THÈSE

présentée par

Sueli FERREIRA DA CUNHA

pour obtenir le titre de

Docteur de l'Université Joseph Fourier - Grenoble I

(Arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)

Spécialité : **Informatique**
Formation Doctorale : **Recherche Opérationnelle**

Comparaison de résolutions de problèmes exprimées par un modèle de tâches

Soutenue le 18 janvier 1999 devant le jury composé de :

M.	Gerd	FINKE	Président
Mme.	Sabine	MOISAN	Rapporteur
M.	François	CHARPILLET	Rapporteur
M.	François	RECHENMANN	Examinateur
Mme.	Danielle	ZIÉBELIN	Directeur

Thèse préparée au sein de l'INRIA/Rhône-Alpes

Remerciements

Je remercie toutes les personnes qui ont contribué pour la réalisation de ce travail. Parmi ces personnes, je tiens à remercier :

Les membres du jury qui ont accepté de juger mon travail :

Monsieur Gerd Finke, Professeur à l'Université Joseph Fourier, pour avoir accepté de présider le jury. Je le remercie également de m'avoir reçue tout au début de mon séjour en France lors de ma formation en DEA de Recherche Opérationnelle.

Mes rapporteurs : Madame Sabine Moisan, Chargée de Recherche à l'INRIA Sophia-Antipolis, et Monsieur François Charpillet, Chargé de Recherche à l'INRIA Nancy, pour avoir accepté de lire et de juger mon travail.

Monsieur François Rechenmann, Directeur de Recherche à l'INRIA Rhône-Alpes, pour m'avoir accueillie dans son équipe et avoir accepté de faire partie du jury de ma thèse.

Madame Danielle Ziébelin, Maître de Conférences à l'Université Joseph Fourier, qui a encadré cette thèse. Je la remercie également pour sa patience lors de nos discussions et de la préparation de ce mémoire, ainsi que du temps qu'elle a consacré à mes articles.

Je remercie également ...

La CAPES, organisme du gouvernement brésilien qui a financié ce travail de thèse.

Les membres du projet SHERPA : Françoise (merci de ton aide sur toute la partie administrative) ; Gilles (merci beaucoup pour ton attention lors de la lecture de ma thèse et de mes articles) ; Isabelle (chère amie, merci pour ces trois années que nous avons partagées, pas simplement un bureau) ; Christophe Bruley (oui, ton dictionnaire de synonymes m'a été très très utile ; un grand merci) ; Jean-Marc, Thibault, Petko, Vincent (les garçons d'à côté) ; Loïc, Sylvie, Alain, Camille, Corinne, les Jérôme, Michel et Philippe ; les "petits" derniers Lars et Martine (merci d'avoir partagé le bureau, avec bonne humeur, ces derniers temps). Sans oublier les anciens : Jutta, Cécile, Florence, Nina, Olivier, les Christophe et Karine.

Aux amis de l'INRIA, pas "sherpiens" : Thierry (le "siraquien") ; Catherine, Véronique (les "statisticiennes") ; Manoel ("Mexico, Mexiiiico"), Joseph ("Oi, Zé"). Sans oublier mes chers amis "au bout du couloir" : Isabel et Álvaro, pour m'avoir soutenue surtout lors de la "dernière ligne droite" (allez Isabel, à toi de jouer maintenant).

Mes amis du laboratoire ARTEMIS, avec qui j'ai partagé ma première année "d'étudiante française", lors de mon DEA : Loan, Bernard, Agnès, Wafa, Laurence, Vincent, Bouras, Ahmed, Djahed, Karina et Carlos. Ai-je oublié quelqu'un ? Si oui, toutes mes excuses.

Je tiens à remercier également ceux qui m'ont soutenue dans la vie de tous les jours et qui m'ont permis d'avoir une bonne condition émotionnelle pour avoir mené à bien ce travail jusqu'au bout. Un grand merci à :

La "petite famille" que je me suis créée : Ma "maman" Marcelle (vous avez rendu surtout ma première année en France très facile à vivre) ; mon "papa" Pierre (en plus de mon "papa", tu es mon "ange gardien" ; merci de m'avoir reçue très chaleureusement tout au début de ma vie française) ; ma "sœur indonésienne" Pina ("burum burum teribam") et finalement mon "fils allemand" Wolfgang. Ces deux derniers sont déjà retournés chez eux, mais ils restent encore dans mon cœur. Et les plus nouveaux membres de la "famille" Elisabeth et Bruno ("Óia, shumiu mashaia").

La "communauté brésilienne" de Grenoble. Je vous remercie pour tous les moments vécus ensemble, lesquels ont fait que la "saudade da terrinha" a été réduite.

Mes très chers amis Bernadette et Carlos avec qui j'ai partagé non seulement les bons moments de fête et de joie mais aussi les moments les plus difficiles. Merci également à Patrícia, Karina et Afonso. Savoir écouter est une qualité rare. Un très très grand merci à vous. "Ou melhor, muitíssimo obrigada por tudo, amigos".

Nicole, Florent, Annick, Françoise (les deux), Brigitte et Nöelle. Je suis très heureuse que nos chemins se soient croisés. Sachez que je ne pourrai jamais vous oublier.

Je remercie particulièrement mes amis "d'outre Atlantique". Un océan n'est pas suffisant pour que s'éloignent les vrais amis. Alors, un grand merci à :

Fatinha, merci pour t'être occupée de tous mes soucis administratifs et financiers. Sans toi, partir de Rio aurait été beaucoup plus difficile. Mais avant tout, un grand merci pour ton amitié.

Nina et Cidinha, vos lettres et vos coups de fil ont été très importants. Merci également de m'avoir mise au courant de toutes les informations concernant notre très cher CEB. À chaque lettre lue je rencontrais tout notre aimable environnement "ceebense".

Alexandre, très cher ami, un grand merci pour tout. C'est très important de savoir qu'il est possible de compter sur quelqu'un, même lorsqu'un océan nous sépare. Tes mails, coups de fil, lettres et visite ont rendu Rio beaucoup plus "proche" de Grenoble qu'elle ne l'est.

Et enfin, ma famille pour m'avoir comprise, avoir bien vécu mon départ et m'avoir soutenue même à distance tout au long de ces années.

Les modeles de taches permettent de representer des raisonnements utilises pour resoudre des problemes. Ils permettent de construire des Environnements de Resolution de Problemes (ERP) pour le pilotage d'une bibliotheque de programmes scientifiques. Une tache represente une classe de problemes (par ses donnees d'entree et de sortie) ainsi qu'une strategie pour les resoudre. Une strategie de resolution d'un probleme est composee d'une ou plusieurs etapes a accomplir pour resoudre le probleme. Chacune de ces etapes indique a son tour un nouveau probleme de moindre complexite lequel peut egalement etre represente par une nouvelle tache. Les decompositions successives d'une tache en sous-taches conduisent a une hierarchie de taches ou la complexite des problemes representes par les taches se reduit a chaque niveau. Les taches representees par les feuilles d'une hierarchie sont dites taches elementaires, elles representent des problemes dont la resolution correspond a l'execution d'un programme de la bibliotheque.

Or, differentes strategies peuvent etre decrites pour resoudre une meme classe de problemes. Par consequent, un ERP base sur un modele de taches peut contenir plusieurs taches indiquant ces differentes strategies. Ainsi, si l'on sait reconnaitre que deux taches representent une meme classe de problemes, on pourra presenter a l'utilisateur ces differentes options. Ce travail traite de la comparaison de structures de decomposition de taches en sous-taches. Il permet de verifier si deux taches, existantes dans un ERP pour le pilotage de bibliotheque de programmes, contiennent des strategies differentes pour resoudre une meme classe de problemes. La comparaison de strategies se fera par l'intermediaire des E-tache-graphes, definis dans cette these. Ce modele permet non seulement de representer le lien d'execution entre les programmes existants dans une bibliotheque mais aussi l'ensemble des sequences de programmes decrit par une strategie.

Abstract

Task models aim at representing reasoning for solving problems. They allow to build Problems Solving Environments (PSE) created as library guide tools. A *task* represents a problem, by its inputs and outputs, as well as a strategy for solving the problem. A strategy is composed by one or more steps which must be carried out in order to solve the problem. Each step indicates a new problem less complex which can be also represented by a new task. By analyzing recursively a strategy, a task hierarchy can be constructed. The problems represented in each level of such hierarchy are less and less complex. The leaves represent problems whose resolution is realized by the execution of a program in the library. The tasks represented by the leaves are named *elementary tasks*.

A same problem can be solved by different strategies. So, a PSE based on a task model can contain several tasks representing different strategies to solve a same problem. Thus, if we are able to recognize that two tasks represent a same class of problems, it will be possible to offer the users these different options. This work deals with the comparison of strategies in order to verify if two tasks, in a library guide tool based on task model, represent different strategies to solve a same problem. The comparison of strategies is based on a new representation of a strategy proposed in this work, named ***E-task-graphs***, which represents not only the program's execution link but also the set of *plans* described by a strategy. A plan means a sequence of elementary steps to solve a problem whose inputs and outputs are given.

Key words: Problem solving environments, task model, comparison of task graphs.

Table des matières

Introduction

Le contexte

Les modèles de tâches ont pour but de représenter des raisonnements utilisés pour résoudre des problèmes. Or, résoudre un problème consiste, en général, à organiser un *plan* ou une *stratégie* qui permet d'atteindre un but à partir d'un état initial. La résolution d'un problème représente donc une stratégie qui permet d'établir un lien entre les états initial (représenté par les données d'entrée) et final (représenté par les données de sortie). Une stratégie *indique les décompositions successives d'un problème en sous-problèmes* jusqu'à arriver à des problèmes de résolution simple. Prenons l'exemple du diagnostic (qui sera l'exemple sur lequel nous nous appuierons la plupart du temps) : à partir d'un constat, qui caractérise l'état initial (des symptômes donnés par des patients, l'arrêt d'une machine ou un "bruit étrange" dans notre voiture), d'abord on essaie de faire une liste d'hypothèses et ensuite on les évalue en sélectionnant celles qui sont les plus plausibles et en éliminant les autres ; finalement on aboutit à un diagnostic, qui caractérise l'état final (c'est-à-dire, notre but, "savoir ce qui se passe"). Les données du problème de diagnostic sont : (i) donnée d'entrée : le constat ; (ii) donnée de sortie : le diagnostic. La stratégie proposée est composée de deux étapes : l'énumération des hypothèses et la sélection des hypothèses les plus plausibles.

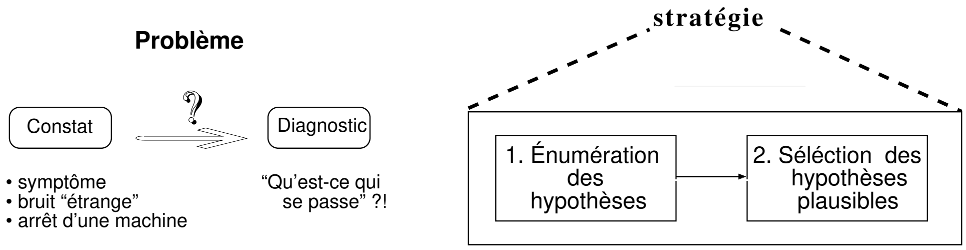


FIG. 1 – *Représentation du raisonnement utilisé pour faire un diagnostic.*

Un modèle de tâche permet de représenter une (ou plusieurs) stratégie(s), définie(s) pour résoudre une classe de problèmes. Sachant qu'un problème est caractérisé par son état initial et un but à atteindre (son état final), une *tâche* représente donc une classe de problèmes à travers ses données d'entrée et de sortie ainsi que une(des) stratégie(s) pour les résoudre.

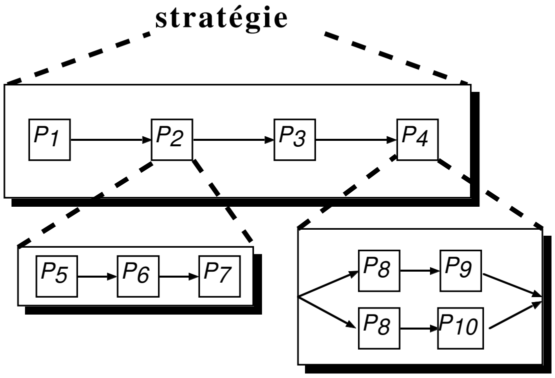


FIG. 2 – *Représentation d'une stratégie dans un modèle de tâches :* Le modèle de tâches permet de décrire une(des) stratégie(s) pour résoudre une classe de problèmes. Une stratégie comporte plusieurs étapes lesquelles peuvent indiquer des problèmes simples ou des problèmes qui nécessitent d'être encore détaillées (comme l'étape P_2 du dessin). Quelques unes peuvent être accomplies de diverses façons (comme l'étape P_4 du dessin).

Une stratégie exprimée par les décompositions successives d'une tâche **T** (qui représente une classe P de problèmes) en sous-tâches (qui représentent les sous-problèmes dans lesquels P se décompose) peut être représentée par une hiérarchie de tâches, dénommée par la suite *arbre de décomposition*, dont la racine est la tâche **T** et les feuilles sont les *tâches élémentaires* (qui représentent les problèmes non décomposables c'est-à-dire, problèmes de résolution simple). Cet arbre représente non seulement une séquence d'étapes qui composent la stratégie représentée par **T** (c'est-à-dire les étapes qui doivent être accomplies pour résoudre les problèmes P), mais elle indique également l'existence de différentes stratégies pour réaliser une des étapes, s'il en existe. La figure 3 illustre l'arbre de décomposition qui représente la stratégie illustrée par la figure 2.

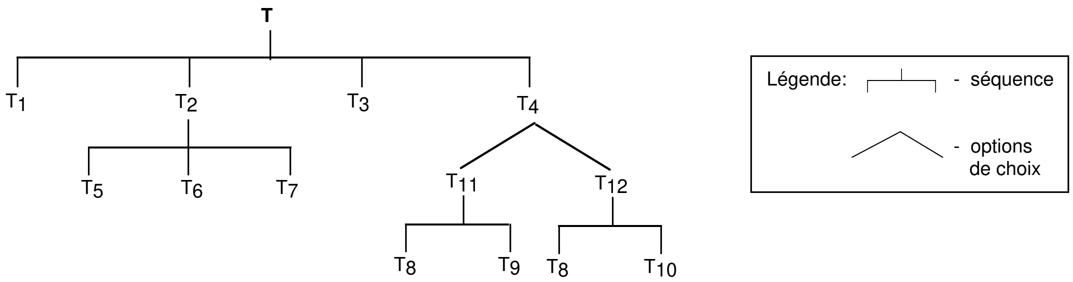


FIG. 3 – *Arbre de décomposition qui représente la stratégie illustrée par la figure 2.*

L'une des caractéristiques les plus marquantes des modèles de tâches est la façon *modulaire* de traiter un problème à résoudre, c'est-à-dire sa décomposition successive en sous-problèmes. Parmi les avantages d'une telle approche pour la résolution d'un problème on peut citer :

i) *la décentralisation (ou la répartition) des connaissances et l'utilisation des expertises :* les différentes étapes de résolution d'un problème peuvent être analysées, de façon indépendante, par différents spécialistes de chaque domaine concerné ;

ii) *la réutilisation de connaissances :* une tâche représente un problème et une stratégie pour le résoudre indépendamment de son utilisation. Elle peut être utilisée pour résoudre le problème en soi ou pour représenter une étape de résolution d'un problème plus complexe, ce qui caractérise son utilisation en tant que *sous-tâche* d'une tâche plus complexe.

Utilisation des modèles de tâches

Les *modèles de tâches* permettent de construire des Systèmes à Base de Connaissances [1] (SBC) capables de résoudre des problèmes complexes. Dans ces systèmes, les *tâches élémentaires* sont réalisées soit par une action "humaine", ou par l'exécution d'un programme ou encore par une inférence basée sur les connaissances du SBC.

Les SBC basés sur les modèles de tâches peuvent être séparés en deux groupes : les *conceptuels* et les *opérationnels*. Les premiers aident à l'acquisition des connaissances et la modélisation du raisonnement du SBC [All93, Cha92, Cha93b, Pie91, Sch94, Wie92]. Dans ces systèmes, la description d'une stratégie pour résoudre un problème reste abstraite (tâches génériques [Cha92, Cha93b], CommonKADS [Wie92], par exemple). Ces systèmes ont pour but de définir des méthodologies de résolution de problèmes. Les deuxièmes donnent origine à des Environnements de Résolution de Problèmes [2] (ERP), dans lequel une tâche élémentaire est réalisée automatiquement [Cha93a, Cle90, Eri95, Hai98, Mol95, Par98, Pue93, Rou88, Wil94], comme c'est le cas des systèmes qui permettent de piloter une bibliothèque de programmes ([Cha93a, Cle90, Hai98, Par98, Rou88, Wil94], par exemple). Divers travaux ont été réalisés dans le but de rendre opérationnels des modèles conceptuels ([Del93, Mol95, Eri95, Pue93, Tor98], par exemple).

La problématique

Une stratégie indique les étapes, plus ou moins complexes, de résolution d'un problème. Cependant, différentes stratégies peuvent être décrites pour résoudre un même problème. Par conséquent, différentes tâches peuvent représenter différentes stratégies pour résoudre une même classe de problèmes. Ainsi, un SBC basé sur un modèle de tâches peut contenir différentes tâches représentant des diverses stratégies pour résoudre une même classe de problèmes. Si l'on sait reconnaître que deux tâches représentent la même classe de problèmes, il est possible de proposer à l'utilisateur les différentes tâches existantes dans un SBC susceptibles de résoudre un problème donné.

La question qui se pose est : *Comment identifier que deux tâches représentent deux stratégies différentes pour résoudre un même problème* ? Ou si on va plus loin, *Est-ce que deux tâches*

1. Une base de connaissances permet de représenter ce qui est connu sur un domaine particulier. La représentation de connaissances consiste à modéliser les différents éléments du domaine, les relations existantes entre eux et leur comportement. Un SBC permet d'exploiter les connaissances existantes dans une base de connaissances.

2. Un ERP "est un SBC qui aide l'utilisateur à planifier et à contrôler l'exécution d'actions pour résoudre une catégorie de problèmes" [Cha93a].

représentent des stratégies pour résoudre des problèmes similaires ? Pour répondre à ces questions, il faut d'une part vérifier si les données (d'entrée et de sortie) des problèmes représentés par ces tâches sont identiques ou similaires ; d'autre part, comparer les stratégies représentées par les tâches.

La comparaison de données manipulées par deux tâches, existantes dans un SBC, est dépendante de la technique utilisée pour la représentation des connaissances du SBC, lors de son implémentation. Par exemple, pour un SBC qui utilise la représentation de connaissances par objets, comparer les données manipulées par deux tâches revient à comparer des attributs de deux objets *tâche*. Par contre, la comparaison de stratégies représentées par deux tâches **ne dépend pas** de la façon dont le système a été implémenté. En effet, la hiérarchie de tâches, qui représente une stratégie décrite dans une tâche, est propre à la stratégie qu'elle représente et est indépendante d'une implémentation. Ainsi, comparer deux stratégies, contenues dans un SBC basé sur un modèle de tâches, revient à comparer les deux hiérarchies de décomposition qui les représentent, indépendamment de la méthode utilisée pour l'implémentation du SBC.

Dans ce travail, nous nous intéressons à la comparaison de stratégies représentées par deux tâches qui manipulent des données (d'entrée et de sortie) identiques ou similaires. Autrement dit, nous nous attachons au problème de la comparaison de la *structure des décompositions successives de deux tâches*, représentant deux stratégies. Nous allons nous intéresser plus précisément aux *systèmes d'aide à la résolution de problèmes*, modélisés par des *tâches*, qui sont opérationnels (par exemple, ceux qui permettent de *piloter une bibliothèque de programmes*). L'intérêt d'une étude sur la comparaison de structures de décompositions de tâches est justement son utilisation indépendante de la méthode utilisée pour implémenter les systèmes. Évidemment, lors de la comparaison de deux stratégies représentées par des tâches, une comparaison préalable des données manipulées par les tâches comparées, ainsi que l'identification de la structure de décomposition de ces tâches, sont nécessaires.

Comment comparer deux stratégies représentées par des tâches ?

Étant donné qu'une stratégie, représentée par des décompositions successives d'une tâche en sous-tâches, peut être représentée par un *arbre de décomposition*, deux stratégies peuvent être considérées structurellement équivalentes si elles sont représentées par deux arbres identiques ou isomorphes [3] [Ham93].

Or, même si une stratégie représente les différents niveaux de décompositions récursives d'une classe de problèmes en sous-problèmes, lors de la résolution effective d'un problème dont les données ont été bien précisées, c'est une séquence d'étapes élémentaires (c'est-à-dire non décomposables) qui sera effectivement réalisée. Une telle séquence est dénommée *plan d'actions*. Une stratégie représente donc un ensemble de plans d'actions. En effet, si pour une des étapes qui composent une stratégie pour résoudre un problème P il existe différentes façons de

3. Deux arbres sont isomorphes si et seulement si après une réorganisation des sommets du premier, l'arbre obtenu est identique au deuxième.

la réaliser, alors il faut considérer l'existence d'une telle option de choix pour réaliser cette étape lors de la détermination d'un plan d'actions pour résoudre P. La figure 4 illustre un exemple d'une stratégie qui comporte deux plans d'actions.

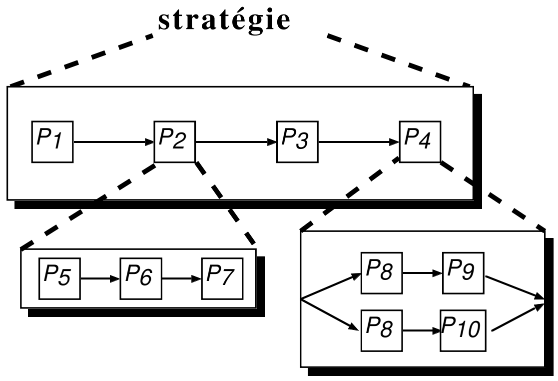


FIG. 4 – *Analyse d'une stratégie :* La stratégie représentée dans cette figure est composée de quatre étapes : la première et la troisième représentent des problèmes simples (non décomposables). La deuxième correspond à un problème encore complexe dont la stratégie associée comporte trois étapes, toutes les trois indiquent des problèmes simples. Finalement, pour la dernière étape, il a été possible de définir deux stratégies différentes : l'une composée de deux étapes correspondant à la résolution des problèmes simples P_8 et P_9 ; l'autre également composée de deux étapes, correspondant à la résolution des problèmes, également simples, P_8 et P_{10}. Cette stratégie de résolution comporte donc deux plans d'actions, représentés respectivement par la résolution des séquences de problèmes simples P_1 P_5 P_6 P_7 P_3 P_8 P_9 et P_1 P_5 P_6 P_7 P_3 P_8 P_{10}.

Deux stratégies qui représentent un même ensemble de plans d'actions sont *équivalentes*. Cependant, la simple vérification d'un isomorphisme entre des arbres de décomposition ne suffit pas car il se peut que deux arbres non isomorphes représentent le même ensemble des plans d'actions, comme l'illustre la figure 5.

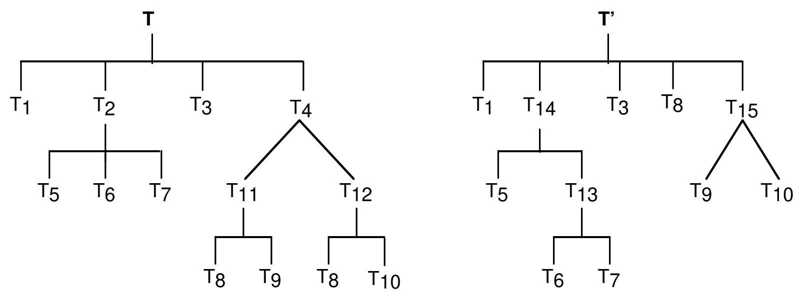


FIG. 5 – Ces deux arbres ne sont pas isomorphes et pourtant ils représentent le même ensemble des plans d'actions composé de deux plans : $\mathbf{T}_1$ $\mathbf{T}_5$ $\mathbf{T}_6$ $\mathbf{T}_7$ $\mathbf{T}_3$ $\mathbf{T}_8$ $\mathbf{T}_9$ et $\mathbf{T}_1$ $\mathbf{T}_5$ $\mathbf{T}_6$ $\mathbf{T}_7$ $\mathbf{T}_3$ $\mathbf{T}_8$ $\mathbf{T}_{10}$.

Dans le but de pouvoir identifier que deux stratégies sont équivalentes même si elles sont représentées par des arbres de décomposition non isomorphes, nous proposons une représentation de la structure d'une stratégie indiquée par une tâche, dénommée *E-tâche-graphe*, qui ne comporte que des étapes élémentaires. Autrement dit, un modèle qui met en évidence la représentation de l'ensemble des plans d'actions décrit par une stratégie plutôt que sa hiérarchie de décomposition. Les avantages les plus intéressants de ce modèle de représentation sont les propriétés de ses opérations de construction. Ces propriétés rendent possible la détermination de l'ensemble des plans d'actions décrits par une stratégie ainsi que la détermination d'une forme normale pour la représentation de sa structure. La comparaison de deux stratégies se fait donc par la comparaison des formes normales de leurs E-tâche-graphes respectifs.

Plan du mémoire

Ce mémoire est organisé comme suit :

Le **chapitre 1** présente quelques exemples de systèmes à base de connaissances conçus à l'aide des modèles de tâches. Puis, dans le but de comparer deux stratégies représentées par des tâches, deux types d'arbres de décomposition sont montrés. La sémantique d'une stratégie est également étudiée. Ensuite, nous montrons comment comparer les structures de décomposition de deux tâches à l'aide des *arbres de décomposition* et les limites d'une telle démarche de comparaison. Finalement, nous introduisons le modèle que nous proposons pour la représentation d'une stratégie, les *E-tâche-graphes*.

Le **chapitre 2** est dédié à l'étude des *E-tâche-graphes*. Nous présentons leurs opérations de construction et les propriétés de ces dernières. Nous définissons également l'opération d'*extension* d'un E-tâche-graphe laquelle permet de déterminer l'ensemble des plans d'actions décrit par la stratégie qu'il représente. Finalement, un système de réécriture est proposé pour déterminer la forme normale d'un E-tâche-graphe. Les règles de ce système se basent sur les propriétés des opérations de construction.

Le **chapitre 3** est réservé à la comparaison de stratégies en utilisant les E-tâche-graphes. Dans un premier temps, nous définissons la notion de *différence* entre deux E-tâche-graphes laquelle a pour but d'identifier les modifications nécessaires pour transformer la forme normale d'un E-tâche-graphe en celle d'un autre. Nous identifions à partir de cette différence une relation d'ordre partiel, *plus général que*, entre les E-tâche-graphes. Puis, nous analysons l'utilisation d'E-tâche-graphes, par rapport aux arbres de décomposition, pour comparer deux stratégies. Ensuite, nous montrons l'apport des E-tâche-graphes pour la comparaison de deux stratégies représentées par des tâches existantes dans un système d'aide à la résolution de problèmes basé sur un modèle de tâches. Pour ce faire, nous utilisons l'exemple de MYOSYS. Nous présentons d'abord l'application MYOSYS [Zie93, Mor96] (un système d'aide au diagnostic des maladies neuro-musculaires, réalisé en collaboration avec le projet SHERPA et le service d'électromyographie du CHU de Grenoble). MYOSYS a été développé en utilisant SCARP [Wil94], un environnement coopératif générique de résolution de problèmes basé sur le modèle de tâches. Finalement, nous faisons la comparaison de deux stratégies dont les arbres de décomposition de leur tâches respectives ne sont pas isomorphes mais, à travers leur E-tâche-graphes respectifs nous avons pu identifier qu'elles représentent le même ensemble des plans d'actions.

Ce document comporte cinq annexes :

Dans l'**Annexe A** nous donnons la terminologie et les notations de la théorie des graphes utilisées dans ce mémoire.

Dans l'**Annexe B** nous faisons une petite discussion sur l'analogie entre les E-tâche-graphes et les graphes orientés série-parallèle.

