Composition flexible par planification automatique

Cyrille Martin

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : MENS0602085A du 7 août 2006

Présentée par

Cyrille MARTIN

Thèse dirigée par **Gaëlle CALVARY**
et codirigée par **Humbert FIORINO**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
et de l'**École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Composition flexible
par planification automatique

Thèse soutenue publiquement le **jeudi 4 octobre 2012**,
devant le jury composé de :

Mr Olivier BOISSIER
PR, LSTI, ENS Mines Saint-Etienne, Rapporteur
Mr René MANDIAU
PR, LAMIH, Université de Valenciennes, Rapporteur
Mr Patrick REIGNIER
PR, LIG, Grenoble INP, Examinateur
Mr Thierry VIDAL
MCF, LGP, ENI de Tarbes, Examinateur
Mme Gaëlle CALVARY
PR, LIG, Grenoble INP, Directeur de thèse
Mr Humbert FIORINO
MCF, LIG, Université Joseph Fourier, Co-Directeur de thèse

Remerciements

Je souhaite remercier ici les personnes qui, chacune à sa manière, ont permis à ce travail d'aboutir et ont ainsi fait le docteur ès sciences que je suis aujourd'hui.

Les plus impliqués dans mon travail, autant scientifiquement que humainement, sont mes codirecteurs de thèses, Gaëlle et Humbert. Avec deux approches très différentes, vous avez contribué à façonner mes idées. Merci d'avoir partagé un peu de vous avec moi : je me suis approprié le maximum que j'ai pu. Gaëlle, je te remercie particulièrement pour tes qualités humaines, notamment d'écoute, qui ont permis un travail agréable et efficace durant toutes ces années. Humbert, si nos réunions furent plus régulières et, peut-être en conséquence, les échanges plus intenses, je te remercie pour tes conseils avisés, qui ont toujours été entendues. Notamment, ton attachement à ce que la forme soit au service du fond tient un rôle certain dans l'intelligibilité du manuscrit.

Les premiers concernés par cette intelligibilité furent Olivier et René en tant que rapporteurs de ma thèse. Je vous remercie pour le travail que vous avez effectué sur ma thèse, pour la qualité de vos retours. Lire dans vos rapports votre compréhension profonde de mes contributions et de leurs cohérences respectives fut pour moi un vrai soulagement ! Je vous remercie aussi avec Patrick et Thierry en tant que jury de ma thèse pour les questions que vous avez soulevé durant la soutenance, pour vos commentaires éclairants et vos conseils issus de cette rencontre.

Merci Damien pour ta librairie java pddl4j et pour l'implémentation de `GraphPlan` que j'ai téléchargé, essayé, utilisé, modifié avec plaisir et sur lequel j'ai fondé mon planificateur. Merci Barbara pour le travail effectué durant ton stage d'IRL. Merci à Béatrice, Valérie et Christine qui ont répondu avec le sourire à chacune de mes sollicitations, même lorsqu'elles sortaient du cadre stricte de la thèse...

Les membres de mes deux équipes de recherche, IIHM et MAGMA, tiennent un rôle particulier dans ces années passées au LIG, et je les en remercie. Rien de tout cela n'aurait été possible sans mon accueil par Joëlle puis par Laurence en tant que responsables de l'équipe IIHM, et par Yves puis par Julie du côté de MAGMA. Je vais m'attarder un peu à MAGMA, comme je l'ai fait en tant que doctorant... Outre Yves et Julie chez lesquels j'ai particulièrement apprécié la droiture tout en ouverture d'esprit, je pense en premier lieu à Yoann : nos deux parcours de thèse ont des similitudes fortes, tant scientifiquement que humainement, et je te remercie pour nos échanges autant professionnels que personnels. J'ai de même une pensée particulière pour mes premiers jours de recherche en IA, alors en tant que stagiaire : Joris, Guillaume, Shadi, Ludivine et Hussein furent les acteurs bienveillants de mon intégration au sein de ce qui deviendra "mon équipe". Au fil des années, j'ai passé d'excellents moments avec les membres de cette équipe : Catherine, Fabien, Laure et Benoît, Sylvie, Laurent [1], Jérémy [2],

1. Les moments furent furtifs... Mais ça compte tout de même.
2. Des moments tout en humour.

Robin [3], Carole [4], Wafa, Uktu, Lyuba... Merci.

Durant la préparation de ma thèse, j'ai apprécié l'animation de mon espace de travail. Bien entendu, son rôle premier a été de supporter l'écriture de thèses, et singulièrement celles d'Axelle et Nathalie, mais mon bureau fut aussi un point de rendez-vous pour le départ au RU ou comme départ de jogging, avec notamment Mickaël et Martin, une salle de stockage pour LIG-Synergy où Emeric, Yves et d'autres de l'association se sont fournis, une salle de réunion pour diverses activités... Merci à tous ceux qui sont passés par là, car ce fut systématiquement un plaisir de partager ces moments avec vous. En parlant de LIG-Synergy, je remercie tous ceux qui m'ont aidé à monter l'association, et tous ceux qui s'investirent, s'investissent et s'investiront pour le laboratoire à travers l'association.

En sortant du cadre professionnel, j'arrive naturellement à Pauline. Merci Pauline d'avoir été présente, autant dans les périodes agréables que difficiles : tu m'as beaucoup donné, pardonné mon investissement dans la thèse au détriment d'autres activités, "supporté" mes humeurs, mes envies, mon humour, et cela fut essentiel. Je remercie chaleureusement tous mes proches, en commençant par mes parents, Gérard et Claire, pour m'avoir suivi sur le chemin que je me suis choisi. Je remercie mes frères et sœurs, Peggy, Boris, Valentin, Rachel et Sylvain : selon vos connaissances, vos intérêts, votre curiosité, chacun de vous s'est intéressé, m'a questionné, a commenté, et vous formiez le panel privilégié de personnes sur lesquelles tester la vulgarisation de mes travaux... Mes amis, les plus proches comme les plus éloignés, ont apporté l'équilibre nécessaire face à l'intensité de la période de thèse... Sans vous nommer, car je ne saurais être exhaustif, je remercie chacun d'entre vous.

Encore merci à tous : ces quelques lignes ne suffisent pas à exprimer toute ma gratitude... Et merci aussi à tous ceux qui sont venus participer à ma soutenance, le 4 octobre 2012. Et finalement, je te remercie, lecteur de ces lignes, pour l'intérêt que tu portes sur mon travail et j'espère que la lecture de ce manuscrit te sera utile et agréable.

3. Souvent en musique.
4. Sportivement.

Résumé

Nous nous positionnons dans un contexte d'informatique ambiante dans lequel il arrive que les besoins de l'utilisateur n'aient pas été prévus, notamment en situation exceptionnelle. Dans ce cas, il peut ne pas exister de système préconçu qui réponde exactement à ces besoins. Pour les satisfaire, il faut alors pouvoir composer les systèmes disponibles dans l'environnement, et le système composé doit permettre à l'utilisateur de faire des choix à l'exécution. Ainsi, l'utilisateur a la possibilité d'adapter l'exécution de la composition à son contexte. Cela signifie que la composition intègre des structures de contrôle de l'exécution, destinées à l'utilisateur : la composition est dite « flexible ».

Dans cette thèse, nous proposons de répondre au problème de la composition flexible en contexte d'intelligence ambiante avec un planificateur produisant des plans flexibles. Dans un premier temps, nous proposons une modélisation de la planification flexible. Pour cela, nous définissons les opérateurs de séquence et d'alternative, utilisés pour caractériser les plans flexibles. Nous définissons deux autres opérateurs au moyen de la séquence et de l'alternative : l'entrelacement et l'itération. Nous nous référons à ce cadre théorique pour délimiter la flexibilité traitée par notre planificateur Λ-`GraphPlan`. L'originalité de Λ-`GraphPlan` est de produire des itérations en s'appuyant sur une approche par graphe de planification. Nous montrons notamment que Λ-`GraphPlan` est très performant avec les domaines se prêtant à la construction de structures itératives.

Abstract

In a context of Ambient Intelligence, some of the user's needs might not be anticipated, e.g. when the user is in an unforeseen situation. In this case, it is possible that there is no system that exactly meets their needs. By composing the available systems, the user could obtain a new system that satisfies their needs. In order to adapt the composition to the context, the composition must allow the user to make choices at runtime. This means that the composition should include control structures for the user : the composition is so-called "flexible".

In this thesis, I deal with the problem of the flexible composition by automated planning for which I propose a model. The sequence and the choice operators are defined and used to characterize flexible plans. Two other operators are then derived from the sequence and the choice operators : the interleaving and the iteration operators. I refer to this framework in order to define the flexibility produced by my planner, Λ-`GraphPlan`, which is based on the planning graph. The originality of Λ-`GraphPlan` is to produce iterations. I show that Λ-`GraphPlan` is very efficient on domains that allow the construction of iterative structures.

Sommaire

Première partie

Positionnement

Chapitre 1

Planification flexible pour l'intelligence ambiante

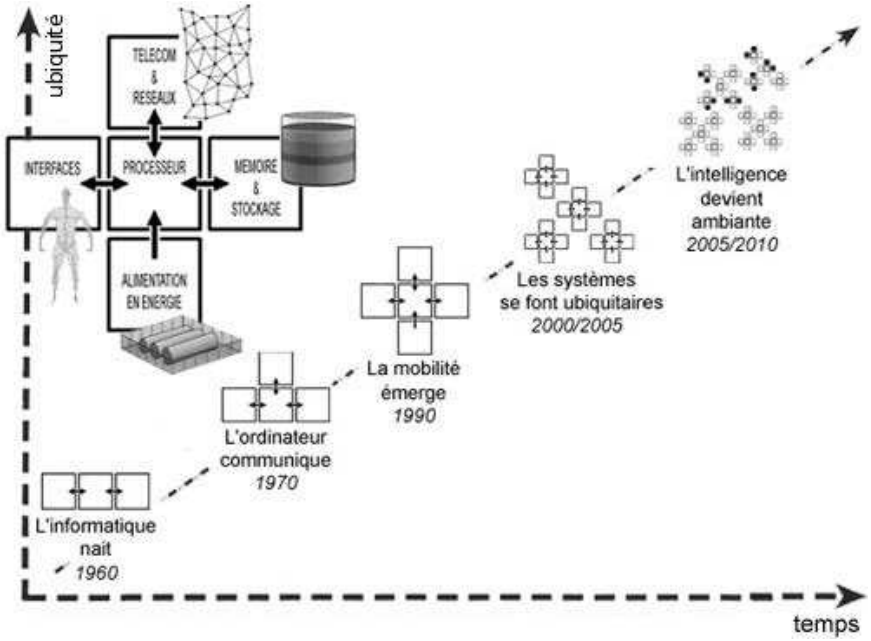


Figure 1.1 – Évolution de l'informatique (adapté de [Wal07]), de l'ordinateur partagé par plusieurs dans les années 60 jusqu'à l'interconnexion entre différents dispositifs après l'an 2000, associés aux environs de l'année 2010 à l'intelligence ambiante.

L'informatique a profité de progrès technologiques pour développer son ubiquité, comme le représente la figure 1.1. Notamment, l'autonomie énergétique des dispositifs [1] ainsi que leur miniaturisation ont rendu l'informatique mobile. Il n'est pas rare que les utilisateurs possèdent plusieurs dispositifs, par exemple un téléphone, un ordinateur portable voire un objet communiquant comme un Nabaztag [HBT11]. En outre, les progrès des télécommunications permettent un accès constant à l'Internet, et donc à un nombre important de services [2] (par exemple les *Pages Jaunes* en ligne, *Skype* pour communiquer ou encore *Google Maps* pour localiser). Ces services répondent individuellement à un objectif précis et bien délimité. L'informatique est donc *ubiquitaire* [Wei99] puisque disponible partout et tout le temps. L'utilisateur, son environnement et les dispositifs et services disponibles à un instant donné forment le *contexte d'usage* [CCT+03] : du fait de l'ubiquité de l'informatique, les contextes sont très *variés* et *variables*. En effet, l'utilisateur est imprévisible

1. Nous appellerons *dispositif* tout matériel informatique.
2. Nous appellerons *service* tout logiciel informatique permettant de réaliser un objectif (logiciel, service web, application, *etc.*).

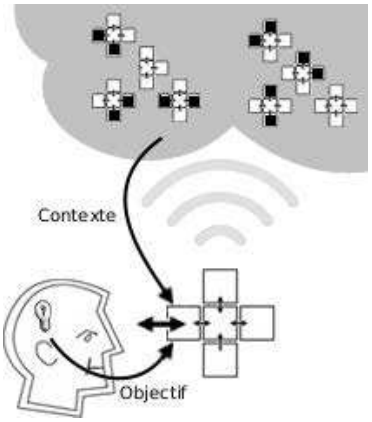


FIGURE 1.2 – En intelligence ambiante, un service utile et utilisable est proposé à l'utilisateur lorsque son objectif est perçu : ce service est adapté au contexte, et adaptable (par l'interaction avec l'utilisateur).

(il peut avoir des exigences inconnues, des préférences d'usage, des objectifs cachés, des attentes particulières) et le contexte est incertain (par exemple lorsque l'utilisateur est en condition de mobilité, qu'un dispositif n'a plus de batterie ou que de nouveaux services sont accessibles).

L'*intelligence ambiante* est l'intersection entre l'informatique ubiquitaire et l'intelligence artificielle [RN03] : les dispositifs et services analysent en continu le contexte et s'y adaptent. Elle tend à rendre l'informatique pro-active [Rei10]. Notamment, lorsqu'il est perçu que l'utilisateur a un objectif précis, une solution logicielle adaptée au contexte est proposée, avec laquelle l'utilisateur interagit. Ainsi, la recherche d'un service qui convienne aux objectifs de l'utilisateur est à la charge de l'intelligence ambiante. La figure 1.2 illustre un utilisateur au sein d'un environnement d'intelligence ambiante : un dispositif lui propose un service avec lequel interagir pour satisfaire son objectif, en respectant les contraintes de son contexte. Ce service répond à l'objectif de l'utilisateur, c'est-à-dire lui est *utile*, et respecte les critères ergonomiques usuels [BS93] pour être *utilisable*. Particulièrement, pour être utilisable, il est *adapté* au contexte, y compris dynamiquement en cas d'évolution du contexte d'usage. Par exemple, lorsqu'un utilisateur dans son salon a besoin de contacter une personne, le service *Skype* doit être proposé puisqu'il permet de réaliser l'objectif. Possédant un téléphone et un ordinateur portable, et plusieurs personnes dans le salon souhaitant participer à la conversation, le service est proposé sur l'ordinateur car il permet une communication de meilleure qualité : le service est adapté au contexte. Au bout d'un temps, l'utilisateur souhaitant continuer en privé et ailleurs, la conversation est transférée sur le téléphone : le service est adapté à l'exécution. Ainsi, les exigences fonctionnelles et non fonctionnelles évoluent dynamiquement.

Nous nous intéressons plus particulièrement aux situations d'intelligence ambiante dans lesquelles aucun service disponible ne répond aux objectifs de l'utilisateur. C'est le cas typiquement lorsque les objectifs de l'utilisateur émergent d'un contexte imprévu : aucun service n'a été préconçu pour satisfaire les objectifs. Pour remédier à cette situation, nous proposons de chercher une composition des services pour répondre aux objectifs. Par exemple [Gab11], si l'utilisateur tombe malade dans une ville qu'il ne connait pas (contexte imprévu), il a pour objectif de se soigner (objectif émergeant). Aucun service particulier ne permet spécifiquement à l'utilisateur de se soigner. En revanche, la composition des services disponibles

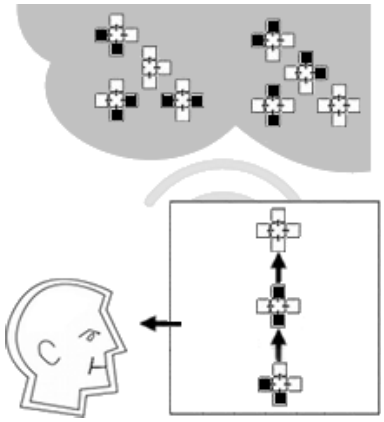


FIGURE 1.3 – Composition de services qui répond à l'objectif de l'utilisateur : la composition est utile pour l'utilisateur.

le permet : le service composé qui propose les *Pages Jaunes* pour trouver un médecin, puis d'appeler le médecin avec *Skype* pour prendre rendez-vous et de le localiser avec *Google Maps* pour s'y rendre. Dans la perspective de limiter la charge de l'utilisateur [3], l'intelligence ambiante doit lui composer *automatiquement* un service utile et utilisable. Notre contexte de travail est la composition automatique en intelligence ambiante.

1.1 Problème de composition automatique flexible

Notre travail se situe dans un contexte de composition automatique, où les objectifs de l'utilisateur sont connus (par exemple, les objectifs sont déduits de l'observation du contexte [SRb03]) mais pour lesquels il n'existe pas de service pré-calculé. Nous soulevons deux questions sur le service composé : composer quoi, c'est-à-dire quels services ; et comment composer, c'est-à-dire avec quels opérateurs de composition. Dans l'exemple du malade, la composition inclut des services qui peuvent être *Pages Jaunes*, *Skype*, *Google Maps*, *etc.* Ces services réalisent individuellement une partie de l'objectif de l'utilisateur. La composition requiert la sélection de ces services. Les opérateurs de composition peuvent être temporels [AH85] (par exemple la séquence « *Pages Jaunes* puis *Skype* puis *Google Maps* »), logiques (par exemple le choix entre *Skype ou iCall*), spatiaux [EF91] (pour une représentation graphique de la composition), *etc.* Les services sélectionnés sont composés par ces opérateurs de façon à ce que le service résultant soit utile et utilisable. La figure 1.3 illustre une composition utile de services disponibles. Pour être utilisable en tout contexte d'usage, la composition doit être adaptable et adaptative au contexte d'usage [BS93]. Pour être adaptable et adaptative, nous soutenons que la composition doit intégrer des structures de contrôle à destination de l'utilisateur (adaptable) et/ou du système (adaptative) : les opérateurs de composition doivent permettre de prendre des décisions durant l'exécution. Ainsi, des choix de composition peuvent être réalisés durant l'exécution en accord avec le contexte : nous appelons cette propriété de la composition la *flexibilité*. C'est cette problématique de la composition automatique flexible que nous abordons dans cette thèse.

3. L'utilisateur en intelligence ambiante n'est pas censé être expert en composition, et ses capacités peuvent être limitées : connaissances approximatives du contexte, et notamment des services disponibles ; inaptitude à composer des services entre eux ; compréhension partielle de la composition globale ; *etc.*

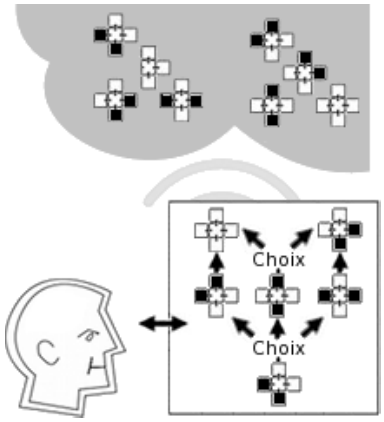


FIGURE 1.4 – Composition flexible des systèmes disponibles : la composition répond à l'objectif de l'utilisateur de cinq façons différentes. L'utilisateur et/ou le système pourront choisir la meilleure composition compte tenu du contexte d'usage.

1.1.1 Flexibilité des compositions

Dans l'exemple du malade, la flexibilité de la composition se traduit, par exemple, par la possibilité de choisir entre *Skype* et *iCall*. La figure 1.4 propose une composition des services disponibles, avec des choix (à la charge de l'utilisateur, ou du système) parmi plusieurs compositions qui répondent à l'objectif : quel que soit le choix, l'objectif sera atteint. Ainsi, pour répondre aux incertitudes sur le contexte d'usage, la composition proposée se traduit par l'enchaînement de l'utilisation des services formant la composition, avec des structures de contrôle de l'exécution permettant d'adapter la composition au contexte. Les choix ne peuvent pas être anticipés lors de la conception de la composition. Ainsi, dans un contexte où tous les services à composer n'offrent aucun choix, notre problématique revient à apporter dans la figure 1.4 la flèche de l'utilisateur vers le service composé, qui symbolise le choix de l'utilisateur : la composition n'impose notamment pas des décisions que l'utilisateur est capable de prendre à l'exécution. Cette propriété est qualitative, et se rapporte à l'*utilisabilité, en contexte,* de la composition.

1.1.2 Verrous du problème

La composition automatique et flexible est difficile à réaliser du fait de :
- la combinatoire liée aux nombreux services qui peuvent être composés ;
- la combinatoire liée aux structures de contrôle que doit intégrer la composition pour être flexible.

Dans les disciplines de l'intelligence artificielle, la planification [GNT04] vise le développement d'algorithmes produisant une composition d'actions élémentaires à accomplir pour résoudre un problème donné, l'objectif. Les algorithmes de planification automatique sont des planificateurs. La principale motivation de l'utilisation des planificateurs est de combattre la combinatoire. Aussi, nous explorons comme approche la planification automatique.

1.2 Approche par planification automatique

La figure 1.5 présente le découpage conceptuel de notre vision du problème. Un composeur, selon un contexte et un objectif imprévu de l'utilisateur, produit une composition à

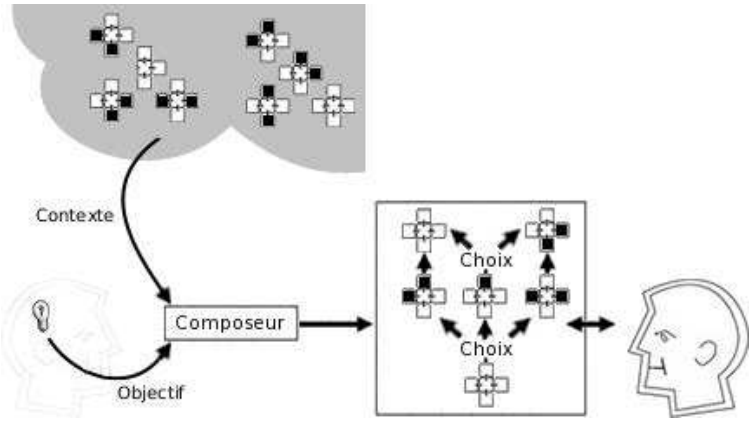


FIGURE 1.5 – Décomposition conceptuelle du problème de composition automatique flexible : l'utilisateur en contexte exprime ses besoins, un composeur génère un système composé qui réponde aux besoins de façon flexible, c'est-à-dire en intégrant des structures de contrôle.

partir des services disponibles qui d'une part répond à l'objectif et d'autre part est adaptable à l'exécution grâce à sa flexibilité. Nous proposons comme composeur un planificateur.

1.2.1 Modèle conceptuel

Le modèle conceptuel de la planification est formé de trois éléments [GNT04], en interaction :

$$\xrightarrow{\text{problème}} \text{Planificateur} \xrightarrow{\text{plan}} \text{Contrôleur} \underset{\text{observation}}{\overset{\text{action}}{\rightleftharpoons}} \text{Système de transition d'états} \xleftarrow[\text{évènement}]{}$$

- un *système de transition d'états* représente la dynamique du monde sous la forme d'un ensemble d'*états* possibles et d'*actions* permettant de changer d'état ;
- un *planificateur*, qui, pour un *problème* composé d'une description du système de transition d'états (le domaine de planification), d'un état initial et d'un objectif utilisateur, synthétise un *plan* d'actions permettant d'atteindre l'objectif.
- un *contrôleur* choisit une *action* à déclencher selon l'état du système de transition d'états et conformément au *plan* fourni par le *planificateur* ;

La comparaison de ce modèle conceptuel avec celui de notre problème de composition automatique flexible est détaillé dans les tableaux 1.1, 1.2 et 1.3. Dans la suite, nous rapprochons les concepts du problème de composition automatique flexible et ceux de l'approche par planification automatique.

Intelligence Ambiante	Planification
monde (physique et informatique)	système de transition d'états
exécuteur	contrôleur
composeur	planificateur

TABLE 1.1 – Mise en correspondance de la planification automatique avec la composition automatique flexible.

Dans notre approche, le système de transition d'états a pour rôle de représenter la dynamique du monde. Chaque état dans le système de transition d'états correspond alors à

un état du monde, dans lequel l'action effectuée par chaque service est définie, le contexte de l'utilisateur est connu, *etc.* Un état est ainsi un ensemble de propriétés sur l'ensemble des entités du monde : sur l'utilisateur (son identité, sa forme physique, *etc.*), sur les services (disponibilité des *Pages Jaune*, crédit disponible dans *Skype*, *etc.*), sur l'environnement physique (notamment la liste des médicaments dans la pharmacie, les véhicules prêts à transporter le malade ou encore l'adresse du patient), *etc.*

Le contrôleur décide quelle action appliquer sur le système de transition d'états selon le plan fourni. Aussi, il correspond à l'exécuteur de la composition en intelligence ambiante : l'utilisateur, le système et les services utilisés tiennent ce rôle.

Finalement, le planificateur est le composeur de la figure 1.5. C'est lui qui fournit la composition à partir d'une description du problème.

Nous comparons dans la suite les entrées et sorties disponibles pour composer dans les deux domaines : composeur flexible en intelligence ambiante et planificateur.

Intelligence Ambiante	Planification
système (service web, objet communiquant, *etc.*)	action
contexte	état initial
objectif	objectif

TABLE 1.2 – Mise en correspondance des éléments disponibles pour calculer la composition, d'une part en intelligence ambiante et d'autre part en planification automatique.

Tous les services disponibles et que l'utilisateur peut manipuler sont décrits sous la forme d'actions en planification : un service permet de réaliser une action. Une action appliquée sur un état permet d'atteindre un nouvel état. Le contexte de l'utilisateur est l'état du monde, l'état initial en planification. L'objectif de planification est un ensemble de propriétés à atteindre. Par exemple, le fait d'être soigné est une propriété du malade, définie dans chacun des états : l'utilisateur est soit malade soit bien portant. La composition doit donc permettre – par son application sur l'état initial – d'atteindre un état dans lequel les propriétés attendues sont satisfaites.

1.2.2 Planification flexible

Intelligence Ambiante	Planification
composition flexible de services	plan d'actions

TABLE 1.3 – Mise en correspondance de la composition flexible en intelligence ambiante et en planification.

Le plan d'actions en planification correspond à la composition de services de notre problématique. Aussi, nous proposons d'intégrer des structures de contrôle dans les plans d'actions pour permettre au contrôleur d'effectuer des choix à l'exécution.

Pour illustrer notre approche, nous détaillons le domaine de planification GRIPPER [BPG09] : *un robot peut attraper une balle (l'opérateur de planification* `PICK`*), la relâcher (*`DROP`*) et changer de pièce (*`MOVE`*). Dans l'état initial, une première pièce (notée* `r-a`*) contient 3 balles (*`b-1`*,* `b-2` *et* `b-3`*), et un robot se situe dans une seconde pièce (*`r-z`*).* L'utilisateur a pour objectif de *déplacer les 3 balles de la pièce* `r-a` *vers* `r-z` au moyen du robot. La figure 1.6

illustre ce problème. La description de GRIPPER est proposée en annexe B.3 page 186 en

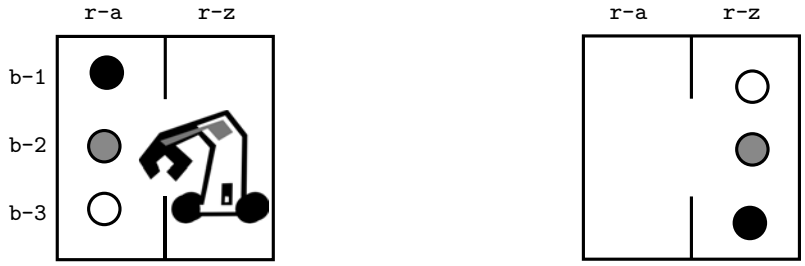


FIGURE 1.6 – Etat initial et objectif de GRIPPER :
(a) L'état initial positionne les trois balles dans la pièce `r-a` et le robot dans `r-z` ;
(b) L'objectif de planification est d'obtenir les trois balles dans la pièce `r-z`.

Planning Domain Definition Language, PDDL [AD05]. GRIPPER sera utilisé comme cas d'étude tout au long de ce manuscrit, avec les domaines de planification GRIPPER-CART, PAINTER et GRIPPER-B+C détaillés en annexe A.

L'action `PICK(b-1, r-a)` signifie que le robot attrape la balle `b-1` dans la pièce `r-a`, l'action `MOVE(r-a, r-z)` que le robot se déplace de la pièce `r-a` à la pièce `r-z` et enfin `DROP(b-1, r-z)` que le robot relâche la balle `b-1` dans la pièce `r-z`. Ces trois actions forment une partie d'un plan solution, puisque elles permettent de déplacer la balle `b-1` de `r-a` à `r-z`. En planification classique, un plan solution serait la séquence d'actions du plan 1.1. Il n'y a

Plan 1.1 Plan solution séquentiel pour GRIPPER.

```
1  MOVE(r-z, r-a);
2  PICK(b-1, r-a);
3  MOVE(r-a, r-z);
4  DROP(b-1, r-z);
5  MOVE(r-z, r-a);
6  PICK(b-2, r-a);
7  MOVE(r-a, r-z);
8  DROP(b-2, r-z);
9  MOVE(r-z, r-a);
10 PICK(b-3, r-a);
11 MOVE(r-a, r-z);
12 DROP(b-3, r-z);
```

dans ce plan aucun choix à réaliser : aucune structure de contrôle ne permet à l'utilisateur d'interagir avec la composition. En introduisant une nouvelle primitive `choose` qui propose à l'utilisateur de faire un choix, nous présentons par le plan 1.2 une composition flexible d'actions qui permet d'atteindre l'objectif quels que soient les choix effectués à l'exécution. La représentation adoptée utilise des itérations pour permettre les choix successivement offerts à l'exécution. La représentation dépliée de ces itérations, sous la forme d'un arbre de décision, fait apparaitre des branchements dus aux choix : pour chaque choix possible, une nouvelle branche apparait. Ce plan est dit flexible puisque il répond au besoin de l'utilisateur tout en laissant libre l'ordre de traitement des balles, en proposant une itération sur un traitement commun à toutes les balles. Ainsi, dans le plan séquentiel, des choix non nécessaires sont effectués par le planificateur : l'ordre de traitement des balles imposé

Plan 1.2 Plan solution pour GRIPPER proposant des choix à l'exécution.

```
1 B ← {b-1, b-2, b-3 };
2 while B is not empty do
3 |   MOVE(r-z, r-a);
4 |   choose ball ∈ B ;
5 |   B ← B − ball ;
6 |   PICK(ball, r-a);
7 |   MOVE(r-a, r-z);
8 |   DROP(ball, r-z);
```

par le premier plan n'est pas justifié par la définition du problème de planification, mais est le résultat d'une limite des planificateurs actuels qui, nous le montrerons, ne peuvent pas produire du non déterministe en sortie lorsque les entrées sont déterministes. Notre problématique se traduit dans le domaine de la planification en terme d'expressivité du plan, dans cet exemple avec l'ajout de la primitive de choix et de la structure itérative.

1.2.3 Positionnement de notre approche

Nos contributions à la problématique de la composition automatique flexible sont les suivantes :

- un modèle de la flexibilité en planification (*cf.* partie II) ;
- un planificateur (*cf.* partie III) qui calcule des plans flexibles permettant notamment :
  - les itérations successives (comme dans GRIPPER-CART, annexe A.1) ;
  - les itérations imbriquées (comme dans PAINTER, annexe A.2) ;
  - les itérations parallèles (comme dans GRIPPER-B+C, annexe A.3).

La planification s'analyse à partir des propriétés suivantes [GNT04] :

le système de transition d'états est fini : un nombre fini d'états décrit le monde et sa dynamique ;

le système de transition d'états est complètement observable : chaque état est entièrement décrit et connu ;

le système de transition d'états est déterministe : pour chaque état e et pour chaque action a, l'application de a sur e mène dans un unique état du système de transition d'états ;

le système de transition d'états est statique : l'ensemble des événements est vide, et le système de transition d'états n'a pas de dynamique interne ;

le temps est implicite : les actions et les événements n'ont pas de durée ;

la planification est « hors ligne » : le planificateur n'est concerné par aucune modification qui peut se produire dans le système de transition d'états durant le temps de planification ;

le plan est séquentiel : un plan solution en tant que composition d'actions est un ensemble totalement ordonné d'actions ;

l'objectif est déclaratif : un ensemble de propriétés de l'état souhaité décrit l'objectif de planification lorsque des propriétés sur la composition du plan pourraient être exprimées ;

Cette grille de lecture permet de situer les travaux en planification. Par exemple, des travaux s'intéressent à considérer un système de transition d'états non déterministe, d'autres proposent des algorithmes pour manipuler implicitement la durée des actions, *etc.*

Dans cette thèse, c'est la propriété des plans séquentiels qui est levée par l'intégration de structures de contrôle destinées à fournir des choix au contrôleur durant l'exécution du plan. Un tel plan est dit flexible. Toutes les autres propriétés ne concernent pas directement notre problème de composition automatique flexible. Le tableau 1.4 récapitule nos hypothèses de travail et leur caractère nécessaire ou simplificateur vis à vis de notre problème.

1.3 Structure du manuscrit

Le manuscrit est divisé en quatre parties.

La première concerne le positionnement de la thèse. Elle est constituée de cette introduction sur la composition automatique flexible. Le chapitre 2 est une étude bibliographique des travaux proposant de la composition automatique, suivie au chapitre 3 d'une étude des différents domaines de planification qui fournissent des plans « flexibles ».

La seconde partie comporte nos propositions théoriques. Le modèle de planification dans lequel s'inscrit la thèse est formalisé dans le chapitre 4. La planification flexible est définie dans le chapitre 5 comme une extension du modèle formel du chapitre 4, notamment par l'utilisation dans les « plans flexibles » des opérateurs de séquence et de choix.

La troisième partie expose nos propositions pratiques. Dans le chapitre 6, nous définissons à partir des opérateurs de séquence et de choix deux autres opérateurs : l'entrelacement et l'itération. Nous nous référons à ce cadre théorique pour délimiter la flexibilité traitée par notre planificateur Λ-`GraphPlan`. Nous donnons le détail du planificateur Λ-`GraphPlan` dans le chapitre 7. Ses performances sont évaluées et discutées au chapitre 8.

La quatrième partie discute des perspectives et conclut la thèse.

	Hypothèse	Explication
✗	système de transition d'états fini	dans un contexte d'informatique ambiante, le nombre d'états peut être considéré comme infini
✗	système de transition d'états complètement observable	il est possible de rendre compte des limites de l'observation par une description partielle ou trivaluée des états
✗	système de transition d'états déterministe	le monde physique n'est pas déterministe
✗	système de transition d'états statique	des événements peuvent survenir durant la composition ou pendant l'exécution de la composition
✗	temps implicite	cela signifie que chaque action est atomique, ce qui n'est pas le cas des systèmes à composer
-	planification « hors ligne »	la planification « en ligne » se justifierait en considérant que les objectifs changent au fil du temps, ou que le système de transition d'états est dynamique
✓	plan « flexible »	donner des choix à l'exécution est le centre de notre problématique
✓	objectif déclaratif	l'objectif ne peut pas être exprimé en terme de composition ou de propriétés sur la composition, seulement comme des propriétés sur l'état à atteindre

TABLE 1.4 – Ces hypothèses de planification sont celles qui sont utilisées dans l'approche de résolution du problème de composition automatique flexible en contexte d'intelligence artificielle. Le symbole de croix indique les hypothèses simplificatrices vis à vis de la problématique, et le symbole de validation indique les hypothèses nécessaires.

Chapitre 2

État de l'art pour la composition dynamique

Les *services* sont des éléments logiciels conçus notamment pour être composés entre eux. Cet état de l'art présente les travaux qui se sont intéressés à la composition automatique de services. Ces travaux répondent notamment au problème lié à la multiplication des services, de plus en plus complexe à composer à la main et nécessitant une automatisation du processus. Nous verrons que ces compositions souffrent du manque de flexibilité.

2.1 Services comme éléments de composition

Nous trouvons dans la littérature de nombreuses définitions de ce qu'est un service, comme, par exemple, celle du W3C [1] [W3C] :

> « A Web service [2] is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. »

Pour nous, un service est simplement une unité logicielle offrant des interfaces explicitement spécifiées et avec lesquelles il est possible de raisonner.

2.1.1 Fonctionnement

Nous introduisons l'architecture des services, avec la figure 2.1. Nous observons trois partenaires, communiquant grâce à des messages de type SOAP [3] :

1. *fournisseur de services* : c'est le fournisseur du ou des services, qui, à travers un fichier de description du service de type WSDL [4], permet l'invocation de celui-ci.
2. *consommateur de services* : c'est l'utilisateur du service, qui après avoir trouvé le service adéquat, l'exécute à travers le réseau.
3. *négociateur de services* : c'est un annuaire UDDI [5], c'est-à-dire le liant entre le *fournisseur de services* et le *consommateur de services*, permettant au premier de publier ses services sous forme de fichier WSDL, et au second de récupérer ces fichiers.

1. World Wide Web Consortium, à l'url http://www.w3.org/
2. Service et Web service sont considérés dans ce chapitre comme étant une même notion (les Web services sont définis comme étant des services sur le Web)
3. Simple Object Access Protocol, à l'url http://www.w3.org/TR/soap/
4. Web Services Description Language, à l'url http://www.w3.org/TR/wsdl/
5. Universal Description Discovery and Integration, à l'url http://uddi.xml.org/

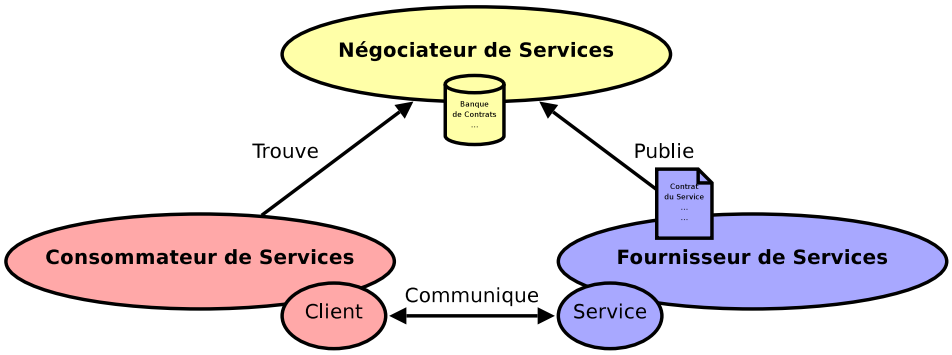


FIGURE 2.1 – Architecture décrivant le fonctionnement global des architectures orientée services.

Un processus métier (ou *workflow* [Win]) est un ensemble d'*activités*, s'exécutant dans un ordre prédéfini. Une activité peut être une exécution de code, des boucles et conditions, un appel de services, de workflow ou de méthodes externes, un minuteur, la gestion de transactions et d'erreurs... Ces activités peuvent s'enchaîner en fonction de conditions, d'interactions avec des processus informatiques ou d'interactions humaines. Le langage WSDL permet uniquement de décrire des services atomiques, qui sont alors des « activités simples ». Par l'enchaînement entre services, des activités complexes sont décrites puis réalisées.

2.1.2 Composition de services

La composition de services offre la possibilité de réaliser des activités complexes à partir de services existants. Des services composés forment un nouveau service, qui peut être réutilisé dans une autre composition.

Les services répondent aux besoins de flexibilité et d'indépendance, de partage des informations et des applications, d'inter-opérabilité et enfin de sécurité qu'ont les entreprises. De plus, les activités d'entreprises sont elles-mêmes des processus métier d'ordre supérieur, que l'on pourrait dire composées. Ainsi, l'utilisation de services complexes promet la réalisation des activités dans une organisation par l'intermédiaire de technologie standardisée de services, tout en respectant les besoins énumérés. L'espérance donnée par l'utilisation de compositions a été un conducteur important dans les efforts déployés par l'industrie dans ce domaine. Le fait que l'industrie est consciente des possibilités offertes peut être constaté par les nombreuses normes qui ont été développées dans l'objectif de modéliser ces compositions. Nous parlons notamment des langages de composition tels WS-BPEL [AAA+07], BPMN [Obj06], WS-CDL [KFB+05] et OWL-S [BvHH+04], ainsi que d'autres de plus bas niveaux tels WSCL [BBB+02], XPDL [Wor07] ou BPSS [Bus01]. La figure 2.2 présente l'évolution de certains de ces langages dans le temps et dans leurs influences. Plusieurs organismes se font de la concurrence pour proposer ces langages de descriptions, tels que le W3C, OASIS [6], OMG [7] et le WfMC [8].

6. Organization for the Advancement of Structured Information Standards, à l'url http://www.oasis-open.org/
7. Object Management Group, à l'url http://www.omg.org/
8. Workflow Management Coalition, à l'url http://www.wfmc.org/

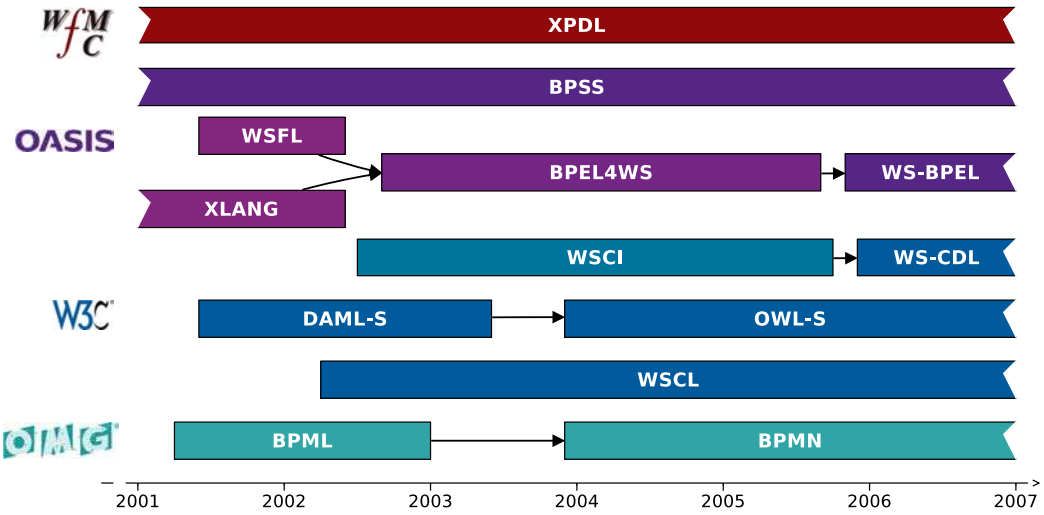


FIGURE 2.2 – Historique, évolution et influence des langages de description de composition de services, extrait de [Mar08].

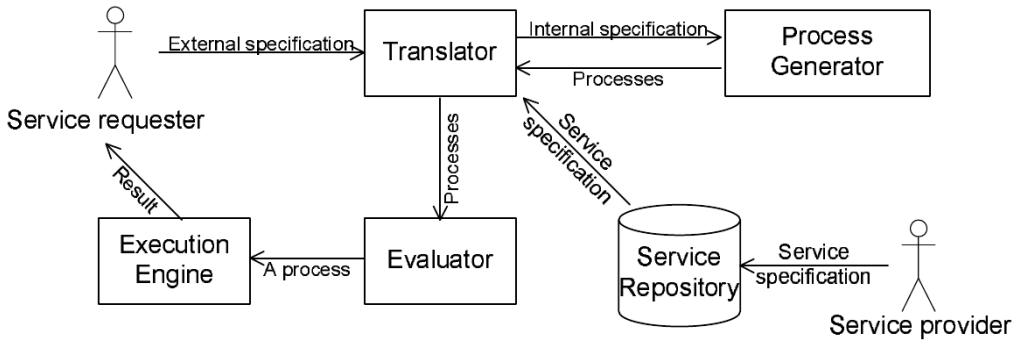


FIGURE 2.3 – Cadre général d'un système pour la composition de services, extrait de [RS04].

2.1.3 Composition dynamique de services

Les travaux traitant de la composition dynamique de services sont motivés par le nombre croissant de services et l'insuffisance actuelle des architectures orientées services : composer manuellement les services est un travail complexe, et par conséquent le nombre de compositions ainsi formé est fortement limité.
Un cadre général de la composition dynamique est proposé dans [RS04]. Nous le reproduisons à la figure 2.3. Celui-ci se découpe en :

- *service repository :* sert à présenter les services de base disponibles à la composition.
- *translator :* permet au système de composition de distinguer les spécifications internes et externes des services, pour ceux qui font cette différence. Les traductions doivent être effectuées entre :
  - le langage de description des *spécifications externes* et le langage du *process generator* (pour obtenir les *spécifications internes*) ;
  - le langage de description des *services* du *service repository* et le langage du *process generator*, et vice et versa.

- *process generator :* trouve une composition répondant à une requête ou un but que le requérant de services exprime ;
- *evaluator :* fait le choix entre les services composites proposés en les évaluant selon des critères non fonctionnels ;
- *execution engine :* exécute la composition choisie préalablement.

D'autres travaux, tels que [Pee05b] ou [HBP07], complètent ce descriptif.

2.2 Étude des approches existantes

Cet état de l'art complète un travail antérieur [Mar08]. La portée de l'étude concerne la composition automatique ou semi-automatique de services, à la demande d'un expert ou d'un utilisateur. Notre objectif est d'analyser ces travaux vis à vis de notre problématique et ses hypothèses.

2.2.1 Par calcul de situation

Golog [LRL+97] et son successeur ConGolog [DGLL00] sont des langages de programmation logique destinés aux domaines dynamiques (comme la planification) et orientés agents, dont les bases sont une version étendue du *situation calculus* [MH87].

Des adaptations de Golog et ConGolog ont été proposées pour faire de la composition dynamique de services [NM02, MS02]. Le principe de résolution est le suivant : les services disponibles, décrits en DAML-S [ABH+02], sont traduits dans un formalisme permettant de les manipuler dans le domaine de la planification. Dans ce travail, deux traductions sont opérées :

1. en Golog pour manipuler les services à un niveau logique (avec les outils dédiés au *situation calculus*) ;
2. en réseaux de Petri [Pet66] pour pouvoir utiliser les outils dédiés (vérification d'un réseau, simulation, *etc.*).

Dans [NM02], les auteurs présentent les fonctions suivantes :

- la *simulation* de l'évolution d'un service selon différentes conditions,
- la *validation* en observant ce qui est obtenu par rapport à ce qui est attendu,
- la *vérification* du maintien de certaines propriétés,
- la *composition* d'un service répondant à un but spécifique,
- l'*analyse de performances* évaluant la capacité de fournir un service dans le respect du temps, du niveau de services et d'utilisation de ressources.

La plupart de ces fonctions sont réalisées grâce à la traduction en réseau de Petri. Ceci s'explique par les preuves que peut fournir un tel modèle et ses techniques d'analyse. Ainsi, la totalité des structures de composition de DAML-S sont traduites en réseau de Petri.

En ce qui nous concerne, seule la composition est en lien avec notre problématique. La traduction en Golog est utilisée pour la réaliser. Nous trouvons des précisions quant à la notion de composition dans [NM02] :

> « Let A be a set of atomic Web services [...]. Further, let Φ represent the user's goal [...]. Then $a_1, a_2, \cdots, a_n$ $[ndla :] \in A$ is a sequential composition of atomic services that achieves user goal Φ [...]. »

Nous notons que la composition fournie est une séquence de services atomiques : c'est une limitation du point de vue de la flexibilité, puisque la structure de la composition fournie ne permet aucune alternative. Cette définition du problème est étendue dans [MS02], et les services disponibles sont décrits dans des *procédures* en ConGolog. Ces procédures

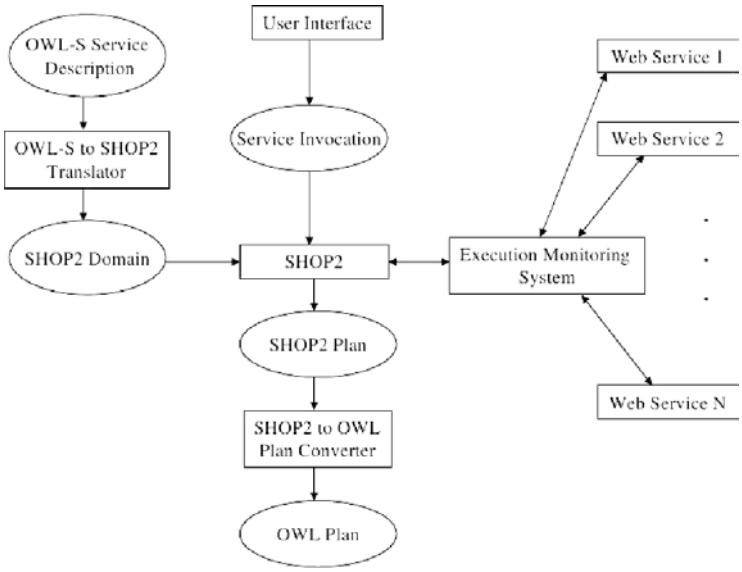


FIGURE 2.4 – Architecture de [SPW+04].

permettent de répondre de façon générique aux objectifs de l'utilisateur : une procédure existe pour chacun des objectifs de l'utilisateur. ConGolog permet notamment d'exprimer la concurrence. L'utilisateur décrit formellement ses préférences et contraintes, qui sont ensuite utilisées pour la recherche d'une composition permettant de respecter ses contraintes.

2.2.2 Par décomposition hiérarchique de l'objectif

Nous présentons ici trois approches différentes qui utilisent une décomposition explicite de l'objectif. Cet objectif est décrit sous la forme d'un modèle hiérarchique du service attendu par l'utilisateur.

Première approche

Après 2003, à l'université du Maryland, ont été menées de nombreuses recherches sur la composition automatique [WPS+03, SPW+04] et semi-automatique [SHP03]. En semi-automatique, un système d'aide à la composition a été proposé : une IHM affiche les services disponibles et l'utilisateur construit sa composition. Ce système se limite à une aide à l'utilisateur. L'utilisateur est supposé être un expert en composition. En automatique, la méthode de résolution par Golog est reprise, mais en utilisant le planificateur SHOP2 [NAI+03, NAC+]. Le travail effectué s'inspire de [NM02] et se positionne par rapport à celui-ci. La figure 2.4 présente l'architecture générale adoptée.

Une grande partie des structures de OWL-S [BvHH+04] a été traduite en opérateurs et méthodes HTN [JDKR]. Pour ce qui est du système inspiré de Golog, il se heurte aux mêmes limitations : seuls des plans séquentiels sont fournis par SHOP2. De plus, une autre limitation de ce travail apparait, due à l'utilisation de SHOP2 : la concurrence ne peut pas y être exprimée.

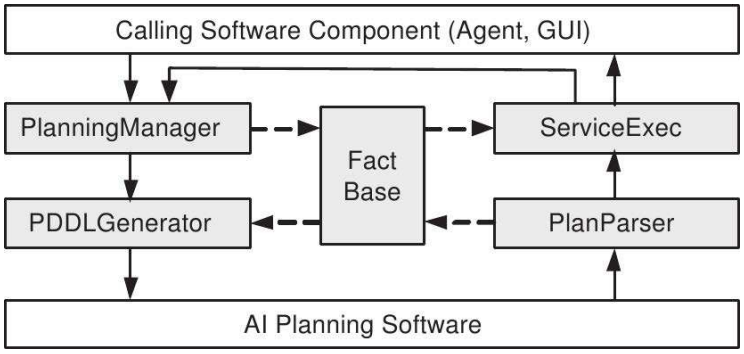


FIGURE 2.5 – L'architecture de WSPlan, un système de planification de services, extraite de [Pee04].

Critères ergonomiques

Nous avons évoqué en introduction (*cf.* chapitre 1) le cas d'étude d'un malade issu de [Gab11]. Dans ces travaux, ce n'est pas une composition de service qui est effectuée mais d'interface homme-machine. Le centre des préoccupations est donc lié au respect des critères ergonomiques de la composition calculée. Une décomposition de l'objectif est définie en HTN [JDKR], un langage de description hiérarchique du domaine de planification. Le planificateur du type de SHOP2 calcule alors une instance de la décomposition qui respecte le contexte à l'exécution.

Composition distribuée

Dans [PF09] l'innovation vient de la distribution du calcul du plan : chaque service est un agent qui peut composer de façon autonome avec les autres services. Là aussi HTN est utilisé pour décrire la décomposition hiérarchique (totalement ordonnée dans leur cas) de l'objectif de l'utilisateur. Le plan calculé souffre donc des limites déjà mentionnées de l'approche par décomposition de l'objectif à la conception.

2.2.3 WSPlan, planificateur d'ordre partiel

Joachim PEER de l'université de S^{t}Gallen en Suisse propose une nouvelle vision concernant la composition automatique de services. Son approche diffère surtout des autres par sa notion de re-planifications à l'exécution. L'architecture adoptée, illustrée figure 2.5, montre le fonctionnement général du système développé [Pee04], WSPlan (celui-ci n'est, à ce jour, pas disponible au public). Cette vision du problème se fonde sur l'idée qu'un planificateur ne peut pas être suffisamment expressif pour répondre à toutes les exigences de la composition de services [Pee04] :

> « In general, it seems not feasible to develop one monolithic (AI-) planning solution that fits all the possible requirements of web service composition. Instead, we argue that it is more fruitful to create a flexible non-monolithic framework, which allows to plug in those planners that are best suited for certain planning domains and tasks ; this is essential for addressing changing or new requirements of the users. »

Le planificateur en question, présenté dans [Pee05a], fournit un plan séquentiel. Certaines connaissances du domaine ne sont pas encodées à la main comme dans la plupart des

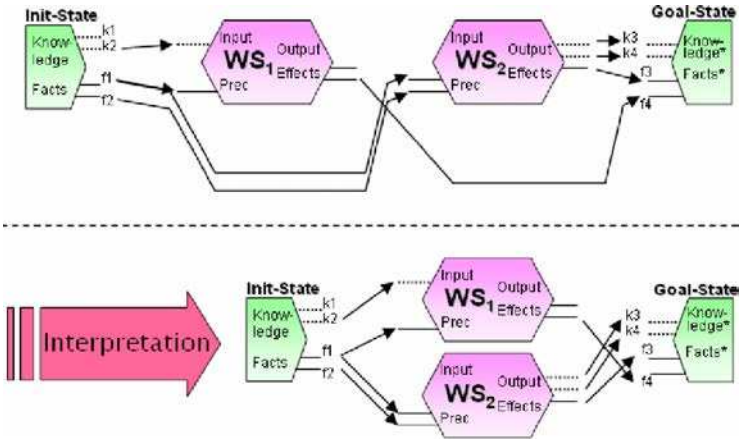


FIGURE 2.6 – Technique d'interprétation a posteriori du plan pour obtenir des compositions complexes, ici d'une séquence de services à un `<Split + Join>` du langage de description OWL-S [KGS05].

solutions proposées : elles sont générées par l'interaction dynamique avec les services, c'est-à-dire durant l'exécution du plan en gardant la possibilité de re-planification. Cette vision de la composition de services, avec re-planification, permet de surmonter l'indéterminisme de l'exécution de certains services (par exemple concernant les ressources disponibles, ou non, au moment de la planification).

2.2.4 CASCOM

CASCOM [CAS04] est un projet européen finalisé (2004-2007). Son objectif principal est décrit sur la page du projet [KHK] :

> « The main objective of the project is to implement, validate, and trial value-added supportive infrastructure for business application services for mobile workers and users across mobile and fixed networks. »

Pour cela, il propose entre autre de la composition dynamique de services. Nous trouvons la description générale de l'outil de planification, XPlan, développé dans le cadre de ce projet dans [KGS05] :

> « OWLS-Xplan consists of several modules for preprocessing and planning. It takes a set of available OWL-S services, a domain description consisting of relevant OWL ontologies and a planning query as input, and returns a plan sequence of composed services that satisfies the query goal. »

Nous retrouvons une composition de services exprimée par un plan séquentiel. Une solution est proposée pour obtenir une expressivité supérieure, illustrée par la figure 2.6. Elle repose sur une interprétation du plan a posteriori pour permettre la transformation de la séquence en des structures plus complexes. Le principe, dans le cas de l'illustration, est le suivant :

- les entrées et préconditions de WS_1 ne dépendent en rien des sorties et effets de WS_2, et inversement. Par contre, leurs entrées et préconditions dépendent du même état. Ces deux remarques permettent d'affirmer que ces deux services peuvent s'exécuter parallèlement, avec un `<Split>` (opérateur de OWL-S [BvHH+04]) ;
- de plus, nous voyons que les sorties et effets de ces deux services permettent d'atteindre un certain état, ce qui se traduit par un `<Join>` entre les deux services ;

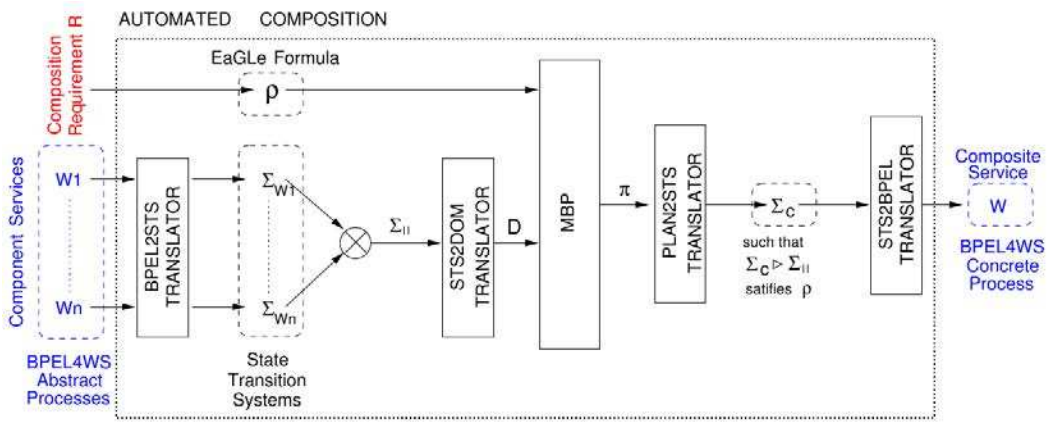


FIGURE 2.7 – L'architecture d'ASTRO, un système de planification de services. Extrait de [PTB05].

– ces deux dernières remarques provoquent finalement un `<Split + Join>` entre WS_1 et WS_2.

Un autre outil, OWLS2PDDL, permet la traduction de toutes les structures contenues dans OWL-S en PDDL [AD05], un langage de description des domaines de planification. Malheureusement, Xplan n'est sémantiquement pas assez riche pour permettre toutes les interprétations a posteriori possibles [KGS05] :

> « However, the plan structures "choice" and "unordered sequence" are not realizable by proper interpretation of plan sequences created by Xplan. Though, the latter problem is a hard problem for any AI planner in general, including, for example, Shop2. »

Les structures de choix que nous cherchons à intégrer dans les plans ne sont donc pas traitées par [CAS04] (ni par le reste des propositions). Il est souligné dans cette citation la difficulté inhérente à notre problématique, puisque nos exemples de plans flexibles issus de GRIPPER présentent tous des « séquences non ordonnées d'actions ».

2.2.5 ASTRO

À l'université de Trente, en Italie, avec notamment le chercheur Marco PISTORE ou encore Paolo TRAVERSO, s'est construit le projet ASTRO [TP04, PBB+04, PTB05] qui a pour but de développer des outils soutenant l'évolution et l'adaptation des services distribués au cours de leur cycle de vie, de la conception à l'exécution, pour finalement automatiser la composition de ces services. La méthode de résolution est illustrée par la figure 2.7. Son avantage et son apport concernent le type de planificateur utilisé. En effet, après avoir traduit les services en automate (STS pour « State Transition System »), les techniques fournies par le *model checking* sont appliquées à travers un planificateur utilisant cette technologie (MBP pour « Model Based Planner »). Le pouvoir expressif de ce dernier en termes de structures de plan est bien supérieur aux autres, grâce à l'utilisation de conditions, qui permet d'introduire d'autres structures que la séquence (de façon implicite, puisque le plan est une *politique*, c'est-à-dire un ensemble de relation entre un état atteint et le service suivant à exécuter). Le tableau 2.1 page ci-contre est un exemple de politique. Cette politique est modélisé comme un automate : à un état [9] e et une observation o est associée une action

9. L'article original utilise la notion de *contexte*, connexe à celle d'état.

$e \in E$	$o \in O$	$\alpha(e,o)$	$\epsilon(e,o) \in E$
e_0	–	AcceptDoP&SRequest	e_1
e_1	product $\neq$ Prod_1	AnswerDoP&SRequest(NoAvailable)	done
e_1	product = Prod_1	DoProductRequest(Prod_1)	e_2
e_2	na = NoAvailable	AnswerDoP&SRequest(NoAvailable)	done
e_2	$\neg$ acceptable(duration)	AnswerDoP&SRequest(NoAvailable)	e_3
e_2	acceptable(duration)	AnswerDoP&SRequest(Cost_2,Dur_2)	e_4
e_3	–	RefuseProductOffer	done
...	...	...	...

TABLE 2.1 – Fragment d'un plan fourni par le planificateur d'ASTRO, extrait de [TP04]

$\alpha(e_i, o)$ à appliquer, ce qui mène à l'état $\epsilon(e, o)$. Pour un même état, plusieurs transitions sont possibles, la séquence est donc enrichie de conditions et branchements. Le plan obtenu est ensuite traduit en BPEL [ACD+03], un langage de description de services.

EaGLe est le langage de *but étendu* utilisé dans l'approche ASTRO permettant à l'utilisateur de définir ses besoins. Ceux-ci sont en fait quasiment une description de haut niveau de la composition désirée par l'utilisateur. Nous ne pouvons donc pas vraiment parler le composition dynamique, puisque celle-ci se trouve presque entièrement définie par les nombreuses contraintes fournies par le biais de la description du but utilisateur, qui est finalement un expert EaGLe. De plus, la connaissance préalable des services disponibles, et donc du contexte à l'exécution, est obligatoire pour pouvoir exprimer des contraintes entre eux.

2.2.6 INFRAWEBS

Le projet européen INFRAWEBS [INF04] (2004–2007) avait pour objectif le développement d'une application axée sur les outils logiciels pour la création, le maintien et l'exécution de services sémantiques dans l'ensemble de leur cycle de vie. Dans [AM07], l'approche de résolution est un peu différente des techniques de planification vues précédemment. Ce n'est pas explicitement une technique de planification qui est utilisée, mais le fait qu'un but est décrit comme un ensemble d'expressions logiques pouvant être découpées en sous-ensembles : des sous-buts. Cette décomposition s'apparente à la décomposition que l'on trouve en planification hiérarchique [JDKR]. Les expressions logiques sont décrites à l'aide d'ontologie, et peuvent décrire ce que l'utilisateur pourra fournir au système, ainsi que ce qu'il attend des services capables de satisfaire le but. Pour revenir à la méthode employée, l'algorithme principal cherche un service capable de satisfaire le but. Si aucun n'est disponible, alors il décompose ce but en deux sous-buts qu'il cherche à satisfaire de façon récursive. Nous devons donc observer cette décomposition en sous-buts pour connaître la structure des compositions finales :

> « [...] application [...] begin to construct a composite goal with the most important [...] sub-goal, and continue this process by sequentially adding other sub-goals according to the degree of their importance for realization of the whole goal. »

Nous obtenons, après l'exécution de l'application, une séquence d'utilisation des services : pour chaque sous-but, les services compatibles sont proposés à l'utilisateur pour qu'il puisse sélectionner celui qui lui semble le meilleur. Comme les sous-buts sont organisés séquentiellement, les services le sont aussi.

Un système de substitution de services, dans le cas où le service préalablement sélectionné ne peut pas fonctionner, est mis en place. Lors de la sélection du service par l'utilisateur, les autres services disponibles pour le même but sont gardés en mémoire, pour permettre la substitution durant l'exécution de la composition. La décomposition en sous-buts signifie que le but de l'utilisateur ne peut pas être imprévu : une solution a déjà été conçue pour le but, l'objectif de ce système est de trouver des instances de services permettant de répondre aux sous-buts. Malgré cette première limitation, nous retrouvons une composition pauvre en structures : les sous-buts sont ordonnés séquentiellement lorsque une composition flexible demande de pouvoir proposer à l'exécution des choix à l'utilisateur. Mais dans ce cas précis, nous pouvons considérer ce point comme minime puisque l'utilisateur choisit lui-même le service à utiliser. Cela nous amène au fait que cet utilisateur est fortement impliqué dans le processus : il est co-concepteur de la composition, donc posède des compétences dans ce domaine.

2.2.7 Autres travaux

[SPAS03] propose un travail théorique couvrant en particulier la composition de services sémantiques, c'est-à-dire décrit dans des ontologies. Toutefois, l'apport principal concerne la démonstration des avantages de l'utilisation de la sémantique. Concernant l'obtention d'une composition par planification, les difficultés suivantes sont indiquées :

- le choix des capacités attendues des services, c'est-à-dire décider comment transformer une requête utilisateur précise en un but de composition abstrait ;
- le plan ne peut pas être complet tant que tous les services à utiliser ne sont pas sélectionnés ;
- le planificateur peut avoir besoin de l'exécution partielle du plan pour continuer à planifier.

Les deux dernières difficultés sont inhérentes à leur vision du problème. La première est récurrente : le souhait concret de l'utilisateur doit être compris à un niveau abstrait pour comprendre ce que doit fournir le service composite résultant.

[HB03] propose un cadre théorique pour la composition de services. Il est fondé sur les réseaux de Petri. Un travail préliminaire sur l'expressivité de ces réseaux est effectué dans le but de démontrer leur intérêt, et de montrer comment utiliser une traduction de services en réseaux de Pétri pour inférer des propriétés via leur analyse. Nous avons déjà vu des systèmes qui utilisent les réseaux de Pétri pour la composition, comme [NM02].

Dans [MBE03], les auteurs proposent de la composition dynamique de services sémantiques. Ils proposent une architecture, illustrée à la figure 2.8 page suivante, qui se découpe en quatre phases :

1. *specification* permet la description de haut niveau de la composition désirée dans le langage CSSL [10],
2. *matchmaking* utilise les règles de composition pour générer un plan conforme à la spécification du service requis.
3. *selection* choisit le meilleur plan selon la qualité de composition si plusieurs sont disponibles,
4. *generation* génère automatiquement la description du service composite, qui est présenté à l'utilisateur.

10. Composite Service Specification Langage

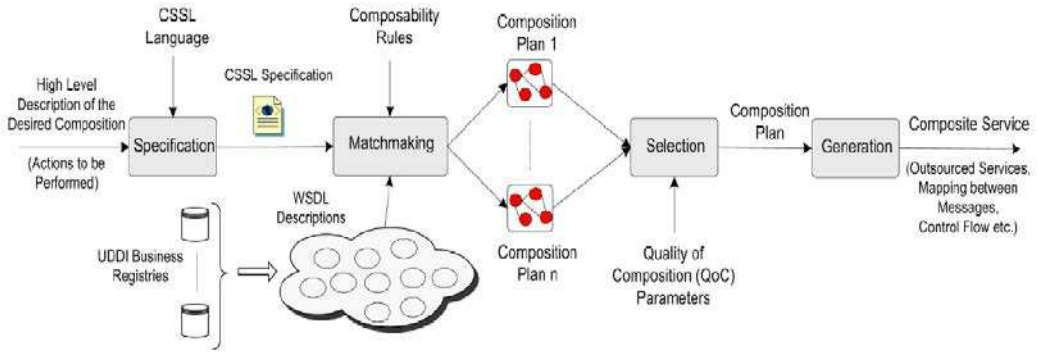


FIGURE 2.8 – Approche adoptée dans [MBE03] pour faire de la composition dynamique de services sémantiques.

Lors de la première phase, l'utilisateur décrit le service composite qu'il désire obtenir, sans connaissance réelle des services existants.

Dans [McD02], Drew MCDERMOTT introduit un nouveau type de connaissance, qui exprime le type de donnée retournée par l'application d'une action. Cette information manquerait au langage de description pour la planification. De plus, une distinction est soulignée entre les transformations d'informations et les changements d'états. Les informations représentent les entrées et sorties des services. Elles peuvent être dupliquées pour d'autres services, au contraire de l'état du monde, modifié par l'exécution d'un service : l'ancien état disparait pour laisser place au nouveau. Il affirme d'autre part que le domaine de la composition de services requiert l'*ouverture* du monde de la planification (à l'inverse de l'hypothèse habituelle du monde clos), et demande la formation de plans plus complexes.

2.3 Conclusion

La planification automatique est l'approche majoritairement adoptée pour la composition dynamique : la planification a pour objectif de structurer des actions, ici des services, sous la forme d'un plan qui permet, à l'exécution, d'atteindre un objectif exprimé. Nous notons l'existence de nombreux travaux [NM02, KGS05] sur la traduction de la description des services en domaine de planification, utilisant les langages adaptés à la planification (STRIPS [FN71], PDDL [AD05], HTN [JDKR]). Cela souligne encore la pertinence d'une approche par planification pour notre problématique. Nous considérons n'importe quel type de « service » utilisé dans le cadre de la composition automatique en intelligence ambiante, en faisant l'hypothèse qu'il existe une description de ces services dans un langage adapté à la planification.

Aucun planificateur ne présente de résultat satisfaisant selon nos critères : de façon générale, les systèmes de composition dynamique étudiés ne prennent pas en compte notre utilisateur non expert en planification et qui doit interagir avec la composition au moyen de structures de contrôle lui donnant des choix à l'exécution. En effet, une large partie des systèmes proposés [CAS04, INF04, SPW+04] s'appuie sur des planificateurs qui ne permettent pas de produire de plans flexibles. D'autres systèmes [MS02, PTB05] sont destinés à des utilisateurs experts en composition (notamment par l'expression de besoins complexes). Les travaux qui utilisent une décomposition hiérarchique de l'objectif [SPW+04, Gab11, PF09]

pour la composition nécessitent que l'objectif ait été anticipé : la décomposition décrit une solution de l'objectif (même si elle n'est pas instanciée). Or dans notre problématique l'objectif est imprévu. Aucun de ces travaux ne répond à nos attentes.

Les questions concernant les critères d'ergonomie, ou encore l'intégration de l'utilisateur, sont la plupart du temps secondaires, ou ne sont pas soulevées du tout. Notre vision, centrée utilisateur, impose une contribution théorique et pratique. Ainsi, alors qu'en planification le plan solution qui sera le plus apprécié sera celui contenant le moins d'actions et d'opérateurs possibles, la composition devra être riche en structures pour être flexible, et laisser un maximum de degrés de liberté à l'utilisateur : les algorithmes de planification ne peuvent pas être les mêmes.

Chapitre 3

État de l'art pour la flexibilité en planification

À notre connaissance, il n'existe pas d'approche en planification conçue pour une telle flexibilité, c'est-à-dire pour calculer un ensemble de plans solutions pour un problème donné. Toutefois, quelques approches sont connexes. Nous présentons les domaines de la planification produisant des plans qui intègrent des structures de contrôle ou permettant d'exprimer plus d'une unique séquence d'actions.

Un premier ensemble de travaux qui fournissent des plans structurés sont ceux qui utilisent des stratégies de moindre engagement [Wel94, MBD91], en planification classique : les actions des plans sont alors partiellement ordonnées, et le contrôleur peut choisir l'ordre d'exécution. Nous exposerons dans le paragraphe 3.1 les limites de ces approches vis à vis de la flexibilité en prenant pour exemple les cas d'études détaillés au chapitre introductif (*cf.* paragraphe 1.2.2) : les planificateurs issus de ces approches sont flexibles en partie seulement.

D'autres travaux proposent des structures dans leurs plans pour surmonter des incertitudes sur l'environnement, que ce soit un nombre variable d'objets à traiter, un état initial partiellement connu ou encore des actions aux effets non déterministes. La planification généralisée [HG11], la planification conformante [Bon09] et la planification conditionnelle [PS92] ainsi que certaines approches de planification fondées sur modèle [GT00] (MBP [1]) surmontent ces incertitudes en intégrant des structures dans les plans. Nous montrons dans le paragraphe 3.2 en quoi leurs hypothèses et les nôtres sont incompatibles.

3.1 Planification partiellement flexible

Nous nous intéressons dans ce paragraphe aux différentes approches existantes en planification classique (*cf.* figure 3.10 page 43 sur la taxinomie de la planification). Dans un espace d'états, les plans sont pauvres en structures, mais c'est souvent une approche relativement peu couteuse en ressource et en temps. Par exemple, le planification Fast-Forward [HN01] (FF) est un planificateur qui calcule des séquences d'actions : FF est vainqueur de compétitions de planification classique (comme à IPC [2] en 2002 [LFS+02]). Lorsque la planification navigue dans des espaces de plans, produisant des plans partiellement ordonnés [Wel94], ou dans des graphes, produisant des ensembles ordonnés d'actions [BF97], les performances diminuent mais l'expressivité des plans augmentent.

Nous présentons au paragraphe 3.1.1 la planification dans un espace de plans. Notre proposition algorithmique étant fondée sur la planification de graphe, nous la détaillerons particulièrement dans le paragraphe 3.1.2.

1. Model Based Planning
2. International Planning Competition

3.1.1 Planification partiellement ordonnée (POP)

Un plan partiellement ordonnée est un ensemble d'actions, et un ensemble de contraintes d'ordre entre les actions. Seules les contraintes nécessaires sont dans le plan, de telle façon que l'exécution des actions dans n'importe quel ordre qui respecte les contraintes assure, d'une part, que toutes les actions seront exécutables et, d'autre part, que l'objectif sera atteint. Les premiers travaux [Sac75, Tat77, PW92] amenant à la planification partiellement ordonnée sont issus de réflexions sur la planification « non linéaire ».

En planification partiellement ordonnée, l'espace de recherche est un espace de plans partiels. Un plan partiel contient notamment un ensemble d'actions[3], un ensemble de contraintes d'ordre $\prec$ entre couple d'actions (par exemple, $a_1 \prec a_2$ signifie que a_1 doit s'exécuter avant a_2) et des liens de causalité [Tat77] entre couple d'actions. L'ordre partiel entre les actions définit un graphe orienté : il est consistant lorsque le graphe ne contient pas de cycle. Les liens de causalité permettent de lier les effets d'une première action aux préconditions d'une seconde : par exemple, $a_1 \xrightarrow{p} a_2$ signifie que p est un effet de a_1 et une précondition de a_2, et que $a_1 \prec a_2$. On dit qu'une action a_m *menace* le lien causal $a_1 \xrightarrow{p} a_2$ si p est un effet négatif de a_m alors que les contraintes d'ordre $a_1 \prec a_m$ et $a_m \prec a_2$ sont consistantes vis à vis de $\prec$. Dans un plan partiel, une précondition qui n'est pas impliquée dans un lien de causalité est un sous-objectif « ouvert ».

L'espace de recherche est tel que :

- les noeuds sont des plans partiels ;
- les arcs sont des opérations de raffinement du plan qui ont pour objet de compléter le plan partiel, c'est-à-dire de réaliser un sous-objectif resté ouvert par l'ajout d'un lien de causalité ou de supprimer une menace par l'ajout d'une contrainte d'ordre.

La planification dans un tel espace de recherche commence avec un plan vide qui est le noeud initial. Dans ce plan vide, l'état initial est modélisé par une action virtuelle a_0 qui a toutes les propositions de l'état initial comme effets ; et l'objectif de planification est modélisé par une action virtuelle a_∞ qui a pour préconditions cet objectif. Une contrainte d'ordre relie a_0 et a_∞ de telle façon que l'état initial précède l'objectif : $a_0 \prec a_\infty$. Pour faire évoluer l'espace de recherche, les deux principales opérations à réaliser sont le choix des actions à ajouter aux plans partiels, et les relations (contraintes d'ordre ou liens de causalité) à ajouter entre les actions. Le noeud a atteindre doit être un plan partiellement ordonnée tel qu'aucun objectif n'est ouvert et qu'aucun lien de causalité ne soit menacé par une action du plan : ces conditions garantissent que le plan est solution du problème de planification.

Avec Gripper-B+C (*cf.* annexe B.6), un plan partiellement ordonné solution propose un ordre total pour le déplacement de toutes les balles par le robot `g-b` et un ordre total pour le déplacement de tous les cubes par le robot `g-c`. Par exemple, un tel plan solution pour une unique balle `b-1` et un unique cube `c-1` est composé des contraintes d'ordre suivantes :

- `PICK(g-b, b-1, r-a)` $\prec$ `MOVE(g-b, r-a, r-z)` $\prec$ `DROP(g-b, b-1, r-z)` ;
- `PICK(g-c, c-1, r-z)` $\prec$ `MOVE(g-c, r-z, r-a)` $\prec$ `DROP(g-c, c-1, r-a)`.

Ainsi, aucune contrainte d'ordre n'implique de couple d'actions de chacun des traitements : le déplacement des balles et celui des cubes sont totalement indépendants. Ce plan est donc partiellement ordonné, et toutes les séquences d'actions totalement ordonnées qui respectent $\prec$ sont solutions du problème de planification. À l'exécution, la contrôleur pourra choisir d'exécuter la séquence d'actions qui lui conviendra le mieux selon son contexte : le plan est flexible.

3. Les actions sont en vérité des *opérateurs partiellement instanciés*, et un plan partiel contient aussi un ensemble de contraintes d'instanciation sur ces opérateurs : les contraintes d'ordre nous suffisent pour illustrer les limites de la flexibilité en planification partiellement ordonnée.

Avec GRIPPER (*cf.* annexe B.3), un plan partiellement ordonné ne contient aucune flexibilité : un ordre total contraint les actions du plan, autrement dit une seule exécution séquentielle est possible. Par exemple, prenons un plan partiel de GRIPPER dans lequel se trouvent les actions `PICK(b-1, r-a)` et `PICK(b-2, r-a)`, et dans lequel un lien de causalité a été ajouté entre l'état initial et `PICK(b-1, r-a)` : $a_0 \xrightarrow{\text{FREE}}$ `PICK(b-1, r-a)`. Dans ce plan partiel, `PICK(b-2, r-a)` est une menace pour le lien de causalité. En effet, FREE est un effet négatif de `PICK(b-2, r-a)` et $a_0 \prec$ `PICK(b-2, r-a)` et `PICK(b-2, r-a)` $\prec$ `PICK(b-1, r-a)` est consistant (l'ajout de ces contraintes d'ordre ne crée par de cycle). Ainsi, pour supprimer cette menace, une contrainte d'ordre sera ajoutée au plan partiel : `PICK(b-1, r-a)` $\prec$ `PICK(b-2, r-a)`. Ainsi, dans ce plan partiel en construction, la balle `b-2` sera nécessairement attrapée par le robot après la balle `b-1` : c'est une contrainte nécessaire pour que le plan soit consistant et donc solution, mais c'est un choix arbitraire qu'impose l'approche par planification partiellement ordonnée. Un plan flexible solution pour GRIPPER doit proposer de choisir l'ordre de traitement des balles plutôt que de l'imposer. En planification partiellement ordonnée, le plan est flexible en partie seulement : la flexibilité est limitée par l'expressivité issue des contraintes d'ordre.

3.1.2 Planification fondée sur les graphes (`GraphPlan`)

Soit un langage propositionnel $\mathcal{L}$ et un ensemble d'actions $\mathcal{A}$. Chaque action a de $\mathcal{A}$ possède des préconditions notées $pre(a)$, des effets positifs notés $\mathit{eff}^+(a)$ et des effets négatifs notés $\mathit{eff}^-(a)$, avec $pre(a)$, $\mathit{eff}^+(a)$ et $\mathit{eff}^-(a)$ trois ensembles de propositions de $\mathcal{L}$. Un problème de planification est un triplet $(\mathcal{A}, e_0, goal)$ avec e_0 et $goal$ respectivement l'état initial et l'objectif de planification, deux ensembles de propositions de $\mathcal{L}$. Pour un problème donnés, un graphe de planification $\mathcal{G}$ est un couple (N, E) avec N un ensemble de nœuds et E un ensemble d'arcs. C'est une représentation condensée de l'espace d'états sous la forme de couches successives de propositions et d'actions telles que $N = \mathcal{P}_0 \cup \mathcal{A}_1 \cup \mathcal{P}_1 \cup \mathcal{A}_2 \cup \cdots$:

- une couche de propositions $\mathcal{P}_i$ contient les propositions de $\mathcal{L}$ qui sont atteignables par l'application d'un plan π de longueur i au maximum, π étant applicable sur l'état initial e_0 ;
- une couche d'actions $\mathcal{A}_i$ contient les actions de $\mathcal{A}$ qui sont applicables sur l'état représenté par la couche propositionnelle $\mathcal{P}_{i-1}$.
- les arcs relient les actions de chaque couche $\mathcal{A}_i$ à leurs préconditions dans la couche $\mathcal{P}_{i-1}$ et à leurs effets dans la $\mathcal{P}_i$.

$\mathcal{P}_0$ représente l'état initial du problème : cette couche est composée des propositions de e_0. Un exemple de graphe de planification est proposé à la figure 3.1 page suivante. Cet exemple de graphe correspond aux trois premiers pas de temps pour le problème décrit en annexe B.3, avec initialement le robot et les balles `b-1` et `b-2` dans la pièce `r-1`, le robot ayant sa pince disponible, représenté par la proposition FREE.

Structure du graphe de planification

Les préconditions d'une action $a \in \mathcal{A}_i$ sont des propositions de $\mathcal{P}_{i-1}$, alors liées à a. Ces liens sont matérialisés dans la figure 3.1 pour l'action `PICK(b-1, r-a)` [4] de la couche $\mathcal{A}_1$ par les traits solides reliant cette action à ses préconditions dans la couche $\mathcal{P}_0$.

De même, les propositions de $\mathcal{P}_i$ sont des effets positifs ou négatifs des actions de $\mathcal{A}_i$. L'action `PICK(b-1, r-a)` de la couche $\mathcal{A}_1$ est reliée dans la figure 3.1 à ses effets contenus

4. L'action `PICK(b-1, r-a)` dans le graphe représenté à la figure 3.1 signifie que le robot attrape la balle `b-1` dans la pièce `r-a`.

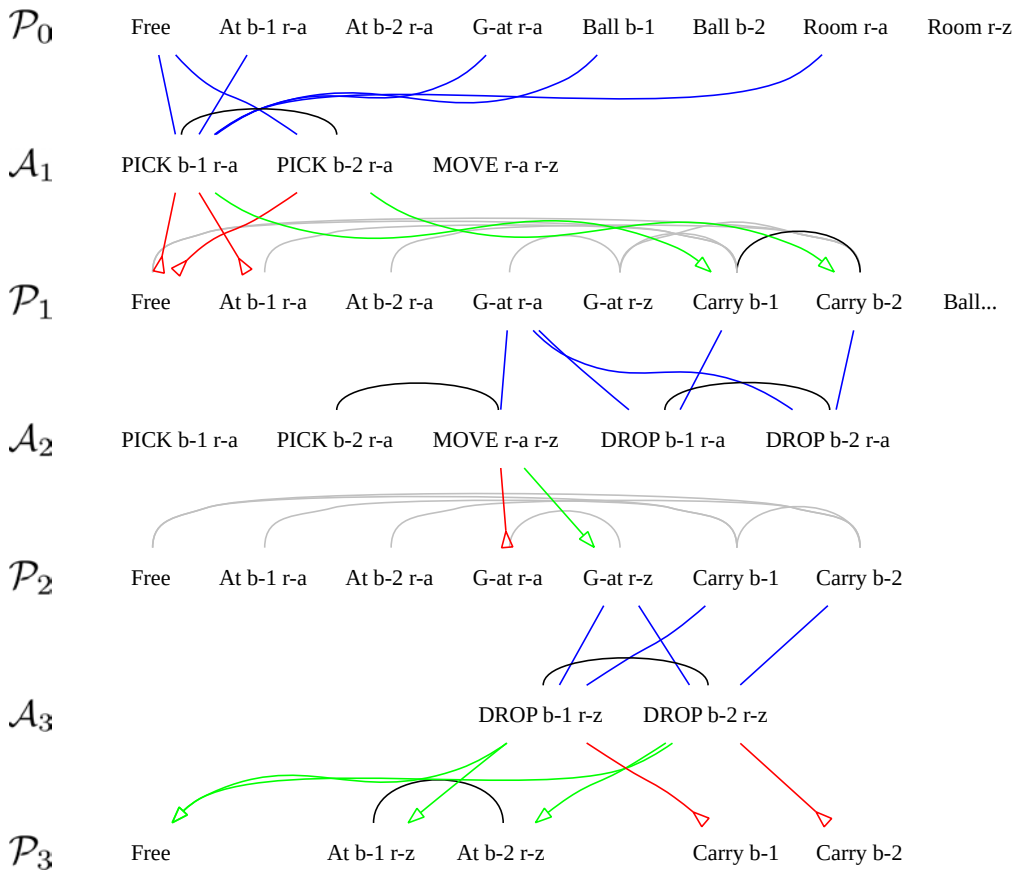


FIGURE 3.1 – Le graphe de planification pour le problème GRIPPER (limité à deux balles) après 3 étapes d'extension. Un trait entre une proposition et une action représente une précondition, et entre deux propositions ou entre deux actions, une relation mutex (les relations mutex invoquées dans le texte sont foncées) ; une flèche (resp. une flèche inversée) entre une action et une proposition représente un effet positif (resp. négatif).

dans la couche $\mathcal{P}_1$: par une flèche droite pour son effet positif [5], et par des flèches renversées pour ses deux effets négatifs [6].

De plus, chaque proposition PRED $\in \mathcal{P}_{i-1}$ est la précondition d'une action *virtuelle* de propagation NOOP-PRED de $\mathcal{A}_i$ qui ne possède qu'un seul effet positif PRED $\in \mathcal{P}_i$ et aucun effet négatif. Ces actions ne sont pas représentées dans la figure 3.1, mais nous pouvons constater qu'aucune action représentée de la couche $\mathcal{A}_1$ ne produit la proposition FREE de la couche $\mathcal{P}_1$. Cette proposition est donc présente dans la couche $\mathcal{P}_1$ grâce à l'action de propagation NOOP-FREE. Il en est de même pour toutes les autres propositions de cette couche, et des suivantes. Ainsi, $\forall i, \mathcal{P}_i \subseteq \mathcal{P}_{i+1}$.

5. L'effet positif de l'action `PICK(b-1, r-a)` est la proposition `CARRY(b-1)` qui signifie que le robot transporte la balle `b-1`.

6. Ses effets négatifs sont les propositions `FREE` (la pince du robot n'est plus libre) et `AT(b-1, r-a)` (la balle `b-1` n'est plus dans la pièce `r-a`).

