettant à l'opérateur humain de valider ou d'invalider les étapes reconnues par le système.
- Une prédiction pour chaque hypothèse des étapes restantes à valider, ce qui permet au besoin de lever des alertes en amont et d'éviter à une situation potentiellement non désirée d'arriver.

### 5.1.6 **Particularités techniques de l'implémentation**

#### Version principale : BOBAC 3.2

Bien que plusieurs versions du système BOBAC existent, la version principalement utilisée a été développement sur deux socles technologiques :

- Microsoft .NET[2]
- Sicstus Prolog[3]

Le modèle de règle présenté peut être vu comme un langage dit CLI[4] étant traduit en langage C#, lui-même compilé dynamiquement par le système

[2] http://msdn.microsoft.com/fr-fr/library/aa496123.aspx

[3] http://sicstus.sics.se/

[4] Common Langage Infrastructure

BOBAC en CIL[5] .NET. Ceci permet ainsi de se reposer sur le compilateur C# proposé par Microsoft ou encore celui de Mono pour pouvoir générer des messages d'erreurs relatifs à la compilation de la grammaire.

Le logiciel BOBAC est une application WPF[6] s'interfaçant avec un ensemble de processus Sicstus Prolog dans lesquels une version interactive du solveur de contraintes sur intervalles Interlog s'exécute. Ceci nous permet de paralléliser le traitement de chacune de nos hypothèses car ces dernières sont indépendantes les unes des autres.

L'application principale a été pensée sous la forme d'application hôte de plugins respectant le modèle des applications extensibles[7]. Un type de simulation est en réalité un *AddIn* pour l'application hôte.

## Utilisation de Wolfram Research Mathematica

Une version de BOBAC s'interfaçant avec un kernel Mathematica a également été développée, ce qui nous a permis d'invalider l'utilisation de solveur symboliques exacts pour des raisons de performances.

D'autre part, une version sommaire des algorithmes utilisés et du modèle conceptuel de règle a été également développé entièrement au sein de l'écosystème Mathematica, principalement pour des tests préliminaires, le code source étant compact, nous l'avons rendu disponible en annexe.

[5] Common Intermediate Language

[6] http://fr.wikipedia.org/wiki/Windows_Presentation_Foundation

[7] http://msdn.microsoft.com/en-us/library/bb909887(v=vs.110).aspx

# 5.2 Exemple de bibliothèque de règles

Au cours de nos travaux, nous avons été amenés à écrire plusieurs bases de connaissances sur les comportements. Ces bases de connaissances pour la plupart ont toutes pour contexte la surveillance maritime. Dans ces dernières, et au-delà des relations géométriques nécessaires à l'écriture de certaines règles, nous avons utilisé une dizaine de grammaires distinctes comportant en moyenne plusieurs dizaines de règles chacune dans lesquelles nous pouvions trouver des comportements de bateaux décrivant les trajectoires pour aller d'un point à un autre.

Nous nous proposons ainsi d'en retranscrire une simple de manière exhaustive, pour permettre au lecteur de mieux comprendre le système déployé.

Il est à noter que la syntaxe diffère légèrement de celle présentée dans ce manuscrit, en effet certaines additions n'ont pu être apportées à l'outil d'édition de règles. Cependant le lecteur devrait retrouver les concepts préalablement énoncés de manière aisée.

La bibliothèque de règle présentée ci-dessous est succincte, et simpliste, mais permet la reconnaissance du comportement d'un navire quittant un port et pouvant soit se diriger immédiatement vers un autre port, ou soit emprunter un canal de navigation pendant un certain temps avant de rejoindre un autre port.

## 5.2.1 Règles liées au domaine

### Règle de départ

```
Behaviour__(Entity) =>
    HarborToHarbor__(Entity = Entity).|
```

### Aller d'un port à un autre

```
HarborToHarbor__(Entity, Harbor1, Harbor2) =>
    AtHarbor_(Entity = Entity, Harbor = Harbor1),
    SailTowardHarbor__(Entity = Entity, Harbor = Harbor2),
    AtHarbor_(Entity = Entity, Harbor = Harbor2).
    |
```

## Etre au port

```
AtHarbor_(Entity, Harbor, EType, HType, Ex, Ey, Hx, Hy) =>
    Position_(X = Ex, Y = Ey, Name = Entity) &
    Position_(X = Hx, Y = Hy, Name = Harbor) &
    Type_(Name = Entity, Type = EType) &
    Type_(Name = Harbor, Type = HType).
    | HType = "EmbargoSimulation.Harbor",EType =
    "EmbargoSimulation.TimedCargo", Ex = Hx , Ey = Hy
```

## Naviguer vers un port de destination

```
SailTowardHarbor__(Entity, Harbor, HarborType, Canal, Hx, Hy) =>
    SailTowardPoint__(Entity = Entity, X = Hx, Y = Hy) &
    Position_(Name = Harbor, X = Hx, Y = Hy)+ &
    Type_(Type = HarborType, Name = Harbor)+.
    |HarborType = "EmbargoSimulation.Harbor", Hx = {-inf, +inf} , Hy = {-inf, +inf}
SailTowardHarbor__(Entity, Harbor, HarborType, Canal, Hx, Hy) =>
    SailTowardCanal__(Entity = Entity, Canal = Canal),
    InCanal_(Canal = Canal, Entity = Entity)+,
    OutsideCanal_(Entity = Entity, Canal = Canal),
    SailTowardPoint__(Entity = Entity, X = Hx, Y = Hy) &
    Position_(Name = Harbor, X = Hx, Y = Hy)+ &
    Type_(Type = HarborType, Name = Harbor)+.
    | HarborType = "EmbargoSimulation.Harbor", Hx = {-inf, +inf} , Hy = {-inf, +inf}
```

## Naviguer vers un canal de navigation

```
SailTowardCanal__(Entity, Canal, WPx, WPy) =>
    OrientedTowardCoord_(Entity = Entity, Tx = WPx, Ty = WPy)+ &
    PointInCanal_(Canal = Canal, X = WPx, Y = WPy)+ &
    OutsideCanal_(Entity = Entity, Canal = Canal)+,
    InCanal_(Entity = Entity, Canal = Canal).
    | WPx = {-inf, +inf} , WPy = {-inf, +inf}
```

## Appartenance d'un point à un canal de navigation

```
PointInCanal_(Canal, CLenght, CWidth, Cx, Cy, X, Y, CAnglex, CAngley) =>
    InRectangleArea(OLenght = CLenght, OWidth = CWidth, Ox = Cx, Oy = Cy, Ex =
    X, Ey = Y, OAnglex = CAnglex, OAngley = CAngley) &
    Lenght_(Name = Canal, Lenght = CLenght) &
    Width_(Name = Canal, Width = CWidth) &
    Position_(Name = Canal, X = Cx, Y = Cy) &
```

```
    Orientation_(Name = Canal, Anglex = CAnglex, Angley = CAngley).
    |
```