Dans l'**Annexe C** nous démontrons que la représentation exhaustive de l'ensemble des plans d'actions représenté par une stratégie (c'est-à-dire, l'expansion d'un E-tâche-graphe) peut résulter en un nombre exponentiel de plans. Nous donnons une borne inférieure du nombre de plans d'actions représentés par un E-tâche-graphe.

Dans l'**Annexe D** nous décrivons en détails l'algorithme pour obtenir la forme normale d'un E-tâche-graphe ainsi que des algorithmes auxiliaires pour la détermination de la différence entre deux E-tâche-graphes et ceux pour l'obtention de la forme normale d'un E-tâche-graphe.

Dans l'**Annexe E** nous évaluons la complexité des algorithmes proposés dans ce travail.

Chapitre 1

Modèle de tâches

Les modèles de tâches ont été introduits au début des années 80 par Chandrasekaran [Cha93b]. Ils ont pour but de représenter des raisonnements utilisés pour résoudre des problèmes.

Dans un modèle de tâches, une tâche représente une classe de problèmes, à travers ses donnés d'entrée et de sortie, ainsi qu'une stratégie pour les résoudre. Par *stratégie* nous entendons "les étapes qui doivent être accomplies pour résoudre un problème". Toutefois, les étapes qui composent une stratégie indiquent, a leur tour, des nouvelles classes de problèmes de moindre complexité. Ces dernières sont également représentées par des tâches [1]. L'analyse récursive d'une stratégie conduit à la construction d'une hiérarchie de classe de problèmes où à chaque niveau la complexité des problèmes se réduit. Ainsi, nous arrivons à des étapes qui indiquent des classes de problèmes dont la solution est simple.

Un modèle de tâches permet de représenter une (ou plusieurs) stratégie(s) définie(s) pour résoudre une classe de problèmes. Les modèles de tâches aident donc à la construction de Systèmes à Base de Connaissances (SBC), capables de résoudre des problèmes complexes.

Récemment, beaucoup de travaux ont porté sur la construction de SBC basés sur les modèles de tâches. Que ce soit des SBC à un niveau *conceptuel* où le but est d'aider à l'acquisition de connaissances et la modélisation du raisonnement du SBC pour développer une méthodologie de résolution de problèmes [All93, Cha92, Cha93b, Pie91, Sch94, Wie92] ; ou à un niveau *opérationnel*, où le but est la construction d'Environnements de Résolution de Problèmes (ERP) qui sont des SBC qui permettent la résolution automatique de problèmes [Cha93a, Cle90, Eri95, Hai98, Mol95, Par98, Pue93, Rou88, Wil94]. Quelques uns de ces travaux ont été motivés par un domaine particulier d'application (par exemple, biométrie [Rou88], traitement d'images [Cle90] ou traitement du signal [Hai98]). D'autres avaient pour but de proposer des systèmes génériques pour la conception de systèmes à base de connaissances

1. L'analyse d'une tâche mène à la description de comment elle peut être décomposée en sous-tâches de moindre complexité et quelles connaissances lui sont nécessaires [Cha92].

capables de résoudre des problèmes (par exemple, TRAM [Cha93a], LISA [Del93], KSM [Mol95], PROTÉGÉ-II [Eri95, Pue93], SCARP [Wil94], PowerTasks [Par98]). Parmi les opérationnels se trouvent des ERP pour piloter[2] une bibliothèque de programmes scientifiques [Cha93a, Cle90, Hai98, Par98, Rou88, Wil94]. De plus, quelques travaux envisagent le développement d'ERP *coopératifs* [Had96, Hai98, Mon96, Wil94] ; d'autres proposent des ERP *distribués* [Par98].

Les ERP *coopératifs* [Had96, Hai98, Mon96, Wil94] sont des SBC qui permettent "d'exploiter la complémentarité des compétences du système et de l'utilisateur dans la résolution d'un problème" [Had96]. Un ERP coopératif permet à l'utilisateur d'intervenir à tout moment dans le processus de résolution d'un problème que ce soit pour le guider complètement, ou pour choisir une tâche parmi des différentes tâches capables de résoudre un même problème, ou encore pour ajuster et/ou valider des résultats intermédiaires [Wil94]. Ainsi, la résolution d'un problème peut être vue comme un "processus d'essai-erreur" où les mêmes calculs sont effectués plusieurs fois en modifiant certains paramètres jusqu'à ce que l'utilisateur soit satisfait du résultat.

L'une des caractéristiques les plus marquantes des modèles de tâches est la façon modulaire d'analyser la résolution d'un problème, à travers les décompositions récursives du problème en sous-problèmes. Ainsi, les différentes étapes de résolution d'un problème peuvent être analysées par des experts de chaque domaine concerné, même s'ils se trouvent loin géographiquement. Un ERP *distribué* permet la définition et l'utilisation de (sous-)tâches (et de bibliothèques de programmes) qui sont sur plusieurs sites, ce qui facilite la diffusion de compétences [Par98].

Dans le but de vérifier si deux tâches existantes dans un ERP représentent des stratégies différentes pour résoudre un même problème, nous nous intéressons à la comparaison de stratégies indiquées par des tâches "candidates", c'est-à-dire des tâches qui manipulent des données (d'entrée et de sortie) identiques ou similaires. Une telle comparaison permettra : (i) dans un ERP coopératif, de proposer à l'utilisateur les différentes tâches qu'il contient lesquelles sont capables de résoudre une même classe de problèmes ; et (ii) dans un ERP distribué, de mieux gérer ses ressources par l'identification des sites qui contiennent des tâches capables de résoudre une même classe de problèmes.

Ce chapitre est organisé de la façon suivante : vu que dans les travaux qui portent sur la construction des SBC modélisés par des tâches on trouve une diversité de vocabulaire, nous fixons tout d'abord une terminologie laquelle sera utilisée dans ce travail (section 1.1). La section 1.2 est dédiée à la présentation des modèles de représentation de la structure de décomposition d'une tâche. Puis, nous présentons la sémantique de la décomposition des tâches (section 1.3). Ensuite, nous montrons quelques exemples de divers travaux concernant la construction de SBC basés sur des modèles de tâches (section 1.4) dans lesquels nous avons pu identifier deux approches différentes pour la représentation d'une stratégie. La section 1.5 montre comment

2. Choisir, enchaîner et exécuter des programmes existants dans une bibliothèque, lors de la résolution d'un problème, n'est pas toujours simple. Un système de pilotage rends plus aisé l'usage d'une telle bibliothèque. Dans ce cas, les tâches élémentaires sont réalisées par l'exécution de programmes.

comparer deux stratégies en utilisant les modèles de représentation de la structure d'une tâche étudiés dans la section 1.2 et les limites d'une telle démarche de comparaison. Finalement, dans la section 1.6 nous introduisons le modèle que nous proposons pour représenter une stratégie. Ce modèle sera étudié en détails dans le chapitre suivant.

1.1 Terminologie

Le but de cette section est de fixer une terminologie relative au modèle de tâches.

Tâche : une **tâche** représente une classe de problèmes à résoudre à travers ses donnés d'entrée et de sortie ainsi qu'une stratégie pour les résoudre.

Stratégie : définit "comment" résoudre un problème en décrivant les étapes de sa résolution. Autrement dit, une stratégie représente la décomposition d'une tâche en sous-tâches.

Tâche complexe (ou **tâche décomposable**) **:** une tâche qui représente une classe de problèmes dont la stratégie de résolution est composée de plusieurs étapes ou pour lesquels il existe plusieurs stratégies pour les résoudre. Dans le premier cas, la tâche est dite décomposée en *séquence* de tâches ; dans le deuxième, décomposée en *choix*. La stratégie représentée par une tâche complexe définit la façon dont elle est décomposée en sous-tâches.

Tâche élémentaire : une tâche qui représente une classe de problèmes dont la stratégie de résolution n'est composée que d'une seule étape. Une telle tâche représente une classe de problèmes de résolution simple. Ce sont les tâches qui peuvent être réalisées directement. Autrement dit, la stratégie associée à une tâche élémentaire est l'exécution d'un module connu lequel soit correspond à un programme existant dans une bibliothèque de programmes, soit correspond à une tâche (complexe ou non) pour laquelle une stratégie a déjà été définie (et par conséquent fait partie des connaissances du SBC).

Plan d'actions : une séquence d'une ou plusieurs actions élémentaires qui sera réalisée lors de la résolution effective d'un problème.

Tâche spécialisable (ou **tâche abstraite**) **:** une tâche dont le niveau de description est trop général. Elle sert, normalement, à regrouper un ensemble de tâches qui représentent des problèmes ayant des caractéristiques communes. Les tâches qui la spécialisent décrivent les cas particuliers représentés par chacune d'entre elles. Dans ce cas, lors de la résolution effective d'un problème, la stratégie utilisée est celle indiquée par la sous-tâche la plus spécifique possible correspondant à la description du problème en cours de résolution.

Sous-tâche : Il existe deux types de *sous-tâches* : celles qui ***composent*** des tâches complexes ; et celles qui ***spécialisent*** des tâches spécialisables.

1.2 Représentation de la structure de décomposition d'une tâche

La nature hiérarchique des décompositions successives d'une tâche en sous-tâches nous amène à une représentation de ces décompositions par un *arbre de décomposition*. La racine d'une hiérarchie de décomposition de tâches représente la tâche à décomposer tandis que les autres sommets représentent les sous-tâches qui la composent. Les arcs représentent les liens de décomposition entre les tâches. Au fur et a mesure que l'on descend dans l'arbre, la complexité des tâches représentées à ces niveaux se réduit jusqu'aux feuilles, qui représentent les tâches élémentaires, c'est-à-dire les tâches non décomposables (qui, dans le contexte de ce travail, sont réalisées par l'exécution d'un programme dans une bibliothèque de programmes).

La spécialisation d'une tâche peut être aussi représentée par un arbre où la racine représente une tâche à spécialiser et les autres sommets les sous-tâches qui la spécialisent. Ainsi, à une tâche qui est à la fois décomposable et spécialisable sont associées deux arbres : une de décomposition et l'autre de spécialisation.

Ci-après nous présentons deux types d'arbres pour la représentation des décompositions successives d'une tâche. La terminologie et les notations de la théorie des graphes utilisées dans cette section, et dans le chapitre suivant, sont données dans l'**Annexe A**.

1.2.1 Arbre de résolution

Dans son travail [Ham93], également consacré à la comparaison de structures de décomposition de tâches, R. Hamadi propose une représentation d'une stratégie qu'il dénomme *arbre de résolution*.

L'arbre **T** de décomposition d'une tâche est, selon Hamadi, un arbre aux sommets étiquetés dont les étiquettes sont formées par (i) soit un couple (*tch,str*) pour les tâches complexes (*tch* signifiant tâche et *str* identifiant la stratégie de décomposition de la tâche : *seq* pour *séquence*, *chx* pour *choix*, etc...) ; (ii) soit (*prd*) s'il s'agit d'une tâche élémentaire, c'est-à-dire représentée par une feuille.

Exemple 1 *Arbre de résolution d'une stratégie*

Reprenons l'exemple donné dans l'Introduction (figure 4, page 5). La figure 1.1 exhibe la structure de décomposition de la tâche qui représente cette stratégie et son respectif arbre de résolution.

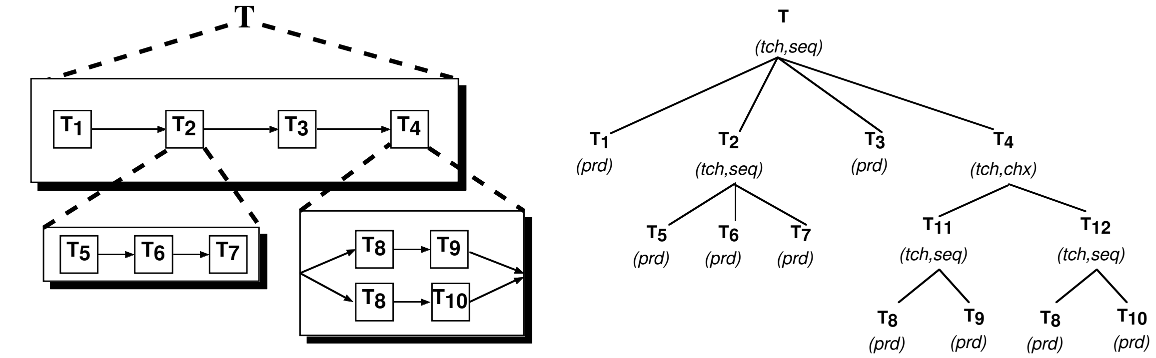


FIG. 1.1 – *Arbre de résolution associé à une tâche représentant une stratégie:* Les première et troisième étapes sont représentées respectivement par les tâches, élémentaires, $\mathbf{T}_1$ et $\mathbf{T}_3$. La deuxième étape, représentée par une tâche complexe $\mathbf{T}_2$, est composée d'une *séquence* de trois étapes représentées respectivement par les tâches, élémentaires, $\mathbf{T}_5$, $\mathbf{T}_6$ et $\mathbf{T}_7$. Pour la quatrième étape, représentée par la tâche $\mathbf{T}_4$, il existe une option de *choix* entre deux stratégies, composées toutes les deux de deux étapes : la première stratégie est représentée par la tâche $\mathbf{T}_{11}$ laquelle est décomposée en une séquence de deux sous-tâches ($\mathbf{T}_8$ et $\mathbf{T}_9$) ; la deuxième est représentée par la tâche $\mathbf{T}_{12}$ laquelle est également décomposée en une séquence de deux sous-tâches ($\mathbf{T}_8$ et $\mathbf{T}_{10}$).

1.2.2 Arbre ET/OU

Une stratégie décrite par une tâche dans un modèle de tâches peut *toujours* être représentée graphiquement par un arbre ET/OU (figure 1.2). En effet, différentes stratégies peuvent être établies pour résoudre un problème (nœud OU) ; ces stratégies sont composées d'une ou plusieurs étapes qui doivent être réalisées successivement (nœud ET). Toutefois, quelques étapes peuvent également être réalisées de plusieurs façons différentes (nœud OU). Dans un arbre ET/OU, un nœud ET représente donc une *séquence* d'étapes à accomplir ; un nœud OU, un *choix exclusif* entre différentes stratégies associées à une étape donnée. Ces options de choix peuvent être établies en fonction de la nature des données du problème. Dans ce cas, il s'agit d'une *spécialisation du problème*.

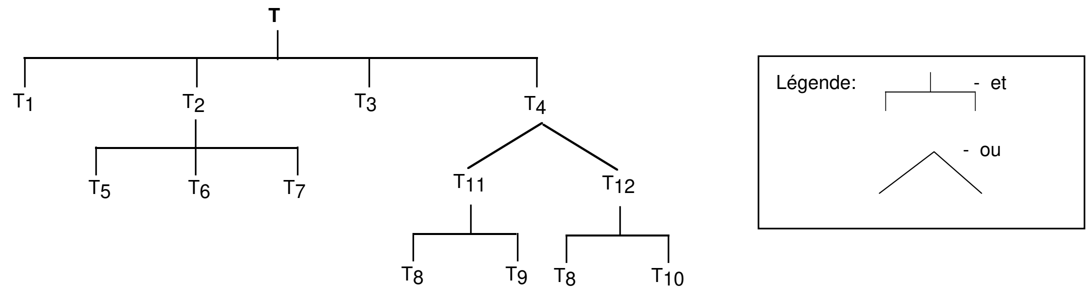


FIG. 1.2 – *Arbre ET/OU représentant la tâche illustré par la figure 1.1.*

Les *arbres de décomposition* présentés ci-dessus décrivent la hiérarchie de décomposition d'une tâche. Cependant, l'ensemble des plans d'actions décrit par une stratégie représentée par une tâche est identifiable grâce à la sémantique de la décompositions et de la spécialisation d'une tâche présentée dans la section suivante.

1.3 Sémantique de la décomposition et de la spécialisation de tâches

Un plan d'actions se construit dynamiquement lors de la résolution d'un problème donné et à chaque étape, si besoin est, un choix est fait : i) en fonction des données du problème, par une *spécialisation* ; ii) par l'utilisateur, dans le cas d'un système coopératif ; iii) par le système, en utilisant différents critères propres à chaque système. Néanmoins, l'ensemble $\mathcal{P}$ des plans d'actions décrits par la stratégie représentée par une tâche **T** peut être connu au préalable.

La détermination de $\mathcal{P}$ suit, de façon récurrente, le schéma suivant [Cra94] :

(i) si **T** est une *tâche complexe*,

• décomposée en *séquence*, $\mathcal{P}$ résulte de l'*enchaînement* des sous-plans représentés par les sous-tâches qui composent **T** ;

• décomposée en *choix*, $\mathcal{P}$ correspond à l'*union* des sous-plans représentés par les sous-tâches qui *composent* **T** ;

(ii) si **T** est une *tâche spécialisable*, $\mathcal{P}$ correspond à l'*union* des sous-plans représentés par les sous-tâches qui *spécialisent* **T**.

• Évidemment, lorsqu'une tâche spécialisable **T** est une sous-tâche d'une tâche décomposable **T′**, seules les spécialisations de **T** qui sont cohérentes avec les données de **T′** sont prises en compte. Prenons l'exemple d'une tâche **T′** qui manipule des matrices à valeurs réelles : seules les spécialisations de **T** qui s'appliquent à des valeurs réelles seront prises en compte pour la définition de l'ensemble des plans d'actions décrits par la stratégie représentée par **T′**. Les spécialisations de **T** qui s'appliquent à des valeurs complexes, par exemple, ne seront pas considérées.

(iii) si **T** est une *tâche élémentaire*, $\mathcal{P}$ correspond à la réalisation d'une seule étape.

Ainsi, une stratégie représente un ensemble de plan d'actions, si l'on développe tous les choix possibles dans la décomposition d'une tâche et de ses sous-tâches, et cela de façon récursive jusqu'à arriver aux tâches élémentaires. Les choix considérés sont représentés d'une part par un opérateur *choix* d'une décomposition et d'autre part par une *spécialisation* (*cf* figure 1.3).

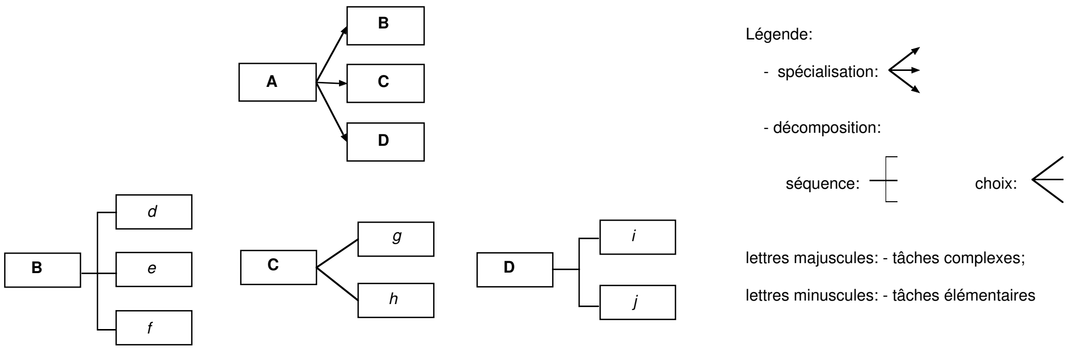


FIG. 1.3 – *Ensemble de plans d'actions décrits par une stratégie représentée par une tâche* . Les tâches **B**, **C** et **D** sont des spécialisations de la tâche **A**. L'ensemble P des plans d'actions décrits par A correspond donc à l'union des ensembles de plans d'actions décrits par **B**, **C** et **D**. Or, **B** et **D** sont décomposées en *séquence* de tâches élémentaires. Les ensembles des plans d'actions décrits par **B** et **D** sont respectivement, $\{def\}$ et $\{ij\}$. Par contre, **C** est décomposée en *choix* entre deux tâches élémentaires ; elle décrit donc l'ensemble des plans d'actions $\{g, h\}$. Finalement, P est composé de 4 plans d'actions : $\{def, g, h, ij\}$.

Nous avons présenté, respectivement dans la section précédente et dans cette section, d'une part deux types d'arbres comme modèle de représentation de la structure de décomposition d'une tâche et d'autre part, la sémantique de la décomposition et de la spécialisation d'une tâche. Pour ce faire, nous ne nous sommes basés que sur les concepts propres aux modèles de tâches ; autrement dit, les décompositions successives d'une tâche en sous-tâches ainsi que celui de la spécialisation de tâches. Ci-après, nous présentons quelques exemples de SBC basés sur des modèles de tâches ; mais auparavant nous présentons les deux approches différentes, que nous avons pu identifier dans ces travaux pour la représentation d'une stratégie.

1.4 Approches pour la représentation de la décomposition d'une tâche

Nous avons identifié deux façons de représenter une stratégie qui représente les étapes de résolution d'un problème, dans un SBC modélisé par des tâches et cela indépendamment du niveau, conceptuel ou opérationnel, du système :

i) la description d'un problème, en fonction de ses données d'entrée et de sortie, et de sa stratégie de résolution est faite à travers une seule entité *tâche* [Par98, Pie91, Wie92, Wil94]. Cette entité possède un attribut qui indique la stratégie de résolution lequel est dénommé *stratégie* dans SCARP [Wil94], *corps* dans TASK [Pie91] ou encore *task-body* dans Power-Tasks [Par98] et dans CommonKADS [Wie92].

ii) l'énoncé d'un problème et la description de la façon de le résoudre sont décrits séparément [Cha93a, Cha93b, Cha92, Eri95, Mol95, Mol96, Pue93]. Dans cette approche, une *tâche* représente un type de problème à résoudre à travers ses données d'entrée et de sortie. À chaque tâche est associée une (ou plusieurs) *méthode(s)* qui représente(nt) un(des) façon(s) de résoudre le problème. Autrement dit, une *méthode* décrit une stratégie de résolution. Par conséquent, chaque méthode est à son tour composée d'une (ou plusieurs) *sous-tâche(s)* qui représente(nt) des nouveaux problèmes. Cette approche possède une structure d'arbre de *tâches/méthodes/sous-tâches* [Cha92, Mol96].

La distinction entre ces deux approches est que dans la première une stratégie est représentée directement dans une tâche. Par contre, dans la deuxième, une stratégie constitue une *méthode de résolution d'un problème* et est décrite séparément ; c'est donc la méthode qui est composée de une (ou plusieurs) étape(s), chacune d'entre elles également représentée par une tâche ; le choix entre diverses stratégies, dans cette approche, se fait par le choix entre diverses méthodes associées à une tâche. Cependant, dans les deux approches, la résolution d'un problème passe par décompositions successives d'une tâche en sous-tâches.

Ci-après nous présentons succinctement quelques exemples de travaux qui portent sur la construction de SBC basés sur des modèles de tâches, conçus selon chacune de ces approches.

Ceux qui utilisent l'approche d'une seule entité *tâche* sont présentés dans la section 1.4.1 ; les travaux qui utilisent l'approche *tâche/méthode/sous-tâche* sont présentés dans la section 1.4.2. Nous ne nous occupons pas ici de donner des détails de la manière dont les systèmes ont été implémentés. Ce qui nous intéresse c'est illustrer la méthodologie utilisée pour représenter une stratégie de résolution d'un problème.

1.4.1 Une entité unique *Tâche*

Nous présentons quatre exemples de travaux qui ont suivi l'approche d'une seule entité tâche. Les deux premiers sont dédiés à la construction d'ERP pour le pilotage de bibliothèques de programmes. Les deux derniers portent sur l'acquisition de connaissances.

SCARP [Wil94], développé au sein du projet SHERPA, est un ERP générique qui a pour but de piloter des bibliothèques de programmes en coopération avec l'utilisateur. Autrement dit, l'utilisateur peut interférer soit en choisissant une stratégie qu'il considère la mieux adaptée pour résoudre un problème particulier, soit en donnant des paramètres qui seront pris en compte dans la suite d'une résolution. Diverses applications ont été développées en utilisant SCARP. On peut citer, par exemple : i) SLOT [Che93, Che94] dans le domaine de l'analyse de données ; ii) Said [Pre91] pour le diagnostic vibratoire de plate-formes *off-shore*, réalisé à l'IFREMER de Brest ; iii) MYOSYS [Zie93, Mor96] pour l'aide au diagnostic des maladies neuro-musculaires, réalisé en collaboration avec le service d'électromyographie du CHU de Grenoble. La figure 1.4 montre la description d'une tâche dans SCARP.

```
{<tâche>   est-un      =         tâche;
   &entrée <entrée>    $un       objet ;
   &sortie <sortie>    $un       objet ;
           stratégie   $défaut   (séquence <liste-de-tâches>) |
                                 (choix <liste-de-tâches>)
                                 (choix-utilisateur <liste-de-tâches>)
                                 ...
   &stache <tâche-1>   $un  { contexte
                                  exec  $un  <tâche> |
                                             <procédure> } }
```

```
{<procédure>  est-un     =        procédure ;
   &entrée    <entrée>   $un      objet ;
   &sortie    <sortie>   $un      objet ;
              nom-fct    $valeur  <programme> }
```

FIG. 1.4 – *Description (partielle) d'une tâche en SCARP :* L'attribut *stratégie* définit les étapes de résolution du problème représenté par la tâche. L'attribut *exec* décrit la façon dont chacune de ces étapes sera réalisée. La réalisation d'une etape à travers l'exécution d'un programme est indiquée par le nom d'une procédure représentant ce programme.

Les étapes de résolution d'un problème, représentées par une tâche, sont décrites dans SCARP par l'attribut *stratégie*. Le contrôle d'exécution de ces étapes est fait à l'aide d'opérateurs comme *séquence, choix, choix-utilisateur* entre autres. À chacune de ces étapes il est associé un attribut *exec* qui contient (i) le nom d'une tâche, dans le cas d'une étape encore complexe ou (ii) le nom d'une procédure, dans le cas d'une étape représentée par une tâche *élémentaire*, c'est-à-dire réalisable par l'exécution d'un programme.

Dans SCARP, il existe plusieurs façons de décrire différentes stratégies pour résoudre un même problème :

i) par l'opérateur *choix*, décrit dans l'attribut stratégie. Dans ce cas, lors de la réalisation de la tâche, le choix de la stratégie est fait par le système ;

ii) par l'opérateur *choix-utilisateur*, décrit également dans l'attribut stratégie. Dans ce cas, le choix de la stratégie à utiliser lors d'une résolution d'un problème particulier, est fait en coopération avec l'utilisateur ;

iii) par une *spécialisation* de la tâche. Dans SCARP, une tâche spécialisable peut indiquer une stratégie. Cependant, cette stratégie ne sera utilisée que dans le cas où aucune des spécialisations de la tâche n'est adaptable au problème en cours de résolution.

La résolution d'un problème passe donc par une alternance de phases de spécialisation et de décomposition (figure 1.5). Chaque tâche (ou sous-tâche) n'est soumise à la phase de spécialisation qu'au début de sa propre exécution.

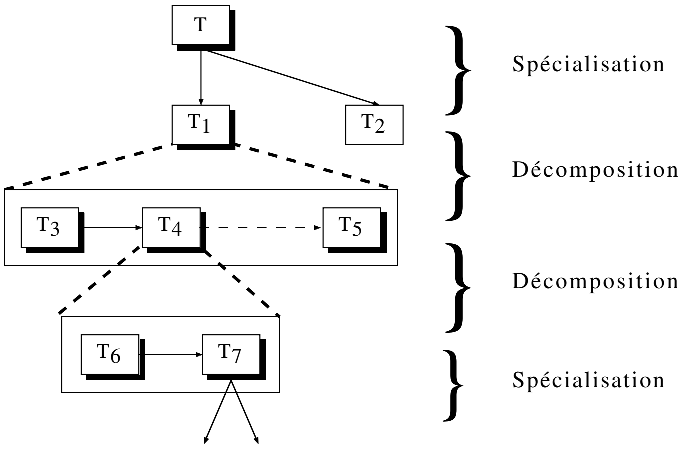


FIG. 1.5 – *Alternance des phases de spécialisation et de décomposition dans SCARP :* La tâche **T** doit être réalisée. Selon les caractéristiques des entrées, après une phase de *spécialisation*, la tâche $\mathbf{T}_1$ est choisie. Cette tâche est décomposée selon la stratégie indiquée. Ses sous-tâches sont à leur tour exécutées. La tâche $\mathbf{T}_4$ est décomposée en deux sous-tâches et, comme elle n'est pas spécialisable, sa *décomposition* est réalisée directement, et ainsi de suite. Une spécialisation sera réalisée pour la tâche $\mathbf{T}_7$.