Relation mutex

Deux actions a_1 et a_2 de $\mathcal{A}_i$ sont liées par une relation *mutex* (de *mut*uelle *ex*clusion) si soit un effet négatif de a_1 est une précondition ou un effet positif de a_2, soit une précondition de a_1 est liée par une relation mutex avec une précondition de a_2 dans $\mathcal{P}_{i-1}$. L'ensemble des relations mutex entre les actions de $\mathcal{A}_i$ est noté $\mu\mathcal{A}_i : \mathcal{A} \times \mathcal{A}$. Les actions sont donc mutuellement exclusives selon différents critères, qui sont :

- l'inconsistance : l'une nie l'effet de l'autre ;
- l'interférence : l'une supprime une précondition de l'autre ;
- la propagation, c'est-à-dire l'existence de ressources conflictuels : elles ont des préconditions mutex.

Deux propositions p_1 et p_2 de $\mathcal{P}_i$ sont *mutex* si toute action de $\mathcal{A}_i$ (les actions de propagation comprises) qui a pour effet positif p_1 est mutex avec toutes les actions de $\mathcal{A}_i$ qui produisent (ayant pour effet positif) p_2, et si aucune action de $\mathcal{A}_i$ ne produit simultanément p_1 et p_2. L'ensemble des relations mutex entre les propositions de $\mathcal{P}_i$ est noté $\mu\mathcal{P}_i : \mathcal{L} \times \mathcal{L}$, et $\mu\mathcal{P}_0$ est vide. Les propositions sont donc mutuellement exclusives selon différents critères, que sont :

- l'une est la négation de l'autre ;
- l'inconsistance du support : toute paire d'actions qui rend l'une et l'autre vrai est mutex.

Formellement, ces définitions se traduisent ainsi [GNT04] :

Définition 3.1 (Relation mutex entre actions)**.** *Deux actions a_1 et a_2 de $\mathcal{A}_i$ sont* mutex *si :*

- *a_1 et a_2 sont inconsistantes ou si elles interfèrent (a_1 et a_2 sont alors* dépendantes [7]*), ou si*
- *une précondition de a_1 est* mutex *avec une précondition de a_2, c'est-à-dire* $\exists\langle p_1, p_2\rangle \in \mathcal{P}_{i-1} \times \mathcal{P}_{i-1} \mid p_1 \in pre(a_1), p_2 \in pre(a_2), \langle p_1, p_2\rangle \in \mu\mathcal{P}_{i-1}$.

Définition 3.2 (Relation mutex entre propositions)**.** *Deux propositions p_1 et p_2 de $\mathcal{P}_i$ sont* mutex *si :*

- *toutes les actions de $\mathcal{A}_i$ qui ont pour effet p_1 sont* mutex *avec chaque action de $\mathcal{A}_i$ qui produit p_2, c'est-à-dire* $\forall\langle a_1, a_2\rangle \in \mathcal{A}_i \times \mathcal{A}_i \mid p_1 \in \mathit{eff}^+(a_1), p_2 \in \mathit{eff}^+(a_2), \langle a_1, a_2\rangle \in \mu\mathcal{A}_i$*, et si*
- *aucune action de $\mathcal{A}_i$ ne produit les deux propositions, c'est-à-dire* $\nexists a \in \mathcal{A}_i \mid p_1, p_2 \in \mathit{eff}^+(a)$.

Dans la figure 3.1, les actions `PICK(b-1, r-a)` et `PICK(b-2, r-a)` de $\mathcal{A}_1$ sont mutex car elles sont dépendantes à travers la proposition `FREE`, qui est l'une de leurs préconditions et l'un de leurs effets négatifs. Ainsi, leurs effets positifs respectifs `CARRY(b-1)` et `CARRY(b-2)` de $\mathcal{P}_1$ sont mutex puisque aucune autre action de $\mathcal{A}_1$ ne les produit. Pour terminer l'illustration des relations mutex, les actions `DROP(b-1, r-a)` et `DROP(b-2, r-a)` de $\mathcal{A}_2$ sont mutex car chacune a respectivement pour précondition les propositions `CARRY(b-1)` et `CARRY(b-2)` de $\mathcal{P}_1$, mutex entre elles.

Critère nécessaire d'atteignabilité

Les relations mutex sont utiles pour connaitre dans chaque couche les couples d'actions qui peuvent être exécutées simultanément, et les couples de propositions qui pourraient être

7. Les propriétés d'inconstance, d'interférence et de dépendance d'un couple d'actions seront définies formellement dans le chapitre 4.

vérifiées ensemble. Notamment, si la dernière couche de propositions $\mathcal{P}_k \in \mathcal{G}$ ne contient pas l'objectif de planification sans relation mutex entre elles, c'est qu'il n'existe aucun plan de taille k permettant de satisfaire cet objectif. Il serait donc inutile d'essayer de chercher un plan de taille k. Comme $\mathcal{L}$ contient un nombre fini de propositions, les couches propositionnelles cessent de croitre en nombre de propositions pour une certaine taille du graphe, et chaque couche au delà de cette taille contient les mêmes propositions. De plus, le nombre de relations mutex entre propositions ne peut que diminuer lorsque les couches propositionnelles successives ont atteint cette taille maximum, car deux propositions qui ne sont pas mutex dans une certaine couche ne peuvent pas devenir mutex du fait des actions de propagation (autrement dit, une paire de propositions atteignable avec un plan de longueur k peuvent être atteint avec un plan plus long). Ainsi, un objectif de planification ne pourra pas être satisfait si le graphe atteint un point fixe où les couches propositionnelles successives ne varient plus et ne contiennent pas cet objectif.

Nous notons $\mathcal{P}_g$ la première couche propositionnelle de $\mathcal{G}$ qui contient l'ensemble des propositions de l'objectif de planification *goal*, sans aucune relation mutex : $\forall \mathtt{g} \in goal, \mathtt{g} \in \mathcal{P}_g$ et $\forall (\mathtt{g}_1, \mathtt{g}_2) \in goal \times goal, (\mathtt{g}_1, \mathtt{g}_2) \notin \mu\mathcal{P}_g$. Le fait que les propositions ne soient pas mutex signifie que chaque proposition de l'objectif est atteignable avec un plan de longueur g, mais ne garantit pas l'existence d'un plan : les propositions sont atteignables de façon indépendante et non nécessairement toutes ensemble. L'apparition de $\mathcal{P}_g$ dans $\mathcal{G}$ est une condition nécessaire mais non suffisante pour qu'un plan solution soit extrait de $\mathcal{G}$.

Dans le graphe de la figure 3.1, l'objectif de planification composé des propositions `AT(b-2, r-z)` et `AT(b-1, r-a)` est atteint à la couche $\mathcal{P}_3$ (donc $g = 3$). Au contraire, l'objectif de planification décrit initialement (composé des propositions `AT(b-2, r-z)` et `AT(b-1, r-z)`) n'est pas encore consistant dans $\mathcal{P}_3$ puisque ses propositions y sont mutex : le graphe doit continuer à être étendu pour atteindre $\mathcal{P}_g$.

L'algorithme `GraphPlan` (`GP`) [BF97] se décompose en deux étapes, comme illustré à la figure 3.2. La première étape de `GP` consiste à *étendre* pas à pas le graphe de planification $\mathcal{G}$

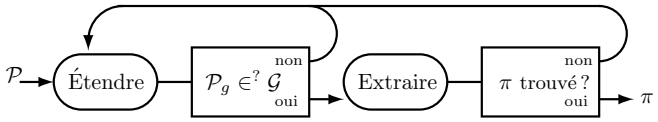


FIGURE 3.2 – Décomposition fonctionnelle de l'algorithme `GP` : pour un problème $\mathcal{P}$ donné, `GP` étend un graphe de planification $\mathcal{G}$ à partir de l'état initial et extrait un plan en couche π.

selon les règles suivantes, et dans cet ordre :

1. une action a est ajoutée à $\mathcal{A}_i$ si ses préconditions ne sont pas mutex dans $\mathcal{P}_{i-1}$, c'est-à-dire si $\nexists \langle p_1, p_2 \rangle \in pre(a) \times pre(a), p_1 \neq p_2 \mid \langle p_1, p_2 \rangle \in \mu\mathcal{P}_{i-1}$;
2. une proposition `PRED` est ajoutée à $\mathcal{P}_i$ si une action de $\mathcal{A}_i$ a `PRED` pour effet positif ou négatif.

En reprenant la figure 3.1 page 28, l'action `DROP(b-2, r-z)` ne peut pas être ajoutée dans $\mathcal{A}_2$ puisque $\mu\mathcal{P}_1$ contient une relation mutex entre les propositions `CARRY(b-2)` et `G-AT(r-z)`, deux préconditions de cette action. En revanche, $\mu\mathcal{P}_2$ ne contient pas cette relation : l'action `DROP(b-2, r-z)` est ajoutée à $\mathcal{A}_3$. Les relations mutex sont ajoutées dans le graphe de planification lors de son expansion.

Deux fonctions, `FixedPoint` et `Termination`, permettent de déterminer s'il existe une solution pour le problème à partir du graphe et d'un ensemble $\mathcal{K}$ de connaissances acquises. Cela est rendu possible par la détection du point fixe du graphe, et par le calcul des n-uplets

de propositions insolubles en tant que sous but. Ces fonctions ne seront pas détaillées ici, mais sont utilisées dans notre proposition.

Lorsque le but se trouve dans l'ensemble des faits atteignables, c'est-à-dire lorsque $\mathcal{P}_g$ est atteint, GP tente d'*extraire* de $\mathcal{G}$ un plan. Le principe de l'extraction est décrit dans le paragraphe suivant.

Extraction de plan en couche

GP extrait des plans « en couche », c'est-à-dire une succession d'ensembles d'actions. Les actions d'un ensemble sont applicables dans n'importe quel ordre.

Pendant l'extraction d'un plan, GP s'assure que chaque proposition sélectionnée d'une couche $\mathcal{P}_i$ est produite par au moins une action de la couche $\mathcal{A}_i$. Les actions qui produisent les propositions sélectionnées dans une couche doivent former un ensemble sans relation *mutex* entre elles : elles peuvent ainsi être exécutées en entrelacement (plus précisément, elles ne sont pas ordonnées). Ces actions se retrouvent dans une même couche du plan extrait. Toutes les préconditions de cet ensemble d'actions de $\mathcal{A}_i$ sont sélectionnées dans $\mathcal{P}_{i-1}$ et ainsi de suite. Cela peut être vu comme la recherche d'un chemin dans un arbre *et/ou* (les nœuds *et* correspondent au soutien des propositions, les *ou* au choix des actions). Ce processus de sélection commence avec les propositions formant l'objectif de planification dans la dernière couche propositionnelle de $\mathcal{G}$ et s'arrête lorsque $\mathcal{P}_0$ est atteint. Si aucun plan n'est trouvé, le graphe de planification est étendu avec une couche supplémentaire, jusqu'à ce qu'un plan soit extrait ou que la condition de terminaison soit atteinte (celle ci n'est pas représentée dans la figure 3.2).

L'extraction d'un plan pour l'objectif de planification contenant uniquement `AT(b-1, r-z)` se fera comme suit :

1. GP commencera l'extraction du plan à partir de la proposition `AT(b-1, r-z)` de $\mathcal{P}_3$;
2. la seule action de $\mathcal{A}_3$ qui soutienne cette proposition, c'est-à-dire l'ayant pour effet positif, est `DROP(b-1, r-z)` ;
3. toutes les préconditions de `DROP(b-1, r-z)` vont être sélectionnées dans $\mathcal{P}_2$: `CARRY(b-1)`, `G-AT(r-z)`, `BALL(b-1)`, `ROOM(r-z)` (certaines ne sont pas indiquées dans le graphe de la figure 3.1) ;
4. les contraintes liées aux relations mutex limitent le choix des actions de $\mathcal{A}_2$: si `PICK(b-1, r-a)` est sélectionné pour soutenir la proposition `CARRY(b-1)`, alors `MOVE(r-a, r-z)` ne pourra pas être sélectionné pour soutenir `G-AT(r-z)`, et inversement, puisque ces deux actions sont mutex dans $\mathcal{A}_2$; si GP choisit `PICK(b-1, r-a)`, alors l'extraction va échouer car les deux propositions `G-AT(r-z)` et `G-AT(r-a)` de $\mathcal{P}_1$ sont mutex (il n'y a pas de couple d'actions dans $\mathcal{A}_1$ qui soutienne les deux propositions). Les actions de propagations (du type `NOOP-PRED`) soutiennent les autres propositions (telles `CARRY(b-1)` ou `BALL(b-1)`) ;
5. en choisissant l'action `MOVE(r-a, r-z)` de $\mathcal{A}_2$ (accompagnée des actions de propagation), GP devra soutenir les propositions `CARRY(b-1)`, `G-AT(r-a)`, `BALL(b-1)`, `ROOM(r-z)`, `ROOM(r-a)` de $\mathcal{P}_1$;
6. `CARRY(b-1)` de $\mathcal{P}_1$ est soutenu uniquement par l'action `PICK(b-1, r-a)` de $\mathcal{P}_1$, qui n'est mutex avec aucune action de propagation permettant de soutenir les autres propositions.

Ainsi, le plan 3.1 est extrait, avec pour chaque ligne un ensemble d'actions (ici des singletons). Pour GRIPPER avec les trois balles `b-1`, `b-2` et `b-3`, GP calcule un plan totalement ordonné, donc un plan qui n'est pas flexible. L'ordre de traitement de chaque balle est imposé par le processus d'extraction du plan. Mais GP peut extraire des plans flexibles grâce à

Plan 3.1 Plan solution extrait par GP pour une version simplifiée de GRIPPER.

```
1 PICK(b-1, r-a);
2 MOVE(r-a, r-z);
3 DROP(b-1, r-z);
```

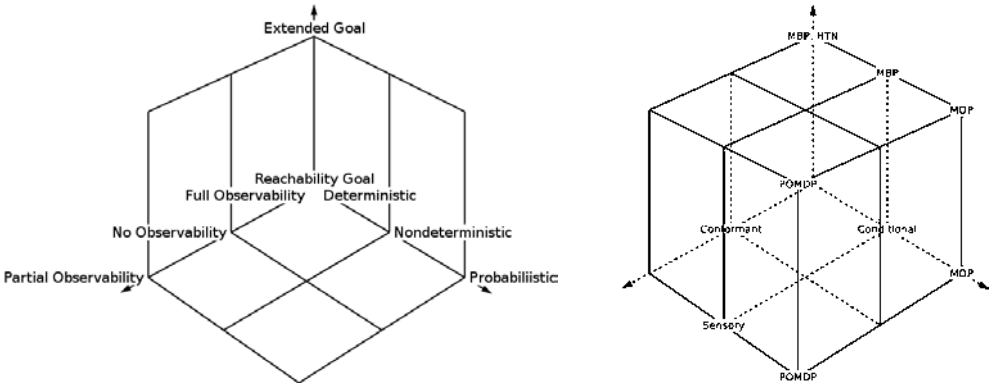


FIGURE 3.3 – Dimensions de l'incertain en planification selon [GNT04] (à gauche) et quelques approches de planification non classiques disposées selon ces dimensions (à droite, la planification classique se trouvant à l'origine du graphique).

leurs structure en couche d'actions. Avec le problème GRIPPER-B+C simple (une seule balle et un seul cube), GP extrait le plan 3.2. Ce plan est flexible : aucune contrainte d'ordre n'est

Plan 3.2 Plan solution extrait par GP pour une version simplifiée de GRIPPER-B+C.

```
1 PICK(g-c, b-1, r-a), PICK(g-b, c-1, r-z);
2 MOVE(g-c, r-a, r-z), MOVE(g-b, r-z, r-a);
3 DROP(g-c, b-1, r-z), DROP(g-b, c-1, r-a);
```

imposée entre les actions d'une même couche. Ainsi, le contrôleur peut choisir parmi les 6 exécutions possibles. La planification partiellement ordonnée était moins contrainte, puisque rien n'obligeait par exemple d'exécuter l'action `PICK(g-c, c-1, r-z)` avant `MOVE(g-b, r-a, r-z)` : les deux traitements étaient totalement indépendants. La planification fondée sur les graphes calcule généralement des plans moins flexibles que la planification partiellement ordonnée.

3.2 Planification sous incertitude

L'incertitude concerne le monde à l'exécution du plan calculé. Elle est exprimée dans le problème de planification, comme l'illustre la figure 3.3 : l'incertitude concerne l'objectif de planification, le déterminisme des actions du domaine de planification et la capacité d'observation du monde. L'incertitude peut donc porter :

1. sur l'objectif de planification : la planification généralisée permet de répondre à un ensemble d'objectifs (*cf.* paragraphe 3.2.2), et la planification fondée sur modèle [GT00] permet l'expression d'objectifs de planification « étendus » (par exemple pour GRIPPER de continuellement naviguer d'une pièce à l'autre) ;

2. sur le domaine de planification : la planification conditionnelle et la planification basée sur modèle (MBP) utilisent des actions non déterministes c'est-à-dire pouvant produire plusieurs effets possibles à l'exécution, ou considèrent que le monde est seulement partiellement observable (*cf.* paragraphe 3.2.3) ;
3. sur l'état initial : la planification conformante [Bon09, CRB04, SW98, HB06] considère un état initial seulement partiellement connu, dans un monde non observable (par exemple, savoir si un outil est stérile), mais n'a pas forcément besoin de structure particulière dans le plan. En effet, le plan doit prendre en compte l'incertitude et prévoir toutes les situations possibles, mais sans savoir dans quelle situation l'exécution se trouve. Sans observation de l'état courant, aucune décision ne peut être prise à l'exécution, et donc aucune flexibilité n'est attendue (qu'un outil soit stérile ou non, il faudra le stériliser lorsque cela est nécessaire). Dans le cas où des actions d'observation sont disponibles, c'est la planification conditionnelle qui prévoit ce cas.

La planification fondée sur modèle diffère des approches de planification introduites ci-dessus par la description explicite du domaine de planification sous la forme d'un graphe d'états : elle est issue de la vérification de modèle (ou *model checking*). Elle permet de surmonter les différents types d'incertitude, mais sous la forme de transitions non déterministes et d'objectifs de planification non classiques. Elle est présentée au paragraphe 3.2.1.

La prise en compte de l'incertitude par la planification implique d'intégrer des structures conditionnelles dans les plans pour faire face à toutes les situations possibles. L'exécution du plan s'adapte aux situations effectivement rencontrées : l'objectif de planification est atteint quelles que soient ces situations, prévues par la modélisation de l'incertitude dans le domaine de planification.

3.2.1 Planification fondée sur modèle (MBP)

Tout un pan de la planification est fondée sur une description explicite du domaine de planification sous la forme d'un automate, et notamment utilise les fondements de la vérification de modèle. La solution d'un problème de planification est une *politique*, c'est-à-dire un ensemble de couple (état atteint, action à exécuter). Les politiques permettent de traiter des problèmes de planification où :

- les transitions ne sont pas déterministes, autrement dit l'application d'une action sur un état donné peut mener dans différents états ;
- l'objectif de l'utilisateur est un objectif qualitatif et non quantitatif, c'est-à-dire que l'utilisateur ne veut pas atteindre un certain état mais veut qu'une propriété soit assurée durant l'exécution du plan.

En Model-Based Planning (MBP) [GT00], les domaines de planification sont des systèmes de transitions d'état $(\mathcal{E}, \mathcal{A}, \mathcal{T})$ qui représentent la dynamique de l'espace d'états : les nœuds sont les états du monde dans $\mathcal{E}$; chaque arc a pour label une action dans $\mathcal{A}$; les transitions dans $\mathcal{T}$ mettent en relation les actions de $\mathcal{A}$ exécutables sur les états de $\mathcal{E}$ et menant dans d'autres états de $\mathcal{E}$. Les transitions dans $\mathcal{T}$ sont des triplets (e_i, a, e_j) qui signifient que l'action a appliquée sur l'état e_i mène à l'état e_j.

Domaine non déterministe

Un domaine non déterministe en MBP [CPRT03] implique des changements d'états incertains. Cette incertitude est due à des actions non déterministes. Pour une action a non déterministe appliquée à un état e donné, cela se traduit dans le graphe par un ensemble de transitions $\{(e, a, e_1), \cdots, (e, a, e_n)\}$. Cet ensemble signifie que l'application de l'action

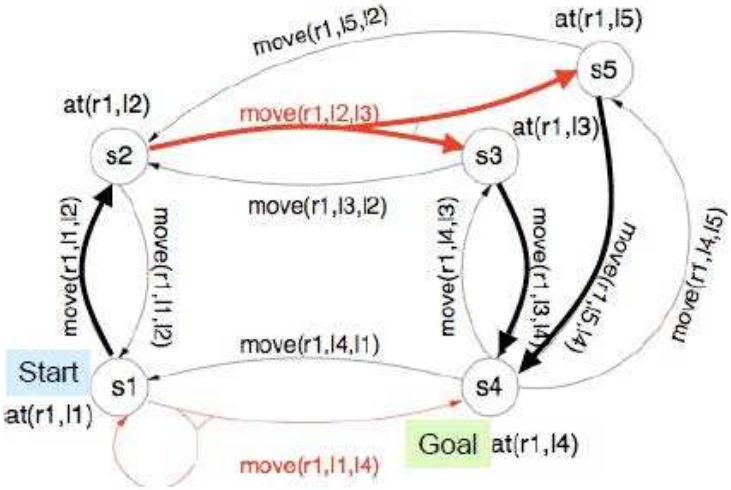


FIGURE 3.4 – Une politique « strong » en MBP, tiré de [GNT04], pour un problème de robotique : le robot doit se déplacer de l'état `s1` (place `l1`) à l'état `s4` (dans la place `l4`), et cela malgré le non déterminisme de l'action permettant au robot de se déplacer de la place `l2` à `l3` mais qui, par erreur, peut mener dans la place `l5`. La politique représentée en **gras** permet d'atteindre l'objectif quel que soit le résultat de l'action non déterministe.

mène dans l'un des états e_i, $i \in [1..n]$. Par exemple, la figure 3.4 illustre un domaine de planification en MBP. Ce domaine concerne un robot qui se déplace de place en place. Certains déplacements ne sont pas déterministes et mènent donc dans des places non désirées. C'est le cas de l'application de l'action `MOVE(r1, l2, l3)`, qui, au lieu de déplacer le robot `r1` de la place `l2` à `l3` peut mener en `l5`. Du fait de ce non déterminisme, certaines politiques sont équivalentes à des plans avec boucles. Par exemple, une autre solution au problème aurait été une politique qui à l'état `s1` fait correspondre l'action non deterministe `MOVE(r1, l1, l4)`. Cette politique peut boucler longtemps dans l'état `s1` mais permet d'atteindre l'objectif de planification au bout d'un temps indéfini.

L'algorithme de planification commence par les états qui satisfont l'objectif. Récursivement, les états qui permettent de mener aux états précédemment visités sont visités, et une fonction d'élagage permet de s'affranchir de l'exploration des états inutiles. L'algorithme termine lorsque l'état initial est visité : une politique est retournée.

Une façon de résoudre le problème GRIPPER en MBP est de décrire le domaine en tant qu'espace d'états où les deux pièces peuvent être vides ou non, et où l'action non déterministe `PICK` peut vider ou non la pièce considérée. L'objectif de l'utilisateur est alors de vider la pièce `r-a` et non de déplacer les balles dans la bonne pièce. Ainsi, les balles en tant qu'objets de planification ne sont pas explicitées. Le plan solution est un chemin dans le graphe : il est implicitement encodé dans le domaine, et contient une structure itérative grâce à la notion implicite de balle. Ce domaine est illustré à la figure 3.5 page suivante. Seul le concepteur du domaine sait que les pièces contiennent des balles et que l'action `PICK` s'applique aux balles. En conséquence, avec ce domaine de planification, il n'est pas possible de spécifier un objectif concernant un sous ensemble de balles. Autrement dit, la description du domaine de planification limite l'expressivité du plan.

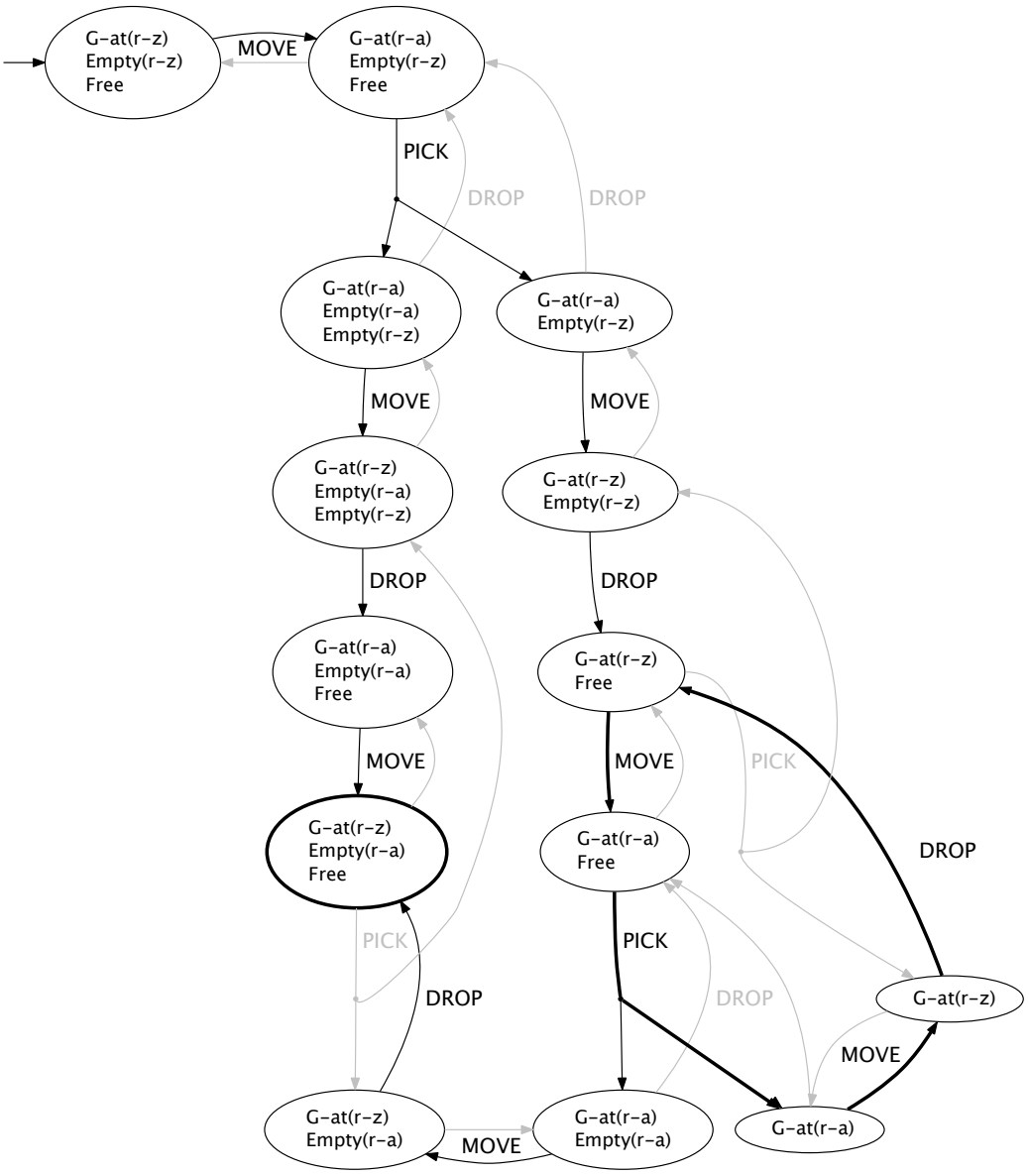


FIGURE 3.5 – Une politique pour GRIPPER en MBP en trait foncé, avec une notion de balle implicitement décrite par les propositions `EMPTY(r-a)` et `EMPTY(r-z)` qui signifient que les pièces `r-a` et `r-z` sont vides. L'objectif de planification est d'atteindre l'état où les propositions `G-AT(r-z)`, `EMPTY(r-a)` et `FREE` sont vérifiées, c'est-à-dire que le robot est dans la pièce `r-z`, que la pièce `r-a` est vide et que le robot a sa pince libre. Cette modélisation de GRIPPER permet d'obtenir une structure implicitement itérative, mise en valeur par les arcs du graphe en **gras**.

La planification fondée sur modèle suppose que tous les objectifs soient anticipés au moment de la conception par la description du domaine de planification.

Objectif étendu

Les objectifs étendus [PT01] sont rendus possibles par la manipulation explicite du domaine de planification, et par les techniques de la vérification de modèle. En effet, plutôt que de limiter les objectifs à l'atteinte d'un objectif, la planification fondée sur modèle permet d'exprimer [GNT04] des objectifs « temporels ». Ces objectifs ne dépendent plus d'un unique état final mais de tous les états successifs (par exemple que le robot ne passe jamais dans une certaine pièce ou que le robot fasse continuellement des aller-retours d'une pièce à l'autre).

Le langage CTL [Eme90] est adapté pour exprimer des objectifs étendus : en plus des connecteurs logiques usuels, il introduit les modaux A (pour tous les chemins d'exécution), E (pour « au moins un chemin d'exécution »), X (« au prochain état »), F (« un état dans le futur garantit »), G (« tous les états futurs garantissent »). Par exemple, le but "AF `at(r1, l4)`" signifie qu'avec la politique recherchée, pour tous les chemins d'exécution un état dans le futur garantit `at(r1, l4)`, c'est-à-dire que le robot peut se retrouver dans la pièce `l4`.

Analyse

Le modèle d'un système compose l'entrée du planificateur en MBP, qui, en sortie, fournit le sous-système qui répond à l'objectif de planification. Cela permet de décrire des objectifs complexes. Le plan en tant que politique peut être très expressif, mais de par la description explicite du domaine de planification, le plan est *au plus* autant expressif que celui ci.

3.2.2 Planification généralisée

Dans les travaux de planification généralisée, ce sont les plans calculés qui sont généralisés : chacun répond à un ensemble de problèmes décrit en une classe de problèmes. La description d'une classe de problèmes contient des abstractions dont les valuations correspondent aux différentes instances de problèmes. Cela s'exprime généralement comme une incertitude sur le nombre d'objets à traiter par le plan, par exemple le nombre de balles à déplacer dans Gripper. Le problème de planification inclut la description de la classe de problèmes à résoudre plutôt qu'un objectif de planification sous la forme d'un ensemble de propositions à satisfaire. Les plans intègrent alors des structures de contrôle permettant d'adapter l'exécution du plan au problème effectivement traité. Ainsi, pour chacun des travaux suivants, la flexibilité des plans calculés est issues de l'incertitude décrite dans le problème de planification.

Aranda

Aranda [SIZ08, SIZ10] utilise la notion de *prédicats d'abstraction* pour définir des *rôles* dans le problème de planification. Ces rôles rendent possible la transformation d'un exemple de plan en un espace d'états abstraits (décrit par une logique tri-valuée). De cette transformation, l'algorithme trouve des boucles sur les objets jouant le rôle défini. Le plan peut alors être appliqué à n'importe quel problème instance de la classe correspondante. De plus, un ensemble de contraintes d'intégrité assure la validité des états abstraits. Pour résoudre Gripper, un rôle « balle dans la pièce `r-a` » doit être défini de façon à abstraire toutes les balles de la pièce `r-a` dans un unique objet abstrait. Ainsi, pour un objectif qui

ne concerne qu'un sous-ensemble des balles, de nouveaux rôles doivent être définis dans le problème de planification sans quoi l'abstraction ne pourra distinguer les balles à déplacer de celles qu'il ne faut pas toucher. Concernant notre problématique, la limite majeure de cette approche est la définition à la conception des différents rôles à travers la liste explicite des *prédicats d'abstraction* : cette liste limite la classe de problèmes que peut résoudre le plan généralisé.

FsaPlanner

Avec KPlanner [Lev05] et son évolution FsaPlanner [HL09], la classe de problème à résoudre contient une variable entière, le paramètre, qui indique le nombre d'objets à traiter. Cette variable est utilisée comme itérateur dans le plan engendré. Par exemple, dans Gripper, le nombre de balles est le paramètre. Le but de l'utilisateur n'est pas explicitement décrit comme une collection de balles à déplacer : l'objectif est uniquement le déplacement de l'ensemble des balles, et le plan solution peut fonctionner avec n'importe quel nombre de balles. En planifiant une première fois avec une petite valeur pour cette variable, un plan conditionnel est obtenu et transformé de façon à présenter une structure itérative. Ce plan itératif est testé avec une valeur supérieure : en cas d'échec, il faut trouver un nouveau plan ; en cas de succès, le plan est considéré comme solution.

LoopDistill

Avec [WV07] la proposition est un peu différente de la planification généralisée. En effet, le but est de fournir un planificateur « domain-specific » (appelé dsPlanner). Un exemple de plan en entrée est observé pour y trouver des patrons d'actions qui se répètent. Ces patrons sont transformés en structure de boucle. Cela nécessite de calculer un premier plan solution avant de le transformer en un plan qui le généralise par l'introduction de structures itératives (non imbriquées). Ce plan généralisé est utilisé comme une heuristique pour de futurs problèmes. Deux structures d'itérations sont définies : les *parallel loops* et les *serial loops*. Une boucle parallèle (resp. une boucle séquentielle) est « une boucle pour laquelle chaque itération est causalement indépendente (resp. liée) aux autres ». Seules les boucles séquentielles pourraient apporter de la flexibilité dans le plan si elle permettait un contrôle à l'exécution. Mais cette approche fournit des plans trop généralisés pour notre problématique. Par exemple, dans le « monde des cubes », une pile de cubes doit être mise à terre, cube après cube. Les dépilages successifs correspondent à la définition des boucles séquentielles. Mais les dépilages successifs ne devraient pas être transformés en une structure itérative. De fait, il n'y a qu'une seule façon de dépiler les cubes (en prenant le cube supérieur à chaque itération) : aucun choix n'est possible pour réaliser l'objectif.

Analyse

Usuellement, la planification généralisée nécessite des informations supplémentaires en entrée permettant d'apporter l'incertitude dans la description du problème et de la surmonter dans le plan. Dans le cadre de cette thèse, nous souhaitons qu'aucune information sur la composition ne soit décrite dans la définition du problème, et cela implique aucune incertitude quant à la description de l'objectif de l'utilisateur. La planification généralisée est conçue pour fournir un plan solution d'un ensemble d'objectifs, alors que notre dessein est de fournir un plan flexible qui réponde à un unique objectif.

3.2.3 Planification conditionnelle (conformant, conditional, MDP)

Des incertitudes peuvent survenir à la conception du problème de planification, c'est-à-dire être issue d'une limitation dans la description du monde, où être propres à l'exécution, c'est-à-dire dépendre de l'application des actions. Les structures de contrôle que nous recherchons dans les plans ont du sens uniquement lorsque le monde est observable, au moins partiellement, à l'exécution. En effet, sans observation, il n'y a pas de choix. Aussi, les incertitudes dans la description des états, doivent pouvoir être résolues à l'exécution : le plan présente ainsi nécessairement des structures de contrôles selon les observations effectuées à l'exécution. Nous distinguons plusieurs types d'actions [GA99] liés aux incertitudes discutées :

- les actions déterministes, qui n'ont qu'un seul comportement possible ;
- les actions conditionnelles pour lesquelles le contexte à l'exécution conditionne le comportement ;
- les actions non déterministes qui ont un ensemble de comportements possibles, indépendamment du contexte ;
- les actions de perception, ou senseurs, qui lèvent des incertitudes sur l'état du monde : les incertitudes sont alors modélisées comme une valeur *inconnue* d'une propriété du monde (par exemple son affirmation ou sa négation), et les senseurs, par perception du monde à l'exécution, donnent l'unique valeur de l'inconnue parmi toutes les valeurs possibles (les senseurs produisent de la connaissance).

Lorsque le domaine de planification contient l'une de ces incertitudes, c'est-à-dire lorsque les actions peuvent avoir plusieurs effets différents selon le contexte à l'exécution, alors les planificateurs doivent prendre en compte chacun des effets.

Pour décrire les effets conditionnels des actions, des langages ont été développés : UWL [EHW+92] spécifiquement créé pour intégrer l'incertitude, ADL [Ped89] qui ajoute notamment l'utilisation des quantifieurs de la logique du premier ordre, puis PDDL [AD05] avec les « when-effects » pour les actions conditionnelles et, plus tard, PPDDL [YL04] pour ajouter des probabilités aux effets. Concernant les probabilité d'application d'un effet, PDL [HKT84] intègre des modalités sur l'exécution des actions (qui produisent « possiblement » ou « nécessairement » leurs effets) et son extension WCPL [GB96] permet notamment de représenter et de raisonner sur des actions avec des informations incomplètes, ou sur des senseurs [GBP96] : une distinction est effectuée entre :

- les changements dans le monde impliqués par l'application d'une action ;
- la connaissance acquise par l'utilisation d'un senseur.

Il est difficile d'être exhaustif en planification conditionnelle, car de nombreux travaux ont été mené pour pallier aux différentes incertitudes et avec des approches très différentes. Dans ce paragraphe, nous présentons seulement quelques planificateurs conditionnels en insistant principalement sur les structures des plans calculés. Nous divisons les travaux selon les approches adoptées.

Planification « linéaire »

[LB74] pourrait être considéré comme le premier planificateur conformant, c'est-à-dire s'adaptant à une description incomplète de l'état initial. En effet, il manipule des préconditions d'actions de valeur « inconnue ». Les plans traitent les deux valeurs possibles des inconnues à l'exécution, « vraie » ou « fausse ». Ainsi, les plans ont une branche conditionnelle pour chaque inconnue : les plans sont des arbres de décision binaire. Le monde est considéré connu à l'exécution : aucun senseur n'est requis.

Warplan-C [War76] est désigné comme le premier planificateur conditionnel. Les actions ont des « effets de bords » imprévisibles mais envisageables (l'existence des effets de bords est avérée et les effets possibles sont connus), autrement dit la dynamique du monde à l'exécution n'est pas complètement connue. Les plans qui contiennent des « structures conditionnelles » sont appelés des « programmes ». Ainsi, les actions sont non déterministes : elles peuvent produire un effet et son contraire. Warplan-C calcule un premier plan solution en faisant une simplification : seul le premier effet de chaque action non déterministe utilisée dans le plan est considéré. Ensuite, Warplan-C raffine le plan en ajoutant les structures conditionnelles au niveau des actions non déterministes pour simuler l'autre effet des actions, et continue la planification pour que chaque branche ainsi créée mène à l'objectif.

Dans Plinth [GB94a, GB94b], le plan est un arbre dont les nœuds sont les actions et les arcs les observations. Une différenciation est proposée entre :

- les actions dont les effets dépendent du contexte d'exécution ;
- les actions dont les effets sont imprévisibles.

Des préconditions « secondaires » sont utilisées pour décrire les actions dont les effets dépendent du contexte : chaque paire de préconditions secondaires est exclusive et les effets liées aux préconditions diffèrent.

Planification partiellement ordonnée

La plupart des planificateurs conçus pour traiter des domaines non déterministes avec des plans partiellement ordonnés sont fondés sur SNLP [MR91], un planificateur « non linéaire » : ses plans sont partiellement ordonnés. Il y a eu une première extension de SNLP [CP92], qui a donné naissance à CNLP [PS92].

Les deux visions rencontrées dans [War76] et [LB74] se rejoignent dans CNLP : des actions « conditionnelles » ont des préconditions qui ont une valeur « inconnue », et qui a pour effet l'évaluation de l'inconnue. Ces actions sont donc des senseurs : des actions non déterministes qui permettent de tester le monde pour combler les lacunes de connaissance au moment de l'exécution. Le plan compose les actions et les senseurs, et des branchements permettent de s'adapter aux résultats des détections. Dans le plan, chaque action est liée à un contexte – un ensemble d'observations – qui détermine si l'action peut être exécutée. Le plan (partiellement ordonné) intègre des structures conditionnelles qui réagissent en accord avec les observations réalisées à l'exécution. Ces observations ne sont pas explicites : le contrôleur connaît le résultat de l'application de chacune des actions aux effets incertains, c'est-à-dire qu'à l'exécution, le résultat est directement connu.

UCPOP [PW92] est une évolution de SNLP [CP92]. Ces planificateurs utilisent en entrée le langage ADL, et notamment dans [PW92] les quantifieurs pour décrire des incertitudes concernant l'état initial, les effets et l'objectif. Cassandra [PC96, GBP96] est fondée sur UCPOP. Une nouveauté dans la structure du plan apparaît avec Cassandra : le plan partiellement ordonné contient des étapes explicites de décision. La figure 3.6 est issue de [PC96]. Elle représente un plan avec un choix explicite entre prendre la route `Western` ou `Belmont` selon les observations effectuées.

Planification fondée sur les graphes (Graphplan)

Les planificateurs conditionnels fondés sur la planification partiellement ordonnée souffrent de leur faible performance. Dans l'objectif d'améliorer les performances de la planification conditionnelle, des travaux [WAS98, SW98] ont étendu l'approche par graphe

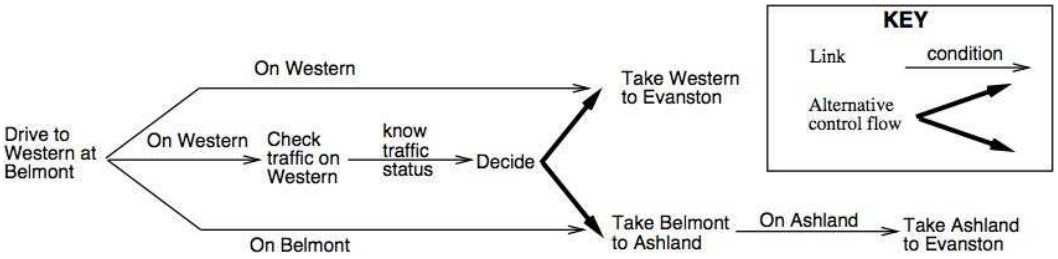


FIGURE 3.6 – Un plan solution issue de [PC96] : l'objectif de planification est de se déplacer jusqu'à `Evanston` en partant de `Western`. Selon l'encombrement de la route observé depuis `Western` sur le premier tronçon menant à `Belmont`, un choix explicite est possible entre continuer sur la route `Western` ou changer pour la route `Belmont` puis `Ashland`.

de planification [BF97]. L'approche consiste à ne plus simuler un unique graphe de planification, mais d'en simuler autant qu'il y a d'états initiaux ou d'effets aux actions non déterministes [SW98]. En plus de cela, la distinction entre les senseurs et les actions est prise en compte dans [WAS98]. Ces planificateurs manipulent une représentation explicite (mais condensée) des états possibles à chaque instant. Les plans qui en sont issus permettent d'atteindre l'objectif quel que soit l'état initial dans lequel les plans sont exécutés. Lorsque l'état à l'exécution est complètement connu, le plan est un plan en couche comme rencontré dans Graphplan [BF97]. Ainsi, pour chaque état possible, un plan en couche est disponible.

Planification probabiliste

Jusque là, l'incertitude n'est pas mesurée car elle provient d'une limitation dans la modélisation du monde : l'incertitude est levée à l'exécution, car le monde n'est pas observable à la conception du plan. L'incertitude dans le monde peut survenir de comportements probabilistes. Il y a deux modélisations possibles des probabilités qu'un événement survienne : la première est qualitative, la seconde quantitative. Qualitativement, les faits sont modulés par les quantifieurs : par exemple « possiblement » et « nécessairement ». Quantitativement, l'incertitude d'apparition des faits est explicitée par une valeur numérique de probabilité. La planification conditionnelle probabiliste a généralement pour objectif de maximiser les chances d'atteindre l'objectif. Lorsque de surcroit le monde est partiellement observable, la plupart des planificateurs naviguent dans des espaces de croyances, qui sont totalement observables et prévisibles.

BURIDAN [KHW94] calcule un plan partiellement ordonné qui garantit d'atteindre l'objectif avec une probabilité supérieure à un seuil donné : le domaine de planification est modélisé comme des actions qui, dans un état donné, ont une probabilité numérique donnée de provoquer chacun de ses effets. Par exemple, une action du domaine GRIPPER dans [KHW94] est présentée par la figure 3.7. Le plan construit dans [KHW94] permettant d'atteindre l'objectif, peindre un objet, est présenté à la figure 3.8. Nous observons que ce plan n'est pas plus expressif qu'un plan partiellement ordonné, mise à part la prise en compte des effets non déterministes des actions.

Dans PTLplan [Kar01], la notion de *situations* dénote un état possible du monde à un instant donné et celle de *situations épistémiques* un ensemble de situations possibles à un instant donné, c'est-à-dire les connaissances et croyances d'un agent à cet instant. Les connaissances expertes du domaine, dites stratégiques, permettent de limiter l'espace de

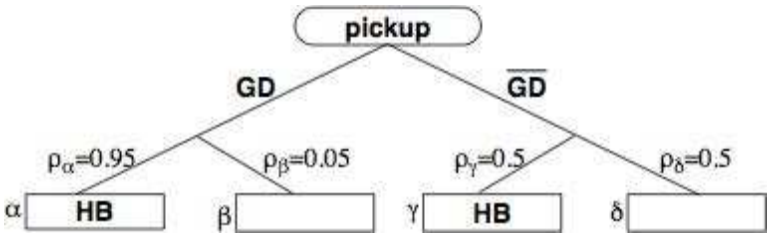


FIGURE 3.7 – Une action non déterministe et probabiliste issue de [KHW94] : les deux premières branches indiquent si le robot de GRIPPER à sa pince sèche (dans l'article, la précondition `GD` pour *Gripper is Dry*) avec la branche de gauche, ou n'est pas sèche avec la branche de droite ; à chacune de ces branches est associée la probabilité d'attraper l'objet souhaité (dans l'article, l'effet `HB` pour *Holding a Block*), qui est par exemple de 0.95 si la pince est sèche.

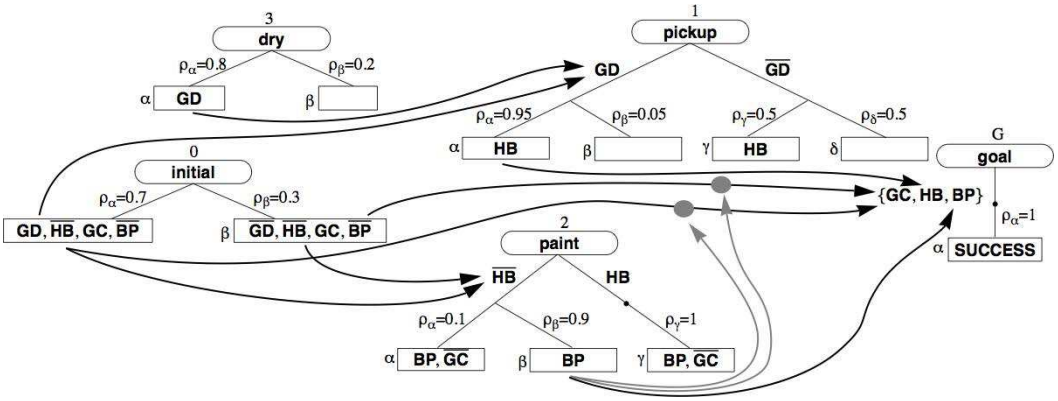


FIGURE 3.8 – Un plan probabiliste solution issue de [KHW94] : les actions `INITIAL` et `GOAL` décrivent le problème de planification (l'état initial contient des incertitudes), les autres actions font partie du domaine de planification ; les chiffres sur les actions indiquent l'ordre d'ajout des actions dans le plan ; les flèches noires indiquent les liens de causalités entre les actions et les flèches grises indiquent les modifications sur les probabilités d'un lien, autrement dit la menace d'une action sur un lien (ici, c'est `GC` qui est menacé) ; la proposition `GC` pour *Gripper is Clean* signifie que la pince est propre, et la proposition `BP` pour *Block is Painted* signifie que l'objet est peint.

plan ::= `success` | `fail` | *action* ; *plan* | `cond` *branch**
branch ::= (*cond* : *plan*)
action ::= *action-name* (*args*)

FIGURE 3.9 – Syntaxe des plans conditionnels dans [Kar01].

recherche. Après l'application d'une action sur une situation épistémique, ce sont les observations qui permettent d'affiner et surtout d'évaluer les différentes situations épistémiques résultantes. Comme ce planificateur utilise de nouvelles notions, il faut ajouter de l'expressivité aux éléments manipulés, notamment concernant le non déterminisme des effets ainsi que les actions permettant l'observation. Un plan se définit selon la syntaxe décrite à la figure 3.9. Sémantiquement, un plan est un ensemble de plans séquentiels, dont les sous-ensembles ont mêmes préfixes jusqu'à ce que des branchements, conditionnés par l'état du monde, font différer leurs suffixes. Nous pouvons aussi le représenter comme un arbre dont les arcs représentent des actions, conditionnées dans le cas où le noeud origine de l'arc possède plusieurs fils. L'exécution d'un plan est un chemin de la racine du plan à une feuille se traduisant par l'application des actions représentées par les arcs de l'arbre sous la condition que les variables du monde vérifient les conditions de branchement présentes sur les arcs.

Dans [BP96], les POMDP – *Partially Observable Markov Decision Process* – sont utilisés pour traiter le problème de l'incertitude quantitative dans un monde partiellement observable à l'exécution. Généralement, les POMDP raisonnent sur une énumération explicite des états. Ce travail exploite par surcroît un espace de croyances structuré, une représentation condensée des états. Le plan calculé est une politique, c'est-à-dire en ensemble de relations entre une situation rencontrée à l'exécution et l'action à exécuter dans cette situation.

Analyse

La planification conditionnelle répond à la distinction entre la conception et l'exécution du plan : les planificateurs conditionnels fournissent des plans avec lesquels le contrôleur est en capacité de surmonter les incertitudes lors de l'exécution des plans. Le contrôleur exécute le plan sans ambiguïté, c'est-à-dire sans réellement décider de la stratégie à adopter pour atteindre l'objectif. En effet, la stratégie est déjà intégrée dans le plan conditionnel. Ainsi, tout comme pour la planification généralisée, c'est dans la définition du problème de planification que se situe le non déterminisme. Ce non déterminisme se retrouve dans les plans conditionnels sous la forme de structures conditionnelles. Les plans conditionnels permettent donc d'atteindre les objectifs en s'adaptant à un contexte modélisé comme non déterministe.

Les structures conditionnelles nécessaires dans ce type de planification ne correspond pas aux structures que nous recherchons. Notre travail s'inscrit dans la planification classique, avec un domaine déterministe et totalement observable, donc à l'opposé du domaine de la planification conditionnelle. Cela signifie simplement que cette approche ne nous permet pas d'obtenir l'expressivité attendue dans une composition. Dans notre cas, les choix offerts à l'exécution ne sont pas issus du non déterminisme des actions ou d'une connaissance partielle du monde, mais de la flexibilité indispensable pour adapter le plan au contexte d'usage. Les plans conditionnels présentés ici ne sont pas flexibles : une unique exécution du plan est possible, même si cette exécution est dépendante du non déterminisme du monde sur lequel le plan est appliqué.

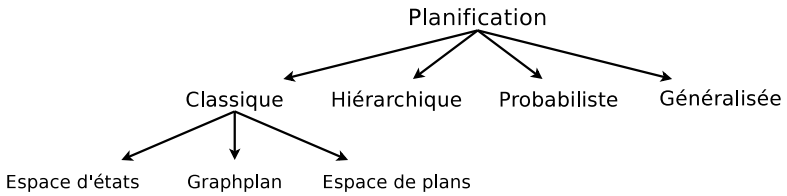


FIGURE 3.10 – Taxinomie simplifiée de la planification, adaptée de [SFJ00].

3.3 Conclusion

Les approches de la planification sous incertitude, que ce soit la planification généralisée ou conditionnelle, ont des hypothèses qui leur permettent de décrire dans le domaine de planification une forme de non déterminisme qui se retrouve dans le plan. Notre approche est différente : le non déterminisme n'est pas issu du domaine, mais bien présent dans le plan. La flexibilité recherchée n'est pas le résultat d'hypothèses fortes sur le domaine mais d'exigence sur l'expressivité du plan solution, qui doit laisser des choix à l'exécution (en toute connaissance du monde lors de la conception et de l'exécution). En conséquence, notre approche s'accorde à la planification classique, illustré dans la taxinomie de la planification à la figure 3.10. En effet, la planification hiérarchique [JDKR] implique que l'objectif de l'utilisateur ne peut pas être totalement imprévu : la hiérarchie décompose a priori l'objectif de l'utilisateur en sous-objectifs jusqu'aux actions atomiques. La planification probabiliste [Bly99] s'accommode d'actions ayant des effets incertains, ce qui est inutile dans notre cas. La planification généralisée [SS09] considère un ensemble d'objectifs plutôt qu'un unique. Nous proposons donc de fonder notre approche sur un planificateur classique. Nous avons choisi `GraphPlan` pour sa représentation condensée des états.

Deuxième partie

Théorie de la flexibilité en planification

Chapitre 4

Modèle de planification classique

L'objectif de notre travail est de fournir au planificateur la capacité de calculer un plan adaptable par le contrôleur au contexte d'exécution. Nous disons que le plan est flexible. Plus précisément, nos contributions se situent au niveau de l'expressivité des plans et de l'algorithme pour les calculer :

$$\xrightarrow{\text{problème}} \text{Planificateur} \xrightarrow{\text{plan}} \text{Contrôleur} \underset{\text{observation}}{\overset{\text{action}}{\rightleftharpoons}} \text{système de transition d'états} \xleftarrow[\text{évènement}]{}$$

Ce chapitre présente un modèle de la planification classique [GNT04]. Le chapitre 5 définira la planification flexible en s'appuyant sur ce modèle.

4.1 Problème de planification

Pour ce paragraphe, nous nous intéressons à la description du problème de planification :

$$\xrightarrow{\text{problème}} \text{Planificateur} \xrightarrow{\text{plan}} \text{Contrôleur}$$

Nous nous plaçons dans le cadre de la planification déterministe, complètement observable. La logique propositionnelle est ainsi suffisante pour décrire le modèle de planification. La logique propositionnelle décrit le problème de planification en extension : chacune des observations possibles est décrite par une proposition, et chacune des actions est explicitement spécifiée par la manipulation de ces propositions. Cette description pouvant s'avérer fastidieuse, les langages de planification proposent une description condensée et implicite du domaine de planification, qui autorise l'utilisation de variables : les actions s'appliquent alors sur un ensemble d'objets. Par exemple, le langage STRIPS [FN71] permet de décrire en intention le problème de planification.

4.1.1 Langage prédicatif

Notre langage prédicatif est construit sur les ensembles suivants :

- $\mathscr{V}$: un ensemble fini de symboles appelés *variables* (dans Gripper, nous avons *ball*, *room*, *etc.*) ;
- $\mathscr{C}$: un ensemble fini, possiblement vide, de symboles appelés *constantes* (dans Gripper, nous avons par exemple `b-1`, `b-2` ou encore `r-a`) ;
- $\mathscr{P}$: un ensemble fini de *symboles de prédicats* (dans Gripper, nous avons Free, Carry, *etc.*), chaque symbole de prédicat ayant une arité qui détermine le nombre d'arguments issus des ensembles de variables et de constantes auxquels il est appliqué ;
- $\mathscr{L}$: un ensemble fini de *prédicats*. Un prédicat est noté $\textsc{Pred}(\alpha_1, \cdots, \alpha_n)$ si Pred est n-aire dans $\mathscr{P}$ et $\alpha_1, \cdots, \alpha_n$ sont des symboles de variable ou de constante, c'est-à-dire $\forall i, \alpha_i \in \mathscr{V} \cup \mathscr{C}$. Par exemple, les prédicats $\textsc{Carry}(\textit{ball})$ et Free font partie du langage prédicatif de Gripper.

- $\mathscr{O}$: un ensemble fini de *symboles d'opérateurs* (de GRIPPER, nous avons `PICK`, `MOVE`, *etc.*), chaque symbole d'opérateur ayant une arité qui détermine le nombre d'arguments issus des ensembles de variables et de constantes auxquels il est appliqué.
- $\mathscr{A}$: un ensemble fini d'opérateurs, notés $\texttt{OPE}(\alpha_1, \cdots, \alpha_n)$ si `OPE` est n-aire dans $\mathscr{O}$ et $\alpha_1, \cdots, \alpha_n$ sont des symboles de variables ou de constantes, c'est-à-dire $\forall i, \alpha_i \in \mathscr{V} \cup \mathscr{C}$. Un opérateur d'arité nulle est noté simplement `OPE` (par exemple, les opérateurs $\texttt{PICK}(ball)$ et $\texttt{MOVE}(from, to)$ sont issus de GRIPPER).

4.1.2 Opérateur

Nous présentons la définition d'un opérateur.

Définition 4.1 (Opérateur)**.** *Un opérateur ope est un tuple* $(nom(ope), pre(ope), eff(ope))$*, où :*

- $nom(ope)$ *est le nom de l'opérateur dans* $\mathscr{A}$ *permettant de l'identifier. Les arguments de* $nom(ope)$ *dans* $\mathscr{V}$ *est un tuple de variables noté* $var(ope)$ *;*
- $pre(ope)$ *est un ensemble de prédicats de* $\mathscr{L}$ *et représente les préconditions de l'opérateur ;*
- $eff(ope)$ *est un couple d'ensembles de prédicats* $eff^+(a)$ *et* $eff^-(a)$*, qui représentent respectivement les effets positifs et négatifs de l'opérateur.*

Les ensembles $eff^+(ope)$ *et* $eff^-(ope)$ *sont disjoints :* $eff^+(ope) \cap eff^-(ope) = \emptyset$*.*

Un opérateur *ope* est complètement et sans ambiguité représenté par le symbole d'opérateur de $nom(ope)$, suivi de ses arguments $var(ope)$. Par exemple, l'opérateur $\texttt{PICK}(ball, room)$ représente l'opérateur tiré du domaine de planification GRIPPER décrit en annexe B.3 dont :

$$
\begin{aligned}
nom(\texttt{PICK}(ball, room)) &= \texttt{PICK};\\
var(\texttt{PICK}(ball, room)) &= (ball, room);\\
pre(\texttt{PICK}(ball, room)) &= \{\textsc{Ball}(ball), \textsc{Room}(room), \texttt{G-AT}(room), \textsc{At}(ball, room), \textsc{Free}\};\\
eff^+(\texttt{PICK}(ball, room)) &= \{\textsc{Carry}(ball)\};\\
eff^-(\texttt{PICK}(ball, room)) &= \{\textsc{At}(ball, room), \textsc{Free}\}.
\end{aligned}
$$

L'opérateur $\texttt{PICK}(ball, room)$ décrit le fait que pour une balle et une pièce données, si le robot et la balle sont dans cette pièce et que le robot a sa pince de libre (les préconditions), alors le robot peut attraper la balle (les effets positifs) mais cela rend indisponible sa pince et la balle n'est plus dans la pièce (les effets négatifs). Un autre opérateur est, par exemple, $\texttt{PICK}(ball, \texttt{r-a})$: les préconditions *invariantes* (c'est-à-dire sans variable) de cet opérateur exigent que le robot se tienne dans la pièce nommée `r-a` avec sa pince libre pour que l'opérateur puisse être exécuté, tandis que les effets *variants* (c'est-à-dire avec variable) font que le robot tient dans sa pince une certaine balle qui n'est plus dans la pièce `r-a`.

4.1.3 Substitution

Nous introduisons la substitution qui permet de faire le lien entre la logique prédicative et la logique propositionnelle utilisée dans la suite du chapitre.

Définition 4.2 (Substitution)**.** *Une* substitution σ *est un tuple de couples associant à une variable une constante, notée* $\sigma = (x_1 \backslash \texttt{c}_1, \cdots, x_n \backslash \texttt{c}_n)$ *tel que* $x_i = x_j$ *implique* $i = j$*, et* $\forall i$*,* $x_i \in \mathscr{V}$ *et* $\texttt{c}_i \in \mathscr{C}$*.*

Pour $\sigma = (x_1 \backslash \mathtt{c}_1, \cdots, x_n \backslash \mathtt{c}_n)$, nous notons $var(\sigma)$ le tuple de variables $(x_1, \cdots, x_n)$ et $val(\sigma)$ le tuple de constantes $(\mathtt{c}_1, \cdots, \mathtt{c}_n)$.

L'application de la substitution σ sur le prédicat $\textsc{Pred}(x_1, \cdots, x_n)$ de $\mathscr{L}$, noté $\sigma(\textsc{Pred})$, donne le prédicat $\textsc{Pred}$ où $\forall x_i \in \{x_1, \cdots, x_n\}$, si $x_i \backslash \mathtt{c}_i \in \sigma$ alors x_i est remplacé par $\mathtt{c}_i$ dans $\textsc{Pred}$. Ainsi, la substitution $\{ball \backslash \mathtt{b\text{-}1}\}$ appliquée sur le prédicat $\textsc{Carry}(ball)$ donne la proposition $\textsc{Carry}(\mathtt{b\text{-}1})$. Nous définissons l'application d'une substitution σ sur un ensemble de prédicats comme étant l'application de σ sur chaque élément de l'ensemble :

$$\text{Avec } \forall i, \textsc{Pred}_i \in \mathscr{L}, \sigma\big(\{\textsc{Pred}_1, \cdots, \textsc{Pred}_n\}\big) = \{\sigma(\textsc{Pred}_1), \cdots, \sigma(\textsc{Pred}_n)\}$$

L'application de la substitution σ sur l'opérateur $\mathtt{OP}(x_1, \cdots, x_n)$, notée $\sigma(\mathtt{OP}(x_1, \cdots, x_n))$, donne l'opérateur $\mathtt{OP}$ où $\forall x_i \in \{x_1, \cdots, x_n\}$, si $x_i \backslash \mathtt{c}_i \in \sigma$ alors x_i est remplacé par $\mathtt{c}_i$ dans $\mathtt{OP}$. Ainsi, la substitution $(ball \backslash \mathtt{b\text{-}1})$ appliquée sur l'opérateur $\mathtt{PICK}(ball, room)$ donne l'opérateur $\mathtt{PICK(b\text{-}1,}\ room\mathtt{)}$. Nous étendons l'application d'une substitution σ à un opérateur *ope* :

$$\sigma(ope) = \big(\sigma(nom\,(ope)), \sigma(pre\,(ope)), \sigma(\mathit{eff}^+(ope)), \sigma(\mathit{eff}^-(ope))\big)$$

4.1.4 Langage propositionnel

Une proposition est un prédicat dont tous les arguments sont des constantes. Chaque proposition a pour rôle de représenter une propriété du monde. Le langage $\mathcal{L}$ est un ensemble fini de *propositions*. Par exemple, les propositions $\textsc{Carry}(\mathtt{b\text{-}1})$ et $\textsc{Free}$ sont issues du langage propositionnel de GRIPPER, et signifient, respectivement :

– que le robot agrippe une balle $\mathtt{b\text{-}1}$;
– qu'au contraire, le robot a sa pince vide.

Nous notons $\mathcal{L}$ le sous-ensemble fini de $\mathscr{L}$ des *propositions*, qui sont les prédicats dont tous les arguments sont des constantes. Elles sont notées $\textsc{Pred}(\mathtt{c}_1, \cdots, \mathtt{c}_n)$ si $\textsc{Pred}$ est n-aire et $\mathtt{c}_1, \cdots, \mathtt{c}_n$ sont des symboles de constante, c'est-à-dire $\forall i, \mathtt{c}_i \in \mathscr{C}$. Par exemple, les propositions $\textsc{Carry}(\mathtt{b\text{-}1})$ et $\textsc{Free}$ sont issus de GRIPPER.

4.1.5 État

Un état est un ensemble de propositions : chaque état représente une situation particulière du monde dans laquelle seules les propriétés représentées par les propositions de l'état sont vérifiées dans la situation.

Définition 4.3 (État)**.** *Un état e est un ensemble de propositions issues de* $\mathcal{L}$*.*

Nous nous plaçons dans le cadre d'un monde fermé : une proposition qui n'appartient pas à l'état est évaluée fausse. Par exemple, un état dans le domaine de GRIPPER est l'ensemble des propositions $\{\textsc{Free}, \textsc{Room}(\mathtt{r\text{-}a}), \textsc{G-at}(\mathtt{r\text{-}a})\}$, qui définit l'état dans lequel :

– le robot a sa pince vide ;
– $\mathtt{r\text{-}a}$ est un emplacement ;
– le robot se situe dans l'emplacement $\mathtt{r\text{-}a}$.

4.1.6 Action

Une action a est le résultat de l'application d'une substitution σ sur un opérateur *ope*. Ainsi, nous pouvons noter a comme un couple $\sigma(ope)$ ou y faire référence uniquement par l'opérateur dans $\mathscr{A}$ résultant de l'application de σ sur $nom(ope)$. Par l'application de la substitution sur l'opérateur, les ensembles de préconditions et d'effets sont composés exclusivement de propositions.

Définition 4.4 (Action)**.** *Une action $a = \sigma(ope)$ est l'application d'une substitution σ sur un opérateur ope, avec σ telle que $var(ope) \subseteq var(\sigma)$, et a est donc composée uniquement d'ensemble de propositions. Une action est ainsi un tuple $(nom(a), pre(a), \mathit{eff}(a))$:*

- *$nom(a)$ est un symbole d'action, son identifiant ;*
- *$pre(a)$ est un ensemble de propositions de $\mathcal{L}$ et représente les préconditions de l'action.*
- *$\mathit{eff}(a)$ est un couple d'ensembles de propositions $\mathit{eff}^+(a)$ et $\mathit{eff}^-(a)$, qui représentent respectivement les effets positifs et négatifs de l'action. Ces ensembles sont disjoints :*

$$\mathit{eff}^+(a) \cap \mathit{eff}^-(a) = \emptyset$$

L'action a est applicable *dans un état e si ses préconditions sont incluses dans e :*

$$a \text{ est applicable dans } e \text{ si et seulement si } pre(a) \subseteq e$$

L'application de a sur l'état e_i, si a est applicable dans e_i, amène dans l'état e_{i+1} qui est l'état e_i dont les effets négatifs de a ont été supprimés et dont les effets positifs de a ont été ajoutés :

$$a \text{ appliquée sur } e_i \text{ produit l'état } (e_{i+1} = e_i \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$$

Par exemple, l'action a dont $nom(a) =$ `PICK(b-1, r-a)` est issue de l'application de la substitution $\{ball \backslash$`b-1`$, room \backslash$`r-a`$\}$ sur l'opérateur `PICK(`$ball, room$`)`. L'action `PICK(b-1, r-a)` permet d'agripper la balle `b-1` depuis l'emplacement `r-a` :

- $pre(a) = \{$`BALL(b-1)`$,$`ROOM(r-a)`$,$`G-AT(r-a)`$,$`AT(b-1, r-a)`$,$`FREE`$\}$, ce qui signifie que pour pouvoir appliquer cette action, il faut être dans un état dans lequel :
  - il y a une balle `b-1` ;
  - il y a un emplacement `r-a` ;
  - le robot et la balle `b-1` se situent à l'emplacement `r-a` ;
  - le robot a sa pince vide.
- $\mathit{eff}^+(a) = \{$`CARRY(b-1)`$\}$, ce qui signifie qu'après avoir appliqué cette action, le robot agrippe la balle `b-1` ;
- $\mathit{eff}^-(a) = \{$`AT(b-1, r-a)`$,$`FREE`$\}$, ce qui signifie qu'après avoir appliqué cette action, la balle `b-1` ne se situe plus dans l'emplacement `r-a`, et le robot n'a plus sa pince vide.

Ainsi, l'application de `PICK(b-1, r-a)` sur l'état composé de $pre($`PICK(b-1, r-a)`$)$ produit l'état $\{$`BALL(b-1)`$,$`ROOM(r-a)`$,$`G-AT(r-a)`$,$`CARRY(b-1)`$\}$.

$\mathcal{A}$ est le sous-ensemble fini de $\mathscr{A}$ des *actions*, qui sont les opérateurs dont tous les arguments sont des constantes. Elles sont notées `OP(`$\mathtt{c}_1, \cdots, \mathtt{c}_n$`)` si `OP` est n-aire et $\mathtt{c}_1, \cdots, \mathtt{c}_n$ sont des symboles de constantes, c'est-à-dire $\forall i, \mathtt{c}_i \in \mathscr{C}$. Par exemple, l'action `PICK(b-1, r-a)` est issue de GRIPPER. L'ensemble des actions $\mathcal{A}$ forme le langage propositionnel qui sera utilisé dans la suite de ce chapitre.

Action nulle

Nous notons ϵ l'action applicable sur tout état e et dont l'application sur e n'a aucun effet :

- $pre(\epsilon) = \emptyset$;
- $\mathit{eff}^+(\epsilon) = \emptyset$;
- $\mathit{eff}^-(\epsilon) = \emptyset$.

Propriétés

Deux actions sont inconsistantes si l'une nie au moins un effet de l'autre.

Définition 4.5 (Inconsistance entre actions)**.** *Deux actions a_1 et a_2 sont inconsistantes (*$\texttt{inconsistance}(a_1, a_2)$*) si et seulement si $\mathit{eff}^-(a_1) \cap \mathit{eff}^+(a_2) \neq \emptyset \vee \mathit{eff}^-(a_2) \cap \mathit{eff}^+(a_1) \neq \emptyset$.*

Deux actions interfèrent si l'une supprime une précondition de l'autre.

Définition 4.6 (Interférence entre actions)**.** *Deux actions a_1 et a_2 interfèrent (*$\texttt{interference}(a_1, a_2)$*) si et seulement si $\mathit{eff}^-(a_1) \cap \mathit{pre}(a_2) \neq \emptyset \vee \mathit{eff}^-(a_2) \cap \mathit{pre}(a_1) \neq \emptyset$.*

Deux actions sont dépendantes si elles sont inconsistantes ou si elles interfèrent.

Définition 4.7 (Dépendance entre actions)**.** *Deux actions a_1 et a_2 sont dépendantes (*$\texttt{dependance}(a_1, a_2)$*) si et seulement si* $\texttt{inconsistance}(a_1, a_2) \vee \texttt{interference}(a_1, a_2)$*.*

L'indépendance entre deux actions a_1 et a_2 est leur non dépendance.

4.1.7 Système de transition d'états

Définition 4.8 (Système de transition d'états)**.** *Un système de transition d'états est un triplet $\Sigma = (\mathcal{U}, \mathcal{A}, \gamma)$, avec :*

- *$\mathcal{U}$ un sous-ensemble de l'ensemble des parties de $\mathcal{L}$*[1]*, appelé l'univers de Σ, donc un ensemble fini d'états ;*
- *$\mathcal{A}$ un ensemble fini d'actions ;*
- *$\gamma : \mathcal{U} \times \mathcal{A} \to \mathcal{U}$ une fonction non déterministe de transition d'états.*

La fonction de transition γ sur $e_i \in \mathcal{U}$ et $a \in \mathcal{A}$ est l'application de a sur e_i :

$$e_{i+1} = \gamma(e_i, a) = \begin{cases} (e_i \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a) & \textit{si } \mathit{pre}(a) \subseteq e_i \\ \textit{indéfini} & \textit{sinon} \end{cases}$$

L'univers $\mathcal{U}$ est fermé selon γ : si $e \in \mathcal{U}$, alors $\forall a \in \mathcal{A}$ qui sont applicables sur e, $\gamma(e, a) \in \mathcal{U}$. Ce système de transition d'états peut être représenté par un graphe orienté dont les noeuds sont les états dans $\mathcal{U}$, et lorsque l'action a est applicable dans l'état $e_i \in \mathcal{U}$ et que $e_{i+1} = \gamma(e_i, a)$, alors l'arc entre le noeud e_i vers le noeud e_{i+1} est décoré de l'action a. Ainsi, nous parlerons de transitions plutôt que d'actions à propos du système de transition d'états sous forme de graphe.

Pour illustrer les prochains concepts, nous utiliserons le système de transition représenté à la figure 4.1. Dans cet exemple, nous considérons sept états différents dont nous ne détaillons pas le contenu propositionnel, numérotés de 0 à 6. L'ensemble des actions de ce système de transition d'états est $\{\texttt{RAZ}, \texttt{SUCC(1)}, \texttt{SUCC(2)}, \texttt{SUCC(3)}, \texttt{SUCC(4)}, \texttt{DOUBLE(2)}, \texttt{DOUBLE(3)}\}$. Le graphe nous indique par exemple que l'application de l'action `SUCC(1)` sur l'état 1 produit l'état 2.

4.1.8 Domaine et problème de planification

Définition 4.9 (Domaine de planification)**.** *Un domaine de planification $\mathcal{D}$ est un système de transition d'états.*

La figure 4.1 page suivante représente donc un domaine de planification.

Définition 4.10 (Problème de planification)**.** *Un problème $\mathcal{P}$ de planification est un triplet $(\mathcal{D}, e_0, \mathit{goal})$, avec :*

1. l'ensemble des parties de $\mathcal{L}$ est souvent noté $\mathfrak{P}(\mathcal{L})$ et $\mathfrak{P}(\mathcal{L}) = 2^{\mathcal{L}}$

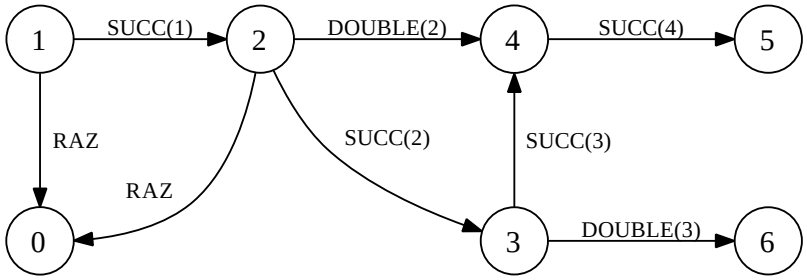


FIGURE 4.1 – Exemple de graphe représentant un système de transition d'états dont l'univers $\mathcal{U}$ est composé de sept états numérotés de 0 à 6 et dont la fonction γ est illustrée par les arcs du graphe.

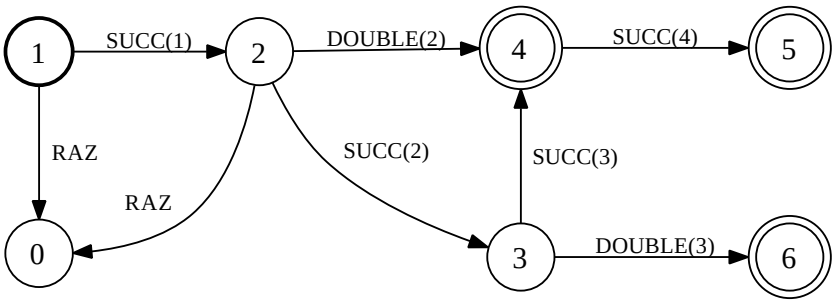


FIGURE 4.2 – Exemple de problème de planification représenté par son système de transition d'états Σ, et où l'état 1 est l'état initial e_0 et où les états 4, 5 et 6 satisfont l'objectif de planification, c'est-à-dire sont dans $\mathcal{E}_g$.

- $\mathcal{D} = (\mathcal{U}, \mathcal{A}, \gamma)$, *le domaine de planification ;*
- $e_0 \in \mathcal{U}$, *un état initial ;*
- *goal, l'objectif de planification, qui est un ensemble de propositions à satisfaire.*

Un état but $e_g \in \mathcal{U}$ satisfait l'objectif *goal*, noté $e_g \models goal$, si toutes les propositions de l'objectif sont satisfaites dans l'état e_g :

$$e_g \models goal \text{ si et seulement si } goal \subseteq e_g$$

Nous notons $\mathcal{E}_g$ l'ensemble des états dans l'univers qui satisfont l'objectif :

$$\mathcal{E}_g = \{e_g \in \mathcal{U} \mid e_g \models goal\}$$

Sur la figure 4.2, l'état initial est l'état 1 et l'ensemble des états qui satisfont l'objectif sont les états 4, 5 et 6.

En pratique, un domaine de planification $\mathcal{D}$ est défini dans un fichier de description `PDDL` [AD05] comme présenté à l'annexe B.3 page 186 pour GRIPPER. Il est décrit par un ensemble d'opérateurs et de prédicats dont tous les paramètres sont des variables. Le problème de planification $\mathcal{P}$ est décrit par un ensemble de constantes, par un ensemble

de propositions décrivant l'état initial, et par un ensemble de propositions pour objectif de planification. Ces propositions sont issues des prédicats définis dans $\mathcal{D}$. L'ensemble des actions $\mathcal{A}$ et l'ensemble des propositions $\mathcal{L}$ sont donc décrits en intention : ils sont déduits de la description de $\mathcal{D}$ et de $\mathcal{P}$ en instanciant chaque prédicat et chaque opérateur de $\mathcal{D}$ avec tous les sous-ensembles de constantes de $\mathcal{P}$.

4.2 Plan d'actions

Dans ce paragraphe, nous nous intéressons à la description du plan calculé par le planificateur :

$$\xrightarrow{\text{problème}} \text{Planificateur} \xrightarrow{\text{plan}} \text{Contrôleur}$$

Informellement, un plan peut être vu comme un chemin dans le système de transition d'états : une succession de transitions. Par exemple, la figure 4.3 représente un plan, noté [2]

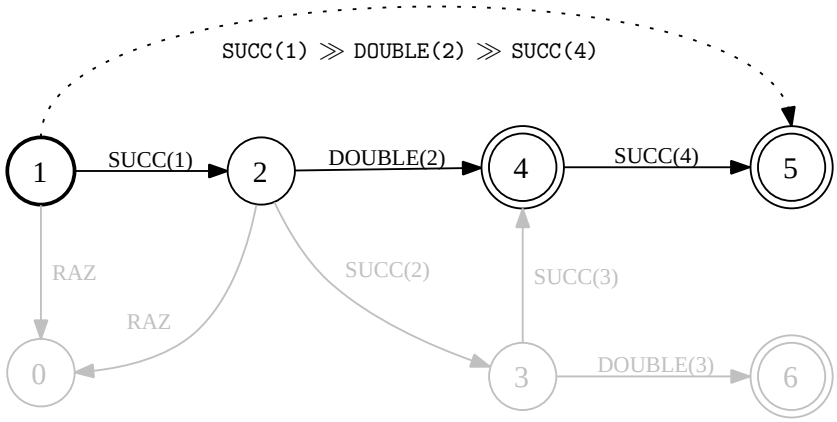


FIGURE 4.3 – Le plan `SUCC(1)` ≫ `DOUBLE(2)` ≫ `SUCC(4)` dans le domaine illustré permet de passer de l'état 1 à l'état 5 par les transitions `SUCC(1)`, `DOUBLE(2)` et `SUCC(4)`. La transition en pointillée n'existe pas dans la définition du système de transition d'états : c'est une représentation en intention de la composition de transitions existantes.

`SUCC(1)` ≫ `DOUBLE(2)` ≫ `SUCC(4)`, qui emprunte la transition `SUCC(1)` issue de l'état 1, puis la transition `DOUBLE(2)` et finalement `SUCC(4)` pour atteindre l'état 5. Nous distinguons deux représentations d'un plan :

1. en extension par la description de sa sémantique *opérationnelle* [Plo04] dans le système de transition d'états c'est-à-dire la succession des états traversés : c'est l'ensemble ordonné des actions du plan qui, appliquées séquentiellement, permettent de connaitre le chemin emprunté lors de son application dans le système de transition d'états. Par exemple dans la figure 4.3 une transition issue de l'état 1 est traversée par l'action `SUCC(1)` pour atteindre l'état 2, sur lequel `DOUBLE(2)` est appliquée et traverse la transition associée jusqu'à l'état 4 pour finalement atteindre l'état 5 par la transition issue de l'application de `SUCC(4)` ;
2. en intention, en définissant la sémantique *dénotationnelle* [SS71] d'un plan, c'est-à-dire la transition « virtuelle » que le plan définit : les préconditions et les effets du plan permettent de passer d'un état à un autre, tout comme sont définies les actions.

2. L'opérateur de composition ≫ est défini au paragraphe 4.2.2

Par exemple, le plan de la figure 4.3 est applicable sur l'état 1 et permet d'atteindre l'état 5 par son application sur l'état 1.

Nous proposons une modélisation des plans unifiant ces deux représentations. Nous considérons la notion de plan comme une généralisation de celle d'action[3], et apportons un opérateur de séquence permettant de construire des plans. Un modèle de plan est représenté à la figure 4.4. Nous modélisons les actions comme étant des plans.

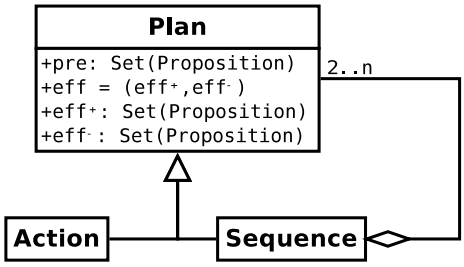


FIGURE 4.4 – Modèle de plan : une action est un plan, et une composition par l'opérateur n-aire de séquence de plans est un plan.

Nous détaillons dans la suite :

1. la formalisation des plans et la sémantique dénotationnelle associée, c'est-à-dire la transition dans le système de transition d'états résultant d'un plan donné ;
2. la définition de l'opérateur de séquence entre plans ;
3. la syntaxe des plans en tant qu'expression bien formée utilisant l'opérateur de séquence ; et la sémantique opérationnelle des plans issue de cette syntaxe, en termes d'enchaînement d'états successifs dans le système de transition d'états ;
4. les propriétés remarquables des plans.

En annexe D.1 page 205, nous démontrons toutes les propriétés déduites de nos définitions dans la suite de ce chapitre, et notamment que les sémantiques dénotationnelle et opérationnelle sont équivalentes : la définition de l'opérateur de séquence correspond bien au sens que nous lui donnons.

4.2.1 Sémantique dénotationnelle d'un plan

Le définition formelle, en intention, d'un plan, identique à celle des actions (cf. définition 4.4), est la suivante :

Définition 4.11 (Plan (intention))**.** *Un plan π est un tuple $(nom(\pi), pre(\pi), eff(\pi))$:*
- *$nom(\pi)$ est son identifiant ;*
- *$pre(\pi)$ est un ensemble de propositions de $\mathcal{L}$ et représente les préconditions du plan ;*
- *$eff(\pi)$ est un couple d'ensembles de propositions $eff^+(\pi)$ et $eff^-(\pi)$, qui représentent respectivement les effets positifs et négatifs du plan. Ces ensembles sont disjoints :*

$$eff^+(\pi) \cap eff^-(\pi) = \emptyset$$

Le plan π est applicable *dans un état e si ses préconditions sont incluses dans e :*

$$\pi \text{ est applicable dans } e \text{ si et seulement si } pre(\pi) \subseteq e$$

3. Nous englobons dans la notion d'action toutes les actions de $\mathcal{A}$, ainsi que ϵ.

	idempotence	associativité	commutativité
$\gg$	non	oui	non

TABLE 4.1 – Lois algébriques de l'opérateur de séquence.

L'application de π sur l'état e_i, si π est applicable dans e_i, amène dans l'état e_{i+1} qui est l'état e_i dont les effets négatifs de π ont été supprimés et dont les effets positifs de π ont été ajoutés :

$$\pi \textit{ appliqué sur } e \textit{ produit l'état } (e \setminus \mathit{eff}^-(\pi)) \cup \mathit{eff}^+(\pi)$$

4.2.2 Opérateur de séquence

Nous proposons un opérateur qui traduit le franchissement de transitions successives dans le système de transition d'états. Nous l'appelons l'opérateur de séquence, noté « $\gg$ ».

Opérateur binaire de séquence

L'opérateur binaire de séquence compose deux plans π_1 et π_2 pour donner comme résultat un nouveau plan $\pi = \pi_1 \gg \pi_2$. Les préconditions et les effets du plan π sont déduits de la définition des plans π_1 et π_2 :

- $nom(\pi) = nom(\pi_1) \gg nom(\pi_2)$;
- $pre(\pi) = pre(\pi_1) \cup (pre(\pi_2) \setminus \mathit{eff}^+(\pi_1))$;
- $\mathit{eff}^+(\pi) = (\mathit{eff}^+(\pi_1) \setminus \mathit{eff}^-(\pi_2)) \cup \mathit{eff}^+(\pi_2)$;
- $\mathit{eff}^-(\pi) = (\mathit{eff}^-(\pi_1) \setminus \mathit{eff}^+(\pi_2)) \cup \mathit{eff}^-(\pi_2)$.

Soient deux plans π_1 et π_2 tels que $\mathit{eff}^+(\pi_1) \cap \mathit{eff}^-(\pi_1) = \emptyset$ et $\mathit{eff}^+(\pi_2) \cap \mathit{eff}^-(\pi_2) = \emptyset$, alors le plan $\pi = \pi_1 \gg \pi_2$ est tel que $\mathit{eff}^+(\pi) \cap \mathit{eff}^-(\pi) = \emptyset$. Les propriétés des plans sont conservées par la composition séquentielle de plans.

La table 4.1 définit les lois qui régissent l'opérateur de séquence. Ces propriétés sont détaillées en annexe C page 191.

Opérateur n-aire de séquence

Nous proposons un opérateur n-aire qui étend l'opérateur binaire de séquence. Cet opérateur traduit le franchissement d'un ensemble de transitions successives dans le système de transition d'états. Il compose n plans $\pi_1, \cdots, \pi_n$ pour donner comme résultat un nouveau plan $\pi = \gg (\pi_1, \cdots, \pi_n) = \ggg_{i=1}^{n} (\pi_i)$. Cet opérateur n-aire est défini par l'opérateur binaire de séquence :

$$\ggg_{i=1}^{n>1} (\pi_i) = \pi_1 \gg \cdots \gg \pi_n$$

À noter que pour $n = 1$, nous avons $\gg (\pi) = \pi$.

L'opérateur de séquence n'étant pas commutatif, l'ordre des plans $\pi_1, \cdots, \pi_n$ importe. Avec l'opérateur n-aire de séquence, le plan `SUCC(1)` $\gg$ `DOUBLE(2)` $\gg$ `SUCC(4)` de l'exemple de la figure 4.3 se réécrit $\gg$ (`SUCC(1)`, `DOUBLE(2)`, `SUCC(4)`).

4.2.3 Sémantique opérationnelle d'un plan

Nous proposons une syntaxe des plans par une expression bien formée sur les actions, alors composées par l'opérateur de séquence. Cette syntaxe sera étendue en section 5.2.1 page 67 pour la description des plans flexibles.

Définition 4.12 (Plan (extension))**.** *Un plan π est bien formé si sa syntaxe est une expression bien formée en vertu des règles de formation suivantes :*

- *ϵ est un plan ;*
- *si a est une action de $\mathcal{A}$, alors a est un plan ;*
- *si π et π' sont deux plans, alors $(\pi \gg \pi')$ est un plan.*
- *si $\pi_1, \cdots, \pi_n$ sont n plans (avec $n > 1$), alors $\ggg_{i=1}^{n}(\pi_i)$ est un plan.*

Cette définition est conforme au modèle des plans proposé en introduction : toute action est un plan, et la séquence entre n plans est un plan. La *longueur* d'un plan π, est le nombre d'actions qui le compose, notée $|\pi|$.

