INSTITUT **S**UPERIEUR
D'**I**NFORMATIQUE
DE **M**ODELISATION
ET DE LEURS **A**PPLICATIONS.

Complexe des cézeaux
BP 125 – 63173 Aubière CEDEX

Pacte Novation

2 rue du Docteur Lombard
92441 Issy-les-Moulineaux Cedex
Tél. : +33 (0) 1 01 45 29 06 06
Fax : 01 45 29 25 00

Rapport de stage de fin d'étude

FASTPLAN 2

Utilisation de la programmation linéaire en nombre entier dans le domaine de la planification

Présenté par : AUGUSTO Stéphane.
Lieu de projet : Pacte Novation.
Responsable de Projet : M P.Morignot.

Date : avril – août 2001

INSTITUT **S**UPERIEUR
D'**I**NFORMATIQUE
DE **M**ODELISATION
ET DE LEURS **A**PPLICATIONS.

Complexe des cézeaux
BP 125 – 63173 Aubière CEDEX

Pacte Novation

2 rue du Docteur Lombard
92441 Issy-les-Moulineaux Cedex
Tél. : +33 (0) 1 01 45 29 06 06
Fax : 01 45 29 25 00

Rapport de stage de fin d'étude

FASTPLAN 2

Utilisation de la programmation linéaire en nombre entier dans le domaine de la planification

Présenté par : AUGUSTO Stéphane.
Lieu de projet : Pacte Novation.
Responsable de Projet : M P.Morignot.

Date : avril – août 2001

Remerciements

Je tiens tout d'abord à remercier l'ensemble de Pacte Novation pour l'ambiance chaleureuse et leur aide tout au long de mon stage. Je remercie également M Philippe MORIGNOT pour m'avoir encadré pendant ce stage.

Abréviation

PLNE : programmation linéaire en nombre entier.
STRIPS : STanford Research Institute Planning System
IA : intelligence artificielle.
RO : Recherche opérationnelle.
PDDL : Planning Domain Definition Language.
UML : Unified Modeling Language.

Table des figures

Résumé

L'utilisation de la programmation linéaire en nombre entier dans le domaine de la planification est très récente. En effet, la planification est un domaine largement traité par l'intelligence artificielle. Cependant la recherche opérationnelle permet d'étendre les problèmes de planification en introduisant des coûts d'actions, des problèmes avec ressource.

Le travail demandé dans le cadre de mon stage de fin d'étude a consisté en l'analyse, la conception et le développement en C++ d'un planificateur utilisant la programmation linéaire en nombre entier avec un parser analysant les fichiers PDDL (Planning Domain Definition Language).

Mot-clefs : planification, programmation linéaire en nombre entier, PDDL

Abstract

Using integer programs in the planning field is quite recent. Indeed, planning is a major field of artificial intelligence. However, operational research allows to extend planning problem in introducing actions costs, resource based problem.

On this thema was based a final year project, which consisted in analysing, conceptualising and develloping a planning program in C++ using integer programming with a parser analysing PDDL fichier (Planning Domain Definition Language).

Keyword : planning, integer programming, PDDL

Table des Matières

Introduction

Au cours de ma dernière année d'étude à l'ISIMA, j'ai réalisé un stage d'une durée de cinq mois, d'avril 2001 à août 2001, au sein de la société Pacte Novation.

Le sujet de mon stage porte sur le domaine de la planification. Un problème de planification se résume en trouver une séquence d'action qui transforme un état initial dans un état final désiré. Depuis quelques années, la recherche opérationnelle avec la programmation linéaire en nombre entier s'est intéressé à ce domaine réservé auparavant à l'intelligence artificielle. La recherche opérationnelle permet notamment d'introduire dans ce type de problème des coûts d'action, des problèmes avec ressource.

Le travail qui a été réalisé porte en premier lieu sur l'étude de l'utilisation de la programmation linéaire en nombre entier pour résoudre des problème de planification et suivi d'une conception et d'un développement en C++ d'un tel planificateur. Par la suite, un parser capable d'analyser les fichiers PDDL (Plan Domain Definition Language) a été implémenté pour permettre de tester le planificateur sur un large éventail de problème.

Dans la première partie de ce document se trouve une présentation de la société, qui m'a accueilli, Pacte Novation. Dans la deuxième partie se trouve ensuite une étude sur le domaine de la planification. La troisième partie décrit l'ensemble du travail de développement. Et enfin la dernière partie décrit les résultats obtenus.

Chapitre1 : Présentation de Pacte Novation

1 .1) Introduction

Pacte Novation est une société d'ingénierie logicielle pluridisciplinaire, intervenant dans des secteurs d'activités très variés comme le transport, la banque-finance, les télécommunications, l'énergie, l'industrie et le tertiaire.

Son savoir-faire, reconnu auprès de très grands comptes, permet d'intervenir sur des applications à haute valeur ajoutée. Prés de 40% de son activité est réalisé au forfait, ce qui lui a permis de fortement capitaliser autour de 3 axes :

Technologies orientés objets et distribuées :

- Conception et réalisation d'applications à base de technologies orientées objets
- Conseil et pratique des méthodes orientées objets
- Intégration d'applications distribuées dans les environnements Client / Serveur

Interfaces Homme Machine avec une spécialisation en ergonomie du logiciel :

- Prise en compte du facteur humain lors de l'informatisation de processus
- Analyse de la tâche et spécifications des Interfaces Homme/Machine
- Evaluation et recommandation ergonomique sur des IHM existantes
- Réalisation d'IHM graphiques et d'outils utilisant le Langage Naturel

Aide à la décision :

Résolution de problèmes intégrant des raisonnements d'experts, une forte complexité ou bien une grande combinatoire

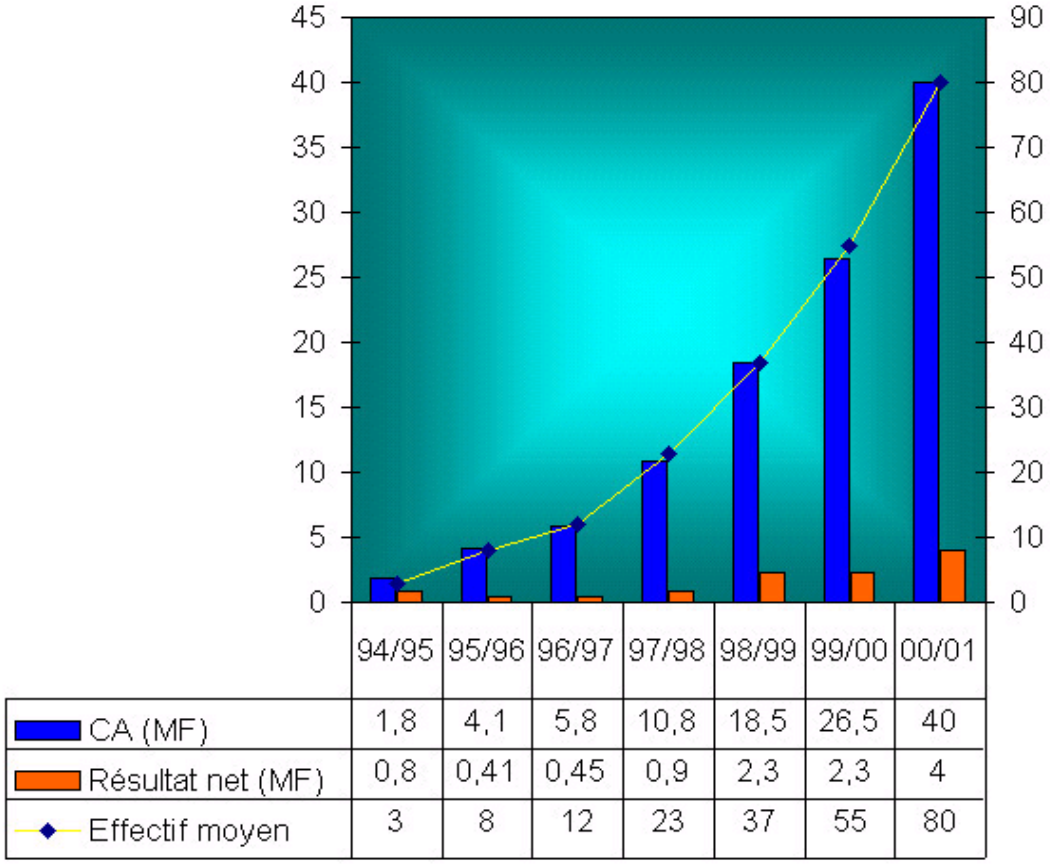
- *Systèmes à Base de Connaissances* : spécification, conception et développement de systèmes à base de connaissances, modélisation et capitalisation de savoir-faire
- *Systèmes à Base de Contraintes* : allocation de ressources, planification et ordonnancement, optimisation de processus, …

1 .2) Profil de l'entreprise

Pacte Novation, créé le 1er avril 1994, est une société anonyme au capital de 512 000 €. Deux associés sont à la tête de Pacte Novation : Christian TORA, Président Directeur Général, ingénieur de

formation et Bruno GAUDINAT, également ingénieur. Les actionnaires principaux sont les cadres et dirigeants de l'entreprise.

A ce jour 83 ingénieurs travaillent dans la société et la figure … montre la croissance de Pacte Novation depuis sa création.



	94/95	95/96	96/97	97/98	98/99	99/00	00/01
CA (MF)	1,8	4,1	5,8	10,8	18,5	26,5	40
Résultat net (MF)	0,8	0,41	0,45	0,9	2,3	2,3	4
Effectif moyen	3	8	12	23	37	55	80

Figure 1- courbe de croissance de Pacte Novation

1.3) Ingénierie chez Pacte Novation

1.3.1) Compétences

Pacte Novation est une société à forte compétence et de grande expérience dans tous les domaines avancés de l'aide à la décision. Au dessus des technologies objet, elle a su réunir et mettre en valeur une équipe soudée, complémentaire, curieuse des métiers de ses clients ainsi que des techniques informatiques à forte valeur ajoutée. Ces compétences s'expriment et trouvent leur place à toutes les étapes d'un projet.

Dans les rangs de Pacte Novation se trouvent

- des cogniticiens et ergonomes chargés d'analyser le métier mais aussi le besoin exact du client en vue de rédiger un cahier des charges précis et exhaustif.

- des spécialistes de la résolution de problèmes complexes et combinatoires ou de la mise en place de systèmes à base de connaissances, problèmes pour lesquels une expérience significative est indispensable pour envisager sereinement l'aboutissement des projets qui y ont trait.

- des architectes capables de concevoir et mettre en place des architectures distribuées ainsi que modélisations objets complexes et évolutives.
- enfin, une direction technique composée de consultants et de chefs de projets à forte compétence technique se chargeant de garantir la fourniture d'un service de qualité irréprochable.

1.3.2) Clients et projets

Pacte Novation intervient dans de nombreux domaines aussi divers que

- l'industrie : SOLLAC, SPSE, Société Le Nickel.
- l'énergie : EDF, GDF.
- Le transport : Renault Sport, GEC Alsthom, ADP, Eurocontrol, ARAAM, Transnucléaire.
- L'industrie de l'informatique et des télécommunications : Alcatel, BULL.
- La finance des salles de marché : Crédit Agricole Indosuez, Crédit Lyonnais, Dresdner, Bank, Société Général.

Pacte Novation a participé à des projets ou programmes d'envergure comme :

- ALICE pour Renault Sport :

  Créé spécifiquement pour Renault Sport dans le contexte de la formule 1, ALICE (Application Logicielle Intelligente pour Course et Essais) analyse les données envoyées par télémesure à chaque tour de piste et déduit les alarmes moteur.

- Simulateur de trafic ferroviaire pour ALSTOM :

Pacte Novation a développé pour ALSTOM une série de simulateurs de trafic ferroviaire. Vendus à l'exportation, ces simulateurs sont destinés à valider des commandes centralisées de métro ou de trains comme pour le Caire, Hong Kong, Athènes, Jakarta, Istanbul, Singapour.

- AGATE pour SPSE

  Planification des arrivées de navires aux môles du Port Autonome de Marseille, placement intelligent des produits pétroliers dans les réservoirs du dépôt, planification des livraisons par pipeline et optimisation des coûts de transport : AGATE a procuré une souplesse incomparable pour tout le personnel de la société du Pipeline Sud Européen.

- MATOS

  MATOS est un outil logiciel supportant les activités de modélisation des tâches sur lesquelles s'appuie la démarche de formalisation du besoin et de spécifications d'IHM de Pacte Novation. MATOS implémente en partie les principes de MAD (Scapin, Sébillote, … de l'INRIA).

- Storia sur Internet

  Le projet Storia sur Internet consiste à visualiser, via une applet dans un browser, le trafic aérien civil sur l'Europe. L'applet montre une carte de l'Europe sur laquelle des points symbolisant les avions se déplacent automatiquement sans demande explicite de rafraîchissement. Elle peut utiliser des données historiques (mode « Replay ») ou des données Temps Réel (Mode « live ») en provenance du salon ATC-Maastricht 2001 et c'était, de l'avis de tous les participants, une grande première européenne et l'innovation la plus « décapante ».

1.3.3) Partenariats

Cette stratégie sérieuse fait qu'un climat de confiance tout naturel s'établit entre le client et PACTE NOVATION. Ainsi, après plusieurs années de collaboration, du statut de fournisseur, PACTE NOVATION devient partenaire. Nous intervenons aujourd'hui aux côtés de certains de nos clients pour les aider à répondre à des cahiers des charges, et ce directement en contact avec leur propre client. Nous leur assurons également des formations sur des sujets pointus dans des services de R&D sensibles.

Notre offre, très souvent enrichie par une double compétence fonctionnelle, permet à PACTE NOVATION de nouer des partenariats durables avec :

ILOG Editeur de composants logiciels.

ProACTIVE Projet de recherche mené dans le cadre du programme Esprit sous l'égide de la Communauté Européenne.

EUROTEAM Consortium accrédité par EUROCONTROL, l'organisme européen fédérant les recherches et développements dans le domaine du contrôle aérien.

TEMPOSOFT Editeur de logiciels Intranet pour la gestion et l'optimisation des ressources humaines

SOLLAC Le plus gros projet de l'industrie utilisant des techniques d'Intelligence Artificielle

Chapitre 2 : La programmation linéaire dans le domaine de la planification

2.1) les différentes modélisations d'un problème de planification en IA

Un problème de planification en Intelligence Artificielle est défini par :

- une description de l'*état initial*,
- une description partielle de l'*état final*,
- une description des différentes actions possibles.

Ces descriptions doivent être formulées dans un langage formel. Actuellement, trois types de modélisations sont très utilisés dans ce domaine : Strips, Satplan, Graphplan. Avant la description de ces modélisations, nous définirons quelques mots de vocabulaires spécifiques à ce domaine.

Le but de l'Intelligence Artificielle dans ce domaine est de trouver une séquence d'actions qui permet de transformer l'*état initial* dans un *état final* qui contienne les buts.

2.1.1) Vocabulaire

Pas (*step*) : correspond à un pas de temps, généralement, on discrétise le temps avec un ensemble fini par exemple de 0 à n. Ainsi un pas correspond au passage d'un temps i à i+1.

Atome, fluent : correspond à la valeur d'une proposition qui décrit le problème, l'ensemble de ces variables permet de décrire l'état du problème.

Opérateur de planification ou action : c'est l'ensemble des actions possibles décrit dans un langage formel permettant de définir les conditions nécessaires à la réalisation de cette action ainsi que l'ensemble des effets de l'action sur les atomes.

Etat : c'est l'ensemble des atomes à un instant t, il décrit ainsi parfaitement le problème à chaque période.

Effet : les changements apportés par une action sur les atomes.

2.1.2) STRIPS

STRIPS (Fikes & Nilsson 1971) est une des plus populaires notations pour la représentation des problèmes de planification dans le domaine de l'Intelligence Artificielle. A l'origine, il a été construit pour planifier la tache d'un robot mobile.

La représentation d'un problème de planification avec STRIPS (STanford Research Institute Problem Solver) est un ensemble R=(S, G, O) défini par :

- S, l'*état initial*, est défini par un ensemble d'atomes qui sont à la valeur vrai.

- G, *l'état final*, est composé des atomes que l'on désire vrai à la fin du problème.

- O est l'ensemble des opérateurs de planifications ou encore des actions possibles. Chaque opérateur est défini par :
  - un nom
  - *liste de pré-condition*, c'est un ensemble d'atomes qui doivent être à vrai, pour que l'action puisse être exécutée.
  - *liste de retrait*, c'est l'ensemble des atomes qui sont mis à faux lorsque l'action est réalisée.
  - *liste d'ajout,* c'est l'ensemble des atomes qui sont mis à vrai lorsque l'action est réalisée.

Ici, chaque action dure exactement un pas de temps et à chaque pas de temps, on a seulement une seule action.

Soit a une action, on note **Pa** sa *liste de pré-condition*, **Da** sa *liste de retrait* et **Aa** sa *liste d'ajout* et enfin E un état quelconque.

Soit E l'état du problème à un instant donné t.

Si E vérifie Pa, on peut appliquer l'action a et on note result(E, a) = (E – Da) $\cup$ Aa. Ce résultat décrit l'état du problème à la période t+1, après avoir exécuté l'action a.

Trouver un plan vérifiant le problème posé revient à trouver une séquence d'ensemble d'actions (a_1, a_2, a_3, …, a_{k-1}, a_k) permettant, en partant de l'état initial S, arrivé dans un état où toutes les conditions de l'état final G sont vérifiés i.e.

- si pour tout i, on a l'état result(result(…(result(S, a_1) …), a_{i-2}), a_{i-1}) qui vérifie Pa_i, et
- G $\subseteq$ result(result(…(result(S, a_1) …), a_{k-1}), a_k).

Exemple d'une modélisation STRIPS :

Problème de transport de cargaison : on veut déplacer une cargaison C avec un camion R d'un endroit X à un endroit Y.

Name : move(R, X, Y)
Pre : at(R, X), has-fuel(R)
Del : at(R, X), has-fuel(R)
Add : at(R, Y)

Name : load(R, X, C)
Pre : at(R, X), at(C, X)
Del : at(C, X)
Add : in(C, R)

Name : unload(R, X, C)
Pre : at(R, X), in(C, R)
Del : in(C, R)
Add : at(C, X)

2.1.3) SAT-PLAN

SAT-PLAN créé par Kautz & Selman (1996) a connu un très grand succès depuis ces dernières années. Deux raisons pour expliquer ce succès :

- l'utilisation des propositions logiques pour modéliser les problèmes de planification
- la puissance des SAT-solvers comme Walksat (Selman, Kautz, et Cohen 1996)

Les Sat-solvers se repose sur la résolution d'un problème de satisfaction de contraintes logiques.

Avec la représentation d'un problème de planification avec la méthode STRIPS, on ne peut pas avoir plusieurs actions dans un même pas de temps. Cependant, avec SAT-PLAN, on peut exécuter plusieurs actions en même temps s'il n'y a pas de conflits (c à d si des actions n'ont pas des effets contradictoires ou si les effets d'une action ne sont pas négatifs sur une pré-condition d'une autre).

Le codage d'un problème de planification avec la méthode SAT-PLAN consiste donc à utiliser des propositions logiques. L'*état initial* donne la valeur de tous les *fluents* au temps 0 et l'*état final* donne tous les *fluents* qui doivent être vrai à la fin (i.e. au temps n).

Ensuite, les propositions logiques doivent traduire les pré-conditions des actions à l'instant t et leurs effets à l'instant t+1.

De plus, comme indiqué ci-dessus, si on veut pouvoir *paralléliser* les actions, il faut traduire toujours avec des propositions logiques le fait que deux actions qui sont en conflits ne peuvent pas se produire en même temps.

Pour trouver le plan en une durée minimale, on code le problème sur une durée n_0 puis on utilise un SAT-solver. Si on ne trouve pas de solution, on incrémente la durée et on réitère le procédé.

Exemple de codage

Par exemple, l'action « l'avion1 vole de Paris à Londres » à la période t implique que l'avion1 doit être à Paris à la période t et à t+1, il est à Londres mais plus à Paris. Cette contrainte peut être formulée avec une proposition logique du premier ordre de la manière suivante :

$$\text{fly(airplane1, Paris, Londres, t)} \Rightarrow \text{at(airplane1, Paris, t)} \wedge \neg\text{at(airplane1, Paris, t)} \wedge \text{at(airplane1, Londres, t+1)}$$

SAT-Solvers :

- solver systématique

  Satz (Li et Annbulagan, 1997)

- solvers stochastiques

  Blackbox (Gomes, 1998), version stochastique du solver *Satz*

  Walksat (Selman, Katz, et Cohen, 1996), le solver le plus utilisé.