## Non appartenance d'un point à un canal de navigation

```
PointOutsideCanal_(Canal, CLenght, CWidth, Cx, Cy, X, Y, Canglex, Cangley) =>
    Lenght_(Name = Canal, Lenght = CLenght) &  Width_(Name = Canal, Width =
    CWidth) &  Position_(Name = Canal, X = Cx, Y = Cy) &
    Orientation_(Name = Canal, Anglex = Canglex, Angley = Cangley) &
    OutsideRectangleArea(OLenght = CLenght, OWidth = CWidth, Ox = Cx, Oy = Cy,
    Ex = X, Ey = Y, OAnglex = Canglex, OAngley = Cangley).
    |
```

## Etre en dehors d'un canal de navigation

```
OutsideCanal_(Entity, Canal, Ex, Ey) =>
    Position_(Name = Entity, X = Ex, Y = Ey) &
    PointOutsideCanal_(X = Ex, Y = Ey, Canal = Canal).
    |
```

## Naviguer vers un point

```
SailTowardPoint__(Entity, X, Y) =>
    StraightSailTowardPoint__(Entity = Entity, X = X, Y = Y),
    Position_(Name = Entity, X = X, Y = Y).
    |
```

## Naviguer vers un point en ligne droite

```
StraightSailTowardPoint__(Entity, X, Y) =>
    OrientedTowardCoord_(Entity = Entity, Tx = X, Ty = Y)+ &
    ApproachingCoord__(Entity = Entity, X = X, Y = Y).
    |
```

## Se rapprocher d'un point

```
ApproachingCoord__(Entity, X, Y, Dist, PrevDist, Ex, Ey) =>
    DistanceBetweenTwoPoints(X1 = Ex, Y1 = Ey, X2 = X, Y2 = Y, Dist = Dist) &
    Position_(Name = Entity, X = Ex, Y = Ey),
    ApproachingCoord__(Entity = Entity, X = X, Y = Y, PrevDist = Dist).
    | Dist < PrevDist
ApproachingCoord__(Entity, X, Y, Dist, PrevDist, Ex, Ey) =>
    Position_(Name = Entity, X = Ex, Y = Ey) &
```

```
    DistanceBetweenTwoPoints(Dist = Dist, X2 = X, Y2 = Y, X1 = Ex, Y1 = Ey).
    | Dist < PrevDist
```

### Etre orienté vers un point

```
OrientedTowardCoord_(Entity, Ex, Ey, Tx, Ty, EAnglex, EAngley) =>
    Position_(X = Ex, Y = Ey, Name = Entity) &
    Orientation_(Name = Entity, Anglex = EAnglex, Angley = EAngley) &
    OrientedToward(Ex = Ex, Ey = Ey, Tx = Tx, Ty = Ty, EAngley = EAngley, EAnglex =
    EAnglex).
    |
```

### Etre dans un canal de navigation

```
InCanal_(Entity, Canal, Ex, Ey) =>
    Position_(Name = Entity, X = Ex, Y = Ey) &
    PointInCanal_(Canal = Canal, X = Ex, Y = Ey).
    |
```

### Avoir atteint un point

```
ReachedPoint_(Entity, X, Y, Ex, Ey) =>
    Position_(Name = Entity, X = Ex, Y = Ey).
    | Ex = X , Ey = Y
```

## 5.2.2 **Relations géométriques usuelles**

### Alignement entre trois points

```
Aligned(Ax, Ay, Bx, By, Cx, Cy) =>
    Colinear(Ax = Ax, Ay = Ay, Bx = Bx, By = By, Cx = Cx, Cy = Cy, Dx = Ax, Dy = Ay).
    |
```

### Mesure d'angle entre deux vecteurs

```
AngleBetweenVectors(Angle, X1, Y1, X2, Y2, Norm1, Norm2, ScalarProduct) =>
    VectorNorm(Norm = Norm1, X = X1, Y = Y1) &
    VectorNorm(Norm = Norm2, X = X2, Y = Y2) &
    ScalarProduct(X1 = X1, Y1 = Y1, X2 = X2, Y2 = Y2, SP = ScalarProduct).
    | cos( Angle ) = ScalarProduct / ( Norm1 * Norm2 )
```

## Colinéarité

```
Colinear(Ax, Ay, Bx, By, Cx, Cy, Dx, Dy) =>
    True().
    | ( By - Ay ) * ( Cx - Dx ) - ( Bx - Ax ) * ( Cy - Dy ) = 0
```

## Distance euclidienne entre deux points

```
DistanceBetweenTwoPoints(Dist, X1, Y1, X2, Y2) =>
    True().
    | Dist = sqrt( ( X2 - X1 )^2 + ( Y2 - Y1 )^2 )
```

## Changement de référentiel

```
FrameOfRefSwitch(Ox, Oy, Ex, Ey, EOx, EOy, OAnglex, OAngley) =>
    True().
    | EOx = ( Ex - Ox ) * OAnglex + ( Ey - Oy ) * OAngley ,
    EOy = ( Ey - Oy ) * OAnglex - ( Ex - Ox ) * OAngley ,
    Ex = EOx * OAnglex - EOy * OAngley + Ox ,
    Ey = EOx * OAngley + EOy * OAnglex + Oy
```

## Appartenance d'un point à une zone rectangulaire orientée

```
InRectangleArea(Ox, Oy, Ex, Ey, EOy, EOx, OWidth, OLenght, OAnglex, OAngley) =>
    FrameOfRefSwitch(Ox = Ox, Oy = Oy, Ex = Ex, Ey = Ey, EOy = EOy, EOx = EOx,
    OAnglex = OAnglex, OAngley = OAngley).
    | abs( EOy ) < abs( OWidth ) / 2 , abs( EOx ) < abs( OLenght ) / 2
```

## Etre orienté vers un point

```
OrientedToward(Ex, Ey, Tx, Ty, Cx, Cy, EAnglex, EAngley) =>
    Aligned(Ax = Ex, Ay = Ey, Bx = Tx, By = Ty, Cx = Cx, Cy = Cy) &
    SameOrientation(Ax = Ex, Ay = Ey, Bx = Tx, By = Ty, Cx = Ex, Cy = Ey, Dx = Cx, Dy
    = Cy).
    | Cx = EAnglex + Ex , Cy = EAngley + Ey
```

## Non appartenance d'un point à une zone rectangulaire orientée

```
OutsideRectangleArea(Ox, Oy, Ex, Ey, EOy, EOx, OWidth, OLenght, OAnglex, OAngley)
=>
```

```
    FrameOfRefSwitch(Ox = Ox, Oy = Oy, Ex = Ex, Ey = Ey, EOy = EOy, EOx = EOx,
    OAnglex = OAnglex, OAngley = OAngley).
    | abs( EOx ) > abs( OLenght ) / 2 , abs( EOy ) > abs( OWidth ) / 2
OutsideRectangleArea(Ox, Oy, Ex, Ey, EOy, EOx, OWidth, OLenght, OAnglex, OAngley)
=>
    FrameOfRefSwitch(Ox = Ox, Oy = Oy, Ex = Ex, Ey = Ey, EOx = EOx, EOy = EOy,
    OAnglex = OAnglex, OAngley = OAngley).
    | abs( EOx ) < abs( OLenght ) / 2 , abs( EOy ) > abs( OWidth ) / 2
OutsideRectangleArea(Ox, Oy, Ex, Ey, EOy, EOx, OWidth, OLenght, OAnglex, OAngley)
=>
    FrameOfRefSwitch(Ox = Ox, Oy = Oy, Ex = Ex, Ey = Ey, EOx = EOx, EOy = EOy,
    OAnglex = OAnglex, OAngley = OAngley).
    | abs( EOx ) > abs( OLenght ) / 2 , abs( EOy ) < abs( OWidth ) / 2
```