PowerTasks [Par98] est un ERP générique coopératif développé par la société Ilog, grâce à un transfert technologique des études réalisées lors du développement de SCARP. Un système coopératif d'aide à l'analyse de séquences génomiques (ImaGene [Med95]) a été développé en utilisant PowerTasks, dans le cadre du projet GREG qui, outre le projet SHERPA et la société Ilog, avait comme partenaire l'Université Claude-Bernard (Lyon) et l'Institut Pasteur (Paris). Une étude sur un ERP distribué (*PowerTasks distribué*) est en cours, dans le cadre d'une thèse. Un tel ERP permet d'accéder à des tâches et/ou bibliothèques existantes dans des sites différents.

TASK [Pie91] est un système d'acquisition de connaissance qui permet d'expliciter le processus de résolution d'un problème. Dans TASK, une *tâche* "est une entité structurée qui rassemble toutes les informations impliquées dans la résolution d'un problème sous la forme d'une association [Contexte / Traitement], qu'il s'agisse d'un problème générique ou d'un problème plus particulier, d'un problème complexe (se décomposant en plusieurs sous-problèmes ou faisant appel à plusieurs alternatives) ou bien d'un problème élémentaire". Le *Traitement* indique toutes les méthodes permettant d'accomplir une tâche. La structure d'une tâche dans TASK est montrée en figure 1.6.

partie contexte
- état-initial
- état-final
- but
- paramètres

partie exécutive
- **corps**

partie relationnelle
- liste-généralisations
- liste-spécialisations
- liste-coopérations

FIG. 1.6 – *Structure d'une tâche* **T** *en TASK :* La *partie contexte* caractérise le problème par ses états initial et final ; l'attribut *but* explicite le but recherché en exécutant la tâche **T**. C'est dans l'attribut *corps*, de la *partie exécutive*, que l'on décrit la stratégie pour résoudre le problème représenté par la tâche **T**. Le corps contient une action, un enchaînement de sous-tâches ou encore une alternative entre différentes tâches. La *partie relationnelle* décrit une relation existante entre **T** et les autres tâches contenues dans le SBC, que ce soit une relation de *spécialisation/généralisation* ou de *coopération*.

Dans TASK, les tâches sont caractérisées, selon le type de leur *corps*, comme suit :

• *Tâches Élémentaires*, dont le corps est une *action*. Ce sont les tâches dont le niveau opérationnel est caractérisable par une entité procédurale insécable.

• *Tâches Composées*, dont le corps est une *structure* pré-définie. Le niveau opérationnel correspond (i) soit à un enchaînement invariable et bien connu de sous-tâches, (ii) soit à une alternative entre différentes tâches. Dans le premier cas, les tâches sont dites *Tâches Figées* puisque la définition de l'enchaînement des sous-tâches est *statique*, préétabli et invariable. Dans le deuxième, elles sont dites *Tâches Alternatives* puisque l'enchaînement de sous-tâches est défini *dynamiquement* lors de la résolution effective d'un problème.

• *Tâches Abstraites*, dont le corps n'est pas prédéterminé. Comme les tâches à spécialiser, une *Tâche Abstraite* **T** dans TASK est utilisée pour représenter au sein d'une même entité les caractéristiques communes relatives à un ensemble de problèmes. Elle décrit un problème général mais ne définit pas de stratégie pour le résoudre. Celle-ci est déterminée au cours d'une résolution et correspond au traitement défini pour la sous-tâche de **T** dont le contexte correspond à celui de la situation donnée.

Les Tâches Élémentaires et les Tâches Figées décrivent donc des stratégies *statiques*. Par contre, pour les Tâches Alternatives et les Tâches Abstraites la stratégie utilisée pour résoudre un problème donné est déterminée *dynamiquement*. La figure 1.7 illustre le processus de génération dynamique d'un plan d'actions pour une tâche abstraite, par exemple.

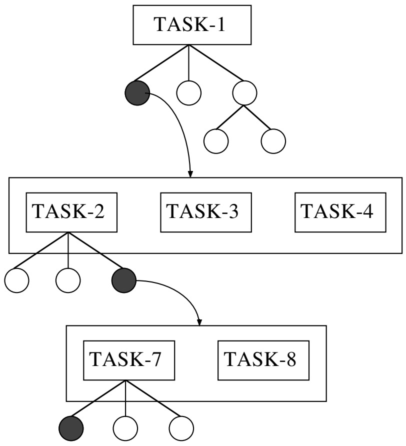


FIG. 1.7 – *Génération dynamique d'un plan d'actions dans TASK :* Une spécialisation de TASK-1 adaptée à son contexte est sélectionnée (cercle noir), dont la décomposition est TASK-2, TASK-3 et TASK-4. De même pour TASK-2 dont la spécialisation retenue est décomposée en TASK-7 et TASK-8. Ce procédé crée dynamiquement l'enchaînement de sous-tâches de la tâche TASK-1.

CommonKADS [Wie92] est une méthodologie pour le développement d'un SBC, issue des travaux menés dans deux projets ESPRIT (KADS-I et KADS-II). Le modèle de tâches n'est qu'une partie de cette méthodologie qui comporte cinq autres modèles, comme l'illustre la figure 1.8.

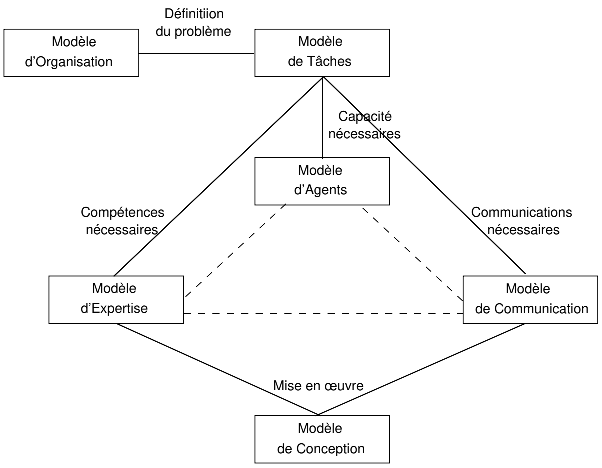


FIG. 1.8 – *Composants de la méthodologie KADS (transcrite de [Wie92] :* 1. Le *modèle d'organisation* permet d'analyser une organisation et de décrire des fonctions qui ont besoin d'un support ; 2. Le *modèle de tâches* récupère donc les "tâches" (ici, "tâches", entre guillemets, a le sens *français* du terme : *travail à réaliser*) qui sont associées à ces fonctions, y compris la relation "tâche"/agent ; 3. Le *modèle d'agents* décrit la capacité des agents ; 4. Le *modèle d'expertise* décrit les compétences nécessaires pour réaliser une "tâche" ; 5. Le *modèle de communication* décrit les communications nécessaires entre les agents ; 6. Finalement, le *modèle de conception* décrit la structure et les mécanismes des outils (normalement des SBC) associés à la "tâche".

Les connaissances manipulées dans CommonKADS sont divisées en deux catégories : les connaissances du domaine (connaissances statiques) et les connaissances de contrôle. Les premières contiennent des concepts, des relations et des faits concernant un domaine d'application. Les deuxièmes définissent la façon dont les connaissances du domaine doivent être exploitées pour résoudre les problèmes posés. C'est dans ces dernières que s'inscrivent les connaissances sur les tâches.

La spécification d'une tâche dans CommonKADS comporte deux parties : i) *task definition*, qui décrit le but de la tâche, c'est-à-dire *ce qui* doit être réalisé ; ii) *task body*, qui décrit les activités nécessaires pour accomplir la tâche, c'est-à-dire *comment* atteindre le but. La figure 1.9 montre les paramètres de définition d'une tâche en CommonKADS.

```
task < name >
    task-definition
        goal : description du but à atteindre
        input :
        output :
    task-body
        type : < composite > | < primitive > | < transfer >
        sub-tasks :
        additional-roles :
        control-structure :
end
```

FIG. 1.9 – *Paramètres de définition d'une tâche dans CommonKADS :* Dans **task-definition** le but à atteindre est décrit et les données d'entrée et de sortie sont définies. Le **task-body** décrit la façon dont la tâche sera réalisée. L'attribut **type** dans **task-body** définit le type de la tâche. Elle peut être de trois types : (i) *composite*, qui doit être décomposée en sous-tâches ; (ii) *primitive*, qui ne peut pas être décomposée (dite tâche "terminale") ; ou encore (iii) *transfer*, qui décrit une interaction avec "le monde extérieur". L'attribut **sub-tasks** contient la liste de sous-tâches qui la composent. Le contrôle d'exécution des sous-tâches est décrit à travers l'attribut **control-structure**. Les données "internes" à la tâche sont définies dans **additional-roles**.

1.4.2 Deux entités distinctes : *Tâche* & *Méthode*

Nous présentons six exemples de systèmes qui ont suivi l'approche *tâche/méthode/sous-tâches*. Dans [Cle90, Del93], même si les auteurs n'utilisent pas le même vocabulaire[3] on y trouve la même approche *tâche/méthode/sous-tâches*. Autrement dit, la définition d'un problème est faite séparément de la description d'une stratégie pour le résoudre ; cette dernière pouvant être la décomposition de la tâche en sous-tâches.

Tâches Génériques [Cha93b, Cha92] Le but des *tâches génériques* est de proposer une méthodologie de résolution d'une classe de problèmes. Une tâche générique représente donc un type de problème, indépendamment du domaine concerné. Par exemple, le *Diagnostic* est une tâche. Que ce soit un diagnostic médical ou d'une panne d'une machine. La tâche *Diagnostic* représente "un type de problème dont l'objectif est de déterminer les causes d'un comportement anormal d'un système". La description d'une tâche ne contient aucune spécification du *comment* la tâche sera accomplie ; elle contient les état initial et final du problème. Dans l'exemple du diagnostic, l'état initial est représenté par l'observation

3. Ils dénomment *but* la représentation d'un problème.

d'un mauvais fonctionnement, et l'état final, par des informations à propos des causes de ce mauvais fonctionnement.

Les moyens d'accomplir une tâche sont décrits par des *méthodes*. Dans [Cha92], l'auteur définit *méthode* comme "un ensemble de sous-tâches qui peuvent être réalisées pour transformer l'état initial d'une tâche en son état final". Toujours dans l'exemple du *Diagnostic*, trois méthodes sont identifiées pour réaliser la tâche : *Explanation Bayesienne, Assemblage Abductif* et *"Cover-and-differentiate"*. Ces trois méthodes peuvent être classifiées comme des *Méthodes Abductives*. La méthode *Assemblage Abductif* comporte deux sous-tâches : *Générer des Hypothèses Plausibles* et *Sélectionner Hypothèses*. La figure 1.10, transcrite de [Cha92], illustre une partie de la description de la tâche *Diagnostic*. La structure d'une tâche est donc un arbre de *tâche, méthode* et *sous-tâches*, appliqué récursivement jusqu'à aboutir à des tâches qui soient, par un moyen ou l'autre, réalisables directement.

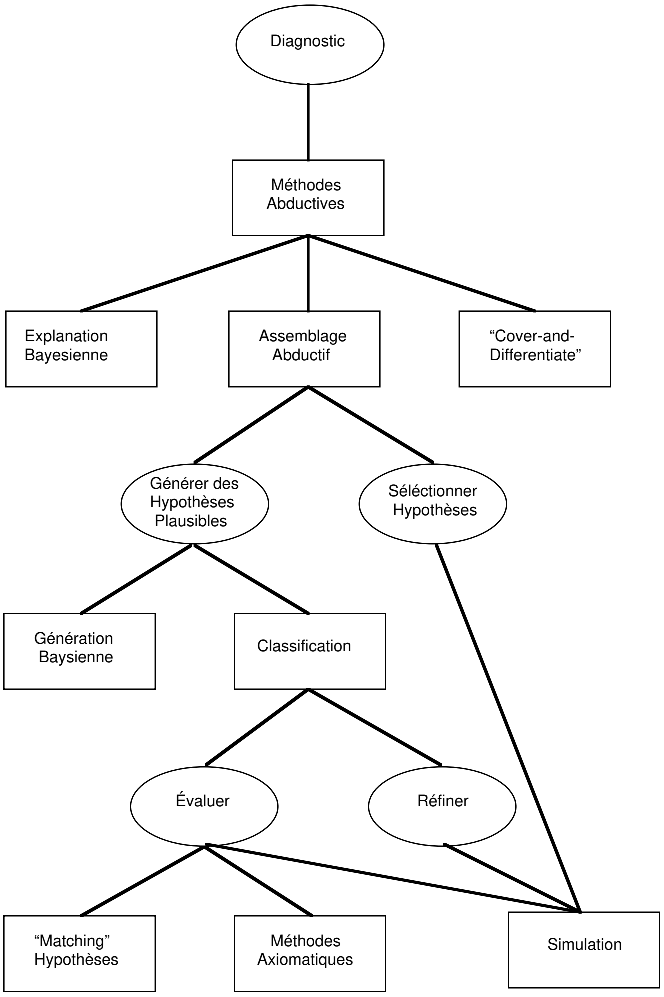


FIG. 1.10 – *Description de la tâche* **Diagnostic** *selon [Cha92] :* Les cercles représentent des tâches et les rectangles, des méthodes. À une tâche il peut y avoir diverses méthodes associées (par exemple, pour la tâche *Générer des Hypothèses Plausibles* il y a deux méthodes : *Génération Bayesienne* et *Classification*) et une méthode peut être composée de diverses tâches (par exemple, la méthode *Classification* est décomposée en deux sous-tâches : *Évaluer* et *Raffiner*).

La structure de tâche, dans le contexte des tâches génériques, n'est qu'un outil d'analyse pour déterminer les étapes de résolution d'un problème. Une mise en œuvre d'un système ne doit pas obligatoirement avoir une "correspondance biunivoque"[4] avec la structure d'une tâche. La structure d'une tâche fournit simplement un vocabulaire pour décrire "comment" un système peut réaliser une tâche.

KSM [Mol95, Mol96] (Knowledge Structure Manager) permet de développer un modèle conceptuel qui est traduit automatiquement en une version opérationnelle du modèle. Pour la création d'un SBC, KSM se base sur le concept d'*unités de connaissances*. Une *unité de connaissances* représente la compétence d'un expert dans un domaine donné. Elle contient non seulement la description des concepts propres au domaine concerné (*description cognitive*) mais également les actions, liées à ces concepts, lesquelles doivent être exécutées par l'expert (*description fonctionnelle*). Les connaissances d'un domaine sont subdivisées en *sous-domaines* en utilisant des nouvelles unités de connaissances (celles-ci associées aux sous-domaines identifiés). Les unités de connaissances qui ne sont pas subdivisées en sous-domaines sont dénommées *unités primaires de connaissances* (figure 1.11).

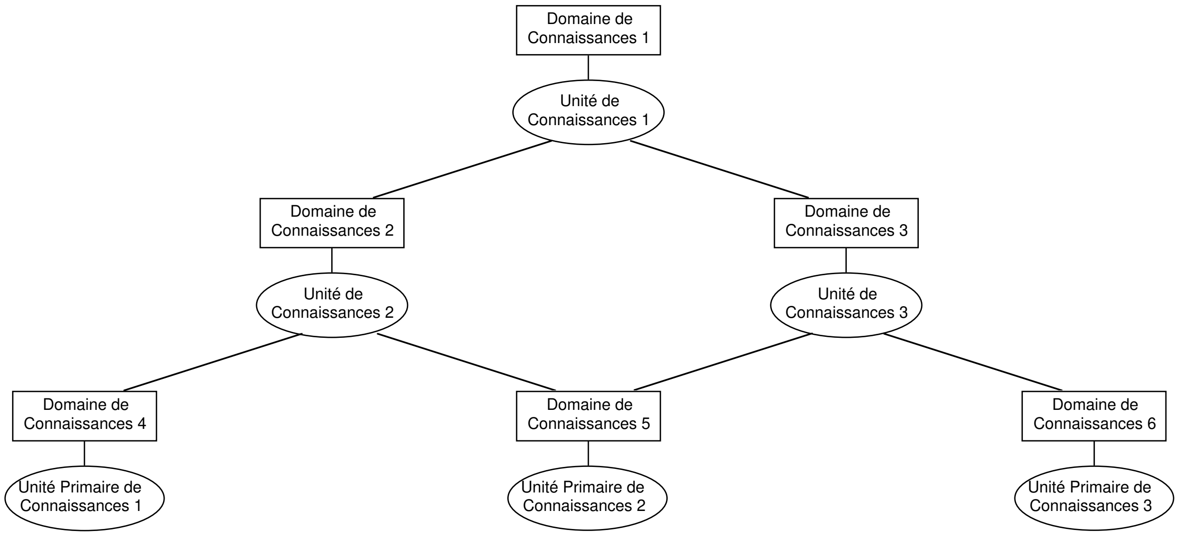


FIG. 1.11 – *Représentation de connaissances dans KSM* (transcrite de [Mol95]) : Les connaissances d'un domaine sont représentées par des *unités de connaissances* qui sont à leur tour subdivisées en sous-domaines. Les *unités primaires de connaissances* représentent les unités de connaissances indivisibles.

Une fonctionnalité d'une unité de connaissances, c'est-à-dire les actions contenues dans sa description fonctionnelle, est représentée à l'aide d'une *tâche*. Une tâche reçoit un ensemble de données d'entrée et fournit comme résultat un ensemble de données de sortie. Le processus d'exécution d'une tâche est représenté par des *méthodes*. Ainsi, une tâche associée à une unité de connaissances doit être divisée en autres tâches en utilisant des méthodes, comme l'illustre la figure 1.12. Une méthode indivisible représente une fonctionnalité d'une unité primaire de connaissances.

4. *Correspondance biunivoque* entre deux ensembles donnés signifie "à chaque élément d'un ensemble correspond un et un seul élément de l'autre". Dans le contexte ci-dessus cela signifie qu'une tâche décrite dans un système conçu en utilisant l'approche des *tâches génériques* n'est pas obligée de correspondre à un seul "élément informatique" lors d'une implémentation puisque, justement le but de ces systèmes est simplement d'offrir une méthodologie pour décrire comment résoudre un problème.

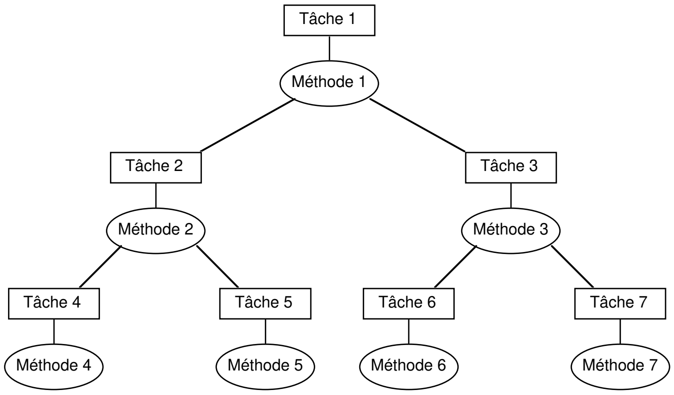


FIG. 1.12 – *Relation tâche/méthode/sous-tâches dans KSM* (transcrite de [Mol95]).

L'opérationnalisation du modèle conceptuel se fait à travers le langage LINK qui permet de décrire des procédures d'inférences (dénommées *primitives de représentation*) sur les unités primaires de connaissances du modèle conceptuel (figure 1.13). KSM gère une bibliothèque de primitives de représentation où chaque élément est un module réutilisable pour la construction de nouvelles méthodes.

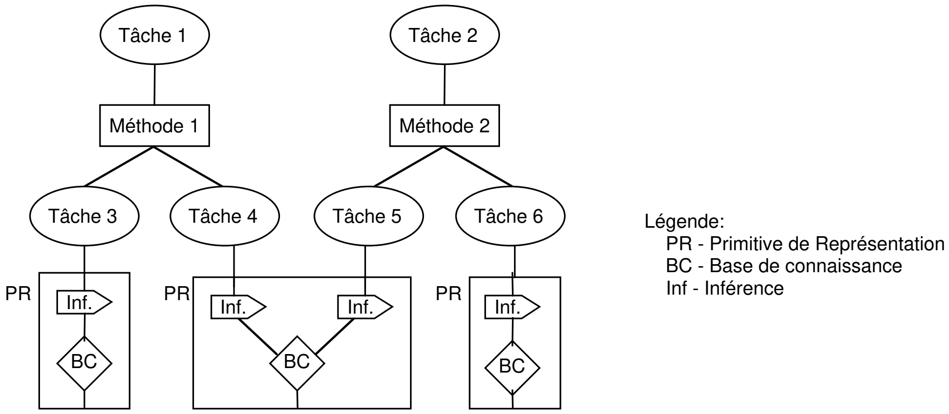


FIG. 1.13 – *Opérationalisation d'une tâche dans KSM* (transcrite partiellement de [Mol96]) : Les méthodes indivisibles sont dénommées *primitives de représentation* et réalisent une *inférence* sur les connaissances du SBC.

PROTÉGÉ-II [Eri95, Mol96, Pue93] Dans PROTÉGÉ-II, une *tâche* représente une activité (ou une abstraction d'une activité) du monde réel. Elle reçoit des données d'entrée et produit des données de sortie. Une tâche elle-même ne spécifie pas "comment" produire les données de sortie à partir des données d'entrée. Cela est fait à travers des *méthodes*. L'application d'une méthode à une tâche résulte souvent en sa décomposition en sous-tâches. Une méthode qui ne décompose pas une tâche en sous-tâches est dénommée *mécanisme*. Les mécanismes sont définis comme "des blocs de construction pour composer des méthodes et qui peuvent être utilisés dans la composition de différentes méthodes" [Pue93]. Ayant comme l'un de ses principaux objectifs la réutilisation de connaissances, ce système permet de construire une *bibliothèque de mécanismes* laquelle contient, outre les mécanismes eux-mêmes, des méthodes et des tâches. Cette bibliothèque "sert de magasin de composants réutilisables" [Pue93]. Ainsi, il est possible construire des nouvelles méthodes par la composition de mécanismes existants dans la bibliothèque. La figure 1.14

illustre la structure d'une tâche en PROTÉGÉ-II[5].

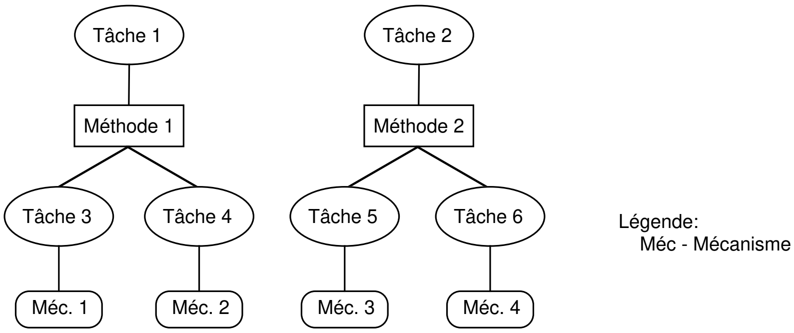


FIG. 1.14 – *Structure d'une tâche en PROTÉGÉ-II* (transcrite partiellement de [Mol96]).

TRAM [Cha93a] est un ERP générique pour le pilotage de bibliothèques de programmes qui est dédié à des *systèmes réactifs*, c'est-à-dire des systèmes qui "réagissent aux évolutions de l'environnement physique en prenant en compte les données issues de capteurs". Dans ces systèmes, une action réactive correspond à la définition d'une réponse que le système doit fournir aux événements qui surviennent pendant la résolution. Une telle réponse est représentée par une *tâche*. À chaque tâche sont associée(s) une (ou plusieurs) méthode(s). Lors de l'exécution d'une tâche, un gestionnaire de tâche sélectionne une méthode, la mieux adaptée à la situation courante, parmi celles qui sont associées à la tâche. Une méthode correspond soit à une décomposition de la tâche en sous-tâches soit à la réalisation directe d'une action. Dans le premier cas, la tâche est dite *composite* ; dans le deuxième, *élémentaire*.

L'exemple donné dans [Cha93a] est celui d'une bibliothèque de calcul matriciel (NAG). À la tâche *inversion d'une matrice* sont associées plusieurs méthodes qui sont organisées en fonction des caractéristiques de la matrice à inverser, comme l'illustre la figure 1.15 (transcrite de [Cha93a]).

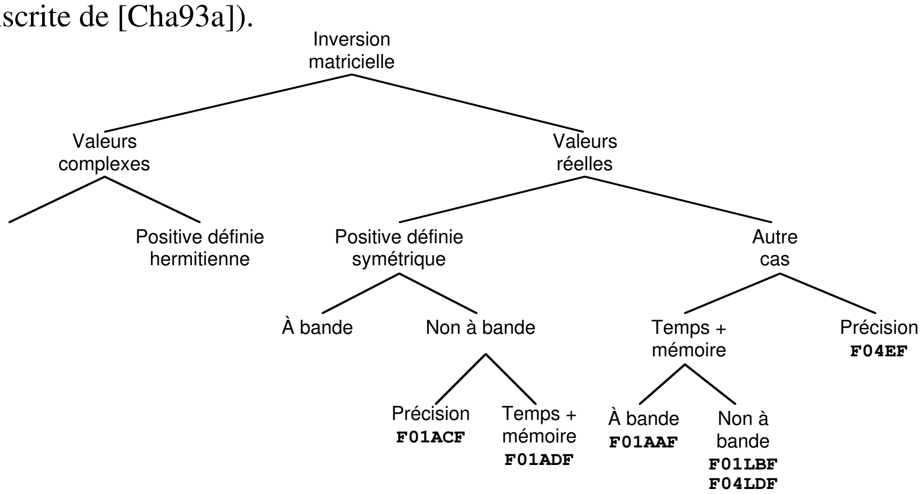


FIG. 1.15 – *Méthodes associées à la tâche inversion d'une matrice :* Les méthodes sont classées en fonction des caractéristiques de la matrice à inverser. Les feuilles de la hiérarchie de spécialisation de méthodes sont associées à des programmes de la bibliothèque NAG.

5. Dans [Mol96], les auteurs ont fait une étude comparative entre KSM et PROTÉGÉ-II. Les deux systèmes offrent des moyens de rendre opérationnels des modèles conceptuels de représentation de connaissances et permettent de construire une bibliothèque de modules et de réutiliser les modules existants dans cette bibliothèque. Les modules réutilisables de KSM étant les *primitives de références* et ceux de PROTÉGÉ-II, les *mécanismes*.

OCAPI [Cle90] a été développé pour piloter une bibliothèque de programmes de traitement d'images. Cependant, il a permis également le pilotage d'une bibliothèque de programmes d'astrophysique (spécialisée en structure interne des étoiles), démontrant ainsi que OCAPI permet de piloter des bibliothèques de programmes indépendamment du domaine couvert par la bibliothèque. Dans ce système, un *but* représente une fonctionnalité du domaine concerné et un *opérateur* contient une connaissance spécifique pour résoudre un but donné. Un but contient également la connaissance qui permet de choisir entre les différents opérateurs sachant le résoudre. Cette connaissance est exprimée par des *règles de choix* qui tiennent en compte les données du problème. Un opérateur peut être *complexe* ou *élémentaire*. Dans le premier cas, il représente un traitement du but décomposable en plusieurs étapes. Dans le deuxième, il fait un appel à l'exécution d'un programme.

La demande de résolution d'un problème s'exprime sous forme d'une *requête* laquelle indique le prochain but à atteindre. Une requête exprime des contraintes sur les données d'entrée et de sortie. Ces contraintes seront prises en compte lors du choix de l'opérateur qui résoudra le problème. La figure 1.16 illustre la structure d'un but et d'un opérateur dans OCAPI. Deux autres systèmes ont été développés comme une extension de OCAPI : PÉGASE et PHÉNIX. Ils suivent également l'approche *but/opérateur* [Moi97].