Par exemple, `PICK(b-1, r-a)` $\gg$ `MOVE(r-a, r-z)` $\gg$ `DROP(b-1, r-z)` est le plan π dans GRIPPER de longueur 3 qui permet de déplacer la balle `b-1` de la pièce `r-a` à la pièce `r-z`, avec :

- $pre(\pi) = \{$`BALL(b-1)`, `ROOM(r-a)`, `ROOM(r-z)`, `G-AT(r-a)`, `AT(b-1, r-a)`, `FREE`$\}$, ce qui signifie que pour pouvoir appliquer ce plan, il faut être dans un état dans lequel :
  - il y a une balle `b-1` ;
  - il y a un emplacement `r-a` et un emplacement `r-z` ;
  - le robot et la balle `b-1` se situent dans l'emplacement `r-a` ;
  - le robot a sa pince vide.
- $eff^+(\pi) = \{$`G-AT(r-z)`, `AT(b-1, r-z)`, `FREE`$\}$: le robot se situera dans l'emplacement `r-z` du fait de l'action `MOVE(r-a, r-z)` après l'application de π ; la balle `b-1` se situe dans l'emplacement `r-z` et le robot a sa pince vide du fait de l'action `DROP(b-1, r-z)`, dont les préconditions sont soutenues par l'action `PICK(b-1, r-a)` de π ;
- $eff^-(\pi) = \{$`G-AT(r-a)`, `AT(b-1, r-a)`$\}$, qui signifie qu'après avoir appliqué ce plan, la balle `b-1` et le robot ne se situent plus dans l'emplacement `r-a`.

Par sa définition et ses propriétés, un plan est très proche de la notion de macro-opérateur définie dans [BEMS05, CFS07, NLFL07] :

- dans [CFS07], la définition diffère de peu puisque seul l'ensemble eff^+ n'est pas calculé de la même façon ;
- dans [BEMS05], la définition est décrite sous forme algorithmique selon la longueur du macro-opérateur ;
- dans [NLFL07], les macro-opérateurs sont décrits comme le résultat d'une opération binaire, non-commutative et associative évaluée par régression, mais non définie formellement.

De fait, les travaux cités composent dans les « macros » des opérateurs de planification (*cf.* définition 4.1), et manipulent donc des variables. Ce n'est pas notre cas : l'opérateur de séquence est appliqué sur des actions. En plus d'expliciter l'opérateur de séquence, nous démontrons que la définition du plan correspond à sa sémantique en tant que composition d'actions appliquées séquentiellement.

Grâce à la définition de l'opérateur de séquence, nous montrons que l'état produit par l'application d'un plan π sur l'état e est l'état produit par l'application dans l'ordre séquentiel des plans qui forment π. Ainsi, la fonction de transition à un plan π peut être définie selon la composition de π :

– si $\pi = \epsilon$, alors :

$$\gamma(e, \pi) = e$$

– si $\pi = a$ avec $a \in \mathcal{A}$, alors :

$$\gamma(e, \pi) = \begin{cases} \gamma(e, a) & \text{si } a \text{ est applicable sur } e \\ \text{indéfini} & \text{sinon} \end{cases}$$

– si $\pi = (\pi_1 \gg \pi_2)$:

$$\gamma(e, \pi) = \begin{cases} \gamma(\gamma(e, \pi_1), \pi_2) & \text{si } \gamma(e, \pi_1) \text{ est défini} \\ \text{indéfini} & \text{sinon} \end{cases}$$

– si $\pi = \ggg_{i=1}^{n} (\pi_i)$, avec $n > 2$:

$$\gamma(e, \pi) = \begin{cases} \gamma(\gamma(e, \pi_1), \ggg_{i=2}^{n} (\pi_i)) & \text{si } \gamma(e, \pi_1) \text{ est défini} \\ \text{indéfini} & \text{sinon} \end{cases}$$

Pour rappel et pour prendre en compte le cas $n = 2$, $\ggg (\pi_1, \pi_2) = (\pi_1 \gg \pi_2)$. Dans cette définition de la fonction de transition d'état sur un plan, nous considérons le plan non plus en intention, c'est-à-dire par ses préconditions et ses effets, mais comme une composition d'actions. Nous généralisons cette définition par l'associativité de l'opérateur de séquence, ce qui nous permet de définir les plans en extension, c'est-à-dire par un ensemble ordonné d'actions. Trivialement, tout plan π a une écriture « aplatie » équivalente en termes de sémantiques dénotationnelle et opérationnelle : l'opérateur de séquence compose alors uniquement les actions de π et $\pi = \ggg_{i=1}^{n} a_i$ avec $\forall i, a_i \in \mathcal{A}$.

La figure 4.5 page suivante représente le plan $\texttt{A} \gg \texttt{B} \gg \texttt{C}$, d'une part défini en intention dans le tableau et d'autre part illustré en extension dans le système de transition d'états. Ainsi, nous observons que les deux représentations du plan, en intention ou en extension, sont équivalentes quant à l'état résultant de l'application du plan sur un état donné. Le franchissement de toutes les transitions étiquetées par les actions qui composent le plan π sur l'état $\{o, l, e, a\}$ mène dans l'état $\{o, d, e\}$:

$$\begin{aligned} \gamma(\{o, l, e, a\}, \pi) =& \gamma(\{o, l, e, a\}, \texttt{A} \gg \texttt{B} \gg \texttt{C}) \\ =& \gamma(\gamma(\{o, l, e, a\}, \texttt{A}), \texttt{B} \gg \texttt{C}) \\ =& \gamma(\{s, l, o, e\}, \texttt{B} \gg \texttt{C}) \\ =& \gamma(\gamma(\{s, l, o, e\}, \texttt{B}), \texttt{C}) \\ =& \gamma(\{o, e, i, l\}, \texttt{C}) \\ =& \{o, d, e\} \end{aligned}$$

Nous observons que π est bien applicable sur l'état $\{o, l, e, a\}$ puisque $pre(\pi) = \{l, a\} \subseteq \{o, l, e, a\}$, et l'état $\{o, d, e\}$ vaut effectivement $(\{o, l, e, a\} \setminus \mathit{eff}^-(\pi)) \cup \mathit{eff}^+(\pi)$:

$$\begin{aligned} \{o, d, e\} =& (\{o, l, e, a\} \setminus \{l, a, s, i\}) \cup \{d, o\} \\ =& \{o, e\} \cup \{d, o\} \\ =& \{o, d, e\} \end{aligned}$$

Cela illustre l'équivalence entre les définitions en intention et en extension des plans.

L'écriture en extension est la définition généralement utilisée dans la littérature pour définir les plans séquentiels : dans [GNT04], un plan est une liste d'actions. Nous sommes au plus près de sa sémantique vis à vis du système de transition d'états. Nous utiliserons indifféremment l'une ou l'autre des écritures selon les propriétés que nous souhaitons mettre en avant. Nous montrons que les définitions de π en intention et en extension sont, du point de vue de γ, équivalentes :

- l'ensemble $pre(\pi)$ forme les préconditions de π de telle façon que si $pre(\pi) \subseteq e$ alors $\gamma(e, \pi)$ est défini ;
- $\gamma(e, \pi) = e \setminus \mathit{eff}^-(\pi) \cup \mathit{eff}^+(\pi)$.

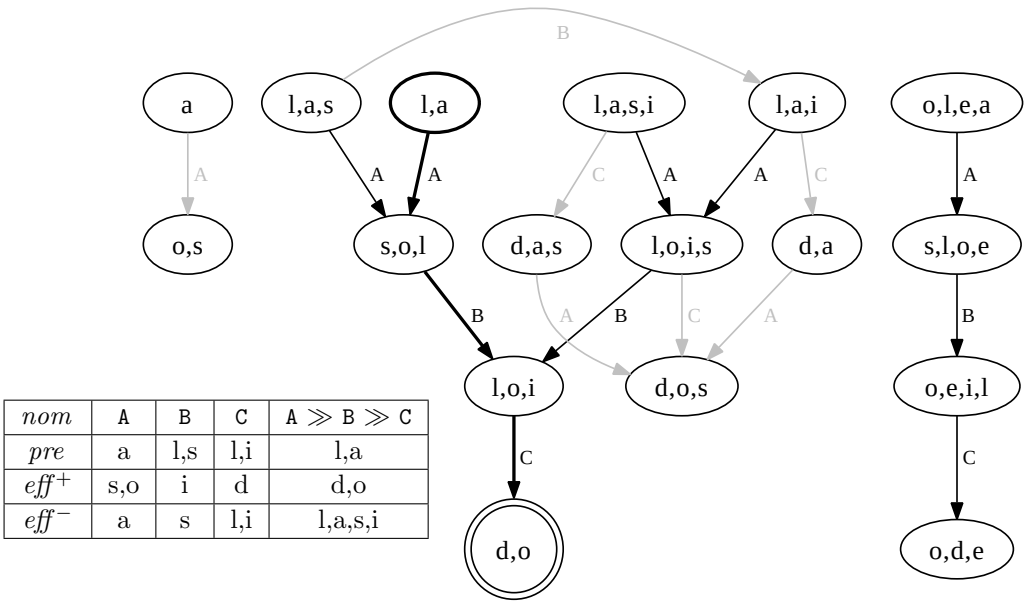
Cela permet d'en déduire que la notion de plan généralise par sa définition et ses propriétés la notion d'action : une action de $\mathcal{A}$ est un plan de longueur 1, et ϵ le plan de longueur 0.

4.2.4 Propriétés des plans

Plan solution

Un plan π est solution du problème de planification $\mathcal{P} = (\mathcal{D}, e_0, goal)$ si l'objectif *goal* est satisfait dans l'état atteint par l'application de π sur l'état initial e_0. Le plan de la figure 4.6 est un plan solution pour le problème donné.

Définition 4.13 (Plan solution)**.** *Un plan π est un plan solution de $\mathcal{P}$ si et seulement si $\gamma(e_0, \pi) \models goal$.*



nom	A	B	C	A ≫ B ≫ C
pre	a	l,s	l,i	l,a
eff^+	s,o	i	d	d,o
eff^-	a	s	l,i	l,a,s,i

FIGURE 4.5 – Dans ce domaine de planification représenté par un système de transition d'états, chaque lettre minuscule est une proposition, et les actions `A`, `B` et `C` sont détaillées par leurs préconditions et effets dans le tableau. Le plan `A` ≫ `B` ≫ `C` est proposé *en intention* dans ce même tableau, et mis en valeur *en extension* dans le système de transition d'états. Le plan mène de l'état initial $\{l, a\}$ à l'état objectif $\{d, o\}$.

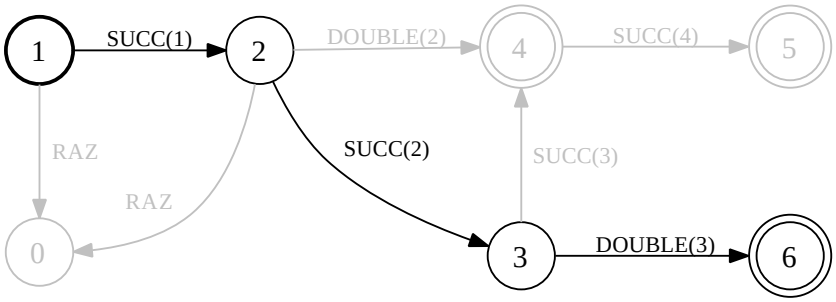


FIGURE 4.6 – Exemple de plan $\pi = \texttt{SUCC(1)} \gg \texttt{SUCC(2)} \gg \texttt{DOUBLE(3)}$ solution du problème de planification où l'état 1 est l'état initial et les états 4, 5 et 6 satisfont l'objectif de planification.

Interaction entre plans

Deux plans sont inconsistants si certaines de leurs actions sont inconstantes.

Définition 4.14 (Inconsistance entre plans)**.** *Deux plans π_1 et π_2 sont inconsistants (*$\texttt{inconsistance}(\pi_1, \pi_2)$*) si et seulement si*

$$\exists a_1 \in \pi_1, \exists a_2 \in \pi_2, \texttt{inconsistance}\,(a_1, a_2)$$

Deux plans interfèrent si certaines de leurs actions interfèrent.

Définition 4.15 (Interférence entre plans)**.** *Deux plans π_1 et π_2 interfèrent (*$\texttt{interference}(\pi_1, \pi_2)$*) si et seulement si*

$$\exists a_1 \in \pi_1, \exists a_2 \in \pi_2, \texttt{interference}\,(a_1, a_2)$$

Nous définissons la notion de dépendance pour les plans.

Définition 4.16 (Dépendance entre plans)**.** *Deux plans π_1 et π_2 sont dépendants (*$\texttt{dependance}(\pi_1, \pi_2)$*) si et seulement si*

$$\exists a_1 \in \pi_1, \exists a_2 \in \pi_2, \texttt{dependance}\,(a_1, a_2)$$

L'indépendance entre deux plans est leur non dépendance.

4.3 Conclusion

Ce chapitre propose une nouvelle vision des plans en les considérant comme des compositions. Ainsi, tout en étant fondé sur les concepts usuellement rencontrés en planification classique, le modèle proposé intègre l'opérateur de séquence. Cet opérateur est utilisé pour définir le concept de plan et les sémantiques dénotationnelle et opérationnelle qui y sont associées. Le chapitre suivant utilise la même approche de modélisation pour la planification dite flexible, par la définition de nouveaux opérateurs de composition.

Chapitre 5

Modèle de planification flexible

Ce chapitre définit en partie 5.1 la flexibilité en planification comme étant la faculté du contrôleur d'adapter le plan solution au contexte d'exécution. Dans la partie 5.2, nous proposons un modèle des plans flexibles par le concept de polyplan.

5.1 Flexibilité en planification

En général, un problème de planification particulier a plusieurs solutions. Dans notre modèle de planification classique (*cf.* chapitre 4), la seule caractéristique permettant de comparer les plans est leur longueur. C'est la raison pour laquelle les travaux qui s'inscrivent dans ce modèle s'accordent pour évaluer les plans selon cette caractéristique : le meilleur plan solution est alors le plus court. Le premier plan minimal trouvé est proposé au contrôleur. Dans notre problématique, nous considérons que la qualité des plans dépend du contexte d'exécution (*cf.* chapitre 1). Or, dans la décomposition fonctionnelle de la planification, le planificateur est chargé de la *conception du plan*, et le contrôleur est chargé de contrôler l'*exécution du plan*. Les temps de conception et d'exécution sont distincts : le planificateur n'a pas connaissance du contexte d'exécution et n'est donc pas à même d'évaluer la qualité des plans. Au moment de la conception, il n'est pas possible de savoir si un plan solution particulier est le meilleur plan possible. Nous proposons de décaler du temps de conception au temps d'exécution le choix d'un plan dans un ensemble de plans solutions.

La flexibilité en planification est définie par la capacité d'un plan à proposer des alternatives à l'exécution, sachant que chacune des alternatives permet d'atteindre un état qui satisfait l'objectif. La flexibilité est donc une propriété des plans : nous parlons de *plans flexibles*. Un plan flexible calculé par le planificateur est adaptable au contexte d'exécution par le contrôleur :

$$\xrightarrow{\text{problème}} \text{Planificateur} \xrightarrow{\text{plan flexible}} \text{Contrôleur} \xrightarrow{\text{action}} \text{Système de transition d'états}$$

Alors qu'un plan est une séquence des actions déterministes, un plan flexible propose des alternatives : l'état atteint par l'application du plan flexible dépend des choix effectués à l'exécution. Plusieurs exécutions d'un même plan flexible sur le même état peuvent mener dans des états différents. Un plan flexible n'est donc pas forcément déterministe. Mais tous les états possiblement atteints par l'exécution d'un plan flexible solution satisfont l'objectif de planification. Dans le système de transition d'états, l'exécution d'un plan flexible correspond aux franchissements successifs de transitions, ce qui correspond à l'exécution d'un plan séquentiel. Mais à l'exécution d'un plan flexible, certaines transitions sont choisies dans un ensemble d'alternatives, par le contrôleur. Un plan flexible peut être vu comme un ensemble de plans séquentiels dont l'un est choisi par le contrôleur à l'exécution. Les plans séquentiels sont dits *dérivés* du plan flexible.

La figure 5.1 page suivante représente le plan flexible Π qui propose comme solution soit le plan `SUCC(1)` $\gg$ `DOUBLE(2)` $\gg$ `SUCC(4)` soit le plan `SUCC(1)` $\gg$ `SUCC(2)` $\gg$ `DOUBLE(3)`. Depuis l'état 1, Π permet d'atteindre soit l'état 5 soit l'état 6 selon le choix effectué à

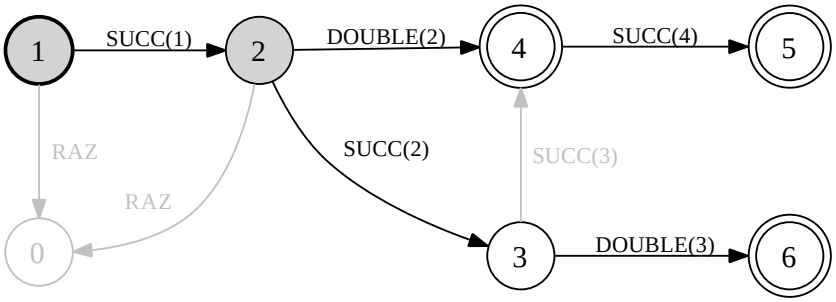


FIGURE 5.1 – Exemple de plan flexible dans le système de transition d'états qui permet de passer de l'état 1 à l'état 5 ou 6, respectivement par les plans `SUCC(1)` ≫ `DOUBLE(2)` ≫ `SUCC(4)` et `SUCC(1)` ≫ `SUCC(2)` ≫ `DOUBLE(3)` selon la décision prise dans l'état 1 ou l'état 2.

l'exécution. Les deux plans sont dérivés de Π : ce sont les deux alternatives proposées par Π. Comme la flexibilité est une caractéristique des plans exploitée à leurs exécution, nous nous rapportons au système de transition d'états pour la définir. Ainsi, nous appelons les plans dérivés de Π des *chemins* dans le système de transition d'états, et les états desquels les chemins diffèrent des *bifurcations*. Les différentes actions applicables sur une certaine bifurcation sont appelées les *alternatives* de la bifurcation. En reprenant l'exemple précédent, deux chemins sont disponibles depuis l'état 1 : `SUCC(1)` ≫ `DOUBLE(2)` ≫ `SUCC(4)` et `SUCC(1)` ≫ `SUCC(2)` ≫ `DOUBLE(3)`. Ces chemins diffèrent au niveau de l'état 2 qui est donc une bifurcation. En effet, l'état 2 est atteint par l'application de l'action `SUCC(1)` commune aux deux chemins. Le nombre de chemins accessibles à partir d'une bifurcation donnée est au moins égale à deux. À l'exécution, l'état dans lequel est prise la décision d'une alternative est un *moment* de décision. Un moment de décision permet d'éliminer au moins un chemin parmi l'ensemble des possibles. Tous les états traversés avant une bifurcation donnée sont des *moments potentiels* de décision. Dans l'exemple de la figure 5.1, les moments potentiels de décision permettant de discriminer l'un des deux chemins sont l'état 1 et l'état 2.

5.1.1 Dimensions de la flexibilité

Nous distinguons deux dimensions dans la flexibilité d'un plan :
- une dimension spatiale, liée aux chemins disponibles dans un plan flexible : cette dimension est définie au moment de la conception du plan ;
- une dimension temporelle pour les choix effectués au moment de l'exécution du plan.

Espace

La spatialité de la flexibilité concerne la quantité d'alternatives différentes décrites par un plan flexible. La *largeur*[1] d'un plan flexible Π est le nombre de plans dérivés de Π. La largeur caractérise la spatialité de la flexibilité d'un plan.

Le planificateur cherche à calculer un plan flexible Π le plus large possible, tandis que le contrôleur réduit à chaque moment de décision cette largeur jusqu'à exécuter un unique plan dérivé de Π.

1. En analogie à la longueur des plans dérivés de Π, le nombre d'actions dont ils sont composés (*cf.* définition 4.12).

Temps

La temporalité de la flexibilité désigne les moments où les décisions sont prises d'emprunter une alternative d'un plan flexible plutôt qu'une autre. En général, pour une alternative donnée plusieurs moments de décision sont possibles. Cette dimension est donc liée à l'exécution du plan flexible, et ne peut pas être définie a priori dans le plan flexible : cela fait partie du non déterminisme des plans flexibles.

Le planificateur n'a aucun rôle quant à la dimension temporelle des plans flexibles. Seul le contrôleur a accès à cette dimension.

5.1.2 Critères de flexibilité

En se référant aux dimensions spatiale et temporelle, nous définissons des critères permettant d'évaluer la flexibilité d'un plan. Les critères sont :

- la largeur des plans, critère entièrement défini par la dimension spatiale ;
- l'anticipation d'une alternative, critère entièrement défini par la dimension temporelle ;
- la couverture de bifurcation, définie par les deux dimensions de la flexibilité : ce critère analyse la variabilité temporelle pour un espace donné ;
- le branchement de décision, défini par les deux dimensions de la flexibilité : ce critère analyse la variabilité spatiale pour un temps donné.

La largeur des plans flexibles étant indépendante du temps, le planificateur peut l'évaluer : ce critère sera discuté lorsque nous proposerons un modèle des plans flexibles au paragraphe 5.2.

Largeur des plans

Informellement, un plan Π est d'autant plus flexible que les chemins disponibles sont nombreux. Le nombre de chemins disponibles est la *largeur* de Π. Ce critère dépend uniquement de la dimension spatiale.

Définition 5.1 (Largeur d'un plan flexible)**.** *La largeur d'un plan flexible* Π *est le nombre de plans qui dérivent de* Π.

Dans l'exemple de la figure 5.1, la largeur du plan flexible est 2. La largeur des plans est le critère qui permet de dire d'un plan qu'il est ou qu'il n'est pas flexible : si la largeur d'un plan est au moins deux, alors le plan est flexible. Nous disons de la largeur qu'elle détermine l'expressivité des plans.

Dire d'un plan qu'il est plus large qu'un autre ne donne pas beaucoup d'informations sur la qualité des plans. En effet, un plan peut proposer de nombreux chemins qui ne conviennent pas au contrôleur à l'égard de son contexte d'exécution, tandis qu'un autre peut en proposer un unique qui lui convient. Ainsi, la largeur permet de mesurer un degré de flexibilité d'un plan, mais pas sa *qualité*[2]. Nous proposons un opérateur de comparaison de l'expressivité de plans, qui garantit que si un plan est dit plus expressif qu'un autre, il est aussi de meilleur qualité vis-à-vis du contexte d'exécution. L'opérateur de comparaison $\prec$ mesure l'expressivité entre deux plans.

Définition 5.2 (Comparaison d'expressivité)**.** *Soient deux plans flexibles* Π *et* Π'*, alors* Π *est plus expressif que* Π'*, noté* $\Pi' \prec \Pi$*, si* Π *dérive au moins tous les plans de* Π'.

Autrement dit, si $\Pi' \prec \Pi$, alors Π permet de parcourir tous les chemins de Π'.

2. La flexibilité d'un plan est un critère particulier de la qualité du plan. Les autres critères sont discutés dans le chapitre 1.

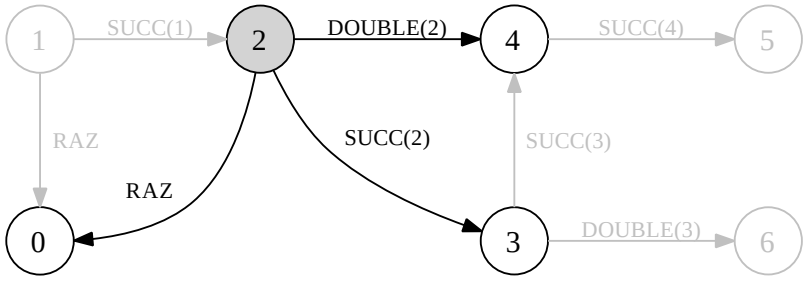


FIGURE 5.2 – Le plan flexible permet de passer de l'état 2 à l'un des trois états 0, 3 ou 4, respectivement par les plans `RAZ`, `SUCC(2)` ou `DOUBLE(2)` selon la décision prise dans l'état 2.

Anticipation d'une alternative

Dans l'exemple de la figure 5.1, considérons que le moment de décision est à l'état 1 alors que la bifurcation est à l'état 2. La décision est alors anticipée vis à vis de la bifurcation concernée : l'état 2 aussi pouvait être un moment de décision adéquat.

Définition 5.3 (Anticipation des décisions)**.** *L'anticipation d'une décision est le nombre d'actions appliquées avant la première bifurcation impliquée par la décision.*

Lorsque une décision est anticipée durant l'exécution, son effet n'est pas directement observable : les états traversés ne diffèrent pas quel que soit le choix effectué jusqu'à atteindre la bifurcation concernée par la décision. Par exemple, dans la figure 5.1, si le moment de décision entre les deux plans se situe dans l'état 1 : l'effet de la décision est observable uniquement lorsque l'état 3 ou l'état 4 est atteint, et l'état 2, traversé quel que soit le choix effectué, ne permet pas de différencier la décision. Une anticipation nulle pour une alternative donnée signifie que l'effet du choix est directement observable.

Un critère de comparaison de la flexibilité de l'exécution d'un plan concerne les anticipations possibles pour chacune des alternatives du plan flexible. Pour l'exemple de la figure 5.1, les états 1 et 2 sont des moments de décision potentiels : le plan flexible permet n'importe quelle anticipation. Ainsi, le contrôleur doit pouvoir prendre ses décisions selon le contexte d'exécution : que ce soit par exemple au plus tôt, c'est-à-dire avant l'exécution de la première action du plan flexible, ou au plus tard, c'est-à-dire avec une anticipation nulle. Une exécution du plan flexible qui impose le moment de décision dans l'état 1 pour l'exemple de la figure 5.1 est moins flexible que l'exécution qui ne l'impose pas.

Couverture de bifurcation

La figure 5.2 illustre un plan flexible Π dont l'unique bifurcation propose trois alternatives : Π est de largeur 3. Une unique alternative doit être choisie à l'exécution. La discrimination de l'une des alternatives peut se faire en deux moments : un premier choix parmi deux premières alternatives, puis un second choix entre l'alternative choisie précédemment et la restante. Par exemple, un premier moment de décision concerne les actions `RAZ` et `SUCC(2)` : supposons que `SUCC(2)` soit choisie. Le second moment concernera `SUCC(2)` et `DOUBLE(2)`. Pour choisir une alternative parmi n et selon le contexte d'exécution, le contrôleur peut préférer prendre une unique décision. Aussi, les décisions sont n-aires mais peuvent ne couvrir qu'un sous ensemble des alternatives d'une bifurcation.

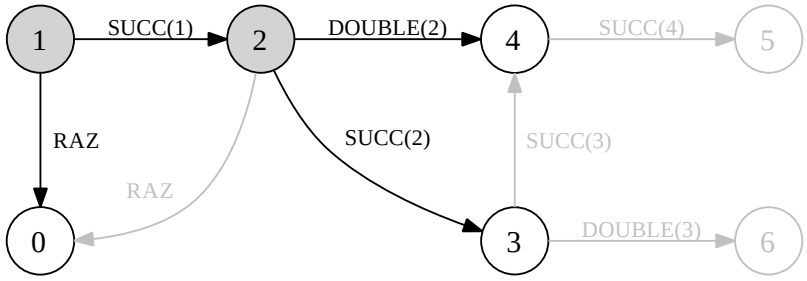


FIGURE 5.3 – Le plan flexible permet de passer de l'état 1 à l'un des trois états 0, 3 ou 4, respectivement par les plans `RAZ`, `SUCC(1)` ≫ `SUCC(2)` ou `SUCC(1)` ≫ `DOUBLE(2)`.

Définition 5.4 (Couverture des décisions)**.** *Pour une bifurcation donnée ayant n alternatives, la couverture c d'une décision liée à cette bifurcation est le rapport entre le nombre k d'alternatives éliminées par la décision et le nombre n d'alternatives moins le plan finalement discriminé et exécuté :* $c = \frac{k}{n-1}$.

La couverture vaut 1 lorsqu'une seule alternative n'a pas été éliminée par le choix : la couverture est alors totale, et l'alternative restante est celle qui est exécutée.

La flexibilité d'un plan a pour critère d'évaluation la couverture de chacun de ses moments de décision : selon le contexte d'exécution, une couverture plus ou moins grande peut être souhaitée. Par exemple avec PAINTER où l'objectif est de peindre un ensemble d'objets de plusieurs couleurs, considérons une bifurcation qui a pour alternatives l'application de chaque couleur sur chaque objet. Un plan flexible pourrait proposer de commencer par discriminer l'objet à peindre parmi ceux disponibles, puis de discriminer la prochaine couleur à appliquer. Cela correspond à une couverture partielle de la bifurcation, et peut être plus adapté au contexte qu'un unique choix sur l'objet à peindre et la couleur à appliquer.

Branchement de décision

Une décision peut impliquer plusieurs bifurcations. Un plan flexible à deux bifurcations est illustré à la figure 5.3. À l'exécution, si le plan `SUCC(1)` ≫ `SUCC(2)` est choisi par une unique décision prise dans l'état 1, alors cette décision implique deux bifurcations, la première à l'état 1 et la seconde à l'état 2. Le branchement du moment de décision est alors double.

Définition 5.5 (Branchement des décisions)**.** *Le branchement d'une décision est le nombre de bifurcations impliquées par la décision.*

Dans les exemples précédents, chaque décision n'impliquait qu'une seule bifurcation : leur branchement était minimal. Le branchement des décisions définit par exemple s'il est possible en un seul moment de décision de discriminer le plan dérivé d'un plan flexible qui sera exécuté. Un branchement supérieur à un implique obligatoirement l'anticipation d'au moins une alternative.

5.1.3 Flexibilité visée par les planificateurs

La planification a pour but d'obtenir un plan Π le plus adaptable au contexte : Π est le plus large possible, et son exécution ne contraint pas les moments de décision selon les trois critères de flexibilité qui sont l'anticipation, la couverture et le branchement des alternatives.

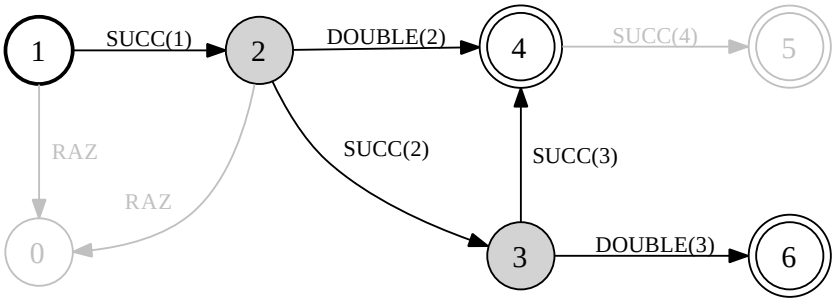


FIGURE 5.4 – Exemple de plan flexible solution dans le système de transition d'états qui permet de passer de l'état 1 à l'état 4 ou 6, respectivement par les plans `SUCC(1)` ≫ `DOUBLE(2)`, `SUCC(1)` ≫ `SUCC(2)` ≫ `SUCC(3)` et `SUCC(1)` ≫ `SUCC(2)` ≫ `DOUBLE(3)` selon les décisions prisent au plus tard dans l'état 2 pour la première bifurcation et 3 pour la seconde, en gris dans le système de transition d'états.

Ces caractéristiques concernent principalement l'interprétation par le contrôleur du plan flexible fourni par le planificateur. À l'exécution, seule la largeur est un critère défini a priori dans les plans flexibles. Ainsi, les planificateurs doivent fournir au contrôleur le plan flexible le plus large possible.

La partie 5.2 présente la modélisation que nous proposons pour les plans flexibles.

5.2 Polyplan d'actions

Le modèle de la planification flexible est fondé sur celui de la planification classique, mais diffère par l'expression des plans fournis au contrôleur. Dans les deux modèles, les actions qui décrivent le système de transition d'états sont déterministes. En revanche, en planification flexible, la solution composée par le planificateur proposera à l'exécution des choix non déterministes au contrôleur : il peut être vu comme un ensemble de plans, de telle façon que son exécution empruntera l'un des plans selon les choix effectués par le contrôleur durant l'exécution. La caractéristique des plans flexibles est de proposer des alternatives. Nous proposons le concept de *polyplan* comme extension des plans, de telle sorte qu'un polyplan est à même de représenter un ensemble de plans :

$$\xrightarrow{\text{problème}} \text{Planificateur} \xrightarrow{\textit{polyplan}} \text{Contrôleur}$$

La figure 5.4 propose un plan flexible solution du problème de planification. Notre objectif dans ce chapitre est de fournir les outils nécessaires pour exprimer un tel plan flexible, à travers notamment de la notion de polyplan. Un modèle de polyplan est représenté à la figure 5.5 page ci-contre. Le concept de polyplan étend celui de plan en intégrant un opérateur d'alternative pour offrir des choix à l'exécution. De la même façon que dans le chapitre 4 pour les plans, nous distinguons les représentation d'un polyplan *en intention* et *en extension*, ainsi que leurs sémantiques *dénotationnelle* et *opérationnelle*.

La suite de ce chapitre présente les concepts de la planification flexible :

1. la formalisation des polyplans et leur sémantique dénotationnelle ;
2. la définition des opérateurs d'alternative et de séquence entre polyplans ;

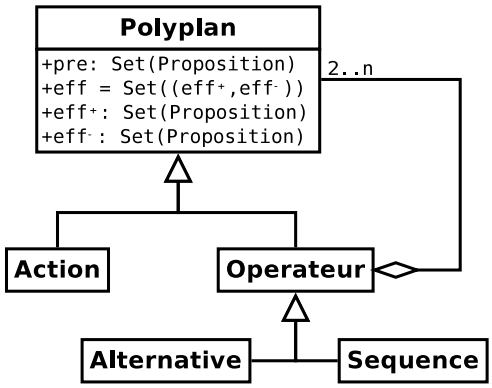


FIGURE 5.5 – Modèle de polyplan : une action est un polyplan, et une composition par les opérateurs n-aires de séquence et d'alternative de polyplans est un polyplan.

3. la syntaxe des polyplans en tant qu'expression bien formée utilisant les opérateurs d'alternative et de séquence ; et la sémantique opérationnelle des polyplans fondée sur le système de transition d'états, c'est-à-dire le comportement de Π dans le système de transition d'états lorsqu'il est exécuté par le contrôleur ;
4. les propriétés des polyplans, et notamment concernant leur flexibilité ;

5.2.1 Sémantique dénotationnelle d'un polyplan

Les effets d'un polyplan Π dépendent des choix effectués à l'exécution. Aussi, pour décrire les effets de Π nous nous inspirons de la notation utilisée pour décrire les actions non déterministes [War76, PS92, PC96]. Dans ces travaux, les effets sont conditionnés à un état particulier rencontré durant l'exécution et non prévisible au moment de la planification. Dans notre cas, le non déterminisme des plans ne dépend pas d'un état mais d'un choix à l'exécution.

Définition 5.6 (Polyplan (intention))**.** *Un polyplan* Π *est un tuple qui étend celui des plans, et* $\Pi = (nom(\Pi), pre(\Pi), eff_1(\Pi), \cdots, eff_n(\Pi))$ *où :*

- $nom(\Pi)$ *est son identifiant ;*
- $pre(\Pi)$ *est un ensemble de propositions de* $\mathcal{L}$ *et représente les préconditions du polyplan ;*
- *pour tout* i *dans* $[1..n]$, $eff_i(\Pi)$ *est un couple d'ensembles de propositions* $eff_i^+(\Pi)$ *et* $eff_i^-(\Pi)$*, qui représentent respectivement des effets positifs et négatifs du polyplan. Ces ensembles sont disjoints :*

$$eff_i^+(\Pi) \cap eff_i^-(\Pi) = \emptyset$$

Le polyplan Π *est* applicable *dans un état* e *si ses préconditions sont incluses dans* e *:*

$$\Pi \text{ est applicable dans } e \text{ si et seulement si } pre(\Pi) \subseteq e$$

L'application de Π *sur l'état* e*, si* Π *est applicable dans* e*, amène dans un des états* e_i*, avec* $i \in [1..n]$ *et un état* e_i *est l'état* e *dont les effets négatifs de* Π_i *ont été supprimés et dont les effets positifs de* Π_i *ont été ajoutés :*

$$\Pi \text{ appliqué sur } e \text{ selon le choix de } i \in [1..n] \text{ produit l'état } e_i = (e \setminus eff_i^-(\Pi)) \cup eff_i^+(\Pi)$$

La largeur *d'un polyplan est* $\#\Pi = n$*, le nombre de couples d'effets positifs et négatifs de* Π*. La largeur d'un polyplan* Π *indique le nombre de plans qui peuvent être dérivés de* Π*, et* $\forall i \in [1..\#\Pi], (nom_i(\Pi), pre(\Pi), eff_i(\Pi))$ *est un* plan dérivé *de* Π*.*

5.2.2 Opérateur d'alternative

Nous proposons un opérateur qui donne les alternatives disponibles dans le polyplan : l'opérateur d'alternative est noté « $\square$ ».

Opérateur binaire d'alternative

L'opérateur d'alternative binaire compose deux polyplans Π_1 et Π_2 pour donner comme résultat un nouveau polyplan $\Pi = \Pi_1 \square \Pi_2$ qui propose l'alternative entre Π_1 et Π_2. Les préconditions et les effets du polyplan Π sont déduits de la définition des polyplans Π_1 et Π_2, avec $k = \#\Pi_1$ et $n = \#\Pi_2$:

- $nom(\Pi) = nom(\Pi_1) \square nom(\Pi_2)$;
- $\forall i \in [1..k], nom_i(\Pi) = nom_i(\Pi_1)$;
- $\forall j \in [k+1..k+n], nom_j(\Pi) = nom_{j-k}(\Pi_2)$.
- $pre(\Pi) = pre(\Pi_1) \cup pre(\Pi_2)$;
- $\forall i \in [1..k], eff_i(\Pi) = eff_i(\Pi_1)$;
- $\forall j \in [k+1..k+n], eff_j(\Pi) = eff_{j-k}(\Pi_2)$.

La table 5.1 définit les lois qui régissent l'opérateur d'alternative. Comme l'opérateur

	idempotence	associativité	commutativité
$\square$	oui	oui	oui

Table 5.1 – Lois algébriques de l'opérateur d'alternative.

d'alternative est associatif, nous pouvons omettre les parenthèses lorsqu'elles ne sont pas nécessaires pour indiquer la portée des opérateurs.

Opérateur n-aire d'alternative

Nous proposons un opérateur n-aire qui étend l'opérateur binaire d'alternative. Cet opérateur propose l'alternative entre un ensemble de polyplans. Il compose n polyplans $\Pi_1, \cdots, \Pi_n$ pour donner comme résultat un nouveau plan $\Pi = \bigsquare(\Pi_1, \cdots, \Pi_n) = \bigsquare_{i=1}^{n}(\Pi_i)$. Cet opérateur n-aire est défini par l'opérateur binaire d'alternative :

$$\bigsquare_{i=1}^{n}(\Pi_i) = \Pi_1 \square \cdots \square \Pi_n$$

À noter que pour $n = 1$, nous avons $\bigsquare(\Pi) = \Pi$.

5.2.3 Opérateur de séquence

L'opérateur a déjà été introduit pour la composition de plans (*cf.* chapitre 4). Nous proposons ici d'étendre l'application de l'opérateur de séquence aux polyplans.

Opérateur binaire de séquence

L'opérateur de séquence, noté $\gg$, compose deux polyplans Π_1 et Π_2 pour donner comme résultat un nouveau polyplan $\Pi = \Pi_1 \gg \Pi_2$ qui est la succession de Π_1 puis Π_2, c'est-à-dire la possibilité de suivre tous les chemins proposés par Π_2 après avoir parcouru n'importe lequel de Π_1. Les préconditions et les effets du polyplan Π sont déduits de la définition des polyplans Π_1 et Π_2, avec $k = \#\Pi_1$ et $n = \#\Pi_2$:

- $nom(\Pi) = nom(\Pi_1) \gg nom(\Pi_2)$;
- $\forall (i,j) \in [1..k] \times [1..n], nom_{(i,j)}(\Pi) = nom_i(\Pi_1) \gg nom_j(\Pi_2)$;
- $pre(\Pi) = pre(\Pi_1) \cup \left(pre(\Pi_2) \setminus \bigcap_{i=1}^{k} eff_i^+(\Pi_1)\right)$;
- $\forall (i,j) \in [1..k] \times [1..n], eff_{(i,j)}(\Pi) = \left(eff^+_{(i,j)}(\Pi), eff^-_{(i,j)}(\Pi)\right)$;
- $\forall (i,j) \in [1..k] \times [1..n], eff^+_{(i,j)}(\Pi) = (eff_i^+(\Pi_1) \setminus eff_j^-(\Pi_2)) \cup eff_j^+(\Pi_2)$;
- $\forall (i,j) \in [1..k] \times [1..n], eff^-_{(i,j)}(\Pi) = (eff_i^-(\Pi_1) \setminus eff_j^+(\Pi_2)) \cup eff_j^-(\Pi_2)$;

Opérateur n-aire de séquence

Nous proposons un opérateur n-aire qui étend l'opérateur binaire de séquence entre polyplans. Cet opérateur traduit le franchissement d'un ensemble de transitions successives dans le système de transition d'états. Il compose n polyplans $\Pi_1, \cdots, \Pi_n$ pour donner comme résultat un nouveau plan $\pi = \gg (\Pi_1, \cdots, \Pi_n) = \ggg_{i=1}^{n} (\Pi_i)$. Cet opérateur n-aire est défini par l'opérateur binaire de séquence :

$$\ggg_{i=1}^{n>1} (\Pi_i) = \Pi_1 \gg \cdots \gg \Pi_n$$

À noter que pour $n = 1$, nous avons $\gg (\Pi) = \Pi$. L'opérateur de séquence n'étant pas commutatif, l'ordre des polyplans $\Pi_1, \cdots, \Pi_n$ importe.

5.2.4 Sémantique opérationnelle d'un polyplan

La syntaxe des polyplans est une expression bien formée sur les actions, composés par l'opérateur de séquence « $\gg$ » et par l'opérateur d'alternative « $\square$ ».

Définition 5.7 (Polyplan (extension))**.** *Un polyplan* Π *est une expression bien formée en vertu des règles de formation suivantes :*

- ϵ *est un polyplan ;*
- *si* a *est une action de* $\mathcal{A}$*, alors* a *est un polyplan ;*
- *si* Π *et* Π' *sont deux polyplans, alors :*
  - $(\Pi \gg \Pi')$ *est un polyplan ;*
  - $(\Pi \square \Pi')$ *est un polyplan.*
- *si* $\Pi_1, \cdots, \Pi_n$ *sont* n *polyplans, alors :*
  - $\gg (\Pi_1, \cdots, \Pi_n)$ *est un polyplan ;*
  - $\square (\Pi_1, \cdots, \Pi_n)$ *est un polyplan.*

Le polyplan de l'exemple illustré à la figure 5.4 page 66 s'écrit `SUCC(1)` $\gg$ (`DOUBLE(2)` $\square$ (`SUCC(2)` $\gg$ (`SUCC(3)` $\square$ `DOUBLE(3)`))). Le polyplan Π, construit à partir du domaine GRIPPER, tel que $\Pi =$ `MOVE(r-z, r-a)` $\gg$ (`PICK(b-1, r-a)` $\square$ `PICK(b-2, r-a)`) permet au robot de se déplacer de l'emplacement `r-z` à `r-a` pour y attraper soit la balle `b-1` soit la balle `b-2` :

- $pre(\Pi) = \{$`Ball(b-1)`, `Ball(b-2)`, `Room(r-a)`, `Room(r-z)`, `G-at(r-z)`, `At(b-1, r-a)`, `At(b-2, r-a)`, `Free`$\}$, ce qui signifie que pour pouvoir appliquer cette action, il faut être dans un état dans lequel :
  - il y a deux balles, `b-1` et `b-2` ;
  - il y a un emplacement `r-a` et un emplacement `r-z` ;
  - le robot se situe dans l'emplacement `r-z` ;
  - les balles se situent dans l'emplacement `r-a` ;
  - le robot a sa pince vide.
- pour le premier plan dérivé de Π :
  - $eff_1^+(\pi) = \{$`G-at(r-a)`, `Carry(b-1)`$\}$: après l'application de Π, le robot se situera dans l'emplacement `r-a` du fait de l'action `MOVE(r-z, r-a)` et la balle `b-1` est attrapée par le robot du fait de l'action `PICK(b-1, r-z)` ;
  - $eff_1^-(\pi) = \{$`G-at(r-z)`, `At(b-1, r-a)`, `Free`$\}$, qui signifie qu'après avoir appliqué ce plan, la balle `b-1` ne se situe plus dans l'emplacement `r-a`, le robot ne se situe plus dans l'emplacement `r-z` et n'a plus sa pince libre.
- les définitions de $eff_2^+(\Pi)$ et $eff_2^-(\Pi)$ sont similaires mais avec la balle `b-2`, et $eff_1(\Pi) = \big(eff_1^+(\Pi), eff_1^-(\Pi)\big)$ et $eff_2(\Pi) = \big(eff_2^+(\Pi), eff_2^-(\Pi)\big)$.

Grâce aux lois de distributivité des opérateurs de séquence et d'alternative (*cf.* annexe C), un polyplan peut s'écrire comme un choix parmi des plans sous forme aplatie. Le polyplan se réécrit $nom_1(\Pi)$ $\square$ $nom_2(\Pi)$, c'est-à-dire (`MOVE(r-z, r-a)` $\gg$ `PICK(b-1, r-z)`) $\square$ (`MOVE(r-z, r-a)` $\gg$ `PICK(b-2, r-z)`).

L'état produit par l'application d'un polyplan Π sur l'état e est l'état produit par l'application des actions de π dans l'ordre séquentiel ou selon le choix effectué. Nous étendons la fonction de transition γ à un plan Π :

- si $\Pi = \epsilon$, alors :

$$\gamma(e, \Pi) = e$$

- si $\Pi = a$ avec $a \in \mathcal{A}$, alors :

$$\gamma(e, \Pi) = \begin{cases} \gamma(e, a) & \text{si } a \text{ est applicable sur } e \\ \text{indéfini} & \text{sinon} \end{cases}$$

- si $\Pi = (\Pi_1 \gg \Pi_2)$:

$$\gamma(e, \Pi) = \begin{cases} \gamma(\gamma(e, \Pi_1), \Pi_2) & \text{si } \gamma(e, \Pi_1) \text{ est défini} \\ \text{indéfini} & \text{sinon} \end{cases}$$

- si $\Pi = (\Pi_1 \square \Pi_2)$:

$$\gamma(e, \Pi) = \begin{cases} \text{choix dans } \left\{ \begin{array}{l} \gamma(e, \Pi_1) \\ \gamma(e, \Pi_2) \end{array} \right\} & \text{si } \gamma(e, \Pi_1) \text{ et } \gamma(e, \Pi_2) \text{ sont définis} \\ \text{indéfini} & \text{sinon} \end{cases}$$

- si $\Pi = \ggg_{i=1}^{n} (\Pi_i)$, avec $n > 2$:

$$\gamma(e, \Pi) = \begin{cases} \gamma(\gamma(e, \Pi_1), \ggg_{i=2}^{n} (\Pi_i)) & \text{si } \gamma(e, \Pi_1) \text{ est défini} \\ \text{indéfini} & \text{sinon} \end{cases}$$

– si $\Pi = \bigsqcup_{i=1}^{n} (\Pi_i)$, avec $n > 2$:

$$\gamma(e, \Pi) = \begin{cases} \text{choix dans } \left\{ \begin{array}{r} \gamma(e, \Pi_1) \\ \vdots \\ \gamma(e, \Pi_n) \end{array} \right\} & \text{si } \gamma(e, \Pi_1), \cdots, \gamma(e, \Pi_n) \text{ sont définis} \\ \text{indéfini} & \text{sinon} \end{cases}$$

L'interprétation d'un polyplan Π dans un système de transition particulier est donnée par la traduction du polyplan en un ensemble de plans $\{\pi_1, \cdots, \pi_n\}$. Le polyplan Π donne le choix d'exécuter un plan parmi l'ensemble de plans. Nous utiliserons indifféremment l'une ou l'autre des écritures selon les propriétés que nous souhaitons mettre en avant. Ainsi, le polyplan représenté dans la figure 5.4 page 66 s'écrit en extension par l'ensemble des plans $\{\texttt{SUCC(1)} \gg \texttt{DOUBLE(2)}, \texttt{SUCC(1)} \gg \texttt{SUCC(2)} \gg \texttt{SUCC(3)}, \texttt{SUCC(1)} \gg \texttt{SUCC(2)} \gg \texttt{DOUBLE(3)}\}$. À l'exécution, un seul des trois plans proposés par Π dans le système de transition d'états sera emprunté, selon le choix effectué.

Nous déduisons de la définition des polyplans les propriétés suivantes :
– $\forall i \in [1..n], \mathit{eff}_i^+(\Pi) \cap \mathit{eff}_i^-(\Pi) = \emptyset$ (car $\mathit{eff}^+(\pi_i) \cap \mathit{eff}^-(\pi_i) = \emptyset$) ;
– l'ensemble $\mathit{pre}(\Pi)$ forme les préconditions de Π de telle façon que si $\mathit{pre}(\Pi) \subseteq e$ alors $\gamma(\Pi, e)$ est défini (car toutes les préconditions de tous les plans dont est formé Π sont dans $\mathit{pre}(\Pi)$) ;
– pour le choix d'un $i \in [1..n]$, $\gamma(\Pi, e) = e \setminus \mathit{eff}_i^-(\Pi) \cup \mathit{eff}_i^+(\Pi)$.

Cela permet d'en déduire que la notion de polyplan généralise par sa définition et ses propriétés la notion de plan. Cela s'observe autant dans sa forme en intention qu'en extension. C'est principalement dans le non déterminisme apporté par les polyplans que réside l'intérêt de la planification flexible.

5.2.5 Propriétés des polyplans

Polyplan solution

Définition 5.8 (Polyplan solution)**.** *Un* polyplan solution *pour un problème de planification donné est tel que n'importe quel état atteint par son application sur l'état initial satisfait l'objectif. Soit le problème de planification $\mathcal{P} = (\mathcal{D}, e_0, goal)$, un polyplan $\Pi = \{\pi_1, \cdots, \pi_n\}$ est solution pour $\mathcal{P}$, écrit $\gamma(e_0, \Pi) \models goal$, si et seulement si*

$$\forall i \in [1..n], \gamma(e_0, \pi_i) \models goal$$

La figure 5.6 page suivante illustre un polyplan solution. Ce polyplan appliqué sur l'état 1 permet d'atteindre, au choix, l'un des trois états qui satisfont l'objectif de planification. Il y a d'autres plans flexibles solution du problème.

Interaction entre polyplan

Définition 5.9 (Dépendance entre polyplans)**.** *Deux polyplans Π_1 et Π_2 sont dépendants (*$\texttt{dependance}(\Pi_1, \Pi_2)$*) si et seulement si*

$$\exists \pi_1 \in \Pi_1, \exists \pi_2 \in \Pi_2, \texttt{dependance}\,(\pi_1, \pi_2)$$

L'indépendance entre deux polyplans est leur non dépendance.

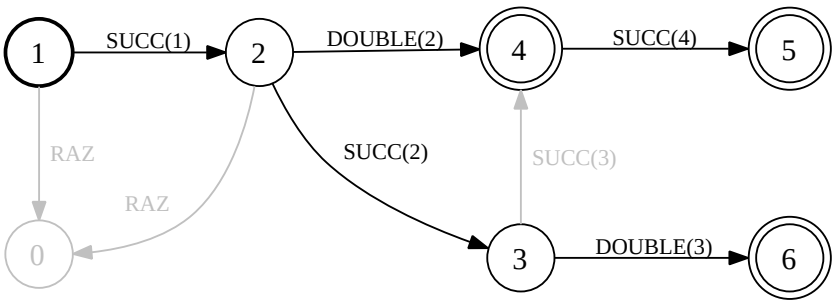


FIGURE 5.6 – Exemple de polyplan solution Π = `SUCC(1)` $\gg$ ((`DOUBLE(2)` $\gg$ (`SUCC(4)` $\square$ ϵ)) $\square$ (`SUCC(2)` $\gg$ `DOUBLE(3)`)) de largeur 3, représenté dans le système de transition d'états sur l'état 1 et dont l'application peut mener aux états 4, 5 ou 6 selon les choix effectués.

Flexibilité

La largeur d'un polyplan Π permet de définir si Π est flexible ou non : Π est flexible si $\#\Pi > 1$. Seule l'utilisation de l'opérateur d'alternative rend les polyplans flexibles. L'opérateur « $\prec$ » de comparaison de l'expressivité entre deux polyplans Π et Π' confronte les plans qui dérivent de Π et de Π'. Par exemple, le plan flexible Π' dont dérivent les plans `SUCC(1)` $\gg$ `DOUBLE(2)` et `SUCC(1)` $\gg$ `SUCC(2)` $\gg$ `DOUBLE(3)` est moins expressif que le plan flexible Π de la figure 5.6 car `SUCC(1)` $\gg$ `DOUBLE(2)` et `SUCC(1)` $\gg$ `SUCC(2)` $\gg$ `DOUBLE(3)` peuvent être réalisés par Π et Π', alors que `SUCC(1)` $\gg$ `DOUBLE(2)` $\gg$ `SUCC(4)` ne peut être réalisé que par Π : $\Pi' \prec \Pi$.

5.3 Conclusion

Après avoir défini ce qu'est la flexibilité en planification, nous avons dans ce chapitre proposé le concept de polyplan grâce aux opérateurs de séquence et d'alternative. Nous avons utilisé la notion de largeur des polyplans pour caractériser la flexibilité de la solution obtenue par un planificateur pour un problème donné. Ce cadre théorique nous sera utile pour spécifier la flexibilité que notre planificateur est capable de calculer.

Troisième partie

Traitement de la flexibilité en pratique

Chapitre 6

Modèle restreint de la planification flexible

Notre objectif est de concevoir un planificateur Λ-GraphPlan[1] (LGP), qui calcule des polyplans. La flexibilité visée par LGP doit être la plus large possible (*cf.* paragraphe 5.1.3) : idéalement, pour un problème de planification donné, tous les plans solutions devraient pouvoir être dérivés du polyplan calculé par LGP. Mais calculer tous les plans solutions revient à explorer de façon exhaustive le système de transition d'états pour s'assurer que tous les plans solutions ont été trouvés. A contrario, les planificateurs tendent habituellement à limiter l'exploration du système de transition d'états pour diminuer le temps de recherche d'une solution, notamment par l'utilisation d'heuristiques. Nous proposons un compromis entre obtenir le polyplan le plus large et minimiser l'exploration du système de transition d'états : le polyplan calculé par LGP ne couvre pas toute la flexibilité décrite dans le chapitre 5. LGP sera détaillé dans le chapitre 7. Dans ce chapitre, nous définissons la flexibilité des polyplans calculés par LGP. Nous appelons λ-plan cette forme restreinte de polyplan.

$$\mathcal{P} \rightarrow \Lambda\text{-GraphPlan} \rightarrow \lambda\text{-plan}$$

Pour définir la flexibilité traitée par LGP, nous proposons de nouveaux opérateurs de composition de polyplans, définis à partir des opérateurs de séquence et d'alternative. Ces opérateurs sont l'entrelacement, la permutation, et une forme de choix itératif, similaire à la permutation. Les opérateurs de séquence, d'entrelacement et de choix itératif sont ensuite utilisés pour définir la grammaire des λ-plans. La figure 6.1 page suivante positionne l'expressivité des solutions calculées selon les opérateurs utilisés : un plan est un cas particulier de λ-plan et un λ-plan est un cas particulier de polyplan.

6.1 Expressivité des λ-plans

Les restrictions de l'expressivité des λ-plans concerne trois points. Premièrement, les λ-plans sont déterministes : toutes les exécutions d'un λ-plan mènent au même état, seuls les chemins diffèrent. Deuxièmement, tous les plans dérivés d'un λ-plan sont composés des mêmes actions. La troisième restriction concerne les contraintes d'ordre d'exécution entre actions qui sont déterminées par les opérateurs de séquence, d'entrelacement et de choix itératif.

6.1.1 Polyplans déterministes

LGP cherche un polyplan permettant d'atteindre un unique état de l'ensemble des états satisfaisant l'objectif. Par exemple, dans le graphe de la figure 6.2 page suivante, deux chemins sont possibles pour atteindre l'état 4 à partir de l'état 1. Le premier correspond au plan $\pi_1 =$ `SUCC(1)` $\gg$ `DOUBLE(2)`, et le second au plan $\pi_2 =$ `SUCC(1)` $\gg$ `SUCC(2)` $\gg$ `SUCC(3)`.

1. Λ-GraphPlan est dérivé de GraphPlan et utilise des relations appelées λ-mutex.

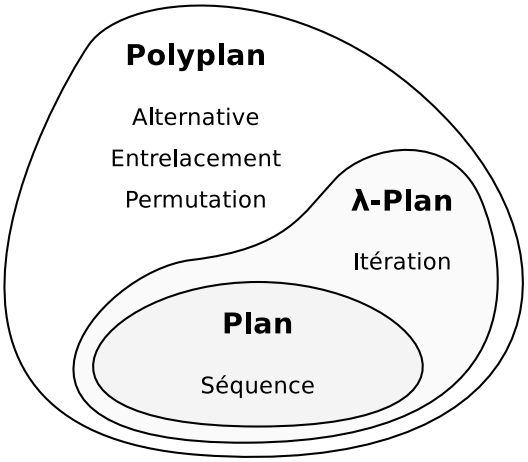


FIGURE 6.1 – Utilisation des opérateurs selon le type de planification effectuée. La séquence est l'unique opérateur utilisé pour composer les plans en planification classique (*cf.* paragraphe 4.2). Les polyplans (*cf.* paragraphe 5.2) utilisent en prime l'opérateur d'alternative, qui permet la flexibilité. Les opérateurs d'entrelacement et de permutation sont définis dans ce chapitre par les opérateurs de séquence et d'alternative. Le choix itératif (itération) s'applique spécifiquement sur des λ-plans.

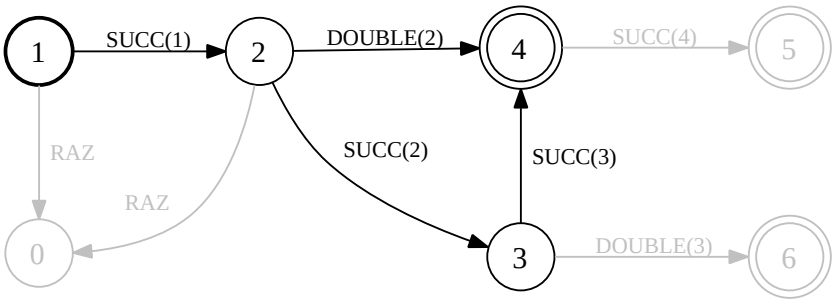


FIGURE 6.2 – Le polyplan `SUCC(1)` $\gg$ (`DOUBLE(2)` $\square$ (`SUCC(2)` $\gg$ `SUCC(3)`)) appliqué à l'état 1 permet d'atteindre uniquement l'état 4 : le polyplan est déterministe car chacune de ses exécutions conduit au même état.

Les effets de π_1 et de π_2 sont les mêmes : $\mathit{eff}(\pi_1) = \mathit{eff}(\pi_2)$. Ainsi, le polyplan $\Pi =$ `SUCC(1)` $\gg$ (`DOUBLE(2)` $\square$ (`SUCC(2)` $\gg$ `SUCC(3)`)) a une largeur de 2 mais n'a qu'un seul effet possible. Cela représente une première limitation de l'expressivité des λ-plans.

6.1.2 Ensemble des actions composées

Quels que soient les choix effectués à l'exécution d'un λ-plan, l'ensemble des actions finalement exécutées est toujours le même. Ainsi, tous les plans dérivés du λ-plan contiennent exactement les mêmes actions, mais dans des ordres différents. Plus précisément, nous nous intéressons à deux cas particuliers de l'alternative :

- dans GRIPPER, le fait de commencer à déplacer une balle plutôt qu'une autre n'influe pas sur l'accomplissement de l'objectif, tant que toutes les balles sont déplacées durant l'exécution. Un ordre particulier de traitement des balles est une des permutations permettant de résoudre GRIPPER. Autrement dit, il n'est pas justifié de proposer une permutation plutôt qu'une autre dans le λ-plan solution. De notre point de vue, le choix de la permutation doit être fait à l'exécution.
- dans GRIPPER, si plusieurs robots sont impliqués dans des plans indépendants, l'exécution de ces plans peut être entrelacée de telle façon que chaque plan évolue indépendamment des autres. La manière d'entrelacer les plans est laissée libre dans le λ-plan solution, l'ordre d'exécution est décidé par le contrôleur.

Les λ-plans sont composés d'actions dont les contraintes d'ordre sont indiquées par les opérateurs de séquence, d'entrelacement et de permutation. La grammaire des λ-plans sera détaillée au paragraphe 6.7.

6.2 Formalisation des λ-plans

Le modèle des λ-plans est représenté à la figure 6.3. Un λ-plan est un polyplan déterministe : son exécution permet d'atteindre un unique état par différents chemins. En utilisant la définition des polyplans (*cf.* définition 5.6), le déterminisme d'un λ-plans Π se note :

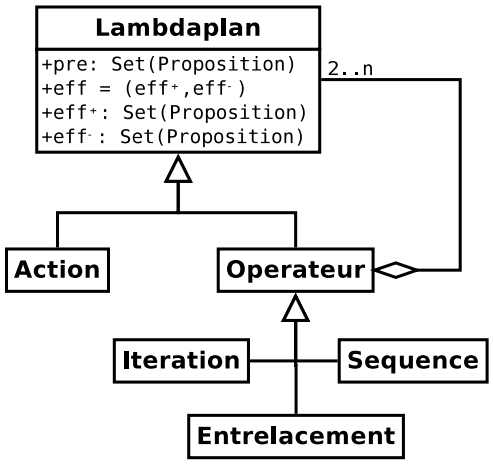


FIGURE 6.3 – Modèle de λ-plan : une action est un λ-plan, et une composition par les opérateurs de séquence, d'entrelacement et de choix itératif (itération) de λ-plans est un λ-plan.

$$\forall\,(i,j) \in [1..\#\Pi] \times [1..\#\Pi],\ \mathit{eff}_i(\Pi) = \mathit{eff}_j(\Pi)$$

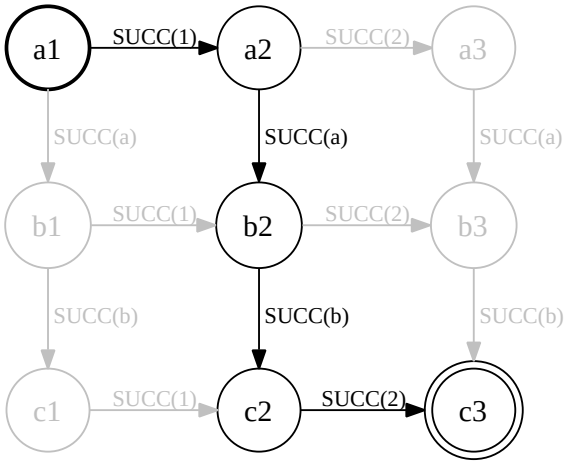


FIGURE 6.4 – Le λ-plan est composé des plans `SUCC(1)` $\gg$ `SUCC(2)` et `SUCC(a)` $\gg$ `SUCC(b)` en entrelacement, et permet d'atteindre l'état $c3$ depuis l'état $a1$: le plan `SUCC(1)` $\gg$ `SUCC(a)` $\gg$ `SUCC(b)` $\gg$ `SUCC(2)` est exécuté.

Nous nous dispensons de dupliquer les effets pour chacun des plans dérivables de Π, et nous notons *eff*(Π) le couple d'effets positifs et négatifs du λ-plan Π. L'ensemble des plans $\{\pi_1, \cdots, \pi_n\}$ dérivés de Π, de largeur n, est tel que :

$$\forall i \in [1..n], \pi_i = (nom_i(\Pi), pre(\Pi), eff(\Pi))$$

Dans la suite de ce chapitre, nous allons définir les opérateurs permettant de construire un λ-plan. Le résultat de la composition par les opérateurs d'entrelacement ou de permutation est en général un polyplan, et ces opérateurs composent des polyplans. Pour la composition de λ-plans, des formes particulières de ces deux opérateurs seront définies : l'entrelacement déterministe et le choix itératif (décrit par une structure plutôt qu'un opérateur).

6.3 Opérateur d'entrelacement

L'entrelacement entre deux polyplans permet d'exécuter chacun des polyplans en respectant l'ordre interne des actions dans chacun des polyplans mais laisse libre l'ordre d'exécution des actions de chacun des polyplans vis à vis des actions de l'autre polyplan. Par exemple le premier polyplan commence à être appliqué, puis le second est complètement appliqué pour terminer l'application du premier. Cela est représenté à la figure 6.4 qui entrelace les plans `SUCC(1)` $\gg$ `SUCC(2)` et `SUCC(a)` $\gg$ `SUCC(b)`. Pour obtenir l'expressivité de l'entrelacement, nous introduisons l'opérateur d'entrelacement entre polyplans, représenté par le symbole « $\|$ ».

6.3.1 Définition de l'entrelacement

Soient deux polyplans Π' et Π''. Nous exprimons la sémantique de l'opérateur d'entrelacement en considérant les polyplans Π' et Π'' comme deux ensembles de plans. L'entrelace-

ment entre deux polyplans correspond à l'entrelacement entre chacun de leurs plans. Nous commençons par définir l'entrelacement entre plans. Soient deux plans π_1 et π_2, tels que :

- $\pi_1 = a_1 \gg \pi'_1$ avec $\pi'_1 = a_2 \gg \cdots \gg a_k$ avec $\forall i \in \{1, \cdots, k\}, a_i \in \mathcal{A}$;
- $\pi_2 = a'_1 \gg \pi'_2$ avec $\pi'_2 = a'_2 \gg \cdots \gg a'_n$ avec $\forall i \in \{1, \cdots, k\}, a'_i \in \mathcal{A}$.

Alors l'entrelacement entre π_1 et π_2 est le polyplan $\Pi = \pi_1 \parallel \pi_2$, défini récursivement :

$$\Pi = (a_1 \gg (\pi'_1 \parallel \pi_2)) \square (a'_1 \gg (\pi_1 \parallel \pi'_2))$$

Par définition, chaque action de π_1 peut précéder à l'exécution n'importe quelle action de π_2, et vice versa. Ainsi, pour que $\pi_1 \parallel \pi_2$ soit défini, aucune action de π_1 ne doit interférer avec les actions de π_2 :

$$\forall (a_1, a_2), a_1 \in \pi_1, a_2 \in \pi_2, \neg \texttt{interference}(a_1, a_2)$$

L'entrelacement entre les polyplans Π' et Π'' a pour résultat le polyplan $\Pi = \Pi' \parallel \Pi''$. Soient $\Pi' = \{\pi'_1, \cdots, \pi'_k\}$ et $\Pi'' = \{\pi''_1, \cdots, \pi''_n\}$ alors le polyplan $\Pi = \Pi' \parallel \Pi''$ est défini par :

$$\Pi = (\pi'_1 \parallel \pi''_1) \square \cdots \square (\pi'_1 \parallel \pi''_n) \square \cdots \square (\pi'_k \parallel \pi''_1) \square \cdots \square (\pi'_k \parallel \pi''_n)$$

Le polyplan $\Pi_1 \parallel \Pi_2$ est défini si et seulement si chaque action de Π_1 interfère avec aucune action de Π_2, et vice versa.

L'opérateur d'entrelacement est associatif et commutatif (*cf.* annexe C.4.1). L'opérateur n-aire d'entrelacement entre n polyplans $\Pi_1, \cdots, \Pi_n$ est défini par l'opérateur binaire de l'entrelacement, et noté $\Pi = \prod\!\!\!\!\!\!\parallel_{i=1}^{n} (\Pi_i) = \Pi_1 \parallel \cdots \parallel \Pi_n$.

6.3.2 Expressivité

Le nombre de choix offert par l'entrelacement de deux polyplans dépend de la longueur des plans qui composent les polyplans. En effet, le nombre de plans déduit du parallélisme entre deux plans $\pi = a_1 \gg \cdots \gg a_k$ et $\pi' = a'_1 \gg \cdots \gg a'_n$, c'est-à-dire la largeur du polyplan $\Pi = \pi \parallel \pi'$, est défini comme suit (*cf.* preuve à l'annexe C.4.2) :

$$\text{avec } |\pi| = k,\ |\pi'| = n \text{ et } \Pi = \pi \parallel \pi', \text{ alors } \#\Pi = \prod_{p=1}^{n} \frac{k+p}{p} = f(k,n) = f(n,k)$$

Par exemple, l'entrelacement (`SUCC(1)` $\gg$ `SUCC(2)`) $\parallel$ (`SUCC(a)` $\gg$ `SUCC(b)`) a pour largeur :

$$f(2,2) = \Pi_{p=1}^{2} \frac{2+p}{p} = 3 * 2 = 6$$

C'est bien le nombre de chemins que l'on retrouve à la figure 6.4 page ci-contre pour cet entrelacement. La largeur du polyplan $\Pi = \Pi_1 \parallel \Pi_2$ avec $\Pi_1 = \left\{\pi^1_1, \cdots, \pi^1_k\right\}$ et $\Pi_2 = \left\{\pi^2_1, \cdots, \pi^2_n\right\}$ vaut :

$$\#\Pi = \sum_{i=1}^{k} \sum_{j=1}^{n} f(\left|\pi^1_i\right|, \left|\pi^2_j\right|)$$

En effet, la largeur de Π est définie selon la largeur de l'entrelacement de chaque plan de Π_1 avec chaque plan de Π_2. Ainsi, la largeur de Π est la somme des largeurs de la permutation de tous les couples possibles d'un plan de Π_1 et d'un plan de Π_2.

Considérons maintenant un ensemble de plans $\left\{\pi^1, \cdots, \pi^n\right\}$. Le polyplan $\Pi = \pi^1 \parallel \cdots \parallel \pi^n$ qui compose en entrelacement tous les plans a la largeur suivante :

$$\#\Pi = f(f(\cdots f(\left|\pi^1\right|, \left|\pi^2\right|), \cdots), |\pi^n|)$$

Autrement dit, la largeur de l'entrelacement des plans $\{\pi^1, \cdots, \pi^n\}$ vaut la largeur de l'entrelacement entre les deux premiers plans π^1 et π^2, lui même entrelacé avec le suivant π^3, ainsi de suite jusqu'au dernier plan π^n. Soit le polyplan $\Pi = \bigparallel_{i=1}^{n} (\Pi_i)$ avec chaque polyplan Π_i défini par un ensemble de plans $\{\pi_1^i, \cdots, \pi_k^i\}$, alors la largeur de Π est :

$$\#\Pi = \sum_{i_1=1}^{\#\Pi_1} \sum_{i_2=1}^{\#\Pi_2} \cdots \sum_{i_n=1}^{\#\Pi_n} f(f(\cdots f(|\pi_{i_1}^1|, |\pi_{i_2}^2|), \cdots), |\pi_{i_n}^n|)$$

La largeur de l'entrelacement entre chaque ensemble de plans $\{\pi_{i_1}^1, \cdots, \pi_{i_n}^n\}$ est calculé comme précédemment. Ce calcul est à sommer avec tous les ensembles possibles de plans[2], ce qui explique les n sommes.

6.3.3 Entrelacement de λ-plans

Pour que l'entrelacement entre deux λ-plans soit défini, il faut que les deux λ-plans soient indépendants. Cette propriété garantit que le but atteint sera le même quel que soit l'ordre d'exécution des actions. Prenons par exemple deux actions a_1 et a_2 qui ne sont pas indépendantes, avec $\mathit{eff}^-(a_1) \cap \mathit{eff}^+(a_2) \neq \emptyset$. Dans ce cas, les plans $\pi_1 = a_1 \gg a_2$ et $\pi_2 = a_2 \gg a_1$ n'ont pas les mêmes effets : l'entrelacement des deux actions n'est pas déterministe. L'entrelacement entre n λ-plans indépendants $\Pi_1, \cdots, \Pi_n$, est un λ-plan défini par (*cf.* preuve à l'annexe C.4.3) :

$$\begin{aligned}
\mathit{nom}(\Pi) &= \bigparallel_{i=1}^{n} (\Pi_i) \\
\mathit{pre}(\Pi) &= \bigcup_{i=1}^{n} \mathit{pre}(\Pi_i) \\
\mathit{eff}^+(\Pi) &= \bigcup_{i=1}^{n} \mathit{eff}^+(\Pi_i) \\
\mathit{eff}^-(\Pi) &= \bigcup_{i=1}^{n} \mathit{eff}^-(\Pi_i)
\end{aligned}$$

6.4 Opérateur de permutation

La permutation d'un ensemble de polyplans permet de choisir l'ordonnancement de l'exécution des polyplans. La permutation entre deux polyplans Π_1 et Π_2 permet d'exécuter ces polyplans séquentiellement mais dans un ordre laissé libre : Π_1 puis Π_2, ou bien Π_2 puis Π_1. Par exemple, dans la figure 6.5 page suivante, le plan `NEXT(num)` $\gg$ `LAST(num)` et `JUMP(alpha)` sont proposés en permutation. Nous introduisons un opérateur de permutation pour exprimer de façon concise l'ordre indéterminé entre des polyplans. Nous notons cet opérateur « $\diamond$ ».

6.4.1 Définition de la permutation

Nous définissons l'opérateur de permutation entre deux polyplans quelconques par les opérateurs de séquence et de choix. Soient deux polyplans Π_1 et Π_2, alors la permutation

2. Chaque ensemble de plans est formé d'un plan de chaque polyplan de $\{\Pi_1, \cdots, \Pi_n\}$.

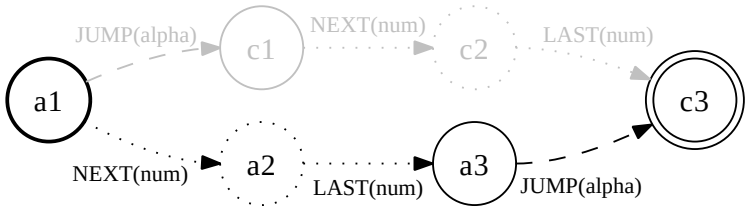


FIGURE 6.5 – Un polyplan permettant d'atteindre l'état `c3` depuis l'état `a1`, par la composition en permutation du plan `NEXT(num)` ≫ `LAST(num)` représenté en traits pointillés et de `JUMP(alpha)` représenté par les tirets : le plan `NEXT(num)` ≫ `LAST(num)` ≫ `JUMP(alpha)` est exécuté.

de Π_1 et Π_2 est le polyplan $\Pi = \Pi_1 \diamond \Pi_2$ tel que :

$$\Pi = (\Pi_1 \gg \Pi_2) \square (\Pi_2 \gg \Pi_1)$$

D'après la définition des opérateurs de séquence et de choix, le polyplan $\Pi_1 \diamond \Pi_2$ est défini si et seulement si les polyplans Π_1 et Π_2 n'interfèrent pas :

$$pre(\Pi_1) \cap eff^-(\Pi_2) = \emptyset \wedge pre(\Pi_2) \cap eff^-(\Pi_1) = \emptyset$$

Grâce à l'associativité et à la commutativité de l'opérateur de permutation (*cf.* annexe C.5.1), nous proposons l'application sur un état e d'un polyplan Π composé de la permutation de n polyplans. Cet opérateur n-aire permet la permutation de n polyplans $\Pi_1, \cdots, \Pi_n$, ce qui forme un nouveau polyplan $\Pi = \diamondsuit(\Pi_1, \cdots, \Pi_n) = \overset{n}{\underset{i=1}{\diamondsuit}}(\Pi_i)$. Cet opérateur est défini par l'opérateur binaire de permutation :

$$\overset{n}{\underset{i=1}{\diamondsuit}}(\Pi_i) = \Pi_1 \diamond \cdots \diamond \Pi_n$$

Soit le polyplan $\Pi = \overset{n}{\underset{i=1}{\diamondsuit}}(\Pi_i)$. Notons $\Pi_{/i}$ l'ensemble des polyplans $\{\Pi_1, \cdots, \Pi_n\}$ privé du polyplan Π_i, composé en permutations : $\Pi_{/i} = \Pi_1 \diamond \cdots \diamond \Pi_{i-1} \diamond \Pi_{i+1} \diamond \cdots \diamond \Pi_n$. Alors Π est défini par :

$$\diamondsuit(\Pi_1, \cdots, \Pi_n) = \left(\Pi_1 \gg \Pi_{/1}\right) \square \cdots \square \left(\Pi_n \gg \Pi_{/n}\right)$$

Informellement, $\overset{n}{\underset{i=1}{\diamondsuit}}(\Pi_i)$ se traduit par le choix successif du prochain polyplan à exécuter parmi l'ensemble proposé, jusqu'à ce que tous les polyplans aient été exécutés.

6.4.2 Expressivité

Le polyplan $\Pi_1 \diamond \Pi_2$ a pour largeur la somme des largeurs des séquences $\Pi_1 \gg \Pi_2$ et $\Pi_2 \gg \Pi_1$, et donc deux fois le produit des largeurs respectives de Π_1 et de Π_2 :

$$f(\Pi_1 \diamond \Pi_2) = 2 \times \#\Pi_1 \times \#\Pi_2$$

La largeur d'un polyplan composant en permutation n polyplans est le produit des largeurs des n polyplans, comme s'ils étaient en séquence, multiplié par le nombre de permutations possibles :

$$\# \overset{n}{\underset{i=1}{\diamondsuit}}(\Pi_i) = n! \Pi_{i=1}^{n} \#\Pi_i$$

6.4.3 Permutation de λ-plans

Une permutation est dite *déterministe* si elle compose deux λ-plans Π_1 et Π_2 tels que le même état est atteint quels que soient les choix effectués à l'exécution :

$$\forall e \in \mathcal{U}, \gamma(e, \Pi_1 \diamond \Pi_2) = \gamma(e, \Pi_1 \gg \Pi_2) = \gamma(e, \Pi_2 \gg \Pi_1)$$

Dans le cas général, la permutation entre deux λ-plans n'est pas déterministe. Or, pour que le résultat de la permutation des λ-plans soit un λ-plan, la permutation doit être déterministe. Cette condition est respectée uniquement si leurs effets positifs et négatifs sont respectivement disjoints. La permutation entre Π_1 et Π_2 est déterministe si :

$$\mathit{eff}^+(\Pi_1) \cap \mathit{eff}^-(\Pi_2) = \emptyset \wedge \mathit{eff}^+(\Pi_2) \cap \mathit{eff}^-(\Pi_1) = \emptyset$$

Cela se généralise à un ensemble $\{\Pi_1, \cdots, \Pi_n\}$ de λ-plans, et $\overset{n}{\underset{i=1}{\diamond}}(\Pi_i)$ est déterministe si :

$$\forall (i,j) \in [1..n] \times [1..n], \mathit{eff}^+(\Pi_i) \cap \mathit{eff}^-(\Pi_j) = \emptyset$$

Ce cas particulier de la permutation entre λ-plans permet de définir le λ-plan $\overset{n}{\underset{i=1}{\diamond}}(\Pi_i)$ par le tuple :

$$\begin{aligned}
\mathit{nom}(\Pi) &= \overset{n}{\underset{i=1}{\diamond}}(\Pi_i) \\
\mathit{pre}(\Pi) &= \bigcup_{i=1}^{n} \mathit{pre}(\Pi_i) \\
\mathit{eff}^+(\Pi) &= \bigcup_{i=1}^{n} \mathit{eff}^+(\Pi_i) \\
\mathit{eff}^-(\Pi) &= \bigcup_{i=1}^{n} \mathit{eff}^-(\Pi_i)
\end{aligned}$$

6.5 Structure de choix itératif

Pour ce paragraphe, nous introduisons une structure de choix itératif. Cette structure exprime le choix offert au contrôleur à l'exécution. Ces choix sont une forme particulière de permutation. Avec Gripper (présenté au paragraphe 1.2.2), le plan flexible permet de choisir l'ordonnancement d'un traitement particulier à appliquer sur l'ensemble des balles : attraper une balle, puis se déplacer d'une pièce à l'autre pour y déposer la balle et retourner dans la première pièce. Le traitement est décrit par des actions capables de manipuler indifféremment chacune des balles : une variable représente *une* balle parmi l'ensemble. Nous présentons à la figure 6.6 page ci-contre le λ-plan (`NEXT(alpha)` $\gg$ `LAST(alpha)`) $\diamond$ (`NEXT(order)` $\gg$ `LAST(order)`) $\diamond$ (`NEXT(num)` $\gg$ `LAST(num)`). Dans cet exemple, nous voulons proposer un choix sur les objets {`alpha`, `order`, `num`} sur lesquels appliquer le plan abstrait `NEXT(`*type*`)` $\gg$ `LAST(`*type*`)`, avec *type* prenant pour valeur l'objet sélectionné : `NEXT(`*type*`)` $\gg$ `LAST(`*type*`)` forme le traitement à appliquer sur l'ensemble des objets. Nous commençons par formaliser l'abstraction des actions et des λ-plans abstraits ainsi que les objets manipulés par de telles abstractions, puis présentons le choix itératif.

6.5.1 Abstraction et traitement spécifique d'objets

Dans les exemples précédents, les actions qui forment le traitement à appliquer sur des objets contient des variables qui représentent ces objets. Informellement, le traitement est

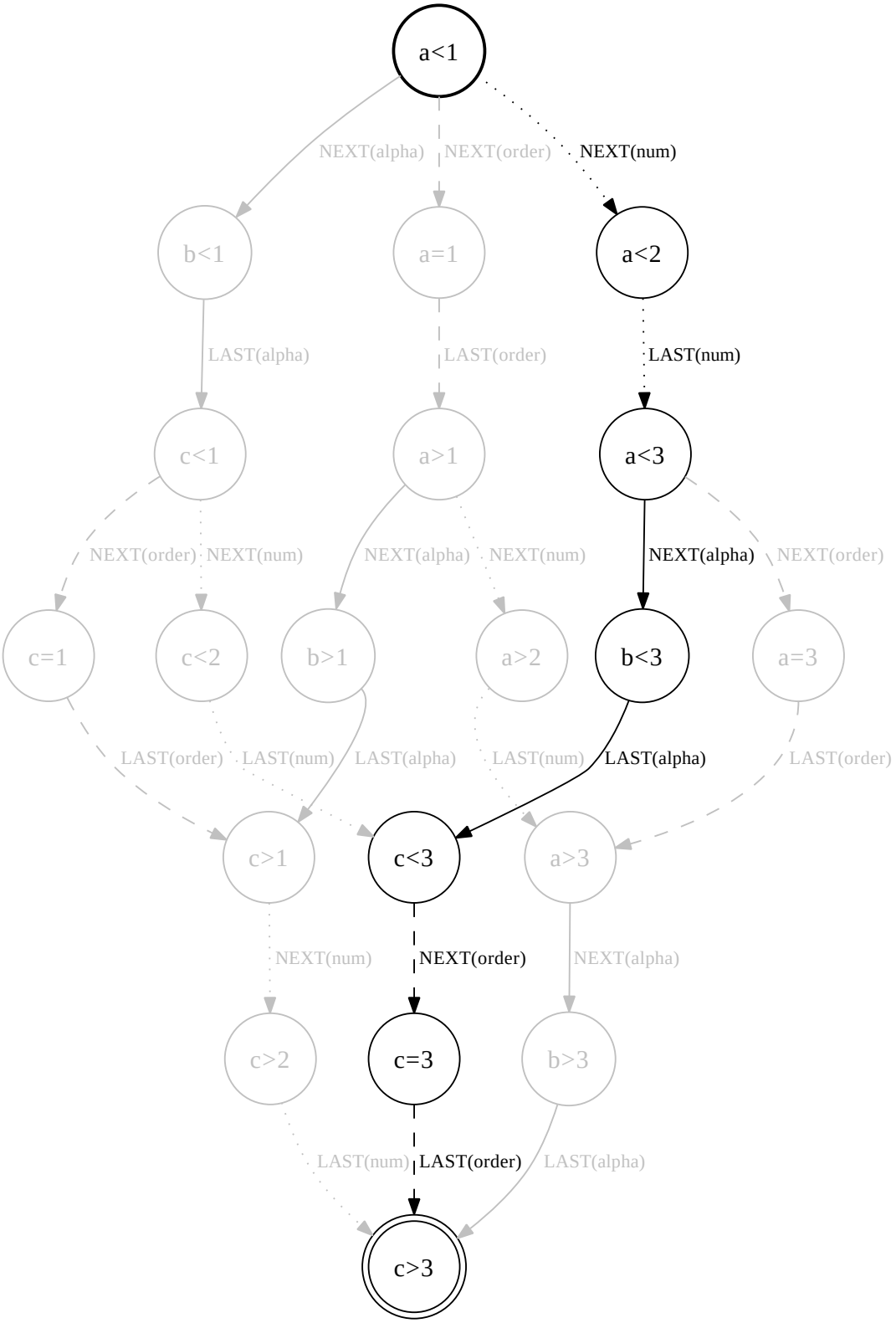


FIGURE 6.6 – Un λ-plan permettant d'atteindre l'état `c>3` depuis l'état `a<1`, qui compose en permutation le λ-plan `NEXT(alpha)` $\gg$ `LAST(alpha)` (traits pleins), le plan `NEXT(order)` $\gg$ `LAST(order)` (tirets) et le plan `NEXT(num)` $\gg$ `LAST(num)` (pointillés) : le plan `NEXT(num)` $\gg$ `LAST(num)` $\gg$ `NEXT(alpha)` $\gg$ `LAST(alpha)` $\gg$ `NEXT(order)` $\gg$ `LAST(order)` est exécuté.

un modèle des transformations à emprunter dans le système de transition d'états, et l'application sur les objets permet d'obtenir une instance de ce modèle c'est-à-dire les transitions effectives du système de transition d'états.