## Deux vecteurs de sens non opposés

```
SameOrientation(Ax, Ay, Bx, By, Cx, Cy, Dx, Dy) =>
    True().
    | ( Bx - Ax ) * ( Dx - Cx ) + ( By - Ay ) * ( Dy - Cy ) > 0
```

## Produit scalaire

```
ScalarProduct(X1, Y1, X2, Y2, SP) =>
    True().
    | SP = X1 * X2 + Y1 * Y2
```

## Norme d'un vecteur

```
VectorNorm(Norm, X, Y) =>
    True().
    | Norm = sqrt( X ^2 + Y ^2 )
```

Le lecteur avertit aura compris que le symbole non terminal $True$ utilisé ici n'est autre que l'observation quelconque $[*]$ que nous avons préalablement cité.

# Chapitre 6

# Conclusion

# 6.1 Retour d'expérience

Nous avons donc proposé au travers de nos travaux un nouveau modèle capable de représenter des interactions spatiotemporelles complexes. Nous avons également vu l'importance de tisser au sein de cette représentation la sémantique extraite d'une expertise du domaine pour pouvoir fournir un résultat riche et justifié à un opérateur humain en charge de comprendre les enjeux de la scène sous surveillance et de prendre des décisions stratégiques.

Ce modèle a été proposé et publié (Vidal, Taillibert, & Aknine, 2010) lors de la conférence *International Conference on Tools with Artificial Intelligence* en 2010.

Nous avons proposé également un outil pour faciliter la saisie de cette expertise et nous permettant d'obtenir l'assurance de la cohérence de la bibliothèque de règles.

Enfin, nous nous sommes intéressés à un cadre applicatif particulier, celui de la surveillance des comportements de navires de plaisance et ou de transport de marchandises sur une zone maritime.

Nous avons développé un simulateur doublé d'un outil d'édition de scenarii pour nous permettre de générer des données et mesures que nous avons volontairement bruitées.

Enfin, grâce à notre système BOBAC, et à l'utilisation du solveur de contrainte sur intervalle Interlog, nous avons pu valider notre modèle de description de comportements, et montré la possibilité de gérer l'imprécision des mesures sans avoir à en impacter l'écriture des règles et contraintes décrivant les comportements.

## 6.1.1 Validation sur des données réelles

Nous n'avons pas pu prendre le temps de valider notre système sur des données réelles. Cependant ces dernières ont commencé à être collectées via des sites comme *Marine Traffic*[8] recensant toutes les positions, vitesses, et caps des navires dotés d'émetteurs AIS[9] sur les zones de la planète équipées de capteurs AIS.

C'est en partie sur ces mêmes données que la seconde perspective applicative décrite dans la section suivante sera évaluée.

[8] http://www.marinetraffic.com/

[9] Automatic Identification System

### 6.1.2 **Ecrire une grammaire efficace**

Parmi les difficultés rencontrées durant l'élaboration de notre modèle se trouve la volonté d'éviter de devoir se préoccuper de considérations de performances lors de l'écriture de la grammaire. Nous n'avons atteint qu'en partie cet objectif.

En effet, pour pouvoir représenter des comportements complexes et diverses, nous avons dû étendre le pouvoir d'expression de notre modèle. En laissant ainsi plus de libertés à la personne en charge de l'écriture des règles, nous lui offrons également la possibilité d'écrire des règles qui auront un impact négatif sur les performances de l'algorithme. Nous pensons être proches du juste milieu entre expressivité et performance vis-à-vis du cadre applicatif donné, mais il reste cependant une marge qu'il serait intéressant de chercher à réduire. Un travail reste donc à fournir pour réduire l'expressivité de notre modèle et en déduire les optimisations appropriées sur l'algorithme de reconnaissance de comportement.

# 6.2 Perspectives

## 6.2.1 Perspectives applicatives

Ces travaux sont présentement en train de donner lieu à de nouvelles applications.

### Application à la surveillance du comportement des personnes dépendantes

Une utilisation du modèle proposé dans nos travaux sera réalisée dans le cadre d'un projet d'aide aux personnes dépendantes au sein du LIRIS[10]. En effet, bien que le cadre applicatif traité ici ne soit pas identique, le modèle de règle reste suffisamment généraliste pour pouvoir être appliqué à d'autres domaines.

De même une partie du système BOBAC réalisé pourra être réutilisé car ayant été pensé pour pouvoir s'interfacer avec d'autres problématiques applicatives.

Il s'agira ici d'intégrer la dimension cognitive et pathologique des personnes atteintes de la maladie d'Alzheimer.

### Application à la reconnaissance de groupes de navires effectuant de la prospection sismique

Le modèle et l'ensemble des algorithmes proposés dans ce manuscrit seront également utilisés pour reconnaitre un type particulier de comportement pouvant être exhibé par un groupe de navire : les flottes effectuant des manœuvres de prospection sismique.

En effet ces ensembles de navires effectuent des manœuvres complexes et très caractéristiques comme peut en témoigner la figure suivante.

[10] Laboratoire d'InfoRmatique en Image et Systèmes d'information

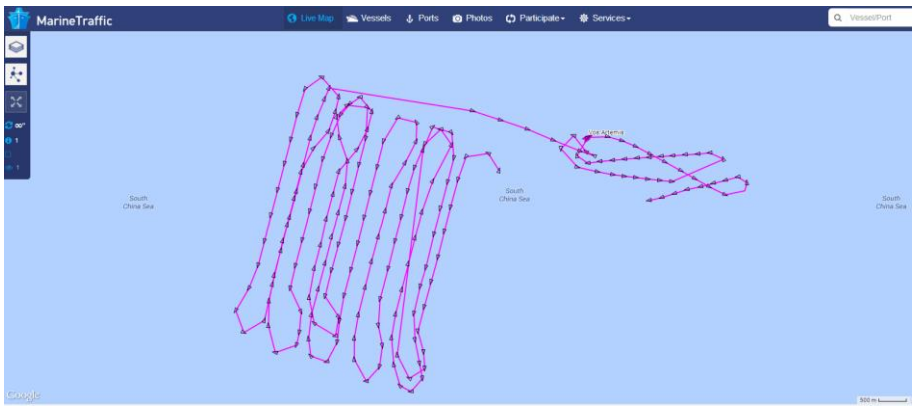


**Figure 6-1 : Tracé réel extrait du site Marine Traffic, datant de Juin 2014, de l'activité du navire de prospection sismique *Vos Artemis* entre train d'effectuer des *lignes de tir*.**

Ainsi, au cours d'une campagne de prospection sismique, une flotte de navire est déployée avec des rôles prédéterminés. Par exemple, un navire sera en charge d'effectuer des allers retours espacés d'une distance prédéterminée pour pouvoir couvrir toute la zone de prospection prédéfinie.

Le long de chacune de ces lignes, un navire sera en charge d'effectuer des *tirs*, à savoir envoyer des impulsions d'air comprimé régulières. Selon la configuration des équipements ce même navire ou un second sera en charge de l'*écoute* et de récupérer les données.