2.1.4) GRAPH-PLAN

Graphplan a été mis au point par Blum et Furst en 1997. C'est une avancée très intéressante dans le domaine de l'Intelligence Artificielle car Graphplan est simple et produit des résultats très rapides dans beaucoup de cas.

La méthode Graphplan consiste à construire à partir du problème de planification un graphe. Sur ce graphe, on a deux types de nœuds, un représente les différents atomes et l'autre représente les actions. Ensuite, on a trois types d'arc :

- des arcs qui ne correspondent à aucune action, c'est à dire que l'on maintient simplement un fluent dans son état

- les arcs qui correspondent à une action : ils relient un nœud des atomes (à une date i-1) qui correspondent à une pré-condition d'une action a (à une date i), au nœud représentant l'action a, mais aussi reliant l'action a aux différents nœuds qui sont les effets de l'action a, ce sont des atomes à la date i+1
- des arcs qui permettent de représenter les actions conflictuelles, ces arcs seront appelés arcs *«mutex»* (mutuellement exclusives). On place un arc de ce type entre deux nœuds d'action si
    - une des deux actions a un effet qui est la négation de l'effet de l'autre action, appelé *effet inconsistant*.
    - une des deux actions annule une pré-condition de l'autre appelé *interférence*.
    - les deux actions ont des pré-conditions opposées

- un arc *«mutex»* est placé aussi entre un arc de maintien et une action si l'atome maintenu est opposé à un effet de l'action
- des arcs *«mutex»* sont placés entre les atomes opposés au même instant i.

On obtient ainsi un graphe du type suivant :

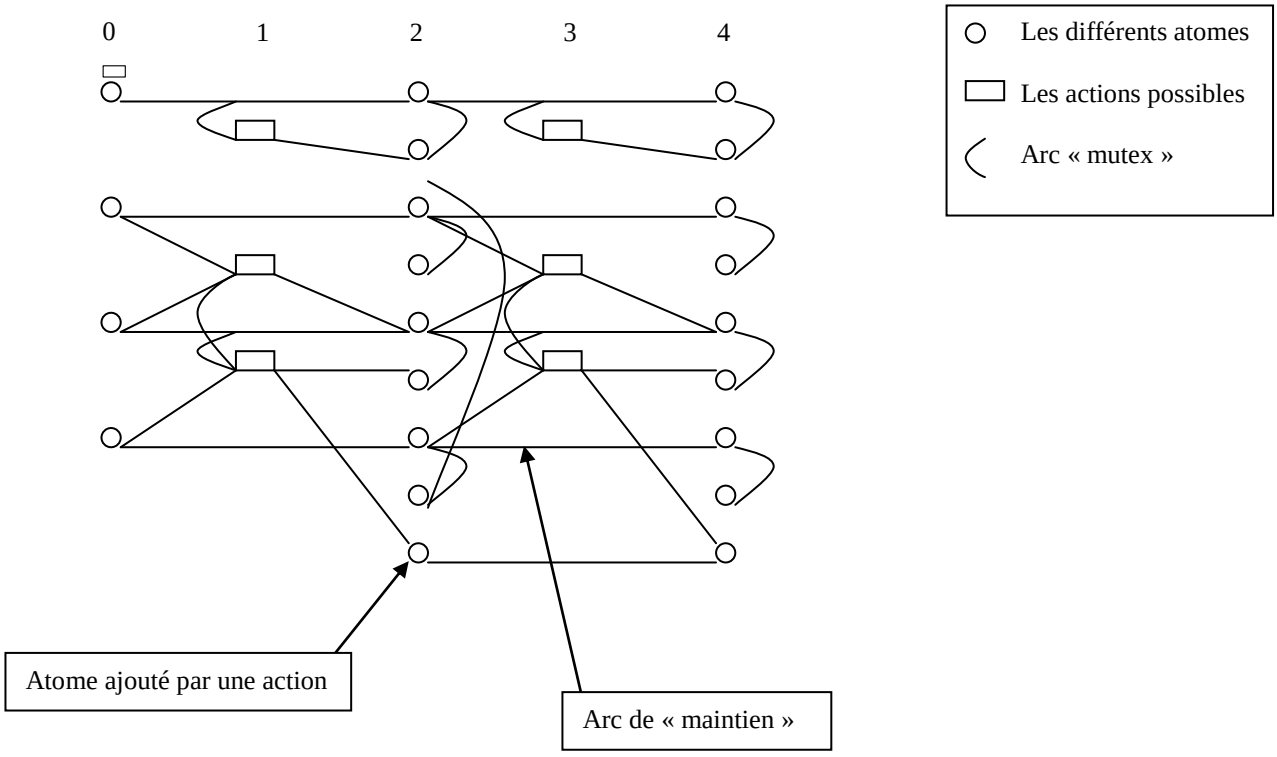


Figure 2 : graphe formé à partir d'un problème de planification

A l'instant 0, les nœuds correspondent à l'*état initial*, ou plus précisément aux atomes de l'*état initial*.

On montre que l'algorithme de création d'un tel graphe est polynomial dans la longueur de la description du problème et avec le nombre de pas de temps.

Pour la recherche d'un plan réalisable dans ce graphe, l'algorithme du Graphplan utilise la méthode de backward-chaining. Il utilise une approche niveau par niveau pour profiter au mieux des contraintes d'exclusion mutuelle.

Les différentes implémentations de Graphplan

- Graphplan, l'original implémenté en C
- IPP, une version optimisée implémentée en C
- STAN, une autre version optimisée implémentée en C
- SGP, une version simple en LISP

2.2) modélisation d'un problème de planification en nombre entier

De récentes recherches ont montré de grandes possibilités de résoudre les problèmes de planification connus dans le monde de l'Intelligence Artificielle en utilisant la programmation linéaire en nombre entier. De plus, la programmation linéaire en nombre entier (PLNE) permet d'ajouter des coûts d'actions, des contraintes de ressource ou de capacité, ce qui correspond plus au domaine de la recherche opérationnelle. Cependant, cela correspond sans doute à un plus grand nombre de problèmes de planification du monde réel. Sur la différence entre l'Intelligence Artificielle et la recherche opérationnelle dans le domaine de la planification, on peut dire que l'IA permet de déterminer la faisabilité du problème et la RO s'occupe plus de l'optimisation de ce même problème.

Un programme linéaire mixte est présenté sous la forme suivante :

$$\begin{cases} A\,x \le b_1 & (1) \\ B\,y \le b_2 & (2) \\ \text{Min } c_1^T x + c_2^T x & (3) \end{cases}$$

Où $A \in \mathbb{R}^{m*n}$, $B \in \mathbb{R}^{l*p}$, $b_1 \in \mathbb{R}^m$, $b_2 \in \mathbb{R}^l$, $x \in \mathbb{Z}^n$, $y \in \mathbb{R}^p$, $c_1 \in \mathbb{R}^m$, $c_2 \in \mathbb{R}^l$.

Il faut noter aussi que si p = 0, on dit que l'on a un programme linéaire en nombre entier (B, b_2 et c_2 disparaisse).

La fonction $c_1^T x + c_2^T x$ est appelée la fonction objectif (notant que maximiser $c_1^T x + c_2^T x$ est équivalent à minimiser $- c_1^T x - c_2^T x$).

Une solution réalisable est une solution qui vérifie (1) et (2).

Si z* est une solution réalisable et si de plus, elle vérifie (3) alors on dit que c'est une solution optimale.

2.2.1) formulation en nombre entier

Dans cette partie, nous allons exposer les méthodes pour traduire un problème en nombre entier.

Les clauses sont traduites en équations linéaires très facilement :

La clause suivante :

$x_1 \vee \ldots \vee x_k \vee \neg x_{k+1} \vee \ldots \vee \neg x_{k+l}$

est équivalent à l'inégalité suivant :

$x_1 + \ldots + x_k - x_{k+1} - \ldots - x_{k+l} \leq 1 - l$ où $x_1 \ldots x_{k+l}$ sont des variables booléennes.

De même, on peut traduire $x_1 \wedge x_2 \wedge \ldots \wedge x_k$ par $x_1 + x_2 + \ldots + x_k \geq k$

ou encore $x_1 - x_2 - \ldots - x_k \leq -k$.

Un autre exemple d'équations logiques qui se traduisent assez simplement en équations linéaires :

$$\begin{cases} s_i \rightarrow \neg a_i \\ s_i \rightarrow (r_{i+1} \geq N) \\ \neg s_i \rightarrow (k_i = 0) \end{cases}$$

qui peut se mettre sous la forme :

$$\begin{cases} a_i + s_i \leq 1 \\ 0 \leq r_{i+1} - Ns_i \\ 0 \leq Ns_i - k_i \end{cases} \quad \text{ou encore} \quad \begin{cases} a_i + s_i \leq 1 \\ -r_{i+1} + Ns_i \leq 0 \\ -Ns_i + k_i \leq 0 \end{cases}$$

Ces techniques vont nous permettre de passer ainsi d'un problème formulé avec STRIPS ou Fastplan en programme linéaire.

En voici un exemple :

Notation

Soit $\Im$, l'ensemble des *fluents*.

Soit A, l'ensemble des actions possibles.

Soit S, l'*état initial*.

Soit G, l'*état final*.

i représente une période, $i \in 1 \ldots t+1$.

pre_f : l'ensemble des actions qui ont f comme pré-conditions.

add_f : l'ensemble des actions qui ajoutent f.

del_f : l'ensemble des actions qui annulent f.

cons : l'ensemble des actions consommatrices de ressource.

prod : l'ensemble des actions productrices de ressource.

Les variables

Pour tout $f \in \mathfrak{I}$ et i, on introduit les variables d'états définies comme ci dessus :

$$x_{f,i} = \begin{cases} 1 \text{ si le fluent f est vrai à la période i} \\ 0 \text{ sinon} \end{cases}$$

Pour tout $a \in A$ et i, on introduit les variables d'actions :

$$y_{a,i} = \begin{cases} 1 \text{ si l'action a est réalisé à la période i} \\ 0 \text{ sinon} \end{cases}$$

Les contraintes

- les contraintes liées à l'état initial et l'état final

$$x_{f,1} = \begin{cases} 1 \text{ si } f \in S \\ 0 \text{ si } f \notin S \end{cases}$$

$x_{f,t+1} = 1$ si $f \in G$.

- les pré-conditions

$y_{a,i} \leq x_{f,i} \quad \forall i, \forall a \in pre_f$

- les effets

$x_{f,i+1} \leq \sum_{a \in add_f} y_{a,i} \quad \forall i, \forall f \in \Im.$ *Car dans les variables d'actions sont comprises aussi les « no – op. » actions.*

- <u>les actions conflictuelles</u>

  $y_{a,i} + y_{a',i} \leq 1 \quad \forall a, a' / \exists f / a \in del_f$ et $a' \in pre_f \cup add_f$

- <u>les ressources, actions consommatrices, productrices, fournisseur.</u>

  Notons s un fournisseur, c_a le coût d'une action a, r_i la valeur de la ressource r à la période i.

  Une manière de simple de coder ce problème en équations linéaire est la suivante :

$$\begin{cases} r_{i+1} = r_i + k_i - \sum_{a \in cons} c_a y_{a,i} + \sum_{a \in prod} c_a y_{a,i} \\ y_{a,i} + y_{s,i} \leq 1 \qquad \forall a \in prod \cup cons \\ 0 \leq r_{i+1} - N\, y_{s,i} \\ 0 \leq N\, y_{s,i} - k_i \\ M \leq r_i \leq N \\ y_{a',i}\, p_{a'} \leq r_i - \sum_{a \in cons \setminus \{a'\}} c_a y_{a,i} \qquad \forall a' \text{ ayant des pré-conditions sur la ressource r} \end{cases} \quad (4)$$

où M, N correspondent aux bornes minimum et maximum de la ressource r,

et l'équation (4) permet de traduire une pré-condition de ressource ,i.e., si une action a' a besoin d'une quantité minimale $p_{a'}$ de ressource r pour se produire.

Il faut noter aussi que pour la modélisation en nombre entier, il faut mieux utiliser des conditions les plus fortes possibles. Ceci dans un souci d'améliorer l'efficacité de l'algorithme utilisé pour résoudre ce problème.

Par exemple, pour modéliser le fait qu'un bus ne puisse pas être dans différentes gares en même temps :

- on introduit la variable $X_{b,g,i}$ qui indique que le bus b est à la gare g à la période i
- $X_{b,g,i} + X_{b,g',i} \leq 1 \ \forall g' \neq g$ (5)
- $X_{b,g,i} + \sum_{g' \neq g} X_{b,g',i} \leq 1$ (6)

Les équations (5) et (6) traduisent la même contrainte, mais il faut utiliser la formulation (6) qui est plus forte ie (6) $\Rightarrow$ (5) mais on n'a pas (5) $\Rightarrow$ (6).

, ou si on a $x_1 + x_2 \leq 1$, $x_1 + x_3 \leq 1$, $x_2 + x_3 \leq 1$, il faut mieux remplacer ces inégalités par :

$x_1 + x_2 + x_3 \leq 1$, qui est une contrainte plus forte.

Dans cette optique, la contrainte portant sur les actions conflictuelles :

$y_{a,i} + y_{a',i} \leq 1 \quad \forall a, a' / \exists f / a \in del_f$ et $a' \in pre_f \cup add_f$ a été remplacé par :

Soit $f \in \Im$,

$$\forall a \in del_f - del_f \cap (pre_f \cup add_f)$$

$$\sum_{a' \in delf \cap (pref \cup addf)} y_{a',i} + y_a \leq 1$$

$$\forall a \in (pre_f \cup add_f) - del_f \cap (pre_f \cup add_f)$$

$$\sum_{a' \in delf \cap (pref \cup addf)} y_{a',i} + y_a \leq 1$$

$$\forall a \in del_f - del_f \cap (pre_f \cup add_f) \text{ et } \forall a' \in (pre_f \cup add_f) - del_f \cap (pre_f \cup add_f)$$

$$y_{a',i} + y_{a,i} \leq 1$$

Si $card(del_f - del_f \cap (pre_f \cup add_f)) == 0$ *et* $card((pre_f \cup add_f) - del_f \cap (pre_f \cup add_f)) == 0$

$$\sum_{a' \in delf \cap (pref \cup addf)} y_{a',i} \leq 1$$

Ces équations sont obtenues à partir de la formulation initiale, en décomposant simplement les ensembles.

2.2.2) formulation appelée « changement d'état »

C'est une alternative à la formulation précédente mais qui donne des résultats plus intéressants (plus rapide). En effet, c'est une formulation qui est plus forte que la précédente.

Elle consiste à choisir des variables qui marquent le fait de changer d'état. Nous allons introduire diverses notations pour présenter cette nouvelle formulation.

Notations

Soit $\Im$, l'ensemble des *fluents*.

Soit A, l'ensemble des actions possibles.

Soit S, l'*état initial*.

Soit G, l'*état final*.

i représente une période, $i \in 1 \ldots t+1$.

Les variables

Les variables d'action sont toujours les mêmes :

Pour tout $a \in \mathsf{A}$, on introduit les variables d'actions :

$$y_{a,i} = \begin{cases} 1 \text{ si l'action a est réalisé à la période i} \\ 0 \text{ sinon} \end{cases}$$

Et ensuite on introduit les variables auxiliaires suivantes :

Pour tout $f \in \Im$,

$x_{f,i}^{maintain}$ ne représente aucune action, laissant ainsi un fluent à vrai s'il était à vrai à la période précédente. Ainsi les « no - op. » actions **ne sont plus prises en compte** dans les variables d'actions.

$x_{f,i}^{pre\text{-}add} = \bigvee_{a \in pre_f / add_f} y_{a,i}$, i.e., =1 ssi une action qui a f comme pré-condition et qui ne l'efface pas. est exécutée à la période i.

$x_{f,i}^{pre\text{-}del} = \bigvee_{a \in pre_f \cap del_f} y_{a,i}$, i.e., =1 ssi une action qui a f comme pré-condition et qui le détruit est exécutée à la période i

$x_{f,i}^{add} = \bigvee_{a \in add_f / pre_f} y_{a,i}$, i.e., =1 ssi une action qui ajoute f sans avoir ce fluent comme pré-condition est exécutée à la période i.

Les contraintes

- les contraintes liées à l'état initial et l'état final

$$\begin{cases} 1 \text{ si } f \in S \end{cases}$$

$x_{f,0}^{add} =$

0 si $f \notin S$

$x_{f,t}^{add} + x_{f,t}^{pre\text{-}add} + x_{f,t}^{maintain} \geq 1$ pour tout $f \in G$.

- liées à l'interprétation des variables, pour tout $i \in 1 \ldots t$

$$\sum_{a \in pre_f \setminus del_f} y_{a,i} \geq x_{f,i}^{pre\text{-}add}$$

$y_{a,i} \leq x_{f,i}^{pre\text{-}add}$ $\forall a \in pre_f \setminus del_f$

$$\sum_{a \in add_f / pre_f} y_{a,i} \geq x_{f,i}^{pre\text{-}add}$$

$y_{a,i} \leq x_{f,i}^{add}$ $\forall a \in add_f \setminus pre_f$

$$\sum_{a \in pre_f \cap del_f} y_{a,i} = x_{f,i}^{pre\text{-}add} \quad (7)$$

L'égalité dans l'équation (7) est justifiée, car toutes les actions de l'ensemble $pre_f \cap del_f$ sont mutuellement exclusives.

- les effets

$x_{f,i}^{add} + x_{f,i}^{maintain} + x_{f,i}^{pre\text{-}del} \leq x_{f,i-1}^{add} + x_{f,i-1}^{maintain} + x_{f,i-1}^{pre\text{-}add}$

si cette somme est à 0, alors f = 0 à la période i.

- les actions conflictuelle

$$x_{f,i}^{add} + x_{f,i}^{maintain} + x_{f,i}^{pre\text{-}del} \leq 1$$

$$x_{f,i}^{pre\text{-}add} + x_{f,i}^{maintain} + x_{f,i}^{pre\text{-}del} \leq 1$$

chapitre 3 : Implémentation

3.1) Le parser

Dans un premier temps , Il nous a fallu développer un outil pour pouvoir analyser des fichier au format PDDL (Planning Domain Definition Language). Ce format est utilisé pour décrire les problèmes de planification. Il se repose sur la formulation STRIPS, avec des extensions permettant d'utiliser les quantificateurs universels.

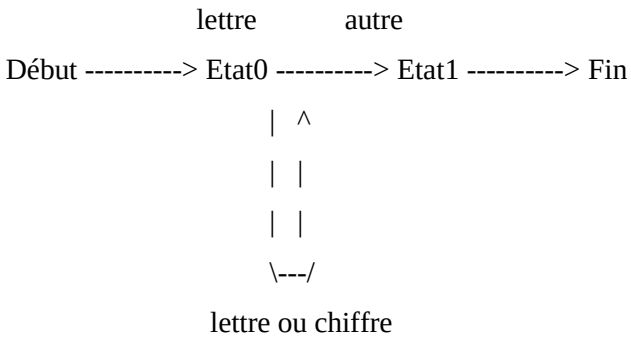
Il est notamment utilisé dans les concours de planification (aips 98, aips 2000).

Ce parser a été élaboré par Lex-Yacc. Lex est un analyseur lexical et Yacc est un analyseur syntaxique.

3.1.1) Lex

Le rôle de l'analyse lexicale est de fournir une suite de "tokens" (les éléments terminaux du langage) à partir d'une suite de caractères fournis en entrée. L'analyse syntaxique est une analyse hiérarchique. L'objectif de cette analyse est de regrouper les "tokens" fournis par l'analyse lexicale en phrases grammaticales.

Comme nous l'avons indiqué ci-dessus, Lex va lire des données afin de convertir les caractères en "tokens" compréhensibles par l'analyseur syntaxique. Pour permettre à Lex d'identifier des patrons de caractères, nous allons utiliser les expressions régulières. A chaque patron est associé une action (en général, il s'agit de retourner le "token" correspondant au patron). Par exemple, supposons qu'un identificateur soit de la forme lettre(lettre|chiffre)*" c'est à dire une lettre suivit d'une lettre ou d'un chiffre zéro ou plusieurs fois (x, x1, xx, etc ...). Il est alors possible de représenter ce patron par un automate à état fini (FSA : finite state automaton):



L'état "Fin" représente un état validant notre identificateur. De façon plus algorithmique, nous pourrions écrire :

```
Début : aller à Etat0

Etat0 : lire c
        si c = lettre aller à Etat1
        aller à Etat0

Etat1 : lire c
        si c = lettre aller à Etat1
        si c = chiffre aller à Etat1
        aller à Fin

Fin   : accepter l'identificateur
```

C'est la technique utilisée par Lex. Les expressions régulières sont représentées sous forme d'un automate.