Action abstraite

Le résultat de l'application d'une substitution σ sur un opérateur *ope* telles que toutes les variables de $var(ope)$ ne sont pas substituées par une constante donne une action abstraite $\alpha : \alpha = \sigma(ope)$, avec σ telle que $\exists v \in var(ope)$ telle que $v \backslash \texttt{c} \notin \sigma$, avec `c` une constante. Les ensembles de propositions de $pre(\alpha)$, $\mathit{eff}^+(\alpha)$ et $\mathit{eff}^-(\alpha)$ sont dits *invariants*, et notés respectivement $\texttt{c-}pre(\alpha)$, $\texttt{c-}\mathit{eff}^+(\alpha)$ et $\texttt{c-}\mathit{eff}^-(\alpha)$: ils ne contiennent aucune variable. Nous définissons ces ensembles comme suit, avec $\mathcal{L}$ le langage propositionnel :

$$\begin{aligned}
\texttt{c-}pre(\alpha) &= \{\textsc{Pred} \mid \textsc{Pred} \in pre(\alpha) \wedge \textsc{Pred} \in \mathcal{L}\} \\
\texttt{c-}\mathit{eff}^+(\alpha) &= \{\textsc{Pred} \mid \textsc{Pred} \in \mathit{eff}^+(\alpha) \wedge \textsc{Pred} \in \mathcal{L}\} \\
\texttt{c-}\mathit{eff}^-(\alpha) &= \{\textsc{Pred} \mid \textsc{Pred} \in \mathit{eff}^-(\alpha) \wedge \textsc{Pred} \in \mathcal{L}\}
\end{aligned}$$

Par opposition, les prédicats de $pre(\alpha)$, $\mathit{eff}^+(\alpha)$ et $\mathit{eff}^-(\alpha)$ qui ne sont pas des propositions sont dits *variants*, et notés respectivement $\textit{v-}pre(\alpha)$, $\textit{v-}\mathit{eff}^+(\alpha)$ et $\textit{v-}\mathit{eff}^-(\alpha)$: ils contiennent au moins une variable de $var(\alpha)$. Par exemple, `MOVE(r-a,` *to*`)` est l'action abstraite dérivée de l'opérateur `MOVE(`$from, to$`)` par l'application de la substitution $\sigma = \{from \backslash \texttt{r-a}\}$, et `MOVE(r-a,` *to*`)` $= \sigma($`MOVE(`$from, to$`)`$)$, et :

- $\texttt{c-}pre(\alpha) = \{$`ROOM(r-a)`, `G-AT(r-a)`$\}$: pour pouvoir appliquer cette action, il faut être dans un état dans lequel le robot se situe dans l'emplacement `r-a` ;
- $\texttt{c-}\mathit{eff}^+(\alpha) = \{\}$: aucun effet positif n'est invariant ;
- $\texttt{c-}\mathit{eff}^-(\alpha) = \{$`G-AT(r-a)`$\}$: après avoir appliqué cette action, le robot ne se situe plus dans l'emplacement `r-a` ;
- $\textit{v-}pre(\alpha) = \{$`ROOM(`*to*`)`$\}$: cette action abstraite pourra être appliquée après substitution de son paramètre *to* par une constante qui représente un emplacement ;
- $\textit{v-}\mathit{eff}^+(\alpha) = \{$`G-AT(`*to*`)`$\}$: le robot se trouvera dans cet emplacement après substitution de *to* et application de l'action dérivée de cette action abstraite ;
- $\textit{v-}\mathit{eff}^-(\alpha) = \{\}$: aucun effet négatif n'est variant.

L'intérêt des actions abstraites est de pouvoir être appliquées à différents objets.

Objet traité par une action abstraite

Soit α une action abstraite, alors $var(\alpha)$ est un tuple de variables. Avec l'action $a = \sigma(\alpha)$, chaque élément de la substitution σ donne une valeur pour le tuple de variables : $var(\alpha)$ a les propriétés d'une « classe d'objets » et les valeurs fournies par la substitution est un « objet » dérivée de cette classe, une instance. Par exemple, $(box, size)$ est une classe d'objets et (`pandora`, `big`) est un objet instance de la classe, et (`magic`, `small`) un autre. Nous pouvons aussi définir des classes d'objets parentes et enfants reliées par la relation est-un entre les classes d'objets. Par exemple, les objets dérivés de la classe $(box, size)$ est-un objet dérivé de la classe (box) et $(box, size, color)$ est-un $(box, size)$. Cette relation est transitive. Nous appelons « dimension » d'une classe d'objets (resp. d'un objet) chaque variable (resp. valeur) qui constitue une classe d'objets (resp. un objet). L'action abstraite `PICK(`*ball*`, r-a)` de GRIPPER avec la substitution $\{ball \backslash \texttt{c-1}\}$ a une classe d'objet $(ball)$ en tant que tuple de variables et a pour objet (`b-1`), instance de la classe. L'action `PICK(b-1, r-a)` est obtenue en appliquant `PICK(`*ball*`, r-a)` sur l'objet (`b-1`). L'application de l'objet sur une action abstraite modifie uniquement ses préconditions et effets variants.

Λ-plan abstrait

Un λ-plan abstrait est composé d'au moins une action abstraite. Ainsi, les effets d'un tel λ-plan sont aussi composés en deux parties : la première partie regroupe les effets invariants, c'est-à-dire les propositions produites ou supprimées par le λ-plan abstrait ; la seconde partie concerne tous les autres effets dits variants, ceux qui manipulent la classe d'objets. Les opérateurs de composition pour former un λ-plan π s'applique autant aux effets invariants qu'aux effets variants.

Objet traité par un λ-plan abstrait

Nous étendons l'application d'une action sur un objet à l'application d'un λ-plan abstrait sur un objet. Soit σ une substitution, alors $obj = val(\sigma)$ est un objet de la classe $var(\sigma)$. L'application de l'objet *obj* sur le λ-plan abstrait π se note $\pi(obj)$ et correspond à l'application de l'objet *obj* sur chacune des actions qui composent π. Par exemple, l'application du λ-plan abstrait `NEXT(`*type*`)` ≫ `LAST(`*type*`)` sur l'objet `num` donne le λ-plan `NEXT(num)` ≫ `LAST(num)`. Un tel λ-plan est appelé un traitement pour l'objet concerné.

6.5.2 Définition du choix itératif

L'opérateur de permutation sur un ensemble de λ-plans est le choix sur toutes les permutations possibles de ces λ-plans. De façon à proposer le choix sur un ensemble d'objets plutôt que sur un ensemble de λ-plans instance d'un même λ-plan abstrait, nous proposons la structure de choix itératif. Soit un λ-plan abstrait π un traitement spécifique à appliquer sur un ensemble d'objets, nous souhaitons un plan qui propose itérativement le choix du prochain objet à traiter par π, jusqu'à ce que tous les objets aient été traités. Dans l'esprit de la primitive `choose` utilisée en planification généralisée dans [SIZ10], qui permet de faire un choix à l'exécution dans un ensemble d'éléments, avec $\mathscr{O} = \{(\texttt{o-1}), \cdots, (\texttt{o-n})\}$ l'ensemble des objets à traiter, on a :

```
1 O ← {(o-1) , ··· , (o-n) } ;
2 while O is not empty do
3 |   obj ← choose-in(O) ;                    ⇔ π (o-1) ◇ ··· ◇ π (o-n)
4 |   O ← O \ obj ;
5 |   π (obj)
```

Dans cette structure itérative, (obj) est la classe des objets sur lesquels appliquer le traitement π, et les instances de cette classe sont les objets $(\texttt{o-1}), \cdots, (\texttt{o-n})$. Nous proposons une écriture équivalente avec une structure de *choix itératif* comme suit, plus compacte que la précédente :

```
choose (obj) in {(o-1), ··· , (o-n)}          ⇔ π (o-1) ◇ ··· ◇ π (o-n)
    | π (obj)
```

Cette structure de choix itératif est équivalente au λ-plan $\pi\,(\texttt{o-1}) \diamond \cdots \diamond \pi\,(\texttt{o-n})$. Ainsi, l'exécution de cette structure de choix itératif selon l'ordre « (`o-1`) puis (`o-2`) puis ... puis (`o-n`) » pour le traitement des objets revient à appliquer le plan $\pi\,(\texttt{o-1}) \gg \pi\,(\texttt{o-2}) \gg \cdots \gg \pi\,(\texttt{o-n})$. L'expressivité de la structure de choix itératif est celle de la permutation entre les λ-plans issus de l'application du λ-plan abstrait sur chaque objet. Par exemple, avec GRIPPER, considérons le λ-plan abstrait π suivant : `PICK(`*ball*`, r-a)` ≫ `MOVE(r-a, r-z)` ≫ `DROP(`*ball*`,`

r-z) ≫ MOVE(r-z, r-a). Pour l'appliquer sur l'ensemble des balles {(b-1), (b-2), (b-3)}, nous écrivons :

> choose (*ball*) in {(b-1), (b-2), (b-3)}
> | PICK(*ball*, r-a) ≫ MOVE(r-a, r-z) ≫ DROP(*ball*, r-z) ≫ MOVE(r-z, r-a)

Selon le choix à l'exécution de la prochaine « *ball* » à appliquer sur π, l'une des trois balles est déplacée de la pièce r-a à r-z, le robot étant initialement et finalement dans la pièce r-a avec sa pince vide.

Décomposition du traitement

Une propriété de la structure de choix itératif proposée est que les polyplans π (o-1), ..., π (o-n) doivent être applicables les uns après les autres : ils se suivent directement, par définition de l'opérateur de permutation. Cela est possible grâce à certaines actions présentes dans les polyplans dont le rôle est uniquement de permettre les applications successives, sans modifier les propriétés des objets manipulés. Par exemple, l'action MOVE(r-z, r-a) du λ-plan pour GRIPPER ne modifie pas les propriétés de la balle manipulée. Cette action est utile uniquement pour l'enchaînement des traitements. Ainsi, après avoir attrapé une balle dans la pièce r-a, s'être déplacé dans la pièce r-z pour y déposer la balle, le robot doit retourner dans la pièce initiale pour pouvoir traiter la balle suivante. Nous différencions ces actions dans le λ-plan, et divisons donc la structure de choix itératif en deux parties :

- la première est un λ-plan abstrait appelé *treat*(*obj*) qui permet de *traiter* les objets,
- la seconde est un λ-plan appelé *reiterate* qui permet de *réitérer*, autrement dit prépare le traitement du prochain objet.

Ainsi, nous avons π (*obj*) = *treat*(*obj*) ≫ *reiterate*, et la structure de choix itératif se réécrit ainsi :

> choose (*obj*) in {(o-1), ⋯, (o-n)}
> | *treat*(*obj*) ≫ *reiterate*

Pour GRIPPER, *treat*(*obj*) correspond au λ-plan abstrait PICK(*ball*, r-a) ≫ MOVE(r-a, r-z) ≫ DROP(*ball*, r-z), tandis que *reiterate* est constitué de l'unique action MOVE(r-z, r-a).

Contraintes nécessaires entre *treat*(*obj*) et *reiterate*

Pour que les λ-plans π_i = (*treat* (o-*i*) ≫ *reiterate*) puissent être permutés, des contraintes entre *treat*(*obj*) et *reiterate* doivent être respectées (*cf.* preuves à l'annexe C.5.3). Notamment, *reiterate* ne doit modifier aucune propriété liée aux objets manipulés par le λ-plan *treat*(*obj*), donc pour tous les objets (o-*i*), aucun effet de *reiterate* n'est dans les effets variants appliqués sur (o-*i*), c'est-à-dire $\forall \sigma_i = (obj \setminus (\text{o-}i))$:

$$\left(\mathit{eff}^+(\mathit{reiterate}) \cup \mathit{eff}^-(\mathit{reiterate})\right) \bigcap \sigma_i \left(\text{v-}\mathit{eff}^+(\mathit{treat}(obj)) \cup \text{v-}\mathit{eff}^-(\mathit{treat}(obj))\right) = \emptyset$$

De plus, *reiterate* ne doit pas annuler les préconditions invariantes de *treat*(*obj*), ni ses propres préconditions qui ne sont pas soutenues par *treat*(*obj*) :

$$\mathit{eff}^-(\mathit{reiterate}) \cap \text{c-}\mathit{pre}\,(\mathit{treat}(obj)) = \emptyset$$

$$\mathit{eff}^-(\mathit{reiterate}) \cap (\mathit{pre}\,(\mathit{reiterate}) \setminus \text{c-}\mathit{eff}^+(\mathit{treat}(obj))) = \emptyset$$

Finalement, pour garantir que le traitement pourra être appliqué successivement sur chaque objet, les préconditions invariantes de $treat(obj)$ qui sont supprimées lors de son application sur un des objets doivent être supportées par *reiterate* :

$$\texttt{c-}pre\,(treat(obj)) \cap \texttt{c-}\mathit{eff}^{-}(treat(obj)) \subseteq \mathit{eff}^{+}(reiterate)$$

Ces contraintes sont utilisées pour déduire la sémantique de la structure de choix itératif.

Traitement du dernier objet

Dans la structure de choix itératif proposée, la dernière préparation à l'itération est inutile : lorsque le dernier objet est traité il n'y a aucune raison d'appliquer le λ-plan *reiterate*. Nous utiliserons donc la structure de choix itératif pour Π :

choose (obj) in $\{(\texttt{o-1}), \cdots, (\texttt{o-n})\}$
| $treat(obj)$
| if-iteration *reiterate*

Dans Gripper, lorsque il ne reste qu'une seule balle `b-i` à déplacer de `r-a` à `r-z`, l'application du traitement seul suffit à atteindre l'objectif : `PICK(b-i, r-a)` $\gg$ `PICK(r-a, r-z)` $\gg$ `DROP(b-i, r-z)`.

Nous augmentons plus que nécessaire [3] les contraintes sur les effets de *reiterate* de façon à garantir que les effets de la structure itérative ne concerne que des effets issus de $treat(obj)$. En effet, dans une structure de choix itératif, $treat(obj)$ est le λ-plan qui nous intéresse : *reiterate* doit uniquement permettre les itérations successives. Aussi, nous imposons les contraintes suivantes, conforment aux contraintes nécessaires (*cf.* preuve C.5.3) présentées ci-avant :

$$\mathit{eff}^{+}(reiterate) \subseteq \texttt{c-}\mathit{eff}^{-}(treat(obj))$$
$$\mathit{eff}^{-}(reiterate) \subseteq \texttt{c-}\mathit{eff}^{+}(treat(obj))$$

Ainsi, les effets positifs (resp. négatif) de *reiterate* ne peuvent être composé que d'effets négatifs (resp. positif) de $treat(obj)$. Par exemple, l'action `MOVE(r-z, r-a)` de Gripper a pour effet positif le fait de déplacer le robot dans la pièce `r-a`. Or cet effet est l'un des effets négatifs du traitement utilisé dans Gripper, puisque le traitement de chaque balle implique un déplacement de la pièce `r-a` à la pièce `r-z` : le robot qui se trouve initialement dans la pièce `r-a` se déplace dans la pièce `r-z` de façon à relâcher la balle précédemment attrapée. Il en est de même pour les effets négatifs de l'action `MOVE(r-z, r-a)` : ce sont des effets positifs du traitement appliqué sur les balles.

3. Les contraintes *nécessaires* permettent de garantir que les itérations sont *possibles*, c'est-à-dire que les préconditions des λ-plans sont vérifiées dans les états successifs.

Sémantique

Du fait du traitement particulier du dernier objet, il faut redéfinir [4] les préconditions et effets d'une telle structure, toujours déterministe. Le λ-plan Π est défini par :

$$pre(\Pi) = \bigcup_{i=1}^{n} pre(treat(\texttt{o-}i)) \cup \big(pre(reiterate) \setminus \texttt{c-}\mathit{eff}^{+}(treat(obj))\big)$$
$$\mathit{eff}^{+}(\Pi) = \bigcup_{i=1}^{n} \mathit{eff}^{+}(treat(\texttt{o-}i))$$
$$\mathit{eff}^{-}(\Pi) = \bigcup_{i=1}^{n} \mathit{eff}^{-}(treat(\texttt{o-}i))$$

Les effets positifs et négatifs de *reiterate* ne sont finalement pas pris en compte dans l'effet d'une structure de choix itératif puisque *reiterate* n'est pas appliqué à la dernière itération. Seules les préconditions qui ne sont pas des effets positifs de $treat(obj)$ doivent être prises en compte pour que la structure puisse être correctement exécutée.

6.6 Comparaison de l'expressivité des opérateurs

Soit un ensemble $\{\pi_1, \cdots, \pi_n\}$ de λ-plans, la comparaison de l'expressivité [5] de leur séquence, leur permutation et leur entrelacement donne :

$$\gg\ (\pi_1, \cdots, \pi_n) \prec \diamond\ (\pi_1, \cdots, \pi_n) \prec \Big\|\ (\pi_1, \cdots, \pi_n)$$

Nous pouvons le montrer en calculant la largeur de la séquence des λ-plans, de la permutation des λ-plans et de l'entrelacement des λ-plans. Mais pour le comprendre, il suffit de se rendre compte que la séquence est une exécution particulière de la permutation, et que la permutation est une exécution particulière de l'entrelacement. Autrement dit, l'ensemble des chemins formés par la séquence des λ-plans est inclus dans celui des chemins formés par leur permutation, lui même inclus dans celui des chemins formés par leur entrelacement.

6.7 Grammaire de λ-plan

Une « couche » est un λ-plan qui compose en entrelacement un ensemble d'actions indépendantes et des structures de choix itératif. La structure principale d'un λ-plan est une séquence de couches. Nous définissons la grammaire correspondant aux λ-plans :

λ**-plan** $:=$ λ**-plan** $\gg$ (**layer**)
λ**-plan** $:=$ ϵ
layer $:=$ **layer** $\|$ a avec $a \in \mathcal{A}$
layer $:=$ **layer** $\|$ **iter**
layer $:=$ ϵ
iter $:=$ `choose` obj `in` $\{(\texttt{o-1}), \cdots, (\texttt{o-n})\}$ $((\lambda\textbf{-plan}(obj))$ `if-iteration` $(\lambda\textbf{-plan}))$

4. Avec $reiterate = \epsilon$, la structure de choix itératif a la sémantique de la permutation.
5. Pour rappel, la définition 5.2 spécifie l'opérateur de comparaison de l'expressivité entre polyplans, noté $\prec$: les plans dérivables du polyplan en opérande gauche le sont aussi de celui en opérande droit.

Le λ-plan de la figure 6.4 page 78 s'écrit :

choose (*type*) in {`alpha`, `order`, `num`}
| `NEXT`(*type*) $\gg$ `LAST`(*type*)
| if-iteration ϵ

Ainsi, les actions `NEXT` et `LAST` sont appliquées séquentiellement sur chacun des objets {`alpha`, `order`, `num`}. Le λ-plan solution de GRIPPER s'écrit :

choose (*ball*) in {(`b-1`), (`b-2`), (`b-3`)}
| `PICK`(*ball*, `r-a`)
| $\gg$ `MOVE`(`r-a`, `r-z`)
| $\gg$ `DROP`(*ball*, `r-z`)
| if-iteration `MOVE`(`r-z`, `r-a`)

Il signifie que pour chaque balle dans {(`b-1`), (`b-2`), (`b-3`)}, il faut l'attraper depuis l'emplacement `r-a` (`PICK`(*ball*, `r-a`)) puis se déplacer dans `r-z` (`MOVE`(`r-a`, `r-z`)) pour finalement y déposer la balle (`DROP`(*ball*, `r-z`)). Si toutes les balles n'ont pas été déplacées, il faut retourner dans l'emplacement `r-a` après le traitement de chaque balle (`MOVE`(`r-z`, `r-a`)). La grammaire permet aussi d'entrelacer et d'imbriquer des structures de choix itératif.

Au chapitre 7, nous décrivons notre planificateur Λ-`GraphPlan` qui à partir d'un problème de planification en `PDDL` produit un λ-plan suivant cette grammaire. Les cas d'études illustreront l'expressivité des λ-plans calculés par Λ-`GraphPlan`.

Chapitre 7

Planificateur Λ-GraphPlan

Ce chapitre présente en détail le planificateur flexible que nous proposons, Λ-GraphPlan, qui, pour un problème de planification $\mathcal{P}$ donné, calcule un plan flexible Π (pour rappel, le problème $\mathcal{P}$ est décrit en 1.1 page 5, et le λ-plan Π en 6.7 page 88) :

$$\mathcal{P} \rightarrow \Lambda\text{-GraphPlan} \rightarrow \lambda\text{-plan}$$

Notre algorithme Λ-GraphPlan (LGP) est une extension[1] de l'approche de planification fondée sur les graphes, présentée au paragraphe 3.1.2 à travers GraphPlan (GP) [BF97]. La structure manipulée par LGP est donc un graphe de planification $\mathcal{G}$, tout comme dans GP. À travers les relations mutex, $\mathcal{G}$ révèle la possibilité d'entrelacer ou non l'exécution d'actions. En revanche, dans GP, $\mathcal{G}$ ne dispose d'aucune information concernant les actions qui constituent des traitements spécifiques (le λ-plan abstrait *treat* présenté en 6.5.1). Ces actions font partie des structures de choix itératif des λ-plans, il est donc nécessaire de les distinguer dans $\mathcal{G}$. LGP cherche à construire un λ-plan solution par la découverte de patterns d'actions dans $\mathcal{G}$. Ces patterns permettent la construction des structures de choix itératif et la découverte des objets manipulés par ces structures.

7.1 Principes illustrés sur un cas d'étude

Dans les structures de choix itératif (*cf.* paragraphe 6.5.2), il est équivalent de sélectionner un objet plutôt qu'un autre. De plus, le traitement à appliquer sur les objets est le même pour tous les objets. Cela signifie que dans un graphe de planification, les actions qui manipulent ces objets apparaissent sur la même couche. Dans le domaine Gripper, c'est le cas par exemple pour toutes les actions permettant d'attraper une balle, PICK. En revanche, il n'est pas possible de prendre toutes les balles en même temps. En substance, les structures de choix itératif sont des représentations compactes pour les manipulations successives des objets, par exemple « une balle après l'autre ». Elles n'ont d'intérêt que s'il n'est pas possible de manipuler simultanément les objets : l'entrelacement qui traduit la simultanéité est plus flexible que les structures de choix itératif (*cf.* chapitre 6). Cela signifie que les manipulations recherchées dans $\mathcal{G}$ sont mutuellement exclusives. La séquence d'actions qui consiste à attraper une des trois balles dans une pièce pour la déplacer dans l'autre forme le traitement spécifique recherché. Avec les balles b-1, b-2 et b-3, nous avons trois séquences d'actions qui forment le traitement spécifique :

- PICK(b-1, r-a) ≫ MOVE(r-a, r-z) ≫ DROP(b-1, r-z) est le traitement de la balle b-1 ;
- PICK(b-2, r-a) ≫ MOVE(r-a, r-z) ≫ DROP(b-2, r-z) celui de la balle b-2 ;
- PICK(b-3, r-a) ≫ MOVE(r-a, r-z) ≫ DROP(b-3, r-z) celui de la balle b-3.

Ainsi, dans les différentes couches de $\mathcal{G}$, les séquences d'exploitation des différents objets se produisent simultanément, mais sont mutuellement exclusives. Cela est visible dans la figure 7.1 page suivante. En effet, les actions PICK de $\mathcal{A}_1$ sont mutex par interférence du fait

1. Cette extension n'a aucun lien avec *Flexible GraphPlan* [MJS00], dont l'objet de recherche est d'utiliser des *propositions flexibles* (en utilisant la logique floue) contre le caractère usuellement binaire de la valeur de vérité.

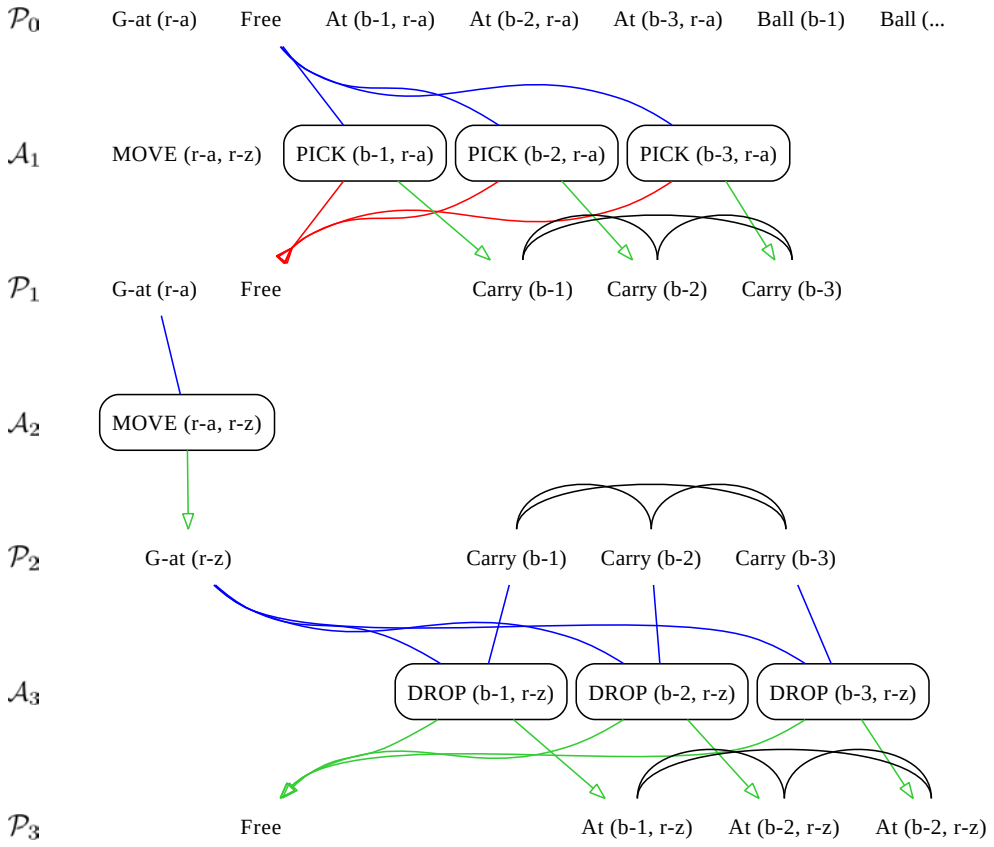


FIGURE 7.1 – Les actions entourées sont les actions présentes dans le λ-plan Π attendu : les actions qui manipulent les objets `b-1`, `b-2` et `b-3` dans Π sont mutex dans $\mathcal{G}$. La signification des liens est la suivante : un trait entre une proposition et une action signifie que la proposition est une précondition de l'action ; une flèche « à l'endroit » entre une action et une précondition signifie que la proposition est un effet positif de l'action ; une flèche « à l'envers » entre une action et une précondition signifie que la proposition est un effet négatif de l'action ; un lien entre deux propositions représente une relation mutex. Ces représentations seront utilisés pour toutes les figures suivantes.

de leur effet négatif FREE, et les actions DROP qui nous intéressent dans $\mathcal{A}_3$ sont mutex par propagation des relations mutex issues des actions PICK précédentes.

LGP progresse par étapes successives d'extension de $\mathcal{G}$ et d'extraction de plan tout comme l'algorithme GP, mais distingue par des étiquettes dans $\mathcal{G}$ toutes les actions qui forment les manipulations des objets. Le marquage, noté λ et source du nom Λ-GraphPlan, est effectué durant l'étape d'extension du graphe. Il est ensuite utilisé durant le processus d'extraction pour relâcher les contraintes dues aux relations mutex : les relations mutex entre actions étiquetées sont ignorées durant l'extraction. Le résultat de l'extraction est un plan en couches[2], noté π, « relâché » de telle façon que les actions de chaque couche sont ou bien sans relation mutex ou sont marquées λ. L'objectif de cette relaxation des relations mutex est d'obtenir un plan dans lequel les manipulations successives des objets soient représentées sur la même couche. Pour GRIPPER avec trois balles b-1, b-2 et b-3 nous obtenons le plan relâché suivant :

```
(PICK(b-1, r-a) ‖ PICK(b-2, r-a) ‖ PICK(b-3, r-a))
                ≫ MOVE(r-a, r-z) ≫
(DROP(b-1, r-z) ‖ DROP(b-2, r-z) ‖ DROP(b-3, r-z))
```

Le plan π extrait n'est pas directement applicable du fait du relâchement de contraintes, et nécessite une transformation. Par définition de l'action PICK (*cf.* annexe B.3), le robot n'est pas capable d'attraper plus d'une balle en même temps. De ce fait, le plan contient des incohérences puisque il propose en entrelacement d'attraper les différentes balles. Le plan doit donc subir des transformations de façon à insérer une structure de choix itératif qui permette le traitement de chaque balle, l'une après l'autre. Ainsi, lorsque un plan π relâché vis à vis des relations mutex est extrait, LGP lui applique successivement des transformations : LGP découvre les objets manipulés, construit les structures de choix itératif, pour finalement obtenir un λ-plan Π. Si la transformation réussit, le λ-plan Π est solution du problème de planification. Le λ-plan correspondant au problème GRIPPER pour trois balles b-1, b-2 et b-3 est le suivant :

```
choose (ball) in {(b-1), (b-2), (b-3)}
      | (PICK(ball, r-a)
      |     ≫ MOVE(r-a, r-z)
      |     ≫ DROP(ball, r-z))
      | if-iteration (MOVE(r-z, r-a))
```

La transformation du λ-plan implique la recherche des actions mutex dans le plan relâché qui initient la structure de choix itératif. Nous faisons l'hypothèse que ce sont les actions dérivées d'un même opérateur et en interférences qui initient de telle structure. Une fois ces actions découvertes, dans l'exemple, les trois actions dérivées de l'opérateur PICK, LGP construit une action abstraite pour les représenter, ce qui révèle les objets manipulés par la structure itérative. Il faut ensuite intégrer d'une part les actions qui manipulent ces objets, dans l'exemple les actions DROP, et intégrer d'autre part celles qui permettent de « fermer de l'intérieur » la structure de choix itératif, ce que réalise l'action MOVE(r-a, r-z) sans laquelle la structure ne pourrait pas être exécutée. Enfin, LGP construit une « fermeture externe »

2. Les plans en couches sont les plans extraits par GP, définis au paragraphe 3.1.2 page 31.

de la structure de choix itératif, avec l'action `MOVE(r-z, r-a)` qui permet l'enchaînement des traitements pour chaque balle.

Si la transformation échoue, le processus d'extension et d'extraction est repris. La décomposition fonctionnelle de `LGP` est illustrée à la figure 7.2. Trois principaux éléments sont

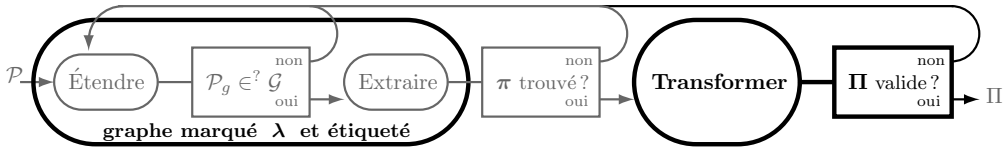


FIGURE 7.2 – Décomposition fonctionnelle de l'algorithme Λ-GraphPlan : les extensions à GP sont en foncé. Le processus d'extension et d'extraction est similaire à GP avec le relâchement de certaines contraintes par le marquage de relations mutex et l'étiquetage d'actions et de propositions. Le marquage permet d'obtenir un plan relâché π. La transformation de π a pour principal objectif de construire des structures de choix itératif qui intègrent les actions étiquetées.

manipulés successivement par `LGP` : le graphe de planification $\mathcal{G}$, le plan relâché π et le λ-plan Π. Le flux de données dans l'algorithme `LGP` est donc $\mathcal{G} \rightarrow \pi \rightarrow \Pi$. Nous présenterons l'algorithme en respectant l'ordre de ce flux.

Les principes de résolution adoptés dans `LGP` ainsi que les principaux concepts manipulés (permettant notamment de marquer les actions utiles et de transformer le plan relâché) sont introduits informellement et illustrés par le graphe de planification issu du problème GRIPPER (paragraphes 7.1.1, 7.1.2 et 7.1.3). Nous illustrons les concepts au paragraphe 7.1.4, en exploitant les cas d'études décrits au chapitre 1. Nous décrivons formellement `LGP` dans les paragraphes 7.2, 7.3 et 7.4 correspondant aux trois structures manipulées par `LGP` (le graphe de planification, le plan relâché et le λ-plan). Le paragraphe 7.6 discute nos propositions et conclut ce chapitre.

7.1.1 Exploitation du graphe de planification

`LGP` doit pouvoir repérer dans le graphe de planification les actions qui présentent un intérêt pour la transformation en λ-plan, c'est-à-dire celles qui pourraient être intégrées à une structure de choix itératif. Nous observons dans le graphe que ces actions sont mutex entre elles. Nous proposons de ne pas en tenir compte lors de l'extraction (c'est-à-dire de faire comme si elles n'étaient pas mutex). Pour cela, `LGP` opère un marquage de ces relations mutex relâchées.

Sources primaires et secondaires de relations λ-mutex

Dans cette partie, nous nous intéressons plus spécifiquement à la source des relations mutex qui nécessitent d'être marquées pour permettre l'extraction d'un plan relâché. Les actions en relation mutex pour le problème GRIPPER sont présentées à la figure 7.3 page ci-contre dans $\mathcal{G}$. Nous observons dans ce graphe que les trois actions `PICK` dans $\mathcal{A}_1$ sont mutex entre elles. C'est par interférence [3] que les actions `PICK(b-1, r-a)`, `PICK(b-2, r-a)` et `PICK(b-`

3. Nous observons dans la figure 7.1 que ces relations mutex sont dues à la proposition FREE, précondition et effet négatif de ces actions.

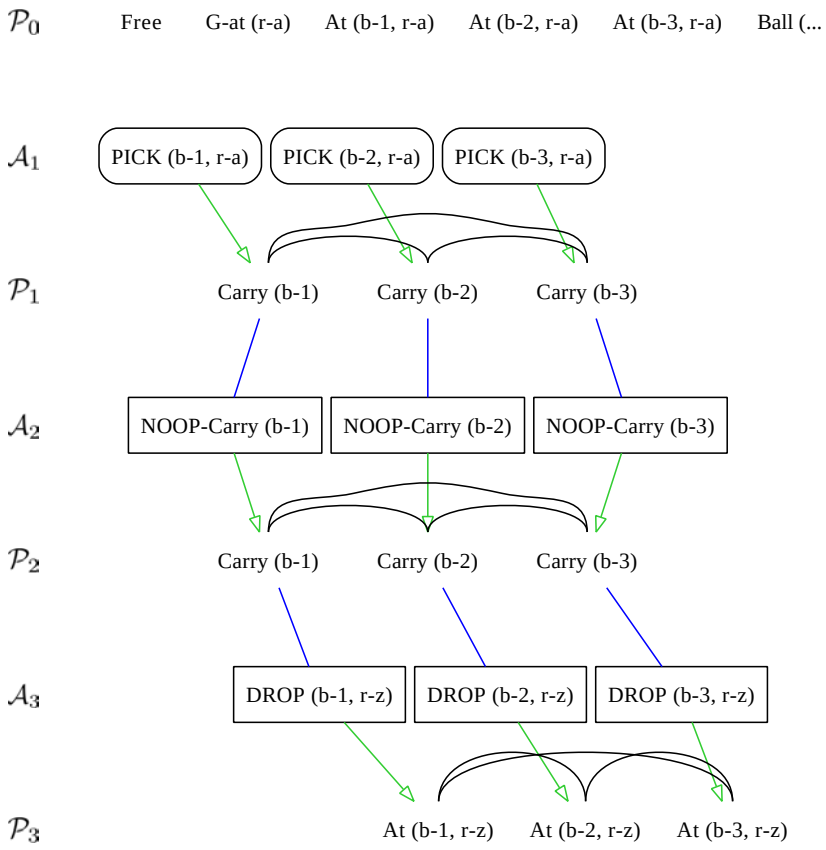


FIGURE 7.3 – Seules les actions sources des relations mutex à marquer sont représentées dans ce graphe issu de GRIPPER : les primaires sont entourées, les secondaires encadrées.

`3, r-a)` sont mutex. En revanche, les actions virtuelles [4] `NOOP-CARRY(b-1)`, `NOOP-CARRY(b-2)` et `NOOP-CARRY(b-3)` de $\mathcal{A}_2$ et les actions `DROP(b-1, r-z)`, `DROP(b-2, r-z)` et `DROP(b-3, r-z)` de $\mathcal{A}_3$ sont mutex à cause de leurs préconditions `CARRY(b-1)`, `CARRY(b-2)` et `CARRY(b-3)`, mutex dans $\mathcal{P}_1$ et $\mathcal{P}_2$. C'est donc par propagation des relations mutex que les trois actions virtuelles `NOOP-CARRY` et les trois actions `DROP` sont mutex.

Nous savons que dans une couche $\mathcal{A}_i$ donnée du graphe de planification, toutes les actions dérivées de `PICK(`*ball*, *room*`)` seront mutex entre elles par interférence. Cette action `PICK(`*ball*, *room*`)` est appelée la *classe* des relations mutex. Ainsi, les actions issues d'une certaine classe, comme les actions `PICK` de $\mathcal{A}_1$, sont a priori candidates pour la construction de λ-plan. Nous disons de ces actions qu'elles sont *sources primaires* du marquage λ des relations mutex, et provoquent le marquage λ des relations mutex entre leurs effets. Le marquage λ des relations mutex entre les actions `NOOP-CARRY` et `DROP` s'explique par le marquage de leurs préconditions : nous appelons ces actions des *sources secondaires* de relations mutex. Elles provoquent aussi le marquage λ des relations mutex entre leurs effets.

4. Les actions virtuelles ou « noop-actions » sont les actions qui ne servent qu'à propager les propositions de couche en couche : lorsqu'une proposition peut être atteinte au pas de temps i, elle peut l'être au temps $i + 1$.

Le marquage concerne un sous-ensemble des relations mutex entre les actions et les propositions. Les relations mutex marquées sont appelées λ-mutex. Elles lient les actions susceptibles d'être intégrées à des structures de choix itératif, et les propositions mutex issues d'actions λ-mutex. Les actions sources primaires du marquage sont λ-mutex *primaires* (elles initient le marquage), et les actions sources secondaires du marquage sont λ-mutex *secondaire* (elles subissent le marquage). Ainsi, les trois actions PICK de $\mathcal{A}_1$ sont λ-mutex primaires dans le graphe. Les propositions CARRY qu'elles produisent sont λ-mutex dans $\mathcal{P}_1$.

Causalité matérialisé par l'étiquetage

En reprenant le graphe présenté à la figure 7.3 page précédente, un lien de causalité entre les relations λ-mutex des actions PICK de $\mathcal{A}_1$ et DROP de $\mathcal{A}_3$ est observable à travers les trois propositions CARRY de $\mathcal{P}_1$. En effet, ces propositions sont λ-mutex dans cette couche à cause des actions PICK dont elles sont les effets positifs, et les actions DROP les ont pour préconditions. Pour matérialiser cette causalité, nous associons une étiquette aux actions et propositions λ-mutex. Cette étiquette nous permet de déterminer les actions responsables de chaque relation λ-mutex dans $\mathcal{G}$. Comme ce sont les actions λ-mutex primaires qui entrainent le besoin de structure de choix itératif, telles les PICK de $\mathcal{A}_1$, ce sont elles qui initient l'étiquetage du graphe. À chaque action impliquée dans une relation λ-mutex primaire est associée une étiquette qui permet de savoir de quelle relation λ-mutex primaire il s'agit. Cette étiquette est propagée dans $\mathcal{G}$ à chaque élément λ-mutex du fait de l'action étiquetée. Ainsi, en observant l'étiquette de la proposition AT(b-1, r-z) ou celle de l'action DROP(b-1, r-z), nous pouvons directement déterminer que c'est l'action PICK(b-1, r-a) qui est responsable des relations λ-mutex qui les lient aux autres propositions AT et respectivement aux autres actions DROP. La causalité est donc matérialisée par les étiquettes.

De plus, l'étiquetage des actions est fait de telle façon que les étiquettes sont liées à la classe de la relation λ-mutex primaire. Cela nous permet de calculer des ensembles d'actions λ-mutex en comparant l'étiquette des actions. Si les étiquettes des actions sont d'une même classe, alors ces actions forment une clique selon la relation λ-mutex. Par exemple, les actions PICK(b-1, r-a), PICK(b-2, r-a) et PICK(b-3, r-a) sont toutes les trois étiquetées. Leurs étiquettes sont liées à la classe des relations λ-mutex primaires entre les trois actions. Cette classe est l'action PICK(*ball*, *room*). Grâce à la propagation de l'étiquetage, nous obtenons dans $\mathcal{G}$ l'étiquetage des trois actions DROP comme les actions PICK. Ainsi, par l'observation des étiquettes des actions DROP, LGP déduit d'une part que ces trois actions sont toutes λ-mutex entre elles, et d'autre part que ces relations λ-mutex proviennent des actions PICK.

7.1.2 Extraction d'un plan relâché et support interne

Le marquage λ des relations mutex et l'étiquetage des actions et des propositions dans $\mathcal{G}$ est nécessaire à l'extraction du plan relâché π.

Relaxation des relations λ-mutex

Dans GP, les relations mutex sont utilisées pour empêcher que deux actions non exécutables au même pas de temps soient présentes dans la même couche du plan extrait. Dans LGP, les relations mutex marquées λ ne sont pas prises en compte durant l'extraction du plan relâché : ces relations mutex sont relaxées, et le graphe est moins contraint. En considérant que les éléments du graphe en relation λ-mutex ne sont pas mutex, des actions ne pouvant être exécutées simultanément se retrouvent tout de même dans une même couche de π. Pour cela, les propositions λ-mutex sont sélectionnées dans $\mathcal{G}$ durant l'extraction, même si par définition elles ne peuvent pas être vérifiées au même pas de temps. C'est le cas

des propositions AT(b-1, r-z), AT(b-2, r-z) et AT(b-3, r-z) de la couche $\mathcal{P}_3$ de GRIPPER, l'objectif de planification. Cette relaxation des relations mutex marquées λ suffit à extraire π.

Durant l'extraction, d'autres informations peuvent être récupérées pour faciliter la transformation de π en λ-plan.

Étiquetage du support interne

Nous nous sommes intéressés pour l'instant aux actions du plan relâché qui manipulent les balles : PICK(*ball*, r-a) et DROP(*ball*, r-z). Or dans la structure de choix itératif, l'action MOVE(r-a, r-z) fait aussi partie de $\mathcal{G}$. Nous analysons dans la figure 7.4 page suivante la raison pour laquelle cette action doit être intégrée à la structure de choix itératif. Nous observons que cette action produit la proposition G-AT(r-z) dans $\mathcal{P}_2$, qui est une précondition partagées par les actions DROP de $\mathcal{A}_3$. Donc MOVE(r-a, r-z) soutient les préconditions propositionnelles (notées c-eff^+) de DROP(*ball*, r-z), l'abstraction dans le λ-plan des actions DROP de $\mathcal{A}_3$. Sans cette action MOVE, les actions λ-mutex secondaires DROP ne pourraient pas être appliquées. C'est pour cette raison que nous appelons *support interne* les actions qui permettent le bon déroulement du traitement spécifique sur les balles. Les préconditions du support interne doivent être soit soutenues par d'autres actions au sein du traitement, soit présentes dans l'état sur lequel le traitement est appliqué. C'est ce dernier cas qui est rencontré ici : les préconditions ROOM(r-a), ROOM(r-z) et G-AT(r-a) de MOVE(r-a, r-z) font partie de $\mathcal{P}_0$, sur lequel le λ-plan est appliqué. C'est pour cette raison que l'action peut être extraite par LGP. Ainsi, toutes les actions qui sont support interne d'un traitement spécifique sont étiquetées durant son extraction, en passant par l'étiquetage des propositions concernées : ces actions supportent les préconditions d'un ensemble d'actions λ-mutex secondaires, ce qui est une nouvelle forme de causalité matérialisée par l'étiquetage. Dans l'exemple, les préconditions des actions DROP auront les étiquettes des actions DROP durant l'extraction de π. Ces étiquettes sont celles des actions λ-mutex primaires PICK. Cet étiquetage permet celui de l'action MOVE(r-a, r-z) puis de ses préconditions dans $\mathcal{P}_1$, par les étiquettes des actions λ-mutex primaires PICK.

En résumé, le marquage λ des relations mutex permet d'extraire le plan relâché π, et l'étiquetage matérialise les causalités dans π.

7.1.3 Transformation en λ-plan et support externe

La construction du λ-plan se fait en deux principales étapes :

1. le parcours du plan relâché π dans l'ordre des couches d'actions, avec l'ajout en entrelacement des actions[5] non étiquetées dans le plan relâché (c'est-à-dire non concernées par les structures de choix itératif) ;
2. pour chaque clique d'actions λ-mutex primaires rencontrées dans Π, la construction d'une structure de choix itératif qui intègre comme traitement spécifique les actions de Π étiquetées par les actions de la clique. La structure de choix itératif ainsi créée est ajoutée à Π.

C'est principalement le deuxième point qui nous intéresse : la formation de structures de choix itératif.

5. en entrelacement pour une même couche

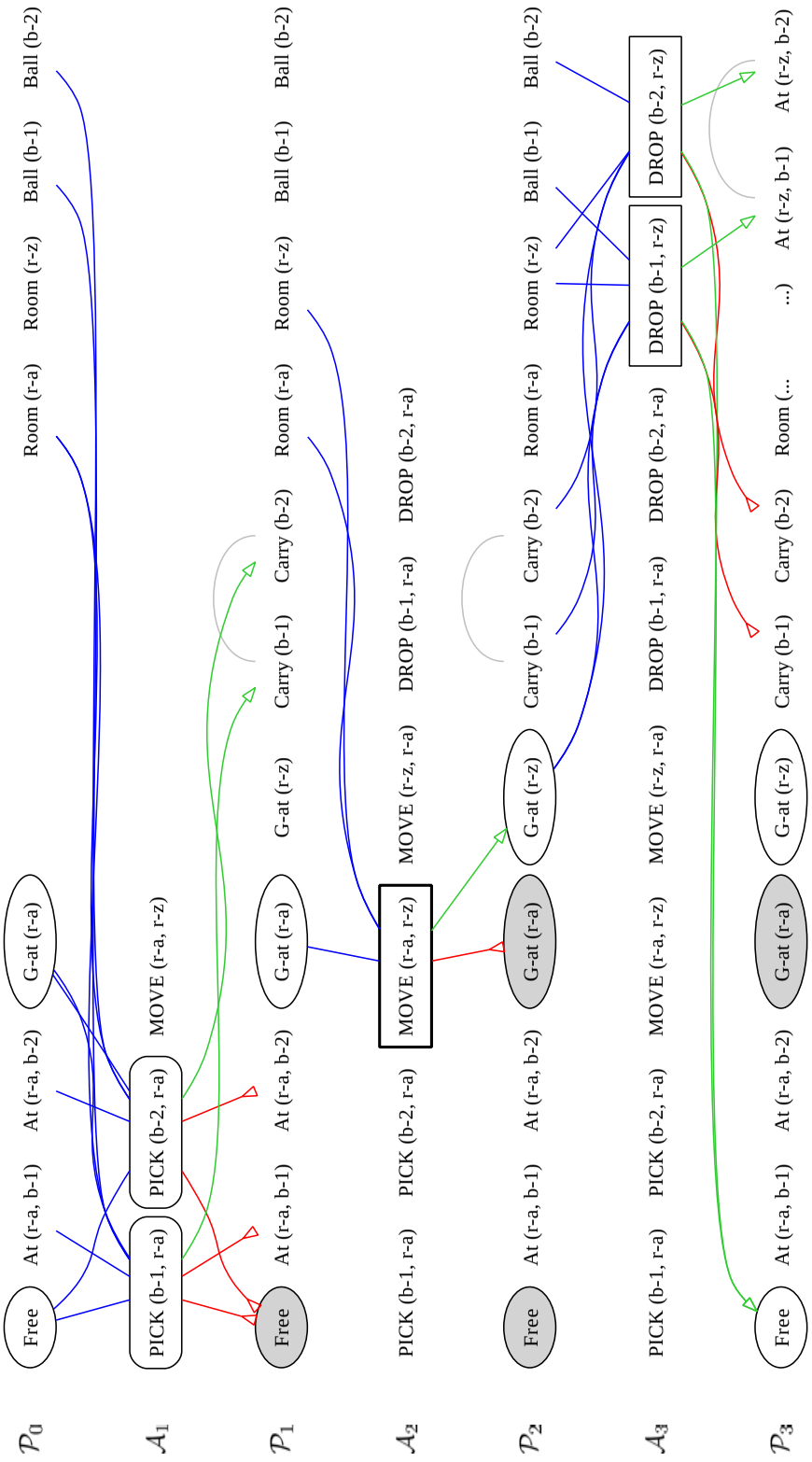


FIGURE 7.4 – L'action `MOVE(r-a, r-z)` est le support interne du λ-plan attendu pour le problème GRIPPER à deux balles. Le support externe doit soutenir les préconditions du λ-plan au fil des itérations.

Construction du traitement des objets

Nous avons remarqué que les actions λ-mutex primaires sont des candidates pour l'initialisation d'une structure de choix itératif. Pour chaque clique d'actions λ-mutex primaires, une telle structure est créée. Toutes les actions qui ont la même classe d'étiquette sont ajoutées. Dans notre exemple, c'est la clique d'actions `PICK` qui initie la structure : `LGP` déduit de cette clique l'action abstraite `PICK(`*ball*`, r-a)` et les objets `b-1`, `b-2` et `b-3` manipulés. L'action `MOVE(r-a, r-z)` est ajoutée à la structure puisqu'elle a été marquée comme fermeture interne. La clique d'actions `DROP` λ-mutex secondaires est transformée en l'action `DROP(`*ball*`, r-z)`, ajoutée à la strucutre. Lorsque plus aucune action de π n'est étiquetée par la clique primaire, la construction du traitement s'arrête. Les actions présentes et étiquetées dans $\mathcal{G}$ forment le traitement à appliquer sur les objets. Pour former entièrement la structure de choix itératif, `LGP` doit encore construire le support externe qui permet les itérations.

Construction du support externe

Dans le λ-plan donné en référence pour Gripper, seule l'action `MOVE(r-z, r-a)` n'a pas été observée dans $\mathcal{G}$:

> choose (*ball*) in {(`b-1`), (`b-2`), (`b-3`)}
> | (`PICK(`*ball*`, r-a)` $\gg$ `MOVE(r-a, r-z)` $\gg$ `DROP(`*ball*`, r-z)`)
> | if-iteration (`MOVE(r-z, r-a)`)

Informellement, cette action, utilisée à chaque itération, permet de traiter les balles les unes après les autres. Formellement, cela signifie que cette action supporte les préconditions du traitement à appliquer sur les balles. Sur le graphe de la figure 7.4 page précédente, nous observons que la proposition `G-AT(r-a)` est une précondition des actions `PICK` de $\mathcal{A}_1$ et de l'action `MOVE` de $\mathcal{A}_2$. Mais cette proposition est supprimée par `MOVE(r-a, r-z)`, et n'est pas reproduite par une autre action du traitement spécifique, au contraire par exemple de la proposition `Free`. Aussi, l'action `MOVE(r-z, r-a)` est calculée pour soutenir les préconditions du λ-plan au fil des itérations, cela après transformation de l'ensemble des actions `PICK`, `MOVE(r-a, r-z)` et `DROP` en traitement spécifique sur les balles. Les actions qui permettent l'enchaînement de l'application d'un traitement sur chaque objet sont appelées le *support externe* du traitement. Pour Gripper, l'action `MOVE(r-a, r-z)` déplace le robot dans la pièce origine des balles à partir de laquelle peut être appliquée le traitement sur ces balles.

7.1.4 Autres cas d'étude

Nous nous intéressons dans ce paragraphe à différents patterns à repérer dans le graphe de planification. Nous observons donc le comportement des relations mutex et des étiquettes pour des problèmes de planification qui produisent des plans flexibles avec :

- des structures de choix itératif successives ;
- des structures de choix itératif imbriquées ;
- des structures de choix itératif entrelacées.

Les domaines de planification utilisés dans cette partie sont décrits au paragraphe 1.2.2, et définis en PDDL [AD05] à l'annexe B.

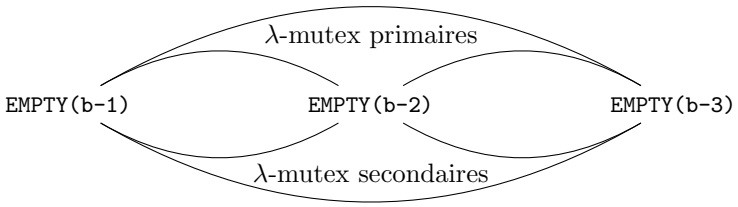
Itérations successives

Nous considérons le domaine Gripper-Cart dans lequel un robot peut attraper dans sa pince une balle située dans la pièce où il se trouve, peut poser la balle qu'il tient dans sa

carriole, reprendre une balle dans sa carriole, poser la balle maintenue dans la pièce où il se trouve, et se déplacer si sa pince est libre. L'objectif de planification est de déplacer des balles `b-1` et `b-2` située dans la pièce `r-a` dans la pièce `r-z`. Le plan flexible correspondant est le suivant :

```
choose (ball) in {(b-1), (b-2)}     ≫ MOVE(r-a, r-z) ≫     choose (ball) in {(b-1), (b-2)}
    | (PICK(ball, r-a)                                          | (EMPTY(ball)
    |     ≫ LOAD(ball))                                        |     ≫ DROP(ball, r-z))
```

Les deux itérations successives permettent de choisir l'ordre dans lequel la carriole est remplie de balles et l'ordre dans lequel elle est vidée. Dans ce λ-plan, l'ordre de vidage de la carriole est indépendant de l'ordre de son remplissage. Dans le graphe de planification présenté à la figure 7.5 page ci-contre, nous observons que les actions `EMPTY` de $\mathcal{A}_4$ sont d'une part λ-mutex secondaires étiquetées par les actions `LOAD` de $\mathcal{A}_2$, et d'autre part λ-mutex primaires :



Ainsi, pour obtenir des itérations successives, un ensemble d'actions en relation λ-mutex initie une structure de choix itératif lorsqu'il forme une clique selon des relations λ-mutex secondaires ainsi qu'une clique selon les relations λ-mutex primaires. Il faut noter qu'un plan flexible pour ce problème est aussi possible sans itérations successives, en déplaçant chaque balle une à une dans la carriole. Mais cette solution est moins flexible que celle proposée, puisque deux itérations successives permettent beaucoup plus d'exécutions différentes qu'une seule sur les mêmes objets.

Itérations imbriquées

Painter permet d'illustrer les itérations imbriquées. Dans ce domaine de planification, le robot doit colorer les deux balles `b-1` et `b-2` de deux couleurs chacune, `black` et `white`. Avant chaque couche de peinture, il faut nettoyer l'espace avec l'action `CLEAN` et préparer la balle à peindre, par exemple la balle `b-1`, avec l'action `WASH(b-1)`. Le plan flexible que calcule `LGP` est le suivant :

```
CLEAN ≫ choose (ball) in {(b-1), (b-2)}
            | (WASH(ball)
            |     ≫ choose (color) in {(black), (white)}
            |           | PAINT(ball, color)
            |           | if-iteration CLEAN ≫ WASH(ball)
            |     if-iteration CLEAN
```

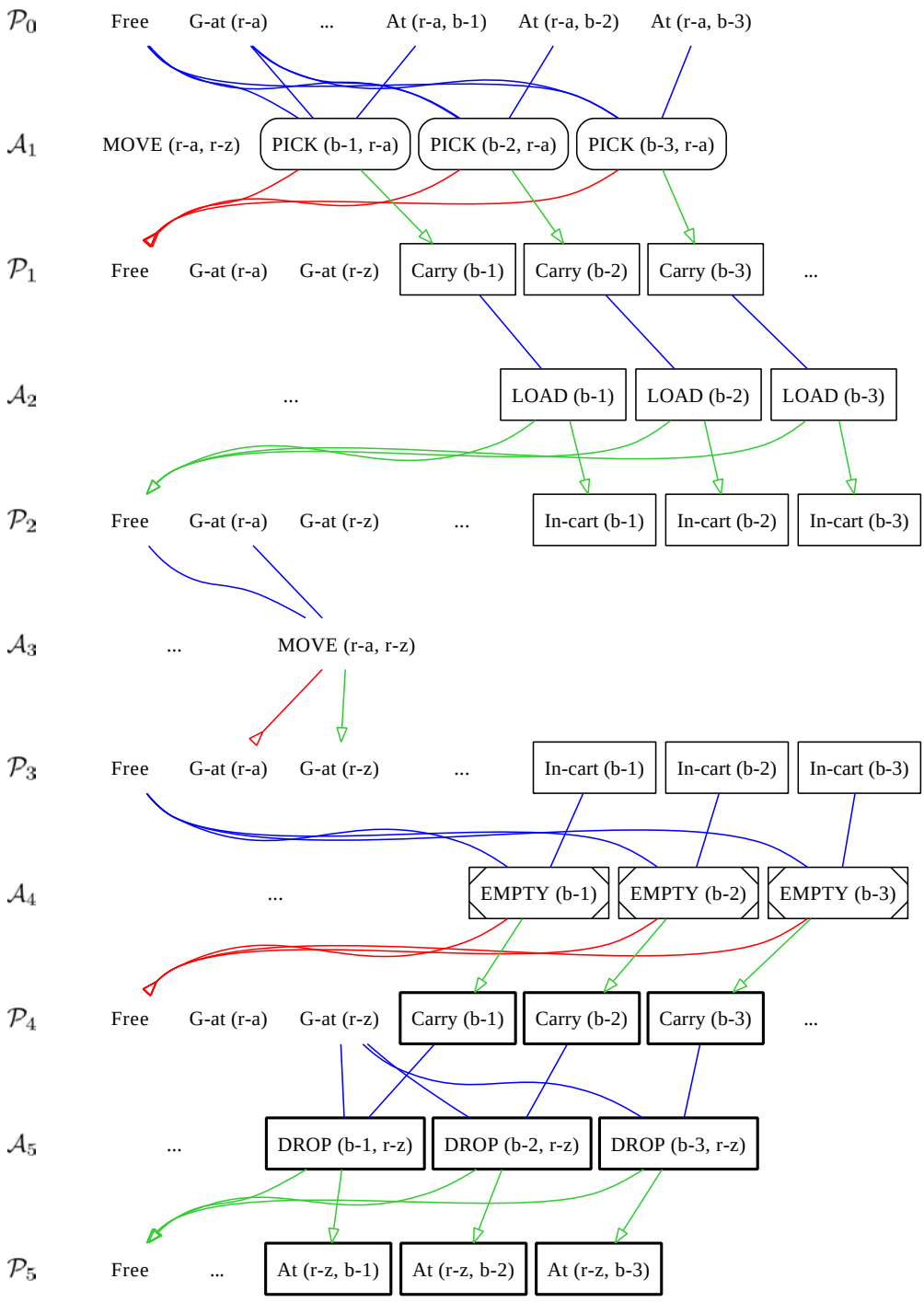
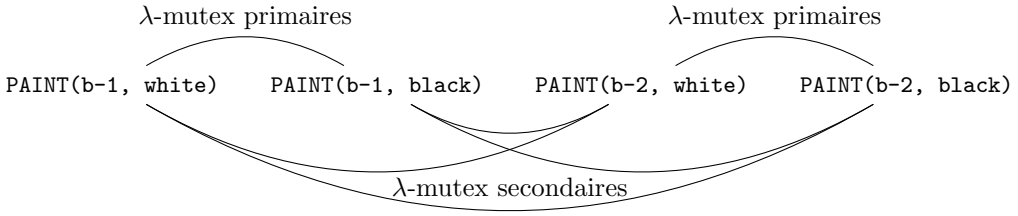


FIGURE 7.5 – Itérations successives avec le domaine de planification GRIPPER-CART : nous observons dans le graphe que les actions dérivées de `EMPTY(`*ball*`)` sont λ-mutex primaires et secondaires, et que dans le plan flexible attendu, elles initient une structure de choix itératif.

Nous observons une première itération sur les balles, et une seconde, imbriquée dans la première, sur les couleurs. Ainsi, à la fin de l'exécution de ce plan flexible, les deux balles sont peintes des deux couleurs. La figure 7.6 page suivante propose une illustration d'une partie du graphe de planification correspondant au problème PAINTER. Dans cet exemple, les propositions qui forment l'objectif de planification sont toutes λ-mutex, mais pour des raisons différentes. En effet, d'une part les actions `PAINT` pour une balle donnée et de couleurs différentes sont λ-mutex primaires entre elles, et d'autre part les actions `PAINT` qui manipulent différentes balles (la couleur est indifférente) sont λ-mutex secondaires :



C'est dans cette configuration que les itérations sont imbriquées : les actions sont soit λ-mutex primaires soit λ-mutex secondaires, et forment ainsi une clique λ-mutex. `LGP` construit une nouvelle structure de choix itératif pour l'un des ensembles d'actions λ-mutex primaires, par exemple pour les actions `PAINT(b-1, white)` et `PAINT(b-1, black)`. Cela donne la structure suivante :

choose (*color*) in {(`black`), (`white`)}
  | `PAINT(b-1,` *color*`)`
  | if-iteration `CLEAN` ≫ `WASH(b-1)`

Cette structure de choix itératif est alors adaptée pour permettre la manipulation de toutes les balles : la balle `b-1` est transformée en variable *ball*. La structure ainsi abstraite est intégrée à la première structure de choix itératif.

Itérations entrelacées

Avec le problème issue de GRIPPER-B+C, deux robots sont disponibles : chacun peut manipuler respectivement des balles ou des cubes à déplacer. Dans cet exemple, les deux robots sont indépendants, ce qui permet d'entrelacer leurs actions. Ainsi, le plan flexible solution propose deux structures de choix itératif en entrelacement :

choose (*ball*) in {(`b-1`), (`b-2`)}
  | (`PICK-B(g-b,` *ball*`, r-a)`
  | ≫ `MOVE(g-b, r-a, r-z)`
  | ≫ `DROP-B(g-b,` *ball*`, r-z)`)
  | if-iteration `MOVE(g-b, r-z, r-a)`

‖

choose (*cube*) in {(`c-1`), (`c-2`)}
  | (`PICK-C(g-c,` *cube*`, r-z)`
  | ≫ `MOVE(g-c, r-z, r-a)`
  | ≫ `DROP-C(g-c,` *cube*`, r-a)`)
  | if-iteration `MOVE(g-c, r-a, r-z)`

La figure 7.7 page 104 reproduit la partie du graphe de planification de GRIPPER-B+C. Ce graphe permet de visualiser l'indépendance de deux cliques d'actions λ-mutex primaires dans la couche $\mathcal{A}_1$. La première clique initie la structure de choix itératif permettant de déplacer les balles `b-1` et `b-2`, tandis que la seconde clique initie celle qui permet de déplacer

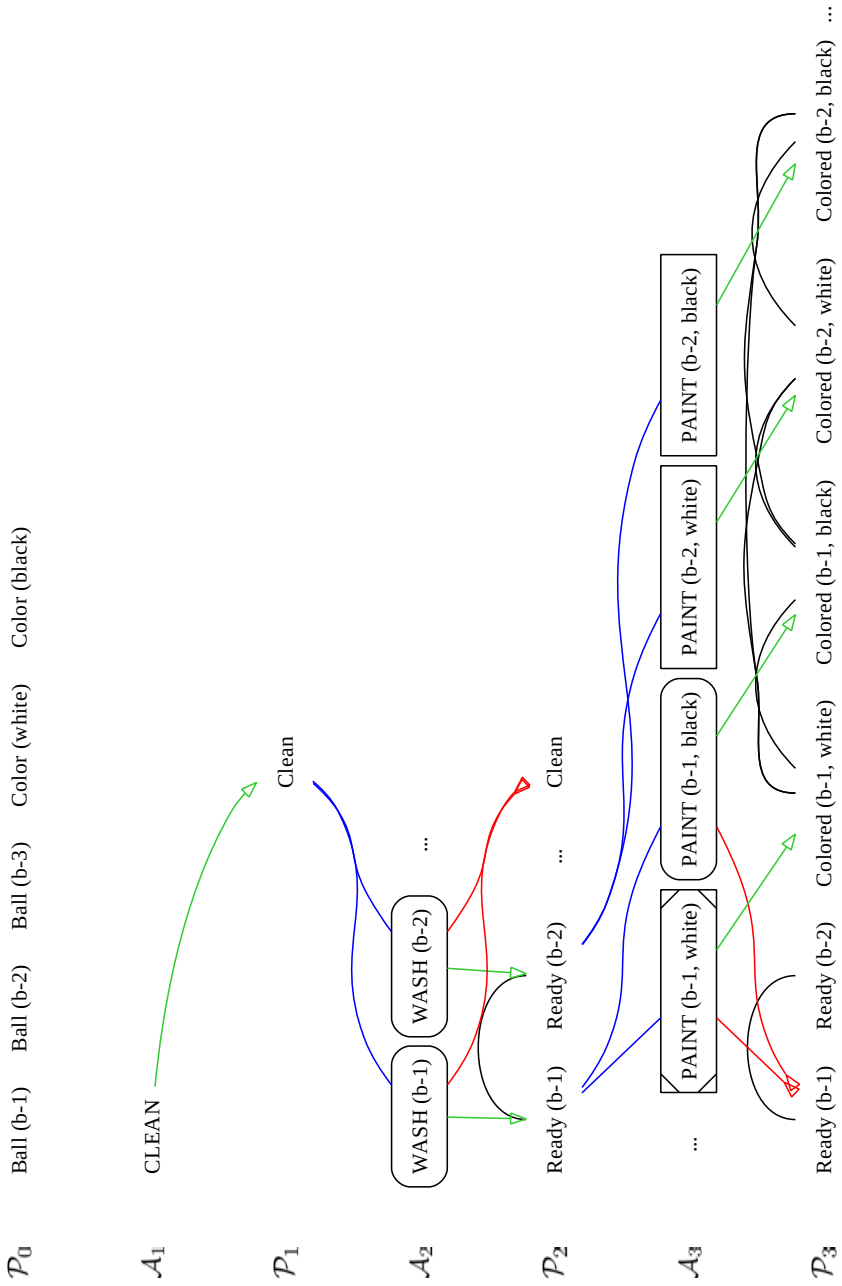


FIGURE 7.6 – Partie du graphe de planification pour le problème PAINTER : ce pattern particulier de relations λ-mutex révèle des itérations imbriquées. Les actions `WASH` sont λ-mutex primaires du fait de la proposition `CLEAN`. Elles produisent les propositions `READY`, λ-mutex entre elles. L'actions `PAINT(b-1,white)` est d'une part λ-mutex primaires avec `PAINT(b-1,black)` par interférence de la proposition `READY(b-1)`, et d'autre part λ-mutex secondaires avec les autres actions du fait des propositions `READY(b-1)` et `READY(b-2)` λ-mutex dans $\mathcal{P}_2$. L'analyse est la même avec chacune des actions `PAINT`.

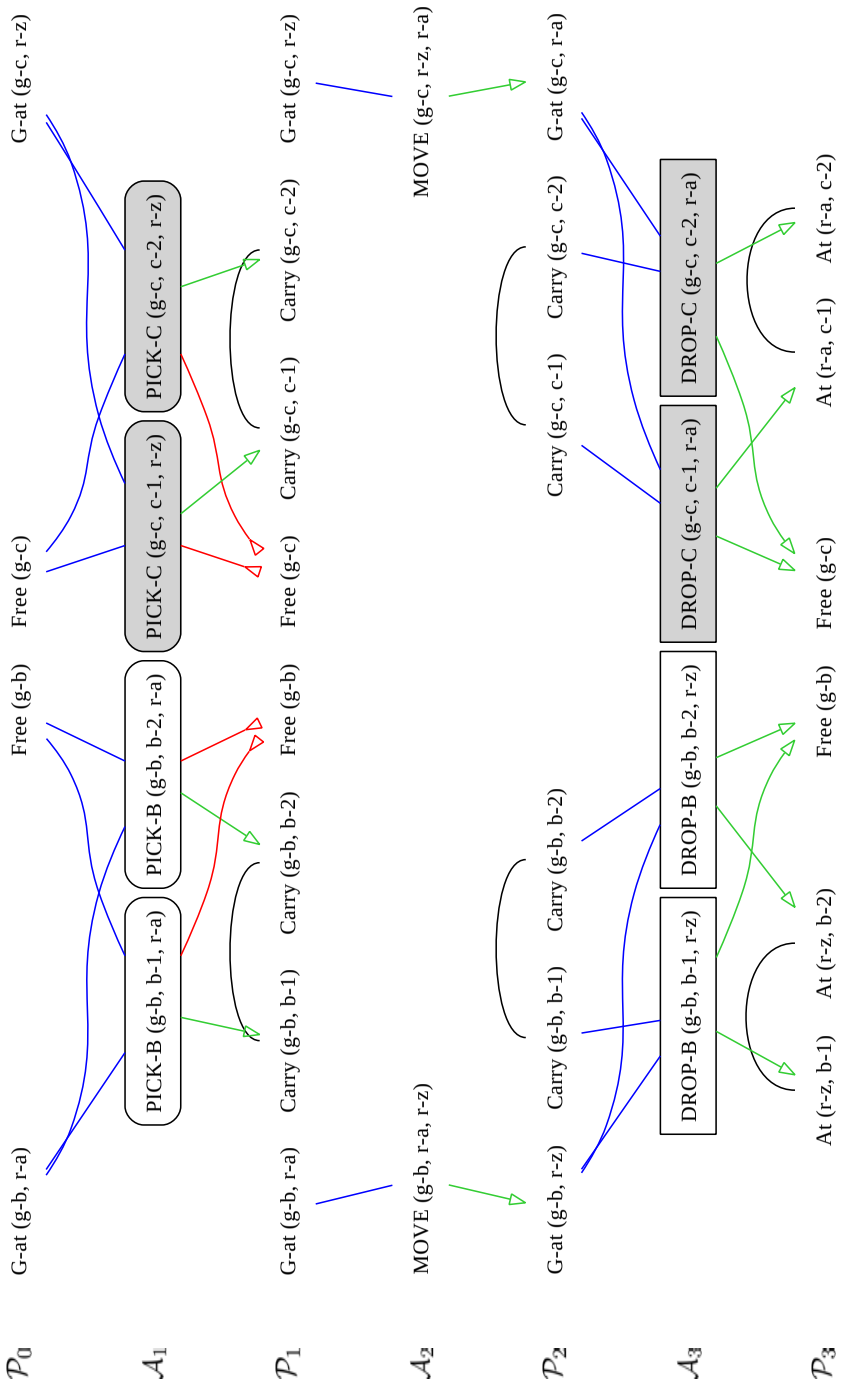


FIGURE 7.7 – Partie du graphe de planification pour le problème GRIPPER-B+C : ce pattern particulier de relations λ-mutex révèle des itérations entrelacées. Le graphe contient deux cliques λ-mutex primaires d'actions dans la couche $\mathcal{A}_1$, sans que ces cliques n'aient de relation entre elles. L'étiquetage des propositions et actions se trouve séparé en deux parties distinctes dans le graphe (claire ou foncée) : cela permet la formation de deux structures indépendantes de choix itératif à partir de la couche $\mathcal{A}_1$.

les cubes `c-1` et `c-2`. Ainsi, lorsque dans une couche plusieurs cliques d'actions λ-mutex primaires coexistent, les structures de choix itératif déduites de ces cliques peuvent être ajoutées dans le plan flexible en entrelacement si les structures sont indépendantes.

7.2 Construction du graphe

L'utilisation d'un graphe de planification permet de manipuler un grand nombre d'informations sur les états successifs de façon condensée : c'est cette concentration d'informations que nous utilisons dans `LGP` pour découvrir les actions susceptibles d'appartenir à un λ-plan solution. Dans cette partie, les concepts manipulés sont formalisés, et l'étape d'extension du graphe est détaillée. Pour rappel, nous notons dans $\mathcal{G}$ pour le pas de temps i :

- $\mathcal{P}_i$ la couche de propositions ;
- $\mathcal{A}_i$ la couche d'actions ;
- $\mu\mathcal{P}_i$ l'ensemble de couples de propositions mutex de $\mathcal{P}_i$;
- $\mu\mathcal{A}_i$ l'ensemble de couples d'actions mutex de $\mathcal{A}_i$.

7.2.1 Relation λ-mutex

Les relations mutex à relâcher sont marquées λ par `LGP`. Nous notons $\lambda\mathcal{A}_i$ l'ensemble de couples d'actions λ-mutex de $\mathcal{A}_i$, et $\forall i, \lambda\mathcal{A}_i \subseteq \mu\mathcal{A}_i$. De la même façon, nous notons $\lambda\mathcal{P}_i$ l'ensemble de couples de propositions λ-mutex de $\mathcal{P}_i$, et $\forall i, \lambda\mathcal{A}_i \subseteq \mu\mathcal{A}_i$. Les définitions formelles des relations λ-mutex sont les suivantes.

Définition 7.1 (Relation λ-mutex entre actions)**.** *Soient deux actions a_1 et a_2 mutex ($(a_1, a_2) \in \mu\mathcal{A}_i$). Soit un opérateur $ope \in \mathcal{D}$ et deux substitutions σ_1 et σ_2 telles que $a_1 = \sigma_1(ope)$ et $a_2 = \sigma_2(ope)$. a_1 et a_2 sont λ-mutex si au moins l'une des deux propriétés suivantes est vérifiée :*

- *λ-mutex primaire : a_1 et a_2 interfèrent (*`interference`*(a_1, a_2)) ;*
- *λ-mutex secondaire : $\forall (p_1, p_2)$ avec $p_1 \in pre(a_1)$ et $p_2 \in pre(a_2)$ alors, soit $(p_1, p_2) \in \lambda\mathcal{P}_{i-1}$, soit $(p_1, p_2) \notin \mu\mathcal{P}_{i-1}$.*

Pour vérifier qu'une relation entre deux actions $\sigma_1(ope)$ et $\sigma_2(ope)$ est λ-mutex primaire, `LGP` vérifie que σ_1 et σ_2 sont telles que l'application de la substitution $\sigma = \sigma_1 \cap \sigma_2$ sur l'opérateur ope vérifie que $pre(\sigma(ope)) \cap eff^-(\sigma(ope)) \neq \emptyset$ (par définition de l'interférence).

Les propositions λ-mutex sont celles qui sont accessibles au pas de temps i et qui dérivent d'actions pouvant être intégrées à une structure de choix itératif. Plus formellement :

Définition 7.2 (Relation λ-mutex entre propositions)**.** *Soient deux propositions p_1 et p_2 mutex ($(p_1, p_2) \in \mu\mathcal{P}_i$). p_1 et p_2 sont λ-mutex si et seulement s'il existe $(a_1, a_2) \in \lambda\mathcal{A}_i$ tel que $p_1 \in eff^+(a_1)$ et $p_2 \in eff^+(a_2)$.*

Notre approche avec `LGP` consiste à construire des structures de choix itératif pour des traitements pouvant être permutés mais pas entrelacés. En effet, si les traitements peuvent être entrelacés, alors les proposer en permutation reviendrait à une perte de flexibilité : le choix itératif est moins flexible que l'entrelacement (*cf.* paragraphe 6.6). L'impossibilité d'entrelacer les traitements se traduit dans $\mathcal{G}$ par les relations mutex, tandis que la possibilité de permuter les traitements est matérialisée par `LGP` avec les relations λ-mutex. Ainsi, les ensembles d'actions à intégrer dans les traitements forment des cliques selon la relation λ-mutex : aucune ne doit pouvoir être proposée en entrelacement au risque de perdre en flexibilité.

7.2.2 Classe d'actions λ-mutex primaires

Nous intégrons dans `LGP` des *classes* d'actions λ-mutex primaires, ou classe de sources primaires, qui garantissent que tout couple d'actions d'une même classe soit en relation λ-mutex primaire.

Dans PAINTER, les actions dérivées de l'opérateur `PAINT(`*ball*, *color*`)` ne sont pas forcément λ-mutex entre elles. Par exemple, les actions `PAINT(b-1, black)` et `PAINT(b-2, black)` n'interfèrent pas. En revanche, toutes les actions dérivées de `PAINT(b-1,` *color*`)` sont λ-mutex primaires : la proposition qui signifie que la balle `b-1` est prête à être peinte, `READY(b-1)`, est une précondition et un effet négatif de l'action abstraite `PAINT(b-1,` *color*`)`. Autrement dit, $\texttt{c-}\mathit{pre}(\texttt{PAINT(b-1,}\ \mathit{color}\texttt{)}) \cap \texttt{c-}\mathit{eff}^-(\texttt{PAINT(b-1,}\ \mathit{color}\texttt{)}) = \{\texttt{READY(b-1)}\} \neq \emptyset$. Ainsi, quels que soient les actions issues de `PAINT(b-1,` *color*`)`, elles sont en relation λ-mutex primaire. `PAINT(b-1,` *color*`)` est appelée une *classe* de sources primaires. Ainsi, les classes de sources primaires sont des actions abstraites telles que leurs ensembles de préconditions invariables (`c-`*pre*) et d'effets négatifs invariables (`c-`eff^-) ont des propositions en commun.

Une source primaire peut être dérivée de plusieurs classes de sources primaires. Prenons par exemple l'opérateur `OPE(`x, y, z`)` dont les préconditions et effets négatifs sont définis comme suit :

- $\mathit{pre}(\texttt{OPE}) = \{\texttt{P(}x\texttt{)}, \texttt{Q(}y\texttt{)}\}$;
- $\mathit{eff}^-(\texttt{OPE}) = \{\texttt{P(}x\texttt{)}, \texttt{Q(}y\texttt{)}\}$.

L'action `OPE(a, b, c)` interfère d'une part avec l'action `OPE(a,` y`,` z`)` à cause de la proposition `P(a)` et d'autre part avec l'action `OPE(`x`, b,` z`)` à cause de la proposition `Q(b)`. Ainsi, l'action `OPE(a, b, c)` est dérivée de deux classes de sources primaires :

- `OPE(a,` y`,` z`)` ;
- `OPE(`x`, b,` z`)`.

Il est nécessaire de connaître toutes les classes dont est dérivée chaque source primaire pour s'assurer de découvrir toutes les cliques d'actions λ-mutex.

Les classes de sources primaires sont déduites de la définition des opérateurs de $\mathcal{D}$. Ainsi, pour un opérateur *ope* donné, l'ensemble des classes de sources primaires est l'ensemble des actions abstraites $\{\alpha = \sigma(\mathit{ope}) \mid \texttt{c-}\mathit{pre}(\alpha) \cap \texttt{c-}\mathit{eff}^-(\alpha) \neq \emptyset\}$.

7.2.3 Étiquetage

Les étiquettes ont deux principaux rôles dans `LGP` :

- indiquer dans $\mathcal{G}$ les ensembles d'actions qui forment des cliques selon la relation λ-mutex ;
- matérialiser la causalité entre les actions dans $\mathcal{G}$ pour découvrir les traitements et les objets sur lesquels appliquer les traitements.

Définition 7.3 (Étiquette)**.** *Une étiquette est un couple* (α, σ)*, où :*

- α *est une* classe *de sources primaires, appelée la classe de l'étiquette ;*
- σ *est l'*objet *de l'étiquette, une substitution.*

La classe des étiquettes est utilisée pour découvrir les cliques d'actions λ-mutex (primaires et secondaires), tandis que l'objet de la classe permet de matérialiser la causalité.

Règles de création d'une étiquette

Pour une action $\sigma_1(\mathit{ope})$ ayant au moins une relation λ-mutex primaires, une étiquette est associée pour chaque classe $\sigma(\mathit{ope})$ telle que $\sigma \subseteq \sigma_1$, et l'étiquette est alors liée à la classe $\sigma(\mathit{ope})$ et à l'objet $\sigma_1 \setminus \sigma$. Pour l'action `OPE(a, b, c)` décrite précédemment, les deux étiquettes qui lui seront associées sont :

– l'étiquette (`OPE(a, y, z)`, $\{y\backslash$`b`$, z\backslash$`c`$\}$) ;
– l'étiquette (`OPE(`x`, b, z)`, $\{x\backslash$`a`$, z\backslash$`c`$\}$) ;

L'action `PAINT(b-1, black)` de PAINTER sera étiqueté avec la classe `PAINT(b-1,` *color*`)` et l'objet (*color*$\backslash$`black`). L'action `PICK(b-1, r-a)` de GRIPPER a dans $\mathcal{G}$ une étiquette de classe `PICK(`*ball*, *room*`)` et d'objet (*ball*$\backslash$`b-1`, *room*$\backslash$`r-a`).

L'étiquetage dans $\mathcal{G}$ débute systématiquement par une source primaire : une étiquette est associée à une source primaire pour chaque classe dont la source est dérivée. Ainsi, chaque action source primaire est associée aux étiquettes dont la classe explique les relations λ-mutex primaires.

Règles de propagation d'une étiquette

Une proposition de la couche $\mathcal{P}_i$ ayant des relations λ-mutex est étiquetée avec toutes les étiquettes de toutes les actions de $\mathcal{A}_i$ qui ont pour effet positif la proposition en question. Une action de la couche $\mathcal{A}_{i+1}$ ayant des relations λ-mutex secondaires est étiquetée avec toutes les étiquettes de toutes ses préconditions de $\mathcal{P}_i$. La figure 7.8 page suivante illustre l'étiquetage des actions et des propositions dans le graphe de planification pour GRIPPER.

7.2.4 Extension du graphe

La première étape de `LGP` consiste à *étendre* pas à pas le graphe de planification $\mathcal{G}$ selon les règles suivantes, et dans cet ordre :

1. une action a est ajoutée à $\mathcal{A}_i$ si ses préconditions ne sont pas mutex dans $\mathcal{P}_{i-1}$, c'est-à-dire si $\nexists (p_1, p_2) \in pre(a) \times pre(a) \mid (p_1, p_2) \in \mu\mathcal{P}_{i-1}$;
2. une proposition p est ajoutée à $\mathcal{P}_i$ si une action de $\mathcal{A}_i$ a p pour effet positif ou négatif.

Les relations mutex et λ-mutex sont ajoutées dans le graphe de planification lors de son expansion. L'algorithme 7.1 résume le processus d'extension par la fonction λ`Expand`. À chaque couche i du graphe de planification $\mathcal{G}$, les informations disponibles sont $\mathcal{A}_i$, $\mu\mathcal{A}_i$,

Algorithme 7.1 λExpand$(\mathcal{G} = \langle \cdots, \mathcal{A}_{i-1}, \mu\mathcal{A}_{i-1}, \lambda\mathcal{A}_{i-1}, \mathcal{P}_{i-1}, \mu\mathcal{P}_{i-1}, \lambda\mathcal{P}_{i-1} \rangle, \mathcal{A}, i)$

```
1  ACTS_i ← {a ∈ A | pre(a) ∈ P_{i-1} ∧ pre(a) × pre(a) ∩ μP_{i-1} = ∅} ;
2  NOOPS_i ← {NOOP-PRED | PRED ∈ P_{i-1}};
   // pre(NOOP-PRED) = PRED, eff+(NOOP-PRED) = PRED, eff−(NOOP-PRED) = ∅
3  A_i ← ACTS_i ∪ NOOPS_i ;
4  P_i ← {p | ∃a ∈ A_i, (p ∈ eff+(a) ∨ p ∈ eff−(a))} ;
5  μA_i ← {(a, b) ∈ A_i × A_i, a ≠ b | a, b are mutex} ;
6  μP_i ← {(p, q) ∈ P_i × P_i, p ≠ q | p, q are mutex} ;
7  λA_i ← {(a, b) ∈ μA_i | a, b are λ-mutex} ;
8  λP_i ← {(p, q) ∈ μP_i | p, q are λ-mutex} ;
9  foreach a ∈ A_i do
10     link a with a precondition arcs to pre(a) in P_{i-1} ;
11     link a with a positive arcs to eff+(a) in P_i ;
12     link a with a negative arcs to eff−(a) in P_i ;
13 return G.⟨A_i, μA_i, λA_i, P_i, μP_i, λP_i⟩ ;
```

$\lambda\mathcal{A}_i$, $\mathcal{P}_i$, $\mu\mathcal{P}_i$ et $\lambda\mathcal{P}_i$. La couche $\mathcal{P}_0$ contient toutes les propositions de l'état initial, et $\mu\mathcal{P}_0 = \lambda\mathcal{P}_0 = \emptyset$.

L'étiquetage des actions et des propositions n'est pas mentionné dans cet algorithme mais a lieu à ce moment là, comme décrit dans le paragraphe 7.2.3 :

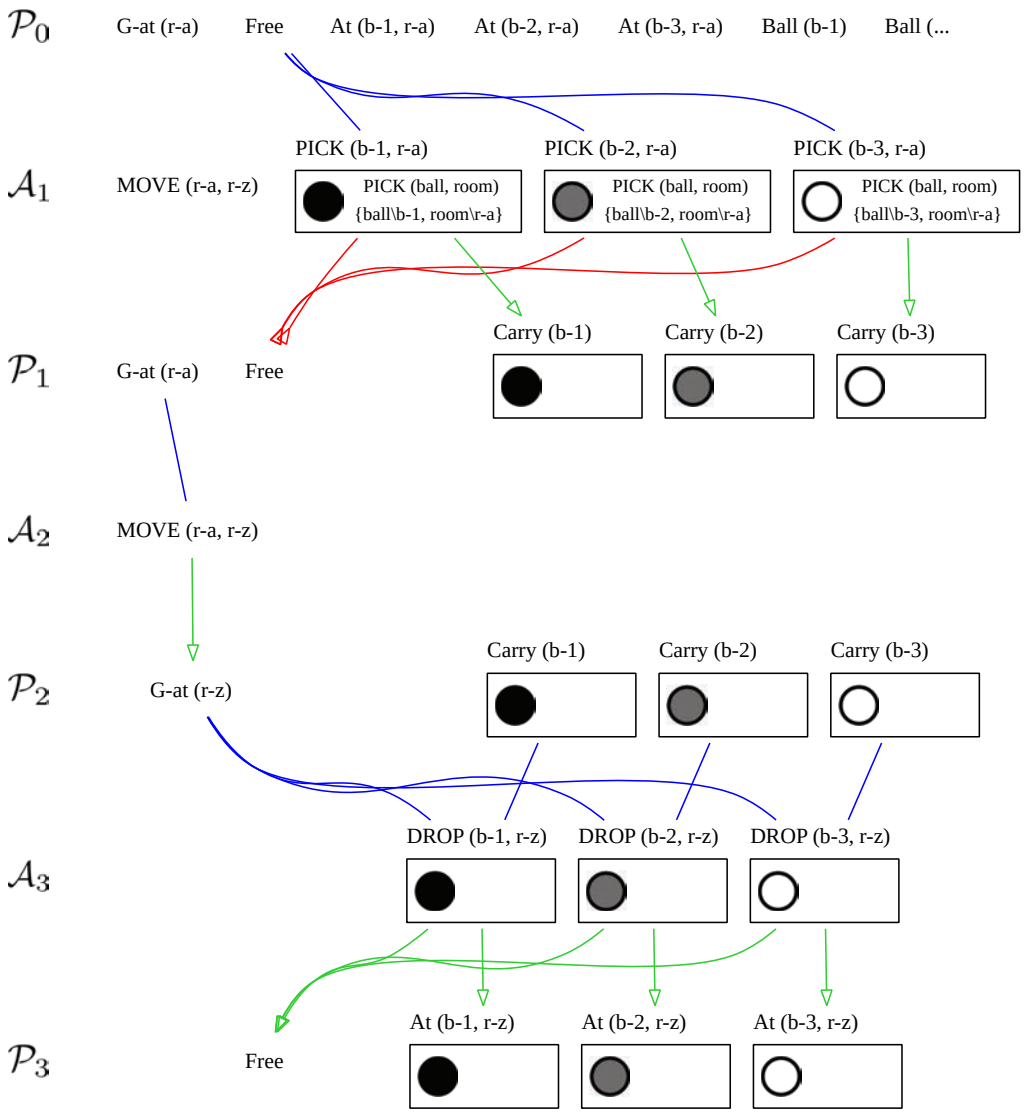


FIGURE 7.8 – Etiquetage du graphe de planification avec le domaine de planification GRIPPER : les étiquettes pour les actions `PICK` sont créées dans la couche $\mathcal{A}_1$, puis propagées dans $\mathcal{P}_1$ aux effets λ-mutex `CARRY` puis les actions `NOOP-CARRY` de $\mathcal{A}_2$ (non représentées) sont étiquetées, ainsi de suite. Les étiquettes $(\texttt{PICK}(ball, room), (ball\backslash\texttt{b-1}, room\backslash\texttt{r-a}))$, $(\texttt{PICK}(ball, room), (ball\backslash\texttt{b-2}, room\backslash\texttt{r-a}))$ et $(\texttt{PICK}(ball, room), (ball\backslash\texttt{b-3}, room\backslash\texttt{r-a}))$ sont représentées respectivement par les symboles ●, ◉ et ○.