Pour pouvoir effectuer ces manœuvres, les navires subissent des contraintes fortes. Par exemple, leurs cycles d'accélération/décélération sont caractéristiques. De même, leurs trajectoires exhibent des constantes géométriques dont nous pouvons nous servir pour décrire les règles et les différencier des autres navires.

Pouvoir déterminer automatiquement ou semi automatiquement quels navires sont en train d'exhiber ce type de comportement sur l'ensemble des océans de la planète est d'un intérêt économique et industriel majeur pour certaines sociétés.

### 6.2.2 **Enrichissement du modèle**

Comme nous l'avons vu, nous avons rajouté certains opérateurs qui pourraient être qualifiés de *sucre syntaxique* car ne modifiant en rien l'expressivité de notre modèle. Ces derniers sont la répétition et la négation.

Cependant ces derniers se sont révélés essentiels dans l'écriture des règles tant le gain en clarté des règles ainsi qu'en réduction de leur nombre est important.

Il nous semble donc tout à fait pertinent de poursuivre dans la proposition de nouveaux opérateurs permettant de condenser l'écriture de motifs répétitifs au sein d'une grammaire.

Par exemple, nous pourrions envisager de chercher à abstraire l'ensemble des opérateurs temporels proposés dans l'algèbre des intervalles (Allen, 1983). Actuellement, d'un point de vue ensemble d'observations consommées, avec $X$ et $Y$ deux symboles non-terminaux, $X \triangleleft Y$ correspondrait à $X\ meets\ Y$ et $X\ \&\ Y$ correspondrait à $X = Y$. Il serait intéressant de montrer que l'on peut aisément représenter les 11 autres.

### 6.2.3 **Comportements entrelacés**

Parmi les hypothèses de travail que nous nous sommes fixés, nous sommes partis du principe que déplier complètement la bibliothèque de comportements en mémoire entrainerait dans notre cadre applicatif un coût en espace mémoire prohibitif. Cependant, dans des cadres applicatifs plus restreints tels que ceux présentés dans (Popa, Rothkrantz, Wiggers, & Shan, 2013) dans lesquels nous sommes cités, cela peut être envisagé. Ainsi, les problématiques liées à la détection de comportements entrelacés sont plus facilement abordables.

Un travail de recherche lié à la possibilité de reconnaître des comportements entrelacés dans le cas où l'ensemble des comportements ne peut être déployé en mémoire serait une piste de recherche difficile, mais très pertinente.

### 6.2.4 **Optimisation des performances**

Décrire un comportement dans notre modèle de grammaire pose un grand nombre de défis dont certains peuvent être directement liés à ceux rencontrés dans l'écriture de programmes informatiques à l'aide de langages de programmation généraux et déclaratifs tels que ParLog (Gregory, 1987).

Certaines bonnes pratiques pour obtenir une reconnaissance efficace ont été mentionnées au cours de ce manuscrit, cependant il serait pertinent de s'inspirer des optimisations réalisées au sein des compilateurs de ces langages généraux.

# Chapitre 7

# Bibliographie


Abney, S. (1997). Stochastic attribute-value grammars. *Computational Linguistics, 23*(4), 597-618.

Adams, S., & Goel, A. (2007). A STAB at Making Sense of VAST Data., (pp. 1-8).

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Journal of the ACM 42*, 832-843.

Avrahami-Zilberbrand, D. (2009). *Efficient Hybrid Algorithms for Plan Recognition and Detection of Suspicious and Anomalous Behavior.* {B}ar {I}lan {U}niversity.

Avrahami-Zilberbrand, D., & Kaminka, G. (2005). Fast and Complete Symbolic Plan Recognition., (pp. 12-20).

Avrahami-Zilberbrand, D., & Kaminka, G. (2006). Hybrid Symbolic-Probabilistic Plan Recognizer: Initial steps.

Avrahami-Zilberbrand, D., & Kaminka, G. (2007). Incorporating observer biases in keyhole plan recognition (efficiently!). In A. Press (Ed.)., (pp. 944-949).

Avrahami-Zilberbrand, D., Kaminka, G., & Zarosim, H. (2005). Fast and complete plan recognition: Allowing for duration, interleaved execution, and lossy observations.

Blaylock, N., & Allen, J. (2006). Fast hierarchical goal schema recognition. *Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999*, *21*, p. 796.

Bobic, A., & Ivanov, Y. (1998). *Application of Stochastic Grammars to Understanding Action.*

Botella, B., & Taillibert, P. (1993). Interlog: constraint logic programming on numeric intervals.

Bouchard, B., Giroux, S., & Bouzouane, A. (2006). A Smart Home Agent for Plan Recognition of Cognitively-impaired Patients., *1*, pp. 53-62.

Bratman, M. (1987). Intentions, Plans, and Practical Reason.

Bui, H. (2003). A general model for online probabilistic plan recognition. *LAWRENCE ERLBAUM ASSOCIATES LTD, 18*, pp. 1309-1318.

Bui, H., Svetha, V., & West, G. (2002). Policy recognition in the Abstract Hidden Markov Models. *17*(451-499).

Camilleri, G. (1999). A generic formal plan recognition theory., *99*, pp. 540-547.

Carter, N., Young, D., & Ferryman, J. (2006). A Combined Bayesian Markovian Approach for Behaviour Recognition., (pp. 761-764).

Carver, N., Lesser, V., & McCue, D. (1984). Focusing in plan recognition., (pp. 42-48).

Castelfranchi, C., & Rino, F. (1995). From Single-Agent to Multi-Agent: Challenges for Plan Recognition Systems., (pp. 24-32).

Chomsky, N. (1965). *Aspects of the Theory of Syntax.* (T. M. Press, Ed.) MIT press.

Correa, N. (1991). An extension of earley's algorithm for S-attributed grammars. (pp. 299-302). Association for Computational Linguistics.

Culpepper, J. (2007). *Comparing a Hidden Markov Model and a Stochastic Context Free Grammar.* Department of Computer Science and Software Engineering, University of Melbourne.

Davies, J. (2006). *Combining Hidden Markov Models and Stochastic-Context Free Grammars.* Worcester College.

Dousson, C., & Ghallab, M. (1994). Suivi et reconnaissance de chroniques. *Revue d'intelligence artificielle*, *8*(1), 29-61.

Duong, T., Bui, H., Phung, D., & Venkatesh, S. (2005). Activity recognition and abnormality detection with the switching hidden semi-markov model., *1*, pp. 838-845.

Erol, K., Hendler, J., & Nau, D. (1994). Semantics for hierarchical task-network planning. *Computer Science Technical Report Series, CS-TR-3239*, 28.

Galstyan, A., Mitra, S., & Cohen, P. (2006). Detecting and Tracking Hostile Plans in the Hats World.

Geib, C. (2007). Using Lexicalized Grammars and Headedness for Approximate Plan Recognition.

Geib, C., & Goldman, R. (2009). A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence, 173*(11), 1101-1132.

Gerber, R., Nagel, H., & Schreiber, H. (2002). Deriving textual descriptions of road traffic queues from video sequences. *Citeseer*, (pp. 736-740).