3.1.2) Yacc

Pour spécifier la syntaxe d'un langage, il est possible d'utiliser ce que l'on appel les grammaires libres de contexte (context-free) ou BNF (forme de Backus-Norm). Une grammaire libre de contexte possède quatre composantes qui sont :

- Un ensemble de "tokens" ou terminaux du langage
- Un ensemble de non-terminaux
- Un ensemble de règles de productions ou chacune des règles consistent en un non-terminal à gauche de la production (left-side), une flèche et une séquence de tokens et/ou de non-terminaux à droite de la production (right-side)
- Un non-terminal de départ.

Les grammaires pour Yacc sont décrites en utilisant cette forme. La plupart des langages modernes peuvent être décris sous cette forme.

Prenons par exemple la grammaire suivante :

(r1) E -> E + E
(r2) E -> E * E
(r3) E -> id

Nous venons de spécifier trois règles de production. Cette grammaire indique qu'une expression peut être la somme de deux expressions, le produit de deux expressions ou un identificateur. Nous pouvons utiliser cette grammaire pour générer l'expression suivante :

E -> E * E (r2)
-> E * z (r3)
-> E + E * z (r1)
-> E + y * z (r3)
-> x + y * z (r3)

A chaque étape, nous avons remplacé le membre droit d'une production avec le membre gauche de cette même production (les productions sont notées r1, r2 et r3). Pour "parser" une expression, Yacc a besoin de faire l'opération inverse. Plutôt que de partir d'un non-terminal et générer une expression à partir de la grammaire, il faut partir d'une expression et la réduire à un non-terminal. Cette technique s'appelle un "parsage" bottom-up ou shift-reduce et, elle utilise des piles pour stocker les termes. Voici la même dérivation que précédemment mais en bottom-up :

1 .x + y * z (shift)
2 x .+ y * z (reduce [r3])
3 E .+ y * z (shift)
4 E + .y * z (shift)
5 E + y .* z (reduce [r3])
6 E + E .* z (shift)

7 E + E * .z (shift)
8 E + E * z. (reduce [r3])
9 E + E * E. (reduce [r2])
10 E + E. (reduce [r1])
11 E. (accept)

Le "." symbolise le "lookhead" qui est le symbole courant. Le terme à gauche du "." est sur la pile. Lorsque le sommet de la pile correspond à un terme se trouvant à droite d'une production, on remplace le token correspondant avec la partie gauche de la règle de production. A l'étape 1, on place x sur la pile. L'étape 2 applique la règle r3 et, x est donc remplacé par la partie gauche de la règle 3 (c'est à dire E). Nous continuons le processus jusqu'à ce qu'il n'y ait plus qu'un seul non-terminal sur la pile.

Dans la grammaire que nous avons utilisé, nous pouvons remarquer des conflits. Par exemple, à l'étape 6, nous avons choisi de décaler le lookhead après l'opérateur '*' alors que nous aurions pu faire une réduction de "E + E" en utilisant la règle r1. Nous avons donc un conflit "shift-reduce". Ceci arrive car notre grammaire est ambiguë (il existe plusieurs dérivations possibles). Nous aurions pu avoir aussi un conflit "reduce-reduce" comme par exemple dans la grammaire suivante :

E -> T
E -> id
T -> id

Yacc résout ces problèmes en adoptant une action par défaut. Dans le cas d'un conflit "shift-reduce", Yacc va choisir le "shift". Dans le cas d'un conflit "reduce-reduce", Yacc va choisir la première règle dans la grammaire.

En cas de conflits, Yacc génère des "warnings" sur les règles problématiques. Il est aussi possible de lui spécifier un comportement.

3.1.3) la grammaire PDDL

Dans cette partie, nous allons énumérer les différentes règles constituant la grammaire PDDL. Il faut noter que la description d'un problème de planification est constitué dans la norme

PDDL de deux fichiers. L'un constituant la description du domaine i.e. la description d'un type de problème avec ces prédicats, permettant de connaître l'état du problème à chaque instant, la description des différentes actions…L'autre constituant une description de l'état initial, de l'état final, et des différents objets.

- Notation

- chaque règle est de la forme <élément syntaxique> : : = expression
- < > délimite le nom d'un élément syntaxique
- [] encadre un élément syntaxique optionnel
- * correspond à « zéro ou plus » et + correspond à « 1 ou plus »
- <liste x> est défini par <list x> : := (x*)

- Grammaire du fichier domaine

<domain> ::= (define (domain <nom>)
[<require-def>]
[<types-def>] :typing
[<constants-def>]
[<predicates-def>]
[<functions-def>]
<structure-def>*)

<require-def> ::= (:requirements <require-key> +)
<types-def> ::= (:types <typed list (nom)>)
<constants-def> ::= (:constants <typed list (nom)>)
<predicates-def> ::= (:predicates <atomic formula skeleton> +)
<atomic formula skeleton> ::= (<predicate> <typed list (variable)>)
<predicate> ::= <nom>
<variable> ::= ?<nom>
<atomic function skeleton> ::= (<function-symbol> <typed list (variable)>)
<function-symbol> ::= <nom>
<functions-def> ::= (:functions <function typed list (atomic functor skeleton)>)
<structure-def> ::= <action-def>
<structure-def> ::= :durations <durative-action-def>
<typed list (x)> ::= x*
<typed list (x)> ::= :typing x+ - <type> <typed list(x)>
<primitive-type> ::= <nom>
<type> ::= (either <primitive-type> +)
<type> ::= <primitive-type>
<function typed list (x)> ::= x*
<function typed list (x)> ::= :typing x + - <function type>
<function typed list(x)>
<function type> ::= nombre
<action-def> ::= (:action <action-symbol>
:parameters (<typed list (variable)>)
<action-def body>)
<action-symbol> ::= <nom>

<action-def body> ::= [:precondition <GD>]
[:effect <effect>]
<GD> ::= <atomic formula(term)>
<GD> ::= <litteral (term)>
<GD> ::= (and <GD>*>
<GD> ::= (or <GD>*>
<GD> ::= (not <GD>>
<GD> ::= (imply <GD> <GD>)
<GD> ::= (exists (<typed list (variable)>)* <GD> >
<GD> ::= (forall (<typed list (variable)>)* <GD> >
<GD> : := <f-comp>
<litteral (t)> : := <atomic formula (t) >
<atomic formula (t) > : := (not <atomic formula (t) >)
<atomic formula (t) > : := (<predicate> t*)
<term> : := <nom>
<term> : := <variable>
<f-exp> : := <nombre>
<f-exp> ::= (<binary-op> <f-exp> <f-exp>)
<f-exp> ::= <f-head>
<f-head> ::= (<function-symbol> <term>*)
<f-head> ::= <function-symbol>
<binary-op> ::= +
<binary-op> ::= -
<binary-op> ::= *
<binary-op> ::= /
<binary-comp> ::= >
<binary-comp> ::= <
<binary-comp> ::= =
<binary-comp> ::= >=
<binary-comp> ::= <=
<effect> ::= <a-effect>
<effect> ::= (forall (<variable>* <effect>)
<effect> ::= (when <GD> <a-effect>)
<a-effect> ::= (and <p-effect>)
<a-effect> ::= <p-effect>
<p-effect> ::= (<assign-op> <f-head> <f-exp>)
<p-effect> ::= (not <atomic formula(term)>)
<p-effect> ::= <atomic formula(term)>
<assign-op> ::= scale-down
<assign-op> ::= scale-up
<assign-op> ::= assign
<assign-op> ::= increase
<assign-op> ::= decrease

<durative-action-def> ::= (:durative-action <da-symbol>
:parameters (<typed list (variable)>)
<da-def body>)

<da-symbol> ::= <nom>
<da-def body> ::= :duration <duration-constraint>
:condition <da-GD>
:effect <da-effect>
<da-GD> ::= <timed-GD>
<da-GD> ::= (and <timed-GD> +)

<timed-GD> ::= (at <time-specifier> <GD>)
<timed-GD> ::= (over <interval> <GD>)
<time-specifier> ::= start
<time-specifier> ::= end
<interval> ::= all
<duration-constraint> ::= (and <duration-constraint>+)
<duration-constraint> : := (<d-op> ?duration <d-value>
<d-op> := <=
<d-op> := >=
<d-op> := =
<d-value> ::= nombre
<d-value> ::= <f-exp>
<da-effect> ::= (and <da-effect>*)
<da-effect> ::= <timed-effect>
<da-effect> ::= (forall (<variable)>)* <da-effect> >
<da-effect> ::= (when (<da-GD)>)* <time-effect> >
<da-effect> ::= (<binary-op> <f-exp-da> <f-exp-da>)
<timed-effect> ::= (at <time-specifier> <a-effect>)
<timed-effect> ::= (at <time-specifier> <f-assign-da>)
<timed-effect> ::= (at <f-head> <f-exp-da>)
<f-exp-da> ::= (<assign-op> <f-head> <f-exp-da>)
<f-exp-da> ::= (<binary-op> <f-exp-da> <f-exp-da>)
<f-exp-da> ::= ?duration
<f-exp-da> ::= <f-exp>
<assign-op-t> ::= increase
<assign-op-t> ::= decrease
<f-exp-t> ::= (* <f-exp> #t)
<f-exp-t> ::= (* #t <f-exp>)
<f-exp-t> ::= #t

<require-key> ::= :strips
<require-key> ::= :typing
<require-key> ::= :negative-preconditions
<require-key> ::= :disjunctive-preconditions
<require-key> ::= :equality
<require-key> ::= :existential-preconditions
<require-key> ::= :universal-preconditions
<require-key> ::= :quantified-preconditions
<require-key> ::= :conditional-effects
<require-key> ::= :fluents
<require-key> ::= :adl
<require-key> ::= :durations
<require-key> ::= :duration-inequalities
<require-key> ::= :continuous-effects

- Grammaire du fichier problème

<problem> ::= (define (problem <nom>)
(:domain <nom>)
[<require-def>]

[<object declaration>]
<init>
<goal> +
[<metric-spec>]
[<length-spec>])

<object declaration> ::= (:objects <typed list (nom)>)
<init> ::= (:init <init-el> +)
<init-el> ::= <literal(nom)>
<init-el> ::= (= <f-head> <nombre>)
<goal> ::= (:goal <GD>)
<metric-spec> ::= (:metric <optimization> <ground-f-exp>)
<optimization> ::= minimize
<optimization> ::= maximize
<ground-f-exp> ::= (<binary-op> <ground-f-exp> <ground-f-exp>)
<ground-f-exp> ::= <nombre>
<ground-f-exp> ::= (<functor> <nom>*)
<ground-f-exp> ::= total-time
<length-spec> ::= (:length [(:serial <integer>)] [(:parallel <integer>)])

3.1.4) la librairie ParserPDDL

L'analyseur des fichiers PDDL se présentent sous forme d'une librairie. Les données du fichier problème.pddl sont stockées dans une classe singleton DonneePb , de même les données du fichier domain.pddl sont stockées dans la classe singleton DonneeDom. Une documentation sur toute la structure donnée utilisée est fournie en annexe.

Dans cette partie, nous montrerons simplement de quelle manière a été implémenté le parser.

Tout d'abord, il faut savoir qu'en réalité quatre parsers ont été implémenté. Un pour analyser le fichier problème. Et les trois autres pour analyser le fichier domaine. En effet, ce dernier ayant une grammaire plus complexe, la grammaire a été découpé en plusieurs partie. Un parser s'occupe d'analyser les actions, un autre pour les « duratives-actions » et enfin, le troisième pour tout le reste c'est à dire tout le début du fichier avec notamment tous les prédicats.

Le problème rencontré avec l'utilisation de plusieurs parsers, est le suivant : Lex-Yacc utilise des variables globales, on a ainsi une redéfinition de plusieurs variables. La solution apportée fut de modifier à la main dans les différents fichiers générés le nom de toutes ces variables.

Pour ce qui est du stockage de donnée, à chaque règle trouvée, les données sont envoyées à un objet Builder qui permet de construire tous les objets correspondant, pour être stockés dans l'objet singleton correspondant.

3.2) Le modèle objet

Dans cette partie, nous allons montrer le modèle objet sur lequel est basé l'implémentation de ce projet.

3.2.1) le modèle UML

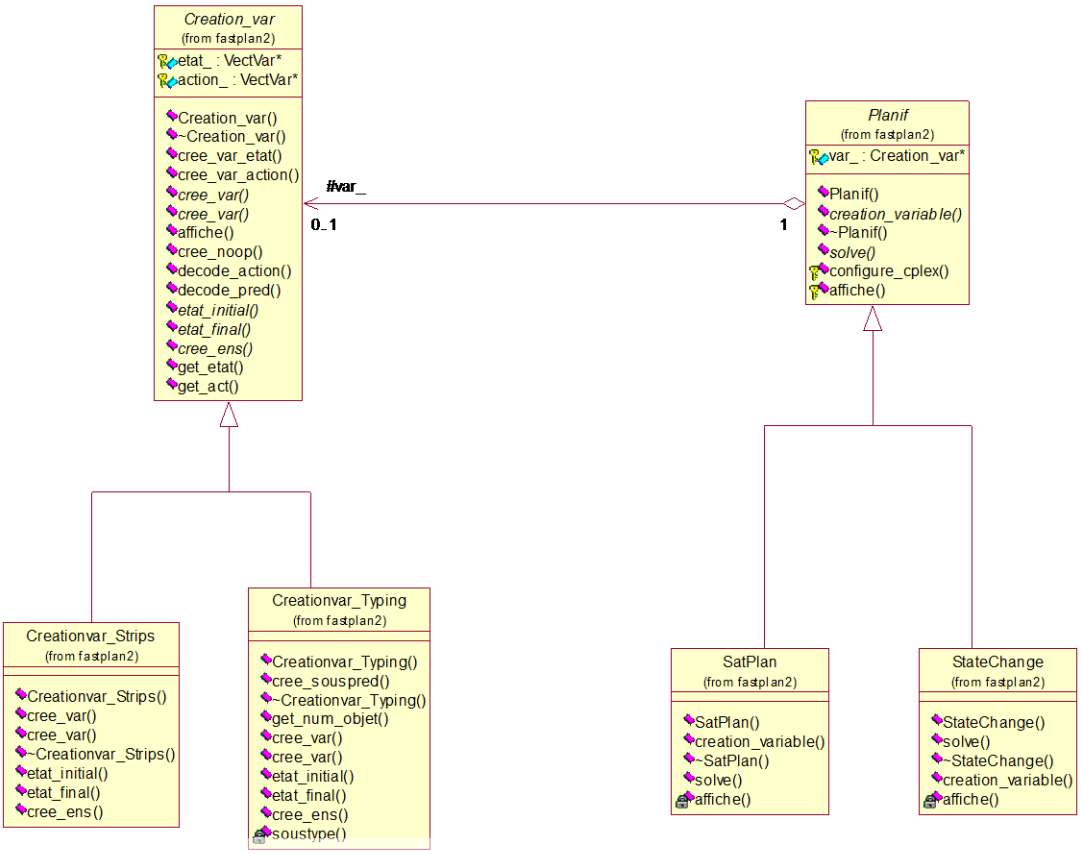


Figure 3 : modéle UML du planificateur

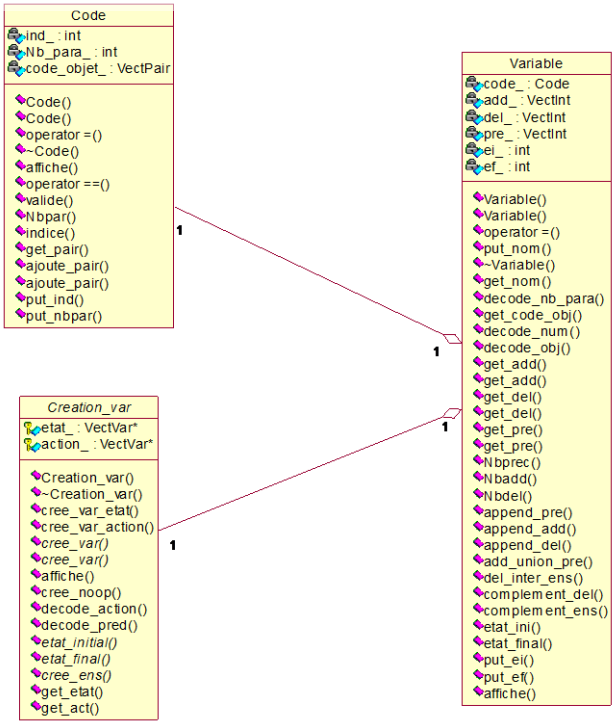


Figure 4 : modéle UML des variables

3.2.2) description des différentes classes

Le design pattern Strategy, classé dans les modèles comportementaux, définit une famille d'algorithmes, en les encapsulant dans des classes différentes. Il est alors possible de changer dynamiquement d'algorithme en gardant toujours la même interface.

Le pattern Strategy est utilisé ici à deux reprises. Une pour implémenter les deux modélisations en nombre entier du problème de planification (SatPlan et StateChange).Et une autre pour la création des variables, en effet nous n'allons pas utiliser les mêmes algorithmes si le problème de planification est typé ou pas.

Le choix de la modélisation (SatPlan ou StateChange) est choisi par l'utilisateur au cours de l'exécution du programme. Mais le choix de l'algorithme de la création des variables est déterminé à la fin de l'analyse des fichiers. En effet, dès l'analyse faite par le parser, on est capable de savoir si le problème est typé ou pas.

3.2.2.1) la classe Creation_Var

Cette classe permet à partir des données des fichiers PDDL de créer toutes les variables du problème, stockées dans deux vecteurs (etat_ et action_), les variables d'état et les variables d'action à travers les deux méthodes suivantes :

- *cree_var_action( )*
- *cree_var_etat()*

Ces deux méthodes passent en revue tous les prédicats pour créer toutes les variables d'état et toutes les actions pour les variables d'actions.

Ce sont les méthodes *cree_var(Predicat *pred, int ind)* et *cree_var(Action *act, int ind)* qui créent pour une action ou un prédicat donné les variables correspondant.

L'implémentation de ces méthodes dépendent du format des fichiers PDDL i.e. si les données du problème sont typés ou pas.

Pour un prédicat donné, on va créer toutes les variables par combinaison et on va leur associé un code (le code choisi sera présenté plus bas).

On présente ci dessus à travers un exemple la manière avec laquelle sont crées les variables à partir d'un prédicat donné (on procède de la même manière pour les actions) :

Sur un problème typé :

Les objets :

Marchandise : *M1, M2*

Lieu : *Paris, Londres, Madrid*

Le prédicat :

at (Marchandise, Lieu)

Les variables générées :

$x_1 \Leftrightarrow$ at(M1,Paris)
$x_2 \Leftrightarrow$ at(M1,Londres)
$x_3 \Leftrightarrow$ at(M1,Madrid)
$x_4 \Leftrightarrow$ at(M2,Paris)
$x_5 \Leftrightarrow$ at(M2,Londres)
$x_6 \Leftrightarrow$ at(M2,Madrid)

Sur un problème non typé :

Les objets :

M1, M2, Paris, Londres, Madrid

Le prédicat :

at (?obj, ?obj)

Les variables générées :

$x_1 \Leftrightarrow$ at(M1,Paris)
$x_2 \Leftrightarrow$ at(M1,Londres)
$x_3 \Leftrightarrow$ at(M1,Madrid)
$x_4 \Leftrightarrow$ at(M1, M2)
$x_5 \Leftrightarrow$ at(M2,Paris)
$x_6 \Leftrightarrow$ at(M2,Londres)
$x_7 \Leftrightarrow$ at(M2,Madrid)
$x_8 \Leftrightarrow$ at(M2,M1)
$x_9 \Leftrightarrow$ at(Paris, Londres)
$x_{10} \Leftrightarrow$ at(Paris, Madrid)
$x_{11} \Leftrightarrow$ at(Paris, M1)
$x_{12} \Leftrightarrow$ at(Paris, M2)
$x_{11} \Leftrightarrow$ at(Londres, Paris)
$x_{12} \Leftrightarrow$ at(Londres, Madrid)
$x_{13} \Leftrightarrow$ at(Londres, M1)
$x_{14} \Leftrightarrow$ at(Londres, M2)
$x_{15} \Leftrightarrow$ at(Madrid, Londres)
$x_{16} \Leftrightarrow$ at(Madrid, Paris)
$x_{17} \Leftrightarrow$ at(Madrid, M1)

$x_{18} \Leftrightarrow$ at(Madrid, M2)

De plus, la formulation SatPlan nécessite de créer des variables représentant les « *noop* ». Ce qui correspond en réalité à l'action de maintenir une variable d'état vrai sur un pas de temps. Cette opération est gérée par la méthode *cree_noop()*. Il suffit simplement de créer une variable d'action pour chaque variable d'état créé à l'étape précédente.