– les actions qui ont des relations λ-mutex primaires sont étiquetées lors de la détection de ces relations ;
– les propositions en relation λ-mutex sont étiquetées lors de la détection de leurs relations λ-mutex ;
– les actions qui ont des relations λ-mutex secondaires sont étiquetées lors de la détection de ces relations.

Les étiquettes associées aux actions `PICK(b-1, r-a)`, `PICK(b-2, r-a)` et `PICK(b-3, r-a)` de $\mathcal{A}_1$ du graphe représenté à la figure 7.3 page 95 sont toutes de la classe `PICK`$(ball, room)$, et leurs objets sont respectivement :
– $\{ball\backslash$`b-1`$, room\backslash$`r-a`$\}$;
– $\{ball\backslash$`b-2`$, room\backslash$`r-a`$\}$;
– $\{ball\backslash$`b-3`$, room\backslash$`r-a`$\}$.

Ces étiquettes sont propagées dans les effets positifs respectifs de ces actions `PICK` s'ils sont λ-mutex dans $\mathcal{P}_1$. Par exemple, la propositions `CARRY(b-1)` est étiquetée (`PICK`$(ball, room)$, $\{ball\backslash$`b-1`$, room\backslash$`r-a`$\}$). En accord avec les règles de propagation des étiquettes, l'action virtuelle `NOOP-CARRY(b-1)` a la même étiquette. Et ainsi de suite, l'étiquette est propagée dans tout le graphe permettant la détection des sources primaires des relations λ-mutex.

La partie suivante détaille l'extraction d'un plan relâché par la relaxation dans $\mathcal{G}$ des relations λ-mutex.

7.3 Extraction d'un plan relâché

Pour extraire des plans relâchés, `LGP` utilise les informations fournies par les relations λ-mutex pour relaxer les contraintes mutex : les actions d'une étape du plan ne doivent toujours pas être mutex entre elles, sauf dans le cas des actions susceptibles d'appartenir à un traitement spécifique. Dans cette partie, la condition pour l'extraction d'un plan relâché est définie et l'étape d'extraction est détaillée.

7.3.1 Condition pour l'extraction d'un plan relâché

Les propositions λ-mutex, donc produites par des actions λ-mutex, doivent pouvoir être utilisées comme effets probablement produits par une structure de choix itératif qui intègre les actions λ-mutex. Ainsi, l'objectif de planification *goal* est considéré comme accessible à la couche $\mathcal{P}_g$ même si celui-ci contient des propositions λ-mutex. La couche $\mathcal{P}_g$ est donc atteinte pour un objectif *goal* lorsque $goal \subseteq \mathcal{P}_g$ et $goal \times goal \cap (\mu\mathcal{P}_g \setminus \lambda\mathcal{P}_g) = \emptyset$. Par le relâchement des relations λ-mutex entre propositions, le critère d'atteignabilité de l'objectif de planification est plus rapidement satisfait dans `LGP` que dans `GP`.

7.3.2 Extraction d'un plan relâché

Lorsque $\mathcal{P}_g$ est atteint, `LGP` tente d'*extraire* un plan de $\mathcal{G}$ (*cf.* algorithmes 7.2 et 7.3) : chaque proposition sélectionnée de la couche $\mathcal{P}_i$ doit être produite par au moins une action de la couche $\mathcal{A}_i$. Toutes les préconditions de cet ensemble d'actions de $\mathcal{A}_i$ sont sélectionnées

Algorithme 7.2 λExtract$(\mathcal{G}, goal, i)$

```
1 if i = 0 then return ϵ ;
2 return λSearch(G, goal, ϵ, i) ;
```

Algorithme 7.3 λSearch$(\mathcal{G} = \langle \ldots, \mathcal{A}_i, \mu\mathcal{A}_i, \lambda\mathcal{A}_i, \ldots \rangle, goal, \pi, i)$

```
1  if goal = ∅ then
2  |  Π ← λExtract(G, ⋃{pre(a) | a ∈ π}, i − 1) ;
3  |  if Π = failure then return failure ;
4  |  return Π ≫ (π) ;
5  else
6  |  choix non déterministe de p ∈ goal ;
7  |  E_a ← μA_i \ λA_i;
   |  // tuples d'actions ne pouvant être utilisées ensembles au pas i
8  |  S_p ← {a ∈ A_i | p ∈ eff⁺(a), ∀b ∈ π, ⟨a, b⟩ ∉ E_a} ;
   |  // ensemble des actions candidates pour soutenir p, selon π
9  |  if S_p = ∅ then return failure ;
10 |  choix non déterministe de a ∈ S_p ;
11 |  return λSearch(G, g \ eff⁺(a), π ∥ a, i) ;
```

dans $\mathcal{P}_{i-1}$ et ainsi de suite. Cela peut être vu comme la recherche d'un chemin dans un arbre *et/ou* (les nœuds *et* correspondent au soutien des propositions, les *ou* au choix des actions). Ce processus de sélection commence avec les propositions formant l'objectif de planification dans la dernière couche propositionnelle de $\mathcal{G}$ et s'arrête lorsque $\mathcal{P}_0$ est atteint (*cf.* ligne 1 de l'algorithme 7.2).

La seule différence entre le processus d'extraction de `GP` et celui de `LGP` concerne les conditions pour sélectionner des actions et des propositions dans une couche donnée de $\mathcal{G}$. Les actions qui produisent les propositions sélectionnées dans une couche doivent former un ensemble sans relation *mutex* entre elles ou être en relation λ-mutex : pour chaque couche i dans le processus d'extraction du plan, $\mathcal{E}_a \leftarrow \mu\mathcal{A}_i \setminus \lambda\mathcal{A}_i$ est l'ensemble des couples d'actions qui ne peuvent pas être sélectionnés ensemble (*cf.* ligne 7 de l'algorithme 7.3). Pour rappel, le plan relâché extrait pour Gripper est le suivant :

```
(PICK(b-1, r-a) ∥ PICK(b-2, r-a) ∥ PICK(b-3, r-a))
                ≫ MOVE(r-a, r-z) ≫
(DROP(b-1, r-z) ∥ DROP(b-2, r-z) ∥ DROP(b-3, r-z))
```

Dans ce plan relâché, les trois actions `PICK` et les trois actions `DROP` sont étiquetées. Les étiquettes de ces actions sont liées à la classe `PICK`$(ball, room)$ et respectivement aux objets $(ball\backslash$`b-1`$, room\backslash$`r-a`$)$, $(ball\backslash$`b-2`$, room\backslash$`r-a`$)$ et $(ball\backslash$`b-3`$, room\backslash$`r-a`$)$. Dans le plan relâché, les actions sont soit sans relation mutex, soit λ-mutex.

7.3.3 Étiquetage des actions de fermeture interne

Les étiquettes associées aux actions λ-mutex sont conservées dans le plan relâché. Mais en plus de cet étiquetage, réalisé lors de l'expansion du graphe, un autre étiquetage a lieu durant l'extraction du plan et concerne les actions qui sont supports internes d'actions λ-mutex secondaires. Ainsi, toutes les actions du plan relâché qui soutiennent une précondition d'une action étiquetée sont étiquetées à leur tour de l'ensemble des étiquettes des actions qu'elles soutiennent, et provoque à leur tour l'étiquetage des actions qui les soutiennent. Par exemple, l'action `MOVE(r-a, r-z)` de Gripper est étiquetées trois fois, par (`PICK`$(ball, room)$, $(ball\backslash$`b-1`$, room\backslash$`r-a`$)$) (`PICK`$(ball, room)$, $(ball\backslash$`b-2`$, room\backslash$`r-a`$)$) et

(PICK(*ball*, *room*), (*ball*\b-3, *room*\r-a)). La figure 7.9 page suivante illustre l'étiquetage matérialisant la fermeture interne. Si une action de $\mathcal{A}_1$ produisant G-AT(r-a) était extraite dans le plan relâché, elle serait étiquetée comme MOVE(r-a, r-z). Cet étiquetage est utilisé ensuite pour la transformation du plan relâché : les actions λ-mutex étiquetées sont ajoutées au traitement, ainsi que les actions non mutex mais étiquetées pour la fermeture interne.

La partie suivante détaille la transformation d'un plan relâché en un λ-plan.

7.4 Transformation en λ-plan

L'originalité de LGP réside principalement dans la transformation du plan relâché en λ-plan intègrant des structures de choix itératif. Dans cette partie, nous faisons l'hypothèse que nous avons obtenu un plan relâché π de l'étape d'extraction. La transformation est détaillée et décomposée en deux algorithmes (*cf.* algorihmes 7.5 et 7.6) à la fin de cette partie, le second permettant spécifiquement les créations de structures de choix itératif. Pour rappel, un λ-plan est une séquence de couches, et dans chaque couche sont entrelacés des actions et des structures de choix itératif (*cf.* grammaire des λ-plans, paragraphe 6.7). Pour opérer la transformation de π, LGP parcourt π de la première à la dernière couche, et le λ-plan solution est construit dans le même ordre en commençant par un λ-plan vide. Dans une couche donnée de π, les actions qui ne sont pas λ-mutex sont ajoutées à la couche du λ-plan en cours de construction (en entrelacement). En revanche, pour chaque clique P^{ri} d'actions λ-mutex primaires, LGP initie une nouvelle structure *Structure* de choix itératif. À partir de P^{ri}, LGP détermine l'action abstraite α dont sont dérivées toutes les actions de la clique. L'ensemble Σ des objets manipulés est déduit de P^{ri} et de α. *Structure* est composé d'un traitement $treat(obj)$ spécifique pour les objets de Σ dont la première action est α. Pour former la suite de $treat(obj)$, LGP parcourt les couches suivantes de π à la recherche des actions étiquetées comme les actions de P^{ri} :

- si une action a est étiquetées par toutes les étiquettes des actions de P^{ri}, a est ajoutée à $treat(obj)$: les étiquettes indiquent que a est un support interne pour $treat(obj)$, et donc que des actions λ-mutex secondaires auront besoin de a pour soutenir leurs préconditions ;
- si les actions étiquetées forment une clique selon les relations λ-mutex secondaires, l'action abstraite β qui les représente est déduite puis β est ajoutée à $treat(obj)$: les étiquettes matérialisent un lien de causalité à respecter par l'ajout de β au sein de $treat(obj)$;
- si les actions étiquetées forment une clique selon les relations λ-mutex primaires et secondaires, une nouvelle structure de choix itératif est initiée et utilisée selon le cas rencontré : les structures sont de type successives (*cf.* paragraphe 7.1.4) ou de type imbriquées (*cf.* paragraphe 7.1.4).

Tout au long de la construction, la validité du plan est vérifiée : chaque action du plan relâché doit apparaitre dans le λ-plan une et une seule fois, et le λ-plan doit être applicable sur l'état initial. Durant la transformation, LGP simule l'exécution du λ-plan sur l'état initial.

Nous présentons dans la suite comment les cliques primaires P^{ri} sont détectées dans une couche d'actions du plan relâché puis traitées pour la construction d'une nouvelle structure de choix itératif *Structure*. Nous détaillons par la suite le comportement de LGP face aux cliques d'actions étiquetées par les actions de P^{ri}, c'est-à-dire les cliques secondaires S^{ec}. Les principes mis en place pour construire les supports internes et externes de *Structure* sont discutés avant de conclure par la description des algorithmes de LGP.

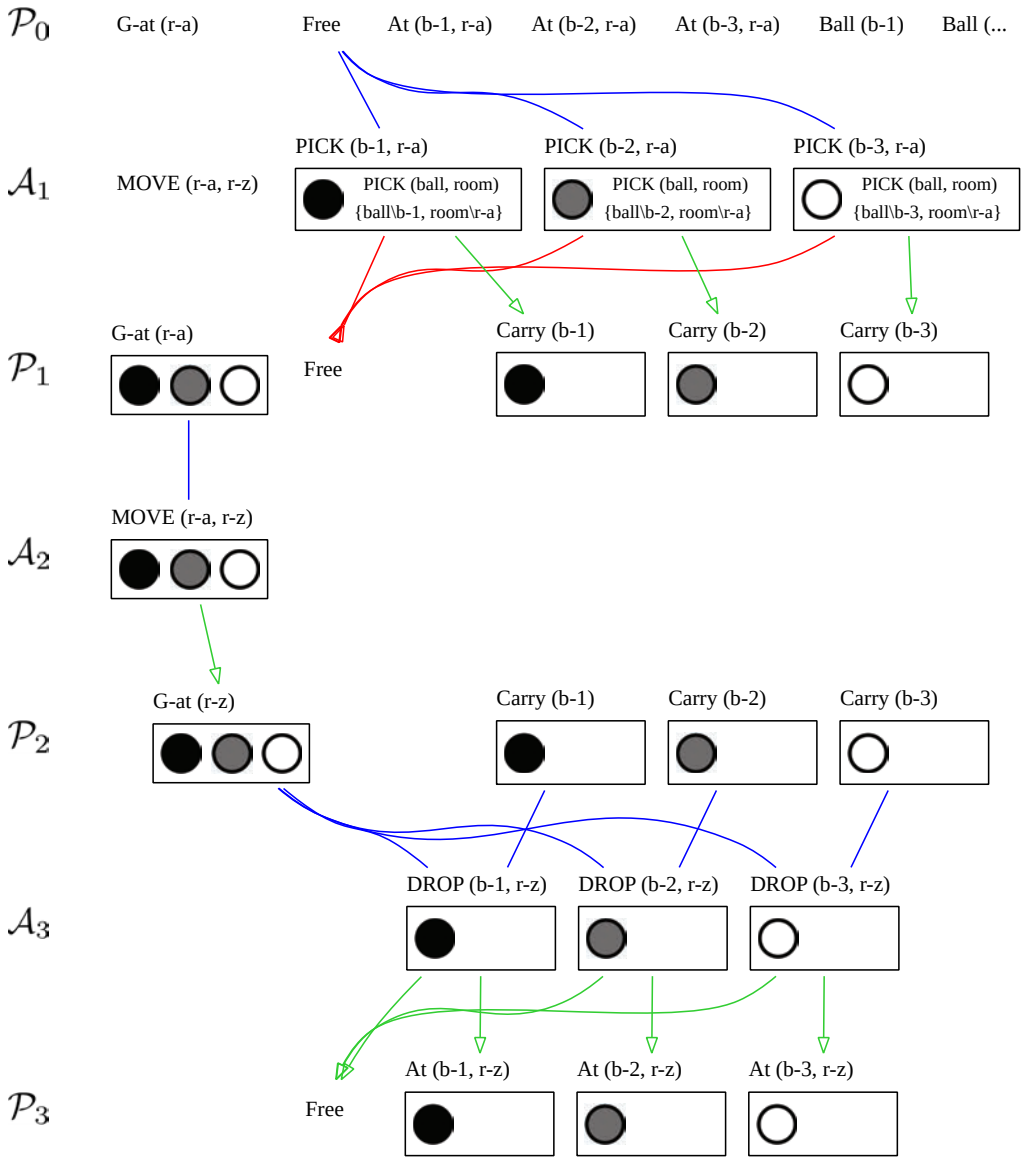


FIGURE 7.9 – Etiquetage du graphe de planification avec le domaine de planification GRIPPER durant l'extraction d'un plan relâché : l'action `MOVE(r-a, r-z)` est la fermeture interne d'un futur traitement. Cette propriété de `MOVE(r-a, r-z)` est déduite de son effet `G-AT(r-z)`, qui est une précondition commune aux actions λ-mutex secondaires `DROP`. Les étiquettes sont respectivement symbolisées par ●, ◍ et ○.

7.4.1 Cliques d'actions λ-mutex primaires

Nous faisons l'hypothèse que les actions en relation λ-mutex primaire sont candidates à l'initiation d'une structure de choix itératif. En effet, ces actions appliquent un même traitement à des objets, puisqu'elles sont déduites d'un même opérateur, mais ne peuvent s'exécuter en entrelacement, puisqu'elles sont dépendantes. Dans LGP, un traitement spécifique est toujours initié par des actions λ-mutex primaires.

Détection de clique primaire

Pour calculer une clique d'actions λ-mutex, c'est-à-dire les cliques d'actions qui vont être impliquées dans une structure de choix itératif, il suffit de prendre la première action qui ait des relations λ-mutex, de comparer les étiquettes avec lesquelles elle a été marquée. Toutes les actions qui ont des étiquettes de la même classe forment une clique d'actions par les relations λ-mutex. Pour les classes de relations λ-mutex primaires, c'est par définition qu'un ensemble d'actions étiqueté de la même classe forment une clique. Par exemple, pour GRIPPER, les actions `PICK(b-1, r-a)`, `PICK(b-2, r-a)` et `PICK(b-3, r-a)` ont toutes des étiquettes de classe `PICK(`*ball*, *room*`)`. Elles forment donc une clique d'actions λ-mutex primaires.

Abstraction et objets manipulés

Une fois l'ensemble des actions $\{\sigma_1(ope), \cdots, \sigma_n(ope)\}$ découvert par la méthode des étiquettes, calculer une action abstraite α qui représente toutes les actions de l'ensemble revient à calculer l'intersection des substitutions à appliquer sur *ope* :

$$\alpha = \sigma(ope) \text{ avec } \sigma = \bigcap_{i=1}^{n} \sigma_i$$

L'ensemble Σ des objets manipulés par la structure de choix itératif sont des substitutions qui assignent une constante à chaque variable de $var(\alpha)$ et que l'on déduit de l'ensemble des actions λ-mutex primaires. Pour chaque action $\sigma_i(ope)$, l'objet manipulé correspondant est $\sigma_i \setminus \sigma$:

$$\Sigma = \{\sigma_i \setminus \sigma \mid i \in [1..n]\}$$

L'action abstraite qui représente les actions `PICK` de GRIPPER est $\alpha =$ `PICK(`*ball*`, r-a)`, et les objets manipulés sont $\Sigma = \{(ball\backslash$`b-1`$), (ball\backslash$`b-2`$), (ball\backslash$`b-3`$)\}$.

La figure 7.10 page suivante présente deux graphes correspondant à deux problèmes de planification différents pour un même domaine de planification. Les λ-plans correspondants à l'extraction dans les graphes représentés à la figure 7.10 page suivante sont les suivants :

– pour le graphe (a) de la figure 7.10 :

choose (*ball*) in {(`b-1`), (`b-2`)}
| `PAINT(`*ball*`, yellow)`
| if-iteration `CLEAN`

– pour le graphe (b) de la figure 7.10 :

choose (*ball*, *color*) in {(`b-1`, `white`), (`b-2`, `black`)}
| `PAINT(`*ball*, *color*`)`
| if-iteration `CLEAN`

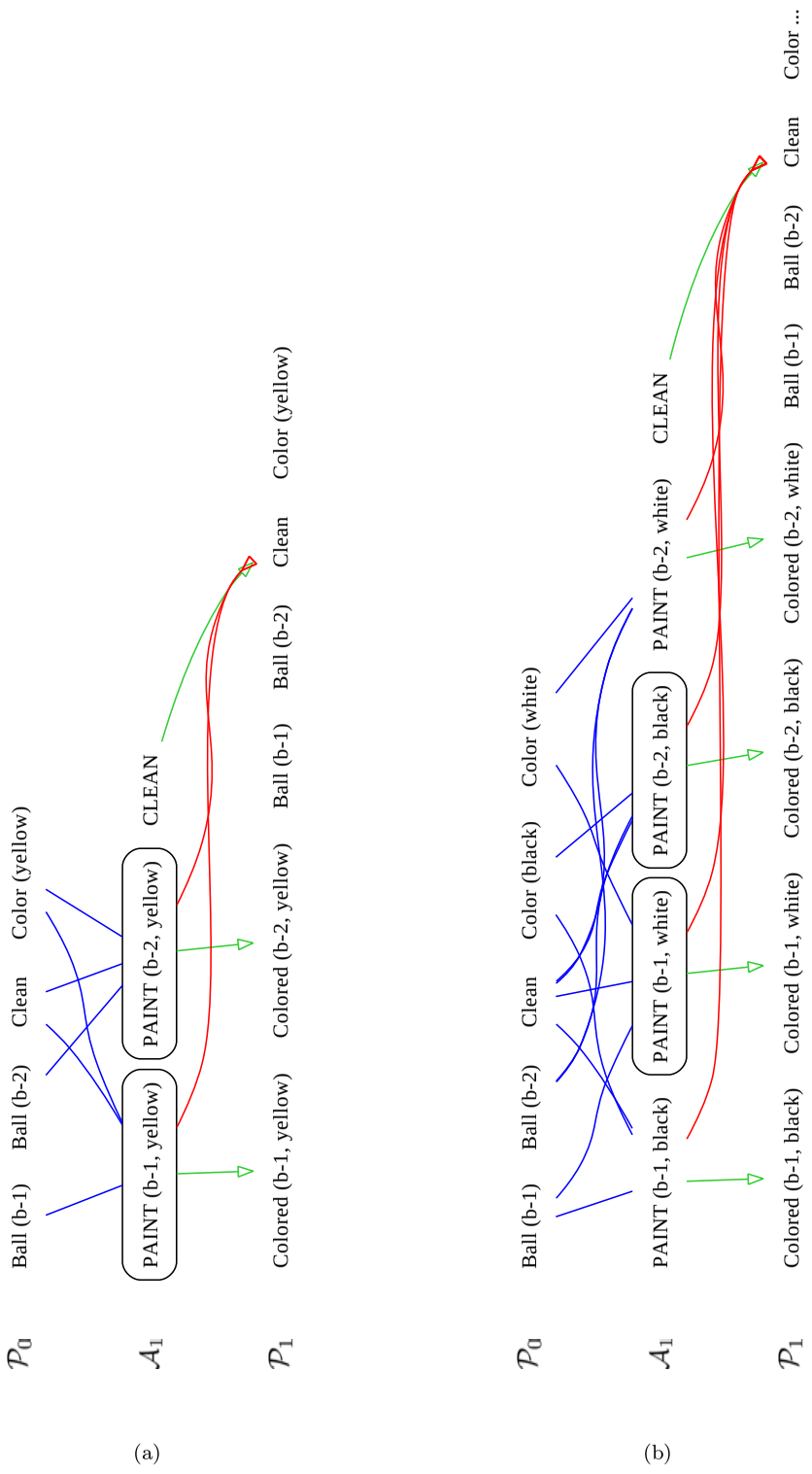


FIGURE 7.10 – Définition des objets manipulés :
(a) L'action abstraite pour représenter les deux actions encadrées est `PAINT(`*ball*`, yellow)` pour les objets $\{(\texttt{b-1}), (\texttt{b-2})\}$;
(b) L'action abstraite pour représenter les deux actions encadrées est `PAINT(`*ball*`,` *color*`)` pour les objets $\{(\texttt{b-1}, \texttt{white}), (\texttt{b-2}, \texttt{black})\}$.

Nous observons ainsi que les actions abstraites et les objets manipulés diffèrent selon les propriétés des actions en relation λ-mutex primaire. Notamment les objets manipulés ne sont pas les mêmes selon les objectifs auxquels répond la structure de choix itératif. Il est de notre point de vue important d'être concis dans le choix proposé, et de ne présenter que les informations pertinentes permettant de faire le choix. Pour le premier plan, il n'y a aucune nécessité de proposer de choisir la couleur[6], seule le nom de la balle diffère. En revanche, dans le second plan la couleur est différente[7] : nous faisons le choix de présenter toutes les informations sur les objets manipulés qui pourraient permettre de les distinguer ou d'être un critère pour le choix pour le contrôleur.

Algorithme de détection

L'algorithme 7.4, `Primary`, permet de calculer l'action abstraite α à partir d'un ensemble P^{ri} d'actions λ-mutex primaires, et renvoie un échec si elle n'existe pas. Une fois α déduite de P^{ri}, `Primary` (*cf.* lignes 6–8) peut calculer l'ensemble des objets que manipule l'action abstraite α.

Algorithme 7.4 Primary($\mathcal{S}$)

```
1  // S = {σ1(ope), ··· , σn(ope)}
2  σ ← σ1 ;
3  for i ∈ [2..n] do
4  |  σ ← σ ∩ σi ;
5  α ← σ(ope) ;
6  if c-pre(α) ∩ c-eff⁻(α) ≠ ∅ then
7  |  Σ ← ∅ ;
8  |  for i ∈ [1..n] do
9  |  |  Σ ← Σ ∪ (σi \ σ) ;
10 |  return (α, Σ) ;
11 else return failure ;
```

7.4.2 Cliques d'actions λ-mutex secondaires

Dans une structure de choix itératif, le traitement spécifique doit être appliqué sur chaque objet de la même façon. Par exemple, dans le λ-plan de Gripper, les balles `b-1`, `b-2` et `b-3` sont toutes les trois manipulées par le même traitement `PICK(`*ball*`, r-a)` $\gg$ `MOVE(r-a, r-z)` $\gg$ `DROP(`*ball*`, r-z)`. L'action abstraite `PICK(`*ball*`, r-a)` appliquée sur les objets $\{$`b-1`, `b-2`, `b-3`$\}$ a pour effet d'attraper les balles `b-1`, `b-2` et `b-3`, c'est-à-dire de produire les propositions λ-mutex `Carry(b-1)`, `Carry(b-2)` et `Carry(b-3)`. Ces propositions λ-mutex sont des préconditions pour les actions issues de `DROP(`*ball*`, r-z)` : dans le plan relâché, les actions `DROP(b-1, r-z)`, `DROP(b-2, r-z)` et `DROP(b-3, r-z)` sont λ-mutex secondaires. La propagation de l'étiquetage est telle que l'ensemble des actions `DROP` ont une étiquette de classe `PICK(`*ball*, *room*`)`. Pour découvrir la clique d'actions λ-mutex secondaires liée au traitement en construction, `LGP` compare les classes des étiquettes des actions `DROP` avec l'action abstraite initiatrice du traitement, `PICK(`*ball*`, r-a)`. L'ensemble *v-eff*$^+$ de `PICK(`*ball*`, r-a)` contient une abstraction des propositions `Carry`, le prédicat `Carry(`*ball*`)`. Ce prédicat appartient à l'ensemble *v-eff*$^+$ de `PICK(`*ball*`, r-a)`, mais doit aussi être dans l'ensemble *v-pre* de

6. Il n'y a qu'une seule possibilité, `yellow`, dans la définition du problème.
7. Les couleurs `black` et `white` sont deux possibilités définies par le problème.

l'action abstraite déduite des actions λ-mutex secondaires `DROP` : `LGP` reproduit le lien de causalité entre les actions `PICK` et `DROP` en ajoutant au traitement l'action abstraite `DROP(`*ball*`, r-z)`. En effet, l'action `DROP(`*ball*`, r-z)` manipule bien la classe d'objets *ball*, comme l'action `PICK(`*ball*`, r-a)`.

Pour la construction du λ-plan abstrait[8] $treat(obj)$, le traitement à appliquer à l'ensemble des objets, les effets positifs λ-mutex dérivés d'un prédicat dans $v\text{-}eff^{+}(treat(obj))$[9] sont directement ou indirectement étiquetés par l'ensemble P^{ri} des actions λ-mutex primaires qui ont initiées $treat(obj)$. Ainsi, l'ensemble S^{ec} des actions qui ont pour préconditions ces effets sont λ-mutex secondaires, étiquetés par les actions de P^{ri} : il y a une causalité entre $treat(obj)$ et l'ensemble des actions de S^{ec}, et l'étiquetage indique pour chaque action de S^{ec} l'objet qu'il manipule vis à vis de P^{ri}. Les actions λ-mutex secondaires sont incluses dans $treat(obj)$ lorsque $treat(obj)$ et les actions de S^{ec} sont liés par des propositions produites et consommées. Ainsi, l'action abstraite β déduite des actions de S^{ec} est telle que :

$$v\text{-}eff^{+}(treat(obj)) \cap v\text{-}pre\,(\beta) \neq \emptyset$$

L'action abstraite β manipule les mêmes objets que $treat(obj)$, et peut être intégrée à $treat(obj)$. Pour Gripper, l'ajout de l'action β = `DROP(`*ball*`, r-z)` dans le traitement `PICK(`*ball*`, r-a)` $\gg$ `MOVE(r-a, r-z)` est justifié par le prédicat `Carry(`*ball*`)`. Une autre façon d'exprimer cela est de dire qu'une action abstraite déduite d'une clique secondaire doit être incluse dans le λ-plan qui contient l'action produisant les propositions sources de cette clique. Le robot ne peut pas poser deux balles simultanément parce qu'il ne peut pas les attraper simultanément. Cette causalité provient du fait que les premières actions font que des balles sont attrapées par le robot, propriété utilisée par les actions suivantes comme préconditions.

Les deux parties suivantes s'intéressent à la détection des cliques secondaires selon le nombre d'actions qui sont dérivées d'un même opérateur et qui sont étiquetées d'une même étiquette :

1. la première, dite « un pour un », s'intéresse au cas rencontré dans Gripper avec les actions `DROP` : pour chaque objet manipulé par $treat(obj)$, il correspond une seule action dérivée d'un opérateur donné (ici `DROP`) qui soit étiquetée par cet objet ;
2. le second cas, dit « un pour n », est rencontré notamment dans le pattern d'actions qui provoque des structures imbriquées : pour chaque objet manipulé par $treat(obj)$, il correspond un ensemble d'au moins deux actions dérivées d'un opérateur donné qui soit étiquetée par cet objet.

Dans les deux cas, il faut commencer par connaître les ensembles d'actions dont l'étiquetage correspond à la structure de choix itératif en cours de construction. Comme indiqué, pour chaque ensemble, les actions doivent être dérivées d'un même opérateur. En effet, nous voulons toujours manipuler les objets de façon identique par la construction d'un traitement spécifique commun à tous les objets. Nous notons S^{ec} l'ensemble des actions dont l'étiquette est de la classe attendue par $treat(obj)$, c'est-à-dire de la classe de la clique primaire qui a initiée $treat(obj)$. Nous notons S^{ec}_{ope} l'ensemble des actions de S^{ec} qui sont dérivées de l'opérateur *ope*. Avec $\Sigma = \{\sigma_1, \cdots, \sigma_n\}$ l'ensemble des objets manipulés par $treat(obj)$, nous notons $S^{ec}_{ope}[i]$ l'ensemble des actions de S^{ec}_{ope} qui sont étiquetées par l'objet σ_i de Σ.

8. Pour rappel, un λ-plan abstrait contient des actions abstraites, pas totalement instanciées (*cf.* paragraphe 6.5.1 page 85).
9. Pour rappel, $v\text{-}eff^{+}$ est l'ensemble des effets positifs abstraits du λ-plan considéré.

Détection d'une clique secondaire « un pour un »

Pour l'exemple GRIPPER, les ensembles $S^{ec}_{\texttt{DROP}}[i]$ ne contiennent qu'une seule action. En effet, $S^{ec}_{\texttt{DROP}} = \{\texttt{DROP(b-1, r-a)}, \texttt{DROP(b-2, r-a)}, \texttt{DROP(b-3, r-a)}\}$ et :

- $S^{ec}_{\texttt{DROP}}[\texttt{b-1}] = \{\texttt{DROP(b-1, r-a)}\}$;
- $S^{ec}_{\texttt{DROP}}[\texttt{b-2}] = \{\texttt{DROP(b-2, r-a)}\}$;
- $S^{ec}_{\texttt{DROP}}[\texttt{b-3}] = \{\texttt{DROP(b-3, r-a)}\}$.

Aussi, les actions de $S^{ec}_{\texttt{DROP}}$ forment une clique selon les relations λ-mutex. En effet, étant étiquetée secondaire, cela signifie qu'elles ont des préconditions produites par une clique d'actions λ-mutex, ce qui les rend λ-mutex. Il correspond par exemple pour chaque action de $P^{ri}_{\texttt{PICK}}$ un effet dans $\{\texttt{CARRY(b-1)}, \texttt{CARRY(b-2)}, \texttt{CARRY(b-3)}\}$ qui est une précondition pour une action de $S^{ec}_{\texttt{DROP}}$. C'est cette correspondance, matérialisée par les étiquettes, qui garantit que ce sont les actions PICK qui rendent les actions DROP λ-mutex secondaires, à travers les propositions CARRY.

Soit un ensemble S^{ec}_{ope} pour lequel à chaque objet σ_i manipulé par $treat(obj)$ correspond une unique action dans $S^{ec}_{ope}[i]$ étiquetée par l'objet. Pour chaque action b_i de S^{ec}_{ope}, il existe une action β_i telle que $b_i = \sigma_i(\beta_i)$ avec σ_i l'objet manipulé : l'action β_i est une action abstraite dont est dérivée b_i, et β_i manipule les mêmes variables que $treat(obj)$. Dans GRIPPER, les b_i sont les actions de $S^{ec}_{\texttt{DROP}}[i]$, étiquetées de la classe de sources primaires PICK$(ball, room)$ et respectivement des objets $\{(ball\backslash\texttt{b-1}, room\backslash\texttt{r-a}), (ball\backslash\texttt{b-2}, room\backslash\texttt{r-a}), (ball\texttt{b-3}\backslash,\ room\texttt{r-a}\backslash)\}$, les σ_i. Dans cet exemple, pour chaque action b_i est calculée une action abstraite β_i qui est DROP$(ball$, r-z$)$.

Déduction des objets manipulés

Par définition, les β_i déduites des actions $b_i \in S^{ec}_{ope}$ sont dérivées d'un même opérateur et manipulent les mêmes variables. En revanche, nous ne pouvons pas en déduire qu'elles sont égales. Considérons que les actions DROP de l'exemple précédent utilisent finalement les couleurs black, gray et white des balles b-1, b-2 et b-3 pour les distinguer. Les actions b_i sont alors :

- $\{\texttt{DROP(b-1, black, r-a)}, \texttt{DROP(b-2, gray, r-a)}, \texttt{DROP(b-3, white, r-z)}\}$.

À partir de ces b_i et des objets σ_i utilisés précédemment, les β_i deviennent :

- $\{\texttt{DROP}(ball\texttt{, black, r-a)}, \texttt{DROP}(ball\texttt{, gray, r-a)}, \texttt{DROP}(ball\texttt{, white, r-z)}\}$.

Dans cet exemple, les β_i sont différents. Une représentation commune à toutes les actions b_i est dans ce cas l'action abstraite DROP$(ball$, $color$, r-z$)$. Ainsi, il est possible que les objets manipulés par $treat(obj)$ n'aient pas encore été totalement définis : le calcul des β_i précise une dimension des objets, c'est-à-dire étend la classe dont sont dérivées les objets (*cf.* paragraphe 6.5.1). Dans notre cas, la dimension *color* des objets n'est pas révélée par les actions PICK mais par les actions DROP.

L'ensemble des objets manipulés par $treat(obj)$ est recalculé à partir de S^{ec}_{ope} par la découverte d'une généralisation β des actions β_i. L'ensemble d'actions λ-mutex secondaire S^{ec}_{ope} est $\{b_1, \cdots, b_n\}$, l'ensemble des objets manipulés par $treat(obj)$ est $\{\sigma_1, \cdots, \sigma_n\}$ tel que l'action b_i est étiqueté par l'objet σ_i, et la substitution σ'_i est telle que $\sigma_i \subseteq \sigma'_i$ et $\forall i \in [1..n], b_i = \sigma'_i(ope) = \sigma_i(\beta_i)$. Avec l'exemple des balles colorées, la substitution σ'_1 pour la balle b-1 est $(ball\backslash\texttt{b-1}, color\backslash\texttt{black}, room\backslash\texttt{r-z})$. Alors :

$$\beta = \sigma'(ope) \text{ avec } \sigma' = \bigcap_{i=1}^{n} \sigma'_i \text{ est la généralisation des } \beta_i$$

L'ensemble des substitution σ'_i permettent de calculer l'action $\beta =$ DROP$(ball$, $color$, r-a$)$. Le nouvel ensemble Σ' des objets manipulés par $treat(obj)$ est composé de toutes les substitutions de Σ auxquelles les nouvelles dimensions de l'objet considéré sont rajoutées :

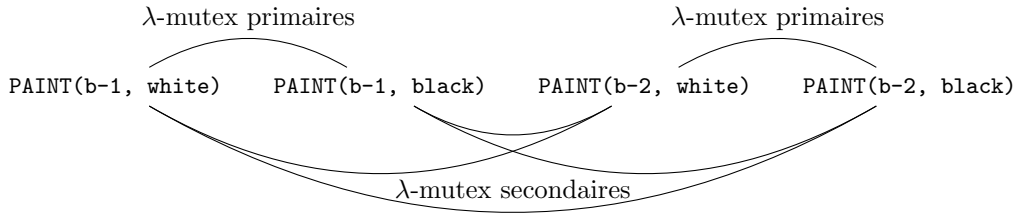
$$\Sigma' = \{\cdots, \sigma_i \cup (\sigma'_i \setminus \sigma'), \cdots\} \text{ avec } \sigma_i, \sigma'_i \text{ et } \sigma' \text{ les substitutions définies ci-avant}$$

Le nouvel ensemble des objets est $\Sigma' = \{(ball\backslash$`b-1`, $color\backslash$`black`), ($ball\backslash$`b-2`, $color\backslash$`gray`), ($ball\backslash$`b-3`, $color\backslash$`white`)$\}$. Nous notons `Merge` la procedure qui permet de calculer les nouvelles dimensions des objets à partir des deux ensembles (le Σ initial et l'ensemble déduit de β).

Détection de clique secondaire « un pour n »

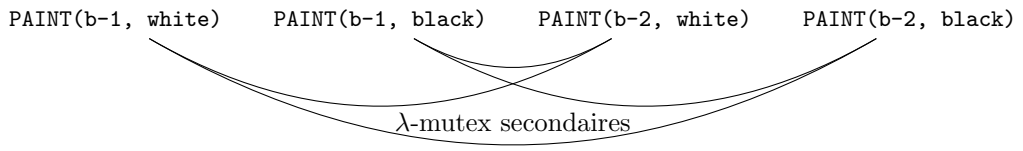
Pour l'exemple PAINTER où chaque balle doit être peinte de deux couleurs différentes, les actions `WASH` sur chaque balle `b-1`, `b-2` et `b-3` sont λ-mutex primaires. Elles rendent λ-mutex secondaires les actions `PAINT`, utilisées pour peindre chaque balle des deux couleurs `black` et `white`. Les étiquettes des actions `PAINT` ont donc pour classe `WASH(`*ball*`)`. L'ensemble des actions `PAINT` étiquetées par `WASH(`*ball*`)` est noté $S^{ec}_{\texttt{PAINT}}$. Lorsque `LGP` rencontre $S^{ec}_{\texttt{PAINT}}$ dans le plan relâché, il est en train de construire le traitement à appliquer sur chaque balle, pour l'instant composé uniquement de l'action `WASH(`*ball*`)`.

Chaque $S^{ec}_{\texttt{PAINT}}[i]$ contient deux actions étiquetées par un même objet, chacune correspondant à une des deux couleurs à appliquer sur l'objet. En particulier, $S^{ec}_{\texttt{PAINT}}[\texttt{b-1}] = \{$`PAINT(b-1, black)`, `PAINT(b-1, white)`$\}$ car les deux actions `PAINT(b-1, black)` et `PAINT(b-1, white)` sont étiquetées par le même objet ($ball\backslash$`b-1`). Dans ce domaine de planification, il n'est pas possible de colorer simultanément une même balle, et les couples des $S^{ec}_{\texttt{PAINT}}[i]$ sont λ-mutex primaires :



Les actions de $S^{ec}_{\texttt{PAINT}}$ forment une clique selon les relations λ-mutex primaires et secondaires.

Imaginons un domaine de planification dans lequel les actions de $S^{ec}_{\texttt{PAINT}}[i]$ ne sont pas λ-mutex primaires : les balles peuvent être peintes simultanément. Ainsi, seules des relations λ-mutex secondaires lient les actions de $S^{ec}_{\texttt{PAINT}}$, sans former une clique. Plus précisément, chaque action de $S^{ec}_{\texttt{PAINT}}[i]$ est en relation λ-mutex secondaire avec toutes les actions de chaque $S^{ec}_{\texttt{PAINT}}[j]$, avec $i \neq j$, et aucune relation λ-mutex (primaire ou secondaire) ne lie les actions d'un même ensemble $S^{ec}_{\texttt{PAINT}}[i]$:



Nous traitons ces deux cas dans la suite : avec relations λ-mutex primaires dans $S^{ec}_{ope}[i]$ (de telle façon que S^{ec}_{ope} forme une clique) puis sans relations λ-mutex dans $S^{ec}_{ope}[i]$.

Avec clique primaire dans chaque $S^{ec}_{ope}[i]$

Si les actions `PAINT` sont λ-mutex primaires pour une balle donnée dans la pince, alors nous nous retrouvons dans le cas d'étude des itérations imbriquées : l'ensemble des actions `PAINT` forme une clique selon la relation λ-mutex, où chaque élément de $S^{ec}_{\texttt{PAINT}}[i]$ est λ-mutex

secondaire avec tous les éléments des autres sous-ensembles de $S^{ec}_{\texttt{PAINT}}$, et les éléments de chaque $S^{ec}_{\texttt{PAINT}}[i]$ sont λ-mutex primaires entre eux. Dans ce cas, une structure de choix itératif est construit pour un premier ensemble d'action λ-mutex primaire, $S^{ec}_{\texttt{PAINT}}[\texttt{b-1}]$ par exemple. Sur l'exemple de GRIPPER, nous obtenons la structure de choix itératif suivante à partir des actions λ-mutex primaires $\{\texttt{PAINT(b-1, black)}, \texttt{PAINT(b-1, white)}\}$:

```
choose (color) in {(black), (white)}
      | PAINT(b-1, color)
      | if-iteration CLEAN ≫ WASH(b-1)
```

À partir de cette structure de choix itératif, et sachant que les actions PAINT(b-1, *color*) sont étiquetées avec (WASH(*ball*), (*ball*\b-1)) à cause de leur précondition commune READY(b-1), LGP modifie la structure de choix itératif pour la rendre plus abstraite et qu'elle puisse manipuler l'ensemble des objets définis dans *treat*(*obj*) :

```
choose (color) in {(black), (white)}
      | PAINT(ball, color)
      | if-iteration CLEAN ≫ WASH(ball)
```

Ensuite, LGP vérifie que toutes les actions dérivées du traitement de tous les objets manipulés par *treat*(*obj*) se retrouvent bien dans le plan relâché. Par exemple, si la balle b-2 ne doit pas être peinte de couleur black, c'est un échec de la transformation puisque l'action PAINT(b-2, black) ne se trouvera pas dans le plan relâché. Sinon, la structure de choix itérative est ajoutée au λ-plan :

```
CLEAN ≫ choose (ball) in {(b-1), (b-2), (b-3)}
              | (WASH(ball)
              |     ≫ choose (color) in {(black), (white)}
              |           | PAINT(ball, color)
              |           | if-iteration CLEAN ≫ WASH(ball)
              | if-iteration CLEAN
```

Sans relation λ-mutex dans chaque $S^{ec}_{ope}[i]$

Lorsque les actions PAINT ne sont pas λ-mutex primaires pour une balle donnée, LGP se trouve devant un nouveau cas : l'ensemble des actions PAINT ne forme pas de clique. Cela est possible si chaque balle peut être peinte de façon simultanée des deux couleurs. Nous souhaitons dans ce cas obtenir le λ-plan suivant :

```
CLEAN ≫ choose (ball) in {(b-1), (b-2), (b-3)}
              | WASH(ball)
              |     ≫ (PAINT(ball, black) ‖ PAINT(ball, white))
              | if-iteration CLEAN
```

LGP déduit une action abstraite de chaque action $b^j_{\texttt{b-1}} \in S^{ec}_{\texttt{PAINT}}[\texttt{b-1}]$ en utilisant l'objet de l'étiquetage : pour $b^{\texttt{black}}_{\texttt{b-1}} = \texttt{PAINT(b-1, black)}$, avec l'objet (*ball*\b-1), LGP déduit l'action

$\beta^{\texttt{black}}$ = `PAINT(`*ball*`, black)`. Ainsi, pour chaque action $b_1^j \in S_{ope}^{ec}[1]$ étiquetée par l'objet σ_1, une action abstraite β^j est telle que $b_1^j = \sigma_1(\beta^j)$. L'ensemble des β^j pour $S_{\texttt{PAINT}}^{ec}$[`b-1`] est {`PAINT(`*ball*`, black)`, `PAINT(`*ball*`, white)`}.

`LGP` vérifie que toutes les actions de $S_{\texttt{PAINT}}^{ec}$ sont exactement représentées par les β^j appliqués sur les objets traités par la structure de choix itératif. Par exemple, l'action `PAINT(b-3, gray)` n'est pas dans l'ensemble des actions obtenues par l'application de `PAINT(`*ball*`, black)` et `PAINT(`*ball*`, white)` sur chacun des objets {(*ball*\`b-1`), (*ball*\`b-2`), (*ball*\`b-3`)}. Si `PAINT(b-3, gray)` est dans $S_{\texttt{PAINT}}^{ec}$, `LGP` échoue à trouver un traitement commun à tous les objets. `LGP` échoue aussi dans la transformation si, par exemple, l'action `PAINT(b-2, black)` n'est pas dans $S_{\texttt{PAINT}}^{ec}$ alors qu'elle est dérivée de `PAINT(`*ball*`, black)` appliquée sur (*ball*\`b-2`). Autrement dit, pour tous les objets $\sigma_i \in \Sigma$ manipulés par $treat(obj)$ et pour tous les β^j, `LGP` s'assure que $\forall i \sigma_i(\beta^j) \in S_{ope}^{ec}$. Si c'est le cas, c'est-à-dire si toutes les actions de S_{ope}^{ec} sont représentées par l'application des β^j sur chacun des objets à manipuler, alors `LGP` ajoute dans $treat(obj)$ les β^j en entrelacement.

7.4.3 Support interne

Durant l'extraction du plan relâché, chaque action a est étiquetée par toutes les actions λ-mutex secondaires dont a supporte les préconditions communes : dans Gripper, l'action `MOVE(r-a, r-z)` est étiquetée par toutes les actions λ-mutex secondaires `DROP` car `MOVE(r-a, r-z)` a pour effet de déplacer le robot dans la pièce `r-z`, ce qui est une précondition pour chacune des actions `DROP`. L'action a est alors le support interne d'un traitement $treat(obj)$ initié par l'ensemble P^{ri} d'actions λ-mutex primaires : P^{ri} est responsable de l'étiquetage des actions λ-mutex secondaires. Dans Gripper, les actions `PICK` sont responsables de l'étiquetage des actions `DROP`, et l'action `MOVE(r-a, r-z)` est le support interne du traitement des balles.

Seules les actions étiquetées avec toutes et seulement les étiquettes issues de P^{ri} sont ajoutées à $treat(obj)$ en tant que support interne. Ce processus est appelé la *fermeture interne* de $treat(obj)$.

Lorsque les actions sont étiquetées par une partie seulement des étiquettes issues de P^{ri}, un comportement spécifique est à prévoir pour au moins l'un des objets concernés : les actions λ-mutex secondaires qui ont provoqué l'étiquetage ne manipulent pas tous les objets manipulés par $treat(obj)$. `LGP` peut découvrir uniquement des traitements communs à tous les objets manipulés. Dans Gripper, l'action `MOVE(r-a, r-z)` est marquée lors de l'extraction par toutes les actions `DROP` du plan relâché, car `MOVE(r-a, r-z)` supporte la proposition `G-AT(r-z)`, précondition des actions `DROP(`*ball*`, r-z)` pour chaque *ball*. Ainsi, `MOVE(r-a, r-z)` sera ajoutée dans la structure de choix itératif en tant que support interne. En revanche, si l'une des balles doit être déplacée dans une pièce `r-y`, alors l'étiquetage de l'action `MOVE(r-a, r-z)` n'est plus complet, et cela entrainera un échec de la transformation.

Lorsque une action n'est pas étiquetée exclusivement par les étiquettes issues de P^{ri}, les effets de l'action sont attendus par des actions λ-mutex secondaires S_{other}^{ec} qui ne dépendent pas de P^{ri}. Les actions de S_{other}^{ec} appartiennent à un autre traitement, intégré à une autre structure de choix itératif. Cette situation demande soit de synchroniser les structures de choix itératif au niveau de cette action partagée, soit de dupliquer l'action dans chacun des traitements. `LGP` ne permet pas la synchronisation entre les structures de choix itératif, et les conséquences de la duplication d'une action n'ont pas été étudiées. Dans Gripper, si le robot a deux pinces indépendantes, permettant deux choix indépendants et en entrelacement dans deux sous-ensembles de balles. Alors nous aurions un λ-plan permettant de choisir d'un côté la balle du premier ensemble à attraper de la première pince, et de l'autre côté la balle du second ensemble à saisir avec l'autre pince. Une fois les deux pinces pleines, le robot

pourrait se déplacer dans la seconde pièce. Cela demande de la synchronisation entre les structures de choix itératif correspondant à chaque pince, ce que ne permet pas LGP.

Nous appelons InternalClosing l'algorithme permettant de réaliser la fermeture interne d'une structure de choix itératif. À partir du plan relâché π, de l'action α initiatrice de $treat(obj)$ et de l'ensemble Σ des objets manipulés, $\texttt{InternalClosing}(\pi, \alpha, \Sigma)$ retourne en entrelacement les actions permettant la fermeture interne ou une erreur le cas échéant.

7.4.4 Support externe

Dans notre exemple, le λ-plan abstrait $treat(obj)$ permet de déplacer une balle de la pièce de gauche dans celle de droite. L'ensemble des préconditions invariables $\texttt{c-}pre(treat(obj))$ contient la proposition AT-G(r-a), ce qui signifie que le robot doit être dans la pièce de gauche pour attraper l'une des balles. Or, l'un des effets invariables de $treat(obj)$ est de déplacer le robot dans la pièce de droite, autrement dit AT-G(r-a) appartient à $\texttt{c-}eff^-(treat(obj))$, et AT-G(r-z) à $\texttt{c-}eff^+(treat(obj))$. Cela signifie que $treat(obj)$ ne suffit pas pour créer une structure de choix itératif *Struct* : LGP calcule alors le *support externe* de *Struct*, un λ-plan *reiterate* qui permet les itérations. Ainsi, *Struct* intègrera $treat(obj)$ et *reiterate* :

$$Struct \leftarrow \texttt{choose } obj \texttt{ in } \Sigma \ \ (treat(obj) \ \texttt{if-iteration } reiterate)$$

Nous détaillons l'obtention par LGP du λ-plan *reiterate*.

Notons e l'état sur lequel sera appliquée *Struct* [10]. Notons e' l'état atteint par l'application des effets invariants ($\texttt{c-}eff^-$ et $\texttt{c-}eff^+$) de $treat(obj)$ sur e. Si $\texttt{c-}eff^-(treat(obj)) \cap \texttt{c-}pre(treat(obj)) \neq \emptyset$ [11], les préconditions invariables de $treat(obj)$ ne sont pas vérifiées dans e' :

$$e \xrightarrow{treat(obj)} e' \qquad \begin{array}{l}\text{avec } e' \leftarrow (e \setminus \texttt{c-}eff^-(treat(obj))) \cup \texttt{c-}eff^+(treat(obj)) \\ \text{et } \texttt{c-}pre(treat(obj)) \not\subseteq e'\end{array}$$

LGP s'assure du bon déroulement de l'enchaînement des itérations en procédant à la *fermeture externe* de *Struct* : il rend possible l'application successive de *Struct* sur chacun des objets à manipuler. Autrement dit, LGP s'assure que l'état e'' obtenu après l'application de *Struct* sur l'un des objets supporte les préconditions de l'application de *Struct* sur un nouvel objet, et ainsi de suite pour chaque objet à traiter.

En conséquence, LGP doit calculer un λ-plan pour supporter les préconditions invariables de $treat(obj)$: avec notre exemple, c'est un problème de planification dans le domaine Gripper où, d'une part $\texttt{c-}pre(treat(obj))$ est l'objectif de planification, d'autre part $e' = (e \setminus \texttt{c-}eff^-(treat(obj))) \cup \texttt{c-}eff^+(treat(obj))$ est l'état initial. Si ce problème de planification est résolu par LGP, le λ-plan solution de ce problème est *reiterate*. Avec *Struct* composé de $treat(obj)$ et *reiterate*, nous avons :

$$e \xrightarrow{treat(obj)} e' \xrightarrow{reiterate} e'' \xrightarrow{treat(obj)} \cdots$$

Dans Gripper, *reiterate* est composé d'une unique action, MOVE(r-z, r-a). Le λ-plan *reiterate* permet d'assurer la bonne exécution de chaque action dans *Struct*. Ainsi, *Struct*

10. L'état e est atteint par l'application sur l'état initial e_0 des couches qui précèdent $treat(obj)$ (et donc juste avant α) dans le λ-plan en construction.

11. Il est commun qu'un traitement annule certaines de ses préconditions invariables : dans Gripper, le traitement modifie l'emplacement du robot ce qui annule une précondition invairable du traitement des balles (G-AT(r-a)).

pourra être exécutée sans problème si l'état du monde e'' à la fin de l'application de *Struct* supporte les préconditions invariables de $treat(obj)$: *reiterate* permet d'appliquer successivement *Struct* sur tous les objets. Comme indiqué dans la description des structures de choix itératif (*cf.* paragraphe 6.5.2 sur les contraintes supplémentaires de *reiterate*), `LGP` vérifie en plus que *reiterate* est tel que $\mathit{eff}^+(reiterate) \subseteq \texttt{c-}\mathit{eff}^+(treat(obj))$ et $\mathit{eff}^-(reiterate) \subseteq \texttt{c-}\mathit{eff}^-(treat(obj))$. Dans le cas où ce critère n'est pas satisfait, un échec est retourné.

Nous appelons `ExternalClosing` l'algorithme permettant de réaliser la fermeture externe d'une structure de choix itératif. À partir du λ-plan abstrait $treat(obj)$ et de l'état e, `ExternalClosing`$(treat(obj), e)$ retourne le λ-plan permettant la fermeture externe, ou une erreur le cas échéant.

7.4.5 Algorithme de transformation

L'algorithme 7.5 décrit la transformation d'un plan relâché π obtenu par l'étape d'extraction et dont les actions sont étiquetées conformément aux causalités observées durant l'extension du graphe de planification ou découvertes durant l'extraction.

Algorithme 7.5 Transform$(\pi, state)$

```
1  Π ← ϵ // Π will be the flexible plan, composed by sequence of flexible plans
2  foreach layer ∈ π do
3  |  Λ ← ϵ // Λ is the flexible plan for the current layer
4  |  foreach set of actions that are
5  |  |  involved in a primary λ-mutex relation of a certain class do
6  |  |  if ∃ action ∈ set that is already marked in layer then return failure ;
7  |  |  (α, Σ) ← Primary(set) // algorithme 7.4 page 115 : α is the abstract
   |  |  |  action for the set, and Σ the manipulated objects
8  |  |  Struct ← IterateChoose(α, Σ, π, state) // algorithme 7.6 page 124 :
   |  |  |  Struct the iterative-choose structure that begins with α
9  |  |  Λ ← Λ ‖ Struct // check the dependency and the applicability
10 |  foreach action that is
11 |  |  not marked ∈ layer and
12 |  |  not involved in λ-mutex relation do
13 |  |  Λ ← Λ ‖ action;
14 |  |  mark in π the action as already added;
15 |  if ∃ action that is not marked in layer then
16 |  |  return failure ;
17 |  Π ← Π ≫ (Λ) ;
18 |  state ← (state \ eff⁻(Λ)) ∪ eff⁺(Λ)
19 return Π ;
```

Une fois π extrait de $\mathcal{G}$, la transformation de π se fait de façon séquentielle, et parcourant couche après couche le plan relâché extrait par λ`Extract` (*cf.* algorithme 7.2). Ainsi, l'algorithme de transformation parcourt chaque couche de π en s'assurant que toutes les actions de chaque couche ont bien été traitées. Pour cela, chaque action ajoutée dans le λ-plan en construction sera marquée (*cf.* algorithme 7.5 ligne 14 et algorithme 7.6 lignes 2 et 28).

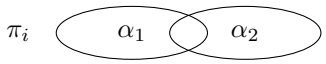
La première étape de transformation (*cf.* algorithme 7.5 ligne 1) est la création d'un λ-plan vide, qui pas à pas sera enrichi des couches successives du plan transformé. Pour

chaque couche de π (*cf.* algorithme 7.5 ligne 2), LGP fait correspondre l'entrelacement d'un ensemble d'actions et de structure de choix itératif : cela traduit la sémantique des actions appartenant à une même couche du plan extrait. Le plan solution est donc composé d'une séquence (*cf.* ligne 17 de l'algorithme 7.5) de plans Λ_i (*cf.* lignes 2—3 de l'algorithme 7.5). Pour chaque couche i, le plan Λ_i compose en entrelacement les structures itératives initiées dans cette couche et les actions sans étiquettes (*cf.* lignes 10—14). Pour chaque clique primaire d'actions dans la couche i, LGP construit une nouvelle structure de choix itératif *Struct*, et découvre par la même occasion l'ensemble des objets manipulés par *Struct* (*cf.* algorithme 7.5 lignes 4–9). L'appel à algorithme 7.6 page suivante s'occupe spécifiquement de la construction de *Struct* à partir de la découverte d'une clique primaire et d'un ensemble d'objets à manipuler (*cf.* algorithme 7.5 ligne 8). LGP intègre au traitement $treat(obj)$ de *Struct* les actions représentant les ensembles d'actions λ-mutex secondaires qui y sont liées : LGP cherche dans les couches suivantes du plan relâché ces ensembles d'actions (*cf.* algorithme 7.6 lignes 3–26). Une fois le traitement spécifique élaboré, LGP construit le λ-plan *reiterate* permettant les itérations successives (*cf.* algorithme 7.6 ligne 31), pour pouvoir finalement construire la structure finale (*cf.* algorithme 7.6 ligne 32).

Le paragraphe suivant détaille chacune de ces étapes avec une unique structure de choix itératif, sans perte de généralité. Deux paragraphes (à partir de la page 127) présentent les cas de structures imbriquées et de structures séquentielles.

Condition d'initialisation d'un traitement spécifique

Pour une action a d'une couche i donnée dans le plan π, LGP calcule l'ensemble des actions en relation λ-mutex primaire avec a. Il suffit pour cela de parcourir l'ensemble des actions a' de la couche i de π, et de vérifier que $(a, a') \in \lambda\mathcal{A}_i$ (*cf.* lignes 4 et 5 de l'algorithme 7.5). Si au moins une action de cet ensemble est marquée, LGP retourne une erreur (*cf.* ligne 6 de l'algorithme 7.5). Cela arrive lorsque des ensembles d'actions λ-mutex primaires ne sont pas disjoints :



ne peut pas être transformé

Pour chacun de ces ensembles α_i disjoints une structure de choix itératif est construite (*cf.* lignes 7–9). Cette construction implique en premier lieu la découverte de l'action représentant l'ensemble des actions λ-mutex primaires ainsi que l'ensemble des objets à manipuler (*cf.* ligne 7 de l'algorithme 7.5), mais surtout la découverte dans le plan de la totalité du traitement spécifique à appliquer à ces objets ainsi que *reiterate*. Cela signifie un parcours des couches suivantes à la recherche des actions à intégrer à la structure, ce qui est réalisé par l'algorithme 7.6, appelé à la ligne 8 de l'algorithme 7.5. Cette construction sera détaillée dans le paragraphe suivant. Une fois la structure construite, elle est ajoutée en entrelacement dans le plan (*cf.* algorithme 7.5 ligne 9), en ayant vérifié que toutes les actions qui la composent sont bien indépendantes de toutes les actions composant la couche, sans quoi un échec est retourné. Une fois tous les ensembles α_i traités (ce qui implique la possible création de structures de choix itératif en entrelacement), alors les actions qui ne sont pas marquées dans la couche sont traitées.

Toutes les actions de cette couche i qui ne sont pas déjà marquées et qui ne sont pas λ-mutex avec une autre action sont marquées et ajoutées en entrelacement dans le plan Λ_i de la couche. Cela correspond aux lignes 10 à 14 de l'algorithme 7.5.

S'il ne reste dans la couche que des actions marquées, la couche suivante peut être traitée après avoir ajouté le λ-plan Λ_i au λ-plan solution (*cf.* algorithme 7.5 ligne 17). Mais dans le cas où des actions de la couche en cours ne sont pas marquées (*cf.* lignes 15–16), alors

Algorithme 7.6 IterateChoose($\alpha, \Sigma, \pi, state$)

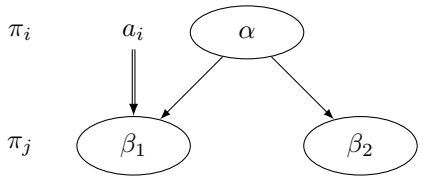
$treat \leftarrow \alpha \parallel$ `InternalClosing`(π, α, Σ) // *treat* is the specific treatment that begins with α interleaved with all actions as internal closing for further secondary λ-mutex actions (*cf.* section 7.4.3 page 120).
mark in π all the actions of *treat* as already added;
foreach *layer* $\in \pi$ *after the* α*'s layer* **do**
  $\Lambda_{treat} \leftarrow \epsilon$ // the layer under construction begins empty.
  foreach *set of actions that are derived from a certain operator and are not marked* $\in$ *layer and are labelled by* α **do**
    if *the set doesn't contain primary* λ*-mutex relations* **then**
      $(\beta, \Xi) \leftarrow$ `Secondary`(set) // β is secondary due to *treat*; Ξ is the set of object manipulated by β. The result depends on cases between 'one for one' or 'one for n' labels.
      $\Sigma \leftarrow$ `Merge`(Σ, Ξ) // the update of the objects' dimensions: Σ is the set of objects manipulated by β and by the other actions in *treat*
      $\Lambda_{treat} \leftarrow \Lambda_{treat} \parallel \beta$;
    else if *the set is a 'sequential iterations' pattern* **then**
      // the *set* will initiate another iterative-choice, and this one must be closed at this layer
      break ;
    else if *the set is a 'nested iterations' pattern* **then**
      // the *set* is primary for other objects: we must find nested iterative-choice structure
      subset $\leftarrow$ the first subset of *set* that is a clique under primary λ-mutex relations for a given *object* $\in \Sigma$;
      $(\alpha', \Sigma') \leftarrow$ `Primary`$(subset)$ // algorithme 7.4 page 115 : α' is the primary action for the *set*, and Σ' the manipulated objects
      $newStruct_{object} \leftarrow$ `IterateChoose`$(\alpha', \Sigma', \pi, state)$ // the $newStruct_{object}$ is the iterate-choose structure that only manipulates the objects of Σ' for the *object* $\in \Sigma$
      newStruct $\leftarrow$ the iterate-choose structure that manipulates the objects of Σ' for each object in Σ, deduced from $newStruct_{object}$;
      foreach *object* $\in \Sigma$ **do**
        if $\exists a \in newStruct(object) \mid a \notin \pi$ **then**
          // an action of $newStruct(object)$ is not in the initial plan
          return failure ;
      $\Lambda_{treat} \leftarrow \Lambda_{treat} \parallel newStruct$ // *newStruct* is nested in *treat*
    else return failure ;
  $\Lambda_{treat} \leftarrow \Lambda_{treat} \parallel$ `InternalClosing`(π, α, Σ) // the layer integrates all the actions without λ-mutex relation in *layer* but that are labelled by *treat*. These actions are interleaved in Λ_{treat} (*cf.* section 7.4.3 page 120).
  mark in π all the actions of Λ_{treat} as already added;
  $treat \leftarrow treat \gg \Lambda_{treat}$;
  $state \leftarrow (state \setminus eff^-(\Lambda_{treat})) \cup eff^+(\Lambda_{treat})$
$reiterate \leftarrow$ `ExternalClosing`$(treat, state)$ // *reiterate* support the precondition of *treat* for each iteration (*cf.* section 7.4.4 page 121).
return (`choose` $obj \in \Sigma\,(treat)$ `if-iterate` $(reiterate)$) // the iterative-choice structure composed by *treat* and *reiterate* that manipulates the objets of Σ

un échec est retourné. En effet, cela signifie que soit des actions λ-mutex primaires n'ont pas été traitées (ce qui dénote un problème lié à la recherche des cliques primaires), soit des actions λ-mutex secondaires appartiennent à la couche et n'ont pas été traitées à partir de couches précédentes (ce qui est normalement le rôle de l'algorithme 7.6 de construction des structures de choix itératif). Le paragraphe suivant détaille cet algorithme.

L'ensemble des actions $\{\sigma_1(ope), \cdots, \sigma_n(ope)\}$ à intégrer dans une structure de choix itératif est représenté par une action α telle que $\alpha = \sigma(ope)$ avec $\sigma = \bigcap_{i=1}^{n} \sigma_i$ et par un ensemble d'objets $\Sigma = \{\sigma_1 \setminus \sigma, \cdots, \sigma_n \setminus \sigma\}$. Les actions $\{\sigma_1(ope), \cdots, \sigma_n(ope)\}$ ne peuvent pas être déclenchées en entrelacement comme le suggère le plan relâché, donc α initiera un choix itératif sur les objets manipulés (*cf.* ligne 8 de l'algorithme 7.5 et ligne 1 de l'algorithme 7.6). Ainsi, une nouvelle structure de choix itératif est créée pour chaque ensemble d'actions λ-mutex primaires. La structure de choix itératif est construite étape par étape. Premièrement, un λ-plan séquentiel $treat(obj)$ est créé avec α pour première action : $treat(obj)$ sera le traitement spécifique à appliquer aux objets. Le rôle de l'algorithme 7.6 est principalement de parcourir la suite de π à la recherche des ensembles d'actions λ-mutex secondaires à intégrer à $treat(obj)$.

Respect de la causalité dans le traitement spécifique

Découvrir les actions λ-mutex secondaires à ajouter à $treat(obj)$ est triviale grâce à l'utilisation de l'étiquetage. Une fois l'ensemble des actions λ-mutex secondaires découvert, leur abstraction β est calculée et ajoutée au plan abstrait $treat(obj)$. Cela correspond aux lignes 5–7 de l'algorithme 7.6. `LGP` s'assure que les préconditions de l'action β sont soutenues en « interne » : le support interne est ajoutée durant le parcours des couches de π (*cf.* `InternalClosing` lignes 1 et 27). Si la fermeture interne n'est pas faite, une erreur est retournée. Cela peut arriver si les actions qui permettaient d'effectuer ce support interne ont été utilisées pour une autre structure itérative, alors que la synchronisation entre structures n'est pas traitée par `LGP`. Par la figure suivante, nous illustrons deux comportements de `LGP` (avec a_i le support interne de β_1, et α la source primaire des relations λ-mutex dans β_1 et β_2) :



est transformé en : $(a_i \parallel \alpha) \gg (\beta_1 \parallel \beta_2)$

Le premier concerne la fermeture interne calculée à la ligne 1 de l'algorithme 7.6 : l'action a_i est dans la couche de α dans π, et elle soutient les préconditions invariables de β_1. L'action a_i est ajoutée à $treat(obj)$ en entrelacement avec α comme support interne (pour β_1). Le deuxième comportement est lié à l'entrelacement de différentes actions β dans une même couche de $treat(obj)$ (*cf.* ligne 11) : dans l'exemple, les actions abstraites β_1 et β_2 sont issues d'une même couche de π, et sont donc ajoutées en entrelacement dans $treat(obj)$.

Si toutes les couches du plan relâché ont été parcourues, et que toutes les actions λ-mutex secondaires liées à $treat(obj)$ ont été ajoutées correctement, alors la fermeture externe de $treat(obj)$ peut être calculée. Comme c'est un nouveau problème de planification, il peut ne pas y avoir de solution, et une erreur est retournée dans ce cas. Sinon, le λ-plan *reiterate* obtenu permet l'enchaînement des itérations.

Création de la structure de choix itératif

Lorsque `LGP` réussit à trouver *reiterate* pour la fermeture externe de *treat*(*obj*), une structure de choix itératif est construite avec *treat*(*obj*) et *reiterate* pour traiter les objets de Σ (*cf.* ligne 32 de l'algorithme 7.6). Pour le dernier objet traité dans chaque structure de boucle, *reiterate* ne doit pas être exécutée. En effet, le plan relâché a été extrait en considérant l'application d'un unique objet : les applications sur les autres objets sont λ-mutex. Ainsi, l'état attendu après l'application de la structure de choix itératif est le résultat de l'application de *treat*(*obj*) sans *reiterate*. Dans Gripper, le plan retourné est donc le suivant :

`MOVE(r-z, r-a)`	(1)	
choose (*ball*) in {(`b-1`), (`b-2`), (`b-3`)}	(2)	
| (`PICK(`*ball*`, r-a)`	(3)	⌈
| ≫ `MOVE(r-a, r-z)`	(4)	*treat*(*obj*)
| ≫ `DROP(`*ball*`, r-z)`)	(5)	⌋
| if-iteration `MOVE(r-z, r-a)`	(6)	*reiterate*

Pour rappel, les lignes 2 à 6 forment la structure de choix itératif, composée du plan abstrait *treat*(*obj*) des lignes 3 à 5 et du plan *reiterate* de la ligne 6. La ligne 4 a permis le support interne et la ligne 6 a permis le support externe (à ne pas exécuter à la dernière itération, pour être dans l'état attendu par les actions suivantes s'il y en a). Si `LGP` réussit la fermeture de toutes les structures de choix itératif et que la dernière couche de π a été atteinte en ajoutant au λ-plan toutes les actions du plan relâché, alors c'est un succès et le λ-plan est retourné comme solution.

Nous avons déroulé l'algorithme dans le cas de structure itérative simple, permettant des structures itératives en entrelacement et en séquence. Dans les paragraphes suivants, nous allons développer l'algorithme dans le cas où les structures itératives sont imbriquées ou dans un cas spécial de structures itératives successives manipulant le même ensemble d'objets. Ces cas se rencontrent durant la construction d'un traitement pour un ensemble d'objets.

Nous avons défini la construction d'une structure itérative lorsque un ensemble d'actions est une clique uniquement selon des relations λ-mutex primaires ou uniquement selon des relations λ-mutex secondaire. Lorsque un ensemble d'actions forme une clique selon un ensemble $\mathcal{R}$ de relations λ-mutex de natures différentes, trois cas peuvent être distingués selon l'ensemble des relations λ-mutex primaires $\mathcal{R}_p$ et l'ensemble des relations secondaires $\mathcal{R}_s$:

1. le premier cas correspond à un traitement à appliquer itérativement, pour chaque objet : une nouvelle structure doit être imbriquée à la structure en construction (*cf.* lignes 15 et 16 de l'algorithme 7.6). L'ensemble des actions forme une clique selon les relations λ-mutex primaires et secondaires :

$$\forall (a_1, a_2) \in \mathcal{R}, \big((a_1, a_2) \in \mathcal{R}_s \wedge (a_1, a_2) \notin \mathcal{R}_p \big) \vee \big((a_1, a_2) \notin \mathcal{R}_s \wedge (a_1, a_2) \in \mathcal{R}_p \big)$$

2. dans le second cas, un nouveau traitement spécifique manipule les mêmes objets, et utilise les propriétés des objets modifiées par la première structure : la nouvelle structure est successive à celle en construction (*cf.* lignes 12 et 13 de l'algorithme 7.6). L'ensemble des actions forme une clique selon les relations λ-mutex secondaires, ainsi

qu'une clique selon les relations λ-mutex primaires :

$$\forall (a_1, a_2) \in \mathcal{R}, (a_1, a_2) \in \mathcal{R}_p \wedge (a_1, a_2) \in \mathcal{R}_s$$

3. les autres cas possibles ne sont pas considérés comme exploitables, car nous n'avons pas trouvé de sémantique à y rattacher.

LGP fait le choix de créer une nouvelle structure pour chaque clique primaire.

Structures de choix itératif imbriquées

Le cas 1 apparait lorsque un traitement $treat(obj)$ est appliqué à un ensemble d'objets d'une certaine classe, et qu'un nouveau traitement $treat'$ concerne une nouvelle classe d'objets spécialisant la première (ces objets incluent les objets initiaux). C'est le cas lorsque pour chaque balle de PAINTER, l'objectif est d'appliquer un certain nombre de couleurs à chaque balle. La nouvelle classe d'objets implique les balles, mais spécialise cette classe avec une dimension de couleur.

Le cas des structures imbriquées est traité par les lignes 15 à 25 de l'algorithme 7.6. En effet, c'est durant la construction d'une structure de choix itératif pour un ensemble d'objets que LGP découvre qu'une nouvelle structure doit être initiée. Cela a été décrit au paragraphe 7.4.2 page 118. LGP construit une nouvelle structure de choix itératif $newStruct_{object}$ pour la première clique d'actions λ-mutex primaires (*cf.* lignes 17 et 19 de l'algorithme 7.6) : $newStruct_{object}$ manipule un objet en particulier. LGP transforme $newStruct_{object}$ en une structure de choix itératif qui manipule tous les objets de Σ (*cf.* lignes 20), $newStruct$. LGP vérifie que toutes les actions déduites de l'application des objets sur $newStruct$ appartiennent au plan relâché (*cf.* lignes 21–24). En cas de succès, $newStruct$ est ajoutée à la couche en construction, et devient alors imbriquée dans $treat(obj)$ (*cf.* ligne 27 de l'algorithme 7.6).

Structures de choix itératif successives pour un nouvel ordre de traitement

Le cas 2 apparait lorsque un traitement est appliqué à un ensemble d'objets, ce qui rend les propriétés de cet ensemble d'objets λ-mutex secondaires, et que des actions qui manipulent ces mêmes objets ou une partie de ces objets sont λ-mutex primaires. Ces actions λ-mutex primaires initient une nouvelle structure de choix itératif pour un nouveau traitement des objets concernés[12], séquentielle à la première structure, ce qui permet de différencier l'ordre de traitement des objets. Ce cas là est traité aux lignes 12–14 de l'algorithme 7.6.

7.5 Discussion

Le paragraphe 7.5.1 aborde quelques limitations de l'exploitation par LGP des informations issues de $\mathcal{G}$, le paragraphe 7.5.2 concerne les limitations du processus de transformation.

7.5.1 Informations issues du graphe de planification

Extension du graphe de planification

Mis à part la création de relations λ-mutex et l'étiquetage des actions d'intérêt, il n'y a pas de différence entre GP et LGP pour l'extension du graphe de planification $\mathcal{G}$. Notamment, une action est ajoutée à la couche $\mathcal{A}_i$ si et seulement si ses préconditions ne sont pas mutex dans la couche $\mathcal{P}_{i-1}$. Par exemple, considérons l'action permettant de ranger une balle

12. Rien n'empêche d'intégrer des objets qui ne font pas partie de l'ensemble des objets sous relation λ-mutex secondaire, ou au contraire de ne traiter qu'un sous ensemble des objets

uniquement si cette balle a été peinte de deux couleurs différentes. Lorsque les actions pour peindre une balle donnée de deux couleurs différentes sont λ-mutex dans $\mathcal{A}_{i-1}$, les propositions qui indiquent que la balle est peinte des deux couleurs sont λ-mutex dans $\mathcal{P}_{i-1}$. L'action de ranger la balle n'apparait alors pas dans $\mathcal{A}_i$. La relaxation des relations mutex ne devrait pas s'appliquer uniquement à l'extraction du graphe : on pourrait aussi relâcher les contraintes λ-mutex à la construction du graphe, les préconditions d'une action ajoutée à $\mathcal{A}_i$ pouvant être λ-mutex dans $\mathcal{P}_{i-1}$.

Extraction du plan relâché

Considérons un domaine de planification Gripper étendu, où les balles, en plus d'être transportées dans une autre pièce, doivent être peintes de différentes couleurs. L'objectif de planification est donc composé d'une part de la position des balles dans la pièce voisine, et d'autre part de la coloration des balles. Le plan flexible attendu sera de la forme suivante :

choose (*ball*) in {(`b-1`), (`b-2`), (`b-3`)}
| `PICK`(*ball*, `r-a`)
| ≫ `MOVE(r-a, r-z)` ‖ choose (*color*) in {(`black`), (`white`)}
| | `PAINT`(*ball*, *color*)
| | if-iteration `WASH`(*ball*)
| ≫ `DROP`(*ball*, `r-z`)
| if-iteration `MOVE(r-z, r-a)`

Dans le graphe de planification correspondant (*cf.* figure 7.11), les deux parts de l'objectif forment deux cliques selon les relations λ-mutex. En effet, les propositions `At(r-z, b-1)`, `At(r-z, b-2)` et `At(r-z, b-3)` sont λ-mutex dans $\mathcal{P}_3$ du fait des actions `DROP(b-1, r-z)`, `DROP(b-2, r-z)` et `DROP(b-3, r-z)`. Il en est de même pour toutes les propositions `Colored` du fait des actions `PAINT` appliquée sur chaque balle et chaque couleur. Mais ces cliques sont en partie mutex entre elles (par exemple, une balle positionnée dans `r-z` est mutex avec une autre balle peinte en `black` et `white`). Par exemple, dans $\mathcal{A}_3$, les actions `DROP(b-2, r-z)` et `DROP(b-3, r-z)` sont mutex avec les actions `PAINT(b-1, black)` et `PAINT(b-1, white)` (ainsi qu'avec les actions virtuelles `NOOP-Colored` appliquées sur `b-1` pour chaque couleur), car leurs préconditions respectives `Carry` et `Colored` sont mutex dans $\mathcal{P}_2$; le raisonnement est le même en remontant dans $\mathcal{A}_2$ pour les actions `PICK` (et `NOOP-Carry`) appliquées sur `b-2` et `b-3` avec les actions `PAINT` appliquées sur `b-1` qui rendent ces préconditions mutex : les relations mutex entre les actions `PICK` (et `NOOP-Carry`) et `PAINT` sont issues des relations λ-mutex entre les propositions `Carry` de $\mathcal{P}_1$. L'extraction ne pourra donc pas débuter, alors que les deux cliques secondaires ont une même source primaire (les actions `PICK`).

La relaxation des relations mutex pourrait être plus grande que dans `LGP`, en considérant non plus les relations elles-mêmes, mais directement les étiquettes associées aux propositions et aux actions. Dans ce cas, les éléments du graphe, propositions ou actions, qui ne peuvent pas être sélectionnés ensemble dans une même couche deviennent les éléments qui sont mutex et qui n'ont pas d'étiquette de même classe. Cela inclut toutes les relations λ-mutex, et rajoute en plus des éléments qui ne peuvent pas être λ-mutex par définition de ces relations.

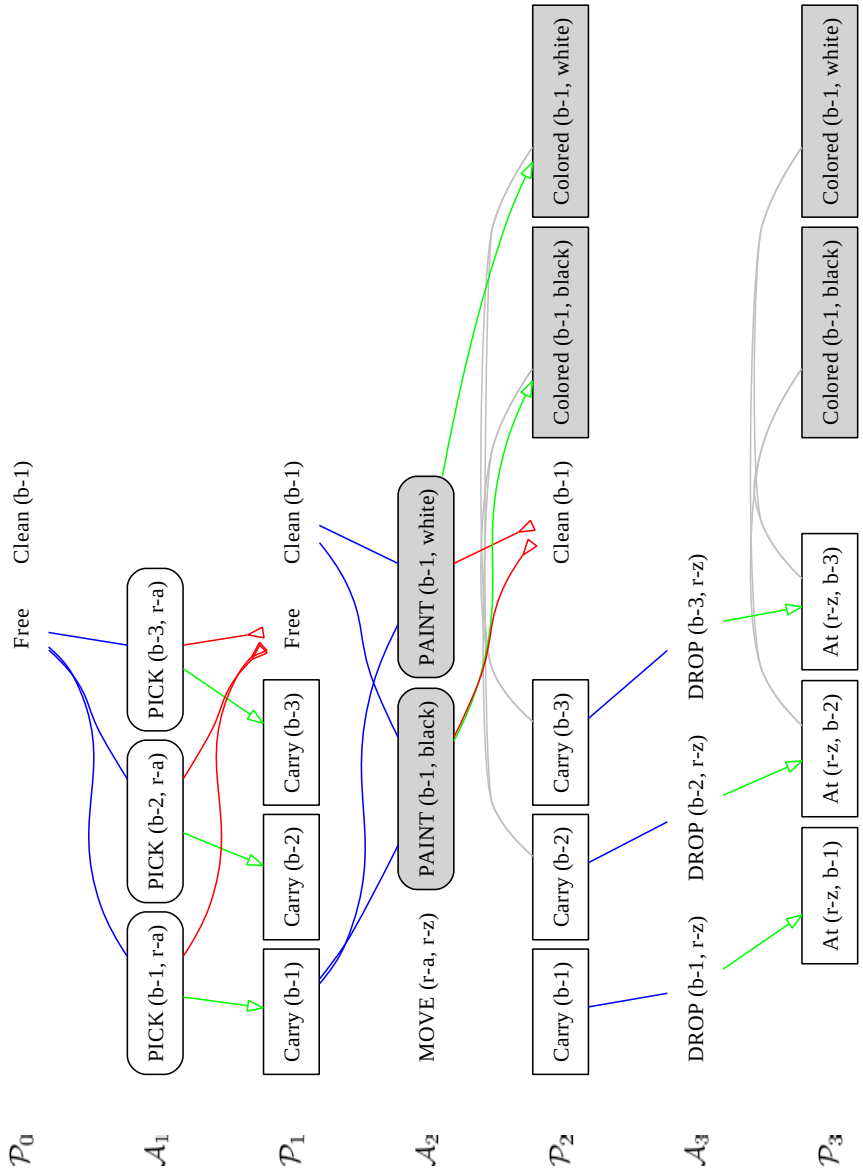


FIGURE 7.11 – Graphe de planification pour le problème consistant à obtenir les trois balles `b-1`, `b-2` et `b-3` dans l'emplacement `r-z` et chacune des balles peintes des couleurs `black` et `white`. Seule la coloration de la balle `b-1` est représentée dans cette figure. Les propositions encadrées en clair et en foncé forment séparément des cliques selon la relation λ-mutex. Les relations mutex entre propositions sont représentées par un trait les reliant. Les actions entourées sont des sources primaires de relations λ-mutex.

7.5.2 Processus de transformation

Recherche de clique primaire

Les actions qui nous intéressent pour apporter la flexibilité recherchée sont toutes λ-mutex entre elles. Autrement dit, elles forment des cliques selon la relation λ-mutex. Pour un ensemble d'actions $\mathcal{C}$, nous avons par définition d'une clique :

$$\forall\,(a_i, a_j) \in \mathcal{C} \times \mathcal{C}, a_i \neq a_j \Rightarrow (a_i, a_j) \in \lambda\mathcal{A}_k$$

par la définition des actions a_i et a_j, soit l'opérateur *ope* et les substitutions σ_i et σ_j :

$$\forall\,(\sigma_i(ope), \sigma_j(ope)) \in \mathcal{C} \times \mathcal{C}, \sigma_i \neq \sigma_j \Rightarrow (\sigma_i(ope), \sigma_j(ope)) \in \lambda\mathcal{A}_k$$

par définition des relations λ-mutex primaires, soit la substitution σ :

$$\forall\,(\sigma_i(ope), \sigma_j(ope)) \in \mathcal{C} \times \mathcal{C}, \sigma_i \neq \sigma_j, \sigma = \sigma_i \cap \sigma_j \Rightarrow \mathit{pre}\,(\sigma(ope)) \cap \mathit{eff}^-(\sigma(ope)) \neq \emptyset$$

Or, nous simplifions dans notre approche la condition d'obtention d'une clique d'actions $\{a_1, \cdots, a_n\}$ selon la relation λ-mutex primaire, avec $\forall i \in [1..n], a_i = \sigma_i(ope)$:

$$\sigma = \bigcap_{i=1}^{n} \sigma_i \Rightarrow \mathit{pre}\,(\sigma(ope)) \cap \mathit{eff}^-(\sigma(ope)) \neq \emptyset$$

Cette simplification ne prend pas en compte toutes les cliques possibles : notre condition implique la condition générale d'obtention d'une clique, mais l'inverse n'est pas vrai. Par exemple, considérons l'opérateur `OPE`(x, y, z) avec pour préconditions et effets négatifs :

- $\mathit{pre}\,($`OPE`$) = \{$`P`(x), `Q`(y), `R`$(z)\}$;
- $\mathit{eff}^-($`OPE`$) = \{$`P`(x), `Q`(y), `R`$(z)\}$.

Soient les actions abstraites α_X`(x, c, c)`, α_Y`(c, y, c)` et α_Z`(c, c, z)` dérivées de `OPE`, avec `c` une constante de $\mathscr{C}$. Ces actions abstraites sont trois classes de sources primaires. Mais en plus, toutes les actions dérivées de α_X, de α_Y et α_Z sont λ-mutex entre elles et forment une clique, car α_X, de α_Y et α_Z sont elles-mêmes en relation λ-mutex. Dans ce cas précis, la représentation commune entre ces actions est `OPE`, et la seule structure qui permette d'exécuter toutes ces actions demande de considérer des triplets comme objets manipulés (de classe (x, y, z)). Comme ce type de clique nous semble complexe à détecter, nous limitons les cliques formées par une seule « classe » de sources primaires déduite de l'opérateur considéré, c'est-à-dire les ensembles d'actions ayant des relations λ-mutex dérivées d'une seule et unique classe. Ainsi, lors de l'extraction d'un plan relâché π composé d'actions formant de telles cliques, `LGP` échoue sur la condition simplifiée d'obtention d'une clique et ne transforme pas π en λ-plan.