Ghahramani, Z. (1998). Learning dynamic Bayesian networks. *Lecture Notes in Computer Science, 1387*, 168-197.

Ghanem, N., DeMenthon, D., Doermann, D., & Davis, L. (2004). Representation and recognition of events in surveillance video using petri nets. *Citeseer*, *7*.

Granvilliers, L. (2001). On the combination of interval constraint solvers. *Reliable Computing, 7*(6), 467-483.

Gregory, S. (1987). *Parallel Logic Programming in Parlog: The Language and Its Implementation.* Boston: Addison-Wesley Longman Publishing Co., Inc.

Gu, T., Wu, Z., Wang, L., Tao, X., & Lu, J. (2009). Mining Emerging Patterns for recognizing activities of multiple users in pervasive computing., (pp. 1-10).

Hamid, R., Maddi, S., Bobick, A., & Essa, I. (2007). Structure from statistics: Unsupervised activity analysis using suffix trees. *Citeseer*, *2007*, p. 1.

Hamid, R., Maddi, S., Johnson, A., Bobick, A., Essa, I., & Isbell, C. (2009). A novel sequence representation for unsupervised analysis of human activities. *Artificial Intelligence, 173*(14), 1221-1244.

Harati-Mokhtari, A., Wall, A., Brooks, P., & Wang, J. (2007). Automatic Identification System (AIS): data reliability and human error implications. *The Journal of Navigation, 60*(03), 373-389.

Jagota, A., Lyngs, R., & Pedersen, C. (2001). Comparing an HMM and an SCFG., (pp. 69-84).

Joo, S., & Chellappa, R. (2006). Recognition of multi-object events using attribute grammars., (pp. 2897-2900).

Kaminka, G., Pynadath, D., & Tambe, M. (2002). Monitoring teams by overhearing: A multi-agent plan-recognition approach. *Journal of Artificial Intelligence Research, 17*(1), 83-135.

Kautz, H. (1987). *A formal theory of plan recognition.* University of Rochester.

Knuth, D. (1968). Semantics of context-free languages. *Theory of Computing Systems, 2*(2), 127-145.

Lesire, C., & Tessier, C. (2006). Estimation and conflict detection in human controlled systems. *3927*, p. 407. Springer.

Mao, W., & Gratch, J. (2004). Decision-theoretic approach to plan recognition. Citeseer.

McConnell, S. (2004). *{Code complete}*. Microsoft Press Redmond, WA, USA.

Medioni, G., Cohen, I., Br{\'e}mond, F., Hongeng, S., & Nevatia, R. (2001). Event detection and analysis from video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 873-889.

Mihajlovic, V., & Petkovic, M. (2001). Dynamic bayesian networks: A state of the art. Centre for Telematics and Information Technology, University of Twente.

Murphy, K. (2002). Dynamic bayesian networks.

Nagel, H. (2004). Steps toward a cognitive vision system. *AI Magazine, 25*(2), 31.

Nevatia, R., Zhao, T., & Hongeng, S. (2003). Hierarchical language-based representation of events in video streams. *Citeseer*.

Nguyen, N., Phung, D., Venkatesh, S., & Bui, H. (2005). Learning and detecting activities from movement trajectories using the hierarchical hidden Markov model., *2*, pp. 955-960.

Nogueira, J., Furtado, A., & Alcazar, J. (1996). A Hybrid Formal Theory of Plan Recognition and Its Implementation. *Lecture Notes in Computer Science*, 31-40.

Older, W., & Vellino, A. (1993). Constraint arithmetic on real intervals. (pp. 175-195). Citeseer.

Popa, M., Rothkrantz, L., Wiggers, P., & Shan, C. (2013). Shopping Behavior Recognition using a Language. *Pattern Recognition Letters*, 1879-1889.

Pynadath, D., & Wellman, M. (2000). Probabilistic state-dependent grammars for plan recognition. *Citeseer*, (pp. 507-514).

Rabiner, L., & Juang, B. (1986). An introduction to hidden Markov models. *ieee assp magazine, 3*(1 Part 1), 4-16.

Rota, N. (2001). *Contribution {\`a} la reconnaissance de comportements humains {\`a} partir de s{\'e}quences vid{\'e}os.* Th{\`e}se, INRIA-Universit{\'e} de Nice Sophia Antipolis, 2001/10.

Rota, N., & Thonnat, M. (2000). Activity Recognition from Video Sequences using Declarative Models., (pp. 673-680).

Saykol, E., Güdükbay, U., & Ulusoy, O. (2009). Scenario-based query processing for video-surveillance archives. *Engineering Applications of Artificial Intelligence*.

Schmidt, C., Sridharan, N., & Goodson, J. (1978). The plan recognition problem. *Artificial Intelligence, 11*, 45-83.

Sukthankar, G., & Sycara, K. (2005). A cost minimization approach to human behavior recognition. *ACM New York, NY, USA*, (pp. 1067-1074).

Sukthankar, G., & Sycara, K. (2005). Automatic recognition of human team behaviors.

Suzic, R., & Svenson, P. (2006). Capabilities-based plan recognition. *Dept. of Syst. Modelling, Swedish Defence Res. Agency, Stockholm;*.

Thompson, K. (1968). Programming Techniques: Regular expression search algorithm. *6*(419–422).

Turaga, P., Chellappa, R., Subrahmanian, V., & Udrea, O. (2008). Machine recognition of human activities: A survey. *IEEE Transactions on Circuits and Systems for Video Technology, 18*(11), 1473-1488.

Vanegas, C., Aliaga, D., Wonka, P., Müller, P., Waddell, P., & Watson, B. (2010). Modelling the Appearance and Behaviour of Urban Spaces. *John Wiley \& Sons, 9999*.

Vidal, N., Taillibert, P., & Aknine, S. (2010). Online Behavior Recognition: A New Grammar Model Linking Measurements and Intents. *ICTAI* (pp. 129-137). Aras: IEEE Computer Society.

Wolf, L., Jhuang, H., & Hazan, T. (2007). Modeling appearances with low-rank SVM. *Citeseer*.

# Annexes

# Code Mathematica des principaux algorithmes utilisés dans le système BOBAC

# Cleaning NoteBook

```
Remove["Global`*"];Remove["testContext`*"];
```

# Rule construction and library

## Rule Library

```
ClearAll[RuleLibrary];
RuleLibrary={};
 ClearAll[rulePattern];
wordPattern = WordCharacter..;
associationPattern = wordPattern~~"="~~wordPattern;
oneOrMore[pattern_]:=(pattern~~",")...~~pattern;
zeroOrMore[pattern_]:=oneOrMore[pattern]|"";
subSymbolAssociationPattern=(wordPattern)~~"("~~(zeroOrMore[associationPattern])~~"
)";
subSymbolAssociationPatternNamed=(symbolName:wordPattern)~~"("~~(associations:zeroO
rMore[associationPattern])~~")";
rulePattern=

(StartOfString~~(lhs:wordPattern)~~"("~~(variables:zeroOrMore[wordPattern])~~")"~~"
=>"~~(subrules:zeroOrMore[subSymbolAssociationPattern])~~".|"~~(constraints:((WordC
haracter..~~("="|"<"|">")~~WordCharacter..~~",")...~~(WordCharacter..~~("="|"<"|">"
)~~WordCharacter..))))~~EndOfString;
 ruleRewriteRule = (rulePattern  :>
(lhs~~"[{"~~variables~~"},{"~~StringReplace[subrules,subSymbolAssociationPatternNam
```

```
ed:→"{"~~symbolName~~",{"~~associations~~"}}"]~~"},{"~~constraints~~"}]"));
```