Lorsque les variables sont crées, il faut ensuite pour chaque variable construire sa liste de pré-condition, sa liste d'ajout, sa liste de retrait. C'est le rôle de la méthode *cree_ens()*. Elle regarde pour chaque variable d'action les prédicats en pré-condition, regarde le code correspondant à chaque prédicat, recherche dans le vecteur etat_, l'indice de cette variable d'état pour la stocker dans la liste de précondition.

Ensuite il nous faut déterminer les variables d'état présente à l'état initial et celle qui doivent être présente à l'état final. C'est le rôle des méthodes *etat_initial()* et *etat_final()*.

3.2.2.2) la classe Planif

La classe Planif contient seulement l'attribut var_ de type Creation_var, permettant ainsi de créer les variables du problémes, nécessaire à la construction des contraintes.

la méthode solve() :

crée toutes les équations définies dans le chapitre 2 et les envoient dans IlogCplex pour la résolution du problème linéaire en nombre entier généré. Cette méthode est implémenté dans les classes dérivées SatPlan et StateChange. En effet, conformément au chapitre précédent, le type d'équation dépend de la formulation choisie.

la méthode creation_variable() :

Cette méthode est implémentée dans les classes dérivées. En effet, la formulation SatPlan nécessite la création des « no-op » contrairement à la formulation StateChange. Elle permet de gérer la création des variables.

```
try
{
        /*creation des var etats*/
        var_->cree_var_etat();
        /*creation des var Actions*/
        var_->cree_var_action();
        /*creation des var noop*/
```

```
                var_->cree_noop();   (1)
                /*creation de l'ei*/
                var_->etat_initial();
                /*creation de l'ef*/
                var_->etat_final();
                /*creation des ens*/
                var_->cree_ens();
        }
        catch(exception &e)
        {
                cerr<<e.what();
                throw e;
        }
        catch(...)
        {
                throw;
        }
```

(1) Cette ligne n'est pas présente dans la classe StateChange (pas de « no-op »)

3.2.2.3) la classe Variable

Cette classe permet donc de stocker toutes les informations nécessaires à la modélisation du problème sur une variable :

- Code code_ : qui contient le code de la variable présenté dans le point suivant.
- Vector<int> pre_ : qui correspond aux indices des variables qui sont en pré-condition de celle-ci
- Vector<int> add_ : qui correspond aux indices des variables qui sont en ajout de celle-ci
- Vector<int> del_ : qui correspond aux indices des variables qui sont en retrait de celle-ci
- Int ei_ : est à 1 si cette variables est présente à l'état initial sinon à zéro
- Int ef_ : est à 1 si cette variables est présente à l'état final sinon à zéro

3.2.2.4) la classe Code

Cette classe nous permet d'attribuer un code à chaque variable du problème, permettant ainsi de savoir à quoi corresponde chaque variable.

Ce code est constitué de la manière suivante :

- int ind : correspond à l'indice dans le vecteur des prédicats (ou des actions si c'est une variable d'action) du prédicat ou de l'action correspondant. Sauf si cette variable correspond à un « no-op » et dans ce cas, ind = -2.

- int Nb_para_ : Le nombre de paramètre de l'action (ou du prédicat) sauf si cet variable correspond à un « no-op », correspond à l'indice de la variable d'état correspondant.

- vector< Pair < int, int > > Code_Objet_ : chaque paire permet d'identifier un paramètre, le premier entier correspond à l'indice du type (si le problème est typé sinon c'est tout simplement zéro) et le deuxième indice correspond à l'indice de l'objet.

Exemple :

Les objets :

le type : camion

les objets : truck1 ; truck2 ; truck3 ; truck4 ;

le type : lieu

les objets : Paris ; Londres ; Madrid ;

le type : paquet

les objets : obj1 ; obj2 ; obj3 ;

Les prédicats :

at (camion ; lieu)

in-city (paquet ; lieu)

in (paquet ; camion)

Le code correspondant au prédicat *« in (obj2, truck4)* » est :

- ind = 2
- Nb_para_ = 2
- Code_Objet_ = [(2,1) ; (0,3)]

3.3) Présentation de IlogCplex

3.3.1) Utilisation

IlogCplex est un outil permettant de résoudre tout type de problème linéaire. La version utilisée est Ilog Cplex 7.0. Son utilisation est assez explicite. En effet, la formulation du problème se fait de la manière suivante :

On déclare une variable d'environnement et une variable model :

IloEnv env ;

IloModel model(env) ;

Ensuite on peut créer un tableau de variable :

IloNumVarArray x(env) ;

Il faut construire chaque variable du tableau en précisant le type et les bornes de la variables comme sur l'exemple suivant :

x.add(IloNumVar(env, borne_inf, borne_sup, ILOINT) ;

Ensuite pour entrer les différentes équations, on peut déclarer un tableau de contraintes de la manière suivante :

IloRangeArray con(env) ;

Soit on ajoute une contrainte directement :

con.add(-x[0] + x[2] <= 20) ;

Soit on passe par une expression pour exprimer la contrainte :

IloExpr exp(env) ;

exp = x[1] ;

exp += x[2] ;

con.add(IloRange(env, borne_inf, exp, borne_sup)) ;

Dans les deux cas, il faut ajouter les contraintes au model :

model.add(con) ;

Et enfin, pour la fonction à optimiser

*model.add(IloMaximize(env, x[0] + 2*x[1])) ;*

Pour résoudre le problème, on déclare un objet Cplex

IloCplex cplex(model) ;

cplex.solve() ;

3.3.2) Paramétrage

Ilog Cplex permet de nombreux paramétrage pour la résolution. On peut en outre choisir l'algorithme principal de résolution :

1)Primal

2)Dual
3)Barrier
4)BarrierPrimal
5)BarrierDual

Mais aussi l'algorithme utilisé à la résolution de chaque nœud :

1)Primal
2)Dual
3)Barrier
4)DualBarrier

On peut utiliser également un « preprocessing » qui permet de diminuer la taille du problème en éliminant le contraintes redondantes.

De plus, divers heuristiques sont également proposées pour le « Branch & Cut ».

Chapitre 4 : Les résultats

Nous présenterons les résultats obtenus sous forme de tableau.

4.1) Comparaison entre les deux formulations

Dans un premier, nous exposerons le temps utilisé pour les deux formulations sur différents exemple :

Domaine	Problème	SatPlan Temps en sec.	StateChange Temps en sec.
Block	probBLOCKS-4-0.pddl	0.19	0.141
	bw-12step.pddl		55.6
	bw-large-a.pddl	**	465.72
	bw-large-b.pddl	**	5559.74
Logistics	probLOGISTICS-4-0.pddl	1.883	1.222
	probLOGISTICS-6-0.pddl	2.594	1.512
	prob002-rocket-a.pddl	**	504.245
	prob004-log-a.pddl	**	*

(**) correspond à l'absence de résultat après un jour de calcul
(*) out of memory

Figure 5 : tableau comparatif des résultats des 2 formulations

A la vue de ces résultats, on peut aisément conclure que la formulation StateChange est beaucoup plus performante que celle appelé SatPlan. Cependant il faut noter que cette dernière a l'avantage d'être beaucoup plus explicite.

4.2) Comparaison avec les différents paramétrages de Cplex

Les résultats obtenus avec les différentes paramétrisation de Cplex sont présenté dans le tableau ci-dessus. Il faut noter que tous ces tests ont été fait avec la formulation StateChange et que le problème Bw-12step.pddl

Domaine : block Problème : bw-12step.pddl	
RootAlgorithm : Dual NodAlgorithm : Dual Preprocessing	406.44
RootAlgorithm : barrier RootAlgorithm : barrier Preprocessing	318.488
RootAlgorithm : Dual NodAlgorithm : Dual Sans Preprocessing	1163.6
RootAlgorithm : BarierDual NodAlgorithm : BarierDual Preprocessing	1238.6

Figure 6 : tableau comparatif entre les différents paramétrisations de Cplex

De toute évidence, l'utilisation de Cplex avec ses paramètres par défaut est celle qui donne les meilleurs résultats (en temps de calcul).

Conclusion

La programmation linéaire dans le domaine de la planification est très récente. Elle présente l'avantage de pouvoir introduire aisément des actions avec coût et de traiter des problèmes avec ressource. Ce type de problème est sans doute beaucoup plus proche des problèmes du monde réel.

Le travail effectué s'est porté essentiellement sur des problèmes assez simple, seulement ceux basés sur la formulation STRIPS et typé ou non typé. On peut envisager un prochain stage qui permettrait d'élargir le type de problème traité. En introduisant par exemple la gestion des quantificateurs universels, la gestion de problème avec coût. En effet, maintenant, on dispose d'un parser capable de stocker toutes les données d'un problème de planification au format PDDL.

Les résultats obtenus montrent cependant la nécessité d'utiliser des machines assez puissantes. Le temps de calcul se montre en effet assez long dés que le nombre de variable du problème augmente, le planificateur utilise totalement le processeur pour trouver la solution.

Au cours de ce projet de fin d'études chez Pacte Novation, j'ai été amené à réaliser un travail demandant rigueur, autonomie, et méthodologie. En effet, les étapes de mon stage correspondent au cycle de vie d'un projet : étude et analyse des besoins, conception du modèle, réalisation du produit, pour finir avec les tests et validations. C'est pourquoi, grâce à ce projet, j'ai pu avoir un réel aperçu du travail d'un ingénieur en informatique.

De plus, j'ai pu mettre à profit mes connaissances acquises lors de mon enseignements à l'ISIMA et j'ai pu les enrichir avec de nouvelles technologies. Enfin, j'ai trouver des équipes compétentes et chaleureuses, si bien que j'aurai le plaisir de les rejoindre à l'issue de mon stage.

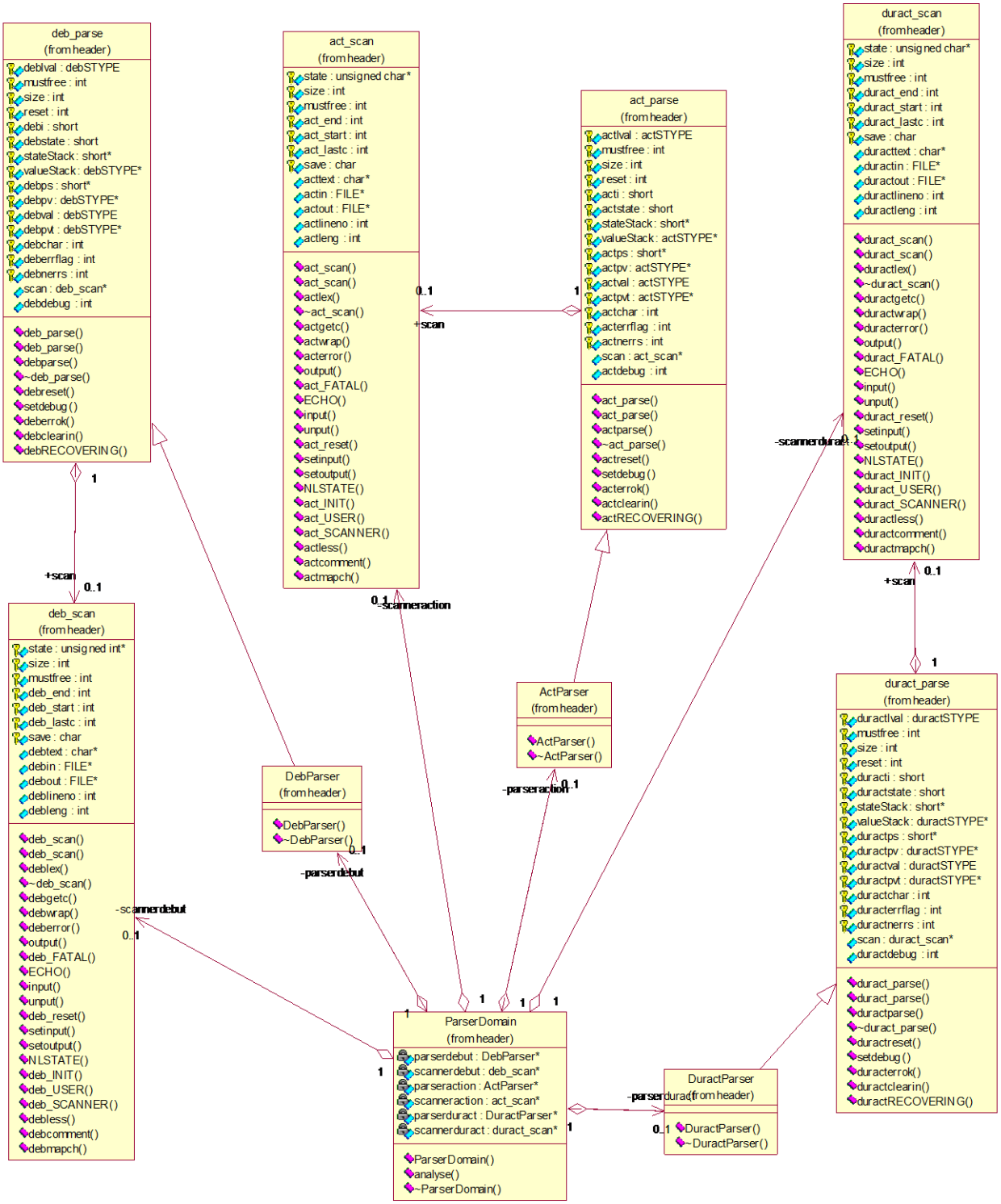
BIBLIOGRAPHIE

Avrim L. Blum & Merrick L. Furst *Fast planning through planning graph analysis*, 1997
Daniel S. Weld, *Recent advances in AI planning*, 1998
Thomas Vossen, Michael Ball, Amnon Lotem & Dana Nau *Applying integer programming to AI planning*,
Thomas Vossen, Michael Ball, Amnon Lotem & Dana Nau, *On the use of integer programming models in AI planning*, 1999
Henry Kautz & Joachim P. Walser, *Integer optimization models of AI planning problems*, 1999
Henry Kautz & Joachim P. Walser, *State-space planning by integer optimization*
Alexandre Bockmayr & Yannis Dimopoulos, *Integer programs and valid inequalities for planning problems*,
Henry Kautz, David McAllester & Bart Selman, *Encoding plans in propositional logic*, 1996.
Drew McDermott et al *PDDL –The planning Domain Definition Language. 1998.*

ANNEXE

TABLE DES MATIERES

ANNEXE A : Documentation de la librairie ParserPDDL.lib

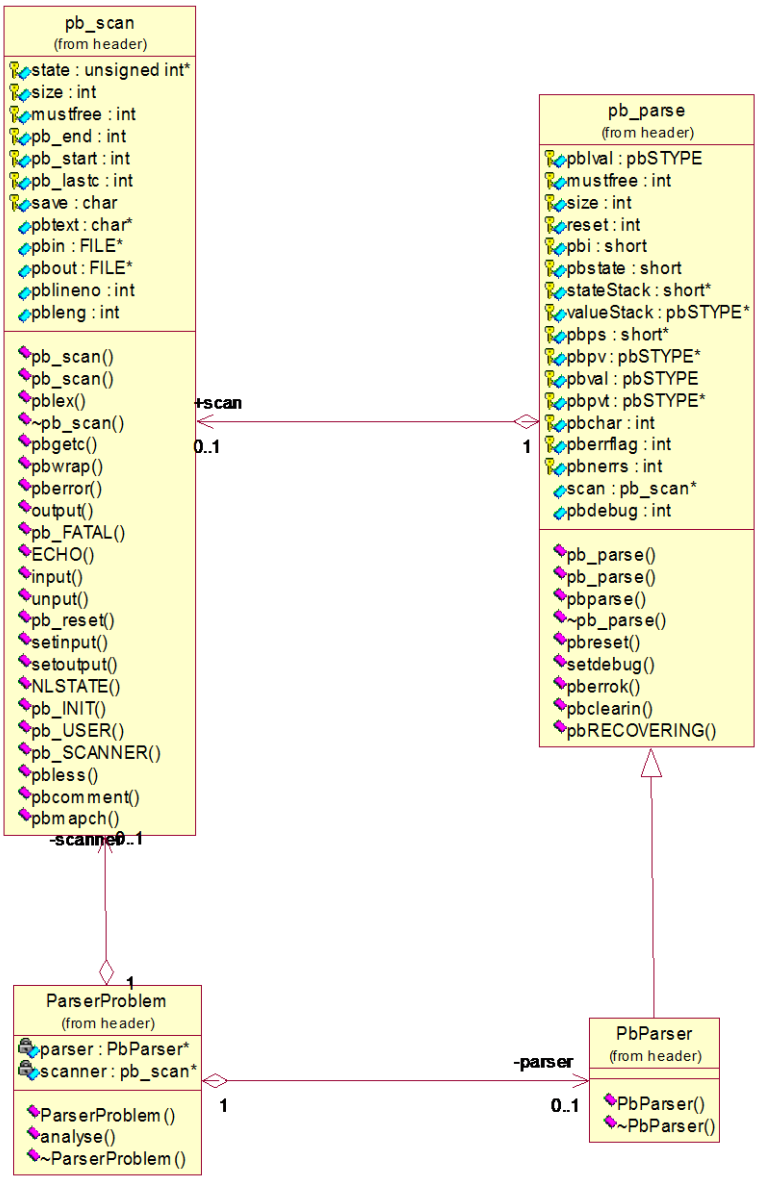

deb_parse
(from header)
deblval : debSTYPE
mustfree : int
size : int
reset : int
debi : short
debstate : short
stateStack: short*
valueStack: debSTYPE*
debps : short*
debpv : debSTYPE*
debval : debSTYPE
debpvt : debSTYPE*
debchar : int
deberrflag : int
debnerrs : int
scan : deb_scan*
debdebug : int
deb_parse()
deb_parse()
debparse()
~deb_parse()
debreset()
setdebug()
deberrok()
debclearin()
debRECOVERING()
act_scan
(from header)
state : unsigned char*
size : int
mustfree : int
act_end : int
act_start : int
act_lastc : int
save : char
acttext : char*
actin : FILE*
actout : FILE*
actlineno : int
actleng : int
act_scan()
act_scan()
actlex()
~act_scan()
actgetc()
actwrap()
acterror()
output()
act_FATAL()
ECHO()
input()
unput()
act_reset()
setinput()
setoutput()
NLSTATE()
act_INIT()
act_USER()
act_SCANNER()
actless()
actcomment()
actmapch()
act_parse
(from header)
actlval : actSTYPE
mustfree : int
size : int
reset : int
acti : short
actstate : short
stateStack: short*
valueStack: actSTYPE*
actps : short*
actpv : actSTYPE*
actval : actSTYPE
actpvt : actSTYPE*
actchar : int
acterrflag : int
actnerrs : int
scan : act_scan*
actdebug : int
act_parse()
act_parse()
actparse()
~act_parse()
actreset()
setdebug()
acterrok()
actclearin()
actRECOVERING()
duract_scan
(from header)
state : unsigned char*
size : int
mustfree : int
duract_end : int
duract_start : int
duract_lastc : int
save : char
duracttext : char*
duractin : FILE*
duractout : FILE*
duractlineno : int
duractleng : int
duract_scan()
duract_scan()
duractlex()
~duract_scan()
duractgetc()
duractwrap()
duracterror()
output()
duract_FATAL()
ECHO()
input()
unput()
duract_reset()
setinput()
setoutput()
NLSTATE()
duract_INIT()
duract_USER()
duract_SCANNER()
duractless()
duractcomment()
duractmapch()
deb_scan
(from header)
state : unsigned int*
size : int
mustfree : int
deb_end : int
deb_start : int
deb_lastc : int
save : char
debtext : char*
debin : FILE*
debout : FILE*
deblineno : int
debleng : int
deb_scan()
deb_scan()
deblex()
~deb_scan()
debgetc()
debwrap()
deberror()
output()
deb_FATAL()
ECHO()
input()
unput()
deb_reset()
setinput()
setoutput()
NLSTATE()
deb_INIT()
deb_USER()
deb_SCANNER()
debless()
debcomment()
debmapch()
DebParser
(from header)
DebParser()
~DebParser()
ActParser
(from header)
ActParser()
~ActParser()
duract_parse
(from header)
duractlval : duractSTYPE
mustfree : int
size : int
reset : int
duracti : short
duractstate : short
stateStack: short*
valueStack: duractSTYPE*
duractps : short*
duractpv : duractSTYPE*
duractval : duractSTYPE
duractpvt : duractSTYPE*
duractchar : int
duracterrflag : int
duractnerrs : int
scan : duract_scan*
duractdebug : int
duract_parse()
duract_parse()
duractparse()
~duract_parse()
duractreset()
setdebug()
duracterrok()
duractclearin()
duractRECOVERING()
ParserDomain
(from header)
parserdebut : DebParser*
scannerdebut : deb_scan*
parseraction : ActParser*
scanneraction : act_scan*
parserduract : DuractParser*
scannerduract : duract_scan*
ParserDomain()
analyse()
~ParserDomain()
DuractParser
(from header)
DuractParser()
~DuractParser()
+scan
0..1
1
-scanneraction
-parseraction
-parserdebut
-scannerdebut
-parserduract
-scannerduract

La classe ParserDomain

Cette classe permet d'analyser le fichier domain.pddl. Il parce ce dernier et stocke les informations dans la classe DonneesDom. Elle renvoie une exception si elle rencontre un problème durant l'analyse (en précisant la ligne ainsi que la partie début, action ou durative-action)

analyse(const char *nomfic)	**:**	Lance l'analyse du fichier dont le nom est passé en paramètre.