Fermeture externe

`LGP` évite d'annuler des effets attendus du traitement $\mathit{treat}(obj)$ en interdisant au support externe *reiterate* de manipuler les propriétés des objets traités par la structure de choix itératif. Or, dans certaines situations, le fait de modifier des effets du traitement est souhaitable. Par exemple, en reprenant le domaine de planification Gripper étendu avec pour seul objectif de peindre les balles, le plan flexible solution est :

choose (*ball*) in {(b-1), (b-2), (b-3)}
| PICK(*ball*, r-a)
| ≫ choose (*color*) in {(black), (white)}
| | PAINT(*ball*, *color*)
| | if-iteration WASH(*ball*)
| if-iteration DROP(*ball*, r-a)

Ce plan flexible nécessite de déposer chacune des balles après les avoir peintes : l'action DROP(*ball*, r-a), *reiterate*, modifie des propriétés des balles. LGP ne possède pas d'éléments dans le graphe de planification permettant de distinguer les effets nécessaires à la résolution de l'objectif des autres. Les travaux sur la symétrie [FL99] dans les domaines de planification pourraient apporter quelques éléments de réponses : les actions *symétriques* présentent un intérêt équivalent pour atteindre l'objectif de planification. Les informations qui sont déduites de l'objectif de planification pourraient être utilisées dans notre cas pour découvrir les effets des structures de choix itératif nécessaires au bon déroulement du plan flexible.

Transformation du plan relâché

Dans notre approche, l'extraction précède la transformation. Ce choix de conception a ses limites. Notamment, lorsque la transformation échoue, LGP ne peut pas reprendre l'extraction pour tenter la transformation sur un autre plan relâché [13]. LGP ne capitalise pas d'informations [14] sur son échec et sur les raisons de cet échec durant la transformation. Nous avons fait le choix d'une unique tentative de transformation du premier plan relâché extrait par LGP, suivie par une résolution de type GraphPlan à la suite d'un échec. N'importe quel autre algorithme de planification aurait pu prendre la suite de la résolution. Nous considérons donc que LGP ne trouvera pas de solution flexible dès son premier échec de transformation.

7.6 Conclusion

LGP est le premier algorithme de planification pour des domaines de planification déterministes permettant d'exprimer la permutation de traitements sur une classe d'objets. L'idée directrice de LGP est l'exploitation de la représentation condensée des états dans le graphe de planification, et notamment d'informations supplémentaires aux relations mutex : les actions λ-mutex primaires sont considérées comme des actions susceptibles d'initier des structures de choix itératif. Autour de cette idée, la matérialisation des causalités dans le graphe de planification par l'étiquetage des actions et des propositions facilite la transformation. En effet, l'étiquetage permet une détection aisée [15] des cliques d'actions par l'exploitation des définitions des relations λ-mutex primaires et secondaires. En outre, l'étiquetage rend compte des liens de causalité entre les actions qui manipulent les mêmes objets, et qui forment ainsi un traitement spécifique à ces objets. Enfin, l'étiquetage des actions durant l'extraction du plan relâché permet, durant la construction du traitement,

13. L'extraction dans LGP est un processus intégrant des choix non déterministes (*cf.* algorithme 7.3 page 110), et plusieurs plans relâchés sont potentiellement extractibles d'un graphe de planification donné.
14. Nous pourrions imaginer un processus similaire aux tables des *nogood* de GraphPlan.
15. Les algorithmes Primary (*cf.* paragraphe 7.4.1) et Secondary (*cf.* paragraphe 7.4.2) ont une complexité linéaire en fonction du nombre d'actions, par l'utilisation des classes des étiquettes.

d'ajouter « couche après couche » la fermeture interne, c'est-à-dire d'anticiper les attentes des actions prochainement intégrées au traitement.

Chapitre 8

Évaluation expérimentale de Λ-GraphPlan

Nous nous inspirons des critères d'évaluation [SIZ10] de la planification généralisée pour définir ceux de la planification flexible. Ainsi, nous proposons de comparer :

- la *couverture* de la flexibilité, évaluée par les opérateurs que chaque planificateur utilise dans ses plans, comme la séquence, la permutation et l'entrelacement pour LGP, et, pour un problème donné, par la largeur des plans flexibles comparée au nombre de plans solutions [1] du problème.
- la *complexité* du calcul du plan flexible, pour comparer les performances des planificateurs, en temps de calcul.

Nous n'avons pas trouvé dans la littérature de planificateurs flexibles avec lesquels nous pourrions comparer la couverture de la flexibilité. Seuls les travaux proposant un moindre engagement durant la planification, comme la planification partiellement ordonnée ou GP, produisent des plans potentiellement flexibles. Mais leur comparaison n'apporterait aucun crédit à LGP : ces planificateurs n'ont pas été conçus pour être flexible mais principalement pour accélérer les temps de calcul. Aucun planificateur de notre connaissance ne propose de permutation comme nous le faisons avec les structures de choix itératifs dans LGP, et aucun planificateur classique ne propose d'opérateurs de composition plus expressifs que ceux de LGP.

La plupart des propositions en planification classique s'intéresse à de nouvelles heuristiques ou approches permettant d'accélérer le calcul des plans solutions : le critère de complexité est primordial. Les compétitions de planification classent régulièrement les concourants selon leurs performances en temps de calcul. En ce qui concerne la planification flexible, il y a plusieurs façons de l'interpréter. Nous proposons :

- d'une part d'observer le comportement des planificateurs avec des problèmes de différentes difficultés (par exemple, évaluer l'évolution du temps de calcul lorsque le nombre de balles dans Gripper augmente) ;
- d'autre part de prendre en compte la flexibilité des plans calculés en rapportant le temps de calcul pour un problème donné à la largeur du plan flexible calculé, de façon à comparer l'évolution de la flexibilité et celle du temps de calcul : nous obtenons alors un temps « normalisé ».

Dans ce chapitre, les évaluations ont deux principaux objectifs. Le premier est d'observer le comportement de LGP sur un ensemble de problèmes dont les caractéristiques varient (par exemple le nombre d'objets à traiter, le nombre d'actions dans le traitement ou le type de structure de choix itératif). Le second objectif est de comparer les performances de LGP avec celles de deux planificateurs :

1. La largeur des plans est en général potentiellement infinie, puisque la plupart des domaines de planification présentent des cycles : par exemple, le domaine Gripper permet au robot d'effectuer un nombre infini d'allers et retours dans les différentes pièces. La couverture de la flexibilité par un plan pourrait s'évaluer vis à vis d'un plan flexible « étalon » qui permet d'atteindre l'objectif par l'ensemble des plans solutions optimaux, c'est-à-dire les plans solutions qui permettent d'atteindre l'objectif en un minimum d'actions, ou selon les plans solutions respectant une caractéristique à définir.

– FF [HN01], généralement très performant [2] ;
– GP [BF97] sur lequel est fondé LGP [3].

LGP est développé en java [4], et utilise la bibliothèque pddl4j [5]. Il est exécuté sur un MacBookPro 2.6 GHz Intel Core 2 Duo processor, 4 Go of 667 MHz DDR2. L'implémentation n'a pas été optimisée [6] : notre préoccupation fut l'expressivité des plans avant la performance de calcul. Nous proposons une évaluation des performances de LGP en plusieurs étapes :

1. une évaluation préliminaire sur deux exemples élémentaires, uniquement pour exposer les premières caractéristiques de LGP ;
2. une évaluation des performances sur les cas d'études utilisés dans ce manuscrit ;
3. l'utilisation de domaines « linéaires », c'est-à-dire ne permettant pas l'intégration de structures de choix itératif (et donc non favorables à LGP vis à vis de GP).

8.1 Évaluations préliminaires

Dans ce paragraphe, nous présentons les expérimentations sur deux domaines simples, qui permettent d'observer le comportement de LGP lorsque les λ-plans sont composés d'une structure de choix itératif telle que :

1. le traitement comporte une seule action abstraite et la fermeture externe nécessite une unique action, avec le domaine Checker (*cf.* annexe B.1) ;
2. le traitement est composé de deux actions abstraites (une primaire, une secondaire), sans fermeture externe, avec le domaine Loader (*cf.* annexe B.2).

Le motif de ces expérimentations est l'observation du comportement de LGP quant à la construction de telles structures de choix itératif.

8.1.1 Expérimentations sur Checker

Le premier domaine utilisé pour l'évaluation de LGP est appelé Checker (*cf.* annexe B.1) : l'objectif est de *vérifier* le poids des balles, en faisant la *tare* de l'appareil utilisé avant chaque pesée. Pour trois balles b-1, b-2 et b-3, le λ-plan calculé par LGP est le suivant :

choose (*ball*) in {b-1, b-2, b-3}
| CHECK(*ball*)
| if-iteration TARE

Ce λ-plan permet d'obtenir à l'exécution un plan optimal, c'est-à-dire le plus court possible en terme de nombre d'actions à exécuter pour atteindre l'objectif. Le λ-plan a une largeur de 6 : le nombre de permutations possibles du traitement des 3 balles. En considérant uniquement les plans optimaux solutions de ce problème, ce λ-plan est le plus expressif possible : tous ces plans sont exécutables à partir du λ-plan proposé par LGP.

Tous les planificateurs de notre connaissance calculent pour le problème Checker un plan qui correspondra à l'une des 6 permutations, par exemple :

2. Les performances de FF sont comparées dans les compétitions internationales de planification, comme IPC [LFS+02] : il expose d'excellentes performances, notamment pour les domaines déterministes.
3. Le code de LGP est fondé sur le code de GP donné en exemple d'utilisation de pddl4j [5].
4. L'implémentation utilisée de LGP est disponible à l'url http://membres-liglab.imag.fr/martin/documents/PhD/LGP.zip.
5. pddl4j est proposé par Damien Pellier, à l'url http://sourceforge.net/projects/pdd4j/.
6. Notamment, la gestion de la mémoire cause des irrégularités lors de l'exécution de l'algorithme.

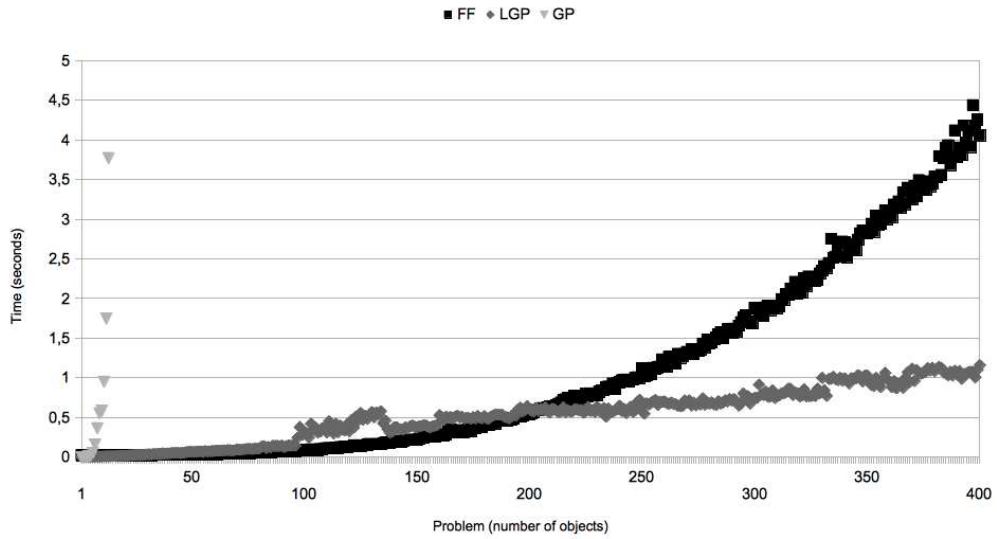


FIGURE 8.1 – La comparaison des temps de planification (en seconde) pour chaque problème issu de CHECKER (en nombre de balles, de 1 à 400) pour les planificateurs FF, LGP et GP.

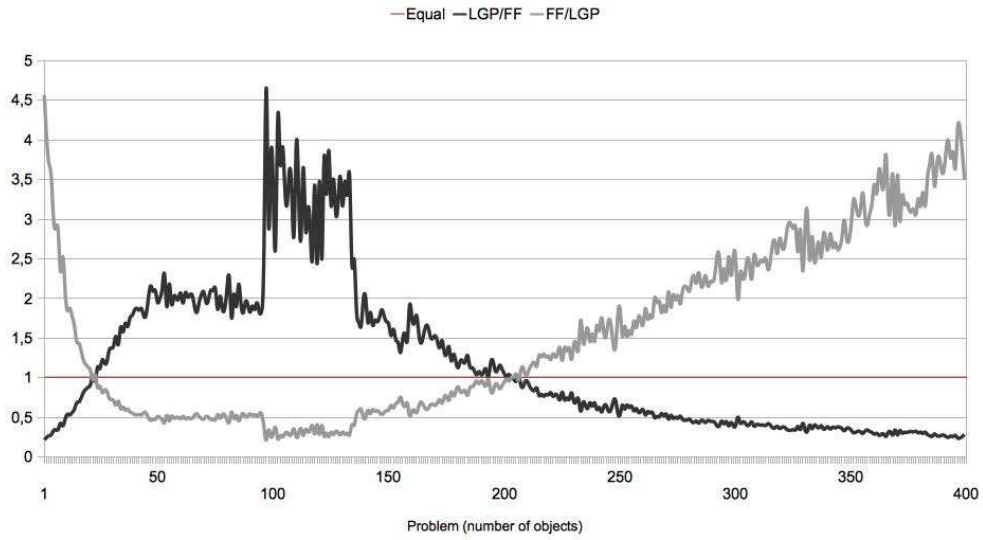
```
CHECK(b-3) ≫ TARE ≫ CHECK(b-2) ≫ TARE ≫ CHECK(b-1)
```

Sur ce problème particulier, nous observons qu'en terme de nombre d'actions, les plans calculés par FF et GP sont aussi des plans optimaux : ce critère ne permet pas de différencier la qualité des plans calculés par chacun des planificateurs. Ce sera le cas pour tous les exemples de ce chapitre. En revanche, le plan calculé par LGP est 6 fois plus large que ceux des deux autres planificateurs : les plans de FF et de GP ne sont pas flexibles. D'après nos critères d'évaluation de l'expressivité, cette première comparaison qualitative des plans est favorable à LGP. Dans la suite, les λ-plans étant « au pire » autant expressif que les plans de GP et a fortiori que ceux de FF, nous ne détaillerons pas davantage la flexibilité des λ-plans calculés.

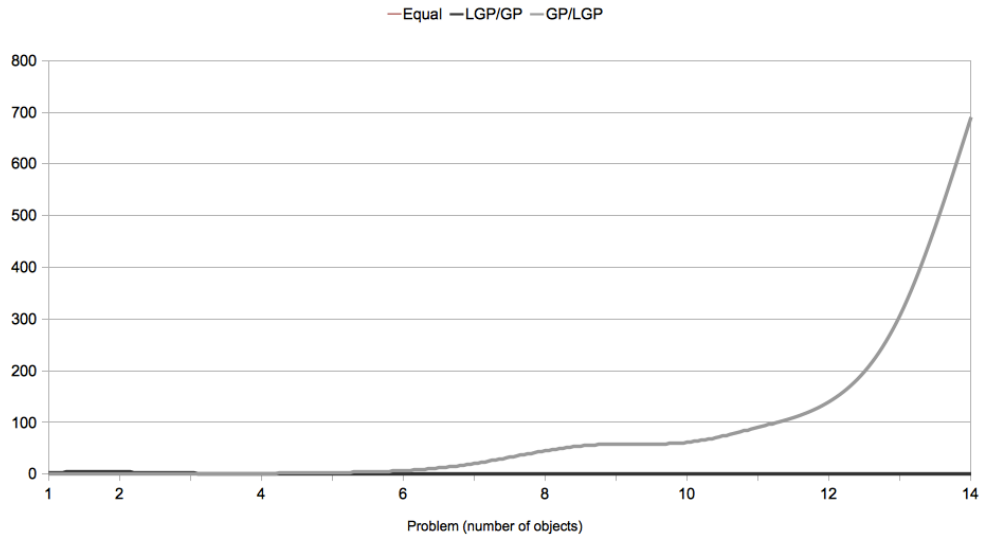
Comparaison entre planificateurs

La comparaison du temps de planification est proposée à la figure 8.1 pour les problèmes issus de CHECKER avec un nombre de balles variant de 1 à 400. Les courbes du temps de calcul de FF et de GP présentent respectivement un profil polynomial et exponentiel en fonction du nombre de balles à traiter, tandis que celle de LGP a un profil linéaire. Les figures 8.2(a) et 8.2(b) proposent les temps de planification de FF et de GP, relativement au temps de planification de LGP pour ces problèmes. FF est plus rapide[7] que LGP pour les problèmes situés entre 20 et 180 balles environ. Mais la figure 8.2(a) confirme que le temps de planification de FF croît plus rapidement que celui de LGP : le rapport des temps de planification augmente (linéairement) lorsque le nombre de balles augmente. Ainsi, LGP rattrape rapidement les performances de FF et les courbes se croisent à l'avantage de LGP lorsque le problème contient environ 180 balles.

7. FF est plus de 5 fois plus rapide que LGP pour le problème à 85 balles, représenté par le pic de la courbe LGP/FF de la figure 8.2(a).



(a)



(b)

FIGURE 8.2 – Le rapport entre les temps de planification (en seconde) pour chaque problème issu de CHECKER (en nombre de balles, de 1 à 400) :
(a) entre les planificateurs `FF` et `LGP` ;
(b) entre les planificateurs `GP` et `LGP` (jusqu'à 14 balles).

Pour LGP, la structure de choix itératif est toujours la même, seul l'ensemble des objets à traiter diffère. Ainsi, le graphe de planification est étendu du même nombre de couches pour chaque problème : l'objectif de planification est λ-mutex dans la même couche propositionnelle, quel que soit le problème de CHECKER traité. La simplicité de CHECKER fait que les temps d'extension du graphe et d'extraction du plan relâché augmentent linéairement [8]. Cela explique les excellentes performances de LGP sur ces problèmes. A contrario GP étend un graphe qui contient $2n-1$ couches [9] avec n le nombre de balles. Pour chaque problème, GP tente d'extraire un plan à partir d'une même couche [10]. Or, les performances de GP sont fortement liées au nombre d'extractions tentées dans son graphe [GNT04] : les performances diminuent fortement lorsque la difficulté des problèmes augmente.

Observation de LGP

Le temps de planification est composé des temps des processus :
- de construction du graphe ;
- d'extraction du plan relâché ;
- de transformation du plan relâché en λ-plan, lui même composé notamment du temps de fermeture externe des structures de choix itératif.

Ainsi, la fermeture externe fait partie de la transformation, et la transformation fait partie de la planification. Nous proposons à la figure 8.3 le détail de l'évolution des temps de planification, de transformation du plan relâché et de fermeture externe des structures de choix itératif. Quelques piques sont dus à la gestion de la mémoire, et à partir de 275 balles environ le comportement contient de fortes irrégularités. Mais globalement, nous observons que les temps de fermeture externe, de transformation et de planification ont des évolutions linéaires en fonction du nombre de balles. Les figures 8.4(a) et 8.4(b) se concentrent sur les comportements de LGP suffisamment réguliers pour être interprétés : elles limitent les observations aux problèmes 2 à 80.

La transformation nécessite de construire toujours le même traitement, mais pour un nombre d'objets variants. La difficulté majeure réside donc dans la découverte des objets, c'est-à-dire des cliques d'actions CHECK. La figure 8.4(a) montre que dans LGP, la découverte de ce type de clique se fait en temps linéaire du nombre d'objets. Par l'utilisation des étiquettes, il suffit d'un parcours des actions concernées.

La figure 8.4(b) expose les rapports entre les processus : la part de la transformation dans la planification, et la part de la fermeture externe dans la transformation. La fermeture externe de la structure de choix itératif est formée uniquement de l'action TARE, quel que soit le nombre d'objets traités. Elle représente rapidement moins de 5% de la transformation : il est plus difficile pour LGP de transformer les actions CHECK en traitement que de calculer cette fermeture externe. La construction du graphe et l'extraction du plan relâché sont beaucoup plus couteux que la transformation : même si la transformation représente environ 70% du temps de planification pour les premiers problèmes, ce rapport chute rapidement et se stabilise à 25% à partir du problème impliquant 40 balles [11]. Comme les rapports se stabilisent, l'augmentation du nombre de balles dans les problèmes affecte pro-

8. Des expérimentations sur un domaine CHECKER « relâché », c'est-à-dire où la tare ne doit pas être effectuée après chaque pesée, ont été menées pour vérifier le comportement linéaire de l'extension et de l'extraction, avec LGP et GP.

9. Le plan extrait par GP contient $2n-1$ couches d'actions (chaque couche étant composée d'une unique action).

10. Dans le cas de CHECKER, l'objectif de planification est sans relation mutex à partir de la couche $\mathcal{P}_3$ (du fait des actions virtuelles), quel que soit le problème (sans compter le problème à une balle).

11. Nous prenons en compte uniquement les temps qui présentent un comportement régulier : à partir de 50 balles les premières irrégularités sont perceptibles.

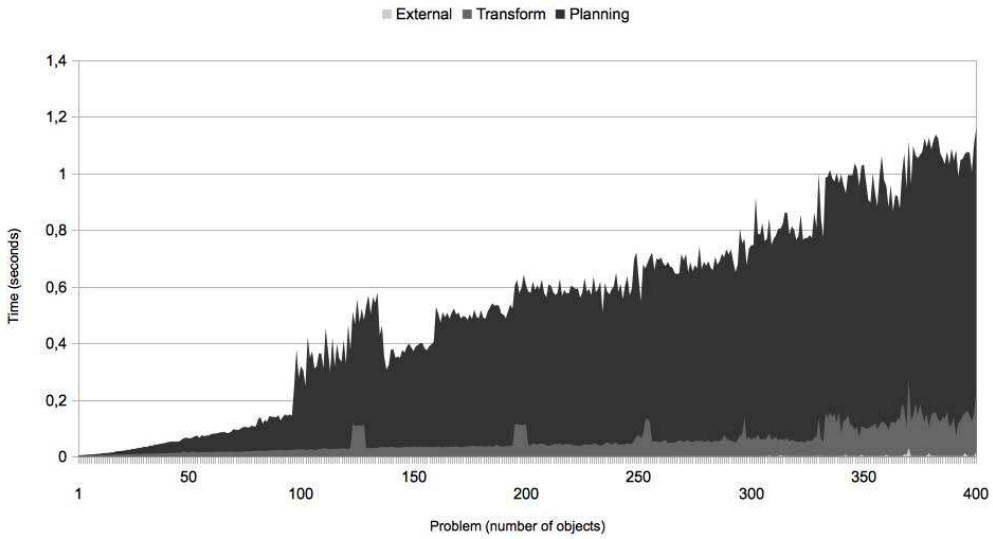


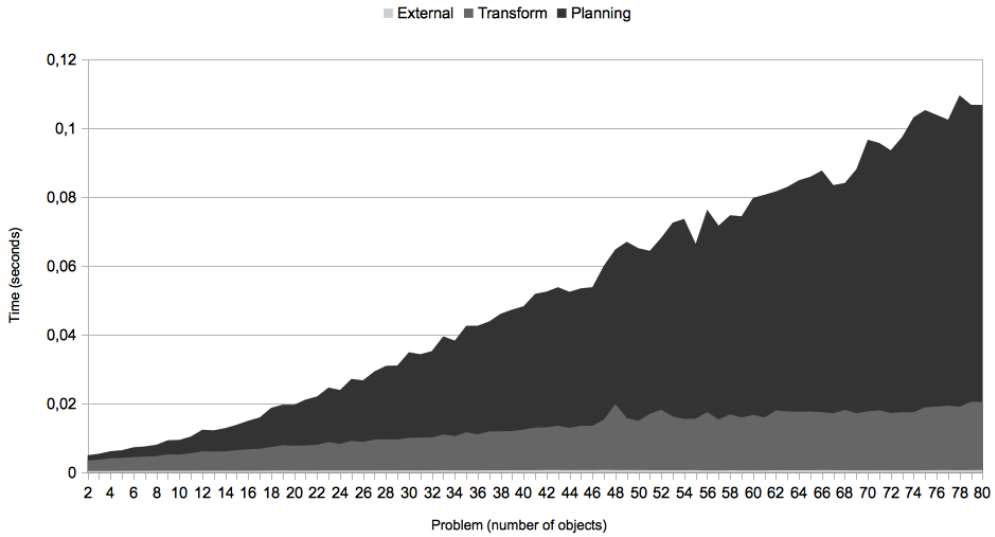
FIGURE 8.3 – Les temps (en seconde) de planification (Planning), de transformation (Transform) et de fermeture externe (Extern) pour chaque problème issu de CHECKER (en nombre de balles, de 1 à 400) pour le planificateur LGP.

portionnellement les trois processus, et les évolutions respectives des temps de calcul des processus sont du même ordre.

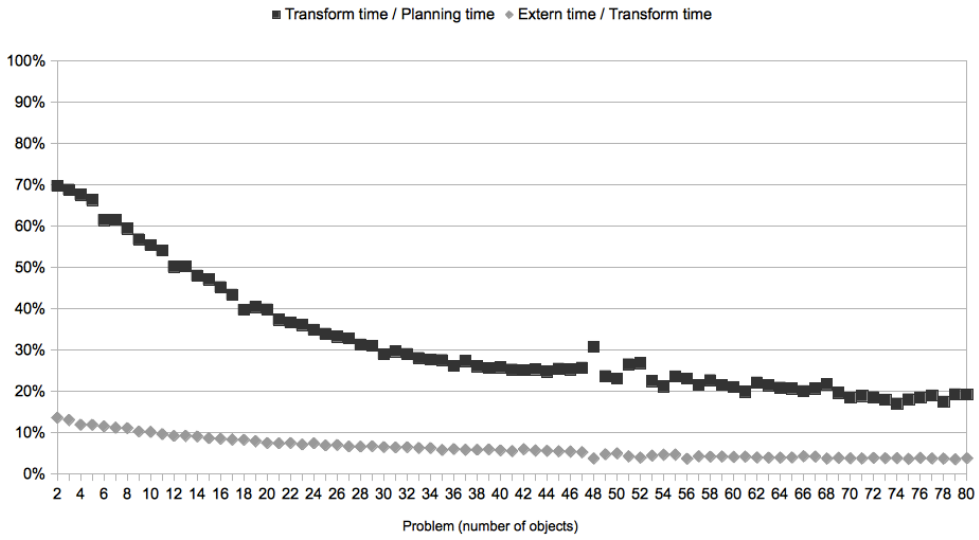
Temps normalisés pour LGP

Les plans calculés par FF et GP sont moins expressifs que les plans calculés par LGP. Nous proposons de diviser le temps de planification de chaque λ-plan (solution d'un problème donné) par sa largeur. Ainsi, nous obtenons un temps de planification « normalisé » sur le nombre de plans séquentiels exprimés par chaque λ-plan. La figure 8.5 présente les résultats de LGP en temps de planification normalisé sur les problèmes issus de CHECKER, pour les 8 premiers problèmes. La courbe décroit très rapidement, puisque le temps de planification augmente de façon linéaire tandis que la largeur du λ-plan augmente en factorielle du nombre de balles. Comparer le temps de calcul rapporté à la flexibilité des plans de FF et de GP ne modifie pas leurs courbes respectives, car la largeur des plans calculés vaut 1 quel que soit le nombre de balles. Ce mode de calcul est très favorable à LGP : pour les problèmes issus de CHECKER, la largeur des λ-plans augmente plus rapidement que le temps de planification de LGP. Par exemple, pour le problème CHECKER à 10 balles, LGP met environ 1 centième de seconde à calculer un λ-plan d'une largeur 3.628.800 (le nombre de permutations pour le traitement des 10 balles). Pour égaler les performances de LGP et obtenir un plan autant expressif, il faudrait qu'un planificateur classique utilise moins de 10 cycles de processeur [12] pour calculer chacun des plans décrits par le λ-plan (en considérant que chaque exécution de ce planificateur permet d'obtenir un plan différent). Ainsi, dans l'état actuel de la planification automatique, LGP est le planificateur le plus performant pour obtenir un plan d'une telle flexibilité avec le domaine CHECKER. Dès lors

12. Il est bien entendu impossible de calculer un plan en moins de 10 cycles de processeur : pour rappel, le processeur effectue $2,6 \times 10^9$ cycles/seconde, et $10 > (0.01 \text{ seconde } / 3.628.800) * 2,6 \times 10^9$.



(a)



(b)

FIGURE 8.4 – Comportement du planificateur `LGP` pour les premiers problèmes issus de CHECKER (en nombre de balles, de 2 à 80) :
(a) les temps (en seconde) de planification (Planning), de transformation (Transform) et de fermeture externe (External) ;
(b) les rapports (en pourcentage) du temps de transformation par rapport au temps de planification, et du temps de fermeture externe par rapport au temps de transformation.

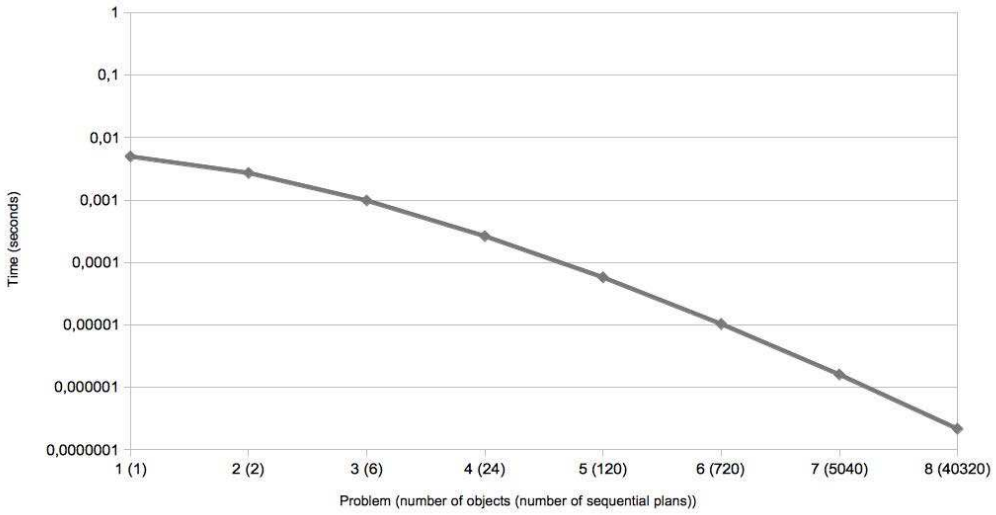


FIGURE 8.5 – Le temps de planification normalisé (en seconde) pour chaque problème issu de CHECKER (en nombre de balles, de 1 à 8, avec indiqué entre parenthèses la largeur du λ-plan) pour le planificateur `LGP`.

que le temps de planification normalisé chute en dessous de 10^{-8} seconde, cette conclusion est raisonnablement admissible [13].

8.1.2 Expérimentations sur Loader

L'objectif de planification dans LOADER (*cf.* annexe B.2) est d'*attraper* et de *charger* toutes les balles dans une boite. Pour trois balles `b-1`, `b-2` et `b-3`, le λ-plan calculé par `LGP` est le suivant :

> choose (*ball*) in {`b-1`, `b-2`, `b-3`}
> | `PICK`(*ball*) ≫ `LOAD`(*ball*)

`FF` et `GP` calculent un plan séquentiel, par exemple :

> `PICK(b-3)` ≫ `LOAD(b-3)` ≫ `PICK(b-2)` ≫ `LOAD(b-2)` ≫ `PICK(b-1)` ≫ `LOAD(b-1)`

Comparaison des planificateurs

Les expérimentations ont été menées sur les problèmes issus de LOADER avec un nombre de balles à charger variant de 1 à 200. La figure 8.6 page suivante permet de comparer les temps de calcul des trois planificateurs. Les figures 8.7(a) et 8.7(b) proposent les temps de planification de `FF` et de `GP` relativement au temps de planification de `LGP` pour ces problèmes. `GP` est très peu performant comparé à `FF` et `LGP` pour ces problèmes. Le rapport

13. Calculer un plan en moins de 26 instructions machine nous semblent raisonnablement impossible.

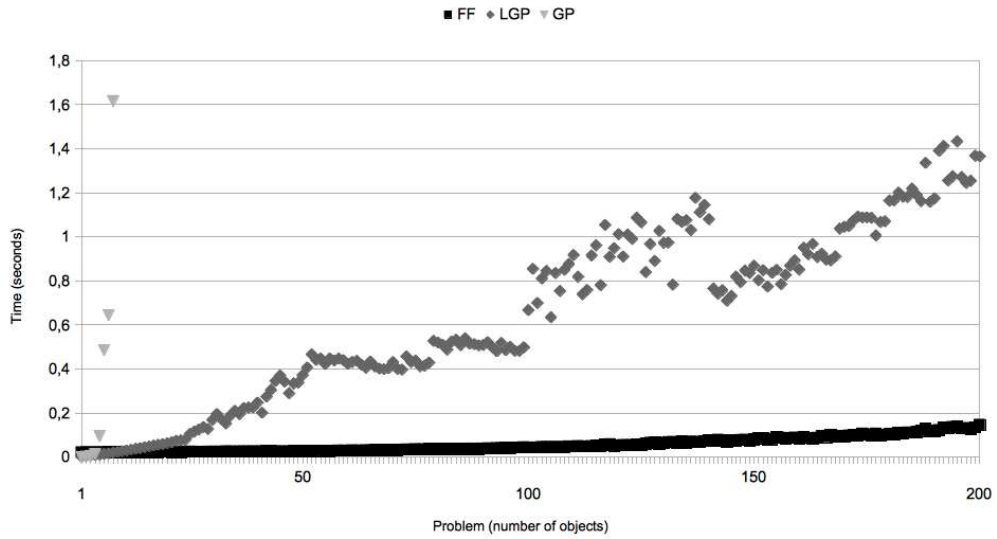


FIGURE 8.6 – La comparaison des temps de planification (en seconde) pour chaque problème issu de LOADER (en nombre de balles, de 1 à 200) pour les planificateurs `FF`, `LGP` et `GP`.

entre les temps de calcul de `LGP` et de `FF` oscille entre 10 et 15 à partir d'un certain nombre de balles (environ 125) : ces temps de calcul croissent apparemment de la même façon. Or, `FF` a un profil au moins polynomial sur ces problèmes, donc `LGP` aussi [14].

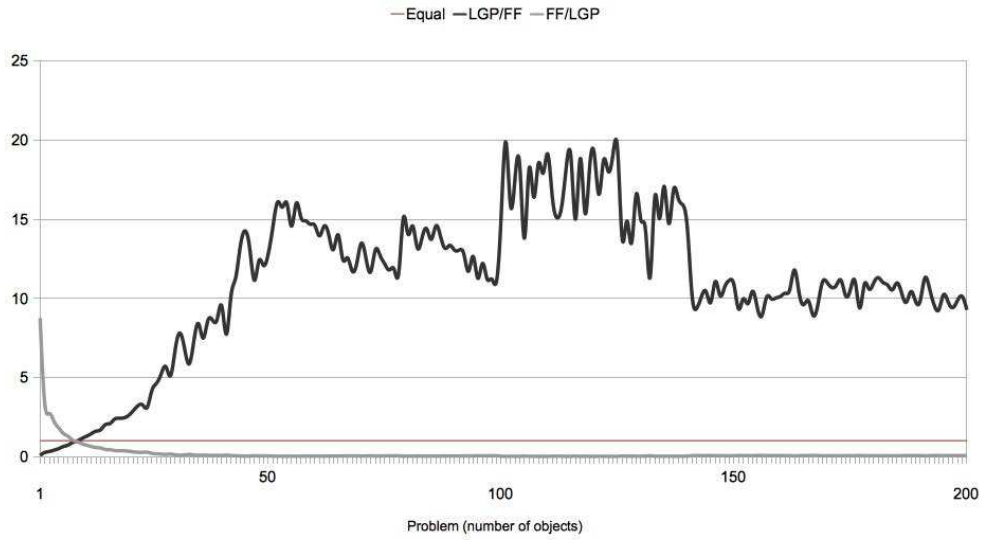
Observation de `LGP`

La figure 8.8 détaille l'évolution des temps de planification, de transformation et de fermeture externe. Les figures 8.9(a) et 8.9(b) limitent les observations aux problèmes 2 à 40. Les expérimentations illustrées à la figure 8.8 montrent que l'évolution du temps de calcul pour `LGP` n'est pas aussi forte que le laisse penser la vue restreinte de la figure 8.9(a). Mais nous observons tout de même que cette évolution est plus importante qu'avec les problèmes issus de CHECKER. Nous expliquons cette différence par quelques éléments liés aux propriétés des domaines :

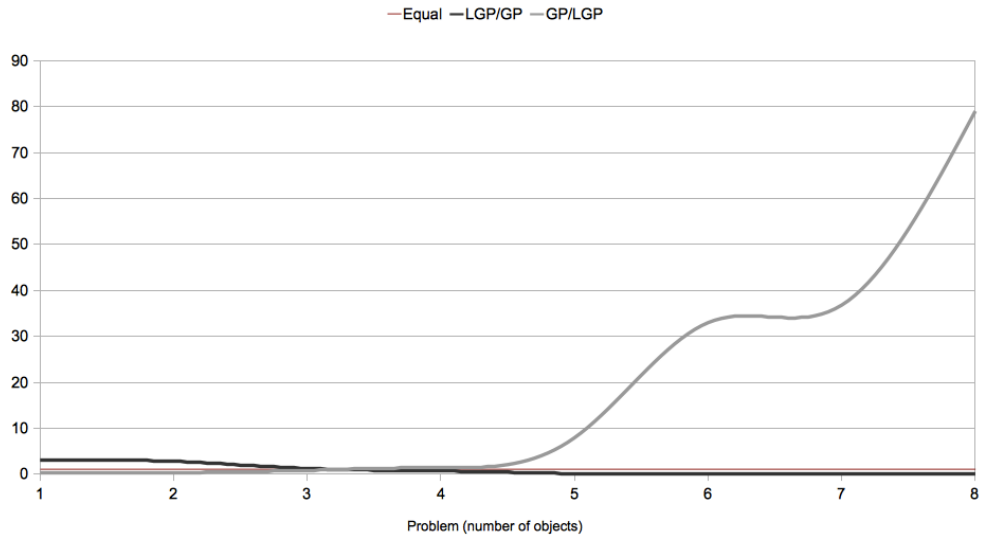
- le graphe de planification lors de l'extraction contient deux couches pour LOADER contre une seule pour CHECKER ;
- pour un même nombre de balles, le plan relâché pour LOADER contient deux fois plus d'actions que celui pour CHECKER ;
- la transformation du plan relâché pour LOADER nécessite la découverte de deux cliques d'actions contre une unique pour CHECKER ;
- le fermeture externe, inexistante dans LOADER, représente une part mineure de la transformation dans CHECKER.

Le rapport entre le temps de transformation et celui de planification se stabilise à 20% environ à partir du problème à 20 balles. La stabilité du rapport de la transformation dans la planification montre que la transformation se fait en temps polynomial au nombre de balles, puisque c'est le cas de la planification. Lorsque un λ-plan contient un traitement composé

14. Les irrégularités de `LGP` rendent difficiles la lecture des expérimentations : ses propriétés apparaissent plus clairement par la comparaison avec `FF`.



(a)



(b)

FIGURE 8.7 – Le rapport entre les temps de planification (en seconde) pour chaque problème issu de LOADER (en nombre de balles, de 1 à 200) :
(a) entre les planificateurs `FF` et `LGP` ;
(b) entre les planificateurs `GP` et `LGP` (jusqu'à 8 balles).

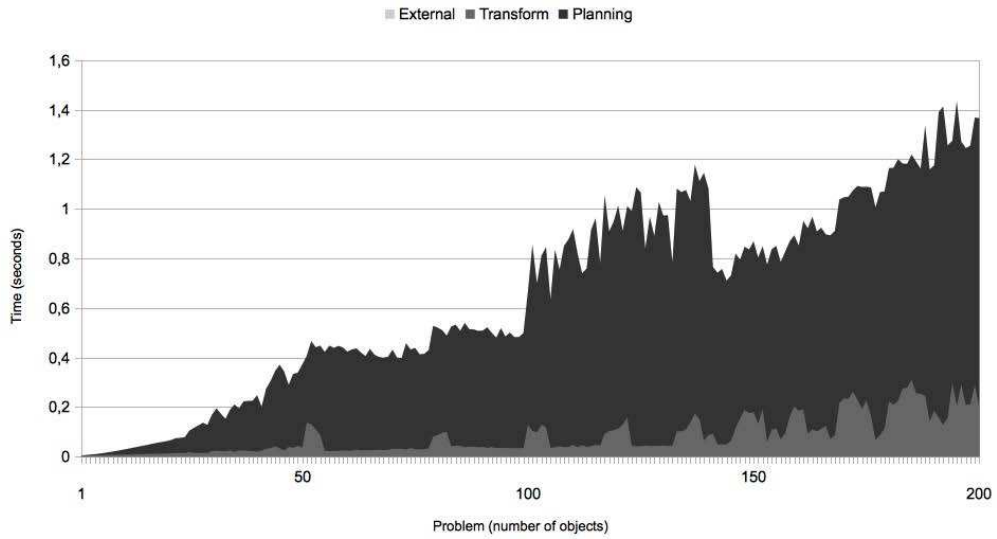


FIGURE 8.8 – Les temps (en seconde) de planification (Planning), de transformation (Transform) et de fermeture externe (Extern) pour chaque problème issu de LOADER (en nombre de balles, de 1 à 200) pour le planificateur `LGP`.

d'au moins deux actions abstraites, nous devrions observer une évolution des temps de calcul au moins polynomial au nombre d'objets traités. Le rapport de la transformation se stabilise plus rapidement que celui observé dans CHECKER. De plus, vis à vis de l'extension et l'extraction, la transformation est proportionnellement moins affectée par l'augmentation du nombre de balles dans LOADER que dans CHECKER : `LGP` a plus de facilité pour construire un traitement avec une action de plus que pour étendre un graphe avec une couche supplémentaire et y extraire un plan relâché. En revanche, pour LOADER, Le temps de fermeture externe est négligeable dans la transformation (le rapport tend vers 0) du λ-plan : les actions `PICK` et `LOAD` suffisent pour permettre les itérations successives du traitement.

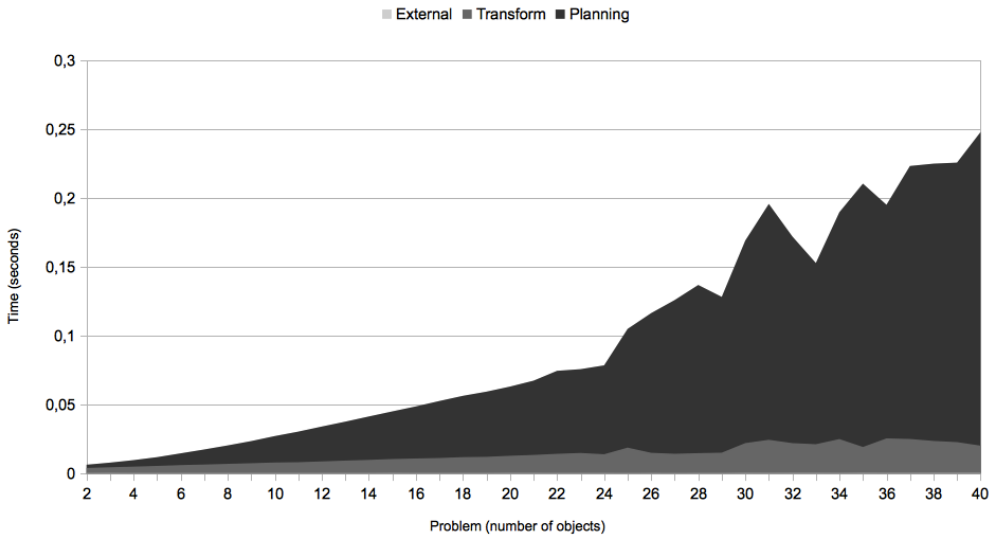
Temps normalisés pour `LGP`

La figure 8.10 présente les précédents résultats en temps de planification normalisés pour `LGP`. Nous observons une rapide chute de ces temps : le temps de planification augmente moins vite que la largeur des λ-plans lorsque le nombre de balles augmente dans les problèmes de LOADER.

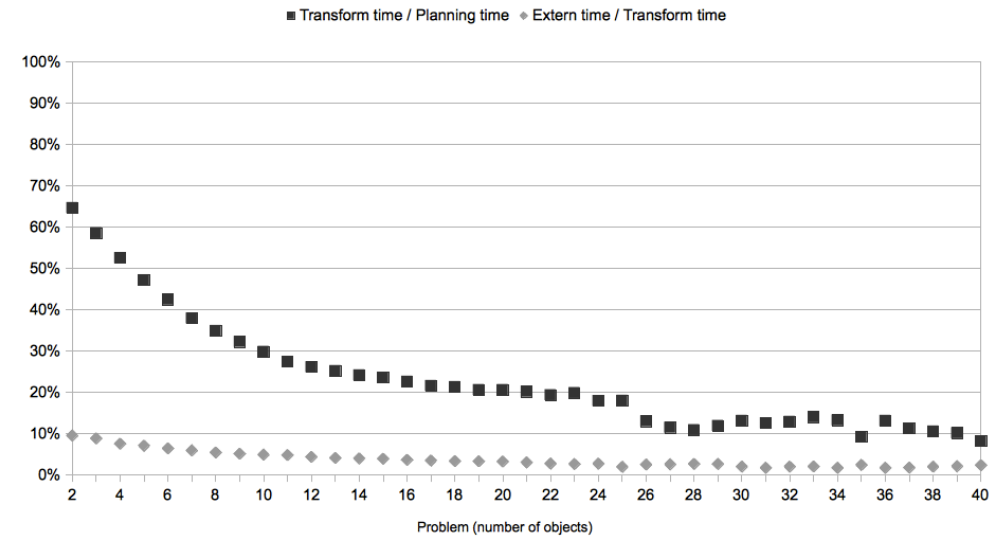
8.2 Cas d'étude

Dans ce paragraphe, nous observons `LGP` sur les problèmes de planification que nous avons développés :

- une itération simple avec GRIPPER, et avec une extension, GRIPPER-AIRLOCKED ;
- des itérations séquentielles avec GRIPPER-CART ;
- des itérations imbriquées avec PAINTER (à 2, 3 et 4 couleurs) ;
- des itérations parallèles avec GRIPPER-B+C.



(a)



(b)

FIGURE 8.9 – Comportement du planificateur LGP pour les premiers problèmes (en nombre de balles, de 2 à 40) issus de LOADER :
(a) les temps (en seconde) de planification (Planning), de transformation (Transform) et de fermeture externe (External) ;
(b) les rapports (en pourcentage) du temps de transformation par rapport au temps de planification, et du temps de fermeture externe par rapport au temps de transformation.

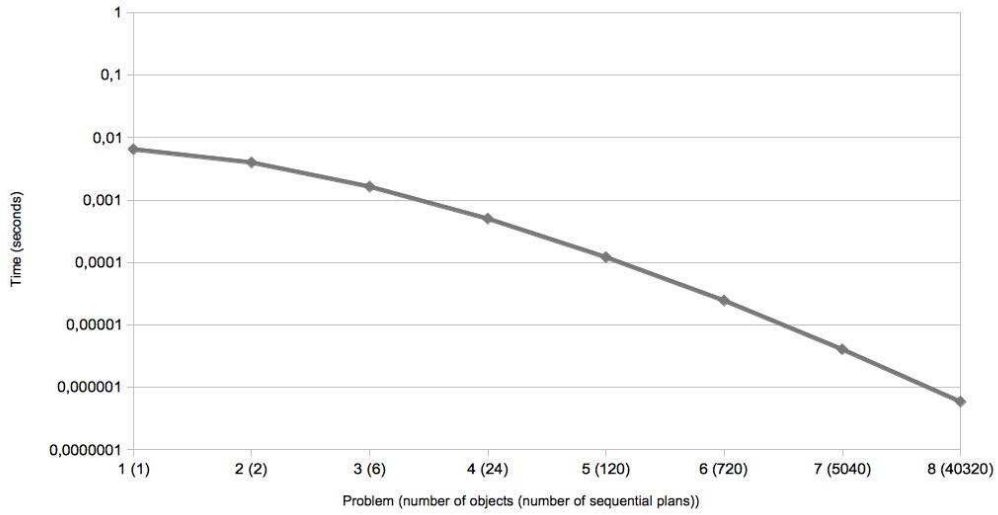


FIGURE 8.10 – Le temps de planification normalisé (en seconde) pour chaque problème issu de LOADER (en nombre de balles, avec indiqué entre parenthèses la largeur du λ-plan) pour le planificateur `LGP`.

8.2.1 Itération simple avec Gripper et Gripper-Airlocked

L'objectif de planification dans GRIPPER (*cf.* annexe B.3) est de déplacer toutes les balles de l'emplacement `b-a` à `b-z`. Pour trois balles `b-1`, `b-2` et `b-3`, le λ-plan calculé par `LGP` est le suivant :

choose (*ball*) in {`b-1`, `b-2`, `b-3`}
| `PICK(`*ball*`, r-a)` $\gg$ `MOVE(r-a, r-z)` $\gg$ `DROP(`*ball*`, r-z)`
| if-iteration `MOVE(r-z, r-a)`

GRIPPER est similaire à LOADER, avec une action supplémentaire en tant que support interne pour le traitement (découverte par le processus de fermeture interne), et une fermeture externe semblable à celle de CHECKER. Nous avons observé avec CHECKER que le temps consacré à la fermeture externe occupait une part constante dans le processus de transformation : avec GRIPPER, seule la fermeture interne n'a pas encore été expérimentée.

Pour rendre la fermeture externe un peu plus complexe, nous proposons le domaine GRIPPER-AIRLOCKED, une extension de GRIPPER, dans lequel le déplacement de `r-z` à `r-a` se fait obligatoirement par le passage d'un sas (décrit par l'emplacement `r-i` : le robot entre et sort de `r-i` par les actions `IN-AIR` et `OUT-AIR`), dans lequel quelques actions de vérification doivent être effectuées pour permettre au robot de ressortir du sas. Le λ-plan suivant est solution du problème à 3 balles :

choose (*ball*) in {`b-1`, `b-2`, `b-3`}
| `PICK(`*ball*`, r-a)` $\gg$ `MOVE(r-a, r-z)` $\gg$ `DROP(`*ball*`, r-z)`
| if-iteration `IN-AIR(r-z, r-i)`
| $\gg$ (`CHECK(r-i, size)` $\|$ `CHECK(r-i, weight)` $\|$ `CHECK(r-i, dirty)`)
| $\gg$ `OUT-AIR(r-i, r-a)`

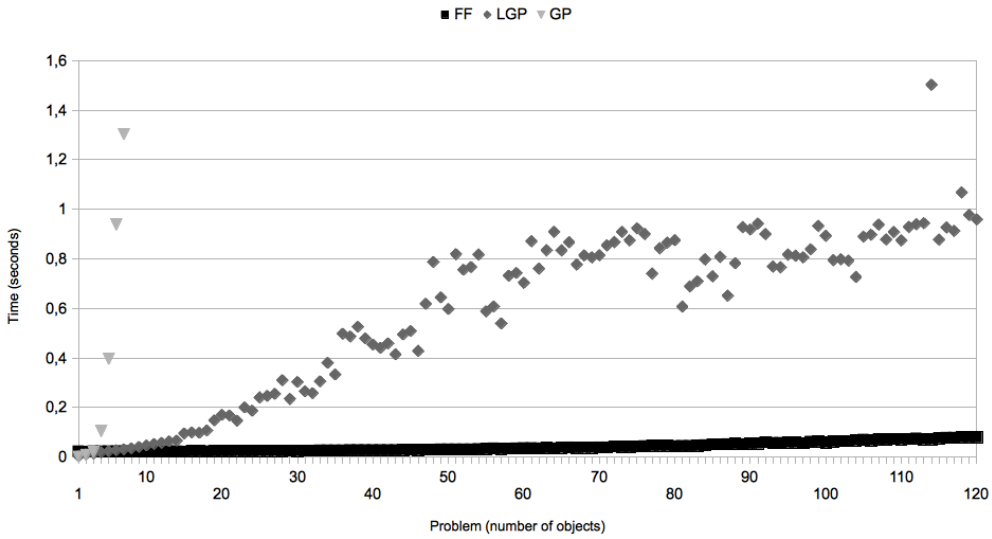


FIGURE 8.11 – La comparaison des temps de planification (en seconde) pour chaque problème issu de GRIPPER (en nombre de balles, de 1 à 120) pour les planificateurs `FF`, `LGP` et `GP`.

Comparaison des planificateurs

La figure 8.11 illustre les temps de calcul de `FF`, `LGP` et `GP` pour les problèmes issus de GRIPPER avec un nombre de balle à déplacer variant de 1 à 120. La figure 8.12 donne les temps de planification de `FF` relativement au temps de planification de `LGP` pour ces problèmes. Les performances de `LGP` se situent entre celles de `GP` et celles de `FF` : `GP` est moins performant que `LGP`, et `LGP` est moins performant que `FF`. En revanche, nous observons à la figure 8.12 que le rapport des performances entre `LGP` et `FF` se stabilise : à une exception près, ce rapport ne dépasse pas 15 au delà de 100 balles. Cela montre que les évolutions respectives des performances de `LGP` et `FF` sont d'un même ordre de croissance, à un facteur près (variant entre 10 et 15 environ) à l'avantage de `FF` avec GRIPPER. Ainsi, les expérimentations sur LOADER et GRIPPER donnent des résultats similaires : ce facteur était de 10 environ avec LOADER.

La figure 8.13 page suivante illustre les temps de calcul de `FF`, `LGP` et `GP` pour les problèmes issus de GRIPPER-AIRLOCKED avec un nombre de balles à déplacer variant de 1 à 120. La figure 8.14 propose les temps de planification de `FF` relativement au temps de planification de `LGP` pour ces problèmes : elle expose un facteur entre les évolutions du temps de planification variant entre 2 et 3 à partir du problème à 50 balles, à l'avantage de `LGP`.

Observation de `LGP`

Les figures 8.15(a) et 8.15(b) détaillent le rapport entre les temps de planification, de transformation et de fermeture externe pour les problèmes issus de GRIPPER et de GRIPPER-AIRLOCKED. Malgré les fortes irrégularités, nous observons pour le domaine GRIPPER-

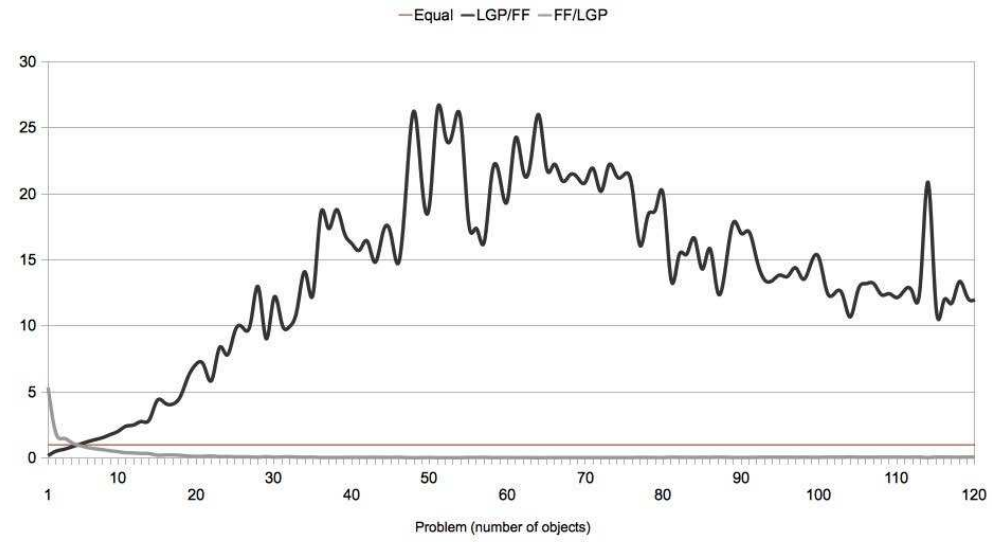


FIGURE 8.12 – Le rapport entre les temps de planification (en seconde) pour chaque problème (en nombre de balles, de 1 à 120) issus de GRIPPER entre les planificateurs `FF` et `LGP`

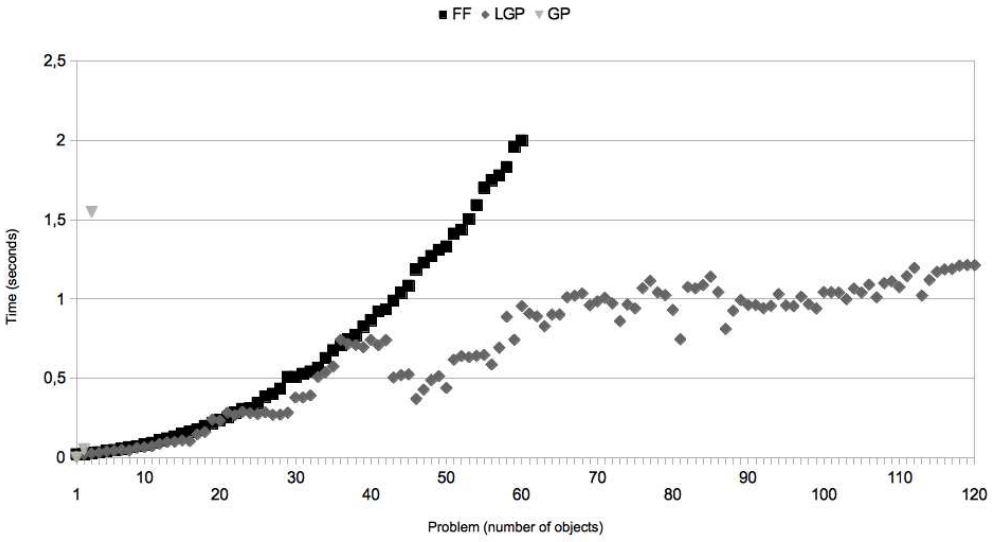


FIGURE 8.13 – La comparaison des temps de planification (en seconde) pour chaque problème issu de GRIPPER-AIRLOCKED (en nombre de balles, de 1 à 120) pour les planificateurs `FF`, `LGP` et `GP`.

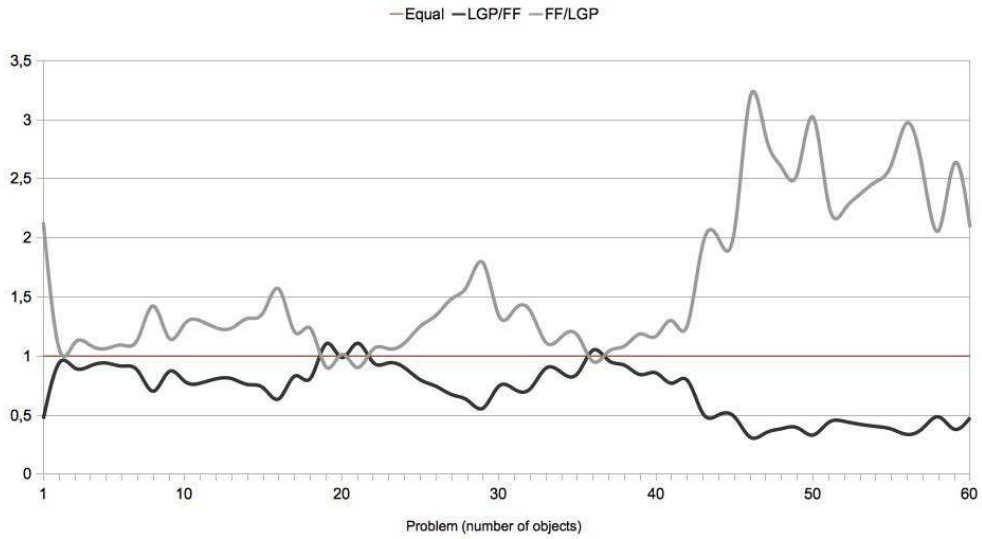


FIGURE 8.14 – Le rapport entre les temps de planification (en seconde) pour chaque problème issu de GRIPPER-AIRLOCKED (en nombre de balles, de 1 à 60) entre les planificateurs `FF` et `LGP`.

AIRLOCKED que la fermeture externe occupe environ 30% du temps de transformation alors pour le domaine GRIPPER ce rapport est d'environ 8%. La fermeture externe étant plus complexe avec GRIPPER-AIRLOCKED, le temps qui y est consacré est plus important.

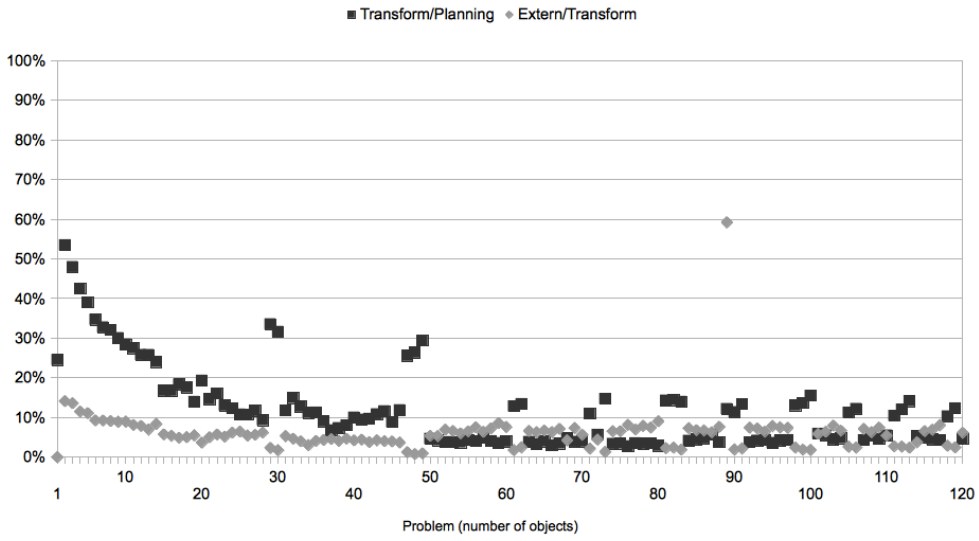
Dans les deux expérimentations, la transformation occupe environ 10% du temps de planification. Comparé aux expérimentations menées avec LOADER, la part de la transformation est deux fois plus petite qu'avec GRIPPER et GRIPPER-AIRLOCKED. Seule le support interne peut expliquer cette différence, puisque la fermeture externe n'a pas d'influence observable sur la transformation. En effet, la diminution de la part de la transformation s'explique par l'augmentation de la taille du graphe de planification lors de l'extraction du plan relâché : une couche supplémentaire est nécessaire dans GRIPPER et GRIPPER-AIRLOCKED pour le support interne.

8.2.2 Itérations séquentielles avec Gripper-Cart

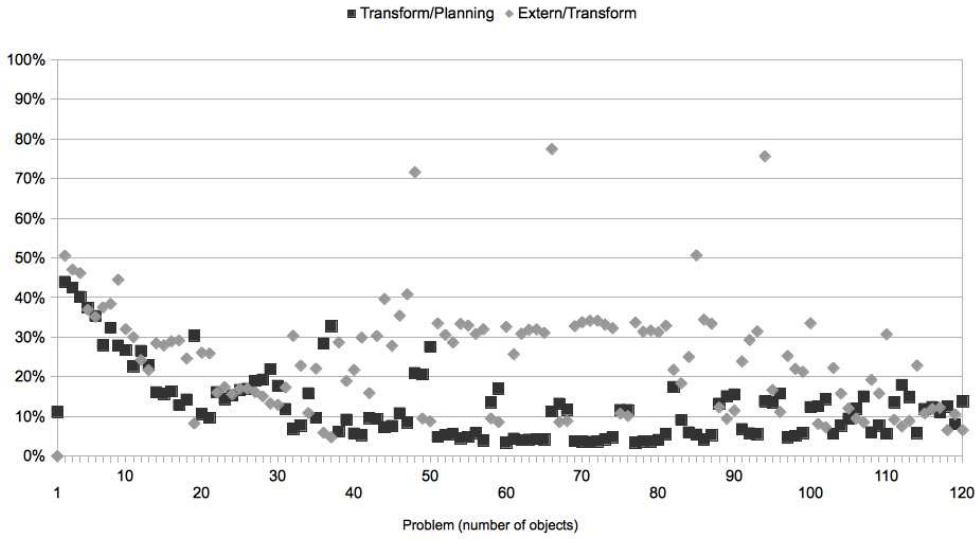
L'objectif de planification dans GRIPPER-CART (*cf.* annexe B.4) est de déplacer toutes les balles de l'emplacement `r-a` à `r-z`, notamment en utilisant une carriole. Pour deux balles `b-1` et `b-2`, le λ-plan calculé par `LGP` est le suivant :

```
choose (ball) in {(b-1), (b-2)}     ≫ MOVE(r-a, r-z) ≫     choose (ball) in {(b-1), (b-2)}
    | (PICK(ball, r-a)                                          | (EMPTY(ball)
    |     ≫ LOAD(ball))                                        |     ≫ DROP(ball, r-z))
```

GRIPPER-CART rassemble plusieurs des caractéristiques des domaines précédents : la première structure de choix itératif de ce λ-plan est la même que celle calculée pour LOADER, et la seconde y est semblable par sa forme ; aucune fermeture externe n'est requise, comme dans CHECKER ; l'action `MOVE(r-a, r-z)` est le support interne dans GRIPPER, et ici se trouve entre les deux structures de choix itératif. GRIPPER-CART nous permettra notamment d'évaluer le coût de la construction en séquence de structures de choix itératif.



(a)



(b)

FIGURE 8.15 – Les rapports (en pourcentage) du temps de transformation par rapport au temps de planification (Tranform/Planning), et du temps de fermeture externe par rapport au temps de transformation(Extern/Tranform) :
(a) pour les problèmes issus de GRIPPER (en nombre de balles, de 1 à 120) ;
(b) pour les problèmes issus de GRIPPER-AIRLOCKED (en nombre de balles, de 1 à 120).

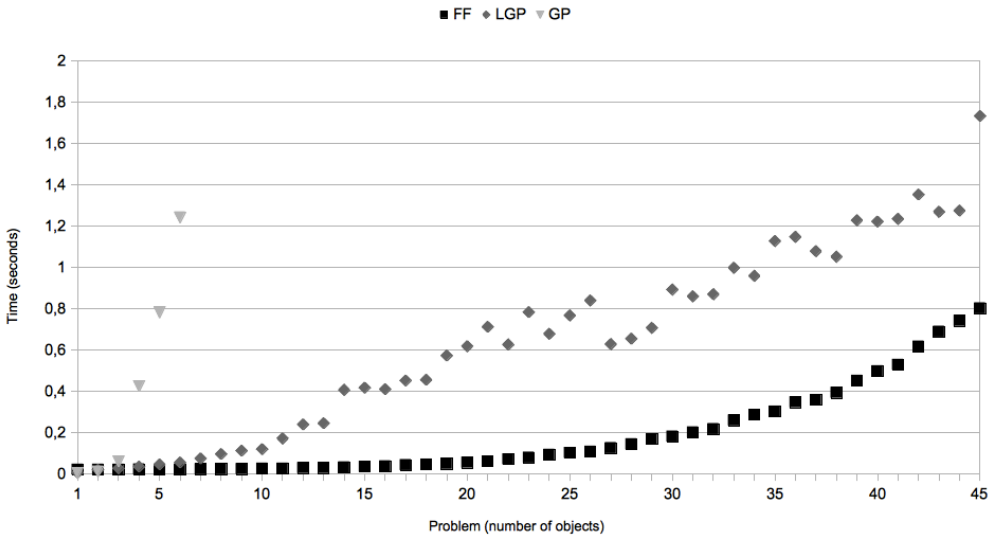


FIGURE 8.16 – La comparaison des temps de planification (en seconde) pour chaque problème issu de GRIPPER-CART (en nombre de balles, de 1 à 30) pour les planificateurs FF, LGP et GP.

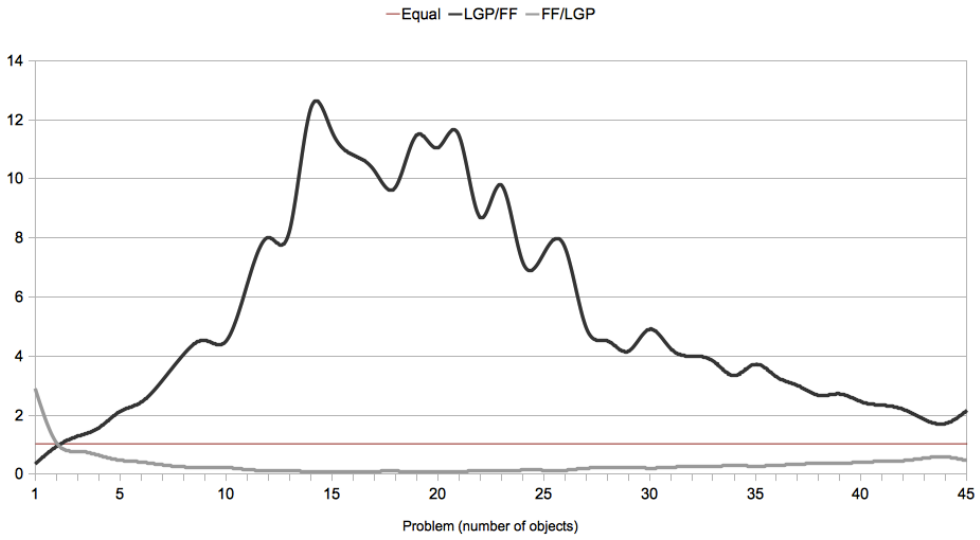
Pour un problème à n balles, la largeur du λ-plan solution vaut $n!^2$, donc ce λ-plan a une largeur de 4 (car il y a 2 manières de remplir la carriole, et 2 manières de la vider).

Comparaison des planificateurs

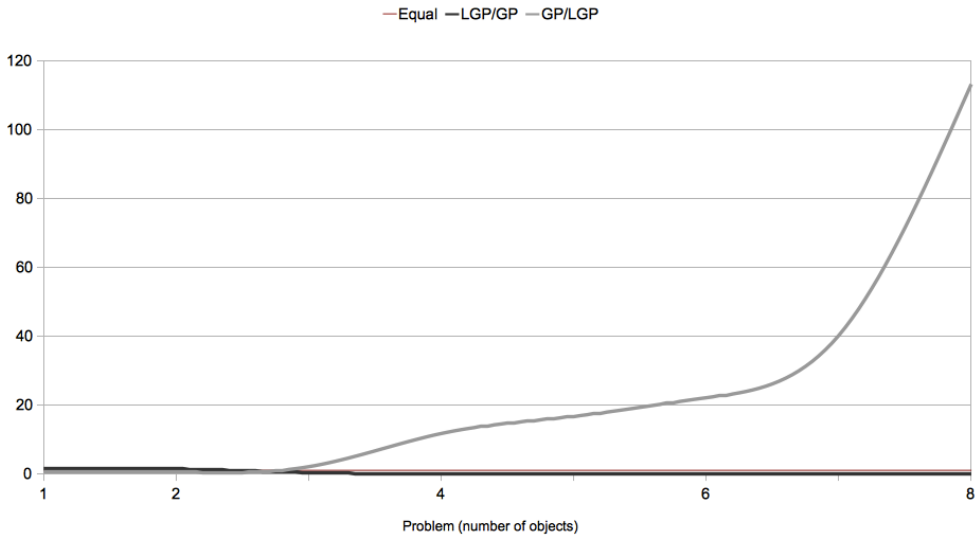
La figure 8.16 illustre les temps de calcul de FF, LGP et GP pour des problèmes issus de GRIPPER-CART avec un nombre de balles à déplacer variant de 1 à 45. Les figures 8.17(a) et 8.17(b) illustrent respectivement les temps de planification de FF et de GP relativement au temps de planification de LGP pour ces problèmes. LGP est plus performant que GP sur ces problèmes. Bien que les performances de LGP soient moins bonnes que celle de FF, l'écart entre leurs temps de planification diminue rapidement à partir du problème à 20 balles. L'espace mémoire est saturé avec LGP pour les problèmes ayant plus de 45 balles, mais en projetant l'évolution respective du temps de planification, LGP devrait dépasser les performances de FF avec l'augmentation du nombre de balles dans les problèmes. En comparant ces expérimentations avec celles de CHECKER, nous déduisons que LGP augmente ses performances, relativement à FF, lorsque le nombre de structures de choix itératif en séquences augmente.

Observation de LGP

La figure 8.18(a) détaille l'évolution des temps de planification, de transformation et de fermeture externe. La figure 8.18(b) reproduit les rapports entre ces temps. Comme avec LOADER, la part de la fermeture externe tend vers 0 car les structures de choix itératif dans les λ-plans pour GRIPPER-CART ne nécessitent pas de support externe. En revanche, la part de la transformation est plus faible avec GRIPPER-CART qu'avec LOADER. Les structures de choix itératif sont séquentielles dans le λ-plan solution : l'augmentation du coût pour leurs constructions (la transformation du plan relâché en λ-plan) est inférieure à l'augmentation

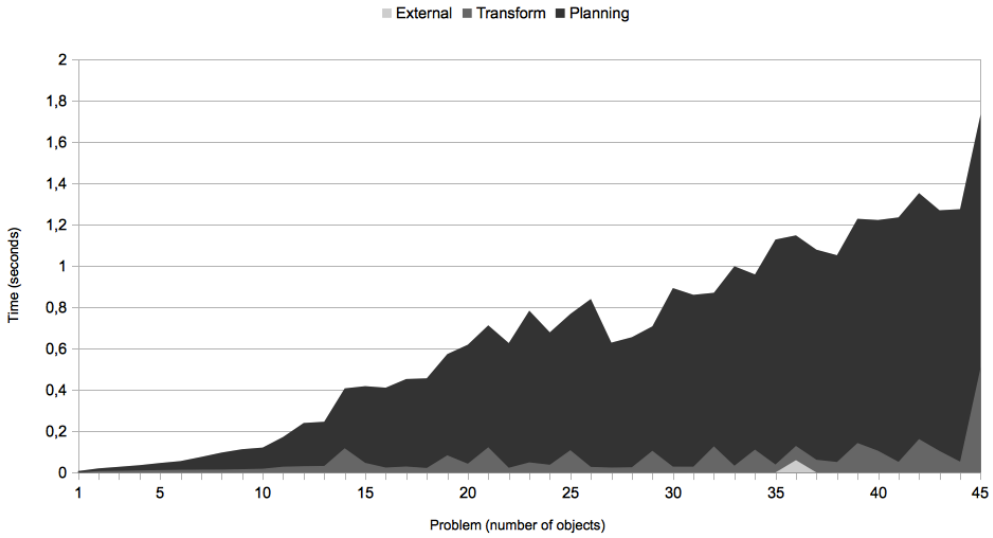


(a)

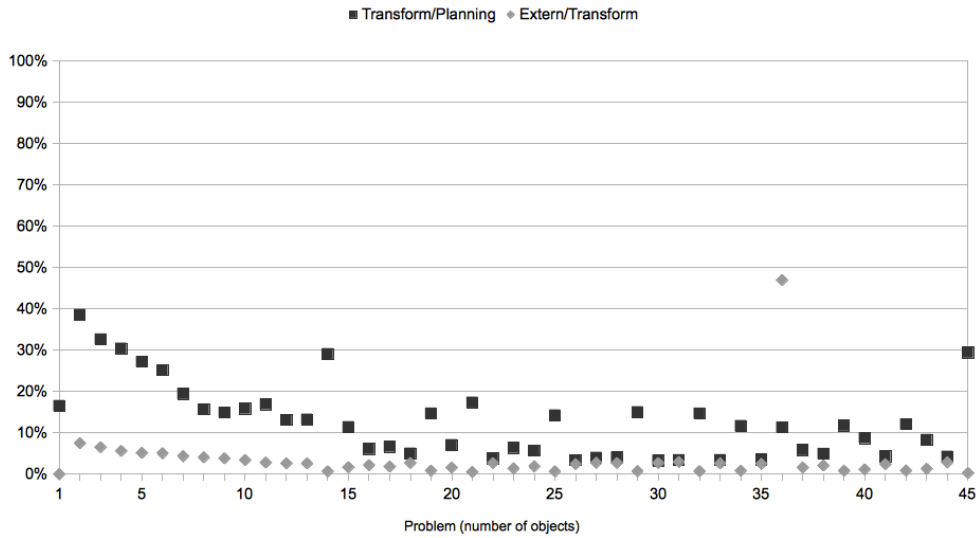


(b)

FIGURE 8.17 – Le rapport entre les temps de planification (en seconde) pour chaque problème (en nombre de balles, de 1 à 30) issus de GRIPPER-CART :
(a) entre les planificateurs `FF` et `LGP` ;
(b) entre les planificateurs `GP` et `LGP` (jusqu'à 8 balles).



(a) Les temps (en seconde) de planification (Planning), de transformation (Transform) et de fermeture externe (Extern).



(b) Les rapports (en pourcentage) du temps de transformation par rapport au temps de planification, et du temps de fermeture externe par rapport au temps de transformation

FIGURE 8.18 – Comportement du planificateur LGP pour les problèmes issus de GRIPPER-CART (en nombre de balles, de 1 à 45).

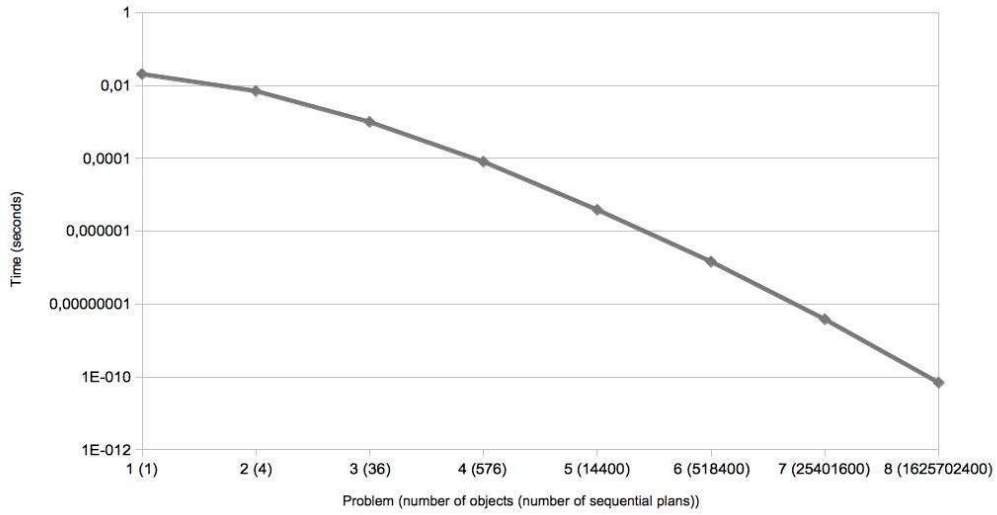


FIGURE 8.19 – Le temps de planification normalisé (en seconde) pour chaque problème issu de GRIPPER-CART (en nombre de balles, de 1 à 8, avec indiqué entre parenthèses la largeur du λ-plan) pour le planificateur `LGP`.

du coût pour l'extension du graphe de planification et l'extraction du plan relâché. Nous avions observé le même phénomène pour la fermeture interne avec GRIPPER.

Temps normalisés pour `LGP`

La figure 8.19 présente les résultats de `LGP` en temps de planification normalisé sur les problèmes issus de GRIPPER-CART, pour les 8 premiers problèmes. Ce temps diminue lorsque le nombre de balles augmente dans les problèmes. Cette diminution est plus rapide qu'avec les domaines CHECKER, LOADER et GRIPPER car la largeur des λ-plans augmente plus rapidement avec GRIPPER-CART qu'avec ces domaines, tandis que l'évolution du temps de planification de GRIPPER-CART est du même ordre que celle de ces domaines.

8.2.3 Itérations imbriquées avec Painter

L'objectif de planification dans PAINTER (*cf.* annexe B.5) est de *peindre* toutes les balles de deux couleurs chacune (`black` et `white`). Pour trois balles `b-1`, `b-2` et `b-3`, le λ-plan calculé par `LGP` est le suivant :

choose $(ball)$ in $\{(\texttt{b-1}), (\texttt{b-2}), (\texttt{b-3})\}$
| $(\texttt{WASH}(ball)$
| ≫ choose $(color)$ in $\{(\texttt{black}), (\texttt{white})\}$
| | $\texttt{PAINT}(ball, color)$
| | if-iteration `CLEAN` ≫ $\texttt{WASH}(ball)$
| if-iteration `CLEAN`

La largeur de ce plan vaut $48 = 3! \times 2!^3$, avec 3 le nombre de balles et 2 le nombre de couleurs : $n!$ différents ordres de traitement de n balles sont possibles, et pour k couleurs,

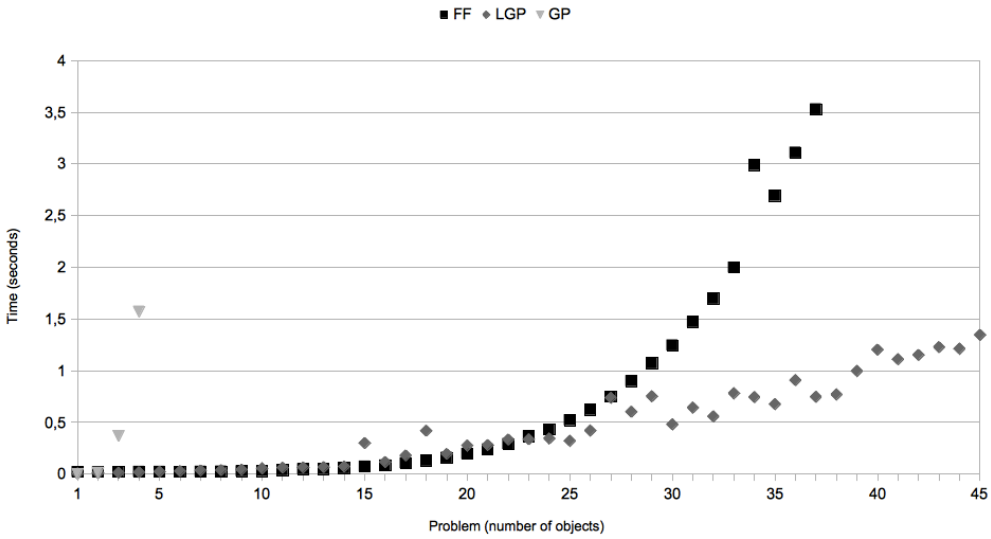


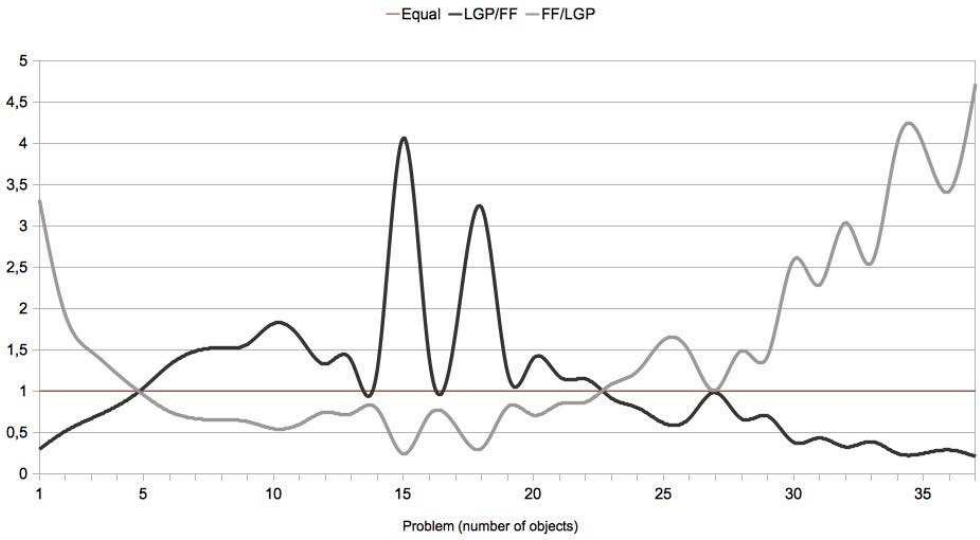
FIGURE 8.20 – La comparaison des temps de planification (en seconde) pour chaque problème issu de PAINTER (en nombre de balles, de 1 à 45) pour les planificateurs FF, LGP et GP.

chacune des balles peut être peinte de $k!$ différentes manières ; ainsi, pour chaque ordre de traitement des balles, il y a $k!^n$ manières de peindre toutes les balles ($k!^n$ correspond à la largeur de la séquence de n λ-plans de largeur $k!$).

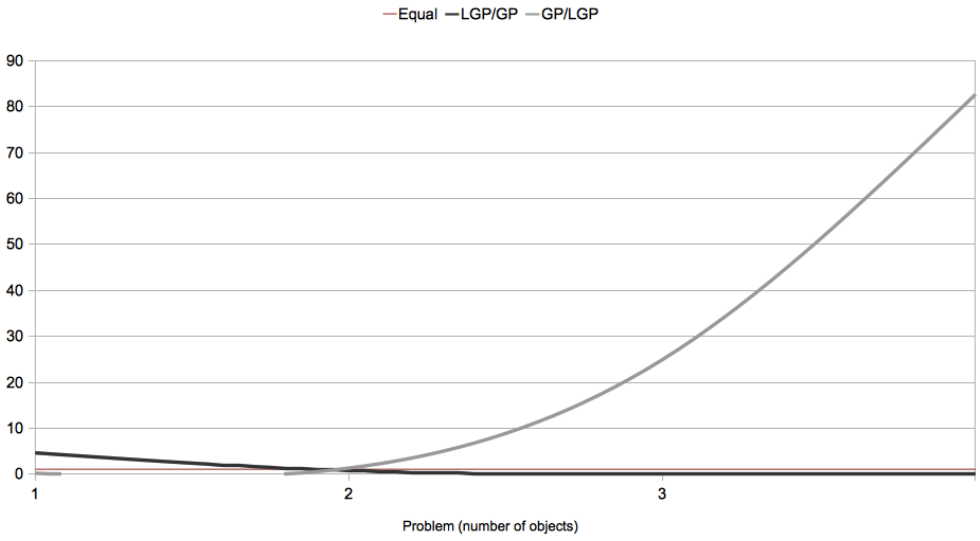
Comparaison des planificateurs

La figure 8.20 illustre les temps de calcul de FF, LGP et GP pour des problèmes issus de PAINTER avec un nombre de balle à peindre variant de 1 à 45. Les figures 8.21(a) et 8.21(b) donnent les temps de planification de FF et de GP relativement au temps de planification de LGP pour ces problèmes. LGP est plus performant que GP, et mis à part entre 5 et 23 balles dans les problèmes, LGP est plus performant que FF. De plus, l'écart des performances augmentent avec le nombre des balles, comme c'était le cas avec CHECKER. Mais avec PAINTER, l'augmentation de l'écart est encore plus rapide qu'avec CHECKER. Or les expérimentations montrent que, pour un même nombre de balles, la résolution des problèmes issus de PAINTER est plus difficile que la résolution des problèmes issus de CHECKER : pour les trois planificateurs, les temps de planification sont plus grands avec PAINTER qu'avec CHECKER. Ainsi, l'augmentation de la difficulté de résolution des problèmes affecte moins LGP que FF et GP lorsque les structures sont imbriquées. Cet avantage dans le comportement de LGP sur les deux autres planificateurs se confirme en observant l'évolution des temps de planification de la figure 8.22 page 156 : cette figure présente les temps de calcul des trois planificateurs pour des problèmes issus de PAINTER où les balles doivent être peintes de 3 couleurs différentes. En effet, les performances de LGP ne sont pas affectées [15] par l'augmentation du nombre de couleurs à appliquer, au contraire des performances de FF et GP qui diminuent ostensiblement.