# Rule Creation

```
 ClearAll[createRule];
createRule[Name_, Args_, LHS_, RHS_, Constraints_, ruleID_] :=
  (
  name[ruleID] = Name;
  args[ruleID] = Args;
  lhs[ruleID] = LHS;
  rhs[ruleID]=RHS;
  constraints[ruleID] = Constraints;
  AppendTo[RuleLibrary, ruleID];
    )
 createRule[string_/;StringQ]:=
 expr = ToExpression[StringReplace[string, ruleRewriteRule]]
 createPasteRule[] :=
 (
  createRule[Input[]];
  )
```

# RuleForm

```
 ClearAll[leavesValidated]; leavesValidated[_]={};
ClearAll[leavesToBeValidated];leavesToBeValidated[_]={};
ClearAll[constraints];constraints[_]={};
ClearAll[leftToExpand];leftToExpand[_]={};
ClearAll[attributes];attributes[_]={};
ClearAll[currentnode];currentnode[_]=-1;
ClearAll[validated];validated[_]=False;
ClearAll[expanded];expanded[_]=False;
```

```
ClearAll[parent];
```

# ObservationExtrapolation

```
 ClearAll[any];
any = obs[_,_,_,_];
 ClearAll[observations];
observations={};
 ClearAll[addObservation];
addObservation[SENSORID_,ENTITYID_,PROPERTYID_,TYPEID_,VALUE_,REALTIME_] :=
  (
  Module[{OBSERVATIONID},
    sensor[OBSERVATIONID] ^= SENSORID;
    entity[OBSERVATIONID] ^= ENTITYID;
    type[OBSERVATIONID]^=TYPEID;
    value[OBSERVATIONID] ^= VALUE;
    realtime[OBSERVATIONID] ^= REALTIME;
    property[OBSERVATIONID]^=PROPERTYID;
    observations=AppendTo[observations, OBSERVATIONID];
    ];
    )
 ClearAll[dateToSeconds];
dateToSeconds[date_]:=
 DateDifference[{0,0,0,0,0,0},date,"Second"][[1]]
 ClearAll[buildSnapshots];
observationInterpolation[_,_,_]={};
buildSnapshots[TIMEINTERVAL_:{0,0,0,0,1,0}]:=
 (
  ClearAll[snapshots];
```

```
  snapshots={};
  currentSnapshot=1;
  With[
   {
    minTime=Min[dateToSeconds[realtime[#]]&/@observations],
    maxTime=Max[dateToSeconds[realtime[#]]&/@observations],
    intervalInSeconds=dateToSeconds[TIMEINTERVAL],
    monitoredProperties=DeleteDuplicates[property[#]&/@observations],
    monitoredEntites=DeleteDuplicates[entity[#]&/@observations],
    monitoredSensors=DeleteDuplicates[sensor[#]&/@observations]
    },

With[{numIntervals=Floor[DateDifference[minTime,maxTime,"Second"][[1]]/dateToSecond
s[TIMEINTERVAL]]},
     Do[AppendTo[observationInterpolation[tmpsensor, tmpentity, tmpProperty],
          {dateToSeconds[realtime[#]],value[#]}];&/@(Select[observations,
(tmpsensor==sensor[#]&&tmpentity==entity[#]&&tmpProperty==property[#])&]),
      {tmpProperty,monitoredProperties},
      {tmpentity,monitoredEntites},
      {tmpsensor,monitoredSensors}
      ];
     Do[
      If[observationInterpolation[tmpsensor, tmpentity, tmpProperty]==={},
       Null,
       With[{time=dateToSeconds[TIMEINTERVAL]*(i-1)+minTime},
         value[i,tmpentity,tmpsensor,tmpProperty]=
           (
            Switch[observationInterpolation[tmpsensor, tmpentity,
tmpProperty][[1]][[2]],
```

```
            n_/;IntegerQ[n]||NumberQ[n],

With[{tmpInterp=Interpolation[observationInterpolation[tmpsensor,tmpentity,tmpPrope
rty],time]},
             tmpInterp
             ],
            interval_/;Head[interval]==Interval,

With[{tmpInterp=Interpolation[observationInterpolation[tmpsensor,tmpentity,tmpPrope
rty],time]},
             interval[tmpInterp[[1]][[1]],tmpInterp[[1]][[2]]]
             ],
            s_/;StringQ[s],

With[{orderedObs=SortBy[observationInterpolation[tmpsensor,tmpentity,tmpProperty],(
#[[1]])&]},
             Module[{cnt},
              cnt=1;
              While[cnt≤Length[orderedObs]&&time≥orderedObs[[cnt]][[1]],cnt++];
              Switch[cnt,
               1,{orderedObs[[1]][[2]]},
               Length[orderedObs]+1,{orderedObs[[Length [orderedObs]]][[2]]},
               _,DeleteDuplicates[{orderedObs[[cnt-
1]][[2]],orderedObs[[cnt]][[2]]}]
               ]
              ]
             ]]);
         ];
       ],
```

```
      {i,1,numIntervals},
      {tmpProperty,monitoredProperties},
      {tmpentity,monitoredEntites},
      {tmpsensor,monitoredSensors}
      ];
     ];];
 )
```

# Algorithm

## Initialisation

```
initialisation:=
(
 ClearAll[hypotheses];
 hypotheses={};
 (*SetSharedVariable[hypotheses];*)
 )
```

## Fill the first rule

```
createFirstRule:=
 Module[{tree},
  name[tree[start]]="start";
  hypotheses=Append[hypotheses,tree[start]];
  ];
```

## Clean and nodes

```
 ClearAll[cleanNodes];
cleanNodes:=
  Cases[hypotheses,
   tree_[symbol_]:→
    (
     Cases[UpValues[tree],
       ((Verbatim[HoldPattern][andChilds[_[sym_]]]:→ _)/;
          (leavesToBeValidated[tree[sym]]=!={}&&leavesValidated[tree[sym]]=!={}):→
         (
          hypotheses=DeleteCases[hypotheses,tree[_]];
          Remove[tree];
          )),1
       ];
     )
   ];
```

# Copy a node

```
ClearAll[copyNode];
copyNode[node_ /;MatchQ[node,oldTree_[symbol_]]]:=
Replace[node,oldTree_[symbol_]:→ Module[{tree},
   UpValues[tree]=(UpValues[oldTree]/.oldTree→ tree);
   hypotheses=Append[hypotheses,tree[start]];
   tree[symbol]]]
```