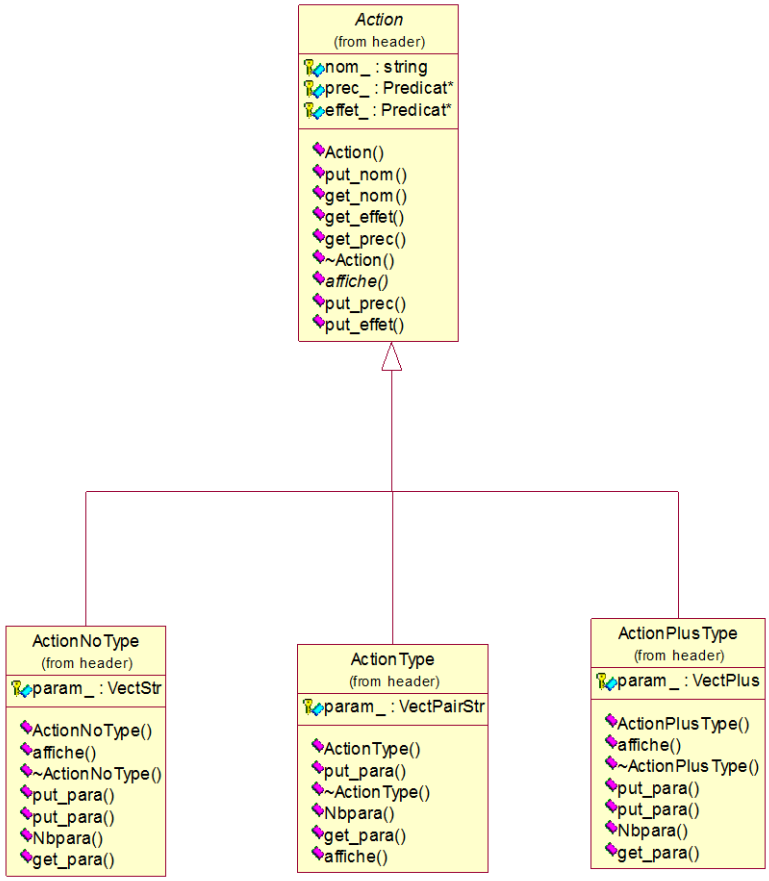
Les autres classes sont des classes générées par Lex et Yacc.



La classe ParserProblem

Cette classe permet d'analyser le fichier PDDL dans lequel est contenu les informations du problème. Il parce ce dernier et stocke les informations dans la classe DonneesPb. Elle renvoie une exception si elle rencontre un problème durant l'analyse (en précisant la ligne)

Analyse(const char *nomfic)	**:**	Lance l'analyse du fichier dont le nom est passé en paramètre.



La classe Action :

c'est une classe virtuelle pure .

string& get_nom()	**:**	permet d'obtenir le nom de l'action
Predicat* get_prec()	**:**	permet d'obtenir le prédicat en précondition

Predicat* get_effet()	**:**	permet d'obtenir le prédicat en effet

La classe ActionNoType :

Elle dérive de la classe Action. Elle permet de stocker une action dans un problème où les données sont non typées.

string& get_para(i)	**:**	permet d'obtenir son ième paramètre par la méthode
int get_para()	**:**	permet d'obtenir le nombre de paramètre par la méthode

La classe ActionType

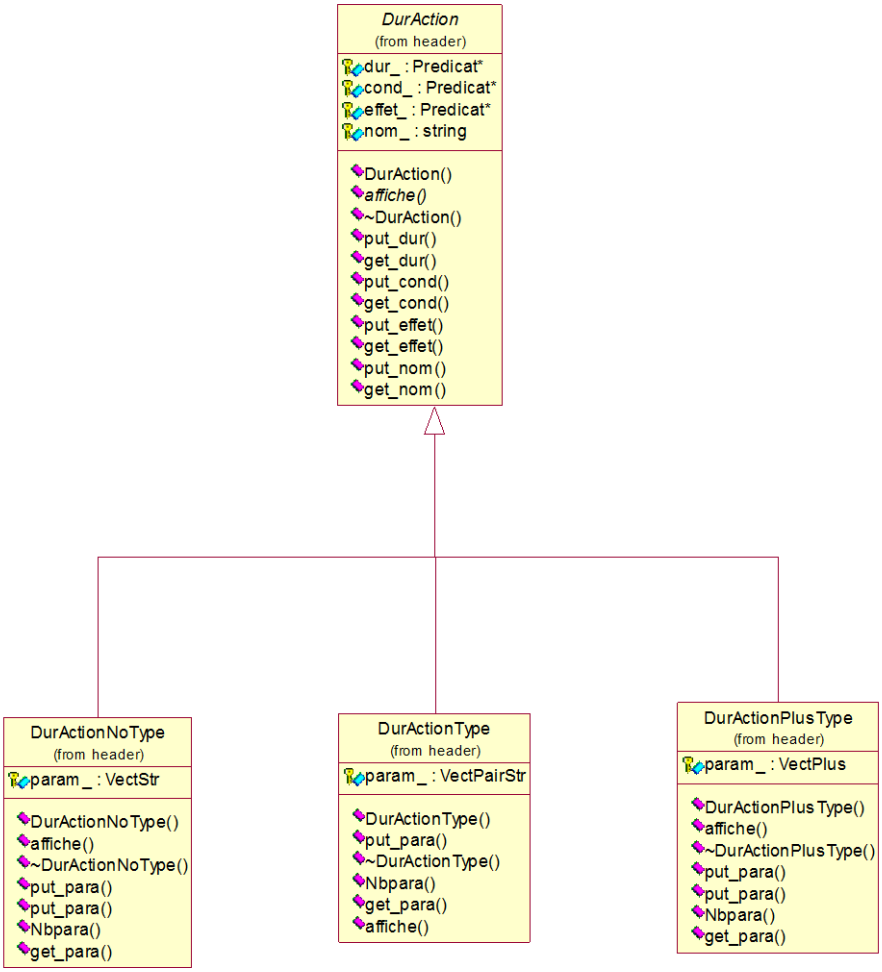
Elle dérive de la classe Action. Elle permet de stocker une action dans un problème typé. Donc chaque paramètre est constitué de pair, le premier élément correspond à la variable, le deuxième au type.

string& get_nom()	**:**	permet d'obtenir le nom de l'action
Pair<string,string>* get_para(i)	**:**	permet d'obtenir son ième paramètre par la méthode
int get_para()	**:**	permet d'obtenir le nombre de paramètre par la méthode
Predicat* get_prec()	**:**	permet d'obtenir le prédicat en précondition
Predicat* get_effet()	**:**	permet d'obtenir le prédicat en effet

La classe ActionPlusType

Elle dérive de la classe Action. Elle permet de stocker une action dans un problème typé. Donc chaque paramètre est constitué de pair, le premier élément correspond à la variable, le deuxième au type. A la différence de la précédente, le type est un vecteur de chaîne de caractère, permettant ainsi de stocker des actions où un des paramètres peut prendre plusieurs types.

Pair<string, vector<string> > * get_para(i)	**:**	permet d'obtenir son ième paramètre par la méthode
int get_para()	**:**	permet d'obtenir le nombre de paramètre par la méthode



La classe DurAction :

C'est une classe virtuelle pure. Elle sert de classe de base pour toutes les classes servant à stocker des actions avec durée.

string& get_nom()	**:**	permet d'obtenir le nom de l'action
Prediact* get_dur()	**:**	Permet d'obtenir le prédicat de la contrainte de durée de cet action

Predicat* get_cond()	**:**	permet d'obtenir le prédicat en précondition
Predicat* get_effet()	**:**	permet d'obtenir le prédicat en effet

La classe DurActionNoType :

Elle dérive de la classe Action. Elle permet de stocker une action dans un problème où les données sont non typées.

string& get_para(i)	**:**	permet d'obtenir son ième paramètre par la méthode
int get_para()	**:**	permet d'obtenir le nombre de paramètre par la méthode

La classe DurActionType

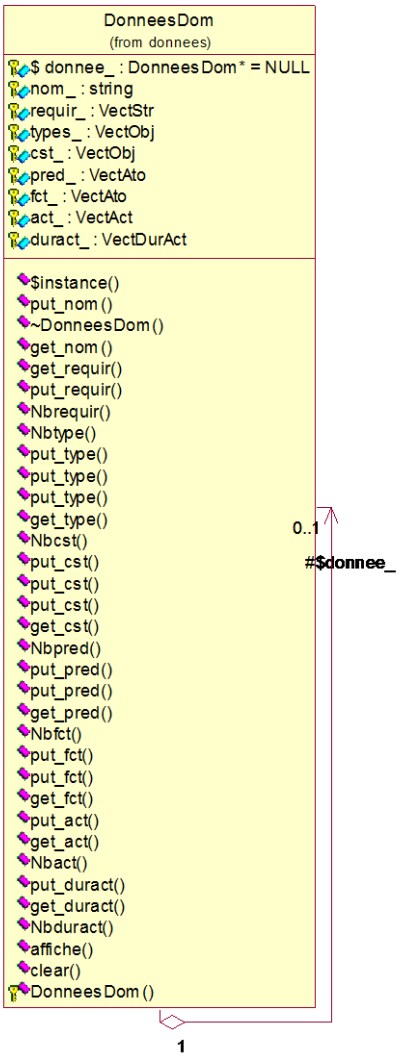
Elle dérive de la classe Action. Elle permet de stocker une action dans un problème typé. Donc chaque paramètre est constitué de pair, le premier élément correspond à la variable, le deuxième au type.

Pair<string,string>* get_para(i)	**:**	permet d'obtenir son ième paramètre par la méthode
int get_para()	**:**	permet d'obtenir le nombre de paramètre par la méthode

La classe DurActionPlusType

Elle dérive de la classe Action. Elle permet de stocker une action dans un problème typé. Donc chaque paramètre est constitué de pair, le premier élément correspond à la variable, le deuxième au type. A la différence de la précédente, le type est un vecteur de chaîne de caractère, permettant ainsi de stocker des actions où un des paramètres peut prendre plusieurs types.

Pair<string, vector<string> > * get_para(i)	**:**	permet d'obtenir son ième paramètre par la méthode
int get_para()	**:**	permet d'obtenir le nombre de paramètre par la méthode



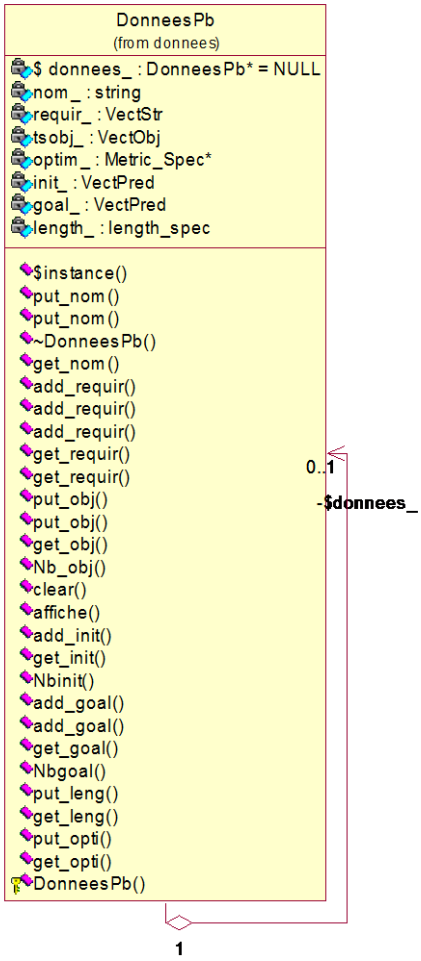
La classe DonneesDom

C'est une classe singleton. Le pattern Singleton, du modèle de création assure qu'une classe ne peut être instanciée qu'une seule fois et que sa portée est globale.

Static DonneesDom* DonneesDom : :instance()	**:**	Permet d'obtenir l'instance.

String& get_nom()	**:**	permet d'obtenir le nom du domaine.
String& get_requir(i)	**:**	permet d'obtenir son ième la caractéristique du domaine (strips, typing…).
int Nbrequir()	**:**	permet d'obtenir le nombre de caractéristique.
int Nbtype()	**:**	Permet d'obtenir le nombre de type.
ObjNoType* get_type(int ind)	**:**	Permet d'obtenir le ième type du problème.
int Nbcst()	**:**	permet d'obtenir le nombre de constante du problème.
ObjNoType* get_cst(int ind)	**:**	Permet d'obtenir la iéme constante du problème.
int Nbpred()	**:**	Permet d'obtenir le nombre de prédicat du problème
Predicat* get_pred(int ind)	**:**	Permet d'obtenir le iéme prédicat du problème.
int Nbfct()	**:**	Permet d'obtenir le nombre de fonction
Atomic* get_fct(int ind)	**:**	Permet d'obtenir la iéme fonction du problème.
Action* get_act(int ind)	**:**	Permet d'obtenir la ième action
int Nbact(void)	**:**	Permet d'obtenir le nombre d'action
DurAction* get_duract(int ind)	**:**	Permet d'obtenir la ième action de durée
int Nbduract(void)	**:**	Permet d'obtenir le nombre d'action avec durée

Void affiche()	**:**	Permet d'afficher toutes les données stockées
Void clear()	**:**	Permet d'effacer les données stockées

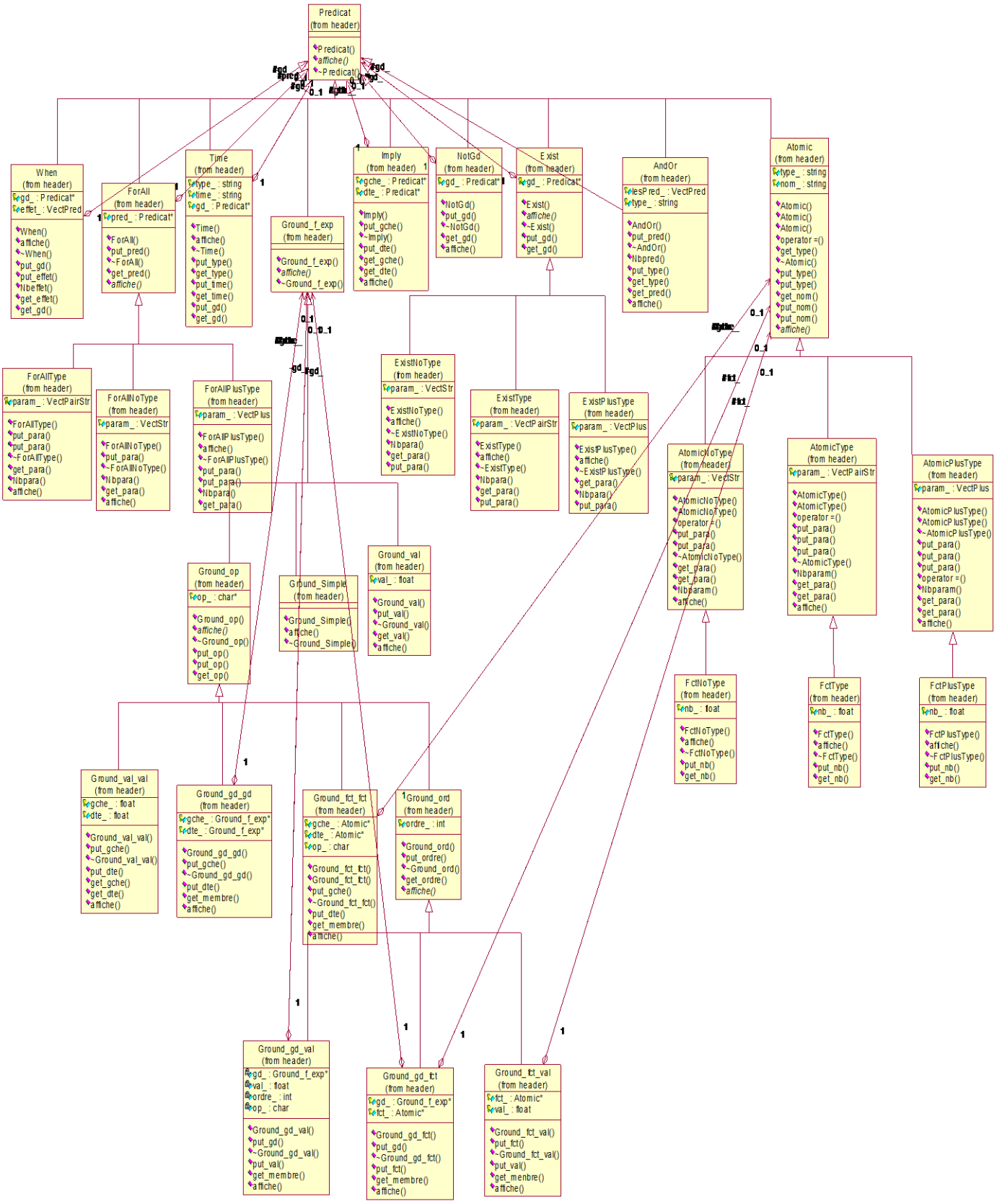


La classe DonneesPb

C'est une classe singleton. Le pattern Singleton, du modèle de création assure qu'une classe ne peut être instanciée qu'une seule fois et que sa portée est globale.

Static DonneesPb* ***DonneesPb : :instance()***	**:**	Permet d'obtenir l'instance.
String& get_nom()	**:**	permet d'obtenir le nom du problème.

Vector<string>* get_requir(i)	**:**	Permet d'obtenir les caractéristiques du problème
int Nb_obj ()	**:**	Permet d'obtenir le nombre d'objet.
int Nbtype()	**:**	Permet d'obtenir le nombre de type.
ObjNoType* get_obj(int ind)	**:**	Permet d'obtenir le ième objet du problème.
int Nbinit ()	**:**	permet d'obtenir le nombre de atome dans l'état initiale du problème.
Predicat* get_init(int ind)	**:**	Permet d'obtenir la iéme atome en condition initiale.
int Nbgoal ()	**:**	Permet d'obtenir le nombre d'atome en en condition final du problème
Predicat* get_goal(int ind)	**:**	Permet d'obtenir le iéme atome en condition final.
length_spec* get_leng()	**:**	Permet d'obtenir la longueur du problème lorsqu'elle est spécifié dans le fichier PDDL.
Metric_Spec* get_opti()	**:**	Permet d'obtenir la fonction à optimiser du problème.
void affiche()	**:**	Permet d'afficher toutes les données stockées
Void clear()	**:**	Permet d'effacer les données stockées



La classe Predicat :

C'est une classe virtuelle pure. C'est la classe à partir de laquelle tous les autres prédicats dérivent.