But	**Opérateur**
nom : *symbole*	nom : *symbole*
fonctionnalité : *commentaire*	fonctionnalité : *commentaire*
arguments-entrée : *liste d'arguments*	arguments-entrée : *liste d'arguments*
arguments-sortie : *liste d'arguments*	arguments-sortie : *liste d'arguments*
choix : liste de règles	**décomposition : arbre de requêtes** (pour les opérateurs complexes)
évaluation : *liste de règles*	**syntaxe : objet** (pour les opérateurs élémentaires)

FIG. 1.16 – *Structure d'un* **but** *et d'un* **opérateur** *dans OCAPI :* En gras, les arguments qui décrivent l'association des buts et des opérateurs. C'est une **liste de règles de choix** qui permet de sélectionner l'opérateur le mieux adapté pour atteindre un but. L'argument **décomposition** d'un opérateur décrit, à travers un arbre de requêtes, les étapes qui doivent être réalisées pour que le but soit atteint. Lorsqu'un opérateur est composé d'une seule étape (un seul programme), l'argument **syntaxe** décrit l'appel à ce programme.

LISA [Del93] est un langage dédié à la conception de SBC qui permet de passer d'un modèle conceptuel à un modèle opérationnel. Dans LISA un *but* caractérise un problème à résoudre et chaque but est associé à des *méthodes* considérées comme les mieux adaptées pour le résoudre. Une *méthode* caractérise un “savoir-faire” ; autrement dit, une méthode décrit une stratégie pour atteindre un but. Cette stratégie peut être l'exécution d'un code de calcul (ou un programme), une base de règles ou un enchaînement de sous-buts à atteindre. Le choix entre les méthodes qui sont associées à un but est régi par des *règles de préférence*, également décrites dans le but. La figure 1.17 illustre les fonctions LISA qui permettent de décrire un but et les méthodes qui lui sont associées.

DEFBUT	*<nom-but>*
(TITRE	<*identificateur*>)
(DESCRIPTION	<*chaîne de caractères*>)
(ÉTAT-BUT	<*état-but*>)
(CONTEXTE	<*contexte*>)
(**méthodes**	<**nom-méthodes**>*)
(**préférences**	<**préférences**>)

DEFMETHOD	*<nom>*
(TITRE	<*identificateur*>)
(DESCRIPTION	<*chaîne de caractères*>)
(PARAMÈTRES	<*paramètres*>)
(**stratégie**	<**nom d'une procédure**> |
	<**base de règles**> |
	<**enchaînement de sous-buts**>)

FIG. 1.17 – *Fonctions LISA pour décrire un but (*DEFBUT*) et une méthode (*DEFMETHOD*) :* En gras, les attributs de ces fonctions qui permettent, respectivement, (i) d'associer des **méthodes** à des buts et donner des règles de **préférence** entre les méthodes ; et (ii) de définir une **stratégie** pour réaliser la méthode choisie. La stratégie **enchaînement de sous-buts**, dans DEFMETHOD, décrit la décomposition d'un but en sous-but. Les deux autres (**nom d'une procédure** et **base de règles**) décrivent une action "simple" qui permet d'atteindre le but, en l'occurrence un programme ou une règle d'inférence sur les connaissances du SBC, respectivement.

1.4.3 Conclusion

Nous avons montré quelques exemples de systèmes d'aide à la résolution de problèmes, modélisés par des tâches. Nous avons pu remarquer qu'il existe une différence de vocabulaire entre les divers travaux concernant la modélisation de résolution de problème par des tâches. Cependant, il est possible d'établir une correspondance entre les définitions données et celles[6] que nous avons fixé dans la section 1.1. Le tableau ci-dessous montre cette correspondance.

concept	**notre vocabulaire**	**travaux présentés**	
		systèmes/approches	**terminologie**
représentation d'un problème	tâche	OCAPI, LISA	but
décompositions successives du problème	stratégie	tâche/méthode/sous-tâche	la décomposition d'une méthode
		OCAPI	corps d'un opérateur complexe
		CommonKADS, PowerTasks	task-body
		TASK	corps
résolution d'un problème simple	tâches élémentaires	TRAM	programmes
		OCAPI	opérateurs élémentaires
		KSM	primitives de représentation
		PROTÉGÉ-II	mécanismes
		SCARP	procédures
		CommonKADS	primitive (ou tâche terminale)

TAB. 1.1 – *Tableau de correspondance entre le vocabulaire des divers travaux présentés et celui fixé dans ce travail :* Dans ce tableau ne figurent que les systèmes dont la terminologie diffère de la nôtre, pour chaque concept.

Dans tous les travaux présentés ci-dessus, nous avons retrouvé la technique de décomposition successives d'un problème en sous-problèmes, principale caractéristique propre aux

6. Extraites, évidemment, des travaux présentés.

modèles de tâches. La figure 1.18 illustre l'utilisation des arbres ET/OU pour la représentation de la structure de décomposition de deux tâches, décrites respectivement en utilisant chacune de deux approches présentées dans cette section.

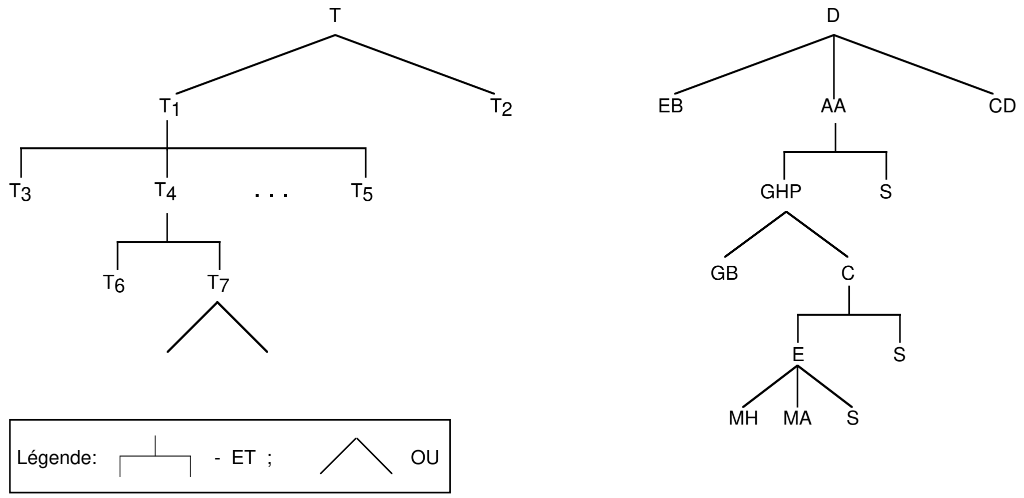


FIG. 1.18 – *Arbre ET/OU :* L'arbre à gauche correspond à l'exemple de SCARP (figure 1.5, page 17). L'arbre à droite correspond à la description de la *tâche générique Diagnsotic* (figure 1.10 page 21) ; les étiquettes des sommets de ce dernier arbre sont des abréviations des noms des tâches et des méthodes de l'exemple. SCARP utilise l'approche d'une entité unique ***tâche***. Les tâches génériques suivent l'approche ***tâche/méthode/sous-tâche***.

Nous n'avons pas donné des détails sur la mise en œuvre des systèmes présentés, parce que notre but était de montrer comment chacun de ce travaux représentait les décompositions successives selon les deux approches identifiées. Le but de ce travail est de comparer les structures de décompositions de deux tâches. Dans la section suivante, nous montrons comment comparer deux arbres de décomposition.

1.5 Comparaison d'arbres de décomposition de tâches

Une stratégie représentée par une tâche décrit les étapes de résolution du problème que la tâche représente, chacune de ces étapes indique à son tour un autre problème également représenté par une tâche. Lors de la résolution effective d'un problème, c'est une séquence de tâches élémentaires qui sera réalisée (l'exécution séquentielle de programmes, en ce qui concerne le contexte de ce travail). Une telle séquence est dénommée *plan d'actions*.

Un utilisateur n'a pas besoin de connaître tous les plans d'actions décrits par une stratégie. Cependant, un même ensemble $\mathcal{P}$ de plans d'actions peut être décrit par différentes stratégies, représentées par différentes tâches (figure 1.19). Lorsque les stratégies représentées par deux tâches, **T** et **T**′ par exemple, représentent le même ensemble de plans d'actions, **T** et **T**′ peuvent êtres exécutées indifféremment pour résoudre un problème donné. Les tâches **T** et **T**′ sont donc dites *équivalentes*.

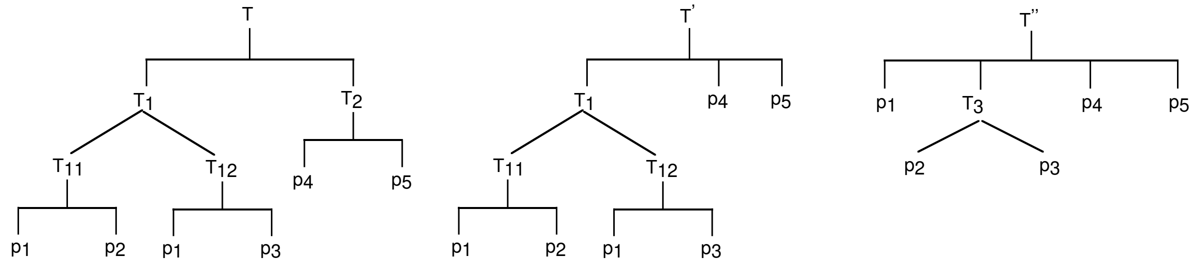


FIG. 1.19 – Ces arbres représentent la structure de décomposition de trois tâches lesquelles représentent des stratégies qui décrivent le même ensemble de plans d'actions composé de deux plans, $\mathcal{P} = \{p_1p_2p_4p_5, p_1p_3p_4p_5\}$.

R. Hamadi [Ham93] considère que "deux tâches sont *structurellement équivalentes* si elles peuvent être représentées par deux arbres identiques ou isomorphes". Cependant, la simple vérification d'un isomorphisme entre arbres de décomposition tâches ne suffit pas car le fait que deux arbres de décomposition ne soient pas isomorphes ne garantit pas qu'ils représentent des tâches non équivalentes. L'exemple simple illustré par la figure 1.19 nous montre trois arbres de décomposition non isomorphes qui pourtant représentent le même ensemble des plans, par conséquent des tâches équivalentes. Il propose donc de rechercher une forme *canonique* d'arbre de décomposition d'une tâche et de comparer deux tâches par la vérification d'un isomorphisme entre formes canoniques d'arbres.

1.5.1 Forme canonique d'un arbre de décomposition

La forme canonique d'un arbre de décomposition est obtenue par la contraction de sommets internes[7] adjacents ayant des étiquettes identiques. Autrement dit, si un sommet interne v de **T** étiqueté par (*tch,str$_i$*) a pour père un sommet v', également étiqueté par (*tch,str$_i$*), alors dans la forme canonique de **T**, v disparaît et ses fils deviennent fils de v' (*cf* figure 1.20). Un arbre sous la forme canonique n'a donc pas deux sommets adjacents avec des étiquettes identiques.

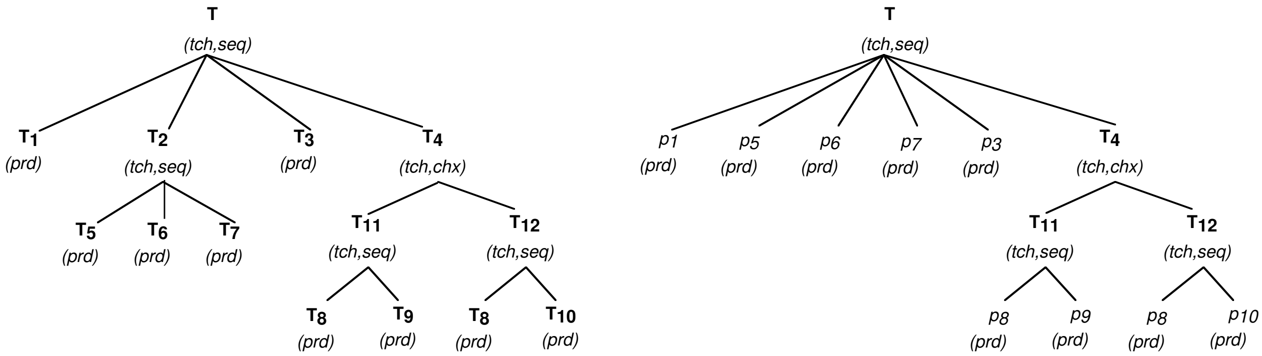


FIG. 1.20 – *À droite l'arbre canonique de l'arbre représenté à gauche.*

7. Pour le vocabulaire concernant aux arbres, en tant que graphes, se reporter à l'Annexe A.

Deux arbres de décomposition de tâches qui possèdent formes canoniques identiques représentent des raisonnements similaires pour résoudre une même classe de problèmes. La différence entre ces raisonnements est le degré de détails donné à chaque niveau de la décomposition d'un problème (figure 1.21) ; mais, à la base ils décrivent les mêmes étapes élémentaires à être réalisées (ou le même ensemble des plans d'actions).

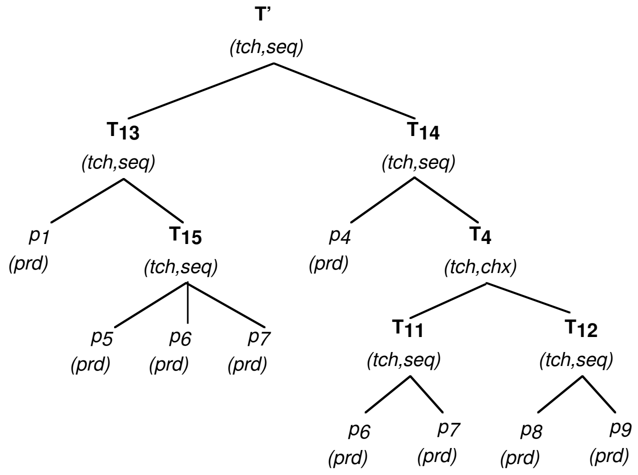


FIG. 1.21 – *Formes canoniques identiques:* Cet arbre possède la même forme canonique de celui de la figure 1.20. Il a un niveau de plus dans l'analyse de la tâche.

La notion d'équivalence de tâches donnée par Hamadi est compatible avec celle adoptée dans ce travail. En effet, deux tâches sont équivalentes [Ham93] si elles possèdent la même forme canonique ; par conséquent elles décrivent le même ensemble $\mathcal{P}$ des plans d'actions : celui décrit par leur forme canonique. Toutefois, il est possible que même si les formes canoniques (comme définies en [Ham93]) de deux arbres de décomposition ne sont pas isomorphes, ils représentent le même ensemble des plans d'actions, représentant donc des tâches *équivalentes*. Par exemple, si l'on utilise l'approche de Hamadi (qui consiste à *contracter des sommets internes adjacents de même type*), pour déterminer la forme canonique des arbres ET/OU de la figure 1.19 on notera que $\mathbf{T}'$ correspond à la forme canonique de $\mathbf{T}$ et que $\mathbf{T}''$ est déjà sous une forme canonique. Or, $\mathbf{T}'$ et $\mathbf{T}''$ sont des formes canoniques non isomorphes qui pourtant représentent le même ensemble des plans d'actions. En conclusion, la vérification d'un isomorphisme de formes canoniques d'arbres de décomposition de tâches ne suffit pas non plus.

Le fait que deux arbres non isomorphes, même sous leurs formes canoniques respectives, représentent des stratégies équivalentes est dû à la sémantique de la décomposition et de la spécialisation des tâches[8] laquelle permet de déterminer l'ensemble des plans d'actions décrit par une stratégie.

8. Voir section 1.3 (**Sémantique de la décomposition et de la spécialisation de tâches**).

1.5.2 Conclusion

Vu la nature hiérarchique des décompositions successives d'une tâche en sous-tâches, la structure de décomposition d'une tâche peut être représentée par un arbre dont la racine représente la tâche à décomposer. Nous avons constaté que la vérification d'un isomorphisme d'arbres de décomposition n'est pas suffisant pour déterminer si les stratégies indiquées par deux tâches décrivent le même ensemble des plans d'actions. En effet, deux arbres, même possédant des formes canoniques non isomorphes, peuvent représenter un même ensemble de plans d'actions. Et cela à cause de la sémantique liée aux décompositions et spécialisations d'une tâche.

En vue d'étendre la possibilité d'identifier que deux tâches sont équivalentes (i.e., possèdent le même ensemble des plan d'actions) pour les cas où les formes canoniques des arbres de décomposition respectifs ne sont pas isomorphes, nous proposons donc une représentation qui permet de, avant tout, identifier l'ensemble des plans d'actions indiqué par une stratégie. Or, une telle représentation doit d'une part, pouvoir indiquer un enchaînement de sous-plans pour composer le plan d'actions représenté par une tâche décomposée en séquence ; et d'autre part, permettre d'indiquer la liste de plans d'actions représentées par une tâche décomposée en choix ou une tâche spécialisable[9]. De plus, sachant qu'un plan d'actions représente une séquence d'actions élémentaires, la représentation recherchée doit se baser sur la représentation de tâches élémentaires.

1.6 Proposition d'un modèle pour représenter une stratégie

Nous proposons un modèle pour la représentation d'une stratégie, représentée par la structure de décompositions d'une tâche en sous-tâches, dénommé *E-tâche-graphes*. Les E-tâche-graphes ont été inspirés des E-graphes (d'où leur dénomination), proposés par Clemente-Salazar [Cle82] comme un modèle de structures de données en traduction automatique. Ci-après nous donnons la définition des E-graphes et nous présentons des aspects de cette structure qui nous ont parus adaptables à notre problème de la *représentation d'une stratégie*.

Nota : La terminologie et les notations de la théorie des graphes utilisées dans ce chapitre sont données dans l'**Annexe A**.

9. Se reporter à la section 1.3 (**Sémantique de la décomposition et de la spécialisation d'une tâche**).

1.6.1 E-graphes

Un E-graphe est un *digraphe*[10] *aux arcs étiquetés* comprenant exactement un sommet initial (dénommé entrée), un sommet final (dénommé sortie) et où seuls les arcs sont étiquetés (figure 1.22).

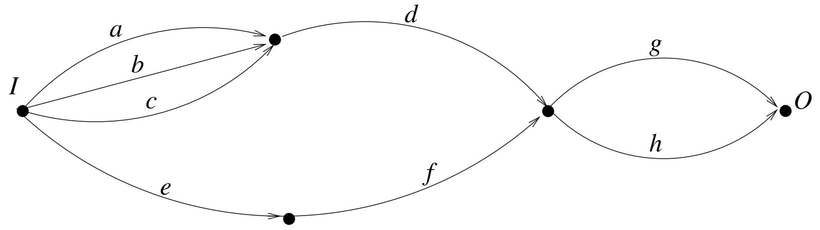


FIG. 1.22 – *Exemple d'un E-graphe.*

Clement-Salazar a identifié deux "formes géométriques" lesquelles reçoivent des noms spéciaux (figure 1.23) : i) *séquentiel*, un E-graphe qui consiste en un seul chemin de l'entrée à la sortie ; ii) et *faisceau*, un E-graphe qui contient n chemins entre l'entrée et la sortie. Les E-graphes séquentiels et les faisceaux constituent une classe d'E-graphes dénommée *E-graphes simples*.

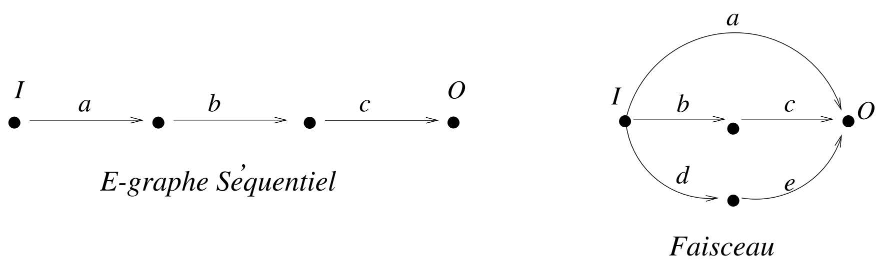


FIG. 1.23 – *Exemple d'un E-graphe séquentiel et d'un faisceau.*

Les E-graphes peuvent être "manipulés", par ce que Clemente-Salazar a dénommé "transformations globales", pour donner origine à de nouveaux E-graphes. Parmi les "transformations globales" définies en [Cle82] nous en soulignons deux : la *concaténation* et l'*alternation*.

Étant donnés deux E-graphes G_1 et G_2,

i) la *concaténation* est la transformation qui consiste à identifier l'entrée de G_1 avec la sortie de G_2 ;

ii) l'*alternation* est la transformation qui consiste à confondre les entrées et les sorties de G_1 et de G_2.

La figure 1.24 illustre l'obtention de l'E-graphe G par l'*alternation* d'un E-graphe G_1 et l'E-graphe résultant de la *concaténation* de deux autres, G_2 et G_3.

10. Un digraphe est un graphe orienté (*directed graph*, en anglais).

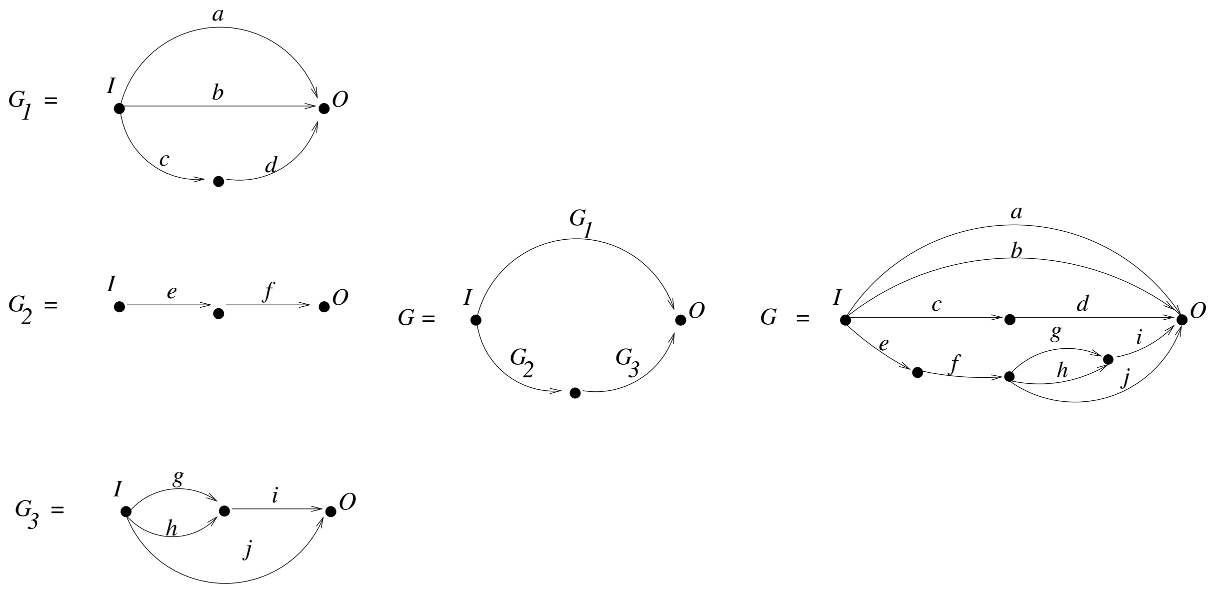


FIG. 1.24 – *Transformations globales sur des E-graphes :* L'*alternation* de G_1 et de l'E-graphe résultant de la *concaténation* de G_2 et G_3 (représentées au milieu de la figure) produit comme résultat l'E-graphe G à droite.

Les E-graphes obtenus par des concaténations et des alternations d'autres E-graphes constituent une classe d'E-graphes dénommée *E-graphes récursifs*. Un E-graphe récursif est composé d'E-graphes simples ou récursifs. Par exemple, dans la figure 1.24, G_1 et G_2 sont des E-graphes simples et l'E-graphe G_3 est récursif.

La concaténation et l'alternation d'E-graphes, ainsi que les E-graphes récursifs, sont tout à fait adaptables à la représentation du processus de construction d'une stratégie définie pour résoudre un problème. En effet, si l'on considère que chaque arc représente une tâche élémentaire,

i) un E-graphe *séquentiel* représente l'exécution en séquence de tâches élémentaires. Autrement dit, un E-tâche-graphe séquentiel représente un *plan d'actions* ;

ii) un *faisceau* représente des options de *choix* entre différents plans ; et

iii) une stratégie indiquée par une tâche complexe peut à son tour être représentée par un E-graphe récursif.

1.6.2 E-tâche-graphes: une nouvelle représentation d'une stratégie

Basé sur les E-graphes, un *E-tâche-graphe* est donc un *digraphe aux arcs étiquetés* où les arcs représentent des tâches élémentaires. L'étiquette d'un arc correspond au nom d'un programme, dans une bibliothèque de programmes, qui permet de résoudre le problème représenté par la tâche élémentaire associée à cet arc. Ce digraphe possède un seul sommet d'entrée (I) et un seul sommet de sortie (O), qui marquent respectivement le début et la fin de l'exécution de la tâche qu'il représente. Les autres sommets représentent les liens d'exécution entre les tâches élémentaires (figure 1.25). Un *plan d'actions* est représenté par un chemin allant de I à O et , par conséquent, l'ensemble $\mathcal{P}$ des tous les plans d'actions décrits par une stratégie est composé de tous les chemins liant I à O.

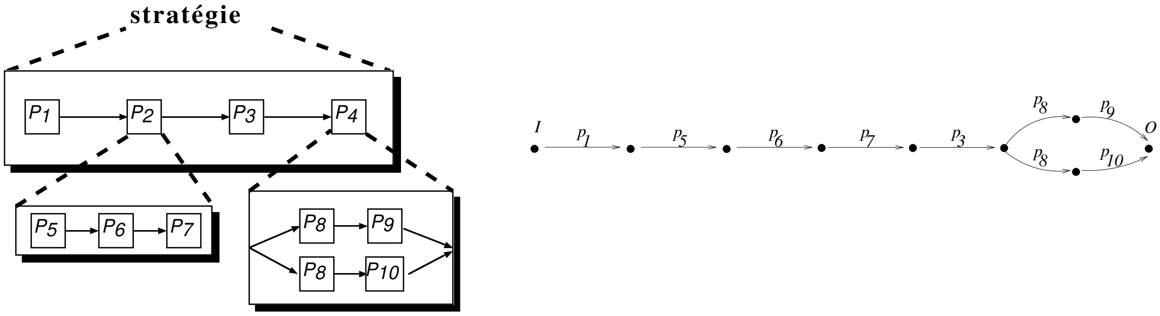


FIG. 1.25 – *Exemple d'un E-tâche-graphe:* L'E-tâche-graphe à droite du dessin représente la stratégie illustrée à gauche. Les arcs représentent les tâches élémentaires. Chaque arc est étiqueté par le nom du programme, existant dans une bibliothèque de programme, lequel permet de réaliser la tâche qu'il représente. Dans cette figure, un programme p_i correspond à la réalisation d'une tâche élémentaire $\mathbf{T}_i$ laquelle représente un problème P_i.

Dans un E-tâche-graphe, un sommet v, dont le degré de sortie, (noté par $d^+(v)$) est *plus grand* que 1, indique l'existence d'un *choix* entre plusieurs stratégies. La fin de la description du choix représenté par v est identifiée par un sommet v' lequel est le *premier* sommet commun à tous les chemins partant de v (voir figure 1.26).

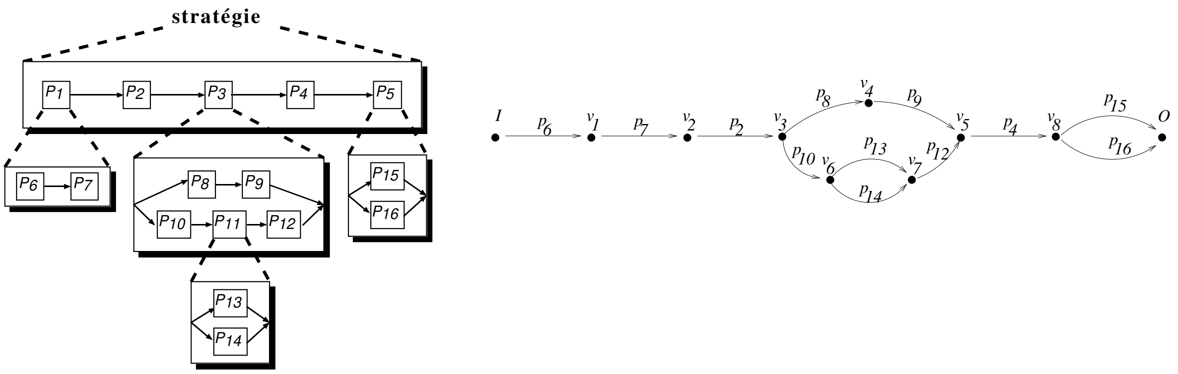


FIG. 1.26 – *Représentation d'un choix dans un E-tâche-graphe:* La troisième étape de la stratégie représentée à gauche du dessin peut être réalisée de deux façons différentes. Cette option de choix est indiquée dans l'E-tâche-graphe par le sommet v_3 dont le degré de sortie est plus grand que 1. La fin de la description de cette option de choix est indiquée par le sommet v_5 lequel est le premier sommet appartenant à tous les chemins partant de v_3.