# Develop a node

## If the node hasn't been expanded, let's find expand rules in the library

```
develop[oldTree_[symbol_] /;!expanded[oldTree[symbol]]]:=
```

```
 (
  If[debug,Print["1 : "~~ToString[oldTree[symbol]]]];
  If[Length[leftToExpand[oldTree]]≥1&&leftToExpand[oldTree][[1]]===oldTree[symbol],
   leftToExpand[oldTree]^=Take[leftToExpand[oldTree],-
Length[leftToExpand[oldTree]]+1];
   ];
  With[{sym=StringSplit[SymbolName[symbol],"$"][[1]],node=oldTree[symbol]},
   expanded[node]^=True;
   With[{matchingRules=Select[RuleLibrary,
(StringSplit[SymbolName[#],"$"][[1]]==sym)&]},
    If[Length[matchingRules]==0,
      If[ValueQ[entityP[node]],
       Cases[DownValues[value],(Verbatim[HoldPattern][value[currentSnapshot,
entityM_,sensorM_,propertyM_]]:→
valueM_)/;MatchQ[entityM,entityP[node]]&&MatchQ[sensorM,sensorP[node]]&&MatchQ[prop
ertyM,propertyP[node]]&&MatchQ[valueM,valueP[node]]:→
         (
          With[{newnode=copyNode[node]},
            With[{newTree=(newnode/.h_[t_]:→ h)},
             Evaluate[entity[newnode]]^=entityM;
             Evaluate[property[newnode]]^=propertyM;
             Evaluate[sensor[newnode]]^=sensorM;
             Evaluate[value[newnode]]^=valueM/.
               {(interval[min_,max_]:→
                  (
                   Module[{valueInInterval},
                    constraints[newTree]^=Join[constraints[newTree],
                    {valueInInterval[newnode]≤ max,valueInInterval[newnode]≥ min}];
                    valueInInterval[newnode]
```

```
                    ]
                   )),
                (l_List:→
                  (
                   Module[{valueInInterval},
                    (constraints[newTree]^=(Join[constraints[newTree],
                    { l//.
                    {l2_List/;Length[l2]≥2:→
(valueInInterval[newnode]==l2[[1]]||Take[l2,-Length[l2]+1]),
                     l2_List/;Length[l2]==1:→ valueInInterval[newnode]== l2[[1]]}}]));
                     valueInInterval[newnode]
                     ]
                    )};
              With[{newconstraints=Reduce[constraints[newTree]]},
               If[AtomQ[newconstraints] && newconstraints==False,
                (
                 hypotheses=DeleteCases[hypotheses,newTree[_]];
                 Remove[newTree];
                 ),
                constraints[newTree]^={newconstraints};
                develop[newnode];
                ]]]];
           ),1];
        hypotheses=DeleteCases[hypotheses, oldTree[_]];
        Remove[oldTree];
        ,
        If[ValueQ[andChilds[node]],
         leavesToBeValidated[node]^=andChilds[node];
         Do[
```

```
          leftToExpand[oldTree]^=Append[leftToExpand[oldTree],andNode];
          parent[andNode]^=node;
          ,
          {andNode,andChilds[node]}];
         develop[leftToExpand[oldTree][[1]]];,
         hypotheses=DeleteCases[hypotheses, oldTree[_]];
         Remove[oldTree];
         ]],
        Map[
        With[{newnode=copyNode[node]},
          With[{newTree=(newnode/.h_[t_]:> h)},
            constraints[newTree]^=Join[constraints[newTree],

constraints[#]/.sym1_Symbol?((Context[#]==="Global`")&):>sym1[newnode]];
            With[{cstrs=Reduce[constraints[newTree]]},
             If[AtomQ[cstrs] && cstrs==False,
               (
                hypotheses=DeleteCases[hypotheses,newTree[_]];
                Remove[newTree];
                ),
              leavesToBeValidated[newnode]^=((
                   newTree[
                    If[Head[#]===obs,
                    Module

[{observation},#/.{obs[entityPattern_,sensorPattern_,propertyPattern_,valuePattern_
]:>
                     (
                     entityP[newTree[observation]]^=entityPattern;
```

```
                    sensorP[newTree[observation]]^=sensorPattern;
                    propertyP[newTree[observation]]^=propertyPattern;
                    valueP[newTree[observation]]^=valuePattern;
                    observation
                    ),

obs[links_,entityPattern_,sensorPattern_,propertyPattern_,valuePattern_]:→
                    (
                    entityP[newTree[observation]]^=entityPattern;
                    sensorP[newTree[observation]]^=sensorPattern;
                    propertyP[newTree[observation]]^=propertyPattern;
                    valueP[newTree[observation]]^=valuePattern;
                    Cases[links,
                    {childArg_,nodeArg_}:→

(With[{eval=Evaluate[childArg[newTree[observation]]],eval2=Evaluate[nodeArg[newnode
]]},

With[{newconstraints=Reduce[Append[constraints[newTree],eval==eval2]]},
                    If[AtomQ[newconstraints] && newconstraints==False,
                    (
                    hypotheses=DeleteCases[hypotheses,newTree[_]];
                    Remove[newTree];
                    ),
                    constraints[newTree]^={newconstraints};]]])
                    ];
                    observation
                    )
                    }],
```

```
                    If[Head[#]===and,
                    Module[{and},
                    andChilds[newTree[and]]^=Map[
                    (
                    If[Head[#]===obs,
                    Module

[{observation},#/.{obs[entityPattern_,sensorPattern_,propertyPattern_,valuePattern_
]:→
                    (
                    entityP[newTree[observation]]^=entityPattern;
                    sensorP[newTree[observation]]^=sensorPattern;
                    propertyP[newTree[observation]]^=propertyPattern;
                    valueP[newTree[observation]]^=valuePattern;
                    newTree[observation]
                    ),

obs[links_,entityPattern_,sensorPattern_,propertyPattern_,valuePattern_]:→
                    (
                    entityP[newTree[observation]]^=entityPattern;
                    sensorP[newTree[observation]]^=sensorPattern;
                    propertyP[newTree[observation]]^=propertyPattern;
                    valueP[newTree[observation]]^=valuePattern;
                    Cases[links,
                    {childArg_,nodeArg_}:→

(With[{eval=Evaluate[childArg[newTree[observation]]],eval2=Evaluate[nodeArg[newnode
]]},
```

```
With[{newconstraints=Reduce[Append[constraints[newTree],eval==eval2]]},
                    If[AtomQ[newconstraints] && newconstraints==False,
                    (
                    hypotheses=DeleteCases[hypotheses,newTree[_]];
                    Remove[newTree];
                    ),
                    constraints[newTree]^={newconstraints};]]])
                    ];
                    newTree[observation]
                    )
                    }],
                    If[Depth[#]==1,
                    newTree[Unique[#]],
                    With[{ch=Unique[Head[#]]},
                    Cases[#[[1]],
                    {childArg_,nodeArg_}:→

(With[{eval=Evaluate[childArg[newTree[ch]]],eval2=Evaluate[nodeArg[newnode]]},

With[{newconstraints=Reduce[Append[constraints[newTree],eval==eval2]]},
                    If[AtomQ[newconstraints] && newconstraints==False,
                    (
                    hypotheses=DeleteCases[hypotheses,newTree[_]];
                    Remove[newTree];
                    ),
                    constraints[newTree]^={newconstraints};]]])
                    ];
                    newTree[ch]]]]
                    )&,
```

```
                    #[[1]]
                    ];
                    and
                    ],
                    (*Unique[If[Depth[#]==1,#,Head[#]]],*)
                    (
                    If[Depth[#]==1,
                    Unique[#],
                    With[{ch=Unique[Head[#]]},
                    Cases[#[[1]],
                    {childArg_,nodeArg_}:→

(With[{eval=Evaluate[childArg[newTree[ch]]],eval2=Evaluate[nodeArg[newnode]]},

With[{newconstraints=Reduce[Append[constraints[newTree],eval==eval2]]},
                    If[AtomQ[newconstraints] && newconstraints==False,
                    (
                    hypotheses=DeleteCases[hypotheses,newTree[_]];
                    Remove[newTree];
                    ),
                    constraints[newTree]^={newconstraints};]]])
                    ];
                    ch]]
                    )]]]
                   )&/@rhs[#]);
              With[{newconstraints=Reduce[constraints[newTree]]},
               If[AtomQ[newconstraints] && newconstraints==False,
                (
                 hypotheses=DeleteCases[hypotheses,newTree[_]];
```

```
                Remove[newTree];
                ),
               constraints[newTree]^={newconstraints};
               (parent[#]^=newnode)&/@leavesToBeValidated[newnode];
               develop[newnode];
               ]];
             ];];];
        ]&,
       matchingRules];
      hypotheses=DeleteCases[hypotheses, oldTree[_]];
      Remove[oldTree];];
    ];
   ]
  )
```