Void affiche( )	**:**	Permet d'afficher le prédicat.

La classe When :

Cette classe dérive de la classe Predicat, elle permet de stocker le prédicat :

When « prédicat » alors « liste de prédicat »

int Nbeffet(()	**:**	permet d'obtenir le nombre de prédicat en effet.
Predicat* get_effet(int ind)	**:**	Permet d'obtenir le ième prédicat en effet.
Predicat* get_gd()	**:**	permet d'obtenir le prédicat en précondition.

La classe ForAll

Cette classe dérive de Prédicat. C'est une classe virtuelle qui sert de classe de base à toutes les classes permettant de stocker un prédicat du type ForAll « liste de variable », « prédicat »

Predicat* get_pred()	**:**	permet d'obtenir le prédicat.

La classe ForAllPlusType

Cette classe dérive de ForAll. Elle permet de stocker le prédicat ForAll dans un problème typé, ce prédicat est de la forme : ForAll « variable – type, variable – type, … », « prédicat ».

Les paramètres sont stockés sous forme de pair, le premier est le nom de la variable et le deuxiéme son type.

Void affiche( )	**:**	Permet d'afficher le prédicat.
PairStr get_para(int ind)	**:**	permet d'obtenir le ième paramètre.
int Nbpara()	**:**	permet d'obtenir le nombre de paramètre en effet.

La classe ForAllNoType

Cette classe dérive de ForAll. Elle permet de stocker le prédicat ForAll dans un problème typé, ce prédicat est de la forme : ForAll « variable, variable, … », « prédicat ».

Void affiche( )	**:**	Permet d'afficher le prédicat.
String get_para(int ind)	**:**	permet d'obtenir le ième paramètre.
int Nbpara()	**:**	permet d'obtenir le nombre de paramètre en effet.

La classe ForAllType

Cette classe dérive de ForAll. Elle permet de stocker le prédicat ForAll dans un problème typé, ce prédicat est de la forme : ForAll « variable – either (type, type, ..) , variable – type, … », « prédicat ».

Les paramètres sont stockés sous forme de pair, le premier est le nom de la variable et le deuxiéme est un vecteur contenant les noms des différents types de la variable.

Void affiche( )	**:**	Permet d'afficher le prédicat.
Pair<string, vector<string>>& get_para(int ind)	**:**	permet d'obtenir le ième paramètre.
int Nbpara()	**:**	permet d'obtenir le nombre de paramètre en effet.

La classe Time :

Cette classe dérive de Predicat. Elle sert à stocker tous les prédicats de temps.

Void affiche( )	**:**	Permet d'afficher le prédicat.
String& get_type()	**:**	permet d'obtenir le type du prédicat (over ou at).
String& get_time()	**:**	permet d'obtenir le quantificateur (start, end, all).
Predicat* get_gd()	**:**	Permet d'obtenir le predicat.

La classe NotGd

Cette classe dérive de la classe Predicat. Elle permet de stocker un prédicat *« not »* s'appliquant à un prédicat autre qu'un predicat atomic.

Predicat* get_gd()	**:**	Permet d'obtenir le prédicat.
Void affiche()	**:**	Permet d'afficher le prédicat.

La classe Imply

Cette classe dérive de la classe Prédicat. Elle permet de stocker un prédicat *« imply »* de la forme imply – prédicat – prédicat.

Predicat* get_gche()	**:**	Permet d'obtenir le prédicat en partie gauche de l'implication.
Predicat* get_dte()	**:**	Permet d'obtenir le prédicat en partie droite de l'implication.
void affiche()	**:**	Permet d'afficher le prédicat.

La classe AndOr

Cette classe dérive de Prédicat. Elle permet de stocker un prédicat *« and »* ou *« or »* de la forme and (or) – liste de prédicat.

string get_type()	**:**	Permet d'obtenir le type du prédicat i.e. si c'est and ou or.
int Nbpred()	**:**	Permet d'obtenir le nombre de prédicat.
Predicat* get_pred(int ind)	**:**	Permet d'obtenir le ième prédicat.
void affiche()	**:**	Permet d'afficher le prédicat.

La classe Exist

Cette classe dérive de la classe Predicat. C'est une classe virtuelle pure. Elle permet de stocker le prédicat *« exist »* de la forme : exist – variable – prédicat.

Predicat* get_pred()	**:**	Permet d'obtenir le prédicat.

La classe ExistNoType

Cette classe dérive de la classe Exist. Elle permet de stocker le prédicat « exist » dans le cas où les variables ne sont pas typées.

int Nbpara()	**:**	Permet d'obtenir le nombre de variable.
string& get_para(int ind)	**:**	Permet d'obtenir le ième paramètre.

La classe ExistType

Cette classe dérive de la classe Exist. Elle permet de stocker le prédicat « exist » dans le cas où les variables sont typées, de la forme : exist – variable – type variable – type … – prédicat. Les variables sont stockées sous forme de pair de chaine, le premier élément correspond au nom de la variable et le deuxième à son type.

int Nbpara()	**:**	Permet d'obtenir le nombre de variable.
pair<string,string>& get_para(int ind)	**:**	Permet d'obtenir le ième paramètre.

La classe ExistPlusType

Cette classe dérive de la classe Exist. Elle permet de stocker le prédicat « exist » dans le cas où les variables sont typées, et une des variables au moins peut prendre plusieurs types, de la forme : exist – variable – type variable – either (type, type, …) … – prédicat. Les variables sont stockées sous forme de pair, le premier élément correspond au nom de la variable et le deuxième à un vecteur où sont stockés les différents types.

int Nbpara()	**:**	Permet d'obtenir le nombre de variable.
pair<string, vector<string>> get_para(int ind)	**:**	Permet d'obtenir le ième paramètre.

La classe Atomic

Cette classe dérive de la classe Predicat. C'est une classe virtuelle. Elle permet de stocker un prédicat de la forme : (not) nom_du_prédicat variable – variable…

string get_type()	**:**	Permet d'obtenir le type du prédicat i.e. si c'est not ou atomic.
string get_nom()	**:**	Permet d'obtenir le nom du prédicat.

La classe AtomicNoType

Cette classe dérive de la classe Atomic. Elle permet de stocker un prédicat « atomic » dans le cas où les variables ne sont pas typées.

int Nbparam()	**:**	Permet d'obtenir le nombre de variable.
string get_para(int ind)	**:**	Permet d'obtenir le ième paramètre.

La classe AtomicType

Cette classe dérive de la classe Atomic. Elle permet de stocker le prédicat « atomic » dans le cas où les variables sont typées, de la forme : (not) nom_du_prédicat – variable – type variable – type … Les variables sont stockées sous forme de pair de chaine, le premier élément correspond au nom de la variable et le deuxième à son type.

int Nbpara()	**:**	Permet d'obtenir le nombre de variable.

pair<string,string>& get_para(int ind)	**:**	Permet d'obtenir le ième paramètre.

La classe AtomicPlusType

Cette classe dérive de la classe Atomic. Elle permet de stocker le prédicat « atomic » dans le cas où les variables sont typées, et une des variables au moins peut prendre plusieurs types, de la forme : (not) nom_du_prédicat – variable – type variable – either (type, type, …) … Les variables sont stockées sous forme de pair, le premier élément correspond au nom de la variable et le deuxième à un vecteur où sont stockés les différents types.

int Nbpara()	**:**	Permet d'obtenir le nombre de variable.
pair<string, vector<string>> get_para(int ind)	**:**	Permet d'obtenir le ième paramètre.

La Classe FctNoType

Cette classe dérive de la classe AtomicNoType. Elle permet de stocker une fonction définie avec un réel dans le cas où les variables ne sont pas typées.

float get_nb ()	**:**	Permet d'obtenir le nombre.

La Classe FctType

Cette classe dérive de la classe AtomicType. Elle permet de stocker une fonction définie avec un réel dans le cas où les variables sont typées.

float get_nb ()	**:**	Permet d'obtenir le nombre.

La Classe FctPlusType

Cette classe dérive de la classe AtomicplusType. Elle permet de stocker une fonction définie avec un réel dans le cas où les variables sont typées et dans au moins une est définie avec plusieurs types.

Float get_nb ()	**:**	Permet d'obtenir le nombre.

La classe Ground_f_exp

Cette classe dérive de la classe Predicat. C'est une classe virtuelle. Elle sert simplement de classe de base.

La classe Ground_Simple

Cette classe dérive de la classe Ground_f_exp. Elle sert à représenter la …

La classe Ground_val.

Cette classe dérive de la classe Ground_f_exp. Elle permet de stocker simplement une valeur.

float get_val()	**:**	Permet d'obtenir la valeur.

La classe Ground_op

Cette classe dérive de la classe Ground_f_exp. C'est une classe virtuelle qui sert de base à tous les prédicat basés sur un opérateur.

char* get_op()	**:**	Permet d'obtenir l'opérateur.

La classe Ground_val_val

Cette classe dérive de la classe Ground_op. Elle est utilisée pour stocker une fonction du type : Opérateur – nombre – nombre .

float get_gche ()	**:**	Permet d'obtenir la valeur gauche.
float get_dte()	**:**	Permet d'obtenir la valeur droite.

La classe Ground_gd_gd

Cette classe dérive de la classe Ground_op. Elle est utilisée pour stocker une fonction du type : Opérateur – Ground_f_exp – Ground_f_exp.

void get_membre(Ground_f_exp *gche, Ground_f_exp *dte)	**:**	Permet d'obtenir les membres de la fonction. Le premier paramètre désigne la partie gauche, le deuxième la partie droite.

La classe Gound_fct_fct

Cette classe dérive de la classe Ground_op. Elle est utilisée pour stocker une fonction du type : Opérateur – Atomic – Atomic.

void get_membre(Atomic *gche, Atomic *dte)	**:**	Permet d'obtenir les membres de la fonction. Le premier paramètre désigne la partie gauche, le deuxième la partie droite.

La classe Ground_ord

Cette classe dérive de la classe Ground_f_exp. C'est une classe virtuelle qui sert de base à tous les prédicat de type Ground_op n'étant pas symétrique et nécessitant un ordre.

int get_ordre()	**:**	Permet d'obtenir l'ordre défini dans chaque sous classe.

La classe Ground_gd_val

Cette classe dérive de la classe Ground_ord. Elle est utilisée pour stocker une fonction du type :

Opérateur – valeur – Ground_f_exp ou Opérateur – Ground_f_exp – valeur.

void get_menbre(Ground_f_exp *fct, float val, int ordre)	**:**	Permet d'obtenir les membres de la fonction. Ordre est à 0 si Ground_f_exp correspond à la partie gauche et un sinon.

La classe Ground_gd_fct

Cette classe dérive de la classe Ground_ord. Elle est utilisée pour stocker une fonction du type :

Opérateur – Atomic – Ground_f_exp ou Opérateur – Ground_f_exp – Atomic.

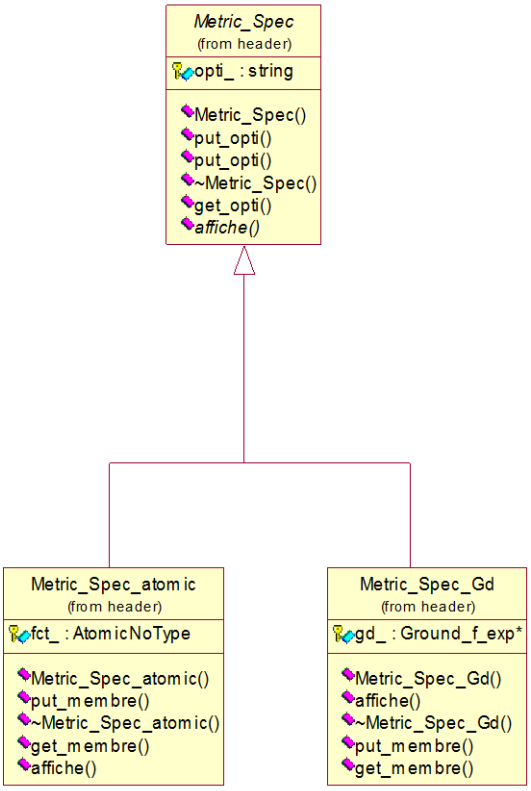
void get_membre(Ground_f_exp *gd, Atomic *fct, int ordre)	**:**	Permet d'obtenir les membres de la fonction. Ordre est à 0 si Atomic correspond à la partie gauche et un sinon.

La classe Ground_fct_val

Cette classe dérive de la classe Ground_ord. Elle est utilisée pour stocker une fonction du type :

Opérateur – valeur – Atomic ou Opérateur – Atomic – valeur.

void get_menbre(Atomic *fct, float val, int ordre)	**:**	Permet d'obtenir les membres de la fonction. Ordre est à 0 si Atomic correspond à la partie gauche et un sinon.



La classe Metric_Spec

Cette classe permet de stocker une fonction à optimiser. C'est une classe virtuelle. Elle sert de classe de base.

String& get_opti()	**:**	Renvoie soit maximise soit minimise

La classe Metric_Spec_Gd

Cette classe dérive de la classe Metric_Spec. Elle permet de stocker une fonction à optimiser de la forme :

Maximise (minimise) nom_de_la_fonction varaible variable…

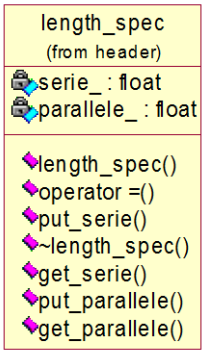
AtomicNoType* get_membre()	**:**	Renvoie la fonction à optimiser.

La classe Metric_Spec_atomic

Cette classe dérive de la classe Metric_Spec. Elle permet de stocker une fonction à optimiser de la forme :

Maximise (minimise) Ground_f_exp…

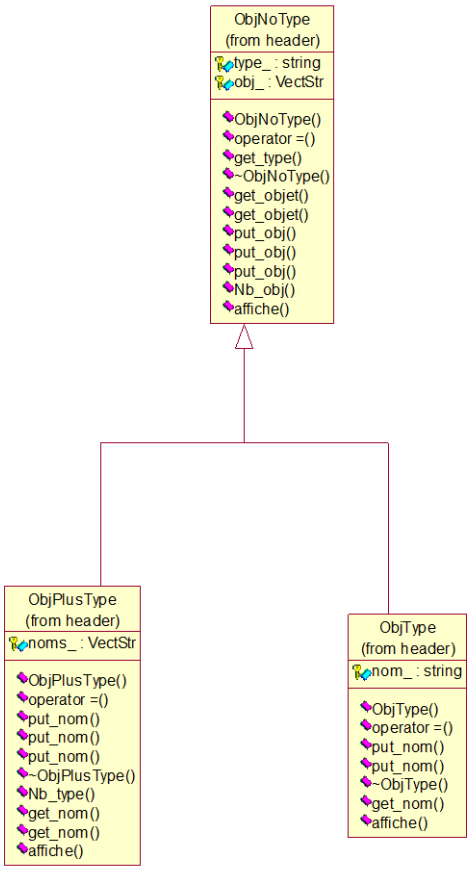
Ground_f_exp* get_membre()	**:**	Renvoie la fonction à optimiser.



La classe length_spec

Cette classe permet de stocker le nombre d'action en parallèle et le la longueur du problème.

float get_serie()	**:**	Renvoie la longueur du problème.
float get_parallele()	**:**	Renvoie le nombre d'action permis sur un pas de temps.



La classe ObjNoType

Cette classe permet de stocker les objets d'un problème non typé, juste une suite de nom.

int Nb_obj()	**:**	Renvoie le nombre d'objet stocké.
string get_objet(int ind)	**:**	Renvoie le ième objet.

La classe ObjPlusType

Cette classe dérive de la classe ObjNoType. Cette classe permet de stocker les objets d'un problème typé, juste une suite de nom suivi d'un type.

string get_nom()	**:**	Renvoie le type de l'objet.

La classe ObjType

Cette classe dérive de la classe ObjNoType. Cette classe permet de stocker les objets d'un problème typé, juste une suite de nom suivi de plusieurs noms de la forme :

obj1, obj2, … - either (type1, type2, ...)

int Nb_type()	**:**	Renvoie le nombre de type.
string get_nom(int ind)	**:**	Renvoie le ième type.

ANNEXE B : Fichiers PDDL des exemples traités dans la partie résultats du rapport

1) le domaine « bw »

1.1) le fichier domain.pddl

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Prodigy blocks world
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain prodigy-bw)
  (:requirements :strips :typing)
  (:predicates (on ?x - block ?y - block)
               (on-table ?x - block)
               (clear ?x - block)
               (arm-empty)
               (holding ?x - block)
              )
  (:action pick-up
           :parameters (?ob1 - block)
           :precondition (and (clear ?ob1) (on-table ?ob1) (arm-empty))
           :effect
           (and (not (on-table ?ob1))
                (not (clear ?ob1))
                (not (arm-empty))
                (holding ?ob1)))
  (:action put-down
           :parameters (?ob - block)
           :precondition (holding ?ob)
           :effect
           (and (not (holding ?ob))
                (clear ?ob)
                (arm-empty)
                (on-table ?ob)))
  (:action stack
           :parameters (?sob - block ?sunderob - block)
           :precondition (and (holding ?sob) (clear ?sunderob))
           :effect
           (and (not (holding ?sob))
                (not (clear ?sunderob))
                (clear ?sob)
                (arm-empty)
                (on ?sob ?sunderob)))
  (:action unstack
           :parameters (?sob - block ?sunderob - block)
           :precondition (and (on ?sob ?sunderob) (clear ?sob) (arm-empty))
           :effect
           (and (holding ?sob)
                (clear ?sunderob)
                (not (clear ?sob))
                (not (arm-empty))
                (not (on ?sob ?sunderob)))))
```

1.2) le fichier probBLOCKS-4-0.pddl

```
(define (problem BLOCKS-4-0)
(:domain BLOCKS)
(:objects D B A C - block)
(:INIT (clear C) (clear A) (clear B) (clear D) (on-table C) (on-table A)
 (on-table B) (on-table D) (arm-empty))
(:goal (and (on D C) (on C B) (on B A)))
)
```

résultat obtenu :

```
/*********************************/
            etat initial
/*********************************/
on-table ( D )
on-table ( B )
on-table ( A )
on-table ( C )
clear ( D )
clear ( B )
clear ( A )
clear ( C )
arm-empty (  )
action déroulée au tps 0
pick-up ( B )
état du problème au tps 1
on-table ( D )
on-table ( C )
clear ( D )
clear ( A )
clear ( C )
holding ( B )

action déroulée au tps 1
stack ( B ; A )
état du problème au tps 2
on ( B ; A )
on-table ( D )
on-table ( C )
clear ( D )
clear ( B )
clear ( C )
arm-empty (  )

action déroulée au tps 2
pick-up ( C )
état du problème au tps 3
on ( B ; A )
on-table ( D )
clear ( D )
clear ( B )
holding ( C )

action déroulée au tps 3
stack ( C ; B )
état du problème au tps 4
on ( B ; A )
on ( C ; B )
```

```
on-table ( D )
clear ( D )
clear ( C )
arm-empty (  )

action déroulée au tps 4
pick-up ( D )
état du problème au tps 5
on ( B ; A )
on ( C ; B )
clear ( C )
holding ( D )

action déroulée au tps 5
stack ( D ; C )
/*********************************/
            etat final
/*********************************/
on ( D ; C )
on ( B ; A )
on ( C ; B )
clear ( D )
arm-empty ( )
```

1.4) le fichier bw-12step.pddl

```
(define (problem bw-12step)           ; graphplan 12 steps
  ;; Relatively hard. Requires 12 steps and takes graphplan 20 seconds.
  ;; (unstack c d) (put-down c) (unstack d e) (put-down d)
  ;; (unstack e f) (put-down e) (unstack f g) (stack f a) (unstack c b)
  ;; (stack c d) (pick-up b) (stack b c)
    (:domain prodigy-bw)
  (:objects A B C D E F G - block)
  (:init (on-table A) (clear A)
       (on-table B) (clear B)
       (on-table G)
       (on F G)
       (on E F)
       (on D E)
       (on C D) (clear C)
       (arm-empty))
  (:goal (and (on B C) (on-table A) (on F A) (on C D)))
  )
```

résultat obtenu :

```
/*********************************/
            etat initial
/*********************************/
on ( C ; D )
on ( D ; E )
on ( E ; F )
on ( F ; G )
on-table ( A )
on-table ( B )
on-table ( G )
clear ( A )
clear ( B )
```