Nous avons décidé de représenter indifféremment la *spécialisation* et le *choix*. En effet, lors de la résolution d'un problème, la spécialisation indique également l'existence d'un *choix*, lequel est réalisé automatiquement, du plan le plus adapté, le plus *spécifique* pour résoudre le problème.

1.6.3 Construction d'un E-tâche-graphe à partir de la définition d'une stratégie

Le but des E-tâche-graphes est de représenter une stratégie, pour résoudre un problème donné P, à travers les étapes les plus simples qui la composent. Ces étapes sont représentées par des tâches élémentaires lesquelles sont à leur tour réalisées, dans le contexte de ce travail, par l'exécution d'un programme existant dans une bibliothèque de programmes.

Au départ, un E-tâche-graphe n'est composé que d'un seul arc (étiqueté par le nom de la tâche **T** qui représente le problème P) lequel lie I à O, les deux sommets représentant le début et la fin de la réalisation de **T**. Puis, cet arc est remplacé par un ou plusieurs chemins qui représentent les différentes options de résolution de P, en fonction de la stratégie représentée par **T**, comme décrit ci-dessous :

- si **T** est décomposée en *séquence* alors l'arc liant I à O est remplacé par un chemin, allant de I à O, contenant autant d'arcs que d'étapes dans lesquelles **T** se décompose ;
- si **T** est une tâche décomposée en *choix* ou une tâche *spécialisable* alors l'arc liant I à O est remplacé par autant d'arcs que d'options de choix représentées par la décomposition ou la spécialisation ;
- si **T** est une tâche *élémentaire* l'étiquette de l'arc est remplacé par le nom du programme qui la réalise ;

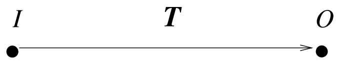
Ce processus se répète pour tous les arcs qui représentent des tâches complexes. Autrement dit, le processus se termine lorsque tous les arcs sont étiquetés par des nom de programmes qui permettent de réaliser des tâches élémentaires.

Exemple 2 *Construction d'un E-tâche-graphe à partir de la définition d'une stratégie*

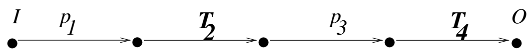
Notation : Une tâche $\mathbf{T}_i$ représente un problème P_i ; une tâche élémentaire $\mathbf{T}_i$ est réalisée par le programme p_i.

Les étapes de construction de l'E-tâche-graphe qui représente la stratégie représentée par la figure 1.25 sont les suivantes :

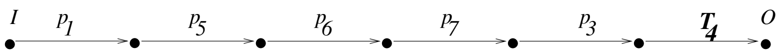
1. Dans un premier temps, un digraphe composé d'un seul arc étiqueté par **T**, liant les sommets I et O est construit. Ce graphe représente une stratégie "globale".



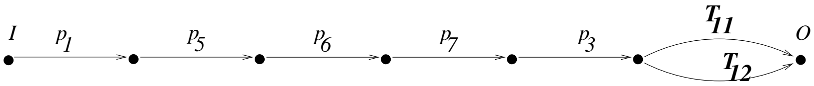
2. Ensuite, cet arc est remplacé par un chemin composé de 4 arcs qui indiquent les 4 tâches lesquelles représentent les étapes qui composent la stratégie représentée par **T**. Chaque arc de ce chemin est étiqueté soit par le nom d'un programme (p_i), s'il indique une étape élémentaire ; soit par le nom d'une tâche complexe ($\mathbf{T}_i$).



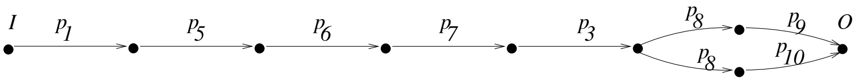
3. La deuxième étape ($\mathbf{T}_2$) correspond à une tâche qui est elle-même décomposée en trois étapes élémentaires. L'arc étiqueté par $\mathbf{T}_2$ est donc remplacé par un chemin composé de trois arcs étiquetés par le nom des programmes qui réalisent les respectives étapes ($c = p_5p_6p_7$).



4. La dernière étape ($\mathbf{T}_4$) correspond à une tâche pour laquelle il existe deux stratégies possibles (représentées par les tâches $\mathbf{T}_{11}$ et $\mathbf{T}_{12}$). L'arc étiqueté par $\mathbf{T}_4$ est donc remplacé par deux autres, étiquetés par $\mathbf{T}_{11}$ et $\mathbf{T}_{12}$, représentant donc cette possibilité de choix.



5. Finalement, chacun des arcs $\mathbf{T}_3$ et $\mathbf{T}_4$ est remplacé par un chemin qui représente les respectives étapes élémentaires qui les composent.



L'algorithme, récursif, ***E-tâche-graphe***, décrit dans l'Annexe D.2, permet de construire un E-tâche-graphe à partir de la stratégie de **T**. La récursivité de l'algorithme assure la représentation complète des décompositions successives de **T** puisqu'à chaque appel récursif de l'algorithme, une sous-étape de la stratégie est analysée. Il tient en compte également si **T** est utilisée en tant que sous-tâche d'une autre tâche décomposable (et/ou spécialisable). L'algorithme s'arrête lorsque les arcs sont tous étiquetés par le nom d'un programme dans la bibliothèque.

1.7 Conclusion

Dans ce travail nous nous intéressons à la comparaison de stratégies représentées par des tâches. Nous signalons que ce n'est pas l'aspect *dynamique* d'une stratégie que nous intéresse, c'est-à-dire la façon dont les choix sont réalisés dynamiquement lors de la construction d'un plan d'actions pour résoudre un problème précis. Nous ne nous intéressons qu'à l'aspect *statique* d'une stratégie, c'est-à-dire l'ensemble des plans d'actions décrit par une stratégie définie pour résoudre un problème (ou, plus précisément comment des modules existants dans une bibliothèque de programmes peuvent être liés pour résoudre un type de problème).

Nous avons constaté que la vérification d'un isomorphisme d'arbres de décomposition, qui représentent des stratégies décrites par des tâches, ne permet pas **dans tous les cas** de déterminer si deux stratégies indiquent le même ensemble des plans d'actions. En effet, nous avons donné un exemple simple de deux arbres non isomorphes qui pourtant représentaient le même ensemble des plan d'actions. Sachant qu'un plan d'actions n'est composé que d'actions simples, lesquelles sont représentées par des tâches élémentaires, nous proposons une nouvelle représentation d'une stratégie, dénommé *E-tâche-graphes*, qui ne représente que des tâches élémentaires. Ce modèle a été introduit dans la section 1.6.2 où nous avons également présenté les étapes de construction d'un E-tâche-graphe à partir de la définition d'une stratégie représentée par les décompositions successives d'une tâche en sous-tâches.

Puisque les E-tâche-graphes ne représentent que des tâches élémentaires, chaque arbre de décomposition T correspond à un E-tâche-graphe G dans lequel ne sont représentées que les feuilles de l'arbre. Rappelons que la racine d'un arbre de décomposition (ou d'un sous-arbre) correspond à une tâche complexe tandis que les feuilles correspondent à des tâches élémentaires. En réalité G correspond à la forme canonique de T. Ainsi, des arbres de décomposition qui possèdent la même forme canonique correspondent au même E-tâche-graphe; la recherche d'une telle forme canonique n'est donc plus nécessaire. De plus, la spécialisation et la décomposition d'une tâche peuvent être représentées dans un même E-tâche-graphe; autrement dit, tous les plans d'actions possibles pour résoudre un problème, qu'ils soient décrits par une spécialisation ou par une décomposition en choix, figurent dans le même digraphe.

L'étude plus détaillée des *E-tâche-graphes* fait l'objet du chapitre suivant.

Chapitre 2

E-tâche-graphes

Le but de ce chapitre est de faire une étude détaillée des *E-tâche-graphes*, le modèle que nous proposons pour la représentation d'une stratégie de résolution d'une classe de problèmes exprimée par une tâche.

Ce modèle s'occupe des liens d'exécution entre les étapes élémentaires qui composent une stratégie de résolution [1]. Autrement dit, les décompositions successives d'un problème en sous-problèmes ne sont pas exprimées dans un E-tâche-graphe. Les éléments de base de construction d'un E-tâche-graphe plus complexe sont donc des *E-tâche-graphes élémentaires* qui représentent les étapes élémentaires.

Les E-tâche-graphes plus complexes sont construits de manière analogue aux stratégies. Autrement dit, les stratégies soit sont composées d'une exécution *séquentielle* d'étapes élémentaires (ou des stratégies plus complexes déjà connues), soit indiquent une possibilité de *choix* entre diverses stratégies déjà établies ; les E-tâche-graphes, eux, sont composés d'une *concaténation* et/ou une *bifurcation* d'E-tâche-graphes représentant l'exécution d'un programme (ou d'une stratégie plus complexe). *Concaténation* et *bifurcation* sont donc des opérations de construction des E-tâche-graphes. Ces deux opérations donnent origine à deux configurations possibles pour les E-tâche-graphes, les *multiplicatifs* et les *additifs*, qui représentent respectivement des stratégies de décomposition d'une tâche en *séquence* et en *choix*.

Les E-tâche-graphes sont munis d'une sémantique qui, naturellement, reproduit celle de la décomposition et de la spécialisation d'une tâche. La sémantique des E-tâche-graphes est basée sur leurs opérations de construction et permet d'établir les propriétés de ces opérations. Ces propriétés à leur tour rendent possible à la fois (i) de déterminer l'E-tâche-graphe G' qui représente l'ensemble de plans d'actions décrits par la stratégie représentée par un E-tâche-graphe G (G' est dénommé *forme étendue de* G) ; et (ii) de définir un système de réécriture qui permet de déterminer une *forme normale* des différents E-tâche-graphes qui représentent des diverses stratégies capables de résoudre une même classe de problèmes.

1. Dans le cadre du pilotage d'une bibliothèque de programmes, un E-tâche-graphe représente les liens d'exécution entre des programmes existants dans la bibliothèque.

La comparaison de deux stratégies, dans le but de vérifier si elles représentent le même ensemble de plans d'actions, pourrait se faire par la comparaison des deux E-tâche-graphes qui représentent ces stratégies, sous leur formes étendues respectives. Cependant, l'énumération exhaustive de l'ensemble des plans d'actions décrit par une stratégie peut devenir prohibitive à cause d'un probable nombre exponentiel de plans. Par conséquent, une forme normale d'un E-tâche-graphe se fait nécessaire.

Ce chapitre est organisé de la façon suivante : nous présentons tout d'abord la définition formelle des E-tâche-graphes ainsi que leurs opérations de construction (section 2.1). Ensuite, nous étudions les configurations qu'ils peuvent avoir (section 2.2). La section 2.3 est réservée à l'étude de sous-E-tâche-graphes spéciaux. Puis, nous présentons la sémantique des E-tâche-graphes (section 2.4) et nous en déduisons les propriétés de leurs opérations de construction (section 2.5). Les deux dernières sections, 2.6 et 2.7, sont dédiées respectivement à l'étude de la forme étendue et de la forme normale d'un E-tâche-graphe.

Nota : La terminologie et les notations de la théorie des graphes utilisées dans ce chapitre sont données dans l'**Annexe A**.

2.1 Définitions et opérations de construction

Cette section a pour but de donner les définitions de base des E-tâche-graphes.

Définitions de base

Définition 1 : Un *E-tâche-graphe* est un digraphe aux arcs étiquetés $G = (V, X, I, O, \lambda, \mathcal{L})$, ayant une seule source I et un seul puits O, où $\mathcal{L}$ est l'ensemble des noms des programmes existants dans une bibliothèque de programmes scientifiques ($\mathcal{L} = \{p_1, p_2, \ldots, p_{|\mathcal{L}|}\}$) et λ est la fonction sur $\mathcal{L}$ d'étiquetage des arcs. Les sommets I et O *marquent* respectivement le *début* et la *fin d'exécution* de la tâche représentée par G.

Définition 2 : Un *E-tâche-graphe élémentaire*, noté par E, est un E-tâche-graphe composé d'un seul arc. Un E-tâche-graphe élémentaire E représente une tâche élémentaire laquelle est réalisée par le programme, existant dans une bibliothèque $\mathcal{L}$, dont le nom correspond à l'étiquette de l'arc de E.

Définition 3 *Trajectoire* : Soit $G = (V, X, I, O, \lambda, \mathcal{L})$ un E-tâche-graphe. Une *trajectoire* τ de G est un chemin allant de I à O. Un sommet d'une trajectoire est dénommé *étape* (I étant l'étape *initiale* et O, l'étape *finale*). Une trajectoire τ d'un E-tâche-graphe G représente un plan d'actions décrit par la stratégie indiquée par la tâche représentée par G.

Définition 4 *Sous-trajectoire* : Soient $G = (V, X, I, O, \lambda, \mathcal{L})$ un E-tâche-graphe, τ une trajectoire de G et v une étape de τ $(v \neq I$ et $v \neq O)$. Le sous-chemin τ' de τ allant de I à v est dénommé *sous-trajectoire* de τ. Le sommet v (l'étape finale de τ') est dénommée *portée* de τ'.

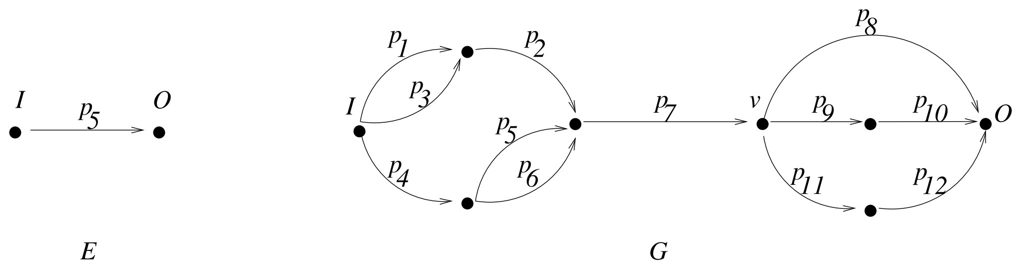


FIG. 2.1 – *Exemples d'E-tâche-graphe:* L'E-tâche-graphe E à gauche est un *E-tâche-graphe élémentaire* qui représente le programme p_5 de $\mathcal{L}$. $\tau = p_1p_2p_7p_8$ est un exemple d'une *trajectoire* de l'E-tâche-graphe G à droite. $\tau' = p_1p_2p_7$ est une *sous-trajectoire* de τ de *portée* v.

E-tâche-graphe linéaire et E-tâche-graphe déterministe

Deux groupes, non distincts, d'E-tâche-graphe ont été identifiés en fonction des étiquettes de arcs : *les linéaires* et les *déterministes*.

Définition 5 *E-tâche-graphe linéaire* : G est dit *linéaire* si une étiquette ne figure qu'une seule fois dans G. Autrement dit,

$$\forall x_i, x_j \in X,\ x_i \neq x_j,\ \lambda(x_i) \neq \lambda(x_j)$$

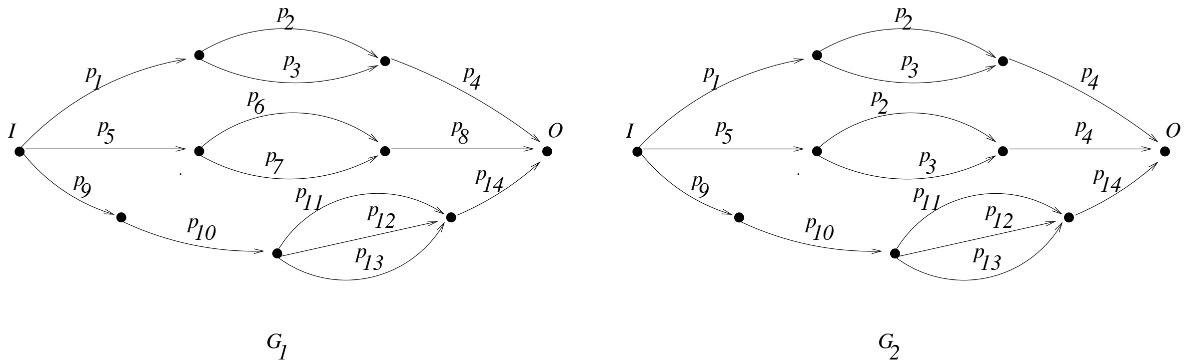


FIG. 2.2 – *E-tâche-graphe linéaire :* L'E-tâche-graphe G_1 à gauche est *linéaire* puisque tous les arcs possèdent des étiquettes différentes. Par contre, l'E-tâche-graphe G_2 à droite est *non linéaire* car il existe des étiquettes qui apparaissent deux fois, comme : p_2, p_3 et p_4.

Définition 6 *E-tâche-graphe déterministe* : G est dit *déterministe* si tous les arcs sortant d'un même sommet ont des étiquettes différentes. Autrement dit,

$$\forall v \in V, \nexists x_i, x_j \in X \text{ tels que } x_i = (\overrightarrow{v,u}),\ x_j = (\overrightarrow{v,w}) \text{ et } \lambda(x_i) = \lambda(x_j)$$

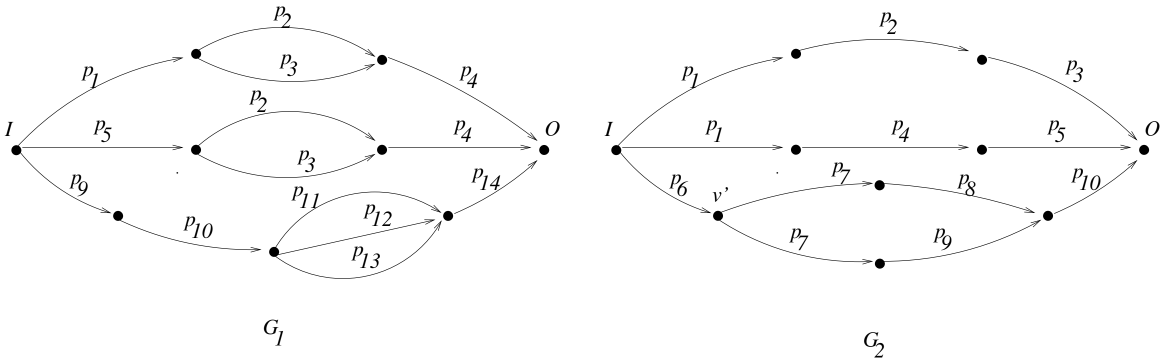


FIG. 2.3 – *E-tâche-graphe déterministe :* l'E-tâche-graphe G_1 à gauche est *déterministe* car il n'existe pas de sommet v dans G_1 tel que les arcs partant de v possèdent des étiquettes égales. Par contre, l'E-tâche-graphe G_2 à droite est *non déterministe* puisque les deux arcs partant de v', par exemple, sont étiquetés par la même étiquette p_7.

Propriété 1 : Tout E-tâche-graphe linéaire est déterministe

En effet, dans un E-tâche-graphe G linéaire, par définition, toutes les étiquettes sont différentes. Par conséquent, il n'existe pas d'arc sortant d'un même sommet avec des étiquettes égales.

Opérations de construction

Tout E-tâche-graphe non élémentaire peut être obtenu à l'aide des **opérations binaires de construction** d'E-tâche-graphes, *concaténation* et *bifurcation*, définies ci-après (voir figure 2.4) :

Soit $\mathcal{L}$ une bibliothèque de programmes. Soient $G_i = (V_i, X_i, I_i, O_i, \lambda_i, \mathcal{L})$ et $G_j = (V_j, X_j, I_j, O_j, \lambda_j, \mathcal{L})$ deux E-tâche-graphes sur $\mathcal{L}$.

Définition 7 *Concaténation* : L'E-tâche-graphe G, résultat de l'opération de *concaténation* de G_i et G_j (noté par la suite $G = G_i\ .\ G_j$), est obtenu en unifiant les sommets O_i et I_j (le *puits* de G_i et la *source* de G_j). G représente l'exécution en *séquence* des tâches représentées par G_i et G_j.

Définition 8 *Bifurcation*[2] : L'E-tâche-graphe G, résultat de l'opération de *bifurcation* de G_i et G_j (noté par la suite $G = G_i \; + \; G_j$), est obtenu par la fusion des *sources* de G_i et G_j (I_i et I_j) aussi bien que leurs *puits* (O_i et O_j). G représente l'existence d'un *choix* entre les tâches associées à G_i et à G_j.

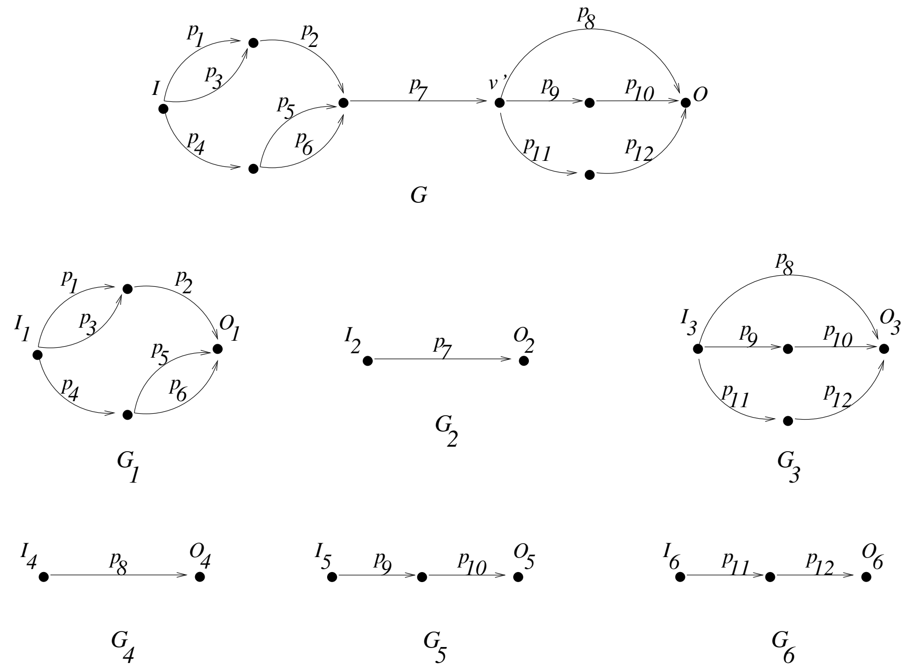


FIG. 2.4 – *Opérations de construction des E-tâche-graphes:* L'E-tâche-graphe G en haut de la figure peut être vu comme une *concaténation* des E-tâche-graphes G_1, G_2 et G_3, ci-dessus. Ce dernier peut être vu comme une *bifurcation* de G_4, G_5 et G_6.

Nota 1 : Par la suite, la *concaténation* de n E-tâche-graphes, $G_1 \; . \; G_2 \; . \; G_3 \; . \; \ldots \; . \; G_n$, sera indiquée par $\prod_{j=1}^{n} G_j$; la *bifurcation* de m E-tâche-graphes, $G_1 + G_2 + G_3 + \ldots + G_m$, sera indiquée par $\sum_{i=1}^{m} G_i$.

Nota 2 : Un E-tâche-graphe peut être vu comme un *graphe orienté série-parallèle* (abrégé par GSP) ; l'opération de *concaténation* d'E-tâche-graphes pouvant être associée à la *composition en série* de GSP et l'opération de *bifurcation* pouvant être associée à la *composition en parallèle*. Une discussion sur l'analogie entre les E-tâche-graphes et les GSP est faite à l'Annexe B.

Définition 9 *Composant* : Si G est un E-tâche-graphe obtenu par la concaténation (ou la bifurcation) de deux E-tâche-graphes G_i et G_j, ces deux derniers sont dits *composants* de G. Par exemple, dans la figure 2.4, G_4, G_5 et G_6 sont des composants de G_3. Les composants d'un E-tâche-graphe G correspondent aux sous-tâches qui composent la tâche représentée par G.

2. Cette opération correspond à l'*alternation* sur les E-graphes. Le terme *bifurcation* nous a apparu plus "parlant" pour indiquer " options de choix entre différents chemins d'accès du sommet initial I au sommet final O".

En utilisant les opérations de construction, des nouveaux E-tâche-graphes peuvent être crées. Ces E-tâche-graphes représentent des nouvelles stratégies lesquelles soit correspondent à l'exécution d'une séquence de programmes (et/ou la réalisation de tâches complexes), soit indiquent des options de choix entre la réalisation de diverses tâches. Ci-après, nous présentons les deux configurations dans lesquelles un E-tâche-graphe peut être classé, selon le type de décomposition qu'il représente (en *séquence* ou en *choix*).

2.2 Configurations

Dans cette section, nous présentons les deux configurations possibles pour un E-tâche-graphe : *additive* et *multiplicative*. Ces configurations peuvent encore être classées en *fondamentales* ou *complexes* en fonction de leurs composants (figure 2.8).

Définition 10 *E-tâche-graphe multiplicatif*: Un E-tâche-graphe multiplicatif M est composé par la *concaténation* de n E-tâche-graphes. Il représente une tâche *décomposée en séquence* de n autre tâches, simples ou complexes.

- **notation :** $M = \prod_{j=1}^{n} G_j$
- Chaque G_j est dénommé *composant multiplicatif* de M
- Si chaque G_j est *élémentaire* alors M est dénommé *séquence* (**notation :** $S = \prod_{j=1}^{n} E_j$).
- Un E-tâche-graphe élémentaire peut être vu comme une séquence d'une seule tâche.

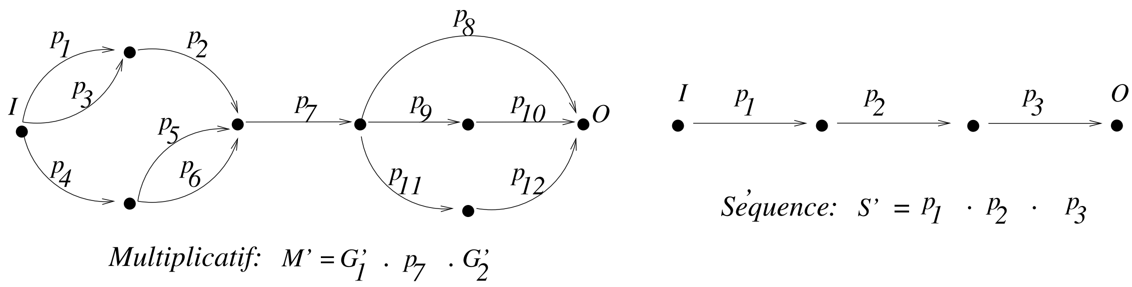


FIG. 2.5 – *E-tâche-graphes multiplicatifs :* Un E-tâche-graphe *multiplicatif* (M) représente une tâche décomposée en *séquence*. Les sous-tâches qui la composent peuvent être élémentaires (e.g. p_7 dans M') ou complexes (e.g. G'_1 et G'_2 dans M'). Un *séquence* (S) est un E-tâche-graphe multiplicatif représentant un tâche dont toutes les sous-tâches sont des tâches *élémentaires*.

Définition 11 *E-tâche-graphe additif*: Un E-tâche-graphe additif A est composé par la *bifurcation* de m E-tâche-graphes multiplicatifs. Il représente une tâche *décomposée en choix*[3] entre m autre tâches, complexes ou non.

- **notation :** $A = \sum_{i=1}^{m} M_i$
- Chaque M_i est dénommé *composant additif* de A
- Si chaque M_i est une *séquence* alors A est dénommé *faisceau* (**notation :** $F = \sum_{i=1}^{n} S_i$).