15. Pour les problèmes à 2 couleurs et à 3 couleurs issus de PAINTER, l'écart des temps de planification de LGP est globalement à l'avantage des problèmes à 2 couleurs (d'environ 30%), mais est légèrement à l'avantage des problèmes à 3 couleurs sur les 5 dernières valeurs (d'environ 3%).



(a)



(b)

FIGURE 8.21 – Le rapport entre les temps de planification (en seconde) pour chaque problème issu de PAINTER (en nombre de balles) :
(a) entre les planificateurs `FF` et `LGP` (jusqu'à 37 balles) ;
(b) entre les planificateurs `GP` et `LGP` (jusqu'à 4 balles).

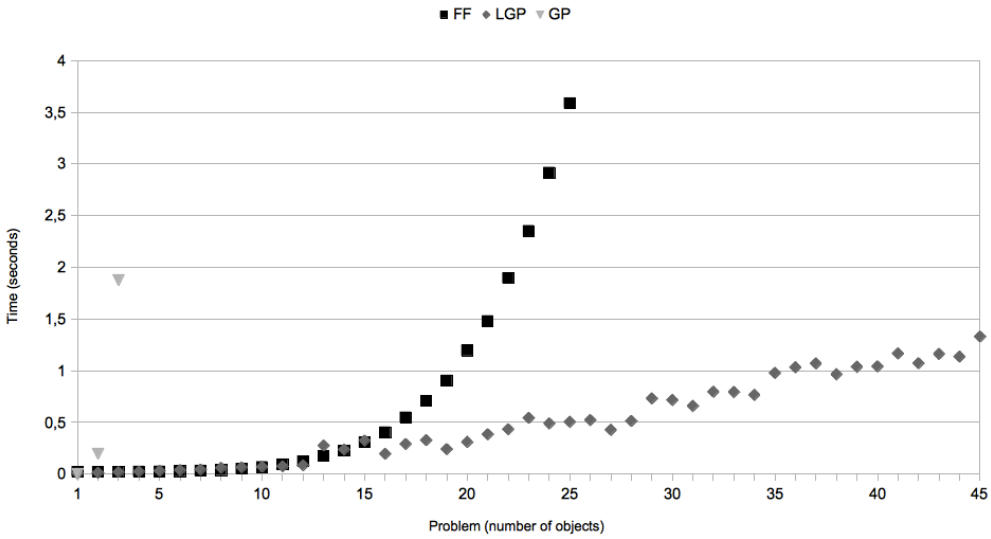


FIGURE 8.22 – La comparaison des temps de planification (en seconde) pour chaque problème issu de PAINTER « à 3 couleurs » (en nombre de balles, de 1 à 45) pour les planificateurs `FF`, `LGP` et `GP`.

Observation de `LGP`

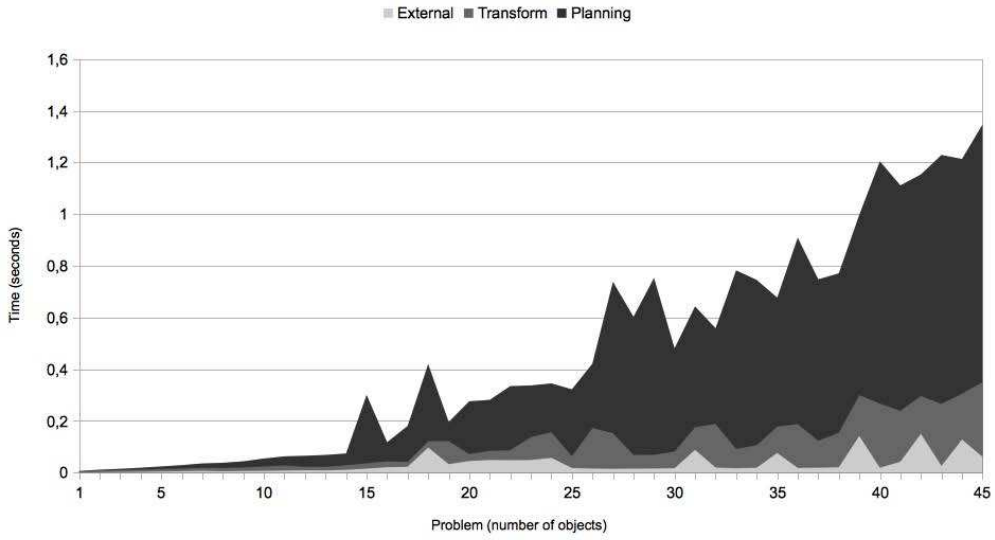
Les figures 8.23(a) et 8.23(b) détaillent l'évolution des temps de planification, de transformation et de fermeture externe de `LGP` pour les problèmes issus de PAINTER (avec 2 couleurs à appliquer). Les irrégularités dues à la gestion de la mémoire ne permettent pas d'observer de comportement manifeste de `LGP` sur PAINTER à 2 couleurs. Nous proposons de comparer ses performances avec un problème à 4 couleurs, exposées à la figure 8.24. Nous notons ainsi qu'au contraire des expérimentations précédentes, la part de la fermeture externe pour les structures imbriquées augmente avec le nombre de balles. Nous observons aussi, dans ces deux expérimentations, qu'à partir du problème à 25 balles, la transformation avoisine une part de 20% dans le temps de planification, alors que, dans les plans relâchés respectifs, les actions `PAINT` des problèmes à 2 couleurs sont deux fois moins que les actions `PAINT` à 4 couleurs. Ainsi, quel que soit le nombre de couleurs, la transformation semble garder une même part dans la planification.

Temps normalisés

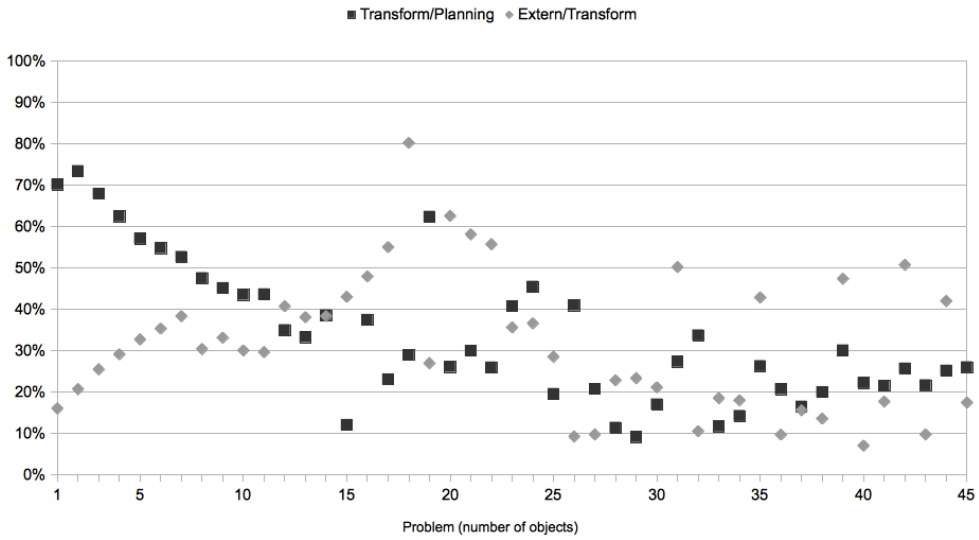
La figure 8.25 présente les résultats de `LGP` en temps de planification normalisé sur les problèmes issus de PAINTER (à 2 balles), pour les 8 premiers problèmes.

8.2.4 Itérations parallèles avec Gripper-B+C

L'objectif de planification dans GRIPPER-B+C (*cf.* annexe B.6) est de déplacer toutes les balles de l'emplacement `r-a` à `r-z` et tous les cubes de l'emplacement `r-z` à `r-a` à l'aide de deux robots, `g-b` et `g-c`. Pour trois balles `b-1`, `b-2` et `b-3` et trois cubes `c-1`, `c-2` et `c-3`, le λ-plan calculé par `LGP` est le suivant :

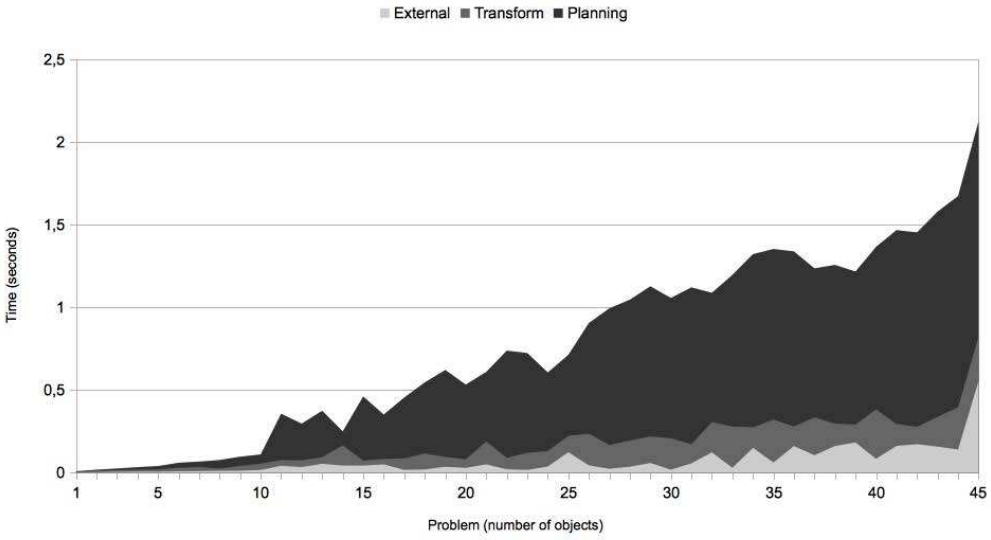


(a)

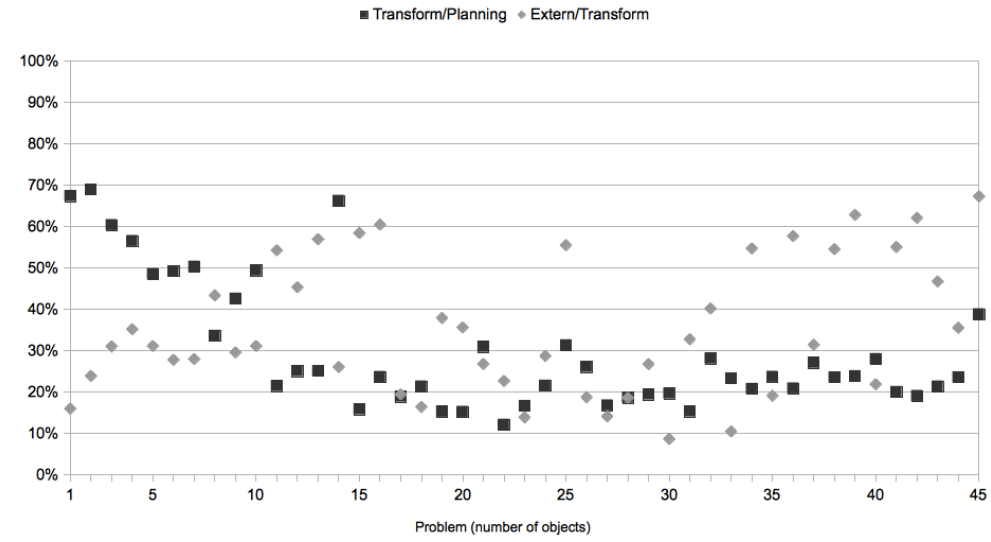


(b)

FIGURE 8.23 – Comportement du planificateur `LGP` pour les problèmes issus de PAINTER (en nombre de balles, de 1 à 45) :
(a) les temps (en seconde) de planification (Planning), de transformation (Transform) et de fermeture externe (External) ;
(b) les rapports (en pourcentage) du temps de transformation par rapport au temps de planification (Tranform/Planning), et du temps de fermeture externe par rapport au temps de transformation(Extern/Tranform).



(a)



(b)

FIGURE 8.24 – Comportement du planificateur LGP pour les problèmes issus de PAINTER « à 4 couleurs » (en nombre de balles, de 1 à 45) :
(a) les temps (en seconde) de planification (Planning), de transformation (Transform) et de fermeture externe (External) ;
(b) les rapports (en pourcentage) du temps de transformation par rapport au temps de planification (Tranform/Planning), et du temps de fermeture externe par rapport au temps de transformation(Extern/Tranform).

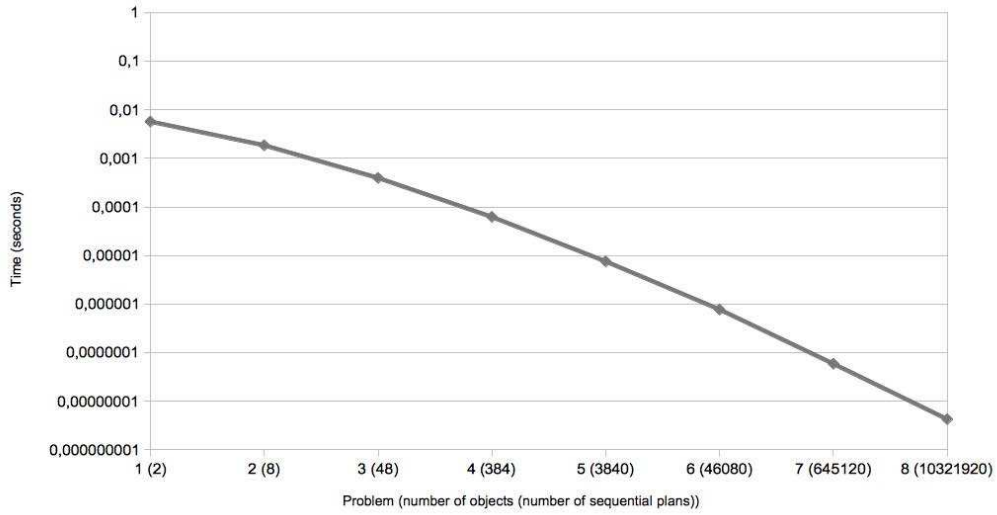


FIGURE 8.25 – Le temps de planification normalisé (en seconde) pour chaque problème issu de PAINTER (en nombre de balles, de 1 à 8, avec indiqué entre parenthèses la largeur du λ-plan) pour le planificateur `LGP`.

	1	2	3	4
`LGP`	20	13728	25395552	89347691520
`GP`	8	128	2048	32768

TABLE 8.1 – Comparaison de la largeur des plans calculés par `LGP` et `GP` pour les problèmes issus de GRIPPER-B+C, avec un nombre de balles et de cubes compris entre 1 et 4.

choose (*ball*) in {(`b-1`), (`b-2`), (`b-3`)}
| (`PICK(g-b,` *ball*`, r-a)`
| ≫ `MOVE(g-b, r-a, r-z)`
| ≫ `DROP(g-b,` *ball*`, r-z)`)
| if-iteration `MOVE(g-b, r-z, r-a)`

‖

choose (*cube*) in {(`c-1`), (`c-2`), (`c-3`)}
| (`PICK(g-c,` *cube*`, r-z)`
| ≫ `MOVE(g-c, r-z, r-a)`
| ≫ `DROP(g-c,` *cube*`, r-a)`)
| if-iteration `MOVE(g-c, r-a, r-z)`

La largeur $\#\Pi$ de ce λ-plan dépend du nombre n de balles et de cubes. Notons $l = 4n - 1$ la longueur du traitement de toutes les balles (ainsi que la longueur du traitement de tous les cubes) : 4 actions sont appliquées pour une balle donnée (resp. pour un cube donné), sauf à la dernière itération pour laquelle la dernière action n'est pas appliquée. Ainsi, $\#\Pi = n! \times n! \times f(l, l)$ avec $f(l, l)$ la fonction permettant de calculer la largeur de l'entrelacement de deux plans de longueur l (*cf.* paragraphe 6.3.2 page 79), et $n!$ pour la largeur de chacune des structures de choix itératif. `FF` calcule un plan séquentiel, non flexible. En revanche, `GP` permet d'entrelacer deux actions pour chaque couche de son plan solution (*cf.* plan 8.1 page suivante). La largeur de ce plan pour le traitement de n balles et n cubes vaut 2^{4n-1} : il y a 2 possibilités pour chaque couche du plan. Le tableau 8.1 donne les largeurs des plans calculés respectivement par `LGP` et par `GP` pour les 4 premiers problèmes de GRIPPER-B+C. Quel que soit le problème issu de GRIPPER-B+C, la largeur des plans de `LGP` est supérieure à celle des plans de `GP`. De plus, l'évolution de la largeur des plans avec le nombre de balles est plus rapide dans `LGP` que dans `GP`.

Plan 8.1 Plan solution extrait par GP pour Gripper-B+C.

```
 1 PICK(g-c, b-1, r-a), PICK(g-b, c-3, r-z);
 2 MOVE(g-c, r-a, r-z), MOVE(g-b, r-z, r-a);
 3 DROP(g-c, b-1, r-z), DROP(g-b, c-3, r-a);
 4 MOVE(g-c, r-z, r-a), MOVE(g-b, r-a, r-z);
 5 PICK(g-c, b-2, r-a), PICK(g-b, c-2, r-z);
 6 MOVE(g-c, r-a, r-z), MOVE(g-b, r-z, r-a);
 7 DROP(g-c, b-2, r-z), DROP(g-b, c-2, r-a);
 8 MOVE(g-c, r-z, r-a), MOVE(g-b, r-a, r-z);
 9 PICK(g-c, b-3, r-a), PICK(g-b, c-1, r-z);
10 MOVE(g-c, r-a, r-z), MOVE(g-b, r-z, r-a);
11 DROP(g-c, b-3, r-z), DROP(g-b, c-1, r-a);
```

Comparaison des planificateurs

La figure 8.26 page suivante illustre les temps de calcul de FF, LGP et GP pour les problèmes issus de Gripper-B+C avec un nombre de balles et de cubes à déplacer variant de 1 à 30. Les figures 8.27(a) et 8.27(b) proposent les temps de planification de FF et de GP relativement au temps de planification de LGP pour ces problèmes. La mémoire est saturée pour les problèmes contenant plus de 30 balles et 30 cubes. LGP est plus performant que GP, et FF est plus performant que LGP. D'après la figure 8.27(a), il semble que le rapport entre les temps de planification de FF et de LGP ne diverge pas. Ce rapport varie entre 10 et 30 à partir du problème à 15 balles et 15 cubes.

Observation de LGP

Les figures 8.28(a) et 8.28(b) détaillent l'évolution des temps de planification, de transformation et de fermeture externe. Les expérimentations sur Gripper-B+C ont révélée un comportement non déterministe de l'implémentation de LGP : les λ-plans calculés pour un même problème n'étaient pas toujours les mêmes. Nous supposons que cela est du à la gestion de la mémoire : ce non déterminisme s'est révélé à partir du problème à 7 balles et 7 cubes. Sur les 6 premiers problèmes, le comportement de LGP est régulier : le temps de la fermeture externe est négligeable pour la résolution de ces problèmes, et le rapport du temps de la transformation dans la planification est le plus faible que l'on ait observé jusque là. En comparant le temps d'obtention des 6 premiers λ-plans avec ceux pour les 6 premiers problèmes issus de Gripper, nous observons qu'avec Gripper-B+C, l'extension du graphe et l'extraction du plan relâché est environ 4 fois plus couteux, tandis que la transformation est environ 4 fois moins couteuse. Cela explique la diminution de la part de la transformation sans que nous ayons d'explication pour cette diminution du temps de transformation entre Gripper et Gripper-B+C.

Temps normalisés

La figure 8.29 présente les résultats de LGP en temps de planification normalisé sur les problèmes issus de Gripper-B+C, pour les 4 premiers problèmes. Ce temps normalisé est comparé à celui de GP, puisque GP calcule un plan flexible. Le temps normalisé est beaucoup plus faible avec LGP qu'avec GP et diminue beaucoup plus rapidement lorsque le nombre de balles augmentent. L'évolution avec le nombre de balles de la largeur des plans est plus rapide dans LGP que dans GP, et est plus rapide que l'évolution des temps de calcul respectif.

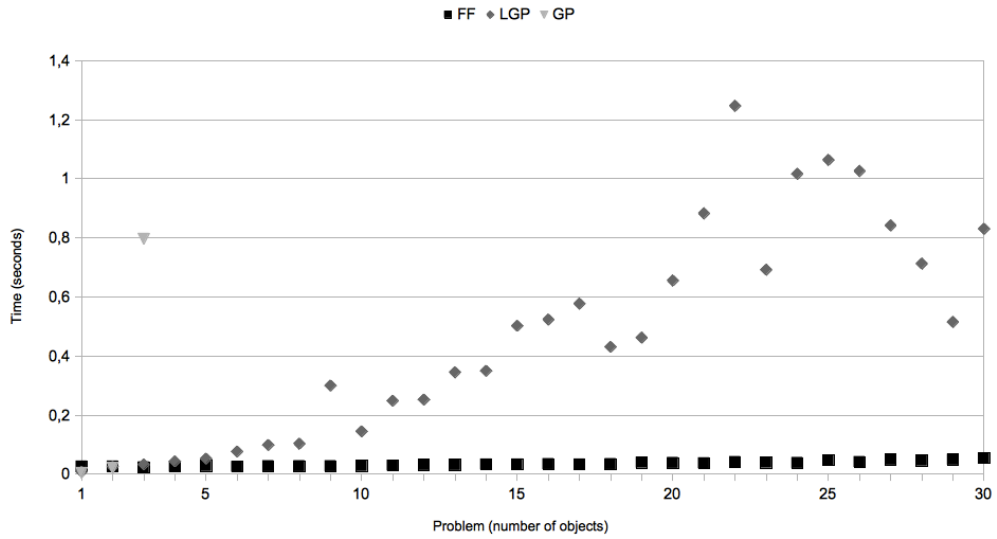


FIGURE 8.26 – La comparaison des temps de planification (en seconde) pour chaque problème issu de GRIPPER-B+C (en nombre de balles et de cubes, de 1 à 30) pour les planificateurs `FF`, `LGP` et `GP`.

8.3 Domaines linéaires

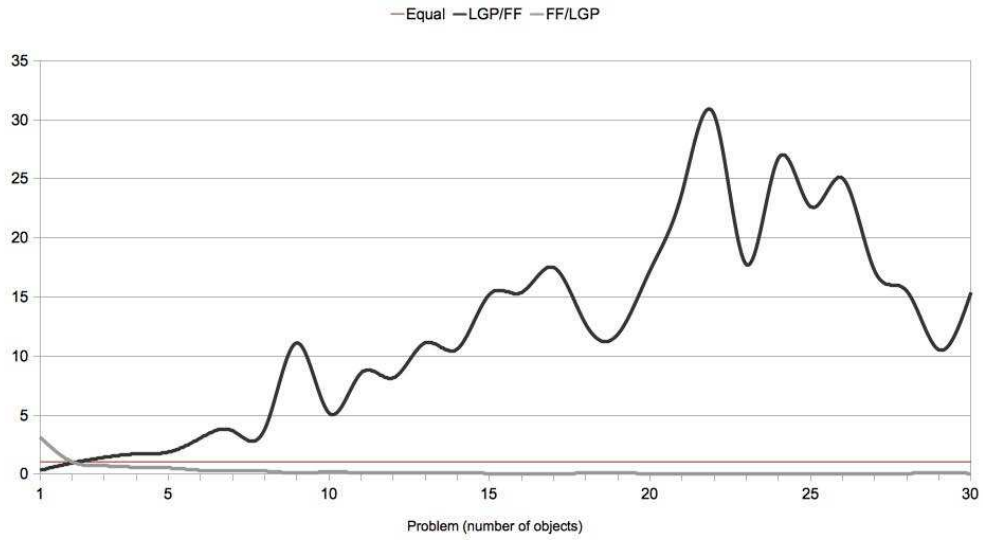
Dans ce paragraphe, nous évaluons les performances de `LGP` en tant qu'extension de `GP`, c'est-à-dire en mesurant le coût du marquage des relations mutex et de l'étiquetage du graphe de planification. Nous étudions ces performances dans les situations les plus défavorables pour `LGP`, c'est-à-dire lorsque un unique plan séquentiel est solution. C'est par exemple le cas avec un « monde des blocs » (BLOCKSWORLD) dans lequel les n blocs d'une pile doivent être posés sur la table. Nécessairement, le plan solution dépile le bloc du sommet, le dépose sur la table et ainsi de suite (il n'existe donc aucune option ouverte pour le contrôleur). Le problème des « tours de Hanoï » (HANOI) présente les mêmes caractéristiques : l'ordre des actions imposé dans chaque plan séquentiel solution est nécessaire à la réalisation de l'objectif de planification. Nous utiliserons ces deux domaines « linéaires » pour exposer les performances de `LGP`.

8.3.1 Expérimentations sur BlocksWorld

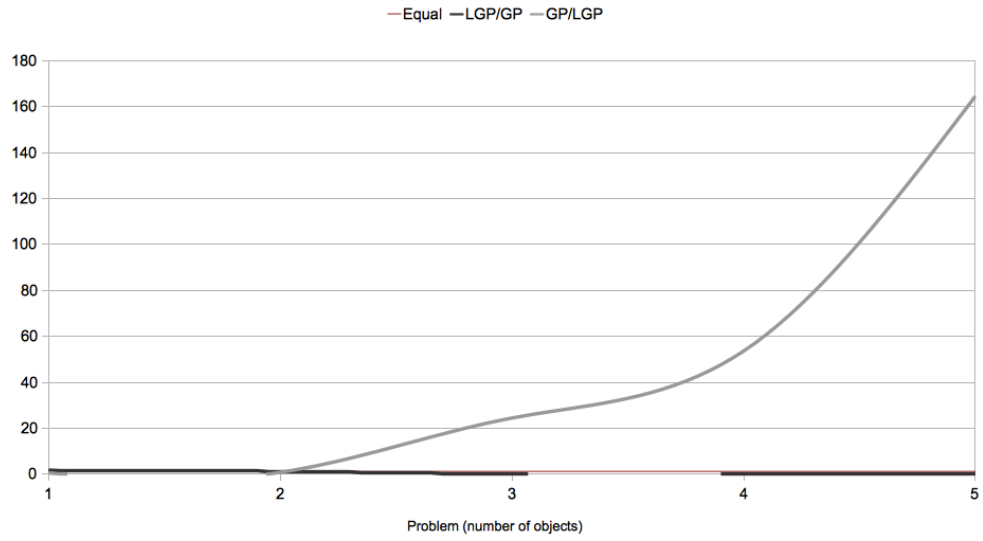
Le domaine BLOCKSWORLD est l'un des exemples les plus courant en planification. Nous utilisons un ensemble de problèmes où une pile de n blocs doit être complètement dépliée : l'objectif est d'obtenir tous les blocs sur la table. Un plan solution pour une pile de trois blocs (`b-1` sur `b-2` sur `b-3`) est le suivant :

```
UNSTACK(b-1, b-2) ≫ PUTDOWN(b-1) ≫ UNSTACK(b-2, b-3) ≫ PUTDOWN(b-2)
```

Ce plan n'est pas flexible.

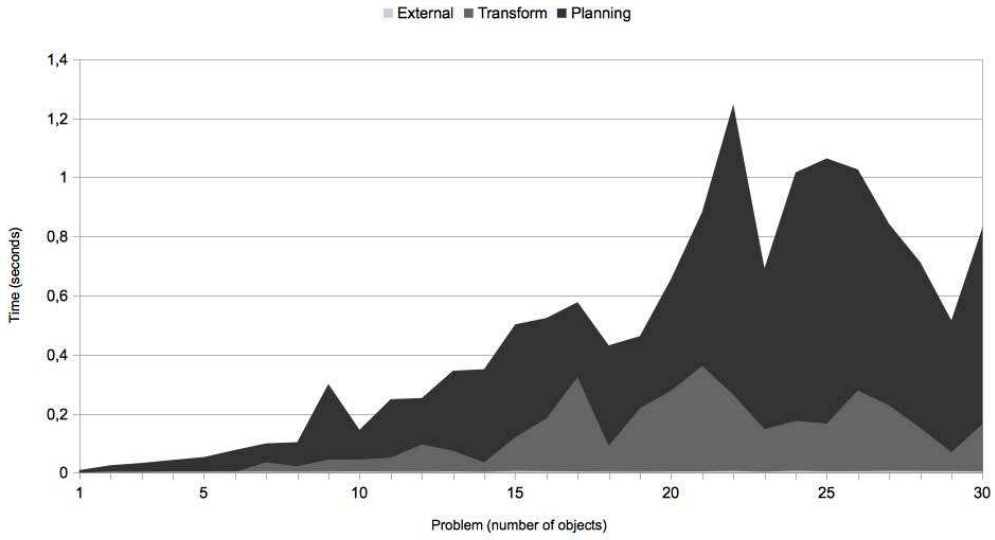


(a)

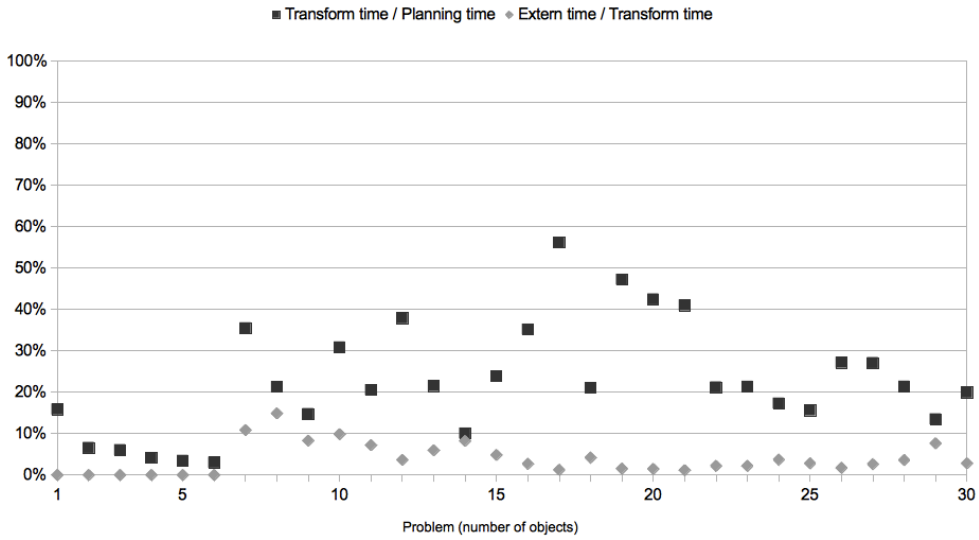


(b)

FIGURE 8.27 – Le rapport entre les temps de planification (en seconde) pour chaque problème (en nombre de balles et de cubes, de 1 à 30) issus de GRIPPER-B+C :
(a) entre les planificateurs `FF` et `LGP` ;
(b) entre les planificateurs `GP` et `LGP` (jusqu'à 5 balles).



(a)



(b)

FIGURE 8.28 – Comportement du planificateur LGP pour les problèmes issus de GRIPPER-B+C (en nombre de balles et de cubes, de 1 à 30) :
(a) les temps (en seconde) de planification (Planning), de transformation (Transform) et de fermeture externe (External) ;
(b) les rapports (en pourcentage) du temps de transformation par rapport au temps de planification (Tranform/Planning), et du temps de fermeture externe par rapport au temps de transformation(Extern/Tranform).

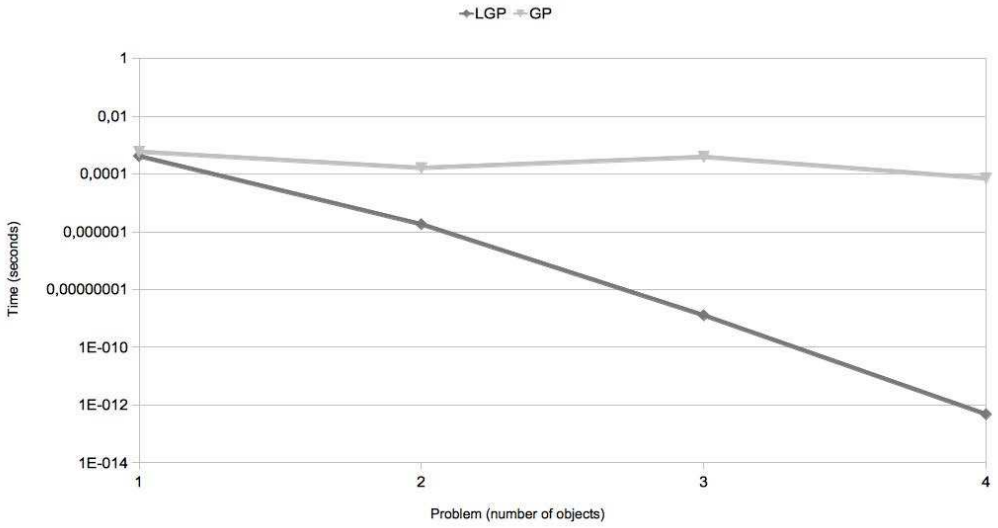


FIGURE 8.29 – Les temps de planification normalisé (en seconde) pour chaque problème issu de GRIPPER-B+C (en nombre de balles, de 1 à 4) pour `LGP` et `GP`.

Comparaison des planificateurs

La figure 8.30 page ci-contre illustre les temps de calcul de `FF`, `LGP` et `GP` pour les problèmes issus de BLOCKSWORLD avec un nombre de cubes à déplacer variant de 1 à 15. `LGP` est moins performant que `GP` pour ce problème. Cela est conforme à nos attentes : la baisse de performance est due à la charge liée à la gestion des relations λ-mutex. Ces problèmes ne permettent pas la découverte de structures de choix itératif. Or ce sont ces structures qui permettent à `LGP` d'être performant : elles permettent une limitation de l'extension du graphe de planification. Les figures 8.31(a) et 8.31(b) proposent les temps de planification de `FF` et de `GP` relativement au temps de planification de `LGP` pour ces problèmes. Le rapport des temps de planification entre `LGP` et `FF` augmente avec le nombre de cubes des problèmes, à l'avantage de `FF`. En revanche, ce rapport entre `LGP` et `GP` varie entre 3 et 7. Cela signifie que la charge d'utilisation des relations λ-mutex est proportionnelle aux performances de `GP` pour ce domaine linéaire.

Observation de `LGP`

Les figures 8.32(a) et 8.32(b) détaillent l'évolution des temps de planification, de transformation et de fermeture externe et les rapports entre ces temps. Rapidement, la part de la transformation tend vers 0 : c'est l'extension du graphe de planification et l'extraction du plan qui prennent quasiment la totalité du temps de planification.

8.3.2 Expérimentations sur Hanoi

Le domaine HANOI est un autre des exemples courant en planification. Un certain nombre de *disques* de taille croissante sont empilées à un emplacement `p-a`, et deux emplacements `p-i` et `p-z` sont libres. Un disque ne peut être déplacé que sur un disque plus grand ou sur un emplacement libre (l'action `MOVE(d-i, q-j, q-k)` signifie que le disque `d-i` est enlevé de sur `q-j` pour être mis sur `q-k`). L'objectif est de déplacer toute la pile sur

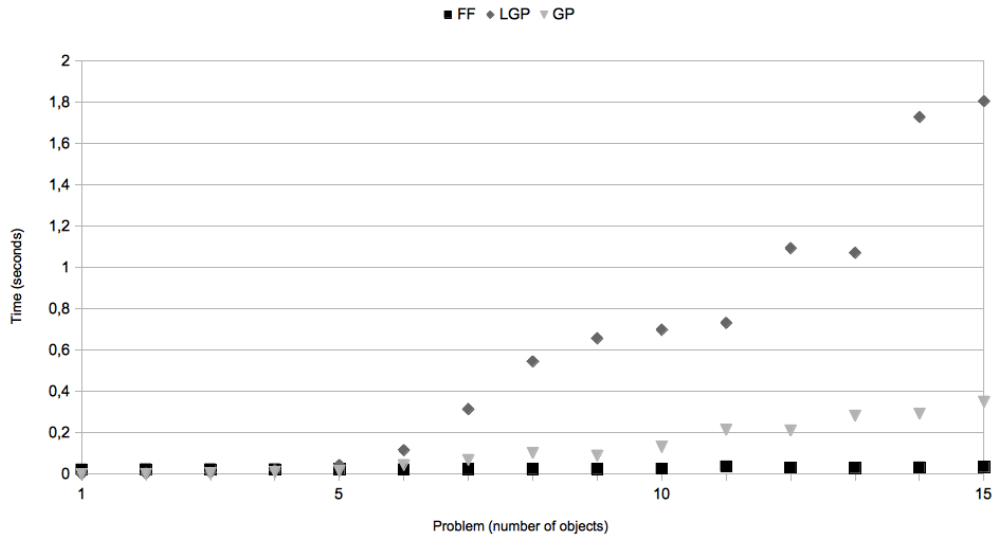


FIGURE 8.30 – La comparaison des temps de planification (en seconde) pour chaque problème issu de BLOCKSWORLD (en nombre de cubes, de 1 à 15) pour les planificateurs `FF`, `LGP` et `GP`.

l'emplacement `p-z`. Par exemple, avec trois disques `d-1`, `d-2` et `d-3`, avec `d-1` sur `d-2` sur `d-3` sur `p-a`, un plan solution est :

```
MOVE(d-1, d-2, p-z)
≫ MOVE(d-2, d-3, p-i)
≫ MOVE(d-1, p-z, d-2)
≫ MOVE(d-3, p-a, p-z)
≫ MOVE(d-1, d-2, p-a)
≫ MOVE(d-2, p-i, d-3)
≫ MOVE(d-1, p-a, d-2)
```

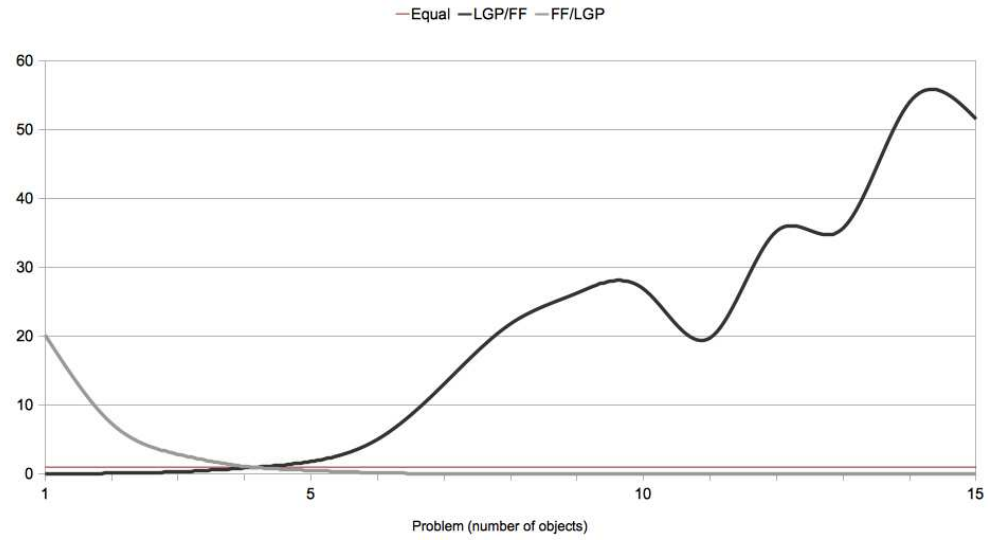
Ce plan n'est pas flexible.

Comparaison des planificateurs

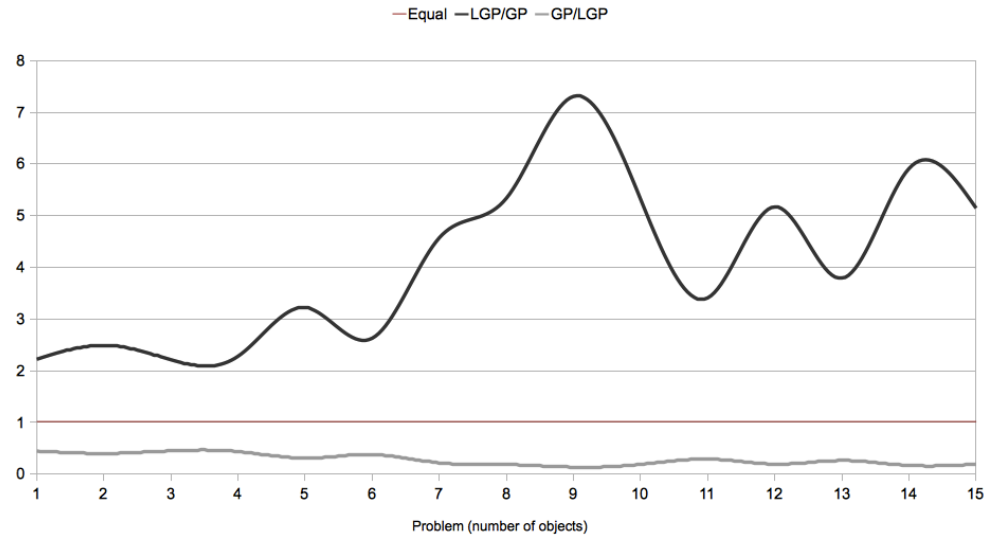
La figure 8.33 page 168 illustre les temps de calcul de `FF`, `LGP` et `GP` pour les problèmes issus de HANOI avec un nombre de disques variant de 2 à 5. Les figures 8.34(a) et 8.34(b) proposent les temps de planification de `FF` et de `GP` relativement au temps de planification de `LGP` pour ces problèmes. Les observations sont similaires que pour les expérimentations sur BLOCKSWORLD. En effet, les performances de `LGP` sont très inférieures à celles de `FF`, mais sont semblables à celles de `GP` : pour ces problèmes, les temps de planification de `LGP` sont « au pire » 2,5 fois plus long que ceux de `GP`.

Observation de `LGP`

Les figures 8.35(a) et 8.35(b) détaillent l'évolution des temps de planification, de transformation et de fermeture externe et les rapports entre ces temps. L'extension du graphe

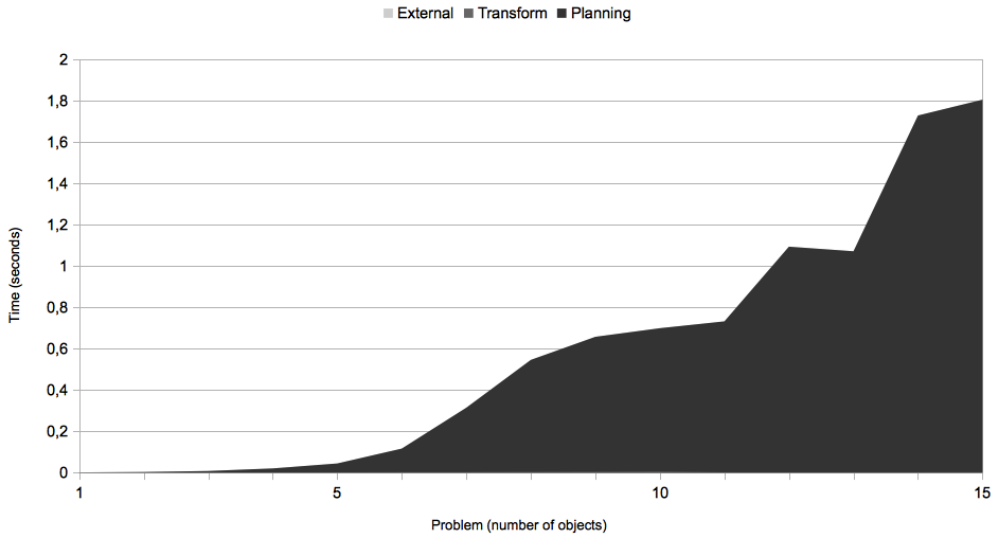


(a)

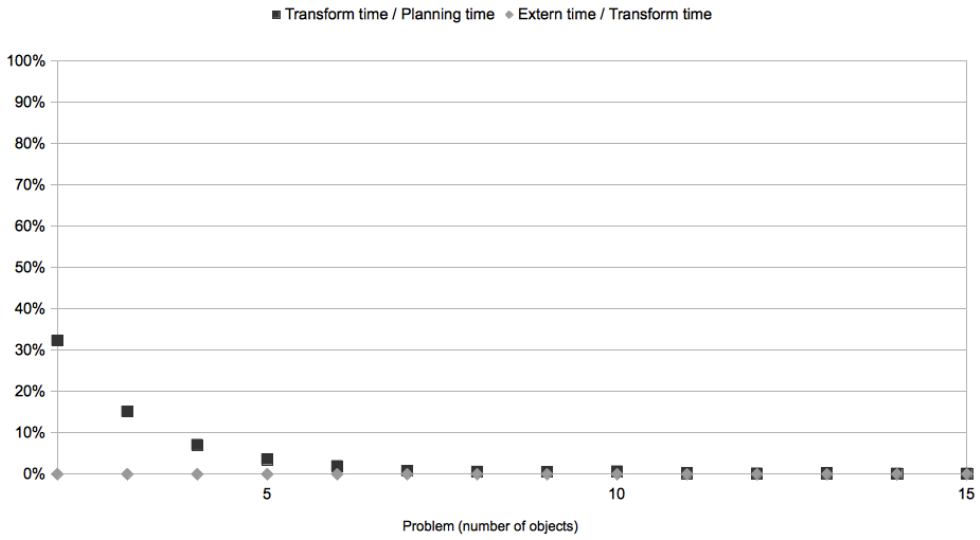


(b)

FIGURE 8.31 – Le rapport entre les temps de planification (en seconde) pour chaque problème (en nombre de cubes, de 1 à 15) issus de BLOCKSWORLD :
(a) entre les planificateurs `FF` et `LGP` ;
(b) entre les planificateurs `GP` et `LGP`.



(a)



(b)

FIGURE 8.32 – Comportement du planificateur `LGP` pour les problèmes issus de BLOCKS-WORLD (en nombre de cubes, de 1 à 15) :
(a) les temps (en seconde) de planification (Planning), de transformation (Transform) et de fermeture externe (External) ;
(b) les rapports (en pourcentage) du temps de transformation par rapport au temps de planification, et du temps de fermeture externe par rapport au temps de transformation.

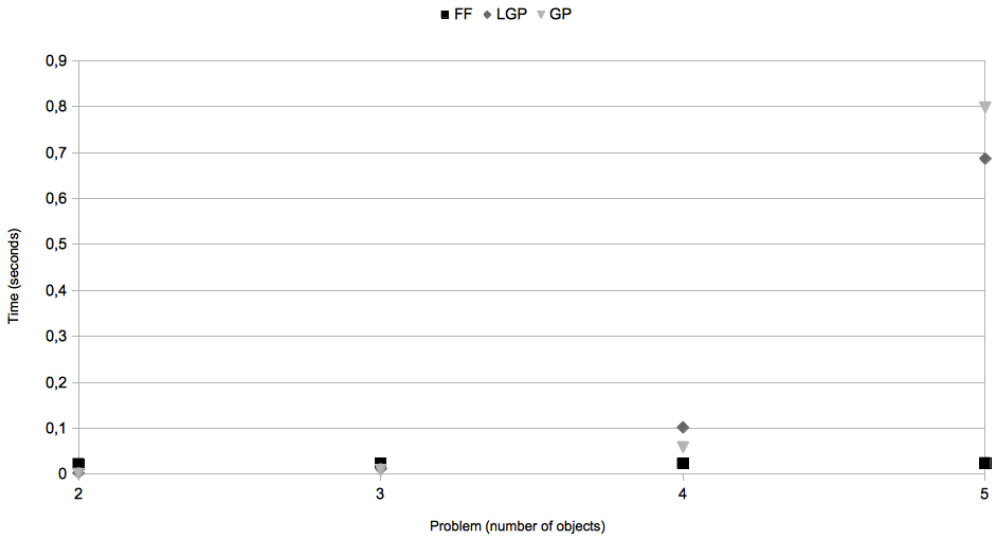


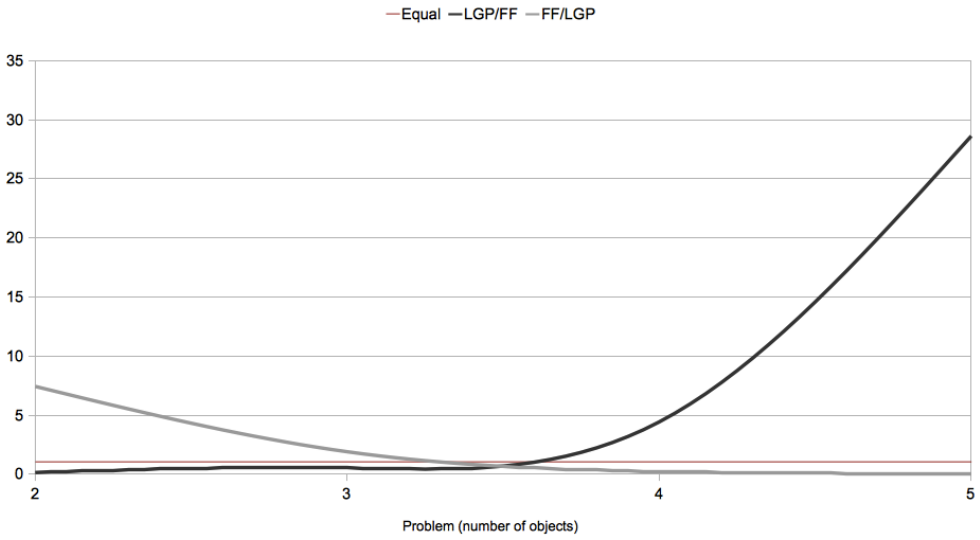
FIGURE 8.33 – La comparaison des temps de planification (en seconde) pour chaque problème issu de HANOI (en nombre de disques, de 2 à 5) pour les planificateurs FF, LGP et GP.

et l'extraction du plan prennent une part de plus en plus importante dans le temps de planification lorsque le nombre de disques augmente. Cela est une caractéristique commune à la résolution par LGP de problèmes issus de domaines linéaires.

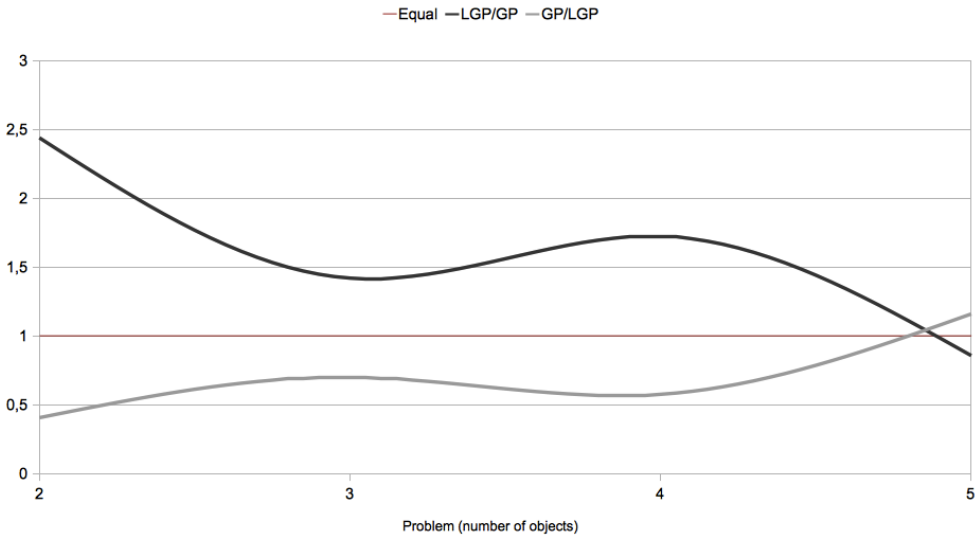
8.4 Conclusion

Les expérimentations menées exposent d'excellentes performances de la part de LGP lorsque le domaine de planification se prête à la construction de structure de choix itératif. Ainsi, LGP est régulièrement plus performant sur ce type de domaine que l'un des meilleurs planificateurs du moment (notamment sur les domaines que nous traitons), FF. Ainsi, le gain d'expressivité des λ-plans de LGP est accompagné d'un gain des performances dans le calcul de ces λ-plans.

Pour les domaines ne permettant pas l'intégration de structure de choix itératif, LGP a les mêmes défauts que GP : ses performances chutent rapidement lorsque le graphe grandit. Mais une solution serait d'utiliser un autre planificateur que LGP (comme FF par exemple) lorsque le graphe de planification ne contient pas ou peu de relations λ-mutex, une forme d'évaluation de la flexibilité potentielle des plans solutions. L'extension du graphe se fait en temps polynomial [BF97], donc cette évaluation devrait être rapide comparée au calcul effectif d'un plan solution.

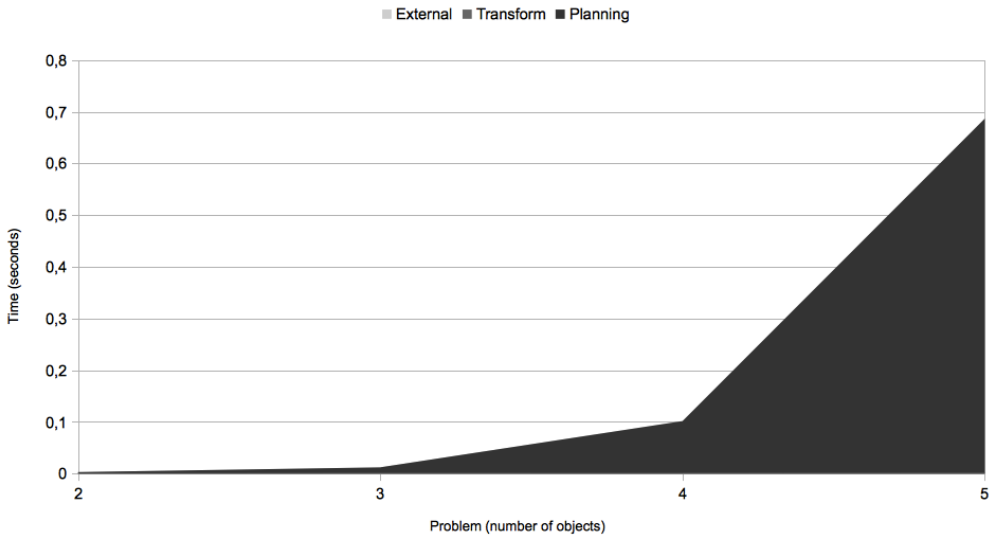


(a)

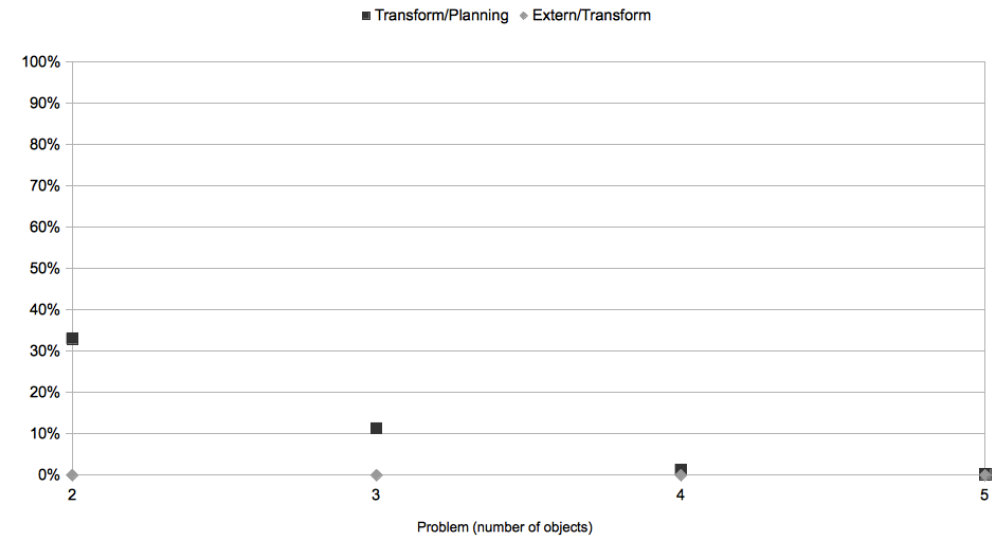


(b)

FIGURE 8.34 – Le rapport entre les temps de planification (en seconde) pour chaque problème issu de HANOI (en nombre de disques, de 2 à 5) :
(a) entre les planificateurs `FF` et `LGP` ;
(b) entre les planificateurs `GP` et `LGP`.



(a)



(b)

FIGURE 8.35 – Comportement du planificateur LGP pour les problèmes issus de HANOI (en nombre de disques, de 2 à 5) :
(a) les temps (en seconde) de planification (Planning), de transformation (Transform) et de fermeture externe (External) ;
(b) les rapports (en pourcentage) du temps de transformation par rapport au temps de planification, et du temps de fermeture externe par rapport au temps de transformation.

Quatrième partie

Épilogue

Chapitre 9

Conclusion et perspectives

Cette thèse apporte un nouveau regard sur la planification, centré sur la flexibilité et donc sur l'adaptabilité et l'adaptivité des plans à l'exécution. Cette préoccupation est à l'opposé de la vision généralement attribuée à la planification : la flexibilité nécessite de trouver dans le système de transition d'états le plus de chemins solutions possible, tandis que la planification tente de calculer un unique chemin le plus efficacement possible et donc explore au minimum le système de transition d'états. Ainsi, la planification doit s'adapter aux attentes de l'utilisateur du plan, et non l'inverse : c'est le défi que nous avons relevé.

La modélisation de la planification flexible que nous proposons en partie II permet de couvrir la problématique de la composition automatique flexible. Les plans sont définis comme une composition de plans, en considérant les actions comme des plans particuliers. La définition de la composition est ainsi conforme avec celles des approches à services, à composant, *etc.* De plus, la distinction entre sémantique opérationnelle et dénotationnelle permet d'un côté de nous préoccuper de l'exécution du plan (liée à la sémantique des opérateurs), et d'un autre côté de vérifier des propriétés sur la composition (comme les effets de la composition dans sa globalité). Notamment, le nombre de chemins disponibles dans le plan (sa largeur) doit être maximisé pour que le plan soit le plus flexible possible. En utilisant les opérateurs de séquence et d'alternative, tous les plans flexibles peuvent en théorie être décrits. Mais la longueur des chemins et la largeur des plans peuvent être illimitées (avec une boucle dans le système de transition d'états par exemple). Ainsi, en pratique, nous proposons de nouveaux opérateurs dont la sémantique est définie par les opérateurs de séquence et d'alternative.

En partie III, les opérateurs d'entrelacement, de permutation et les structures de choix itératif spécifient la flexibilité traitée par notre planificateur. `Λ-GraphPlan` utilise comme base `GraphPlan` de façon à profiter de la représentation condensée des états dans le graphe. L'exploitation des relations λ-mutex permet de découvrir des patterns d'actions candidates pour un traitement spécifique à un ensemble d'objets. `Λ-GraphPlan` extrait un « plan relâché » du graphe, dont les contraintes des relations λ-mutex sont abandonnées. Le plan relâché est ensuite transformé en λ-plan. L'expressivité des λ-plans est unique : aucun planificateur classique de notre connaissance ne parvient à une telle flexibilité. De plus, les expérimentations montrent que `Λ-GraphPlan` est très performant lorsque le problème de planification permet l'intégration de structures de choix itératif. Ces performances sont dues à l'extraction précoce du plan relâché, alliée au faible cout de la transformation. Lorsque le problème de planification n'est pas favorable à `Λ-GraphPlan`, la charge supplémentaire émanante de notre approche est faible vis à vis de `GraphPlan`.

De nombreux travaux, comme [ASW98, SW98, WAS98, GK97], proposent des évolutions de `GraphPlan` pour prendre en compte les incertitudes de l'environnement. Ainsi, nos hypothèses de travail (détaillées au paragraphe 1.2.3 page 10) ne restreignent pas la généralité : nos propositions pourront par la suite être étendues, en optant pour des hypothèses plus proches de notre problématique (la composition automatique flexible en intelligence

ambiante). Nous évoquons d'autres types d'évolutions de Λ-GraphPlan au paragraphe 9.1, et des ouvertures disciplinaires au paragraphe 9.2.

9.1 Prolongements

Cette partie donne quelques pistes à explorer pour prolonger le développement de Λ-GraphPlan.

9.1.1 Évolution des itérations

Les structures de choix itératif dans Λ-GraphPlan sont contraintes : tous les objets doivent être traités de la même façon, sans synchronisation entre structure, le support externe ne peut manipuler les objets, *etc.* La couverture de la flexibilité pourrait être agrandie par l'introduction de nouvelles formes d'itération. Nous donnons ici quelques pistes à développer.

Traitement non uniforme

La recherche de patterns dans le graphe de planification n'est pas complètement exploitée dans cette thèse. En effet, il est envisageable à court terme de prendre en compte des structures plus complexes, et notamment des structures conditionnelles selon les objets traités. Dans Gripper, deux sous-ensembles de balles à déplacer dans deux emplacements différents depuis un même emplacement doivent pouvoir être traités : pour cela, seule la contrainte du traitement uniforme de LGP doit être levée. Ainsi, pour chaque sous-ensemble d'actions λ-mutex secondaires, il est envisageable de traiter les objets différemment. Dans le graphe, cela correspond à des cliques secondaires incomplètes : pour chaque objet de la clique primaire correspond 0 ou 1 objet dans la clique secondaire.

Synchronisation entre traitement

La synchronisation entre traitement s'observe dans le graphe notamment par des actions étiquetées par plusieurs sources comme support interne. Ainsi, ces actions forment le support interne de plusieurs traitements, et devraient être « intégrées » dans chacun des traitements. En fait, il est généralement plus sûr de synchroniser les traitements au niveau de ces actions, car certaines de ces actions peuvent ne pas pouvoir être appliquées successivement par les différents traitements. Par exemple, le déplacement du « robot aux pinces indépendantes » (*cf.* paragraphe 7.4.3) doit être synchronisé pour une utilisation correcte de ses pinces.

9.1.2 Opérateurs de composition

La diversité des opérateurs utilisés par LGP est limitée. La structure de choix itératif est la contribution majeure de la thèse. Il est fort probable que d'autres types de structures soient observables dans le graphe de planification, de la même façon que les actions λ-mutex primaires sont des candidates à former des structures de choix itératif.

Par exemple, lorsque deux actions produisent un même effet, il n'y a pas de raison de choisir l'un plutôt que l'autre au moment de l'extraction du plan relâché. En sélectionnant les deux, il pourrait être possible de proposer l'alternative à l'exécution. Ou encore, après qu'une action est exécutée par le contrôleur, il serait intéressant de lui permettre d'annuler l'exécution lorsque cela est possible. Par exemple, lorsque la balle `b-1` est sélectionnée et attrapée (par l'action `PICK(b-1, r-a)`), le planificateur devrait pouvoir reposer `b-1` pour

finalement choisir `b-2`. La possibilité d'annuler est aisée à chercher dans le graphe : pour chacune des actions a du plan, il faut trouver une action b qui annule exactement les effets de a (autrement dit, $\mathit{eff}^+(b)=\mathit{eff}^-(a)$ et $\mathit{eff}^-(b)=\mathit{eff}^+(a)$). L'action `DROP(b-1, r-a)` vérifie cette propriété vis à vis de `PICK(b-1, r-a)`. Un opérateur d'annulation pourrait être intégré pour représenter ce type d'alternative [1].

La couverture de la flexibilité est encore faible dans `LGP`, et des efforts devraient être consentis dans cette direction : proposer plus de flexibilité sans dégrader les performances de calcul.

9.2 Ouvertures disciplinaires

Dans ce paragraphe, nous proposons trois ouvertures disciplinaires et revenons sur le problème de composition flexible en intelligence ambiante. La première ouverture envisage l'évolution de la planification flexible vers la planification généralisée grâce aux structures de choix itératif. La seconde positionne la planification flexible vis à vis de la robotique, et plus globalement vis à vis de la planification dynamique (c'est-à-dire qui prend en compte l'exécution du plan). La troisième ouverture rejoint notre problématique initiale de composition automatique en intelligence ambiante en la transposant à la composition automatique de processus.

9.2.1 Planification

Le domaine de la planification généralisée a pour objet la résolution d'un ensemble de problèmes. Pour cela, une classe de problème est décrite par des variables dont les valeurs représentent différentes instances de problèmes. Dans notre approche, les λ-plans ont une expressivité telle que des structures itératives manipulent des variables. Par exemple, avec Gripper, une structure de choix itératif manipule l'ensemble des balles `b-1`, `b-2` et `b-3`. En observant le rôle des trois balles dans l'état initial et dans l'objectif de planification, ou en calculant les prédicats des structures de choix itératif, il est envisageable de rendre plus abstraits les λ-plans calculés par Λ-`GraphPlan`. Aussi, plutôt que de limiter les structures de choix itératif aux objets définis par le problème initial, un λ-plan pourrait être appliqué à un ensemble de problèmes, et rejoindre la problématique de la planification généralisée. Pour Gripper, il faudrait découvrir que le λ-plan s'applique sur un ensemble de balles disposées initialement à l'emplacement `r-a`, et que l'objectif est de déplacer l'ensemble des balles dans l'emplacement `r-z`. Ainsi, notre approche appliquée à la planification généralisée permettrait de relâcher les contraintes fortes de ce domaine : la description d'une classe de problèmes devient inutile puisqu'elle serait déduite d'un unique problème, généralisé par notre approche.

9.2.2 Robotique

Nous avons présenté la nécessité de flexibilité lorsque le plan est destiné à un humain (*cf.* chaptire 1), en soulignant le fait qu'un humain est le mieux disposé à sélectionner l'exécution qui lui convient le plus. Ce constat peut être étendu pour n'importe quel *contrôleur*. Le modèle conceptuel de la planification que nous avons exposé précédemment considère qu'il n'y a aucun besoin de prendre en compte les différences entre le modèle du système de transition d'états et le résultat de l'exécution des actions. Du fait que le contrôleur agit

1. La possibilité d'annulation est un critère ergonomique de contrôle explicite. Un plan qui intègre la possibilité d'annuler les effets d'une action rend la largeur du plan infinie : cette introduction apportera de nouvelles discussions sur l'évaluation de la flexibilité des plans.

directement sur le système de transition d'états, c'est-à-dire « en ligne », tandis que le planificateur calcule le plan « hors ligne », l'exécution peut ne pas correspondre à ce qui était prévu lors de la planification. Pour surmonter les incertitudes de l'exécution, la planification et l'application du plan sont alternées [GNT04], notamment en robotique :

Dans ce modèle, une boucle fermée permet au contrôleur de demander au planificateur un nouveau plan en accord avec le *statut* d'exécution : c'est la re-planification. Ce processus est robuste aux incertitudes de l'exécution, c'est-à-dire aux différents contextes d'exécution, mais nécessite de calculer un nouveau plan dès lors que les actions du plan exécuté ne peuvent être appliquées sur l'état courant du système de transition d'états. Les plans flexibles permettent de diminuer le besoin de re-planification en calculant des alternatives dès la planification, en prévoyant à l'avance une multitude de contextes d'exécution : le contrôleur aura plus de facilité à sélectionner une action qui convient à l'état courant que s'il manipule un plan non flexible.

9.2.3 Processus

L'expressivité des plans flexibles s'approche par la définition des opérateurs de composition de l'expressivité décrite en algèbre des processus. Par exemple, CCS [2] utilise l'opérateur « * » pour la séquence, « ? » pour l'alternative et « ‖ » pour la composition parallèle [Bae05]. Ces trois opérateurs sont communs à toute algèbre de processus. Proche de notre problématique, la *synthèse de contrôleur* [RW89] a pour objectif de limiter l'exécution d'un processus pour satisfaire une spécification. C'est alors le contrôleur qui décide, lorsque le processus est contrôlable, des prochains événements à déclencher. Nous proposons une approche différente : composer un processus par planification pour satisfaire une spécification, à partir d'une description des processus. Par le besoin de la flexibilité, nous avons introduit de nouveaux opérateurs de composition dans les plans. Notamment, nos opérateurs « $\gg$ », « $\square$ » et « ‖ » pour la composition de plans ont une sémantique proche de ceux utilisés en algèbre de processus. Ainsi, les travaux en algèbre de processus pourraient être utilisés pour augmenter l'expressivité des compositions (en introduisant de nouveaux opérateurs dans les plans), et nos travaux de planification forment une voie possible pour la composition de processus. Les λ-plans pourraient par exemple être décrits en LOTOS [3] [BB87]. D'ailleurs, LOTOS a inspiré la modélisation des tâches en IHM [PMM97].

9.2.4 Intelligence ambiante

En contexte d'intelligence ambiante, nous soutenons qu'il est nécessaire pour les compositions d'intégrer des structures de contrôle permettant de différer des choix à l'exécution. Notamment, les utilisateurs souhaitent pouvoir contrôler la composition proposée [Gab11]. La flexibilité des compositions est une des dimensions de l'adaptabilité, qui est un des critères d'utilisabilité. Pour que la composition soit conforme aux attentes des utilisateurs [CNM83, ND86], tous les critères ergonomiques [BS93] des systèmes interactifs (formés par composition dans notre cas) doivent pouvoir être respectés. La flexibilité est donc un critère nécessaire mais pas suffisant pour obtenir une composition « centrée utilisateur ». Tous les opérateurs de composition que nous avons introduits sont fondés sur les opérateurs

2. Calculus of Communicating Systems
3. Language Of Temporal Ordering Specification

de séquence et d'alternative, c'est-à-dire sur un des opérateurs temporels et un des opérateurs logiques. Ces deux opérateurs sont suffisants pour décrire la flexibilité, mais d'autres sont nécessaires pour décrire les autres dimensions qualitatives des compositions. Ainsi, la planification nécessite de nouvelles adaptations pour répondre aux attentes des utilisateurs.

Cinquième partie

Annexes

Annexe A

Cas d'études pour la flexibilité

Nous faisons le choix du domaine de planification Gripper pour illustrer la flexibilité recherchée en planification. Mais toute la flexibilité permise par notre approche n'est pas décrite par Gripper. Aussi, les domaines de planification suivants illustres d'autres facettes de la flexibilité recherchée. Un plan flexible permet notamment :

- des itérations successives ;
- des itérations imbriquées ;
- des itérations parallèles.

A.1 Itérations successives avec Gripper-Cart

Pour illustrer les itérations successives, nous ajoutons au domaine Gripper la possibilité pour le robot d'utiliser une remorque pour transporter les balles. Ainsi, une fois la balle `b-1` attrapée, le robot peut la mettre dans la remorque avec l'action `LOAD(b-1)`, et ensuite peut, s'il n'a pas attrapé une autre balle entre temps, récupérer de la remorque la balle `b-1` avec `EMPTY(b-1)`. Ce domaine est décrit en PDDL à l'annexe B.4 page 187. En empêchant le robot de se déplacer avec une balle dans sa pince, le plan attendu doit permettre de choisir dans quel ordre remplir la remorque puis, une fois dans la bonne pièce, permettre de choisir dans quel ordre vider la remorque. Cela se traduit par le plan A.1. Ce plan traduit par deux

Plan A.1 Plan solution pour Gripper-Cart proposant des choix à l'exécution.

```
1  MOVE(r-z, r-a);
2  B ← {b-1, b-2, b-3 };
3  while B is not empty do
4  |   choose ball ∈ B ;
5  |   B ← B − ball ;
6  |   PICK(ball, r-a);
7  |   LOAD(ball);
8  MOVE(r-a, r-z);
9  B ← {b-1, b-2, b-3 };
10 while B is not empty do
11 |   choose ball ∈ B ;
12 |   B ← B − ball ;
13 |   EMPTY(ball);
14 |   DROP(ball, r-z);
```

itérations successives la possibilité pour l'exécuteur de choisir l'ordre de remplissage aussi bien que l'ordre de vidage de la remorque.

A.2 Itérations imbriquées avec Painter

Le domaine PAINTER, décrit en PDDL en annexe B.5 page 187 est utilisé pour illustrer les itérations imbriquées. Cette fois, l'objectif est de peindre des couleurs `black` et `white` chacune des 3 balles `b-1`, `b-2` et `b-3`. Pour cela, 3 actions sont disponibles :

- `CLEAN` permet de nettoyer l'espace de travail, nécessaire avant chaque couche de peinture ;
- `WASH` permet de préparer une des balles `b-1`, `b-2` ou `b-3` avant de la peindre ;
- `PAINT` permet de peindre une des balles d'une couleur donnée.

Le plan A.2 est une solution flexible à ce problème., qui intègre deux itérations. Ainsi,

Plan A.2 Plan solution pour PAINTER proposant des choix à l'exécution.

```
1  B ← {b-1, b-2, b-3 };
2  while B is not empty do
3  |  choose ball ∈ B ;
4  |  B ← B − ball ;
5  |  C ← {black, white };
6  |  while C is not empty do
7  |  |  choose color ∈ C ;
8  |  |  C ← C − color ;
9  |  |  CLEAN();
10 |  |  WASH(ball);
11 |  |  PAINT(ball, color);
```

à l'exécution de ce plan, il est proposé itérativement un premier choix parmi toutes les balles à traiter. Un second choix permet de décider de la couleur à appliquer sur la balle sélectionnée, jusqu'à ce que toutes les couleurs nécessaires aient été appliquées sur la balle. Ainsi, toutes les balles auront été peintes des deux couleurs, mais ni l'ordre des balles, ni l'ordre des couches de peinture n'aura été imposé à l'exécution.

A.3 Itérations parallèles avec Gripper-B+C

Le domaine GRIPPER-B+C est une autre extension de GRIPPER, décrit en PDDL à l'annexe B.6 page 188. Un ensemble de balles et un ensemble de cubes sont entreposés respectivement dans les pièces `r-a` et `r-z`. L'objectif est de déplacer ces balles et ces cubes dans l'autre pièce, au moyen de robots `g-b` et `g-c` spécialisé respectivement dans le déplacement de balles ou de cubes. Le plan A.3 permet, en parallèle, de déplacer les balles et les cubes, chacun des robots s'occupant de déplacer de façon indépendante l'ensemble d'objets dont il a la charge. Ainsi, à l'exécution, l'ordre de déplacement des balles et l'ordre de déplacement

Plan A.3 Plan solution pour GRIPPER-B+C proposant des choix à l'exécution.

```
1  B ← {b-1, b-2, b-3 };                    C ← {c-1, c-2, c-3 };
2  while B is not empty do                  while C is not empty do
3  |  MOVE(g-b, r-z, r-a);                 |  MOVE(g-c, r-a, r-z);
4  |  choose ball ∈ B ;                     |  choose cube ∈ C ;
5  |  B ← B − ball ;                        |  C ← C − cube ;
6  |  PICK-BALL(g-b, ball, r-a);            |  PICK-BOX(g-c, cube, r-z);
7  |  MOVE(g-b, r-a, r-z);                  |  MOVE(g-c, r-z, r-a);
8  |  DROP-BALL(g-b, ball, r-z);            |  DROP-BOX(g-c, cube, r-a);
```

des cubes sont libres de choix. La partie gauche du plan donne la possibilité de déplacer les balles, la partie droite de déplacer les cubes, chacune de ces parties étant indépendante de l'autre et donc exécutable en parallèle, sans aucune contrainte d'ordre entre elles.

Annexe B

Domaines et problèmes de planification

B.1 Checker

```
(define (domain Checker)
    (:requirements :strips)
    (:predicates
         (ball ?b)
         (checked ?b)
         (ready)
    (:action CHECK
         :parameters (?obj)
         :precondition (and (ball ?ball) (ready))
         :effect (and (checked ?ball) (not (ready))))
    (:action TARE
         :parameters ()
         :precondition (and )
         :effect (and (ready)))
```

```
(define (problem pb3)
    (:domain Checker)
    (:requirements :strips)
    (:objects b-1 b-2 b-3)
    (:init (ball b-1) (ball b-2) (ball b-3) (ready))
    (:goal (and (checked b-1) (checked b-2) (checked b-3))))
```

B.2 Loader

```
(define (domain Loader)
    (:requirements :strips)
    (:predicates
         (ball ?b)
         (in-box ?b)
         (free)
         (carry ?b))
    (:action PICK
         :parameters (?obj)
```

```
        :precondition (and (ball ?obj) (free))
        :effect (and (carry ?obj) (not (free))))
    (:action LOAD
        :parameters (?obj)
        :precondition (and (carry ?obj))
        :effect (and (free) (not (carry ?obj)) (in-box ?obj)))
    (:action DROP
        :parameters (?obj)
        :precondition (and (carry ?obj))
        :effect (and (free) (not (carry ?obj))))
```

```
(define (problem PB3)
    (:domain LOADER)
    (:requirements :strips)
    (:objects b-1 b-2 b-3)
    (:init (ball b-1) (ball b-2) (ball b-3) (free))
    (:goal (and (in-box b-1) (in-box b-2) (in-box b-3))))
```

B.3 Gripper

```
(define (domain GRIPPER)
    (:requirements :strips)
    (:predicates
        (room ?room) (ball ?ball)
        (g-at ?room) (at ?ball ?room)
        (free) (carry ?ball))
    (:action MOVE
        :parameters (?from ?to)
        :precondition (and (room ?from) (room ?to) (g-at ?from))
        :effect (and (g-at ?to) (not (g-at ?from))))
    (:action PICK
        :parameters (?ball ?room)
        :precondition (and (ball ?ball) (room ?room) (g-at ?room)
                           (at ?ball ?room) (free))
        :effect (and (carry ?ball) (not (at ?ball ?room)) (not (free))))
    (:action DROP
        :parameters (?ball ?room)
        :precondition (and (ball ?ball) (room ?room) (g-at ?room)
                           (carry ?ball))
        :effect (and (not (carry ?ball)) (at ?ball ?room) (free))))
```

```
(define (problem PB3)
    (:domain GRIPPER)
    (:requirements :strips)
    (:objects r-a r-z b-1 b-2 b-3)
    (:init (room r-a) (room r-z) (ball b-1) (ball b-2) (ball b-3)
           (g-at r-a) (free) (at b-1 r-a) (at b-2 r-a) (at b-3 r-a))
    (:goal (and (at b-1 r-z) (at b-2 r-z) (at b-3 r-z))))
```

B.4 Gripper-Cart

```
(define (domain GRIPPER-CART)
    (:requirements :strips)
    (:predicates
         (room ?room) (ball ?ball)
         (g-at ?room) (at ?ball ?room)
         (free) (carry ?ball))
    (:action MOVE
         :parameters (?from ?to)
         :precondition (and (room ?from) (room ?to) (g-at ?from))
         :effect (and (g-at ?to) (not (g-at ?from))))
    (:action PICK
         :parameters (?ball ?room)
         :precondition (and (ball ?ball) (room ?room) (g-at ?room)
                            (at ?ball ?room) (free))
         :effect (and (carry ?ball) (not (at ?ball ?room)) (not (free))))
    (:action DROP
         :parameters (?ball ?room)
         :precondition (and (ball ?ball) (room ?room) (g-at ?room)
                            (carry ?ball))
         :effect (and (not (carry ?ball)) (at ?ball ?room) (free))))
    (:action LOAD
         :parameters (?obj)
         :precondition (and (ball ?obj) (carry ?obj))
         :effect (and (not (carry ?obj)) (in-cart ?obj) (free)))
    (:action EMPTY
         :parameters (?obj)
         :precondition (and (ball ?obj) (in-cart ?obj) (free))
         :effect (and (carry ?obj) (not (in-cart ?obj)) (not (free))))


(define (problem PB3)
    (:domain GRIPPER-CART)
    (:requirements :strips)
    (:objects r-a r-z b-1 b-2 b-3)
    (:init (room r-a) (room r-z) (ball b-1) (ball b-2) (ball b-3)
           (g-at r-a) (free) (at b-1 r-a) (at b-2 r-a) (at b-3 r-a))
    (:goal (and (at b-1 r-z) (at b-2 r-z) (at b-3 r-z))))
```

B.5 Painter

```
(define (domain PAINTER)
    (:requirements :strips)
    (:predicates
         (ball ?ball) (ready ?ball)
         (color ?color) (painted ?ball ?color)
         (clean))
    (:action WASH
         :parameters (?ball)
```

```
        :precondition (and (ball ?ball) (clean))
        :effect (and (ready ?ball) (not (clean))))
    (:action PAINT
        :parameters (?ball ?color)
        :precondition (and (ball ?ball) (color ?color) (ready ?ball))
        :effect (and (painted ?ball ?color) (not (ready ?ball))))
    (:action CLEAN
        :parameters ()
        :precondition (and )
        :effect (and (clean)))


(define (problem PB3X2)
    (:domain PAINTER)
    (:requirements :strips)
    (:objects b-1 b-2 b-3)
    (:init (ball b-1) (ball b-2) (ball b-3)
           (color black) (color grey) (color white)
    (:goal (and (painted b-1 black) (painted b-1 grey) (painted b-1 white)
                (painted b-2 black) (painted b-2 grey) (painted b-2 white)
                (painted b-3 black) (painted b-3 grey) (painted b-3 white))))
```

B.6 Gripper-B+C

```
(define (domain GRIPPER-B+C)
    (:requirements :strips)
    (:predicates
        (room ?room) (ball ?ball) (cube ?cube) (at ?obj ?room)
        (g-ball ?g) (g-cube ?g) (g-at ?g ?room)
        (free ?g) (carry ?g ?pbj))
    (:action MOVE
        :parameters (?g ?from ?to)
        :precondition (and (room ?from) (room ?to) (g-at ?g ?from))
        :effect (and (g-at ?g ?to) (not (g-at ?g ?from))))
    (:action PICK-BALL
        :parameters (?g ?ball ?room)
        :precondition (and (ball ?ball) (room ?room) (g-ball ?g)
                           (g-at ?g ?room) (at ?ball ?room) (free ?g))
        :effect (and (carry ?g ?ball) (not (at ?ball ?room)) (not (free ?g))))
    (:action DROP-BALL
        :parameters (?g ?ball ?room)
        :precondition (and (ball ?ball) (room ?room) (g-ball ?g)
                           (g-at ?g ?room) (carry ?g ?ball))
        :effect (and (not (carry ?g ?ball)) (at ?ball ?room) (free ?g))))
    (:action PICK-BOX
        :parameters (?g ?cube ?room)
        :precondition (and (cube ?cube) (room ?room) (g-cube ?g)
                           (g-at ?g ?room) (at ?cube ?room) (free ?g))
        :effect (and (carry ?g ?cube) (not (at ?cube ?room)) (not (free ?g))))
    (:action DROP-BOX
        :parameters (?g ?cube ?room)
        :precondition (and (cube ?cube) (room ?room) (g-cube ?g)
```

```
                        (g-at ?g ?room) (carry ?g ?cube))
        :effect (and (not (carry ?g ?cube)) (at ?cube ?room) (free ?g))))


(define (problem pb3)
    (:domain Gripper-B+C)
    (:requirements :strips)
    (:objects r-a r-z
              b-1 b-2 b-3
              c-1 c-2 c-3
              g-b g-c)
    (:init (room r-a) (room r-z)
           (ball b-1) (ball b-2) (ball b-3)
           (cube c-1) (cube c-2) (cube c-3)
           (g-ball g-b) (g-cube g-c)
           (g-at g-b r-a) (g-at g-c r-z) (free g-b) (free g-c)
           (at b-1 r-a) (at b-2 r-a) (at b-3 r-a)
           (at c-1 r-z) (at c-2 r-z) (at c-3 r-z))
    (:goal (and (at b-1 r-z) (at b-2 r-z) (at b-3 r-z)
                (at c-1 r-a) (at c-2 r-a) (at c-3 r-a))))
```

B.7 BlocksWorld

```
(define (domain BlocksWorld)
    (:requirements :strips)
    (:predicates
         (clear ?x)
         (on-table ?x)
         (free)
         (holding ?x)
         (on ?x ?y))
    (:action PICKUP
         :parameters (?obj)
         :precondition (and (free) (clear ?obj) (on-table ?obj))
         :effect (and (not (free)) (not (clear ?obj)) (not (on-table ?obj))
                      (holding ?obj)))
    (:action PUTDOWN
         :parameters (?obj)
         :precondition (and (holding ?obj))
         :effect (and (clear ?obj) (free) (on-table ?obj)
                      (not (holding ?obj))))
    (:action STACK
         :parameters (?obj ?under)
         :precondition (and (clear ?under) (holding ?obj))
         :effect (and (free) (clear ?obj) (on ?obj ?under)
                      (not (clear ?under)) (not (holding ?obj))))
    (:action UNSTACK
         :parameters (?obj ?under)
         :precondition (and (on ?obj ?under) (clear ?obj) (free))
         :effect (and (not (free)) (not (clear ?obj)) (not (on ?obj ?under))
                      (clear ?under) (holding ?obj))))
```

```
(define (problem PB3)
    (:domain BLOCKSWORLD)
    (:requirements :strips)
    (:objects c-1 c-2 c-3)
    (:init (free)
           (on-table c-3)
           (on c-2 c-3)
           (on c-1 c-2)
           (clear c-1))
    (:goal (and (on-table c-3)
                (on-table c-2)
                (on-table c-1))))
```

B.8 Hanoi

```
(define (domain HANOI)
    (:requirements :strips)
    (:predicates
         (clear ?x)
         (on ?x ?y)
         (smaller ?x ?y))
    (:action MOVE
         :parameters (?disk ?from ?to)
         :precondition (and (smaller ?to ?disk)
                            (on ?disk ?from)
                            (clear ?disk)
                            (clear ?to))
         :effect (and (clear ?from)
                      (on ?disk ?to)
                      (not (on ?disk ?from))
                      (not (clear ?to)))))
```

```
(define (problem PB3)
    (:domain HANOI)
    (:requirements :strips)
    (:objects p-a p-i p-z d-1 d-2 d-3)
    (:init (smaller p-a d-1) (smaller p-a d-2) (smaller p-a d-3)
           (smaller p-i d-1) (smaller p-i d-2) (smaller p-i d-3)
           (smaller p-z d-1) (smaller p-z d-2) (smaller p-z d-3)
           (smaller d-3 d-1) (smaller d-3 d-2) (smaller d-2 d-1)
           (clear p-i)
           (clear p-z)
           (clear d-1)
           (on d-3 p-a)
           (on d-2 d-3)
           (on d-1 d-2))
    (:goal (and (on d-3 p-z)
                (on d-2 d-3)
                (on d-1 d-2))))
```

Annexe C

Propriétés des opérateurs

C.1 Opérateurs de séquence entre plans

L'élément neutre de la séquence est l'action nulle, ϵ.