```
clear ( C )
arm-empty (  )
action déroulée au tps 0
unstack ( C ; D )
état du problème au tps 1
on ( D ; E )
on ( E ; F )
on ( F ; G )
on-table ( A )
on-table ( B )
on-table ( G )
clear ( A )
clear ( B )
clear ( D )
holding ( C )

action déroulée au tps 1
put-down ( C )
état du problème au tps 2
on ( D ; E )
on ( E ; F )
on ( F ; G )
on-table ( A )
on-table ( B )
on-table ( C )
on-table ( G )
clear ( A )
clear ( B )
clear ( C )
clear ( D )
arm-empty (  )

action déroulée au tps 2
unstack ( D ; E )
état du problème au tps 3
on ( E ; F )
on ( F ; G )
on-table ( A )
on-table ( B )
on-table ( C )
on-table ( G )
clear ( A )
clear ( B )
clear ( C )
clear ( E )
holding ( D )

action déroulée au tps 3
put-down ( D )
état du problème au tps 4
on ( E ; F )
on ( F ; G )
on-table ( A )
on-table ( B )
on-table ( C )
on-table ( D )
on-table ( G )
clear ( A )
clear ( B )
clear ( C )
```

```
clear ( D )
clear ( E )
arm-empty (  )

action déroulée au tps 4
pick-up ( C )
état du problème au tps 5
on ( E ; F )
on ( F ; G )
on-table ( A )
on-table ( B )
clear ( A )
clear ( B )
clear ( D )
clear ( E )
holding ( C )

action déroulée au tps 5
stack ( C ; D )
état du problème au tps 6
on ( C ; D )
on ( E ; F )
on ( F ; G )
on-table ( A )
on-table ( B )
clear ( A )
clear ( B )
clear ( C )
clear ( E )
arm-empty (  )

action déroulée au tps 6
pick-up ( B )
état du problème au tps 7
on ( C ; D )
on ( E ; F )
on ( F ; G )
on-table ( A )
clear ( A )
clear ( C )
clear ( E )
holding ( B )

action déroulée au tps 7
stack ( B ; C )
état du problème au tps 8
on ( B ; C )
on ( C ; D )
on ( E ; F )
on ( F ; G )
on-table ( A )
clear ( A )
clear ( B )
clear ( E )
arm-empty (  )

action déroulée au tps 8
unstack ( E ; F )
état du problème au tps 9
on ( B ; C )
```

```
on ( C ; D )
on ( F ; G )
on-table ( A )
clear ( A )
clear ( B )
clear ( F )
holding ( E )

action déroulée au tps 9
stack ( E ; B )
état du problème au tps 10
on ( B ; C )
on ( C ; D )
on ( E ; B )
on ( F ; G )
on-table ( A )
clear ( A )
clear ( E )
clear ( F )
arm-empty (  )

action déroulée au tps 10
unstack ( F ; G )
état du problème au tps 11
on ( B ; C )
on ( C ; D )
on-table ( A )
clear ( A )
clear ( G )
holding ( F )

action déroulée au tps 11
stack ( F ; A )
/*********************************/
           etat final
/*********************************/
on ( B ; C )
on ( C ; D )
on ( F ; A )
on-table ( A )
clear ( F )
arm-empty ( )
```

1.5) le fichier bw-large-a.pddl

```
;;; bw-large-a
;;;
;;; Initial:  3/2/1   5/4   9/8/7/6
;;; Goal:     1/5     8/9/4/    2/3/7/6
;;; Length: 12

(define (problem bw-large-a)
  (:domain prodigy-bw)
  (:objects 1 2 3 4 5 6 7 8 9 - block)
  (:init (arm-empty)
       (on 3 2)
       (on 2 1)
       (on-table 1)
```

```
      (on 5 4)
      (on-table 4)
      (on 9 8)
      (on 8 7)
      (on 7 6)
      (on-table 6)
      (clear 3)
      (clear 5)
      (clear 9))
  (:goal (and
       (on 1 5)
       (on-table 5)
       (on 8 9)
       (on 9 4)
       (on-table 4)
       (on 2 3)
       (on 3 7)
       (on 7 6)
       (on-table 6)
       (clear 1)
       (clear 8)
       (clear 2)
       )))
```

résultat obtenu :

```
/*********************************/
           etat initial
/*********************************/
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 4 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on ( 9 ; 8 )

on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
clear ( 3 )
clear ( 5 )
clear ( 9 )
arm-empty (  )
action déroulée au tps 0
unstack ( 5 ; 4 )
état du problème au tps 1
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on ( 9 ; 8 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
clear ( 3 )
clear ( 4 )
clear ( 9 )
holding ( 5 )

action déroulée au tps 1
```

```
put-down ( 5 )
état du problème au tps 2
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on ( 9 ; 8 )
on-table ( 1 )
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 3 )
clear ( 4 )
clear ( 5 )
clear ( 9 )
arm-empty (  )

action déroulée au tps 2
unstack ( 9 ; 8 )
état du problème au tps 3
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on-table ( 1 )
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 3 )
clear ( 4 )
clear ( 5 )
clear ( 8 )
holding ( 9 )

action déroulée au tps 3
stack ( 9 ; 4 )
état du problème au tps 4
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on ( 9 ; 4 )
on-table ( 1 )
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 3 )
clear ( 5 )
clear ( 8 )
clear ( 9 )
arm-empty (  )

action déroulée au tps 4
unstack ( 8 ; 7 )
état du problème au tps 5
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 7 ; 6 )
on ( 9 ; 4 )
on-table ( 1 )
```

```
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 3 )
clear ( 5 )
clear ( 7 )
clear ( 9 )
holding ( 8 )

action déroulée au tps 5
stack ( 8 ; 9 )
état du problème au tps 6
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on-table ( 1 )
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 3 )
clear ( 5 )
clear ( 7 )
clear ( 8 )
arm-empty (  )

action déroulée au tps 6
unstack ( 3 ; 2 )
état du problème au tps 7
on ( 2 ; 1 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on-table ( 1 )
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 2 )
clear ( 5 )
clear ( 7 )
clear ( 8 )
holding ( 3 )

action déroulée au tps 7
stack ( 3 ; 7 )
état du problème au tps 8
on ( 2 ; 1 )
on ( 3 ; 7 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on-table ( 1 )
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 2 )
clear ( 3 )
clear ( 5 )
clear ( 8 )
```

```
arm-empty (  )

action déroulée au tps 8
unstack ( 2 ; 1 )
état du problème au tps 9
on ( 3 ; 7 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on-table ( 1 )
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 1 )
clear ( 3 )
clear ( 5 )
clear ( 8 )
holding ( 2 )

action déroulée au tps 9
stack ( 2 ; 3 )
état du problème au tps 10
on ( 2 ; 3 )
on ( 3 ; 7 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on-table ( 1 )
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 1 )
clear ( 2 )
clear ( 5 )
clear ( 8 )
arm-empty (  )

action déroulée au tps 10
pick-up ( 1 )
état du problème au tps 11
on ( 2 ; 3 )
on ( 3 ; 7 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 2 )
clear ( 5 )
clear ( 8 )
holding ( 1 )

action déroulée au tps 11
stack ( 1 ; 5 )
/*********************************/
            etat final
/*********************************/
on ( 1 ; 5 )
on ( 2 ; 3 )
```

```
on ( 3 ; 7 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on-table ( 4 )
on-table ( 5 )
on-table ( 6 )
clear ( 1 )
clear ( 2 )
clear ( 8 )
```
arm-empty ( )

1.6) le fichier bw-large-b.pddl

```
;;; bw-large-b
;;;
;;; Initial:  3/2/1   11/10/5/4/   9/8/7/6
;;; Goal:     1/5/10/  8/9/4/   2/3/11/7/6
;;; Length:   18

(define (problem bw-large-b)
  (:domain prodigy-bw)
  (:objects 1 2 3 4 5 6 7 8 9 10 11 - block)
  (:init (arm-empty)
       (on 3 2)
       (on 2 1)
       (on-table 1)
       (on 11 10)
       (on 10 5)
       (on 5 4)
       (on-table 4)
       (on 9 8)
       (on 8 7)
       (on 7 6)
       (on-table 6)
       (clear 3)
       (clear 11)
       (clear 9))
  (:goal (and
        (on 1 5)
        (on 5 10)
        (on-table 10)
        (on 8 9)
        (on 9 4)
        (on-table 4)
        (on 2 3)
        (on 3 11)
        (on 11 7)
        (on 7 6)
        (on-table 6)
        (clear 1)
        (clear 8)
        (clear 2)
        )))
```

résultat obtenu :

```
/*********************************/
            etat initial
/*********************************/
```

```
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 4 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on ( 9 ; 8 )
on ( 10 ; 5 )
on ( 11 ; 10 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
clear ( 3 )
clear ( 9 )
clear ( 11 )
arm-empty (  )
action déroulée au tps 0
unstack ( 11 ; 10 )
état du problème au tps 1
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 4 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on ( 9 ; 8 )
on ( 10 ; 5 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
clear ( 3 )
clear ( 9 )
clear ( 10 )
holding ( 11 )

action déroulée au tps 1
put-down ( 11 )
état du problème au tps 2
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 4 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on ( 9 ; 8 )
on ( 10 ; 5 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 11 )
clear ( 3 )
clear ( 9 )
clear ( 10 )
clear ( 11 )
arm-empty (  )

action déroulée au tps 2
unstack ( 10 ; 5 )
état du problème au tps 3
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 4 )
on ( 7 ; 6 )
```

```
on ( 8 ; 7 )
on ( 9 ; 8 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 11 )
clear ( 3 )
clear ( 5 )
clear ( 9 )
clear ( 11 )
holding ( 10 )

action déroulée au tps 3
put-down ( 10 )
état du problème au tps 4
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 4 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on ( 9 ; 8 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
on-table ( 11 )
clear ( 3 )
clear ( 5 )
clear ( 9 )
clear ( 10 )
clear ( 11 )
arm-empty (  )

action déroulée au tps 4
unstack ( 5 ; 4 )
état du problème au tps 5
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on ( 9 ; 8 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
on-table ( 11 )
clear ( 3 )
clear ( 4 )
clear ( 9 )
clear ( 10 )
clear ( 11 )
holding ( 5 )

action déroulée au tps 5
stack ( 5 ; 10 )
état du problème au tps 6
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 10 )
on ( 7 ; 6 )
```

```
on ( 8 ; 7 )
on ( 9 ; 8 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
on-table ( 11 )
clear ( 3 )
clear ( 4 )
clear ( 5 )
clear ( 9 )
clear ( 11 )
arm-empty (  )

action déroulée au tps 6
unstack ( 9 ; 8 )
état du problème au tps 7
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
on-table ( 11 )
clear ( 3 )
clear ( 4 )
clear ( 5 )
clear ( 8 )
clear ( 11 )
holding ( 9 )

action déroulée au tps 7
stack ( 9 ; 4 )
état du problème au tps 8
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 8 ; 7 )
on ( 9 ; 4 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
on-table ( 11 )
clear ( 3 )
clear ( 5 )
clear ( 8 )
clear ( 9 )
clear ( 11 )
arm-empty (  )

action déroulée au tps 8
unstack ( 8 ; 7 )
état du problème au tps 9
on ( 2 ; 1 )
on ( 3 ; 2 )
```

```
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 9 ; 4 )

on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
on-table ( 11 )
clear ( 3 )
clear ( 5 )
clear ( 7 )
clear ( 9 )
clear ( 11 )
holding ( 8 )

action déroulée au tps 9
stack ( 8 ; 9 )
état du problème au tps 10
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
on-table ( 11 )
clear ( 3 )
clear ( 5 )
clear ( 7 )
clear ( 8 )
clear ( 11 )
arm-empty (  )

action déroulée au tps 10
pick-up ( 11 )
état du problème au tps 11
on ( 2 ; 1 )
on ( 3 ; 2 )
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
clear ( 3 )
clear ( 5 )
clear ( 7 )
clear ( 8 )
holding ( 11 )

action déroulée au tps 11
stack ( 11 ; 7 )
état du problème au tps 12
on ( 2 ; 1 )
```

```
on ( 3 ; 2 )
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on ( 11 ; 7 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
clear ( 3 )
clear ( 5 )
clear ( 8 )
clear ( 11 )
arm-empty (  )

action déroulée au tps 12
unstack ( 3 ; 2 )
état du problème au tps 13
on ( 2 ; 1 )
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on ( 11 ; 7 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
clear ( 2 )
clear ( 5 )
clear ( 8 )
clear ( 11 )
holding ( 3 )

action déroulée au tps 13
stack ( 3 ; 11 )
état du problème au tps 14
on ( 2 ; 1 )
on ( 3 ; 11 )
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on ( 11 ; 7 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
clear ( 2 )
clear ( 3 )
clear ( 5 )
clear ( 8 )
arm-empty (  )

action déroulée au tps 14
unstack ( 2 ; 1 )
état du problème au tps 15
on ( 3 ; 11 )
on ( 5 ; 10 )
```

```
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on ( 11 ; 7 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
clear ( 1 )
clear ( 3 )
clear ( 5 )
clear ( 8 )
holding ( 2 )

action déroulée au tps 15
stack ( 2 ; 3 )
état du problème au tps 16
on ( 2 ; 3 )
on ( 3 ; 11 )
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on ( 11 ; 7 )
on-table ( 1 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
clear ( 1 )
clear ( 2 )
clear ( 5 )
clear ( 8 )
arm-empty (  )

action déroulée au tps 16
pick-up ( 1 )
état du problème au tps 17
on ( 2 ; 3 )
on ( 3 ; 11 )
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on ( 11 ; 7 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
clear ( 2 )
clear ( 5 )
clear ( 8 )
holding ( 1 )

action déroulée au tps 17
stack ( 1 ; 5 )
/*********************************/
            etat final
/*********************************/
on ( 1 ; 5 )
on ( 2 ; 3 )
on ( 3 ; 11 )
```

```
on ( 5 ; 10 )
on ( 7 ; 6 )
on ( 8 ; 9 )
on ( 9 ; 4 )
on ( 11 ; 7 )
on-table ( 4 )
on-table ( 6 )
on-table ( 10 )
clear ( 1 )
clear ( 2 )
clear ( 8 )
```
arm-empty ( )

2) le domaine « logistics »

2.1) le fichier domain.pddl

```
;; logistics domain Typed version.
;;

(define (domain logistics)
  (:requirements :strips :typing)
  (:types truck
          airplane - vehicle
          package
          vehicle - physobj
          airport
          location - place
          city
          place
          physobj - object)

  (:predicates    (in-city ?loc - place ?city - city)
            (at ?obj - physobj ?loc - place)
            (in ?pkg - package ?veh - vehicle))

(:action LOAD-TRUCK
   :parameters    (?pkg - package ?truck - truck ?loc - place)
   :precondition  (and (at ?truck ?loc) (at ?pkg ?loc))
   :effect        (and (not (at ?pkg ?loc)) (in ?pkg ?truck)))

(:action LOAD-AIRPLANE
  :parameters   (?pkg - package ?airplane - airplane ?loc - place)
  :precondition (and (at ?pkg ?loc) (at ?airplane ?loc))
  :effect       (and (not (at ?pkg ?loc)) (in ?pkg ?airplane)))

(:action UNLOAD-TRUCK
  :parameters   (?pkg - package ?truck - truck ?loc - place)
  :precondition (and (at ?truck ?loc) (in ?pkg ?truck))
  :effect       (and (not (in ?pkg ?truck)) (at ?pkg ?loc)))

(:action UNLOAD-AIRPLANE
```

```
  :parameters    (?pkg - package ?airplane - airplane ?loc - place)
  :precondition  (and (in ?pkg ?airplane) (at ?airplane ?loc))
  :effect        (and (not (in ?pkg ?airplane)) (at ?pkg ?loc)))

(:action DRIVE-TRUCK
  :parameters (?truck - truck ?loc-from - place ?loc-to - place ?city -
city)
  :precondition
   (and (at ?truck ?loc-from) (in-city ?loc-from ?city) (in-city ?loc-to
?city))
  :effect
   (and (not (at ?truck ?loc-from)) (at ?truck ?loc-to)))

(:action FLY-AIRPLANE
  :parameters (?airplane - airplane ?loc-from - airport ?loc-to - airport)
  :precondition
   (at ?airplane ?loc-from)
  :effect
   (and (not (at ?airplane ?loc-from)) (at ?airplane ?loc-to)))
)
```

2.2) le fichier probLOGISTICS-4-0.pddl

```
(define (problem logistics-4-0)
(:domain logistics)
(:objects
 apn1 - airplane
 apt1 apt2 - airport
 pos2 pos1 - location
 cit2 cit1 - city
 tru2 tru1 - truck
 obj23 obj22 obj21 obj13 obj12 obj11 - package)

(:init (at apn1 apt2) (at tru1 pos1) (at obj11 pos1)
 (at obj12 pos1) (at obj13 pos1) (at tru2 pos2) (at obj21 pos2) (at obj22
pos2)
 (at obj23 pos2) (in-city pos1 cit1) (in-city apt1 cit1) (in-city pos2
cit2)
 (in-city apt2 cit2))

(:goal (and (at obj11 apt1) (at obj23 pos1) (at obj13 apt1) (at obj21
pos1)))
)
```

résultat obtenu :

```
/*********************************/
            etat initial
/*********************************/
in-city ( apt1 ; cit1 )
in-city ( apt2 ; cit2 )
in-city ( pos2 ; cit2 )
in-city ( pos1 ; cit1 )
at ( apn1 ; apt2 )
at ( obj23 ; pos2 )
at ( obj22 ; pos2 )
at ( obj21 ; pos2 )
at ( obj13 ; pos1 )
```

```
at ( obj12 ; pos1 )
at ( obj11 ; pos1 )
at ( tru2 ; pos2 )
at ( tru1 ; pos1 )
action déroulée au tps 0
LOAD-TRUCK ( obj23 ; tru2 ; pos2 )
LOAD-TRUCK ( obj21 ; tru2 ; pos2 )
LOAD-TRUCK ( obj13 ; tru1 ; pos1 )
état du problème au tps 1
in-city ( apt1 ; cit1 )
in-city ( apt2 ; cit2 )
in-city ( pos2 ; cit2 )
in-city ( pos1 ; cit1 )
at ( apn1 ; apt2 )
at ( obj11 ; pos1 )
at ( tru2 ; pos2 )
at ( tru1 ; pos1 )
in ( obj23 ; tru2 )
in ( obj21 ; tru2 )
in ( obj13 ; tru1 )

action déroulée au tps 1
LOAD-TRUCK ( obj11 ; tru1 ; pos1 )
DRIVE-TRUCK ( tru2 ; pos2 ; apt2 ; cit2 )
état du problème au tps 2
in-city ( apt1 ; cit1 )
in-city ( apt2 ; cit2 )
in-city ( pos2 ; cit2 )
in-city ( pos1 ; cit1 )
at ( tru2 ; apt2 )
at ( apn1 ; apt2 )
at ( tru1 ; pos1 )
in ( obj23 ; tru2 )
in ( obj21 ; tru2 )
in ( obj13 ; tru1 )
in ( obj11 ; tru1 )

action déroulée au tps 2
UNLOAD-TRUCK ( obj23 ; tru2 ; apt2 )
UNLOAD-TRUCK ( obj21 ; tru2 ; apt2 )
état du problème au tps 3
in-city ( apt1 ; cit1 )
in-city ( pos1 ; cit1 )
at ( obj23 ; apt2 )
at ( obj21 ; apt2 )
at ( tru2 ; apt2 )
at ( apn1 ; apt2 )
at ( tru1 ; pos1 )
in ( obj13 ; tru1 )
in ( obj11 ; tru1 )

action déroulée au tps 3
LOAD-AIRPLANE ( obj23 ; apn1 ; apt2 )
LOAD-AIRPLANE ( obj21 ; apn1 ; apt2 )
état du problème au tps 4
in-city ( apt1 ; cit1 )
in-city ( pos1 ; cit1 )
at ( apn1 ; apt2 )
at ( tru1 ; pos1 )
in ( obj13 ; tru1 )
```