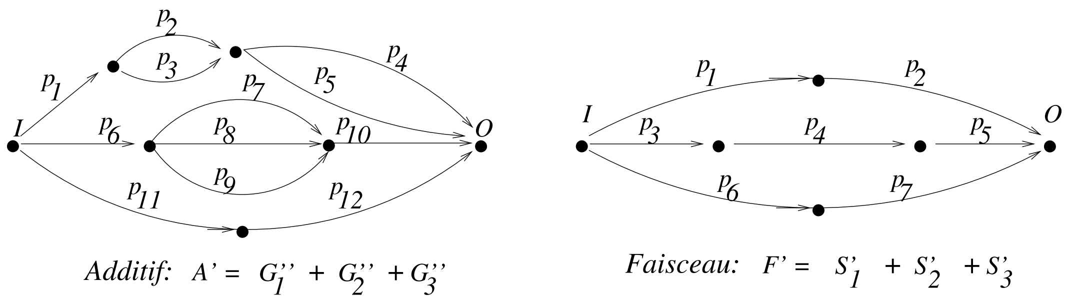


FIG. 2.6 – *E-tâche-graphes additifs :* Un E-tâche-graphe *additif* (A) représente un *choix* entre différentes tâches. Par exemple, A' représente un choix entre trois tâches (G''_1, G''_2 et G''_3). Un *faisceau* (F) représente un choix entre différentes *séquences*.

Le tableau ci-dessous résume ces configurations et donne leur interprétation par rapport aux tâches.

Configurations des E-tâche-graphes	**Notation**	**Interprétation des tâches**
Multiplicatif	M	tâche décomposée en une **séquence** de tâches complexes ou élémentaires
Additif	A	tâche **spécialisable** ou tâche décomposée en un **choix** entre des tâches complexes ou élémentaires
Séquence	S	tâche décomposée en une **séquence** de tâches **élémentaires** ; autrement dit, une tâche qui ne décrit qu'un seul plan d'actions
Faisceau	F	ensemble de plans d'actions

TAB. 2.1 – *Relation entre les configurations des E-tâche-graphes non élémentaires et les tâches complexes représentées par chacune de ces configurations.*

D'après les définitions, un E-tâche-graphe élémentaire E est une séquence (d'une seule tâche), et une séquence S est un E-tâche-graphe multiplicatif (concaténation d'E-tâche-graphes élémentaires). De même, un faisceau F est un E-tâche-graphe additif (bifurcation de séquences). Si, par ailleurs, on considère qu'un E-tâche-graphe multiplicatif M indique un choix n'ayant

3. Puisque la *spécialisation* et la *décomposition en choix* sont représentées indifféremment dans un E-tâche-graphe, une *spécialisation* est également représentée par un *E-tâche-graphe additif*.

qu'une seule possibilité, M peut être vu comme un E-tâche-graphe additif (tout E-tâche-graphe peut donc être vu comme un E-tâche-graphe additif). Ceci nous amène à une relation d'inclusion entre les configurations (figure 2.7) et nous permet d'établir une expression unique pour les E-tâche-graphes non élémentaires, à savoir :

$$G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij} \qquad \text{où } m > 1 \textit{ ou } \exists\, i \mid n_i > 1 \textit{ ou } G_{ij} \text{ est élémentaire} \tag{2.1}$$

Cette expression indique qu'une stratégie est composée d'un choix entre une ou plusieurs séquences de (une ou plusieurs) tâches, élémentaires ou non.

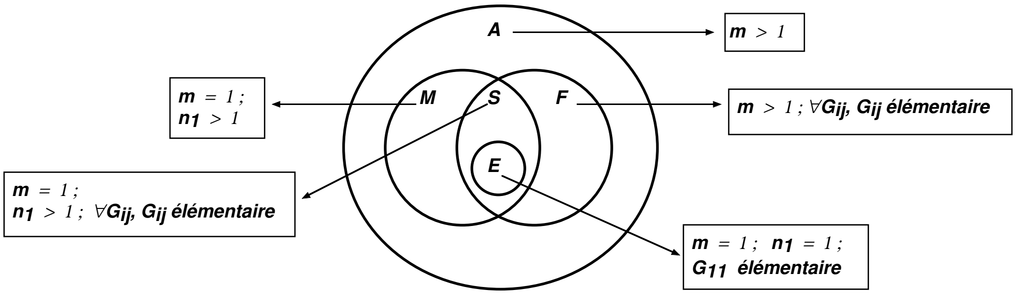


FIG. 2.7 – *Relation d'inclusion existante entre les configurations des E-tâche-graphes.* D'après les définitions, $E \subset S \subset M$ et $F \subset A$. Si l'on considère qu'un E-tâche-graphe *multiplicatif* indique un choix n'ayant qu'*une seule possibilité*, une séquence S (qui est un E-tâche-graphe multiplicatif), indique un choix n'ayant qu'*une seule séquence*. Sachant qu'un faisceau représente un *choix entre séquences*, S est également un faisceau. Ce qui explique $S = M \cap F$. Les expressions associées à chaque sous-ensemble indiquent les conditions de l'expression 2.1 qui sont satisfaites pour chaque configuration.

Le tableau 2.2 explique comment chacune des configurations est identifiée par l'expression ci-dessus.

Configuration des G_{ij}	m	n_i	Expression de G ($G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij}$)	Configuration résultante
$\forall G_{ij},\ G_{ij}$ est élémentaire	$= 1$	≥ 1	$G = \prod_{j=1}^{n} E_j$	séquence (S)
$\forall G_{ij},\ G_{ij}$ est élémentaire	≥ 1	≥ 1	$G = \sum_{i=1}^{m}(\prod_{j=1}^{n_i} E_{ij}) = \sum_{i=1}^{m} S_i$	faisceau (F)
quelconque	$= 1$	> 1	$G = \prod_{j=1}^{n} G_j$	multiplicatif (M)
quelconque	≥ 1	≥ 1	$G = \sum_{i=1}^{m}(\prod_{j=1}^{n_i} G_{ij}) = \sum_{i=1}^{m} M_i$	additif (A)

TAB. 2.2 – *Expression des E-tâche-graphes et l'identification des diverses configurations.*

Configurations fondamentales et configurations complexes

Définition 12 *Configuration fondamentale* : Un E-tâche-graphe $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij}$, où $\forall\, i, \forall\, j$, G_{ij} est élémentaire, est dit sous une *configuration fondamentale*. Dans ce groupe se trouvent donc les *séquences* et les *faisceaux*. Par extension, un E-tâche-graphe élémentaire, est dit sous une configuration fondamentale. Puisque $F \supset S \supset E$, un E-tâche-graphe G sous une configuration fondamentale peut être évoqué, sans ambiguïté, comme un faisceau.

Définition 13 *Configuration complexe* : Un E-tâche-graphe $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij}$, où $\exists\ G_{ij}$ tel que G_{ij} n'est pas élémentaire, est dit sous une *configuration complexe*.

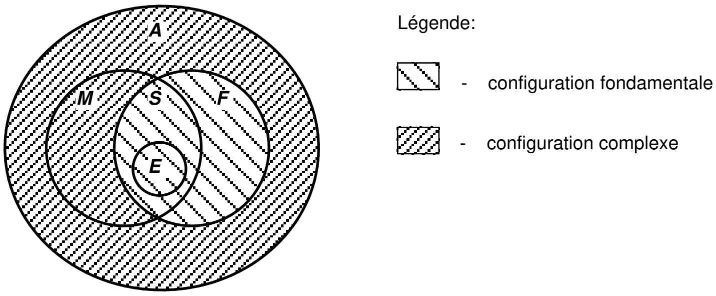


FIG. 2.8 – *Configurations fondamentales et configuration complexes :* Un E-tâche-graphe $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij}$ est sous une configuration fondamentale si $\forall\ i, \forall\ j,\ \ G_{ij}$ est élémentaire. D'après le tableau 2.2, un E-tâche-graphe G qui satisfasse telle condition est soit une séquence, soit un faisceau. Par contre, un E-tâche-graphe $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij}$ dans lequel $\exists\ G_{ij} \mid G_{ij}$ n'est pas élémentaire, G est dit sous une configuration complexe.

2.3 Sous-E-tâche-graphes

Un sous-graphe d'un E-tâche-graphe G, dénommé *sous-E-tâche-graphe*, symbolise une partie de la stratégie représentée par G. Ci-après, nous examinons deux types spéciaux de sous-E-tâche-graphe.

Définition 14 *Sous-E-tâche-graphe additif*: Soit $G = (V, X, I, O, \lambda, \mathcal{L})$ un E-tâche-graphe. Soient $v \in V$ tel que $d^{+}(v) > 1$ et $v' \in V$ le premier sommet commun à tous les chemins partant de v. Le sous-E-tâche-graphe $G' = (V', X', I', O', \lambda, \mathcal{L})$ de G tel que $I' = v$, $O' = v'$ est dit *sous-E-tâche-graphe additif* de G.

Un sous-E-tâche-graphe additif représente une étape de résolution d'un problème pour laquelle il existe un choix entre diverses stratégies de résolution. La figure 2.9 donne des exemples de sous-E-tâche-graphes additifs

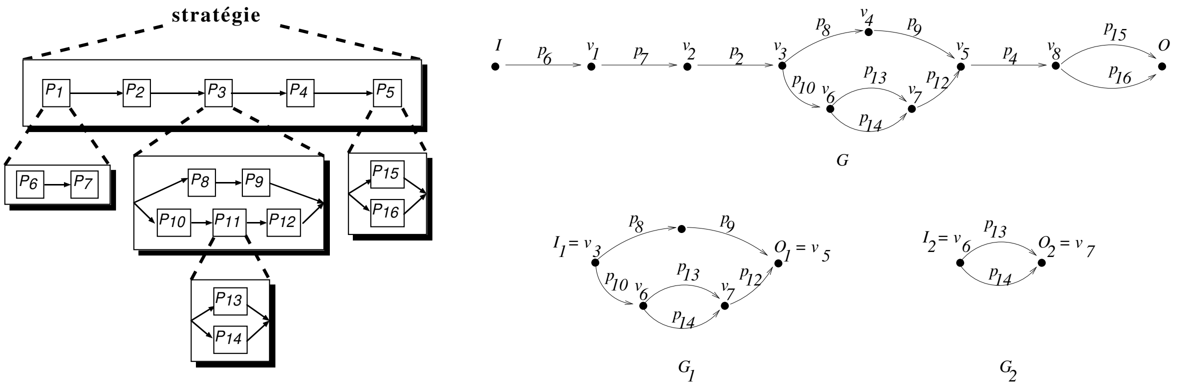


FIG. 2.9 – *Sous-E-tâche-graphe additif :* L'E-tâche-graphe G_1 est le sous-E-tâche-graphe additif de G qui correspond à la troisième étape de la stratégie représentée à gauche. Pour réaliser cette étape il existe un choix entre deux autres stratégies. L'une de ces sous-stratégies contient une étape pour laquelle il existe également un choix entre deux autres stratégies. Le E-tâche-graphe G_2 est le sous-E-tâche-graphe additif de G_1 qui représente ce choix.

L'algorithme ***Sous-E-tâche-graphe-additif***, décrit dans l'annexe D.1, permet de déterminer le sous-E-tâche-graphe additif d'un E-tâche-graphe G, ayant comme sommet initial un sommet donné $v \in V$ tel que $d^+(v) > 1$.

Définition 15 *Composant fondamental* : Soit G un E-tâche-graphe. Tout sous-graphe G' de G tel que G' est sous une configuration fondamentale, est dénommé *composant fondamental* de G.

Un E-tâche-graphe G sous une configuration complexe peut être analysé récursivement jusqu'à arriver à une description qui n'est formée que de *composants fondamentaux de G*. La figure 2.10 illustre une telle analyse justifiée par une grammaire de description des E-tâche-graphes, définie ci-après.

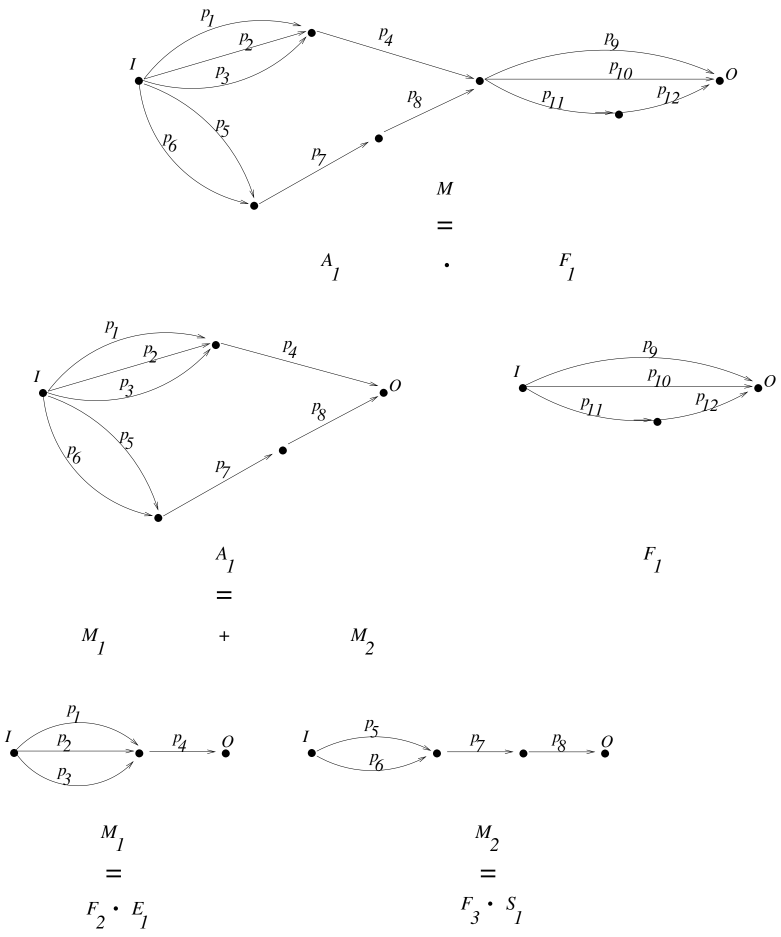


FIG. 2.10 – *Analyse récursive d'un E-tâche-graphe sous une configuration complexe :* G est, en réalité, un E-tâche-graphe *multiplicatif* M composé par la *concaténation* d'un E-tâche-graphe *additif* A_1 et un *faisceau* F_1 $(M = A_1 \ . \ F_1)$. Le composant A_1, sous une configuration complexe, est à son tour composé par la *bifurcation* de deux E-tâche-graphes *multiplicatifs* M_1 et M_2 $(A_1 = M_1 + M_2)$. Et ainsi de suite. Les composants fondamentaux de G sont donc : l'E-tâche-graphe élémentaire $E_1 = p_4$; la séquence $S_1 = p_7 \ . \ p_8$ et les faisceaux $F_1 = (p_9 + p_{10} + (p_{11} \ . \ p_{12})), F_2 = (p_1 + p_2 + p_3)$ et $F_3 = (p_5 + p_6)$. L'E-tâche-graphe G peut donc être représenté à l'aide de ses *composants fondamentaux* $(G = ((F_2 \ . \ E_1) + (F_3 \ . \ S_1)) \ . \ F_1)$, ou à l'aide de ses E-tâche-graphes élémentaires $(G = (((p_1 + p_2 + p_3) \ . \ p_4) + ((p_5 + p_6) \ . \ p_7 \ . \ p_8)) \ . \ (p_9 + p_{10} + (p_{11} \ . \ p_{12})))$.

Grammaire

L'expression d'un E-tâche-graphe G, $G = \sum_{i=1}^{m} \prod_{i=1}^{m_i} G_{ij} = \sum_{i=1}^{m} M_i$, indique qu'un E-tâche-graphe soit est un E-tâche-graphe multiplicatif ($m = 1$), soit correspond à la bifurcation de m $(m > 1)$ E-tâche-graphes multiplicatifs. Dans le cas particulier où G correspond à la bifurcation de séquences, G est un faisceau.

Un E-tâche-graphe multiplicatif M est représenté par l'expression $M = \prod_{i=1}^{m} G_j,\ G_j$ quelconque. Dans le cas particulier où tous les G_j sont des E-tâche-graphes élémentaires, G représente une séquence.

Un faisceau F correspond à une bifurcation de séquences. L'expression d'un faisceau est donc $F = \sum_{i=1}^{m} S_i$. Lorsque $m = 1$, il représente une seule séquence.

Finalement, une séquence correspond à la concaténation d'E-tâche-graphes élémentaires. Ces derniers représentent les programmes existants dans une bibliothèque $\mathcal{L}$ de programmes.

Ainsi, la grammaire des E-tâche-graphes est définie comme suit :

Définition 16 *Grammaire* :

Soit $\mathcal{L}$ une bibliothèque de programmes ($\mathcal{L} = \{p_1, p_2, \ldots, p_{|\mathcal{L}|}\}$).

Soit G un E-tâche-graphe sur $\mathcal{L}$.

La grammaire des E-tâche-graphes est définie comme ci-dessous :

$$\begin{array}{rcl} < G > & := & < M > + < M > \ | \ < M > \ | \ < F > \\ < M > & := & < G > . < G > \ | \ < S > \\ < F > & := & < S > + < F > \ | \ < S > \\ < S > & := & < E > . < S > \ | \ < E > \\ < E > & := & \text{un E-tâche-graphe élémentaire étiqueté par un } p_i \in \mathcal{L}. \end{array}$$

En résumé

Un E-tâche-graphe G est un graphe orienté qui lie deux sommets spéciaux, I et O. Ces sommets représentent, respectivement, le début et la fin de l'exécution d'une tâche **T**. Chaque *trajectoire* de G représente un *plan d'actions* (séquence de programmes à exécuter), autrement dit, la stratégie associée à la résolution d'un problème bien précis. L'ensemble $\mathcal{P}$ de tous les plans d'actions décrits par la stratégie représentée par **T** est, lui, représenté par un faisceau (qui contient toutes les trajectoires de G).

La détermination de $\mathcal{P}$ à partir de G se fait à l'aide des propriétés des opérations de construction des E-tâche-graphes. Ces propriétés sont issues d'une sémantique définie pour les E-tâche-graphes en tant que modèle de représentation de la stratégie d'une tâche.

2.4 Sémantique des E-tâche-graphes

Rappelons qu'un E-tâche-graphe a pour but de représenter une stratégie pour résoudre un problème (exprimée par une tâche) pour le pilotage d'une bibliothèque de programmes. La sémantique d'un E-tâche-graphe à la fois est associée à l'exécution de programmes et reproduit la sémantique de la décomposition et de la spécialisation d'une tâche ; elle est définie par une fonction d'interprétation $\mathcal{I}$, décrite ci-dessous, des E-tâche-graphes vers une bibliothèque de programmes.

Définition 17 *Sémantique des E-tâche-graphes*:

Soit $\mathcal{L}$ l'ensemble des programmes existants dans une bibliothèque ($\mathcal{L} = \{p_1, p_2, \ldots, p_{|\mathcal{L}|}\}$).

Soit $\mathcal{G}$ un ensemble d'E-tâche-graphes sur $\mathcal{L}$; $G \in \mathcal{G}, G = (V, X, I, O, \lambda, \mathcal{L})$. Évidemment, pour chaque $p_i \in \mathcal{L}$, il existe $E_i \in \mathcal{G}$ tel que $E_i = (\{I_i, O_i\}, \{x_i\}, I_i, O_i, \lambda(x_i), \mathcal{L})$. Autrement dit, pour chaque programme d'une bibliothèque $\mathcal{L}$ il existe un E-tâche-graphe élémentaire E dans $\mathcal{G}$ qui le représente.

La sémantique des E-tâche-graphes est définie par[4]

$$\begin{array}{rcl} \mathcal{I} : \mathcal{G} & \longrightarrow & 2^{\mathcal{L}} \\ \mathcal{I}(E) & \longmapsto & \lambda(x) \\ \mathcal{I}(G_i \,.\, G_j) & \longmapsto & \mathcal{I}(G_i)\,\mathcal{I}(G_j) \\ \mathcal{I}(G_i + G_j) & \longmapsto & \mathcal{I}(G_i) \cup \mathcal{I}(G_j) \end{array}$$

Autrement dit,

Soient G_i et G_j, deux E-tâche-graphes dans $\mathcal{G}$, représentant les stratégies décrites par deux tâches $\mathbf{T}_i$ et $\mathbf{T}_j$, respectivement. Et soient τ_{iu} une trajectoire de G_i et τ_{jv} une trajectoire de G_j.

L'interprétation d'un E-tâche-graphe G

i) *élémentaire*, est l'exécution du programme qui lui est associé.

ii) résultant d'une *concaténation* d'E-tâche-graphes ($G = G_i \,.\, G_j$) est l'exécution de la stratégie représentée par $\mathbf{T}_j$ **après** l'exécution de celle représentée $\mathbf{T}_i$, i.e.

$$\mathcal{I}(G) \Longrightarrow \forall\, \tau \in G,\ \exists\, \tau_{iu} \in G_i \text{ et } \tau_{jv} \in G_j \mid \tau = \tau_{iu}\tau_{jv}$$

iii) résultant d'une *bifurcation* d'E-tâche-graphes ($G = G_i + G_j$) est l'exécution d'une des deux stratégies représentées par $\mathbf{T}_i$ et $\mathbf{T}_j$, i.e.

$$\mathcal{I}(G) \Longrightarrow \forall\, \tau \in G,\ \tau \in G_i \text{ ou } \tau \in G_j$$

4. $2^{\mathcal{L}}$ symbolise *l'ensemble des parties de $\mathcal{L}$*.

La fonction d'interprétation $\mathcal{I}$ permet d'établir quelques propriétés des opérations de construction des E-tâche-graphes. Ces propriétés rendent possible l'identification de stratégies qui ont la même interprétation, par conséquent des stratégies équivalentes.

2.5 Propriétés des opérations de construction

Les propriétés des opérations de construction sont basées sur la sémantique des E-tâche-graphes. Ainsi, leur démonstrations sont faites en utilisant la fonction d'interprétation.

2.5.1 Propriétés de la bifurcation

Les propriétés de la bifurcation sont dues aux propriétés correspondantes de l'union d'ensembles.

Propriété 2 *Idempotence* : $G + G = G$

En effet, $\mathcal{I}(G + G) = \mathcal{I}(G) \cup \mathcal{I}(G) = \mathcal{I}(G)$

Propriété 3 *Commutativité* : $G_i + G_j = G_j + G_i$

En effet, $\mathcal{I}(G_i + G_j) = \mathcal{I}(G_i) \cup \mathcal{I}(G_j) = \mathcal{I}(G_j) \cup \mathcal{I}(G_i) = \mathcal{I}(G_j + G_i)$

Propriété 4 *Associativité* : $(G_i + G_j) + G_k = G_i + (G_j + G_k)$

En effet,

$$\begin{aligned}\mathcal{I}(\ (G_i + G_j) + G_k) &= (\ (\mathcal{I}(G_i)\ \cup\ \mathcal{I}(G_j)\ )\ \cup\ \mathcal{I}(G_j) \\ &= \mathcal{I}(G_i)\ \cup\ (\ (\mathcal{I}(G_j)\ \cup\ \mathcal{I}(G_j)\ ) = \mathcal{I}(G_i + (G_j + G_k)\ )\end{aligned}$$

L'*idempotance* permet d'éliminer des possibles redondances, lors de la bifurcation d'E-tâche-graphes. La *commutativité* rend possible la mise en disposition des sous-E-tâche-graphes, d'un E-tâche-graphe additif, dans un certain ordre (par exemple, lexicographique des étiquettes).

2.5.2 Propriétés de la concaténation

La *concaténation* est :

Propriété 5 *Associative* : $(G_i \,.\, G_j) \,.\, G_k = G_i \,.\, (G_j \,.\, G_k)$

En effet,

$$
\begin{aligned}
&\mathcal{I}(G = (\,(G_i \,.\, G_j) \,.\, G_k)) \\
&\quad\Longleftrightarrow \forall\, \tau \in G,\ \exists\, \tau' \in G',\ G' = G_i \,.\, G_j \text{ et } \exists\, \tau_{kw} \in G_k \mid \tau = \tau'\tau_{kw} \text{ et} \\
&\qquad\quad \forall\, \tau' \in G',\ \ \exists\, \tau_{iu} \in G_i \text{ et } \tau_{jv} \in G_j \mid \tau' = \tau_{iu}\tau_{jv} \\
&\quad\Longleftrightarrow \tau = \tau_{iu}\tau_{jv}\tau_{kw} \\
&\text{Or, si } G'' = G_j \,.\, G_k,\ \tau'' = \tau_{jv}\tau_{kw} \in G'' \text{ et } \tau_{iu} \in G_i \Longleftrightarrow \tau = \tau_{iu}\tau_{jv}\tau_{kw} = \tau_{iu}\tau'' \\
&\text{Alors si } G''' = G_i \,.\, (G_j \,.\, G_k),\ \tau = \tau_{iu}\tau_{jv}\tau_{kw} = \tau_{iu}\tau'',\ \tau \in G''' \\
&\quad\Longleftrightarrow \mathcal{I}(G''' = G_i \,.\, (G_j \,.\, G_k)\,)
\end{aligned}
$$

Propriété 6 *Distributive à gauche sur la bifurcation* : $G_i \,.\, (G_j + G_k) = (G_i \,.\, G_j) + (G_i \,.\, G_k)$

En effet,

$$
\begin{aligned}
&\mathcal{I}(G = G_i \,.\, (G_j + G_k)) \\
&\quad\Longleftrightarrow \forall\, \tau \in G,\ \exists\, \tau_{iu} \in G_i \text{ et } \exists\, \tau' \in G',\ G' = (G_j + G_k) \mid \tau = \tau_{iu}\tau' \\
&\text{Or, } \tau' \ \in\ G' = (G_j + G_k) \ \Longleftrightarrow \tau' \in G_j \text{ ou } \tau' \in G_k \\
&\quad\Longleftrightarrow \tau = \tau_{iu}\tau',\ \tau \in (G_i \,.\, G_j) \text{ ou } \tau \in (G_i \,.\, G_k) \\
&\quad\Longleftrightarrow \mathcal{I}(G'' = (G_i \,.\, G_j) + (G_i \,.\, G_j)\,)
\end{aligned}
$$

Propriété 7 *Distributive à droite sur la bifurcation* : $(G_i + G_j) \,.\, G_k = (G_i \,.\, G_k) + (G_j \,.\, G_k)$

En effet,

$$
\begin{aligned}
&\mathcal{I}(G = (G_i + G_j) \,.\, G_k) \\
&\quad\Longleftrightarrow \forall\, \tau \in G,\ \exists\, \tau' \in G',\ G' = (G_i + G_j) \text{ et } \exists\, \tau_{kw} \in G_k \mid \tau = \tau'\tau_{kw} \\
&\text{Or, } \tau' \ \in\ G' = (G_i + G_j) \ \Longleftrightarrow \tau' \in G_i \text{ ou } \tau' \in G_j \\
&\quad\Longleftrightarrow \tau = \tau'\tau_{kw},\ \tau \in (G_i \,.\, G_k) \text{ ou } \tau \in (G_j \,.\, G_k) \\
&\quad\Longleftrightarrow \mathcal{I}(G'' = (G_i \,.\, G_k) + (G_j \,.\, G_k)\,)
\end{aligned}
$$

La *distributivité* de la concaténation sur la bifurcation permet de représenter différemment l'instant de réalisation d'un choix, comme illustre la figure 2.11. Autrement dit, soit un choix *global* est réalisé, soit des parties communes sont réalisées avant (ou après) la réalisation d'un choix.

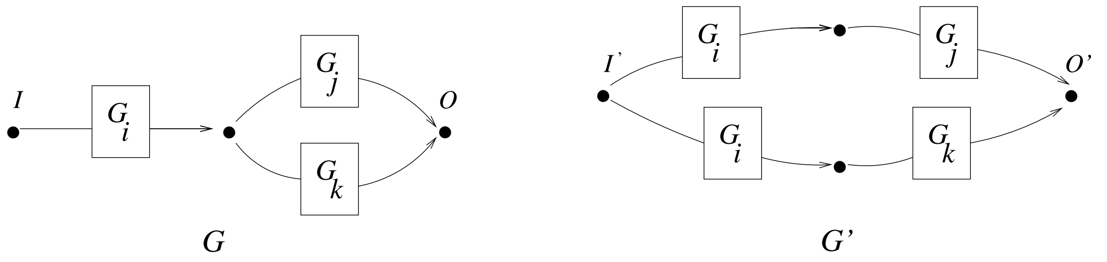


FIG. 2.11 – *Distributivité à gauche de la concaténation sur la bifurcation :* Le graphe G (à gauche) indique la réalisation d'un choix entre G_j et G_k après avoir exécuté la tâche représentée par G_i. Alors, après avoir été réalisé ce choix, soit sera exécuté $G_i\ .\ G_j$, soit $G_i\ .\ G_k$. Le choix de l'exécution complète est représenté par le graphe G', à droite, obtenu de G en utilisant la propriété distributive de la concaténation sur la bifurcation.