If the node has been expanded, and some child are left to be validated

```
 develop[node_/;expanded[node]&& Count[leavesToBeValidated[node],_]>0]:=
 (
  With[{oldTree=Head[node]},
   If[Length[leftToExpand[oldTree]]≥1&&leftToExpand[oldTree][[1]]===node,
    leftToExpand[oldTree]^=Take[leftToExpand[oldTree],-
Length[leftToExpand[oldTree]]+1];
    ]];
  If[debug,Print["2 : "~~ToString[node]]];
  If[ValueQ[andChilds[node]],
   Do[
    leftToExpand[Head[node]]^=Append[leftToExpand[Head[node]],andNode];
    ,
    {andNode,andChilds[node]}];
   develop[leftToExpand[Head[node]][[1]]],
```

```
   develop[leavesToBeValidated[node][[1]]]
   ];
  )
```

If the node has been expanded and no child is left to be validated

```
develop[oldtree_[symbol_]/;expanded[oldtree[symbol]]&&Count[leavesToBeValidated[old
tree[symbol]],_]==0 && ValueQ[entityP[oldtree[symbol]]]]:=
 (
  If[Length[leftToExpand[oldtree]]≥1&&leftToExpand[oldtree][[1]]===oldtree[symbol],
   leftToExpand[oldtree]^=Take[leftToExpand[oldtree],-
Length[leftToExpand[oldtree]]+1];
   ];
  With[{node=oldtree[symbol]},
   If[debug,Print["3 : "~~ToString[node]]];
   validated[node]^=True;
   If[ValueQ[parent[node]],
childValidated[parent[node]],node,childValidated[parent[node]]]
   ]
  )

develop[oldtree_[symbol_]/;expanded[oldtree[symbol]]&&Count[leavesToBeValidated[old
tree[symbol]],_]==0 && !ValueQ[entityP[oldtree[symbol]]]]:=
 (
  If[debug,Print["3REMOVED : "~~ToString[oldtree[symbol]]]];
  hypotheses=DeleteCases[hypotheses, oldtree[_]];
  Remove[oldtree];
  )
```

# Validate a child

```
 childValidated[node_]:=
 (
  If[debug,Print["4 : "~~ToString[node]]];

leavesValidated[node]^=Append[leavesValidated[node],leavesToBeValidated[node][[1]]]
;
  leavesToBeValidated[node]^=Delete[leavesToBeValidated[node],1];
  If[Count[leavesToBeValidated[node],_]==0,
   (
    If[debug,Print["Completely Validated : "~~node]];
    validated[node]^=True;
    childValidated[parent[node]]),
   If[Length[leftToExpand[Head[node]]]≥1,
     develop[leftToExpand[Head[node]][[1]]];];
   ];
  )
```

# Print node

```
 getLinks[node_]:=
 (With[{val=If[validated[node],node^ok,node]},
   Flatten[Join[Cases[leavesValidated[node],child_:→ (val→child^ok)],
     Cases[leavesToBeValidated[node],child2_:→ (val→child2)],
     If[Count[leavesValidated[node],_]==0,{},Cases[leavesValidated[node],child3_:→
getLinks[child3] ]],

If[Count[leavesToBeValidated[node],_]==0,{},Cases[leavesToBeValidated[node],child4_
```

```
:→ getLinks[child4] ]]]]]
  )

printNode[node_]:=TreePlot[getLinks[node],Top,node,VertexLabeling→All,VertexRenderi
ngFunction→({White,EdgeForm[White],Disk[#,0.3],Black,Text[StringReplace[ToString[Re
place[#2, {_[symbol_]:→  symbol, _[symbol_]^ok:→  symbol^ok}]],name__~~"$"~~num__:→
name],#1]}&),DirectedEdges→True,PackingMethod→"Layered"]
```

# Follow Hypotheses Evolution

```
Dynamic[Map[printNode,hypotheses]]
```

hypotheses

# Tests

## Dummy Tests

### Library

```
ClearAll[RuleLibrary];
RuleLibrary={};

createRule[start,{},start,{zero[{{num,length}}],first,second,third},{},Unique[start
]];
createRule[start,{},start,{and[{f1,f2}],f3,f4},{},Unique[start]];
createRule[zero,{},zero,{f1,f2,f3,f4},{},Unique[zero]];
createRule[f1,{},f1,{and[{any,any}],any},{},Unique[f1]];
 (*createRule[f2,{},f2,{any,and[{any,any}]},{},Unique[f2]];*)
```

```
 (*createRule[f3,{},f3,{any,and[{any,any}]},{},Unique[f3]];*)
 (*createRule[start,{},start,{obs[{{value,
valeur}},_,_,_,_],obs[obs[{{value,valeur}}],_,_,_,_]},{},Unique[start]];*)
```

## Observations

```
 ClearAll[observations];
observations={};
 addObservation["globalSensor","kalee","speed","interval",42,{2013,10,22,18,41,0}]
 addObservation["globalSensor","kalee","speed","interval",55,{2013,10,22,18,42,0}]
 addObservation["globalSensor","kalee","speed","interval",48,{2013,10,22,18,50,0}]
 addObservation["globalSensor","kalee","speed","interval",44,{2013,10,22,18,55,0}]
 addObservation["globalSensor","kalee2","speed","interval",10,{2013,10,22,18,41,0}]

addObservation["globalSensor","kalee2","speed","interval",200,{2013,10,22,18,42,0}]

addObservation["globalSensor","kalee2","speed","interval",199,{2013,10,22,18,50,0}]

addObservation["globalSensor","kalee2","speed","interval",230,{2013,10,22,18,55,0}]
 buildSnapshots[]
```

## Initialization

```
 initialisation;
createFirstRule;
```

## Iteration

```
 hypotheses//Length
 1
 debug=False;
(Map[develop, hypotheses];cleanNodes;currentSnapshot++)//Timing
 {0.046800,1}
```