Loi 1 (élément neutre de $\gg$)**.** *Soit un plan* π*, alors* $\pi \gg \epsilon = \epsilon \gg \pi = \pi$*.*

Démonstration. Soit un plan π, alors :

$$\begin{aligned}
pre(\pi \gg \epsilon) &= pre(\pi) \cup \big(pre(\epsilon) \setminus \mathit{eff}^+(\pi)\big) \\
&= pre(\pi) \cup \big(\{\} \setminus \mathit{eff}^+(\pi)\big) \\
&= pre(\pi) \\
\mathit{eff}^+(\pi \gg \epsilon) &= \big(\mathit{eff}^+(\pi) \setminus \mathit{eff}^-(\epsilon)\big) \cup \mathit{eff}^+(\epsilon) \\
&= \big(\mathit{eff}^+(\pi) \setminus \{\}\big) \cup \{\} \\
&= \mathit{eff}^+(\pi) \\
\mathit{eff}^-(\pi \gg \epsilon) &= \big(\mathit{eff}^-(\pi) \setminus \mathit{eff}^+(\epsilon)\big) \cup \mathit{eff}^-(\epsilon) \\
&= \big(\mathit{eff}^-(\pi) \setminus \{\}\big) \cup \{\} \\
&= \mathit{eff}^-(\pi)
\end{aligned}$$

Donc $\pi \gg \epsilon = \pi$.

La preuve est similaire pour $\epsilon \gg \pi = \pi$.

Ainsi, ϵ est l'élément neutre de $\gg$. □

La séquence n'est pas idempotente.

Loi 2 (non idempotence de $\gg$)**.** *Soit un plan* π*, alors* $\pi \gg \pi \not\Longrightarrow \pi$*.*

Démonstration. Triviale avec un plan π tel que $pre(\pi) \cap \mathit{eff}^-(\pi) \neq \emptyset$: une première application de π est définie sur un état satisfaisant les préconditions, mais deux applications successives ne le sont pas. Ainsi, $\gg$ n'est pas idempotente. □

Nous ne pouvons pas distinguer le comportement de deux plans en séquence suivis d'un troisième plan, de celui de ces trois plans en séquence.

Loi 3 (associativité de $\gg$)**.** *Soient* π_1*,* π_2 *et* π_3 *des plans,* $(\pi_1 \gg \pi_2) \gg \pi_3 = \pi_1 \gg (\pi_2 \gg \pi_3) = \pi_1 \gg \pi_2 \gg \pi_3$*.*

Démonstration. Soient trois plans π_1, π_2 et π_3.

Notons $\pi_t = \pi_1 \gg \pi_2$ et $\pi = \pi_t \gg \pi_3$. Par définition :

$$\begin{aligned}
pre(\pi_t) &= pre(\pi_1) \cup \big(pre(\pi_2) \setminus \mathit{eff}^+(\pi_1)\big) \\
\mathit{eff}^+(\pi_t) &= \big(\mathit{eff}^+(\pi_1) \setminus \mathit{eff}^-(\pi_2)\big) \cup \mathit{eff}^+(\pi_2) \\
\mathit{eff}^-(\pi_t) &= \big(\mathit{eff}^-(\pi_1) \setminus \mathit{eff}^+(\pi_2)\big) \cup \mathit{eff}^-(\pi_2)
\end{aligned}$$

Or $\mathit{eff}^-(\pi_2) \cap \mathit{eff}^+(\pi_2) = \emptyset$, donc :

$$\begin{aligned}\mathit{eff}^+(\pi_t) &= \big(\mathit{eff}^+(\pi_1) \cup \mathit{eff}^+(\pi_2)\big) \setminus \mathit{eff}^-(\pi_2)\\ \mathit{eff}^-(\pi_t) &= \big(\mathit{eff}^-(\pi_1) \cup \mathit{eff}^-(\pi_2)\big) \setminus \mathit{eff}^+(\pi_2)\end{aligned}$$

De la même façon, nous avons :

$$\begin{aligned}\mathit{pre}(\pi) &= \mathit{pre}(\pi_t) \cup \big(\mathit{pre}(\pi_3) \setminus \mathit{eff}^+(\pi_t)\big)\\ &= \big(\mathit{pre}(\pi_1) \cup \big(\mathit{pre}(\pi_2) \setminus \mathit{eff}^+(\pi_1)\big)\big) \cup \big(\mathit{pre}(\pi_3) \setminus \big(\big(\mathit{eff}^+(\pi_1) \cup \mathit{eff}^+(\pi_2)\big) \setminus \mathit{eff}^-(\pi_2)\big)\big)\end{aligned}$$

Si $\mathit{eff}^-(\pi_2) \cap \mathit{pre}(\pi_3) \neq \emptyset$, $\pi_1 \gg \pi_2 \gg \pi_3$ est indéfini et la preuve de la loi est triviale, sinon :

$$= \big(\mathit{pre}(\pi_1) \cup \big(\mathit{pre}(\pi_2) \setminus \mathit{eff}^+(\pi_1)\big)\big) \cup \big(\mathit{pre}(\pi_3) \setminus \big(\mathit{eff}^+(\pi_1) \cup \mathit{eff}^+(\pi_2)\big)\big)$$

Par factorisation de la soustraction de $\mathit{eff}^+(\pi_1)$:

$$\begin{aligned}&= \mathit{pre}(\pi_1) \cup \big(\mathit{pre}(\pi_2) \cup \big(\mathit{pre}(\pi_3) \setminus \mathit{eff}^+(\pi_2)\big)\big) \setminus \mathit{eff}^+(\pi_1)\\ \mathit{eff}^+(\pi) &= \big(\mathit{eff}^+(\pi_t) \setminus \mathit{eff}^-(\pi_3)\big) \cup \mathit{eff}^+(\pi_3)\\ &= \big(\big(\big(\mathit{eff}^+(\pi_1) \setminus \mathit{eff}^-(\pi_2)\big) \cup \mathit{eff}^+(\pi_2)\big) \setminus \mathit{eff}^-(\pi_3)\big) \cup \mathit{eff}^+(\pi_3)\\ \mathit{eff}^-(\pi) &= \big(\mathit{eff}^-(\pi_t) \setminus \mathit{eff}^+(\pi_3)\big) \cup \mathit{eff}^-(\pi_3)\\ &= \big(\big(\big(\mathit{eff}^-(\pi_1) \setminus \mathit{eff}^+(\pi_2)\big) \cup \mathit{eff}^-(\pi_2)\big) \setminus \mathit{eff}^+(\pi_3)\big) \cup \mathit{eff}^-(\pi_3)\end{aligned}$$

Notons $\pi_q = \pi_2 \gg \pi_3$ et $\pi' = \pi_1 \gg \pi_q$. Nous obtenons :

$$\begin{aligned}\mathit{pre}(\pi_q) &= \mathit{pre}(\pi_2) \cup \big(\mathit{pre}(\pi_3) \setminus \mathit{eff}^+(\pi_2)\big)\\ \mathit{eff}^+(\pi_q) &= \big(\mathit{eff}^+(\pi_2) \setminus \mathit{eff}^-(\pi_3)\big) \cup \mathit{eff}^+(\pi_3)\\ \mathit{eff}^-(\pi_q) &= \big(\mathit{eff}^-(\pi_2) \setminus \mathit{eff}^+(\pi_3)\big) \cup \mathit{eff}^-(\pi_3)\end{aligned}$$

De la même façon, nous avons :

$$\begin{aligned}\mathit{pre}(\pi') &= \mathit{pre}(\pi_1) \cup (\mathit{pre}(\pi_q) \setminus \mathit{eff}^+(\pi_1))\\ &= \mathit{pre}(\pi_1) \cup \big(\mathit{pre}(\pi_2) \cup \big(\mathit{pre}(\pi_3) \setminus \mathit{eff}^+(\pi_2)\big)\big) \setminus \mathit{eff}^+(\pi_1)\\ &= \mathit{pre}(\pi)\\ \mathit{eff}^+(\pi') &= \big(\mathit{eff}^+(\pi_1) \setminus \mathit{eff}^-(\pi_q)\big) \cup \mathit{eff}^+(\pi_q)\\ &= \big(\mathit{eff}^+(\pi_1) \setminus \big(\big(\mathit{eff}^-(\pi_2) \setminus \mathit{eff}^+(\pi_3)\big) \cup \mathit{eff}^-(\pi_3)\big)\big) \cup \big(\mathit{eff}^+(\pi_2) \setminus \mathit{eff}^-(\pi_3)\big) \cup \mathit{eff}^+(\pi_3)\end{aligned}$$

Par simplification de $\mathit{eff}^+(\pi_3)$:

$$= \big(\mathit{eff}^+(\pi_1) \setminus \big(\mathit{eff}^-(\pi_2) \cup \mathit{eff}^-(\pi_3)\big)\big) \cup \big(\mathit{eff}^+(\pi_2) \setminus \mathit{eff}^-(\pi_3)\big) \cup \mathit{eff}^+(\pi_3)$$

Par factorisation de la soustraction de $\mathit{eff}^-(\pi_3)$:

$$\begin{aligned}&= \big(\big(\big(\mathit{eff}^+(\pi_1) \setminus \mathit{eff}^-(\pi_2)\big) \cup \mathit{eff}^+(\pi_2)\big) \setminus \mathit{eff}^-(\pi_3)\big) \cup \mathit{eff}^+(\pi_3)\\ &= \mathit{eff}^+(\pi)\\ \mathit{eff}^-(\pi') &= \big(\mathit{eff}^-(\pi_1) \setminus \mathit{eff}^+(\pi_q)\big) \cup \mathit{eff}^-(\pi_q)\\ &= \big(\mathit{eff}^-(\pi_1) \setminus \big(\big(\mathit{eff}^+(\pi_2) \setminus \mathit{eff}^-(\pi_3)\big) \cup \mathit{eff}^+(\pi_3)\big)\big) \cup \big(\mathit{eff}^-(\pi_2) \setminus \mathit{eff}^+(\pi_3)\big) \cup \mathit{eff}^-(\pi_3)\end{aligned}$$

	neutre	idempotence	associativité	commutativité
$\gg$	ϵ	non	oui	non

TABLE C.1 – Lois algébriques de l'opérateur de séquence entre plans.

Par simplification de $\mathit{eff}^-(\pi_3)$:

$$= \big(\mathit{eff}^-(\pi_1) \setminus \big(\mathit{eff}^+(\pi_2) \cup \mathit{eff}^+(\pi_3)\big)\big) \cup \big(\mathit{eff}^-(\pi_2) \setminus \mathit{eff}^+(\pi_3)\big) \cup \mathit{eff}^-(\pi_3)$$

Par factorisation de la soustraction de $\mathit{eff}^+(\pi_3)$:

$$= \big(\big(\big(\mathit{eff}^-(\pi_1) \setminus \mathit{eff}^+(\pi_2)\big) \cup \mathit{eff}^-(\pi_2)\big) \setminus \mathit{eff}^+(\pi_3)\big) \cup \mathit{eff}^-(\pi_3)$$
$$= \mathit{eff}^-(\pi)$$

Ainsi, $\pi = \pi'$, donc la séquence est associative. □

La séquence n'est pas commutative.

Loi 4 (non commutativité de $\gg$)**.** *Soient deux plans π_1 et π_2, alors $\pi_1 \gg \pi_2 \implies \pi_2 \gg \pi_1$.*

Démonstration. Triviale avec deux plans π_1 et π_2 tels que $\mathit{eff}^+(\pi_1) \cap \mathit{eff}^-(\pi_2) \neq \emptyset$: on a alors $\pi_1 \gg \pi_2 \neq \pi_2 \gg \pi_1$ et donc $\gg$ n'est pas commutative. □

La table C.1 résume les lois qui régissent l'opérateur de séquence. Ces lois impliquent les règles suivantes :

$$\pi \gg \epsilon = \pi = \epsilon \gg \pi$$

ϵ est l'élément neutre de la séquence

$$(\pi \gg \pi') \gg \pi'' = \pi \gg (\pi' \gg \pi'')$$

associativité de la séquence

Les règle de l'opérateur binaire de séquence nous permet d'écrire des règles généralisées à l'opérateur n-aire :

$$\gg (\cdots, \pi_i, \epsilon, \pi_j, \cdots) = \gg (\cdots, \pi_i, \pi_j, \cdots)$$

ϵ est l'élément neutre de la séquence

$$\gg \big(\gg (\pi_{1_1}, \cdots, \pi_{1_n}), \cdots, \gg (\pi_{k_1}, \cdots, \pi_{k_m})\big) = \gg (\pi_{1_1}, \cdots, \pi_{1_n}, \cdots, \pi_{k_1}, \cdots, \pi_{k_m})$$

associativité de la séquence

Cette dernière règle est utile pour montrer qu'il existe une « forme normale » de tout plan π composé par l'opérateur de séquence sur les actions de $\mathcal{A}$: $\pi = \gg (a_1, \cdots, a_n)$ avec $\forall i \in [1..n], a_i \in \mathcal{A}$.

C.2 Opérateurs de séquence entre polyplans

C.2.1 Lois algébriques

L'opérateur de séquence entre polyplan admet un élément neutre, ϵ.

Loi 5 (élément neutre de $\gg$)**.** *Soit un polyplan* Π*, alors* $\Pi \gg \epsilon = \epsilon \gg \Pi = \Pi$*.*

Démonstration. Semblable à la preuve de la loi 1, mais pour chaque couple d'effets positifs et négatifs. □

Nous ne pouvons pas distinguer le comportement de deux polyplans en séquence suivis d'un troisième polyplan, de celui de ces trois polyplans en séquence.

Loi 6 (associativité de $\gg$)**.** *Soient trois plans flexibles* Π_1*,* Π_2 *et* Π_3*, alors* $(\Pi_1 \gg \Pi_2) \gg \Pi_3 = \Pi_1 \gg (\Pi_2 \gg \Pi_3) = \Pi_1 \gg \Pi_2 \gg \Pi_3$*.*

Démonstration. Semblable à la preuve de la loi 3, mais pour chaque couple d'effets positifs et négatifs. □

La table C.2 définit les lois qui régissent l'opérateur de séquence. Ces lois impliquent les

	neutre	idempotence	associativité	commutativité
$\gg$	ϵ	non	oui	non

TABLE C.2 – Lois algébriques de l'opérateur de séquence entre polyplan.

égalités qui suivent.

$$\Pi \gg \epsilon = \Pi$$

ϵ est l'élément neutre de la séquence

$$(\Pi \gg \Pi') \gg \Pi'' = \Pi \gg (\Pi' \gg \Pi'')$$

associativité de la séquence

Comme l'opérateur de séquence est associatif, nous pouvons omettre les parenthèses lorsqu'elles ne sont pas nécessaires pour indiquer la portée des opérateurs. Les règle de l'opérateur binaire de séquence nous permet d'écrire des règles généralisées à l'opérateur n-aire :

$$\gg (\cdots, \Pi_i, \epsilon, \Pi_j, \cdots) = \gg (\cdots, \Pi_i, \Pi_j, \cdots)$$

ϵ est l'élément neutre de la séquence

$$\gg (\gg (\Pi_{1_1}, \cdots, \Pi_{1_n}), \cdots, \gg (\Pi_{k_1}, \cdots, \Pi_{k_m})) = \gg (\Pi_{1_1}, \cdots, \Pi_{1_n}, \cdots, \Pi_{k_1}, \cdots, \Pi_{k_m})$$

associativité de la séquence

C.2.2 Évaluation de la largeur

Soient Π' et Π'' deux polyplans tels que $\#\Pi' = k$ et $\#\Pi'' = n$, et le polyplan $\Pi = \Pi' \gg \Pi''$, alors $\#\Pi = k \times n$.

Démonstration. Trivialement déduit de la définition de l'opérateur de séquence : $\forall\,(i,j) \in [1..k] \times [1..n]$, $\mathit{eff}_{i,j}(\Pi)$ est défini sur un plan i dérivé de Π' et sur un plan j dérivé de Π''. □

Si $k = 1$, alors la largeur de la séquence entre les deux polyplans est limitée par la largeur du polyplan Π'', et réciproquement si $n = 1$.

C.3 Opérateur d'alternative entre polyplans

C.3.1 Lois algébriques

Faire le choix entre deux mêmes polyplans revient à ne pas faire de choix.

Loi 7 (idempotence de $\square$)**.** *Soit* Π *un polyplan,* $(\Pi \square \Pi) = \Pi$.

Démonstration. Trivialement déduit de la sémantique opérationnelle de l'opérateur d'alternative. □

Pour ne pas surcharger les notations, cela n'a pas été formalisé dans la sémantique dénotationnelle. Pour être conforme à cette loi, il faut empêcher l'ajout d'un plan déjà exprimé dans un plan flexible.

L'ordre de présentation des alternative est sans importance.

Loi 8 (symétrie de $\square$)**.** *Soient* Π_1 *et* Π_2 *deux polyplans,* $(\Pi_1 \square \Pi_2) = (\Pi_2 \square \Pi_1)$.

Démonstration. Trivialement déduit de la sémantique opérationnelle de l'opérateur d'alternative. □

Un choix entre trois polyplans peut être éclaté en deux choix binaires successifs, l'ordre étant sans importance, et la réciproque est vraie.

Loi 9 (associativité de $\square$)**.** *Soient* Π_1, Π_2 *et* Π_3 *trois polyplans,* $(\Pi_1 \square \Pi_2) \square \Pi_3 = \Pi_1 \square (\Pi_2 \square \Pi_3) = (\Pi_1 \square \Pi_2 \square \Pi_3)$.

Démonstration. Trivialement déduit de la sémantique opérationnelle de l'opérateur d'alternative. □

La table C.3 définit les lois qui régissent l'opérateur d'alternative. Ces lois impliquent

	neutre	idempotence	associativité	commutativité
$\square$	non	oui	oui	oui

TABLE C.3 – Lois algébriques de l'opérateur d'alternative.

les égalités qui suivent.

$$\Pi \square \epsilon \neq \Pi$$

ϵ n'est pas l'élément neutre de l'alternative

$$\Pi \square \Pi = \Pi$$

idempotence de l'alternative

$$(\Pi \square \Pi') \square \Pi'' = \Pi \square (\Pi' \square \Pi'')$$

associativité de l'alternative

$$\Pi \square \Pi' = \Pi' \square \Pi$$

commutativité de l'alternative

Comme l'opérateur d'alternative est associatif, nous pouvons omettre les parenthèses lorsqu'elles ne sont pas nécessaires pour indiquer la portée des opérateurs. Les règle de l'opérateur binaire d'alternative nous permet d'écrire des règles généralisées à l'opérateur n-aire :

$$\square\ (\cdots, \Pi_i, \epsilon, \Pi_j, \cdots) \neq \square\ (\cdots, \Pi_i, \Pi_j, \cdots)$$

ϵ n'est pas l'élément neutre de l'alternative

$$\square\ (\Pi, \cdots, \Pi) = \Pi$$

idempotence de l'alternative

$$\square\ \Big(\square\ (\Pi_{1_1}, \cdots, \Pi_{1_n}), \cdots, \square\ (\Pi_{k_1}, \cdots, \Pi_{k_m})\Big) = \square\ (\Pi_{1_1}, \cdots, \Pi_{1_n}, \cdots, \Pi_{k_1}, \cdots, \Pi_{k_m})$$

associativité de l'alternative

$$\square\ (\cdots, \Pi_i, \Pi_j, \cdots) = \square\ (\cdots, \Pi_j, \Pi_i, \cdots)$$

commutativité de l'alternative

C.3.2 Évaluation de la largeur

Soient Π' et Π'' deux polyplans tels que $\#\Pi' = k$ et $\#\Pi'' = n$, et le polyplan $\Pi = \Pi' \square \Pi''$, alors $\#\Pi = k + n$.

Démonstration. Trivialement déduit de la définition de l'opérateur d'alternative : $\forall i \in [1..k], \mathit{eff}_i(\Pi) = \mathit{eff}_i(\Pi')$ et $\forall j \in [k+1..k+n], \mathit{eff}_j(\Pi) = \mathit{eff}_{j-k}(\Pi'')$. □

La largeur du choix entre les deux polyplans est supérieure à la largeur des deux polyplans Π' et Π''.

C.4 Opérateur d'entrelacement entre polyplans

C.4.1 Lois algébriques

La première loi algébrique régissant l'entrelacement est déduit de la définition de l'opérateur, qui permet d'affirmer que l'ordre du parallélisme ne modifie pas sa sémantique.

Loi 10 (symétrie de $\|$)**.** *Soient* Π_1 *et* Π_2 *deux polyplans,* $(\Pi_1 \parallel \Pi_2) = (\Pi_2 \parallel \Pi_1)$.

L'entrelacement de trois polyplans peut être éclaté en deux entrelacement binaires successifs, l'ordre étant sans importance, et la réciproque est vraie.

Loi 11 (associativité de $\|$)**.** *Soient* Π_1, Π_2 *et* Π_3 *trois polyplans,* $(\Pi_1 \parallel \Pi_2) \parallel \Pi_3 = \Pi_1 \parallel (\Pi_2 \parallel \Pi_3) = (\Pi_1 \parallel \Pi_2 \parallel \Pi_3)$.

L'associativité de l'opérateur nous permet de nous dispenser des parenthèses.
Nous déduisons de la définition de l'opérateur d'entrelacement les lois de la table C.4.

	neutre	idempotence	associativité	commutativité
$\|\|$	ϵ	non	oui	oui

TABLE C.4 – Lois algébriques de l'opérateur d'entrelacement.

C.4.2 Évaluation de la largeur

Le nombre de choix offert par l'entrelacement de deux plans augmente de façon exponentielle par rapport à la longueur des plans. En effet, le nombre de plans déduit du parallélisme entre $\pi = a_1 \gg \cdots \gg a_k$ et $\pi' = a'_1 \gg \cdots \gg a'_n$, c'est-à-dire la largeur du polyplan $\Pi = \pi \parallel \pi'$ est définie comme suit :

$$\text{si } |\pi| = k,\ |\pi'| = n \text{ et } \Pi = \pi \parallel \pi' \text{ alors } \#\Pi = \prod_{i=1}^{n} \frac{k+i}{i}$$

Démonstration. Nous définissons naturellement le nombre de plans déduit de $\pi \parallel \pi'$ par sa définition récursive (*cf.* paragraphe 6.3.1), avec $f(k,n)$ le nombre de plans :

$$\begin{aligned}
&f(k,n) = f(n,k) && \text{(car l'opérateur } \square \text{ est symétrique)}\\
&f(0,n) = 1 && \text{(un seul plan si rien ne doit être entrelacé)}\\
&f(k,n) = f(k-1,n) + f(k,n-1) && \text{(selon la définition de } \Pi = \pi_1 \parallel \pi_2)
\end{aligned}$$

Nous voulons montrer que $f(k,n) = \prod_{i=1}^{n} \frac{k+i}{i}$.

1. Montrer que $f(k,n) = \prod_{i=1}^{n} \frac{k+i}{i}$ est symétrique :
   (a) pour $n = k$, la symétrie est triviale
   (b) pour $n > k$:

$$\begin{aligned}
f(n,k) &= \frac{(n+1)(n+2)\cdots(n+k)}{1 \times \cdots \times k} && \text{(pas de simplification car } k < n)\\
f(k,n) &= \frac{(k+1)(k+2)\cdots(k+n)}{1 \times \cdots \times n} && \text{(simplification des facteurs entre } k+1 \text{ et } n)\\
&= \frac{(n+1)(n+2)\cdots(n+k)}{1 \times \cdots \times k}\\
f(n,k) &= f(k,n)
\end{aligned}$$

(c) pour $n < k$, preuve similaire.

Donc c'est symétrique.

2. Montrer que $f(k,n) = \prod_{i=1}^{n} \frac{k+i}{i}$ par récurrence, avec pour hypothèse de récurrence que $f(k-1,n) = \prod_{i=1}^{n} \frac{k-1+i}{i}$ (et par symétrie, $f(k,n-1) = \prod_{i=1}^{n-1} \frac{k+i}{i}$) :
   (a) cas de base $f(0,n) = \prod_{i=1}^{n} \frac{0+i}{i} = \underbrace{1 \times 1 \times \cdots \times 1}_{n \text{ fois}} = 1$.
   (b) cas de base $f(n,0) = \prod_{i=1}^{0} \frac{n+i}{i} = 1$.
   (c) pour $f(k,n)$

$$\begin{aligned} f(k,n) &= f(k-1,n) + f(k,n-1) \\ &= \prod_{i=1}^{n} \left(\frac{k-1+i}{i}\right) + \prod_{i=1}^{n-1} \left(\frac{k+i}{i}\right) \quad \text{(par hypothèse de récurrence)} \end{aligned}$$

Par ailleurs :

$$\begin{aligned} \prod_{i=1}^{n} \left(\frac{k+i}{i}\right) &= \frac{k+n}{n} \prod_{i=1}^{n-1} \left(\frac{k+i}{i}\right) \\ &= \frac{k}{n} \prod_{i=1}^{n-1} \left(\frac{k+i}{i}\right) + \prod_{i=1}^{n-1} \left(\frac{k+i}{i}\right) \\ &= \frac{k}{n} \prod_{i=1}^{n-1} \left(\frac{k+i}{i}\right) + f(k,n-1) \qquad \text{(par hypothèse de récurrence)} \end{aligned}$$

Il reste à montrer que $\frac{k}{n} \prod_{i=1}^{n-1} \left(\frac{k+i}{i}\right) = \prod_{i=1}^{n} \left(\frac{k-1+i}{i}\right)$:
– partie gauche :

$$\begin{aligned} \frac{k}{n} \prod_{i=1}^{n-1} \left(\frac{k+i}{i}\right) &= \frac{k}{n} \times \frac{n}{k+n} \prod_{i=1}^{n} \left(\frac{k+i}{i}\right) \\ &= \frac{1}{k+n} \left(\frac{\prod_{i=0}^{n}(k+i)}{\prod_{i=1}^{n} i}\right) \end{aligned}$$

– partie droite :

$$\begin{aligned} \prod_{i=1}^{n} \left(\frac{k-1+i}{i}\right) &= \prod_{i=1}^{n} \left(\frac{k+(i-1)}{i}\right) \\ &= \frac{\prod_{i=0}^{n-1}(k+i)}{\prod_{i=1}^{n} i} \\ &= \frac{1}{k+n} \left(\frac{\prod_{i=0}^{n}(k+i)}{\prod_{i=1}^{n} i}\right) \end{aligned}$$

On a montré que si l'hypothèse de récurrence était vraie, alors $f(k,n) = \prod_{i=1}^{n} \left(\frac{k+i}{i}\right)$. Or, nous avons montré que dans le cas de base $f(0,n) = 1 = \prod_{i=1}^{n} \left(\frac{0+i}{i}\right)$, donc, par récurrence, $f(k,n) = \prod_{i=1}^{n} \left(\frac{k+i}{i}\right)$.

□

Par exemple, si $p = a_1 \gg a_2$ et $p' = a'_1 \gg a'_2$, alors

$$p \parallel p' = \left\langle \begin{array}{l} (a_1 \gg a_2 \gg a'_1 \gg a'_2) \square (a_1 \gg a'_1 \gg a_2 \gg a'_2) \square (a'_1 \gg a_1 \gg a_2 \gg a'_2) \square \\ (a_1 \gg a'_1 \gg a'_2 \gg a_2) \square (a'_1 \gg a_1 \gg a'_2 \gg a_2) \square (a'_1 \gg a'_2 \gg a_1 \gg a_2) \end{array} \right\rangle$$

Soient Π' et Π'' deux polyplans tels que $\#\Pi' = k$ et $\#\Pi'' = n$, et le polyplan $\Pi = \Pi' \parallel \Pi''$, alors $\#\Pi = \sum_{i=1}^{k} \left(\sum_{j=1}^{n} \#\Pi_{i,j}\right)$ avec $\Pi_{i,j} = \pi_i \parallel \pi_j$, et π_i le $i^{\text{ème}}$*plan* dérivé de Π' et π_j le $j^{\text{ème}}$*plan* dérivé de Π''.

C.4.3 Propriétés

Pour donner une sémantique dénotationnelle à l'entrelacement entre deux λ-plans indépendants π_1 et π_2, nous exprimons $\Pi = \pi_1 \parallel \pi_2$ par le triplet de préconditions et d'effets suivant :

$$\pi_1 \parallel \pi_2 = \big(pre(\pi_1) \cup pre(\pi_2), \mathit{eff}^+(\pi_1) \cup \mathit{eff}^+(\pi_2), \mathit{eff}^-(\pi_1) \cup \mathit{eff}^-(\pi_2)\big)$$

Démonstration. Soient deux plans indépendants π_1 et π_2, et le polyplan $\Pi = \pi_1 \parallel \pi_2$.

Par définition, l'entrelacement permet notamment d'exécuter les plans en séquence : $\pi_1 \gg \pi_2$ ou $\pi_2 \gg \pi_1$. Or pour que ces deux séquences puisse être exécutée sur un même état, il faut que cet état satisfasse d'une part les préconditions de π_1, et d'autre part celles de π_2. Aussi, les préconditions de Π sont l'union les préconditions de π_1 et les préconditions de π_2.

Tous les couples d'actions (a_1, a_2) tels que $a_1 \in \pi_1$ et $a_2 \in \pi_2$ sont par définition indépendants. Ainsi, $\mathit{eff}^+(a_1) \cap \mathit{eff}^-(a_2) =\neq$ et $\mathit{eff}^-(a_1) \cap \mathit{eff}^+(a_2) =\neq$. Quel que soit l'ordre d'exécution des actions de π_1 et π_2, les effets positifs de l'un ne sera jamais annulés par l'autre. Donc les effets positifs de Π sont l'union des effets positifs de π_1 et des effets positifs de π_2. Le raisonnement est le même pour les effets négatifs. Pour être plus formel, il faut prouver cette propriété par induction sur la longueur des plans entrelacés.

Nous généralisons aux λ-plans cette sémantique dénotationnelle : un λ-plan est un polyplan déterministe, donc représente un ensemble de plans, et la réflexion est la même pour chaque couple de plans représenté par deux λ-plans. Pour être plus formel, il faut prouver cette propriété par induction sur la largeur des λ-plans entrelacés. □

Les preuves formelles sont semblables aux preuves des propriétés de l'opérateur de permutation, en annexe C.5.

C.5 Opérateur de permutation

C.5.1 Lois algébriques

L'ordre de la permutation ne modifie pas sa sémantique.

Loi 12 (symétrie de $\diamond$)**.** *Soient* Π_1 *et* Π_2 *deux polyplans,* $(\Pi_1 \diamond \Pi_2) = (\Pi_2 \diamond \Pi_1)$.

La permutation de trois polyplans peut être éclaté en deux permutations binaires successifs, l'ordre étant sans importance, et la réciproque est vraie.

Loi 13 (associativité de $\diamond$)**.** *Soient* Π_1*,* Π_2 *et* Π_3 *trois polyplans,* $(\Pi_1 \diamond\ \Pi_2) \diamond\ \Pi_3 = \Pi_1 \diamond\ (\Pi_2 \diamond\ \Pi_3) = (\Pi_1 \diamond\ \Pi_2 \diamond\ \Pi_3)$.

Nous déduisons de la définition de l'opérateur de permutation les lois de la table C.5.

	neutre	idempotence	associativité	commutativité
$\diamond$	ϵ	oui	oui	oui

TABLE C.5 – Lois algébriques de l'opérateur de permutation.

C.5.2 Évaluation de la largeur

Le nombre de plans déduit de la permutation entre les polyplans Π' et Π'', c'est-à-dire la largeur du polyplan $\Pi = \Pi' \diamond\ \Pi''$ est définie comme suit :

$$\text{si } \#\Pi' = k, \#\Pi'' = n \text{ et } \Pi = \Pi' \diamond\ \Pi'' \text{ alors } \#\Pi = 2 \times n \times k$$

Démonstration. Soient les polyplans Π' et Π''. Par définition de l'opérateur de permutation, à chaque couple (i, j) tel que $i \in [1..k]$ et $j \in [1..n]$ correspond deux plans $\pi_{i,j} \in \Pi$ et $\pi_{j,i} \in \Pi$ tels que $\pi_{i,j} = \Pi'_i \gg\ \Pi''_j$ et $\pi_{j,i} = \Pi''_j \gg\ \Pi'_i$ avec Π'_i le $i^{\text{ème}}$ plan dérivé de Π' et Π''_j le $j^{\text{ème}}$ plan dérivé de Π''. □

C.5.3 Propriétés

Contraintes

La permutation entre deux π_1 et π_2 est définie si et seulement si les effets négatifs de l'un n'annule pas les préconditions de l'autre :

$$\mathit{pre}(\pi_1) \cap \mathit{eff}^-(\pi_2) = \emptyset \text{ et } \mathit{pre}(\pi_2) \cap \mathit{eff}^-(\pi_1) = \emptyset$$

Démonstration. Soit $E = \mathit{pre}(\pi_2) \cap \mathit{eff}^-(\pi_1)$. Pour prouver que $\mathit{pre}(\pi_2) \cap \mathit{eff}^-(\pi_1) = \emptyset$, nous nous intéressons à $\gamma(e, \pi_1 \gg\ \pi_2) = \gamma(\gamma(e, \pi_1), \pi_2)$. Par définition, $\gamma(e, \pi_1) = (e \setminus \mathit{eff}^-(\pi_1)) \cup \mathit{eff}^+(\pi_2)$, donc $\gamma(e, \pi_1) \subseteq (e \setminus E) \cup \mathit{eff}^+(\pi_2)$. Par définition, $\mathit{eff}^-(\pi_1) \cap \mathit{eff}^+(\pi_1) = \emptyset$, donc $E \cap \mathit{eff}^+(\pi_1) = \emptyset$ et $\gamma(e, \pi_1) \subseteq (e \cup \mathit{eff}^+(\pi_2)) \setminus E$. Si $\mathit{pre}(\pi_1 \gg\ \pi_2) \subseteq e$, alors $\mathit{pre}(\pi_2) \subseteq \gamma(e, \pi_1)$. Nous obtenons par transitivité $\mathit{pre}(\pi_2) \subseteq (e \cup \mathit{eff}^+(\pi_2)) \setminus E$. Aucun élément de E ne doit donc appartenir à $\mathit{pre}(\pi_2)$, or si un élément appartient à E il appartient par définition de l'intersection à $\mathit{pre}(\pi_2)$, donc $E = \mathit{pre}(\pi_2) \cap \mathit{eff}^-(\pi_1) = \emptyset$.
La preuve pour $\mathit{pre}(\pi_1) \cap \mathit{eff}^-(\pi_2) = \emptyset$ est similaire. □

Cette règle se généralise pour la permutation entre deux polyplans, où chaque couple de plans dérivés des polyplans doit satisfaire cette propriété pour que la permutation soit définie.

La permutation entre deux plans π_1 et π_2 est définie si et seulement si :

$$\mathit{eff}^+(\pi_1) \cap \mathit{eff}^-(\pi_2) = \emptyset \text{ et } \mathit{eff}^-(\pi_1) \cap \mathit{eff}^+(\pi_2) = \emptyset$$

Démonstration. Comme $\gamma(e, \pi_1 \gg\ \pi_2) = \gamma(e, \pi_2 \gg\ \pi_1)$ par définition de la permutation, nous avons :

1. $\gamma(e, \pi_1 \gg\ \pi_2) \setminus \gamma(e, \pi_2 \gg\ \pi_1) = \emptyset$ et, de façon symétrique
2. $\gamma(e, \pi_2 \gg\ \pi_1) \setminus \gamma(e, \pi_1 \gg\ \pi_2) = \emptyset$

Par définition de $\gamma(e, \pi_1 \gg \pi_2)$ et de $\gamma(e, \pi_2 \gg \pi_1)$, nous obtenons par l'égalité 1 :

$$\emptyset = \Bigg(\Big(\big((e \setminus \mathit{eff}^-(\pi_1)) \cup \mathit{eff}^+(\pi_1) \big) \setminus \mathit{eff}^-(\pi_2) \Big) \cup \mathit{eff}^+(\pi_2) \Bigg) \setminus \gamma(e, \pi_2 \gg \pi_1)$$

$$\emptyset = \Bigg(\Big(\big((e \setminus \mathit{eff}^-(\pi_1)) \cup \mathit{eff}^+(\pi_1) \big) \setminus \mathit{eff}^-(\pi_2) \Big) \setminus \gamma(e, \pi_2 \gg \pi_1) \Bigg) \bigcup \big(\mathit{eff}^+(\pi_2) \setminus \gamma(e, \pi_2 \gg \pi_1) \big)$$

Nous la décomposons en deux égalités :

$$\emptyset = \Bigg(\Big(\big((e \setminus \mathit{eff}^-(\pi_1)) \cup \mathit{eff}^+(\pi_1) \big) \setminus \mathit{eff}^-(\pi_2) \Big) \setminus \gamma(e, \pi_2 \gg \pi_1)$$

Et :

$$\emptyset = \mathit{eff}^+(\pi_2) \setminus \gamma(e, \pi_2 \gg \pi_1)$$

Après développement, la première est vérifiée par définition. La seconde donne :

$$\emptyset = \mathit{eff}^+(\pi_2) \setminus \Bigg(\Big(\big((e \setminus \mathit{eff}^-(\pi_2)) \cup \mathit{eff}^+(\pi_2) \big) \setminus \mathit{eff}^-(\pi_1) \Big) \cup \mathit{eff}^+(\pi_1) \Bigg)$$

Par simplification :

$$\emptyset = \big(\mathit{eff}^+(\pi_2) \cap \mathit{eff}^-(\pi_1) \big) \setminus \mathit{eff}^+(\pi_1)$$

Comme $\mathit{eff}^-(\pi_1) \cap \mathit{eff}^+(\pi_1) = \emptyset$:

$$\emptyset = \mathit{eff}^+(\pi_2) \cap \mathit{eff}^-(\pi_1)$$

La preuve est symétrique pour $\emptyset = \mathit{eff}^+(\pi_1) \cap \mathit{eff}^-(\pi_2)$.

□

Cette règle se généralise pour la permutation entre deux polyplans, où chaque couple de plans dérivés des polyplans doit satisfaire cette propriété pour que la permutation soit définie.

Sémantique dénotationnelle

Le λ-plan $\Pi = \Pi_1 \diamond \Pi_2$ est défini par le triplet de préconditions et d'effets suivant :

$$\Pi = \Pi_1 \diamond \Pi_2 = \big(\mathit{pre}(\Pi_1) \cup \mathit{pre}(\Pi_2), \mathit{eff}^+(\Pi_1) \cup \mathit{eff}^+(\Pi_2), \mathit{eff}^-(\Pi_1) \cup \mathit{eff}^-(\Pi_2) \big)$$

Démonstration. Pour que $\Pi_1 \diamond \Pi_2$ soit applicable sur l'état e, il faut que $(\Pi_1 \gg \Pi_2)$ et $(\Pi_2 \gg \Pi_1)$ soient applicables sur e. Il suffit donc de faire l'union des deux ensembles de préconditions :

$$\begin{aligned} \mathit{pre}(\Pi_1 \diamond \Pi_2) &= \big(\mathit{pre}(\Pi_1) \cup (\mathit{pre}(\Pi_2) \setminus \mathit{eff}^+(\Pi_1)) \big) \cup \big(\mathit{pre}(\Pi_2) \cup (\mathit{pre}(\Pi_1) \setminus \mathit{eff}^+(\Pi_2)) \big) \\ &= \mathit{pre}(\Pi_1) \cup \mathit{pre}(\Pi_2) \end{aligned}$$

$\mathit{eff}^+(\Pi_1 \diamond \Pi_2)$ vaut soit $\mathit{eff}^+(\Pi_1 \gg \Pi_2)$ soit $\mathit{eff}^+(\Pi_2 \gg \Pi_1)$:

$$\begin{aligned}
\mathit{eff}^+(\Pi_1 \gg \Pi_2) &= \big(\mathit{eff}^+(\Pi_1) \setminus \mathit{eff}^-(\Pi_2)\big) \cup \mathit{eff}^+(\Pi_2) \\
&= \mathit{eff}^+(\Pi_1) \cup \mathit{eff}^+(\Pi_2) && (\text{car } \mathit{eff}^+(\Pi_1) \cap \mathit{eff}^-(\Pi_2) = \emptyset) \\
\mathit{eff}^+(\Pi_2 \gg \Pi_1) &= \big(\mathit{eff}^+(\Pi_2) \setminus \mathit{eff}^-(\Pi_1)\big) \cup \mathit{eff}^+(\Pi_1) \\
&= \mathit{eff}^+(\Pi_2) \cup \mathit{eff}^+(\Pi_1) && (\text{car } \mathit{eff}^+(\Pi_2) \cap \mathit{eff}^-(\Pi_1) = \emptyset) \\
\mathit{eff}^+(\Pi_1 \diamond \Pi_2) &= \mathit{eff}^+(\Pi_2) \cup \mathit{eff}^+(\Pi_1) && (\text{car } \mathit{eff}^+(\Pi_1 \gg \Pi_2) = \mathit{eff}^+(\Pi_2 \gg \Pi_1))
\end{aligned}$$

$\mathit{eff}^-(\Pi_1 \diamond \Pi_2)$ vaut soit $\mathit{eff}^-(\Pi_1 \gg \Pi_2)$ soit $\mathit{eff}^-(\Pi_2 \gg \Pi_1)$:

$$\begin{aligned}
\mathit{eff}^-(\Pi_1 \gg \Pi_2) &= \big(\mathit{eff}^-(\Pi_1) \setminus \mathit{eff}^+(\Pi_2)\big) \cup \mathit{eff}^-(\Pi_2) \\
&= \mathit{eff}^-(\Pi_1) \cup \mathit{eff}^-(\Pi_2) && (\text{car } \mathit{eff}^-(\Pi_1) \cap \mathit{eff}^+(\Pi_2) = \emptyset) \\
\mathit{eff}^-(\Pi_2 \gg \Pi_1) &= \big(\mathit{eff}^-(\Pi_2) \setminus \mathit{eff}^+(\Pi_1)\big) \cup \mathit{eff}^-(\Pi_1) \\
&= \mathit{eff}^-(\Pi_2) \cup \mathit{eff}^-(\Pi_1) && (\text{car } \mathit{eff}^-(\Pi_2) \cap \mathit{eff}^+(\Pi_1) = \emptyset) \\
\mathit{eff}^-(\Pi_1 \diamond \Pi_2) &= \mathit{eff}^-(\Pi_2) \cup \mathit{eff}^-(\Pi_1) && (\text{car } \mathit{eff}^-(\Pi_1 \gg \Pi_2) = \mathit{eff}^-(\Pi_2 \gg \Pi_1))
\end{aligned}$$

□

Structure de choix itératif

Contraintes nécessairement vérifiées par le traitement

Par définition, le λ-plan *reiterate* ne doit modifier aucune propriété liée aux objets ($\mathtt{o}$-i) manipulés par le λ-plan $\mathit{treat}\,(\mathtt{o}\text{-}i)$ dans la structure de choix itératif, puisque *reiterate* ne s'applique pas sur les objets. Ainsi, aucun effet de *reiterate* n'est dans les effets instanciés par l'application de *treat* sur un objet ($\mathtt{o}$-i) :

$$\begin{aligned}
\forall i, \big(\mathit{eff}^+(\mathit{reiterate}) \cup \mathit{eff}^-(\mathit{reiterate})\big) \bigcap \\
\big(\big(\mathit{eff}^+(\mathit{treat}\,(\mathtt{o}\text{-}i)) \setminus \mathtt{c}\text{-}\mathit{eff}^+(\mathit{treat})\big) \cup \big(\mathit{eff}^-(\mathit{treat}\,(\mathtt{o}\text{-}i)) \setminus \mathtt{c}\text{-}\mathit{eff}^+(\mathit{treat})\big)\big) \quad = \emptyset
\end{aligned}$$

Le λ-plan *reiterate* doit satisfaire quelques propriétés de façon à ce que les λ-plans $\pi_i = (\mathit{treat}\,(\mathtt{o}\text{-}i) \gg \mathit{reiterate})$ puissent être permutées. Notamment, pour chaque couple de λ-plans π_i et π_j, une des propriétés nécessaires à la permutation est $\mathit{pre}\,(\pi_i) \cap \mathit{eff}^-(\pi_j) = \emptyset$, ce qui implique :

$$\begin{aligned}
\mathit{eff}^-(\mathit{reiterate}) \cap \mathit{pre}\,(\mathit{treat}\,(\mathtt{o}\text{-}i)) &= \emptyset \\
\mathit{eff}^-(\mathit{reiterate}) \cap (\mathit{pre}\,(\mathit{reiterate}) \setminus \mathit{eff}^+(\mathit{treat}\,(\mathtt{o}\text{-}i))) &= \emptyset
\end{aligned}$$

Démonstration. Nous utilisons la définition de $\mathit{pre}\,(\pi_i)$ et $\mathit{eff}^-(\pi_j)$:

$$\begin{aligned}
\emptyset =& \mathit{pre}\,(\pi_i) \cap \mathit{eff}^-(\pi_j) \\
\emptyset =& \Big(\mathit{pre}\,(\mathit{treat}\,(\mathtt{o}\text{-}i)) \cup \big(\mathit{pre}\,(\mathit{reiterate}) \setminus \mathit{eff}^+(\mathit{treat}\,(\mathtt{o}\text{-}i))\big)\Big) \bigcap \\
& \Big(\big(\mathit{eff}^-(\mathit{treat}\,(\mathtt{o}\text{-}i)) \setminus \mathit{eff}^+(\mathit{reiterate})\big) \cup \mathit{eff}^-(\mathit{reiterate})\Big)
\end{aligned}$$

Par distribution de $\cap$

$$\begin{aligned}\emptyset =&\Big(\Big(pre\,(treat\,(\texttt{o-}i)) \cup \big(pre\,(reiterate) \setminus \mathit{eff}^{+}(treat\,(\texttt{o-}i))\big)\Big) \cap \\ &\qquad\qquad \big(\mathit{eff}^{-}(treat\,(\texttt{o-}i)) \setminus \mathit{eff}^{+}(reiterate)\big)\Big) \bigcup \\ &\Big(\Big(pre\,(treat\,(\texttt{o-}i)) \cup \big(pre\,(reiterate) \setminus \mathit{eff}^{+}(treat\,(\texttt{o-}i))\big)\Big) \cap \mathit{eff}^{-}(reiterate)\Big)\end{aligned}$$

Comme $\mathit{eff}^{-}(treat\,(\texttt{o-}j)) \subseteq \mathit{eff}^{-}(\pi_j)$, $pre\,(treat\,(\texttt{o-}i)) \subseteq pre\,(\pi_i)$ et que $\mathit{eff}^{-}(\pi_j) \cap pre\,(\pi_i) = \emptyset$, $\mathit{eff}^{-}(treat\,(\texttt{o-}j)) \cap pre\,(treat\,(\texttt{o-}i)) = \emptyset$ et donc la première intersection est vide. Par distribution de l'intersection dans la seconde :

$$\begin{aligned}\emptyset =&\big(pre\,(treat\,(\texttt{o-}i)) \cap \mathit{eff}^{-}(reiterate)\big) \cup \\ &\Big(\big(pre\,(reiterate) \setminus \mathit{eff}^{+}(treat\,(\texttt{o-}i))\big) \cap \mathit{eff}^{-}(reiterate)\Big)\end{aligned}$$

Nous la divisons en deux égalités :

$$\emptyset = pre\,(treat\,(\texttt{o-}i)) \cap \mathit{eff}^{-}(reiterate)$$

Et :

$$\emptyset = \big(pre\,(reiterate) \setminus \mathit{eff}^{+}(treat\,(\texttt{o-}i))\big) \cap \mathit{eff}^{-}(reiterate)$$

□

De la même façon, $pre\,(\pi_j) \cap \mathit{eff}^{-}(\pi_i) = \emptyset$ implique :
- $\mathit{eff}^{-}(reiterate) \cap pre\,(treat\,(\texttt{o-}j)) = \emptyset$; et
- $\mathit{eff}^{-}(reiterate) \cap (pre\,(reiterate) \setminus \mathit{eff}^{+}(treat\,(\texttt{o-}j))) = \emptyset$.

Démonstration. Similaire à la preuve précédente. □

Ainsi, pour tout λ-plan $\pi = treat\,(\texttt{o-}i) \gg reiterate$, nous avons *reiterate* qui ne doit pas annuler les préconditions de $treat\,(\texttt{o-}i)$, ni ses propres préconditions qui ne sont pas soutenues par $treat\,(\texttt{o-}i)$. Par définition de *reiterate*, les préconditions en question ne peuvent pas concerner l'objet manipulé et donc concerne les invariants du λ-plan abstrait $treat(obj)$:

$$\mathit{eff}^{-}(reiterate) \cap \texttt{c-}pre\,(treat(obj)) = \emptyset$$

$$\mathit{eff}^{-}(reiterate) \cap (pre\,(reiterate) \setminus \texttt{c-}\mathit{eff}^{+}(treat(obj))) = \emptyset$$

Pour garantir que les préconditions invariantes de $treat(obj)$ ne soient pas annulées par les effets négatifs de π, les effets positifs de *reiterate* doivent les soutenir :

$$\texttt{c-}pre\,(treat(obj)) \cap \texttt{c-}\mathit{eff}^{-}(treat(obj)) \subseteq \mathit{eff}^{+}(reiterate)$$

Conformité des contraintes supplémentaires sur *reiterate*

Les contraintes suivantes permettent d'interdire à *reiterate* d'être responsable d'effets de bords :

$$\mathit{eff}^{+}(reiterate) \subseteq \texttt{c-}\mathit{eff}^{-}(treat)$$
$$\mathit{eff}^{-}(reiterate) \subseteq \texttt{c-}\mathit{eff}^{+}(treat)$$

Ces deux contraintes supplémentaires n'enfreignent aucune des règles précédentes : elles restreignent l'ensemble des propositions qui peuvent former les effets positifs et négatifs de *reiterate*.

Sémantique dénotationnelle

Avec les contraintes imposées sur le λ-plan *reiterate* et la définition de la permutation, le λ-plan Π est défini par :

$$nom(\Pi) = \texttt{choose}\,(object)\,\texttt{in}\,\{(\texttt{o-1}), \cdots, (\texttt{o-n})\}\,\{treat\,(object)\ \ \texttt{if-iteration}\ \ reiterate\}$$

avec $\texttt{c-}\mathit{eff}^+ = \bigcap_{i=1}^{n} \mathit{eff}^+(treat\,(\texttt{o-}i))$:

$$pre\,(\Pi) = \bigcup_{i=1}^{n} pre\,(treat\,(\texttt{o-}i)) \cup (pre\,(reiterate) \setminus \texttt{c-}\mathit{eff}^+)$$

$$\mathit{eff}^+(\Pi) = \bigcup_{i=1}^{n} \mathit{eff}^+(treat\,(\texttt{o-}i))$$

$$\mathit{eff}^-(\Pi) = \bigcup_{i=1}^{n} \mathit{eff}^-(treat\,(\texttt{o-}i))$$

Annexe D

Propriétés des compositions

D.1 Plan

Par les lois de l'opérateur de séquence, tout plan π peut s'écrire comme une séquence d'actions : $\pi = \gg (a_1, \cdots, a_n)$, avec $n \geq 0$. Nous considérons un plan π selon deux sémantiques :

- opérationnelle, π est alors définit par l'application d'une séquence d'actions ;
- dénotationnelle, π est alors définit un triplet $(nom(\pi), pre(\pi), \mathit{eff}(\pi))$ avec $\mathit{eff}(\pi) = \big(\mathit{eff}^+(\pi), \mathit{eff}^-(\pi)\big)$.

π est défini récursivement selon la longueur n du plan :

- si $n = 0$, le plan est vide et :
  - $nom(\pi) = \epsilon$;
  - $pre(\pi) = \emptyset$;
  - $\mathit{eff}^+(\pi) = \emptyset$;
  - $\mathit{eff}^-(\pi) = \emptyset$;
- si $n = 1$, c'est-à-dire pour un plan $\pi = a$, $a \in \mathcal{A}$:
  - $nom(\pi) = nom(a)$;
  - $pre(\pi) = pre(a)$;
  - $\mathit{eff}^+(\pi) = \mathit{eff}^+(a)$;
  - $\mathit{eff}^-(\pi) = \mathit{eff}^-(a)$;
- si $n > 1$, et avec $\pi' = \gg (a_2, \cdots, a_n)$:
  - $nom(\pi) = nom(a_1) \gg nom(\pi')$;
  - $pre(\pi) = pre(a_1) \cup (pre(\pi') \setminus \mathit{eff}^+(a_1))$;
  - $\mathit{eff}^+(\pi) = (\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \cup \mathit{eff}^+(\pi')$;
  - $\mathit{eff}^-(\pi) = (\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi')) \cup \mathit{eff}^-(\pi')$.

Les sémantiques opérationnelle et dénotationnelle du plan sont équivalentes et donne lieu aux trois théorèmes suivants, qui couvrent :

- son applicabilité (préconditions) ;
- la cohérence de ses effets (intersection des effets produits et consommés) ;
- son comportement vis à vis du système de transition d'état (effets produits et consommés).

Théorème 1. *Si toutes les actions $a_i \in \pi = \gg (a_1, \cdots, a_n)$ sont successivement applicables à partir de l'état e_0, alors $pre(\pi) \subseteq e_0$, c'est-à-dire si $pre(a_1) \subseteq e_0$ et $\forall i \in [2..n]$, $pre(a_i) \subseteq e_{i-1} = \gamma(e_{i-2}, a_{i-1})$, alors $pre(\pi) \subseteq e_0$.*

Démonstration. Par induction sur la longueur n du plan $\pi = \gg (a_1, \cdots, a_n)$:

- pour $n = 0$, $pre(\epsilon) \subseteq e$;
- pour $n = 1$, $pre((a_1)) = pre(a_1) \cup (pre(\epsilon) \setminus \mathit{eff}^+(a_1)) = pre(a_1)$;
- nous faisons l'hypothèse d'induction que, pour le plan $\pi' = \gg (a_2, \cdots, a_n)$, si toutes les actions de π' sont applicables sur e_1, alors π' est applicable sur e_1. Si a_1 est applicable sur e_0 tel que $e_1 = \gamma(e_0, a_1) = (e_0 \cup \mathit{eff}^+(a_1)) \setminus \mathit{eff}^-(a_1)$, il faut prouver que $pre(\pi) \subseteq e_0$. Comme $pre(\pi) = pre(a_1) \cup (pre(\pi') \setminus \mathit{eff}^+(a_1))$, il suffit de prouver que :

– $pre(a_1) \subseteq e_0$, ce qui est vrai par hypothèse ; et que
– $pre(\pi') \setminus \mathit{eff}^+(a_1) \subseteq e_0$, ce qui équivaut à montrer que $pre(\pi') \subseteq e_0 \cup \mathit{eff}^+(a_1)$
Or, comme $e_1 = (e_0 \cup \mathit{eff}^+(a_1)) \setminus \mathit{eff}^-(a_1)$, nous avons $e_1 \subseteq e_0 \cup \mathit{eff}^+(a_1)$ et par hypothèse d'induction $pre(\pi') \subseteq e_1$ donc :
– $pre(\pi') \subseteq e_1 \subseteq e_0 \cup \mathit{eff}^+(a_1)$
Donc $pre(\pi) \subseteq e_0$.

□

Théorème 2. $\mathit{eff}^+(\pi) \cap \mathit{eff}^-(\pi) = \emptyset$

Démonstration. Par induction sur la longueur du plan :
– pour $n = 0$, par définition $\mathit{eff}^+(\epsilon) \cap \mathit{eff}^-(\epsilon) = \emptyset \cap \emptyset = \emptyset$;
– en considérant que la propriété est vraie pour le plan $\pi' = \gg (a_2, \cdots, a_n)$, elle est vraie pour $\pi = \gg (a_1, \cdots, a_n)$ puisque, par définition de $\mathit{eff}^+(\pi)$ et de $\mathit{eff}^-(\pi)$:

$$\begin{aligned}\mathit{eff}^+(\pi) \cap \mathit{eff}^-(\pi) =& \big((\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \cup \mathit{eff}^+(\pi')\big) \bigcap \\ & \big((\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi')) \cup \mathit{eff}^-(\pi')\big)\end{aligned}$$

par distribution de $\cap$:

$$\begin{aligned}\mathit{eff}^+(\pi) \cap \mathit{eff}^-(\pi) =& \big((\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \cap (\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi'))\big) \bigcup \\ & \big(\mathit{eff}^+(\pi') \cap (\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi'))\big) \bigcup \\ & \big((\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \cap \mathit{eff}^-(\pi')\big) \bigcup \\ & (\mathit{eff}^+(\pi') \cap \mathit{eff}^-(\pi'))\end{aligned}$$

par induction $\mathit{eff}^+(\pi') \cap \mathit{eff}^-(\pi') = \emptyset$ donc :

$$\begin{aligned}\mathit{eff}^+(\pi) \cap \mathit{eff}^-(\pi) =& \big((\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \cap (\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi'))\big) \bigcup \\ & \big(\mathit{eff}^+(\pi') \cap (\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi'))\big) \bigcup \\ & \big((\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \cap \mathit{eff}^-(\pi')\big)\end{aligned}$$

comme $A \cap (B \setminus A) = \emptyset$ pour tout ensemble A et B :

$$\mathit{eff}^+(\pi) \cap \mathit{eff}^-(\pi) = \big((\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \cap (\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi'))\big)$$

par définition $\mathit{eff}^+(a_1) \cap \mathit{eff}^-(a_1) = \emptyset$ donc :

$$\mathit{eff}^+(\pi) \cap \mathit{eff}^-(\pi) = \emptyset$$

□

Théorème 3. $\gamma(e, \pi) = (e \cup \mathit{eff}^+(\pi)) \setminus \mathit{eff}^-(\pi)$

Démonstration. Par induction sur la longueur n du plan $\pi = \gg (a_1, \cdots, a_n)$:
– pour $n = 0$, $\gamma(e, \epsilon) = e = (e \cup \mathit{eff}^+(\epsilon)) \setminus \mathit{eff}^-(\epsilon)$;
– en faisant l'hypothèse d'induction que, pour le plan $\pi' = \gg (a_2, \cdots, a_n)$, l'égalité $\gamma(e_1, \pi') = (e_1 \cup \mathit{eff}^+(\pi')) \setminus \mathit{eff}^-(\pi')$ est vraie, nous avons $\gamma(e_0, \pi) = \gamma(e_1, \pi')$ avec

$e_1 = (e_0 \cup \mathit{eff}^+(a_1)) \setminus \mathit{eff}^-(a_1)$, et par définition de l'application d'une action sur un état :

$$\gamma(e_0, \pi) = \big(e_0 \cup \mathit{eff}^+(\pi)\big) \setminus \mathit{eff}^-(\pi)$$

par définition de $\mathit{eff}^+(\pi)$ et $\mathit{eff}^-(\pi)$:

$$\gamma(e_0, \pi) = \Big(e_0 \cup \big((\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \cup \mathit{eff}^+(\pi')\big)\Big) \setminus \big((\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi')) \cup \mathit{eff}^-(\pi')\big)$$

par transitivité de $\cup$ et distribution de $\setminus$:

$$\begin{aligned}\gamma(e_0, \pi) = &\Big(e_0 \setminus \big((\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi')) \cup \mathit{eff}^-(\pi')\big)\Big) \bigcup \\ &\Big((\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \setminus \big((\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi')) \cup \mathit{eff}^-(\pi')\big)\Big) \bigcup \\ &\Big(\mathit{eff}^+(\pi') \setminus \big((\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi')) \cup \mathit{eff}^-(\pi')\big)\Big)\end{aligned}$$

par définition, $\mathit{eff}^+(\pi') \cap \mathit{eff}^-(\pi') = \emptyset$; et $\mathit{eff}^+(\pi') \cap (\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi')) = \emptyset$:

$$\begin{aligned}\gamma(e_0, \pi) = &\Big(e_0 \setminus \big((\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi')) \cup \mathit{eff}^-(\pi')\big)\Big) \bigcup \\ &\Big((\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \setminus \big((\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi')) \cup \mathit{eff}^-(\pi')\big)\Big) \bigcup \\ &\mathit{eff}^+(\pi')\end{aligned}$$

par définition, $\mathit{eff}^+(a_1) \cap \mathit{eff}^-(a_1) = \emptyset$:

$$\begin{aligned}\gamma(e_0, \pi) = &\Big(e_0 \setminus \big((\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi')) \cup \mathit{eff}^-(\pi')\big)\Big) \bigcup \\ &\big((\mathit{eff}^+(a_1) \setminus \mathit{eff}^-(\pi')) \setminus \mathit{eff}^-(\pi')\big) \bigcup \\ &\mathit{eff}^+(\pi')\end{aligned}$$

comme $\mathit{eff}^+(\pi') \cap \mathit{eff}^-(\pi') = \emptyset$, factorisation de $\setminus \mathit{eff}^-(\pi')$:

$$\gamma(e_0, \pi) = \Big(\big(e_0 \setminus (\mathit{eff}^-(a_1) \setminus \mathit{eff}^+(\pi'))\big) \cup \mathit{eff}^+(a_1) \cup \mathit{eff}^+(\pi')\Big) \setminus \mathit{eff}^-(\pi')$$

comme $\big(A \setminus (B \setminus C)\big) \cup C = (A \setminus B) \cup C$:

$$\gamma(e_0, \pi) = \big((e_0 \setminus \mathit{eff}^-(a_1)) \cup \mathit{eff}^+(a_1) \cup \mathit{eff}^+(\pi')\big) \setminus \mathit{eff}^-(\pi')$$

et définition de e_1 et par hypothèse d'induction :

$$\gamma(e_0, \pi) = (e_1 \cup \mathit{eff}^+(\pi')) \setminus \mathit{eff}^-(\pi') = \gamma(e_1, \pi')$$

□

D.2 Polyplan

Par les lois des opérateurs de séquence et d'alternative, tout polyplan Π peut s'écrire comme une alternative entre plans : $\Pi = \square\ (\pi_1, \cdots, \pi_n)$, avec $n \geq 0$. Nous considérons un polyplan Π selon deux sémantiques :

- opérationnelle, Π est alors définit par l'application d'un des plans qu'il défini ;
- dénotationnelle, Π est alors définit un tuple $(nom(\Pi), pre(\Pi), \mathit{eff}_1(\Pi), \cdots, \mathit{eff}_n(\Pi))$ avec $\forall i \in [1..n], \mathit{eff}_i(\Pi) = \big(\mathit{eff}_i^+(\Pi), \mathit{eff}_i^-(\Pi)\big)$.

Π est défini selon la largeur n du plan :

- si $n = 0$, le polyplan est vide et :
  - $nom(\Pi) = \epsilon$;
  - $pre(\Pi) = \emptyset$;
  - $\mathit{eff}^+(\Pi) = \emptyset$;
  - $\mathit{eff}^-(\Pi) = \emptyset$;
- si $n > 0$:
  - $nom(\Pi) = \square\ (\pi_1, \cdots, \pi_n)$;
  - $pre(\Pi) = \bigcup_{i=1}^{n} pre(\pi_i)$;
  - $\forall i \in [1..n], \mathit{eff}_i^+(\Pi) = \mathit{eff}^+(\pi_i)$;
  - $\forall i \in [1..n], \mathit{eff}_i^-(\Pi) = \mathit{eff}^-(\pi_i)$;

Les sémantiques opérationnelle et dénotationnelle du polyplan sont équivalentes et donne lieu aux trois théorèmes suivants, qui couvrent :

- son applicabilité (préconditions) ;
- la cohérence de ses effets (intersection des effets produits et consommés) ;
- son comportement vis à vis du système de transition d'état (effets produits et consommés).

Théorème 4. *Si tous les plans $\pi_i \in \Pi = \square\ (\pi_1, \cdots, \pi_n)$ sont applicables sur l'état e_0, alors $pre(\Pi) \subseteq e_0$.*

Démonstration. Par définition, $pre(\Pi) = \bigcup_{i=1}^{n} pre(\pi_i)$. □

Théorème 5. $\mathit{eff}_i^+(\Pi) \cap \mathit{eff}_i^-(\Pi) = \emptyset$

Démonstration. Par définition des plans qui composent Π : l'opérateur de séquence conserve cette propriété dans les nouveaux plans calculés, et l'opérateur d'alternative ne modifie pas les effets positifs et négatifs des plans. □

Théorème 6. $\gamma(e, \Pi) = (e \cup \mathit{eff}^+(\Pi)) \setminus \mathit{eff}^-(\Pi)$

Démonstration. Par définition de l'application d'un polyplan (un choix parmi n). □

Sixième partie

Listes

Liste des définitions

Liste des figures

Liste des tableaux

Liste des Algorithmes

Liste des plans

Table des matières

Bibliographie

[AAA+07] Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. Web Services Business Process Execution Language (WS-BPEL). OASIS standard, OASIS, April 2007. (Cité en page 14.)

[ABH+02] Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, and Katia Sycara. Daml-s : Web service description for the semantic web. In *roceedings of The First International Semantic Web Conference (ISWC)*, 2002. (Cité en page 16.)

[ACD+03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services. Technical report, BEA and IBM and Microsoft and SAP AG and Siebel Systems, May 2003. (Cité en page 21.)

[AD05] Gerevini Alfonso and Long Derek. BNF Description of PDDL3.0, oct 2005. (Cité en pages 9, 20, 23, 38, 52 et 99.)

[AH85] James F. Allen and Patrick J. Hayes. A common-sense theory of time. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1 of *IJCAI'85*, pages 528–531, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc. (Cité en page 5.)

[AM07] G. Agre and Z. Marinova. An INFRAWEBS approach to dynamic composition of semantic web services. In *Cybernetics and Information Technologies (CTI)*, volume 7, pages 45–61, 2007. (Cité en page 21.)

[ASW98] Corin Anderson, David E. Smith, and Daniel S. Weld. Conditional effects in graphplan, 1998. (Cité en page 173.)

[Bae05] J. C. M. Baeten. A brief history of process algebra. *Theor. Comput. Sci.*, 335(2-3) :131–146, May 2005. (Cité en page 176.)

[BB87] Tommaso Bolognesi and Ed Brinksma. Introduction to the iso specification language lotos. *Comput. Netw. ISDN Syst.*, 14(1) :25–59, March 1987. (Cité en page 176.)

[BBB+02] Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma, and Scott Williams. Web Services Conversation Language (WSCL). W3C note, W3C, March 2002. (Cité en page 14.)

[BEMS05] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-ff : Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24 :581–621, 2005. (Cité en page 56.)

[BF97] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1) :281–300, 1997. (Cité en pages 25, 30, 40, 91, 134 et 168.)

[Bly99] Jim Blythe. Decision-theoretic planning. *AI Magazine*, 20(4) :45–53, 1999. (Cité en page 43.)

[Bon09] Blai Bonet. Conformant plans and beyond : Principles and complexity. *Artificial Intelligence*, 2009. (Cité en pages 25 et 33.)

[BP96] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In William J. Clancey and Daniel S. Weld, editors, *AAAI/IAAI, Vol. 2*, pages 1168–1175. AAAI Press / The MIT Press, 1996. (Cité en page 42.)

[BPG09] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *ICAPS*, 2009. (Cité en page 8.)

[BS93] Christian J. M. Bastien and Dominique L. Scapin. Preliminary findings on the effectiveness of ergonomic criteria for the evaluation of human-computer interfaces. In Stacey Ashlund, Kevin Mullet, Austin Henderson, Erik Hollnagel, and Ted N. White, editors, *INTERCHI Adjunct Proceedings*, pages 187–188. ACM, 1993. (Cité en pages 4, 5 et 176.)

[Bus01] Business Process Project Team. ebXML Business Process Specification Schema. OASIS standard, OASIS, May 2001. (Cité en page 14.)

[BvHH+04] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL-S : Semantic markup for web services. W3c recommendation, W3C, feb 2004. (Cité en pages 14, 17 et 19.)

[CAS04] CASCOM : Context-aware business application service co-ordination in mobile computing environments. Projet européen, sep 2004. Voir `www.ist-cascom.org/`. (Cité en pages 19, 20 et 23.)

[CCT+03] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3) :289–308, 2003. (Cité en page 3.)

[CFS07] Andrew Coles, Maria Fox, and Amanda Smith. Online identification of useful macro-actions for planning. In Mark S. Boddy, Maria Fox, and Sylvie Thiébaux, editors, *ICAPS*, pages 97–104. AAAI, 2007. (Cité en page 56.)

[CNM83] Stuart K. Card, Allen Newell, and Thomas P. Moran. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983. (Cité en page 176.)

[CP92] Gregg Collins and Louise Pryor. Achieving the functionality of filter conditions in a partial order planner. In *Proceedings of the tenth national conference on Artificial intelligence*, AAAI'92, pages 375–380. AAAI Press, 1992. (Cité en page 39.)

[CPRT03] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2) :35–84, 2003. (Cité en page 33.)

[CRB04] A. Cimatti, M. Roveri, and P. Bertoli. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*, 159(1-2) :127–206, November 2004. (Cité en page 33.)

[DGLL00] Giuseppe De Giacomo, Yves Lespérance, and Hector Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2) :109–169, 2000. (Cité en page 16.)

[EF91] Max J. Egenhofer and Robert D. Franzosa. Point-set topological spacial relations. *International Journal of Geographical Information Systems*, 5(2) :161–174, 1991. (Cité en page 5.)

[EHW+92] Oren Etzioni, Steve Hanks, Daniel S. Weld, Denise Draper, Neal Lesh, and Mike Williamson. An approach to planning with incomplete information. In Bernhard Nebel, Charles Rich, and William R. Swartout, editors, *KR*, pages 115–125. Morgan Kaufmann, 1992. (Cité en page 38.)

[Eme90] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume Volume B : Formal Models and Sematics, pages 995–1072. Elsevier and MIT Press, 1990. (Cité en page 36.)

[FL99] Maria Fox and Derek Long. The detection and exploitation of symmetry in planning problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, IJCAI '99, pages 956–961, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. (Cité en page 131.)

[FN71] Richard E. Fikes and Nils J. Nilsson. Strips : A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4) :189–208, 1971. (Cité en pages 23 et 47.)

[GA99] Emmanuel Guere and Rachid Alami. A possibilistic planner that deals with non-determinism and contingency. In *Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, IJCAI'99, pages 996–1001, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. (Cité en page 38.)

[Gab11] Yoann Gabillon. *Composition d'Interfaces Homme-Machine par planification automatique*. PhD thesis, Université de Grenoble, oct 2011. (Cité en pages 4, 18, 23 et 176.)

[GB94a] Robert P. Goldman and Mark S. Boddy. Conditional linear planning. In Kristian J. Hammond, editor, *AIPS*, pages 80–85. AAAI, 1994. (Cité en page 39.)

[GB94b] Robert P. Goldman and Mark S. Boddy. Representing uncertainty in simple planners. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *KR*, pages 238–245. Morgan Kaufmann, 1994. (Cité en page 39.)

[GB96] Robert P. Goldman and Mark S. Boddy. Expressive planning and explicit knowledge. In Brian Drabble, editor, *AIPS*, pages 110–117. AAAI, 1996. (Cité en page 38.)

[GBP96] Robert P. Goldman, Mark S. Boddy, and Louise Pryor. Planning with observations and knowledge. In *In AAAI-97 Workshop on Theories of Action*, 1996. (Cité en pages 38 et 39.)

[GK97] B. Cenk Gazen and Craig A. Knoblock. Combining the expressivity of ucpop with the efficiency of graphplan. In *Proceedings of the 4th European Conference on Planning : Recent Advances in AI Planning*, ECP '97, pages 221–233, London, UK, UK, 1997. Springer-Verlag. (Cité en page 173.)

[GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning – theory and practice*. Morgan Kaufmann Publishers Inc. (Elsevier), San Francisco, CA, USA, 2004. (Cité en pages 6, 7, 10, 29, 32, 34, 36, 47, 58, 137 et 176.)

[GT00] Fausto Giunchiglia and Paolo Traverso. Planning as model checking. In *ECP '99 : 5th European Conference on Planning*, pages 1–20, London, UK, 2000. Springer-Verlag. (Cité en pages 25, 32 et 33.)

[HB03] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In Klaus-Dieter Schewe and Xiaofang Zhou, editors, *ADC*, volume 17 of *CRPIT*, pages 191–200. Australian Computer Society, 2003. (Cité en page 22.)

[HB06] Jörg Hoffmann and Ronen I. Brafman. Conformant planning via heuristic forward search : A new approach. *Artificial Intelligence*, 170(6-7) :507–541, May 2006. (Cité en page 33.)

[HBP07] Jörg Hoffmann, Piergiorgio Bertoli, and Marco Pistore. Web service composition as planning, revisited : In between background theories and initial state uncertainty. In *AAAI*, pages 1013–1018. AAAI Press, 2007. (Cité en page 16.)

[HBT11] Chung-Ching Huang, Jeffrey Bardzell, and Jennifer Terrel. Can your pet rabbit read your email ? A critical analysis of the Nabaztag Rabbit. *Designing Pleasurable Products and Interfaces (DPPI)*, jun 2011. (Cité en page 3.)

[HG11] Yuxiao Hu and Giuseppe De Giacomo. Generalized planning : Synthesizing plans that work for multiple environments. In Toby Walsh, editor, *IJCAI*, pages 918–923. IJCAI/AAAI, 2011. (Cité en page 25.)

[HKT84] David Harel, Dexter Kozen, and Jerzy Tiuryn. Dynamic logic. In *Handbook of Philosophical Logic*, pages 497–604. MIT Press, 1984. (Cité en page 38.)

[HL09] Yuxiao Hu and Hector J. Levesque. Planning with loops : Some new results. In *ICAPS Workshop on Generalized Planning*, 2009. (Cité en page 37.)

[HN01] Jörg Hoffmann and Bernhard Nebel. The ff planning system : fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, 14(1) :253–302, May 2001. (Cité en pages 25 et 134.)

[INF04] INFRAWEBS : Intelligent framework for generating open (adaptable) development platforms for web - service enabled applications using semantic web technologies, distributed decisions support units and multi-agent systems, aug 2004. Voir `http://www.infrawebs.eu/`& `http://www.infrawebs.org/`. (Cité en pages 21 et 23.)

[JDKR] Hendler James, Nau Dana, Erol Kutluhan, and Tsuneto Reiko. Hierarchical task network planning : Formalization and analysis. (Cité en pages 17, 18, 21, 23 et 43.)

[Kar01] Lars Karlsson. Conditional progressive planning under uncertainty. In Bernhard Nebel, editor, *IJCAI*, pages 431–438. Morgan Kaufmann, 2001. (Cité en pages 40 et 42.)

[KFB+05] Nickolas Kavantzas, Tony Fletcher, David Burdett, Greg Ritzinger, Yves Lafon, and Charlton Barreto. Web Services Choreography Description Language (WSCDL). W3C candidate recommendation, W3C, November 2005. (Cité en page 14.)

[KGS05] Matthias Klusch, Andreas Gerber, and Marcus Schmidt. Semantic web service composition planning with OWLS-XPlan. In *1st International AAAI Fall Symposium on Agents and the Semantic Web*, Arlington, VA, USA, 2005. AAAI. (Cité en pages 19, 20 et 23.)

[KHK] Oliver Keller, Heikki J. Helin, and Matthias Klusch. Site de CASCOM. (Cité en page 19.)

[KHW94] Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic least-commitment planning. In Barbara Hayes-Roth and Richard E.

Korf, editors, *AAAI*, pages 1073–1078. AAAI Press / The MIT Press, 1994. (Cité en pages 40 et 41.)

[LB74] D. C. Luckham and J. R. Buchanan. Automatic generation of programs containing conditional statements. In *1st Conf. of the Society for AI and the Simulation of Behaviour (AISB 74)*, pages 102–126, 1974. (Cité en pages 38 et 39.)

[Lev05] Hector J. Levesque. Planning with loops. In *Proceedings of the 19th International Joint Conference on Artificial intelligence (IJCAI)*, IJCAI'05, pages 509–515, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc. (Cité en page 37.)

[LFS+02] Derek Long, Maria Fox, David E Smith, Drew McDermott, Fahiem Bacchus, and Hector Geffner. Site de IPC, 2002. Voir `planning.cis.strath.ac.uk/competition/`. (Cité en pages 25 et 134.)

[LRL+97] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG : A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3) :59–84, 1997. (Cité en page 16.)

[Mar08] Cyrille Martin. Planification pour la composition dynamique de systèmes interactifs. Master's thesis, Université Joseph Fourier, Grenoble, France, 2008. (Cité en pages 15 et 16.)

[MBD91] Steven Minton, John Bresina, and Mark Drummond. Commitment strategies in planning : a comparative analysis. In *Proceedings of the 12th international joint conference on Artificial intelligence*, volume 1 of *IJCAI'91*, pages 259–265, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. (Cité en page 25.)

[MBE03] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing web services on the semantic web. *VLDB J.*, 12(4) :333–351, 2003. (Cité en pages 22, 23 et 213.)

[McD02] Drew V. McDermott. Estimated-regression planning for interactions with web services. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *AIPS*, pages 204–211. AAAI, 2002. (Cité en page 23.)

[MH87] J. McCarthy and P. J. Hayes. *Some philosophical problems from the standpoint of artificial intelligence*, pages 26–45. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. (Cité en page 16.)

[MJS00] I. Miguel, P. Jarvis, and Q. Shen. Flexible graphplan. In W. Horn, editor, *Proceedings of the Fourteenth European Conference on Artificial Intelligence*, pages 506–510, 2000. (Cité en page 91.)

[MR91] David A. McAllester and David Rosenblitt. Systematic nonlinear planning. In Thomas L. Dean and Kathleen McKeown, editors, *AAAI*, pages 634–639. AAAI Press / The MIT Press, 1991. (Cité en page 39.)

[MS02] Sheila McIlraith and Tran Cao Son. Adapting Golog for composition of semantic web services. In *Proceedings of 8th International Conference on Principles of Knowledge Representation and Reasoning (KR'2002)*, pages 482–496, Toulouse, Midi-pyrenees, France, avril 2002. Morgan Kaufmann Publishers, Inc. (Cité en pages 16 et 23.)

[NAC+] Dana S. Nau, Tsz-Chiu Au, Yue (Jason) Cao, Okhtay Ilghami, Ugur Kuter, Amnon Lotem, Steven Mitchell, J. William Murdock, Héctor Mu noz Avila, John Shin, Fusun Yaman, Dan Wu, and Lingling Zhang. Site de SHOP2. (Cité en page 17.)

[NAI+03] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. Shop2 : An htn planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20 :379–404, 2003. (Cité en page 17.)

[ND86] Donald A. Norman and Stephen W. Draper. *User Centered System Design : New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1986. (Cité en page 176.)

[NLFL07] M. A. Hakim Newton, John Levine, Maria Fox, and Derek Long. Learning macro-actions for arbitrary planner and domains. In *In Proceedings of the International Conference on Automated Planning and Scheduling*, 2007. (Cité en page 56.)

[NM02] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *WWW '02 : Proceedings of the 11th international conference on World Wide Web*, pages 77–88, New York, NY, USA, 2002. ACM. (Cité en pages 16, 17, 22 et 23.)

[Obj06] Object Management Group. Business Process Modeling Notation (BPMN) specification. Omg final adopted specification, BPMN.org, February 2006. (Cité en page 14.)

[PBB+04] Marco Pistore, Fabio Barbon, Piergiorgio Bertoli, Dmitry Shaparau, and Paolo Traverso. Planning and monitoring web service composition. In Christoph Bussler and Dieter Fensel, editors, *AIMSA*, volume 3192 of *Lecture Notes in Computer Science*, pages 106–115. Springer, 2004. (Cité en page 20.)

[PC96] Louise Pryor and Gregg Collins. Planning for contingencies : a decision-based approach. *Journal of Artificial Intelligence Research*, 4(1) :287–339, May 1996. (Cité en pages 39, 40 et 67.)

[Ped89] Edwin P. D. Pednault. Adl : Exploring the middle ground between strips and the situation calculus. In *KR*, pages 324–332, 1989. (Cité en page 38.)

[Pee04] Joachim Peer. A pddl based tool for automatic web service composition. In Hans Jürgen Ohlbach and Sebastian Schaffert, editors, *PPSWR*, volume 3208 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2004. (Cité en pages 18 et 213.)

[Pee05a] Joachim Peer. A pop-based replanning agent for automatic web service composition. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2005. (Cité en page 18.)

[Pee05b] Joachim Peer. Web service composition as ai planning - a survey. Technical report, University of St. Gallen, St. Gallen, Switzerland, 2005. (Cité en page 16.)

[Pet66] Carl Adam Petri. Communication with Automata. Technical Report RADC-TR-65–377, New York : Griffiss Air Force Base, Jan 1966. English translation from "Kommunikation mit Automaten". (Cité en page 16.)

[PF09] Damien Pellier and Humbert Fiorino. Un modèle de composition automatique et distribuée de services web par planification. *Revue d'Intelligence Artificielle*, 23(1) :13–46, 2009. (Cité en pages 18 et 23.)

[Plo04] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61 :17–139, 2004. (Cité en page 53.)

[PMM97] Fabio Paternò, Cristiano Mancini, and Silvia Meniconi. Concurtasktrees : A diagrammatic notation for specifying task models. In Steve Howard, Judy Hammond, and Gitte Lindgaard, editors, *INTERACT*, volume 96 of *IFIP Conference Proceedings*, pages 362–369. Chapman & Hall, 1997. (Cité en page 176.)

[PS92] Mark A. Peot and David E. Smith. Conditional nonlinear planning. In *Proceedings of the first international conference on Artificial intelligence planning systems*, pages 189–197, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. (Cité en pages 25, 39 et 67.)

[PT01] Marco Pistore and Paolo Traverso. Planning as model checking for extended goals in non-deterministic domains. In *IJCAI*, pages 479–486, 2001. (Cité en page 36.)

[PTB05] Marco Pistore, Paolo Traverso, and Piergiorgio Bertoli. Automated composition of web services by planning in asynchronous domains. In Susanne Biundo, Karen L. Myers, and Kanna Rajan, editors, *International Conference on Automated Planning & Scheduling (ICAPS'05)*, pages 2–11. AAAI, 2005. (Cité en pages 20, 23 et 213.)

[PW92] J. Scott Penberthy and Daniel S. Weld. Ucpop : A sound, complete, partial order planner for ADL. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the Third International Conference of Principles of Knowledge Representation and Reasoning (KR)*, pages 103–114. Morgan Kaufmann, San Mateo, California, 1992. (Cité en pages 26 et 39.)

[Rei10] Patrick Reignier. *Intelligence Ambiante Pro-Active : de la Spécification à l'Implémentation*. Habilitation à diriger les recherches, Université Joseph Fourier, sep 2010. (Cité en page 4.)

[RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence : A Modern Approach*. Pearson Education, 2003. (Cité en page 4.)

[RS04] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In Jorge Cardoso and Amit P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2004. (Cité en page 15.)

[RW89] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. In *Proceedings of the IEEE*, volume 77-1, pages 81–98, 1989. (Cité en page 176.)

[Sac75] Earl D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the 4th international joint conference on Artificial intelligence - Volume 1*, IJCAI'75, pages 206–214, San Francisco, CA, USA, 1975. Morgan Kaufmann Publishers Inc. (Cité en page 26.)

[SFJ00] David Smith, Jeremy Frank, and Ari Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1) :47–83, 2000. (Cité en page 43.)

[SHP03] Evren Sirin, James A. Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In Jean Bézivin, Jiankun Hu, and Zahir Tari, editors, *Web Services : Modeling, Architecture and Infrastructure (WSMAI)*, pages 17–24. ICEIS Press, 2003. (Cité en page 17.)

[SIZ08] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Learning generalized plans using abstract counting. In *AAAI'08 : Proceedings of the 23rd national conference on Artificial intelligence*, pages 991–997. AAAI Press, 2008. (Cité en page 36.)

[SIZ10] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 2010. (Cité en pages 36, 85 et 133.)

[SPAS03] Katia P. Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1(1) :27–46, 2003. (Cité en page 22.)

[SPW+04] Evren Sirin, Bijan Parsia, Dan Wu, James A. Hendler, and Dana S. Nau. Htn planning for web service composition using shop2. *Journal of Web Semantics*, 1(4) :377–396, 2004. (Cité en pages 17, 23 et 213.)

[SRb03] Sven Schwarz and Thomas R. Roth-berghofer. Towards goal elicitation by user observation. In *Proceedings of the LLWA 2003*, pages 224–228. AIFB, 2003. (Cité en page 5.)

[SS71] Dana Scott and Christopher Strachey. Toward a mathematical semantics for computer languages. In Jerome Fox, editor, *Proceedings of the Symposium on Computers and Automata*, volume XXI, pages 19–46, Brooklyn, N.Y., April 1971. Polytechnic Press. (Cité en page 53.)

[SS09] Shlomo Zilberstein Siddharth Srivastava, Neil Immerman. Challenges in finding generalized plans. In *ICAPS Workshop on Generalized Planning*, 2009. (Cité en page 43.)

[SW98] David E. Smith and Daniel S. Weld. Conformant graphplan. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 889–896, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. (Cité en pages 33, 39, 40 et 173.)

[Tat77] Austin Tate. Generating project networks. In *Proceedings of the 5th international joint conference on Artificial intelligence - Volume 2*, IJCAI'77, pages 888–893, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc. (Cité en page 26.)

[TP04] Paolo Traverso and Marco Pistore. Automated composition of semantic web services into executable processes. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Third International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 380–394. Springer, November 2004. (Cité en pages 20 et 21.)

[W3C] W3C Working Group. Web services glossary. Voir `www.w3.org/TR/ws-gloss/`. (Cité en page 13.)

[Wal07] Jean-Baptiste Waldner. *Nano-informatique et intelligence ambiante : inventer l'ordinateur du XXIe siècle*. Hermes Science Publications, 2007. (Cité en page 3.)

[War76] D. H. Warren. Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference*, pages 344–354, 1976. (Cité en pages 39 et 67.)

[WAS98] Daniel S. Weld, Corin R. Anderson, and David E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 897–904, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. (Cité en pages 39, 40 et 173.)

[Wei99] Mark Weiser. The computer for the twenty-first century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3) :3–11, July 1999. (Cité en page 3.)

[Wel94] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4) :27–61, 1994. (Cité en page 25.)

[Win] Windows. Workflow foundation. Voir `www.workflow-foundation.com`. (Cité en page 14.)

[Wor07] Workflow Management Coalition. Process Definition Interface – XML Process Definition Language (XPDL). Workflow standard, WfMC, dec 2007. Document Number WFMC-TC-1025. Voir `www.wfmc.org/`. (Cité en page 14.)

[WPS+03] Dan Wu, Bijan Parsia, Evren Sirin, James A. Hendler, and Dana S. Nau. Automating DAML-S web services composition using SHOP2. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2003. (Cité en page 17.)

[WV07] Elly Winner and Manuela Veloso. Loopdistill : Learning domain-specific planners from example plans. In *ICAPS Workshop on Planning and Scheduling*, 2007. (Cité en page 37.)

[YL04] Hakan L. S. Younes and Michael L. Littman. Ppddl1.0 : An extension to pddl for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University, Pittsburgh, PA, 2004. (Cité en page 38.)