L'application récursive de la propriété distributive sur un E-tâche-graphe G donne comme résultat un faisceau. Un tel faisceau représente l'ensemble des plans d'actions défini par la stratégie représentée par G.

Ainsi, l'ensemble $\mathcal{P}$ des plans d'actions est déterminé à l'aide d'une *opération unaire*, dénommée *extension*, laquelle est basée sur la propriété distributive.

2.6 Opération d'extension

L'opération d'extension, définie en 2.6.1, a pour but d'exprimer explicitement toutes les trajectoires décrites par un E-tâche-graphe. La forme étendue d'un E-tâche-graphe G est donc un faisceau qui décrit chaque trajectoire de G. L'expression de la forme étendue d'un E-tâche-graphe permet d'établir un ordre, dénommé *ordre vertical*, des trajectoires des G (section 2.6.2). La sémantique de l'extension est donnée en 2.6.3. Les propriétés de la forme étendue des E-tâche-graphes sont démontrées en 2.6.4.

2.6.1 Définition

Définition 18 *Extension* : Soit $G = (V, X, I, O, \lambda, \mathcal{L})$ un E-tâche-graphe sous une configuration complexe[5]. L'E-tâche-graphe G', résultat de l'opération d'*extension* de G (noté par la suite $G' = exp(G)$) est obtenu par l'application récursive de la propriété distributive sur G (G' est dénommé *forme étendue* de G). L'E-tâche-graphe G' est un *faisceau* qui contient toutes les trajectoires de G (figure 2.12). Il représente donc l'ensemble $\mathcal{P}$ des plans d'actions défini par la stratégie indiquée par la tâche représentée par G.

5. Par définition, la propriété distributive n'est appliquée qu'à des E-tâche-graphes sous une configuration complexe.

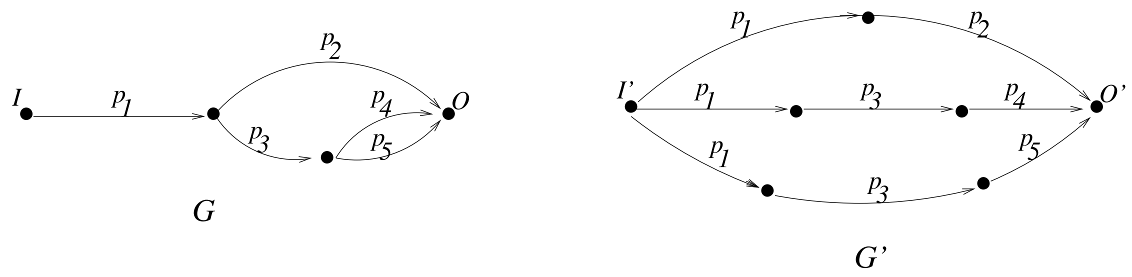


FIG. 2.12 – *Extension d'un E-tâche-graphe:* L'E-tâche-graphe G' est la *forme étendue* de G, obtenue de l'*extension* de G.

Expression de la forme étendue d'un E-tâche-graphe

Tout E-tâche-graphe G est exprimé par, $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij} \Longleftrightarrow G = \sum_{i=1}^{m} M_i$.

Or, chaque E-tâche-graphe multiplicatif M_i est exprimé par

$$M_i = \prod_{j=1}^{n} G_{ij},\ G_{ij} \text{ quelconque} \Longrightarrow G_{ij} \ = \ \sum_{k=1}^{m_j} \prod_{l=1}^{n_k} G_{ijkl} = \sum_{k=1}^{m_j} M_{ijk}.$$

Ainsi, un E-tâche-graphe multiplicatif M_i peut être exprimé par $M_i = \prod_{j=1}^{n_i} \sum_{k=1}^{m_j} M_{ijk}$ et $G = \sum_{i=1}^{m} M_i \Longleftrightarrow G = \sum_{i=1}^{m}(\prod_{j=1}^{n_i} \sum_{k=1}^{m_j} M_{ijk})$

Chacun des M_i peut donc être exprimé par

$$\begin{aligned} M_i &= \prod_{j=1}^{n_i} \sum_{k=1}^{m_j} M_{ijk} \\ &= (\sum_{k_1=1}^{m_1} M_{i1k_1}) \ . \ (\sum_{k_2=1}^{m_2} M_{i2k_2}) \ . \ (\sum_{k_3=1}^{m_3} M_{i3k_3}) \ . \ \ldots \ . \ (\sum_{k_{n_i}=1}^{m_{n_i}} M_{in_ik_{n_i}}) \end{aligned}$$

Après avoir appliqué quelques fois la propriété distributive sur chacun des M_i, un M_i peut être exprimé par

$$A'_i = \sum_{k_1=1}^{m_1} \sum_{k_2=1}^{m_2} \sum_{k_3=1}^{m_3} \ \ldots \ \sum_{k_n=1}^{m_n} M_{i1k_1} \ . \ M_{i2k_2} \ . \ M_{i3k_3} \ . \ \ldots \ . \ M_{in_ik_{n_i}}$$

Si $\forall l = 1, 2, \ldots n_i,\ M_{ilk_l}$ est une séquence alors A'_i est le faisceau $F'_i = exp(M_i)$

Sinon, en appliquant récursivement la propriété distributive sur les M_{ilk_l}, un telle condition sera satisfaite pour chaque M_{ilk_l}, à une itération quelconque, puisqu'on arrivera toujours aux composants fondamentaux de chaque M_{ilk_l} et les composants fondamentaux multiplicatifs d'un E-tâche-graphe sont des séquences.

Ainsi, $G = \sum_{i1=1}^{m} M_i \Longleftrightarrow G = M_1 + M_2 + M_3 + \ldots + M_m$

Après avoir appliqué la propriété distributive à chaque M_i, G peut être exprimé par

$$\begin{aligned} A' \; &= \sum_{k_{11}=1}^{m_{11}} \sum_{k_{12}=1}^{m_{12}} \sum_{k_{13}=1}^{m_{13}} \ldots \sum_{k_{1n_1}=1}^{m_{1n_1}} M_{11k_{11}} \,.\, M_{12k_{12}} \,.\, M_{13k_{13}} \,.\, \ldots \,.\, M_{1n_1k_{1n_1}} \\ &+ \sum_{k_{21}=1}^{m_{21}} \sum_{k_{22}=1}^{m_{22}} \sum_{k_{23}=1}^{m_{23}} \ldots \sum_{k_{2n_2}=1}^{m_{2n_2}} M_{21k_{21}} \,.\, M_{22k_{22}} \,.\, M_{23k_{23}} \,.\, \ldots \,.\, M_{2n_2k_{2n_2}} \\ &+ \sum_{k_{31}=1}^{m_{31}} \sum_{k_{32}=1}^{m_{32}} \sum_{k_{33}=1}^{m_{33}} \ldots \sum_{k_{3n_3}=1}^{m_{3n_3}} M_{31k_{31}} \,.\, M_{32k_{32}} \,.\, M_{33k_{33}} \,.\, \ldots \,.\, M_{3n_3k_{3n_3}} \\ &+ \ldots \\ + \sum_{k_{m1}=1}^{m_{m1}} & \sum_{k_{m2}=1}^{m_{m2}} \sum_{k_{m3}=1}^{m_{m3}} \ldots \sum_{k_{mn_m}=1}^{m_{mn_m}} M_{m1k_{m1}} \,.\, M_{m2k_{m2}} \,.\, M_{m3k_{m3}} \,.\, \ldots \,.\, M_{mn_mk_{mn_m}} \end{aligned}$$

A' peut donc être exprimé par

$$A' = \sum_{i=1}^{m} \sum_{k_{i1}=1}^{m_{i1}} \sum_{k_{i2}=1}^{m_{i2}} \sum_{k_{i3}=1}^{m_{i3}} \ldots \sum_{k_{in_i}=1}^{m_{in_i}} M_{i1k_{i1}} \,.\, M_{i2k_{i2}} \,.\, M_{i3k_{i3}} \,.\, \ldots \,.\, M_{in_ik_{in_i}}$$

Comme dans le cas précédent, si $\forall i = 1, 2, \ldots m,\ j = 1, 2, \ldots, n_i,\ M_{ijk_{ij}}$ est une séquence alors A' est le faisceau $F' = exp(G)$. Sinon, l'application récursive de la propriété distributive sur les $M_{ijk_{ij}}$ nous amènera à une telle condition.

2.6.2 Ordre des trajectoires d'un E-tâche-graphe

L'expression de $exp(G)$, tel comme elle est définie, induit un certain ordre de description des trajectoires de G, dénommé **ordre vertical**[6]. Graphiquement, cet ordre est représenté de haut en bas. Pour faciliter la compréhension de ce concept, nous allons tout d'abord l'exploiter pour les E-tâche-graphes sous une configuration fondamentale et ensuite pour ceux sous une configuration complexe.

Pour un E-tâche-graphe G sous une configuration fondamental, $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} E_{ij} = \sum_{i=1}^{m} S_i$, chaque S_i est une trajectoire et on fixe sa position graphique de haut en bas en respectant l'ordre de son indice. Autrement dit, la première trajectoire de haut en bas est la séquence S_1 ; la deuxième est S_2 ; et ainsi de suite la $m-$ième est S_m.

Exemple 3 *Ordre vertical des trajectoires d'un E-tâche-graphe sous une configuration fondamental*

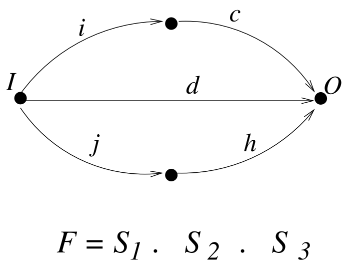


FIG. 2.13 – *Ordre vertical des trajectoires d'un faisceau.* De haut en bas, la première trajectoire de F est $\tau_1 = ic$; la deuxième, $\tau_2 = d$ et finalement la troisième, $\tau_3 = jh$

6. Par la suite, lorsqu'on mentionne *la i-ème trajectoire d'un E-tâche-graphe G*, on entendra la i-ème trajectoire dans l'ordre vertical. Par abus de langage, lorsqu'on mentionne *le i-ème arc sortant d'un sommet v*, on entendra également dans l'ordre vertical.

Pour un E-tâche-graphe G sous une configuration complexe, supposons un E-tâche-graphe multiplicatif dont tous les composants multiplicatifs sont sous une configuration fondamentale[7]. Alors,

$$M = \prod_{j=1}^{n} F_j = \prod_{j=1}^{n} \sum_{i=1}^{m_j} S_{ji}$$

$$= (\sum_{i_1=1}^{m_1} S_{1i_1}) (\sum_{i_2=1}^{m_2} S_{2i_2}) (\sum_{i_3=1}^{m_3} S_{3i_3}) \ \ldots \ (\sum_{i_{n-1}=1}^{m_{n-1}} S_{(n-1)i_{n-1}}) (\sum_{i_n=1}^{m_n} S_{ni_n})$$

La forme étendue[8] de G peut être représenté par

$$exp(G) = \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} \sum_{i_3=1}^{m_3} \ \ldots \ \sum_{i_{n-1}=1}^{m_{n-1}} \sum_{i_n=1}^{m_n} S_{1i_1} \ . \ S_{2i_2} \ . \ S_{3i_3} \ . \ \ldots \ . \ S_{(n-1)i_{n-1}} \ . \ S_{ni_n}.$$

La première séquence de $exp(G)$, τ_1 est formée par $S_{11} \ . \ S_{21} \ . \ S_{31} \ . \ \ldots \ . \ S_{(n-1)1} \ . \ S_{n1}$ puisque chaque $i_k, \ k = 1, 2, \ldots, n$ est initialisé par 1.

Puis, on varie l'index i_n de 1 à m_n. La deuxième séquence de $exp(G)$, τ_2, est donc formée par $S_{11} \ . \ S_{21} \ . \ S_{31} \ . \ \ldots \ . \ S_{(n-1)1} \ . \ S_{n2}$.

La troisième séquence de $exp(G)$, τ_3, est donc formée par $S_{11} \ . \ S_{21} \ . \ S_{31} \ . \ \ldots \ . \ S_{(n-1)1} \ . \ S_{n3}$.

Et ainsi de suite, la $(m_n - 1)$-ème séquence de $exp(G)$, $\tau_{m_{n-1}}$, est formée par

$$S_{11} \ . \ S_{21} \ . \ S_{31} \ . \ \ldots \ . \ S_{(n-1)1} \ . \ S_{nm_{n-1}}$$

La (m_n)-ème séquence de $exp(G)$, τ_{m_n}, est formée par

$$S_{11} \ . \ S_{21} \ . \ S_{31} \ . \ \ldots \ . \ S_{(n-1)1} \ . \ S_{nm_n}$$

La $(m_n + 1)$-ème séquence de $exp(G)$, $\tau_{m_{n+1}}$, est formée par

$S_{11} \ . \ S_{21} \ . \ S_{31} \ . \ \ldots \ . \ S_{(n-1)2} \ . \ S_{n1}$ puisque l'on incrémente, maintenant, l'index i_{n-1} de 1 et on re-initialise i_n par 1.

Et ainsi de suite, jusqu'à la dernière séquence, formée par

$$S_{1m_1} \ . \ S_{2m_2} \ . \ S_{3m_3} \ . \ \ldots \ . \ S_{(n-1)m_{n-1}} \ . \ S_{nm_n}$$

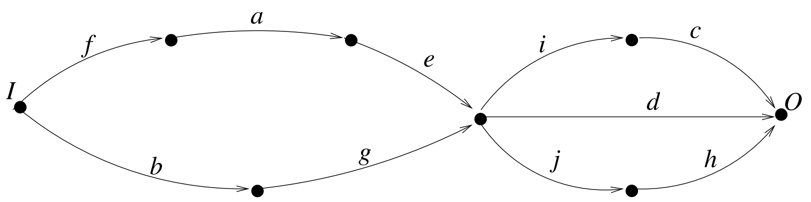
7. Cette hypothèse n'est pas restrictive car on peut toujours faire cette analyse en considérant la forme étendue (qui est un faisceau) de chaque composant multiplicatif.

8. Rappelons que l'opération d'extension d'un E-tâche-graphe G correspond à l'application récursive de la propriété distributive sur G.

Exemple 4 *Ordre vertical des trajectoires d'un E-tâche-graphe sous une configuration complexe*

Soit M un E-tâche-graphe multiplicatif formé par la concaténation de deux faisceaux, comme l'illustre la figure ci-dessous :

$$\begin{aligned} M &= \prod_{j=1}^{2} F_j = \prod_{j=1}^{2} \sum_{i=1}^{m_j} S_{ji} = (\sum_{i_1=1}^{2} S_{1i_1})(\sum_{i_2=1}^{3} S_{2i_2}) \\ &= \sum_{i_1=1}^{2} \sum_{i_2=1}^{3} S_{1i_1} S_{2i_2} \\ &= S_{11}S_{21} + S_{11}S_{22} + S_{11}S_{23} + S_{12}S_{21} + S_{12}S_{22} + S_{12}S_{23} \end{aligned}$$



$G = F_1 . \; F_2$

$S_{11} = (f\,a\,e)$

$S_{12} = (b\,g)$

$S_{21} = (i\,c)$

$S_{22} = d$

$S_{23} = (j\,h)$

Par conséquent, les trajectoires décrites par M sont, dans l'ordre vertical :

$\tau_1 = faeic$
$\tau_2 = faed$
$\tau_3 = faejh$
$\tau_4 = bgic$
$\tau_5 = bgd$
$\tau_6 = bgjh$

2.6.3 Sémantique de l'extension

La forme étendue d'un E-tâche-graphe G, d'après la définition, est un faisceau qui contient toutes les trajectoires de G. Alors, l'interprétation de la forme étendue de G est l'*union des toutes les trajectoires de* G. Par conséquent, à la fonction d'interprétation définie dans la section 2.4, peut être ajoutée la condition suivante :

$$\begin{array}{rcl} \mathcal{I} : \mathcal{G} & \longrightarrow & 2^{\mathcal{L}} \\ \mathcal{I}(exp(G)) & \longmapsto & \bigcup_{i=1}^{|\mathcal{T}|} \tau_i \end{array}$$

où $\mathcal{T}$ est l'ensemble des trajectoires de G.

2.6.4 Propriétés de l'extension

Propriété 8 : La forme étendue d'un E-tâche-graphe G sous une configuration fondamentale est identique à G. Autrement dit, si G un E-tâche-graphe sous une configuration fondamentale alors $exp(G) = G$.

En effet, d'après la définition, $exp(G)$ est un faisceau qui contient toutes les trajectoires de G.

Une séquence S (et par conséquent un E-tâche-graphe élémentaire) est un E-tâche-graphe n'ayant qu'un seul chemin allant de I à O, n'ayant donc qu'une seule trajectoire. Alors, $exp(S) = S$.

Un faisceau $F = \sum_{i=1}^{m} S_i$ est composé de m séquences allant de I à O, donc, de m trajectoires. Alors $exp(F) = F$.

Propriété 9 Si $G = G_i + G_j \Longrightarrow exp(G) = exp(G_i) + exp(G_j)$

$$\mathcal{I}(exp(G)) = \mathcal{I}(exp(G_i + G_j)) = \bigcup_{k=1}^{|\mathcal{T}'|} \tau'_k$$

où $\mathcal{T}'$ est l'ensemble des trajectoires de $G' = G_i + G_j$

$$\Longrightarrow \forall \tau'_k \in \mathcal{T}', \tau'_k \in \mathcal{T}_i \text{ ou } \tau'_k \in \mathcal{T}_j \Longleftrightarrow \tau' \in \mathcal{T}_i \cup \mathcal{T}_j \Longleftrightarrow \mathcal{T}' = \mathcal{T}_i \cup \mathcal{T}_j$$

$$\Longrightarrow \mathcal{I}(exp(G)) = \bigcup_{k=1}^{|\mathcal{T}_i|} \tau_{ik} \ \cup \ \bigcup_{k=1}^{|\mathcal{T}_j|} \tau_{jk} = \mathcal{I}(exp(G_i)) + \mathcal{I}(exp(G_j))$$

Propriété 10 Si $G = G_i \ . \ G_j \Longrightarrow exp(G) = exp(G_i) \ . \ exp(G_j)$

$$\mathcal{I}(exp(G)) = \mathcal{I}(exp(G_i \ . \ G_j)) = \bigcup_{k=1}^{|\mathcal{T}'|} \tau'_k$$

où $\mathcal{T}'$ est l'ensemble des trajectoires de $G' = G_i \ . \ G_j$

$$\Longrightarrow \forall \tau'_k \in \mathcal{T}', \exists \ \tau_{iu} \in \mathcal{T}_i \text{ et } \exists \ \tau_{jv} \in \mathcal{T}_j \mid \tau'_k = \tau_{iu}\tau_{jv}$$

où $\mathcal{T}_i$ et $\mathcal{T}_j$ sont les ensembles des trajectoires de G_i et G_j, respectivement.

Or, $F_i = \sum_{u=1}^{|\mathcal{T}_i|} \tau_{iu} \Longrightarrow F_i = exp(G_i)$;
$F_j = \sum_{v=1}^{|\mathcal{T}_j|} \tau_{jv} \Longrightarrow F_j = exp(G_j)$
$\Longrightarrow \forall \tau'_k \in \mathcal{T}', \exists \ \tau_{iu} \in exp(G_i)$ et $\exists \ \tau_{jv} \in exp(G_j) \mid \tau'_k = \tau_{iu}\tau_{jv}$
$\Longrightarrow \forall \tau'_k \in \mathcal{T}', \tau'_k \in (exp(G_i) \ . \ exp(G_j)) = \mathcal{I}(exp(G_i) \ . \ exp(G_j))$

2.6.5 Conclusion

L'ensemble $\mathcal{P}$ des plans d'actions, décrits par la stratégie représentée par un E-tâche-graphe G, figurent explicitement dans la forme étendue de G (puisque la forme étendue d'un E-tâche-graphe G représente explicitement toutes les trajectoires d'un E-tâche-graphe et une trajectoire est la représentation d'un plan d'actions).

Cependant, le nombre de trajectoires décrites par un E-tâche-graphe est multiplié en fonction de la quantité de concaténations de choix existante. Le nombre exact de trajectoires décrites par un E-tâche-graphe sous une configuration complexe n'est pas toujours évident d'être déterminé. Toutefois il est possible d'en déterminer une borne inférieure (pour l'analyse d'une borne inférieure du nombre de trajectoires d'un E-tâche-graphe complexe, se reporter à l'Annexe C). Puisque cette borne inférieure est d'ordre exponentiel, l'utilisation de la forme étendue d'un E-tâche-graphe pour vérifier l'équivalence entre deux stratégies n'est pas toujours conseillée.

Une forme normale pour des E-tâche-graphes est donc recherchée. Dans la section suivante, nous présentons un système de règles de réécriture des E-tâche-graphes qui nous permet de déterminer la forme normale d'un E-tâche-graphe.

2.7 Forme normale d'un E-tâche-graphe

La forme normale d'un E-tâche-graphe est déterminée en utilisant un système de règles de réécriture (section 2.7.2) basées sur les propriétés des opérations de constructions des E-tâche-graphes. Un système de réécriture est un ensemble de règles qui permettent d'écrire de différentes manières des entités d'un ensemble ayant la même sémantique. La description des entités se fait à l'aide des opérateurs de construction de l'ensemble. Dans le cas des E-tâche-graphes par exemple, une règle de réécriture permet de, étant donné un E-tâche-graphe G, trouver un E-tâche-graphe G' qui décrit le même ensemble des trajectoires. Les règles de réécriture du système proposé dans la section 2.7.2 utilisent les opérations de concaténation et de bifurcation d'E-tâche-graphes ainsi que les propriétés de ces opérations.

Une forme normale unique pour une entité peut être déterminée par un tel système si ce dernier possède les propriétés de *terminaison finie* et de *confluence* [Der90]. Puisque la bifurcation est une opération *commutative* et une forme normale doit être *unique*, nous proposons l'utilisation d'un ordre lexicographique des E-tâche-graphes dans le processus de recherche de la forme normale d'un E-tâche-graphe. Nous avons recouru à un système de *réécriture conditionnel* [Rem82] avec *priorité* [Bae87], toujours au profit de l'unicité de la forme normale.

Dans cette section, nous définissons tout d'abord l'ordre lexicographique des E-tâche-graphes (section 2.7.1). Ensuite, nous proposons un système de réécriture pour les E-tâche-graphes (section 2.7.2). Puis, nous présentons des propriétés des formes normales (section 2.7.3) et finalement, nous décrivons un algorithme pour déterminer la forme normale d'un E-tâche-graphe (section 2.7.4).

2.7.1 Ordre lexicographique des E-tâche-graphes

L'*ordre lexicographique* des E-tâche-graphes est basé sur l'ordre lexicographique des étiquettes de leurs arcs (qui sont des noms de programmes existants dans une bibliothèque). La définition d'un *ordre lexicographique* entre des E-tâche-graphes est nécessaire puisque, par exemple, deux faisceaux F_1 et F_2, composés du même ensemble des séquences, mais indexés différemment, sont plus facilement identifiés comme égaux si les séquences sont disposées en ordre lexicographique dans les deux faisceaux.

Définition 19 *E-tâche-graphe en ordre lexicographique* :

Un E-tâche-graphe $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij} = \sum_{i=1}^{m} M_i,\ G$ est en *ordre lexicographique* si chaque M_i l'est et si $M_i < M_{i+1},\ \forall\, i = 1, 2, \ldots, m-1$.

Un E-tâche-graphe multiplicatif $M = \prod_{j=1}^{n} G_j$, M est en *ordre lexicographique* si chaque G_j l'est.

Exemple 5 *Faisceau en ordre lexicographique*

Soit F un faisceau composé par la bifurcation de 3 séquences :

$$S' = p_4 \,.\, p_3 \,.\, p_2;\ S'' = p_5 .\, p_7 \ \text{ et } \ S''' = p_4 \,.\, p_3 \,.\, p_1 \,.\, p_6.$$

Supposons, maintenant, que la liste ordonnée des noms des p_i soit donnée par $p_1 \prec p_2 \prec p_3 \prec p_4 \prec p_5 \prec p_6 \prec p_7$ (où $\prec$ est l'ordre lexicographique des mots). Alors, la liste ordonnée des "étiquettes" des séquences S_i, obtenues par la concaténation des noms de ses composants élémentaires, est donc $S''' \prec S' \prec S''$.

Alors, F est en ordre lexicographique si et seulement si $F = S''' + S' + S''$.

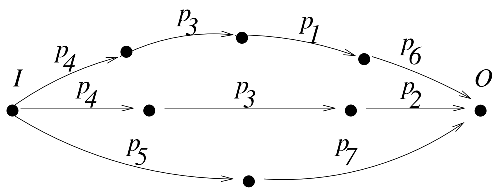


FIG. 2.14 – *Faisceau en ordre lexicographique*

Ainsi, pour vérifier si un E-tâche-graphe est en ordre lexicographique, il est nécessaire de définir une relation d'ordre lexicographique entre E-tâche-graphes multiplicatifs.

Définition 20 *Relation d'ordre lexicographique entre graphes multiplicatifs* :

Soit $G = (V, X, I, O, \lambda, \mathcal{L})$ un E-tâche-graphe sur $\mathcal{L}$.

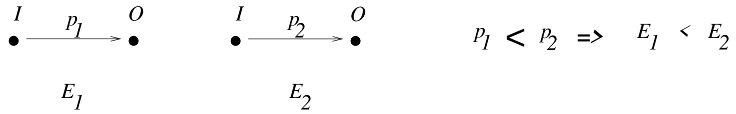
Rappelons que si G est élémentaire, $G = E = (\{I, O\}, \{x\}, I, O, \lambda(x), \mathcal{L})$.

Soit "$\prec$" l'ordre lexicographique des mots. On note que $\forall E_i, E_j \in \mathcal{G}$ (où $\mathcal{G}$ est l'ensemble des E-tâche-graphes sur $\mathcal{L}$), $\lambda(x_i) \neq \lambda(x_j)$ puisque pour chaque programme p_i de $\mathcal{L}$ il existe un et un seul E-tâche-graphe élémentaire E_i en $\mathcal{G}$ qui représente p_i.

La relation d'ordre lexicographique entre graphes multiplicatifs est définie comme ci-après[9] :

1) Soient E_i, E_j deux E-tâche-graphes élémentaires

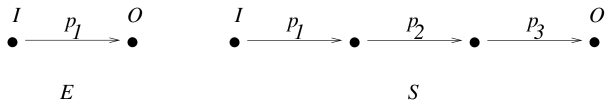
$E_i < E_j$ si $\lambda(x_i) \prec \lambda(x_j)$



Puisque $\lambda(E_1) = p_1 \prec \lambda(E_2) = p_2,\ E_1 < E_2$.

2) Soient E un E-tâche-graphe élémentaire et S une séquence ; $E < S$ si

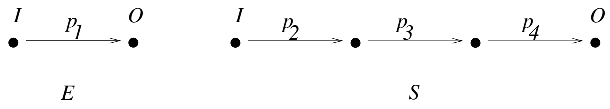
i) S est tel que $S = ES'$



$S = p_1 \, . \, p_2 \, . \, p_3 = p_1 \, . \, (p_2 \, . \, p_3)$.

Si $S' = p_2 \, . \, p_3$ alors $S = E \, . \, S'$ et $E < S$.

ii)S est tel que $S = E'S'$ et $\lambda(x) \prec \lambda(x')$



$S = p_2 \, . \, p_3 \, . \, p_4 = p_2 \, . \, (p_3 \, . \, p_4)$.