```
in ( obj11 ; tru1 )
in ( obj23 ; apn1 )
in ( obj21 ; apn1 )

action déroulée au tps 4
DRIVE-TRUCK ( tru1 ; pos1 ; apt1 ; cit1 )
FLY-AIRPLANE ( apn1 ; apt2 ; apt1 )
état du problème au tps 5
in-city ( apt1 ; cit1 )
in-city ( pos1 ; cit1 )
at ( tru1 ; apt1 )
at ( apn1 ; apt1 )
in ( obj13 ; tru1 )
in ( obj11 ; tru1 )
in ( obj23 ; apn1 )
in ( obj21 ; apn1 )

action déroulée au tps 5
UNLOAD-TRUCK ( obj13 ; tru1 ; apt1 )
UNLOAD-TRUCK ( obj11 ; tru1 ; apt1 )
UNLOAD-AIRPLANE ( obj23 ; apn1 ; apt1 )
UNLOAD-AIRPLANE ( obj21 ; apn1 ; apt1 )
état du problème au tps 6
in-city ( apt1 ; cit1 )
in-city ( pos1 ; cit1 )
at ( obj23 ; apt1 )
at ( obj21 ; apt1 )
at ( obj13 ; apt1 )
at ( obj11 ; apt1 )
at ( tru1 ; apt1 )
at ( apn1 ; apt1 )

action déroulée au tps 6
LOAD-TRUCK ( obj23 ; tru1 ; apt1 )
LOAD-TRUCK ( obj21 ; tru1 ; apt1 )
état du problème au tps 7
in-city ( apt1 ; cit1 )
in-city ( pos1 ; cit1 )
at ( obj13 ; apt1 )
at ( obj11 ; apt1 )
at ( tru1 ; apt1 )
in ( obj23 ; tru1 )
in ( obj21 ; tru1 )

action déroulée au tps 7
DRIVE-TRUCK ( tru1 ; apt1 ; pos1 ; cit1 )
état du problème au tps 8
in-city ( apt1 ; cit1 )
in-city ( pos1 ; cit1 )
at ( obj13 ; apt1 )
at ( obj11 ; apt1 )
at ( tru1 ; pos1 )
in ( obj23 ; tru1 )
in ( obj21 ; tru1 )

action déroulée au tps 8
UNLOAD-TRUCK ( obj23 ; tru1 ; pos1 )
UNLOAD-TRUCK ( obj21 ; tru1 ; pos1 )
/*********************************/
            etat final
```

```
/*********************************/
at ( obj13 ; apt1 )
at ( obj11 ; apt1 )
at ( obj23 ; pos1 )
at ( obj21 ; pos1 )
at ( tru1 ; pos1 )
```

2.3) le fichier probLOGISTICS-6-0.pddl

```
;; original name logistics.c
;; (:length (:parallel 13))
;; optimal
;; #actions 63 #states 10^10
;;

(define (problem log006)
    (:domain logistics-typed)
    (:objects
        package1 - PACKAGE
        package2 - PACKAGE
        package3 - PACKAGE
        package4 - PACKAGE
        package5 - PACKAGE
        package6 - PACKAGE
        package7 - PACKAGE

        airplane1 - AIRPLANE
        airplane2 - AIRPLANE

        pgh - CITY
        bos - CITY
        la - CITY
        ny - CITY

        pgh-truck - TRUCK
        bos-truck - TRUCK
        la-truck - TRUCK
        ny-truck - TRUCK

        pgh-po - LOCATION
        bos-po - LOCATION
        la-po - LOCATION
        ny-po - LOCATION

        pgh-airport - (either LOCATION AIRPORT)
        bos-airport - (either LOCATION AIRPORT)
        la-airport - (either LOCATION AIRPORT)
        ny-airport - (either LOCATION AIRPORT)
    )
    (:init

        (in-city pgh-po pgh)
        (in-city pgh-airport pgh)

        (in-city bos-po bos)
        (in-city bos-airport bos)

        (in-city la-po la)
        (in-city la-airport la)
```

```
        (in-city ny-po ny)
        (in-city ny-airport ny)

        (at package1 pgh-po)
        (at package2 pgh-po)
        (at package3 pgh-po)
        (at package4 ny-po)
        (at package5 bos-po)
        (at package6 bos-po)
        (at package7 ny-po)

        (at airplane1 pgh-airport)
        (at airplane2 pgh-airport)

        (at bos-truck bos-po)
        (at pgh-truck pgh-po)
        (at la-truck la-po)
        (at ny-truck ny-po)

    )
    (:goal (and
        (at package1 bos-po)
        (at package2 ny-po)
        (at package3 la-po)
        (at package4 la-airport)
        (at package5 pgh-po)
        (at package6 ny-airport)
        (at package7 pgh-po)
    ))
)
```

résultat obtenu :

```
/*********************************/
           etat initial
/*********************************/
in-city ( apt2 ; cit2 )
in-city ( apt1 ; cit1 )
in-city ( pos2 ; cit2 )
in-city ( pos1 ; cit1 )
at ( apn1 ; apt1 )
at ( obj23 ; pos2 )
at ( obj22 ; pos2 )
at ( obj21 ; pos2 )
at ( obj13 ; pos1 )
at ( obj12 ; pos1 )
at ( obj11 ; pos1 )
at ( tru2 ; pos2 )
at ( tru1 ; pos1 )
action déroulée au tps 0
LOAD-TRUCK ( obj23 ; tru2 ; pos2 )
LOAD-TRUCK ( obj21 ; tru2 ; pos2 )
LOAD-TRUCK ( obj13 ; tru1 ; pos1 )
LOAD-TRUCK ( obj12 ; tru1 ; pos1 )
LOAD-TRUCK ( obj11 ; tru1 ; pos1 )
état du problème au tps 1
in-city ( apt2 ; cit2 )
in-city ( apt1 ; cit1 )
```

```
in-city ( pos2 ; cit2 )
in-city ( pos1 ; cit1 )
at ( apn1 ; apt1 )
at ( obj22 ; pos2 )
at ( tru2 ; pos2 )
at ( tru1 ; pos1 )
in ( obj23 ; tru2 )
in ( obj21 ; tru2 )
in ( obj13 ; tru1 )
in ( obj12 ; tru1 )
in ( obj11 ; tru1 )

action déroulée au tps 1
DRIVE-TRUCK ( tru2 ; pos2 ; apt2 ; cit2 )
DRIVE-TRUCK ( tru1 ; pos1 ; apt1 ; cit1 )
état du problème au tps 2
in-city ( apt2 ; cit2 )
in-city ( apt1 ; cit1 )
in-city ( pos2 ; cit2 )
in-city ( pos1 ; cit1 )
at ( tru2 ; apt2 )
at ( tru1 ; apt1 )
at ( apn1 ; apt1 )
at ( obj22 ; pos2 )
in ( obj23 ; tru2 )
in ( obj21 ; tru2 )
in ( obj13 ; tru1 )
in ( obj12 ; tru1 )
in ( obj11 ; tru1 )

action déroulée au tps 2
UNLOAD-TRUCK ( obj23 ; tru2 ; apt2 )
UNLOAD-TRUCK ( obj21 ; tru2 ; apt2 )
UNLOAD-TRUCK ( obj13 ; tru1 ; apt1 )
UNLOAD-TRUCK ( obj12 ; tru1 ; apt1 )
UNLOAD-TRUCK ( obj11 ; tru1 ; apt1 )
état du problème au tps 3
in-city ( apt2 ; cit2 )
in-city ( pos2 ; cit2 )
at ( obj23 ; apt2 )
at ( obj21 ; apt2 )
at ( obj13 ; apt1 )
at ( obj12 ; apt1 )
at ( obj11 ; apt1 )
at ( tru2 ; apt2 )
at ( tru1 ; apt1 )
at ( apn1 ; apt1 )
at ( obj22 ; pos2 )

action déroulée au tps 3
LOAD-TRUCK ( obj23 ; tru2 ; apt2 )
LOAD-AIRPLANE ( obj13 ; apn1 ; apt1 )
LOAD-AIRPLANE ( obj12 ; apn1 ; apt1 )
LOAD-AIRPLANE ( obj11 ; apn1 ; apt1 )
état du problème au tps 4
in-city ( apt2 ; cit2 )
in-city ( pos2 ; cit2 )
at ( obj21 ; apt2 )
at ( tru2 ; apt2 )
at ( apn1 ; apt1 )
```

```
at ( obj22 ; pos2 )
in ( obj23 ; tru2 )
in ( obj13 ; apn1 )
in ( obj12 ; apn1 )
in ( obj11 ; apn1 )

action déroulée au tps 4
FLY-AIRPLANE ( apn1 ; apt1 ; apt2 )
état du problème au tps 5
in-city ( apt2 ; cit2 )
in-city ( pos2 ; cit2 )
at ( obj21 ; apt2 )
at ( tru2 ; apt2 )
at ( apn1 ; apt2 )
at ( obj22 ; pos2 )
in ( obj23 ; tru2 )
in ( obj13 ; apn1 )
in ( obj12 ; apn1 )
in ( obj11 ; apn1 )

action déroulée au tps 5
UNLOAD-TRUCK ( obj23 ; tru2 ; apt2 )
UNLOAD-AIRPLANE ( obj13 ; apn1 ; apt2 )
UNLOAD-AIRPLANE ( obj12 ; apn1 ; apt2 )
UNLOAD-AIRPLANE ( obj11 ; apn1 ; apt2 )
état du problème au tps 6
in-city ( apt2 ; cit2 )
in-city ( pos2 ; cit2 )
at ( obj23 ; apt2 )
at ( obj21 ; apt2 )
at ( obj13 ; apt2 )
at ( obj12 ; apt2 )
at ( obj11 ; apt2 )
at ( tru2 ; apt2 )
at ( apn1 ; apt2 )
at ( obj22 ; pos2 )

action déroulée au tps 6
LOAD-TRUCK ( obj13 ; tru2 ; apt2 )
LOAD-AIRPLANE ( obj23 ; apn1 ; apt2 )
état du problème au tps 7
in-city ( apt2 ; cit2 )
in-city ( pos2 ; cit2 )
at ( obj21 ; apt2 )
at ( obj12 ; apt2 )
at ( obj11 ; apt2 )
at ( tru2 ; apt2 )
at ( apn1 ; apt2 )
at ( obj22 ; pos2 )
in ( obj13 ; tru2 )
in ( obj23 ; apn1 )

action déroulée au tps 7
DRIVE-TRUCK ( tru2 ; apt2 ; pos2 ; cit2 )
FLY-AIRPLANE ( apn1 ; apt2 ; apt1 )
état du problème au tps 8
in-city ( apt2 ; cit2 )
in-city ( pos2 ; cit2 )
at ( obj21 ; apt2 )
at ( obj12 ; apt2 )
```

```
at ( obj11 ; apt2 )
at ( apn1 ; apt1 )
at ( obj22 ; pos2 )
at ( tru2 ; pos2 )
in ( obj13 ; tru2 )
in ( obj23 ; apn1 )

action déroulée au tps 8
UNLOAD-TRUCK ( obj13 ; tru2 ; pos2 )
UNLOAD-AIRPLANE ( obj23 ; apn1 ; apt1 )
/*********************************/
            etat final
/*********************************/
at ( obj23 ; apt1 )
at ( obj21 ; apt2 )
at ( obj12 ; apt2 )
at ( obj11 ; apt2 )
at ( apn1 ; apt1 )
at ( obj22 ; pos2 )
at ( obj13 ; pos2 )
at ( tru2 ; pos2 )
```

2.4) le fichier prob002-rocket-a.pddl

```
;; original name rocket_ext.a
;; (:length (:parallel 7))
;; optimal
;;

(define (problem log002)
  (:domain logistics-typed)
  (:objects mxf - package
          avrim - package
          alex - package
          jason - package
          pencil - package
          paper - package
          april - package
          michelle - package
          betty - package
          lisa - package
          airplane1 - airplane
          airplane2 - airplane
          lon-airport - airport
          par-airport -  airport
          jfk-airport -  airport
          bos-airport -  airport)
  (:init (at airplane1 jfk-airport)
       (at airplane2 bos-airport)
       (at mxf par-airport)
       (at avrim par-airport)
       (at alex par-airport)
       (at jason jfk-airport)
       (at pencil lon-airport)
       (at paper lon-airport)
       (at michelle lon-airport)
       (at april lon-airport)
       (at betty lon-airport)
       (at lisa lon-airport)
```

```
      )
  (:goal (and
        (at mxf bos-airport)
        (at avrim jfk-airport)
        (at pencil bos-airport)
        (at alex jfk-airport)
        (at april bos-airport)
        (at lisa par-airport)
        (at michelle jfk-airport)
        (at jason bos-airport)
        (at paper par-airport)
        (at betty jfk-airport)
        )
       )
  )
```

résultat obtenu :

```
/**********************************/
            etat initial
/**********************************/
at ( mxf ; par-airport )
at ( avrim ; par-airport )
at ( alex ; par-airport )
at ( jason ; jfk-airport )
at ( pencil ; lon-airport )
at ( paper ; lon-airport )
at ( april ; lon-airport )
at ( michelle ; lon-airport )
at ( betty ; lon-airport )
at ( lisa ; lon-airport )

at ( airplane1 ; jfk-airport )
at ( airplane2 ; bos-airport )
action déroulée au tps 0
FLY-AIRPLANE ( airplane1 ; jfk-airport ; lon-airport )
FLY-AIRPLANE ( airplane2 ; bos-airport ; par-airport )
état du problème au tps 1
at ( mxf ; par-airport )
at ( avrim ; par-airport )
at ( alex ; par-airport )
at ( jason ; jfk-airport )
at ( pencil ; lon-airport )
at ( paper ; lon-airport )
at ( april ; lon-airport )
at ( michelle ; lon-airport )
at ( betty ; lon-airport )
at ( lisa ; lon-airport )
at ( airplane1 ; lon-airport )
at ( airplane2 ; par-airport )

action déroulée au tps 1
LOAD-AIRPLANE ( avrim ; airplane2 ; par-airport )
LOAD-AIRPLANE ( alex ; airplane2 ; par-airport )
LOAD-AIRPLANE ( pencil ; airplane1 ; lon-airport )
LOAD-AIRPLANE ( paper ; airplane1 ; lon-airport )
LOAD-AIRPLANE ( april ; airplane1 ; lon-airport )
```

```
LOAD-AIRPLANE ( michelle ; airplane1 ; lon-airport )
LOAD-AIRPLANE ( betty ; airplane1 ; lon-airport )
LOAD-AIRPLANE ( lisa ; airplane1 ; lon-airport )
état du problème au tps 2
at ( mxf ; par-airport )
at ( jason ; jfk-airport )
at ( airplane1 ; lon-airport )
at ( airplane2 ; par-airport )
in ( avrim ; airplane2 )
in ( alex ; airplane2 )
in ( pencil ; airplane1 )
in ( paper ; airplane1 )
in ( april ; airplane1 )
in ( michelle ; airplane1 )
in ( betty ; airplane1 )
in ( lisa ; airplane1 )

action déroulée au tps 2
LOAD-AIRPLANE ( mxf ; airplane2 ; par-airport )
FLY-AIRPLANE ( airplane1 ; lon-airport ; jfk-airport )
état du problème au tps 3
at ( jason ; jfk-airport )
at ( airplane1 ; jfk-airport )
at ( airplane2 ; par-airport )
in ( mxf ; airplane2 )
in ( avrim ; airplane2 )
in ( alex ; airplane2 )
in ( pencil ; airplane1 )
in ( paper ; airplane1 )

in ( april ; airplane1 )
in ( michelle ; airplane1 )
in ( betty ; airplane1 )
in ( lisa ; airplane1 )

action déroulée au tps 3
UNLOAD-AIRPLANE ( pencil ; airplane1 ; jfk-airport )
UNLOAD-AIRPLANE ( april ; airplane1 ; jfk-airport )
UNLOAD-AIRPLANE ( michelle ; airplane1 ; jfk-airport )
UNLOAD-AIRPLANE ( betty ; airplane1 ; jfk-airport )
FLY-AIRPLANE ( airplane2 ; par-airport ; jfk-airport )
état du problème au tps 4
at ( jason ; jfk-airport )
at ( pencil ; jfk-airport )
at ( april ; jfk-airport )
at ( michelle ; jfk-airport )
at ( betty ; jfk-airport )
at ( airplane1 ; jfk-airport )
at ( airplane2 ; jfk-airport )
in ( mxf ; airplane2 )
in ( avrim ; airplane2 )
in ( alex ; airplane2 )
in ( paper ; airplane1 )
in ( lisa ; airplane1 )

action déroulée au tps 4
LOAD-AIRPLANE ( jason ; airplane2 ; jfk-airport )
LOAD-AIRPLANE ( pencil ; airplane2 ; jfk-airport )
LOAD-AIRPLANE ( april ; airplane2 ; jfk-airport )
UNLOAD-AIRPLANE ( avrim ; airplane2 ; jfk-airport )
```

```
UNLOAD-AIRPLANE ( alex ; airplane2 ; jfk-airport )
FLY-AIRPLANE ( airplane1 ; jfk-airport ; par-airport )
état du problème au tps 5
at ( avrim ; jfk-airport )
at ( alex ; jfk-airport )
at ( michelle ; jfk-airport )
at ( betty ; jfk-airport )
at ( airplane1 ; par-airport )
at ( airplane2 ; jfk-airport )
in ( mxf ; airplane2 )
in ( jason ; airplane2 )
in ( pencil ; airplane2 )
in ( paper ; airplane1 )
in ( april ; airplane2 )
in ( lisa ; airplane1 )

action déroulée au tps 5
UNLOAD-AIRPLANE ( paper ; airplane1 ; par-airport )
FLY-AIRPLANE ( airplane2 ; jfk-airport ; bos-airport )
état du problème au tps 6
at ( avrim ; jfk-airport )
at ( alex ; jfk-airport )
at ( paper ; par-airport )
at ( michelle ; jfk-airport )

at ( betty ; jfk-airport )
at ( airplane1 ; par-airport )
at ( airplane2 ; bos-airport )
in ( mxf ; airplane2 )
in ( jason ; airplane2 )
in ( pencil ; airplane2 )
in ( april ; airplane2 )
in ( lisa ; airplane1 )

action déroulée au tps 6
UNLOAD-AIRPLANE ( mxf ; airplane2 ; bos-airport )
UNLOAD-AIRPLANE ( jason ; airplane2 ; bos-airport )
UNLOAD-AIRPLANE ( pencil ; airplane2 ; bos-airport )
UNLOAD-AIRPLANE ( april ; airplane2 ; bos-airport )
UNLOAD-AIRPLANE ( lisa ; airplane1 ; par-airport )
/*********************************/
            etat final
/*********************************/
at ( mxf ; bos-airport )
at ( avrim ; jfk-airport )
at ( alex ; jfk-airport )
at ( jason ; bos-airport )
at ( pencil ; bos-airport )
at ( paper ; par-airport )
at ( april ; bos-airport )
at ( michelle ; jfk-airport )
at ( betty ; jfk-airport )
at ( lisa ; par-airport )
at ( airplane1 ; par-airport )
at ( airplane2 ; bos-airport )
```

2.5) le fichier prob004-log-a.pddl

```
;; original name logistics.a
;; extended version of logistics_facts7h
;; (:length (:parallel 11))
;; optimal
;; #actions 54 #states 10^11
;;
;; note: by going to a non-typed representation
;;       of the problems, the instances become (somewhat)
;;       harder to solve.
;;       (larger propositional representation)
;;

(define (problem log004)
    (:domain logistics-typed)
    (:objects
        package1 package2 package3 package4 package5 package6 package7
package8 - package

        airplane1 airplane2 - airplane

        pgh bos la - city

        pgh-truck bos-truck la-truck - truck

        pgh-po bos-po la-po - location

        pgh-airport bos-airport la-airport - (either location airport)
    )
    (:init
        (in-city pgh-po pgh)
        (in-city pgh-airport pgh)

        (in-city bos-po bos)
        (in-city bos-airport bos)

        (in-city la-po la)
        (in-city la-airport la)

        (at package1 pgh-po)
        (at package2 pgh-po)
        (at package3 pgh-po)
        (at package4 pgh-po)
        (at package5 bos-po)
        (at package6 bos-po)
        (at package7 bos-po)
        (at package8 la-po)

        (at airplane1 pgh-airport)
        (at airplane2 pgh-airport)

        (at bos-truck bos-po)
        (at pgh-truck pgh-po)
        (at la-truck la-po)
    )
    (:goal (and
        (at package1 bos-po)
        (at package2 bos-airport)
        (at package3 la-po)
        (at package4 la-airport)
```

```
        (at package5 pgh-po)
        (at package6 pgh-airport)
        (at package7 pgh-po)
        (at package8 pgh-po)
    ))
)
```

pas de résultat obtenu.