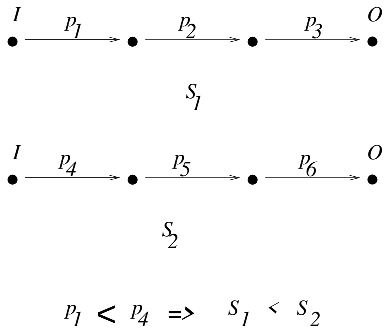
Si $E' = p_2$ et $S' = p_3 \, . \, p_4$, $S = E' \, . \, S'$.

Puisque $\lambda(E) = p_1 \prec \lambda(E') = p_2$, $E < S$.

9. Pour chaque item de la définition on illustre avec un exemple. Pour les exemples, on pose $p_i \prec p_{i+1}$ où $\prec$ est l'ordre lexicographique des noms des programmes dans $\mathcal{L}$.

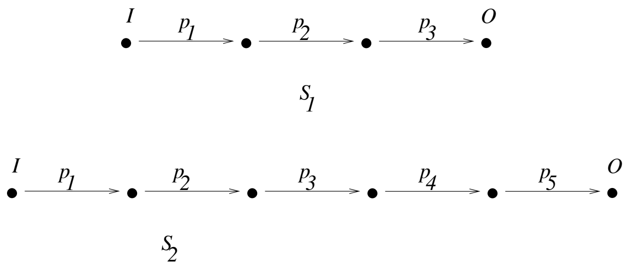
3) Soient S_i, S_j deux séquences. $S_i < S_j$ si

i) S_i et S_j sont telles que $S_i = E'_i S'_i$ et $S_j = E'_j S'_j$ et $\lambda(x'_i) < \lambda(x'_j)$



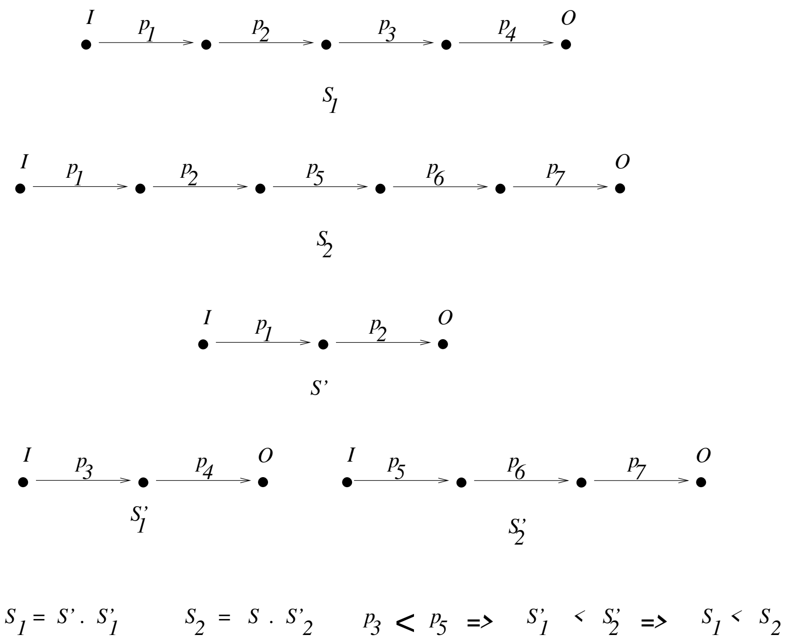
$S_1 = p_1 \, . \, p_2 \, . \, p_3 = p_1 \, . \, (p_2 \, . \, p_3)$;
$S_2 = p_4 \, . \, p_5 \, . \, p_6 = p_4 \, . \, (p_5 \, . \, p_6)$.
Si $E'_1 = p_1$ et $S'_1 = p_2 \, . \, p_3$ alors $S_1 = E'_1 \, . \, S'_1$.
Si $E'_2 = p_4$ et $S'_2 = p_5 \, . \, p_6$ alors $S_2 = E'_2 \, . \, S'_2$.
Puisque $\lambda(E'_1) = p_1 \prec \lambda(E'_2) = p_4$, $S_1 < S_2$.

ii) S_j est tel que $S_j = S_i S'_j$



$S_1 = p_1 \, . \, p_2 \, . \, p_3$;
$S_2 = p_1 \, . \, p_2 \, . \, p_3 \, . \, p_4 \, . \, p_5 = (p_1 \, . \, p_2 \, . \, p_3) \, . \, (p_4 \, . \, p_5)$.
Si $S'_2 = p_4 \, . \, p_5$ alors $S_2 = S_1 \, . \, S'_2 \Longrightarrow S_1 < S_2$.

iii) $\exists \, S'$ tel que $S_i = S' S'_i$ et $S_j = S' S'_j$ avec $S'_i < S'_j$

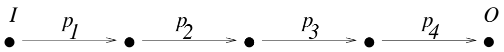


$S_1 = p_1 \, . \, p_2 \, . \, p_3 \, . \, p_4 = (p_1 \, . \, p_2) \, . \, (p_3 \, . \, p_4)$;
$S_2 = p_1 \, . \, p_2 \, . \, p_5 \, . \, p_6 \, . \, p_7 = (p_1 \, . \, p_2) \, . \, (p_5 \, . \, p_6 \, . \, p_7)$.
Si $S' = p_1 \, . \, p_2$; $S'_1 = p_3 \, . \, p_4$ et $S'_2 = p_5 \, . \, p_6 \, . \, p_7$ alors $S_1 = S' \, . \, S'_1$ et $S_2 = S' \, . \, S'_2$.
Puisque $S'_1 < S'_2$ (conforme item 3.i), $S_1 < S_2$.

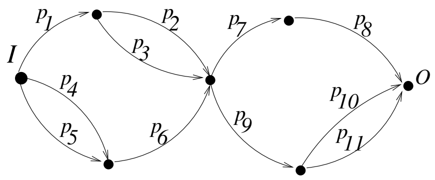
4) Soient S une séquence et M un E-tâche-graphe multiplicatif ;

Soit τ est la première trajectoire de M.

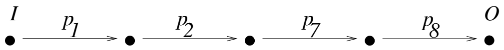
$S < M$ si S et M sont tels que M est en ordre lexicographique et $S < \tau$.



S



M



τ_1

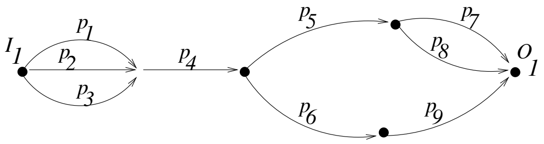
$S < \tau_1 \Rightarrow S < M$

$S = p_1 \cdot p_2 \cdot p_3 \cdot p_4$. La première trajectoire de M est $\tau_1 = p_1 \cdot p_2 \cdot p_7 \cdot p_8$.

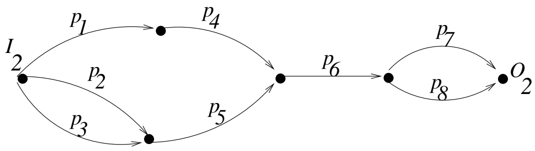
Puisque $S < \tau_1$ (conforme item 3.iii), $S < M$.

5) Soient M_i, M_j deux E-tâche-graphes multiplicatifs.
Et soient n_i le nombre de trajectoires de M_i
n_j le nombre de trajectoires de M_j ;
τ_{il} la l-ème trajectoire de M_i et
τ_{jl} la l-ème trajectoire de M_j ;
$M_i < M_j$ si M_i et M_j sont tels que M_i et M_j sont en ordre lexicographique et

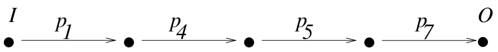
i) $\tau_{i1} < \tau_{j1}$



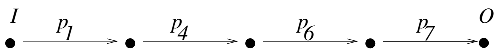
M_1



M_2



τ_{11}



τ_{21}

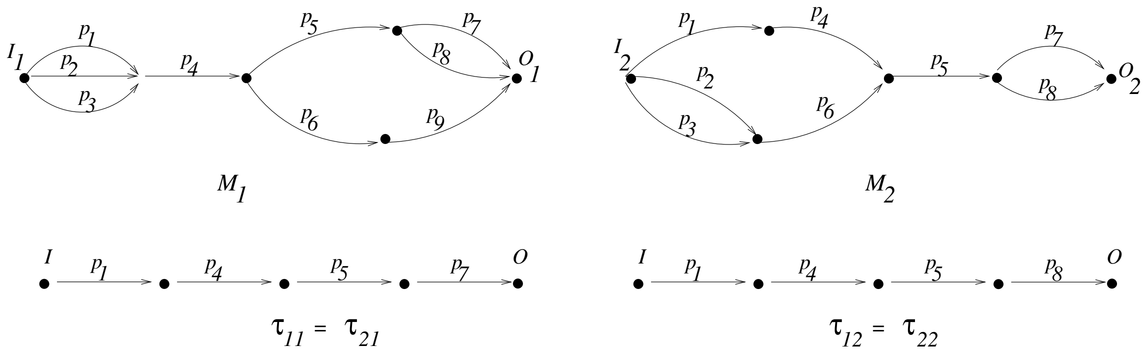
$\tau_{11} < \tau_{21} \Rightarrow M_1 < M_2$

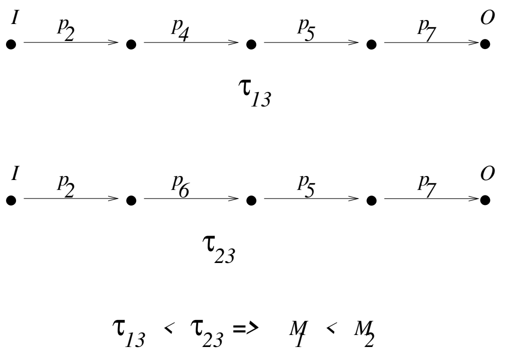
La première trajectoire de M_1 est $\tau_{11} = p_1 \cdot p_4 \cdot p_5 \cdot p_7$.

La première trajectoire de M_2 est $\tau_{21} = p_1 \cdot p_4 \cdot p_6 \cdot p_7$.

Puisque $\tau_{11} < \tau_{21}$ (conforme item 3.iii), $M_1 < M_2$.

ii) $\exists k,\ k \leq min\{n_i, n_j\} \mid \tau_{il} = \tau_{jl}\ \forall\ 1 \leq l < k$ et $\tau_{ik} < \tau_{jk}$



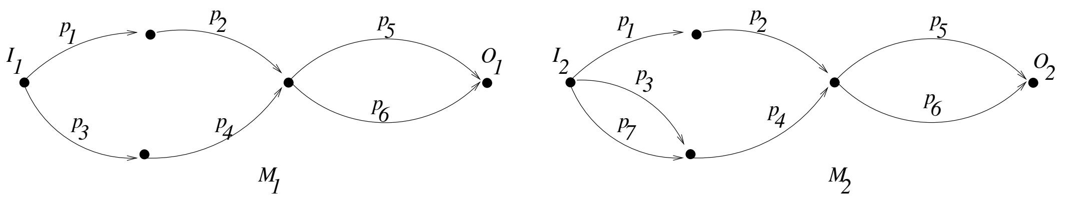


Les deux premières trajectoires de M_1 et M_2 sont égales.

Par contre, la troisième trajectoire de M_1 est $\tau_{13} = p_2 \ . \ p_4 \ . \ p_5 \ . \ p_7$; et la troisième trajectoire de M_2 est $\tau_{23} = p_2 . p_6 . p_5 . p_7$.

Puisque $\tau_{13} < \tau_{23}$ (conforme item 3.i), $M_1 < M_2$.

iii) $n_i < n_j$ et $\tau_{il} = \tau_{jl}\ \forall\ 1 \leq l \leq n_i$



Les quatre premières trajectoires de M_2 sont égales aux trajectoires correspondantes en M_1. De plus, M_2 contient encore deux autres trajectoires : $\tau_{25} = p_7 \ . \ p_4 \ . \ p_5$ et $\tau_{26} = p_7 \ . \ p_4 \ . \ p_6$

Même si en $5.ii$ et en $5.iii$ on compare les trajectoires des deux E-tâche-graphes, il n'est pas nécessaire de toutes les énumérer pour les comparer. Les algorithmes ***Comparer-sommets*** et ***Comparer-arcs***, qui sont caractérisés par des appels récursifs croisés, dont la complexité [10] est $\mathcal{O}(|V_i| + |X_i|)$, permettent de comparer les trajectoires de deux E-tâche-graphes multiplicatifs, $M_i = (V_i, X_i, I_i, O_i, \lambda_i, \mathcal{L})$ et $M_j = (V_j, X_j, I_j, O_j, \lambda_j, \mathcal{L})$, quant à l'ordre lexicographique.

Ci-après nous donnons les grandes lignes de ces algorithmes. Ils sont décrits en détails dans l'Annexe D où nous montrons également quelques exemples de chaque cas particulier.

10. Pour l'analyse de la complexité, se reporter à l'Annexe E, section E.1.

Comparer-sommets (v_i, v_j)
Soient x_{ik} le $k^{ième}$ arc dans l'ordre vertical sortant de v_i et
x_{jk} le $k^{ième}$ arc dans l'ordre vertical sortant de v_j
Pour tout arc x_{ik} *non marqué* sortant de v_i
s'il existe un arc x_{jk} sortant de v_j
Comparer-arcs (x_{ik}, x_{jk})
sinon
$\%$ (le sommet v_i possède au moins un arc *sortant* de plus que le sommet v_j)
si M_j possède encore des trajectoires à évaluer
$M_i < M_j$
sinon
$M_i > M_j$
fin-si
fin-si
fin-pour
si tous les arcs partant de v_j en M_j sont marqués
$\%$ (c'est-à-dire, ils ont été également tous évalués)
si $v_i = I_i$ $\%$ (et, par conséquent, $v_j = I_j$)
$M_i = M_j$
sinon
marquer v_i et v_j
fin-si
sinon
$\%$ (le sommet v_j possède au moins un arc *sortant* de plus que le sommet v_i)
si M_i possède encore des trajectoires à évaluer
$M_i > M_j$
sinon
$M_i < M_j$
fin-si
fin-si
fin

Comparer-arcs (x_{ik}, x_{jk})
Soient $(x_{ik} = (\overrightarrow{v_i, w_i}))$
$(x_{jk} = (\overrightarrow{v_j, w_j}))$
$\lambda_i(x_{ik})$ l'étiquette de x_{ik} et
$\lambda_j(x_{jk})$ l'étiquette de x_{jk}
si $\lambda_i(x_{ik}) \prec \lambda_j(x_{jk})$
$M_i < M_j$
sinon
si $\lambda_i(x_{ik}) \succ \lambda_j(x_{jk})$
$M_i > M_j$
sinon $\%$ (les étiquettes sont égales)
marquer x_{ik} et x_{jk}
si w_i et w_j non marqueés
Comparer-sommets (w_i, w_j)
fin-si
fin-si
fin-si
fin

2.7.2 Système de réécriture

Dans le but de déterminer une forme normale pour des E-tâche-graphes qui décrivent le même ensemble des trajectoires (autrement dit, des E-tâche-graphes qui représentent le même ensemble des plans d'actions), nous proposons un système de réécriture des E-tâche-graphes lequel se base sur les opérations de construction des E-tâche-graphes et leurs propriétés. Toutefois, avant de présenter ce système de réécriture (section 2.7.2.2) nous faisons un rappel des concepts de base relatifs à des systèmes de réécritures (section 2.7.2.1).

2.7.2.1 Définitions

Le but de cette partie est de donner les principales définitions relatives aux systèmes de réécriture et qui sont nécessaires pour la compréhension de notre système. Pour plus de détails, se reporter à [Der90] (systèmes de réécriture), [Rem82] (systèmes de réécriture conditionnels) et [Bae87] (systèmes de réécriture avec priorité).

Définition 21 *Relation de réécriture* : Soit $\mathcal{H}$ un ensemble et $\mathcal{R}$ une *relation de réécriture* entre les éléments de $\mathcal{H}$.

La relation $\mathcal{R}$ est définie par un ensemble de règles,

$$\begin{array}{ll} r_1: & h_i \longrightarrow h_j \\ r_2: & h_k \longrightarrow h_l \\ \ldots & \\ r_n: & h_u \longrightarrow h_v \end{array}$$

Définition 22 *Relation de réduction* : La fermeture réflexive et transitive $\stackrel{*}{\longrightarrow}_{\mathcal{R}}$ de la relation $\longrightarrow_{\mathcal{R}}$ est appelé *relation de réduction* engendrée par $\longrightarrow_{\mathcal{R}}$.

Définition 23 *Paire critique* : S'il existe r_i et $r_j \in \mathcal{R}$ et $h, h_i, h_j \in \mathcal{H}$ tel que $h \longrightarrow_{r_j} h_i$ et $h \longrightarrow_{r_j} h_j$ alors r_i et r_j sont dits *paire critique* de $\mathcal{R}$.

Définition 24 *Forme normale* : Un élément h de $\mathcal{H}$ est une *forme normale* s'il il est irréductible, c'est-à-dire s'il n'existe pas $h' \in \mathcal{H}$ tel que $h \longrightarrow_{\mathcal{R}} h'$.

Si h n'est pas irréductible et il existe $h' \in \mathcal{H}$ tel que $h \stackrel{*}{\longrightarrow}_{\mathcal{R}} h'$ et h' est irréductible alors h' est dit la *forme normale* de h.

Définition 25 *Terminaison finie* : Une relation $\longrightarrow_{\mathcal{R}}$ est à *terminaison finie* s'il n'existe pas de suite infinie de relation de réduction

$$h_0 \longrightarrow_{\mathcal{R}} h_1 \longrightarrow_{\mathcal{R}} h_2 \longrightarrow_{\mathcal{R}} \ldots \longrightarrow_{\mathcal{R}} h_k \longrightarrow_{\mathcal{R}} \ldots .$$

Définition 26 *Confluence* : Une relation $\longrightarrow_{\mathcal{R}}$ est *confluente* si

$$\forall h, h_i, h_j \in \mathcal{H} \text{ on a } h \overset{*}{\longrightarrow}_{\mathcal{R}} h_i \text{ et } h \overset{*}{\longrightarrow}_{\mathcal{R}} h_j \Longrightarrow \exists h' \in \mathcal{H} \text{ tel que } h_i \overset{*}{\longrightarrow}_{\mathcal{R}} h' \overset{*}{\longleftarrow}_{\mathcal{R}} h_j$$

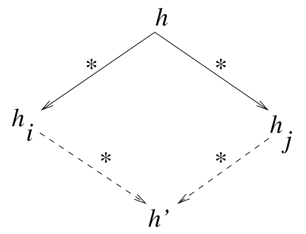


FIG. 2.15 – *Relation confluente*

Définition 27 *Confluence modulo une relation d'équivalence* $\sim$: Une relation $\longrightarrow_{\mathcal{R}}$ est *confluente modulo une relation d'équivalence* $\sim$ si

$\forall\ h_i, h_j, h'_i, h'_j \in \mathcal{H}$ on a $h_i \sim h_j$, $h_i \overset{*}{\longrightarrow}_{\mathcal{R}} h'_i$ et $h_j \overset{*}{\longrightarrow}_{\mathcal{R}} h'_j$ si et seulement si $\exists\ h''_i$ et $h''_j \in \mathcal{H}$ tel que $h'_i \overset{*}{\longrightarrow}_{\mathcal{R}} h''_i$, $h'_j \overset{*}{\longrightarrow}_{\mathcal{R}} h''_j$ et $h''_i \sim h''_j$

De plus, si $h''_i \sim h'_j$ alors la relation $\longrightarrow_{\mathcal{R}}$ est dite *cohérente modulo* $\sim$.

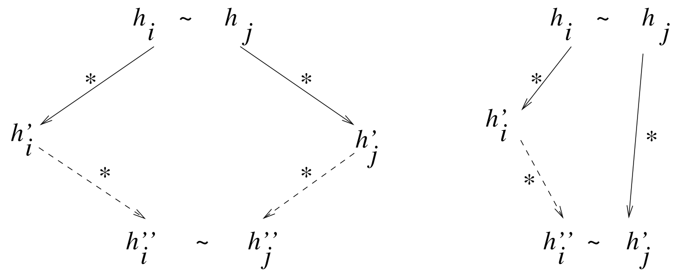


FIG. 2.16 – *Relation confluente modulo une relation d'équivalence* $\sim$: Le schéma à gauche illustre la propriété de *confluence modulo* $\sim$. Celui à droite, la *cohérence modulo* $\sim$.

Lorsqu'une règle exprime une propriété restreinte à un sous-ensemble de $\mathcal{H}$, un *système de réécriture conditionnel* s'impose. Autrement dit, la partie " gauche" d'une règle doit indiquer la condition nécessaire pour que la règle soit applicable.

Définition 28 *Système de réécriture conditionnel* : Dans un *système de réécriture conditionnel* [Rem82] la relation $\mathcal{R}$ possède au moins une règle décrite comme :

$$r_k: \quad cond_k(h) \Longrightarrow h \longrightarrow h_i$$

Lorsque plus d'une règle peut être applicable à un élément de $\mathcal{H}$, un *système de réécriture avec priorité* permet d'indiquer laquelle doit être utilisée en priorité. Ceci est une façon de résoudre le problème des paires critiques.

Définition 29 *Système de réécriture avec priorité* : Un *système de réécriture avec priorité* est une paire $(\mathcal{R}, <)$ où $<$ est une relation d'ordre partiel définie sur l'ensemble des règles [Bae87]. La priorité d'une règle r_k sur une règle r_l (noté par $r_k > r_l$) est indiquée comme ci-dessous :

$$\left\downarrow \begin{array}{ll} r_k\colon & h \longrightarrow h_i \\ r_l\colon & h' \longrightarrow h_j \end{array}\right.$$

où h et h' sont des expressions différentes auxquelles un élément de $\mathcal{H}$ peut être associé.

Par exemple, dans les cas des E-tâche-graphes, si $h = S_i + S_j$ (la bifurcation de deux séquences) et $h' = M_i + M_j$ (la bifurcation de deux E-tâche-graphes multiplicatifs), la règle r_l ne sera appliquée qu'aux E-tâche-graphes dont au moins l'un des composants n'est pas une séquence. La priorité de r_k sur r_l l'oblige, même si " la bifurcation de deux séquences" soit également " la bifurcation de deux E-tâche-graphes multiplicatifs".

De plus, supposons qu'un élément h_u de $\mathcal{H}$ soit exprimé d'une façon identifiable à h' mais pas à h. L'application de la règle r_l n'est autorisée que s'il n'existe pas de relation de réduction $h_u \xrightarrow{*}_{\mathcal{R}} h_v$ tel que h_v est identifiable à h. Autrement dit, l'application des règles dans un système avec priorité est récursive. Cette caractéristique récursive pour les E-tâche-graphes sous une configuration complexe indique qu'il faut l'analyser "en profondeur" jusqu'à leurs composants fondamentaux.

Nous avons recouru à un système de réécriture conditionnel avec priorité à cause des propriétés d'idempotence et de commutativité de la bifurcation (en donnant une priorité à l'idempotence). De plus, une priorité a été donnée à une application de la propriété distributive à gauche pour caractériser le fait qu'un choix sera effectué au plus tardivement possible, dans la forme normale.

2.7.2.2 Règles de réécriture

Le système de réécriture que nous avons défini pour déterminer la forme normale d'un E-tâche-graphe est basé sur les propriétés de ses opérations de construction.

Soit $\mathcal{G}$ un ensemble d'E-tâche-graphes sur une bibliothèque $\mathcal{L}$. La relation de réécriture $\mathcal{R}$ entre les éléments de $\mathcal{G}$ est composée des règles décrites ci-après :

r_1: $G_i = G_j \Longrightarrow G_i + G_j \longrightarrow G_i$

r_2: $(G_i + G_j) + G_k \longrightarrow \sum_{l\in\{i,j,k\}} G_l$
$G_i + (G_j + G_k) \longrightarrow \sum_{l\in\{i,j,k\}} G_l$

r_3: $M_i > M_j \Longrightarrow M_i + M_j \longrightarrow M_j + M_i$ où $>$ est l'ordre lexicographique des E-tâche-graphes

r_4: $(G_i \,.\, G_j) + (G_i \,.\, G_k) \longrightarrow G_i \,.\, (G_j + G_k)$

$\downarrow$ r_5: $(G_i \,.\, G_j) + (G_k \,.\, G_j) \longrightarrow (G_i + G_k) \,.\, G_j$

r_6: $(G_i \,.\, G_j) \,.\, G_k \longrightarrow \prod_{l\in\{i,j,k\}} G_l$
$G_i \,.\, (G_j \,.\, G_k) \longrightarrow \prod_{l\in\{i,j,k\}} G_l$

Les cinq premières règles concernent à la bifurcation : i) La première, élimine des redondances, c'est-à-dire il n'y a pas de sens dans la bifurcation de deux E-tâche-graphes égaux. Elle est prioritaire ; ii) D'après la troisième règle, la commutativité de la bifurcation n'est utilisée que pour réécrire un E-tâche-graphe additif en mettant ses composants en ordre lexicographique (sinon le système ne serait pas à terminaison finie) ; iii) Les règles r_4 et r_5 sont dénommées, respectivement, *factorisation à gauche* et *factorisation à droite*. Une factorisation est basé sur la propriété distributive et a pour but de représenter des parties communes des composants additifs d'un E-tâche-graphe, sous une configuration complexe, un nombre le plus petit possible de fois (la priorité donnée à la factorisation à gauche indique qu'un choix sera réalisé au plus tardivement possible). La priorité définie entre ces règles indique qu'un E-tâche-graphe additif doit être *non redondant* et *en ordre lexicographique* avant d'être factorisé.

Les règles basées sur les propriétés associatives (r_2, de la bifurcation ; r_6 de la concaténation) ne changent pas la représentation graphique de l'E-tâche-graphe auquel elles sont appliquées. En réalité, l'associativité permet d'enlever des *niveaux de parenthésage inutiles* de l'expression d'un E-tâche-graphe.

Exemple 6 *Forme normale (illustré par la figure 2.17 ci-après)* :

Soient $\mathcal{G}$ un ensemble d'E-tâche-graphes sur une bibliothèque $\mathcal{L}$ et

$G = (V, X, I, O, \lambda, \mathcal{L}) \in \mathcal{G}$ tel que $G = (G_1 + G_2) + G_3$ où

$G_1 = (p_5 \,.\, p_4),\ G_2 = (p_1 \,.\, p_3)$ et $G_3 = (p_1 \,.\, p_2)$

Alors, $G = ((p_5 \,.\, p_4) + (p_1 \,.\, p_3)) + (p_1 \,.\, p_2)$.

L'application récursive des règles se fera de la façon suivante :

$((p_5 \,.\, p_4)+(p_1 \,.\, p_3))+(p_1 \,.\, p_2) \longrightarrow_{r_2} (p_5 \,.\, p_4)+(p_1 \,.\, p_3)+(p_1 \,.\, p_2) \overset{*}{\longrightarrow}_{r_3} (p_1 \,.\, p_2)+(p_1 \,.\, p_3) + (p_5 \,.\, p_4) \longrightarrow_{r_4} (p_1 \,.\, (p_2 + p_3)) + (p_5 \,.\, p_4)$.

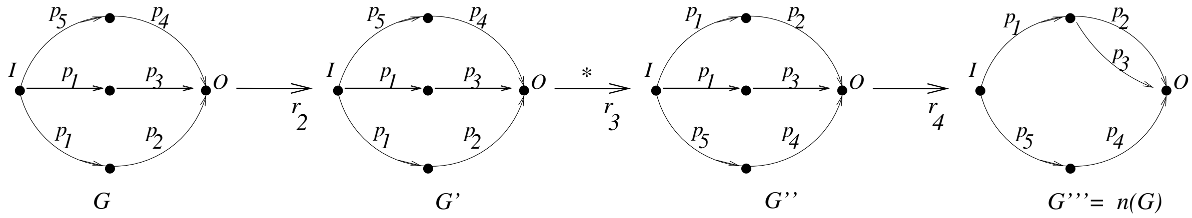


FIG. 2.17 – *Forme normale :* $G \longrightarrow_{r_2} G' \overset{*}{\longrightarrow}_{r_3} G'' \longrightarrow_{r_4} G'''$. Puisque G''' est irréductible $G''' = n(G)$. On note que, effectivement, la règle r_2 (basée sur l'*associativité* de la bifurcation) ne change pas la représentation graphique de l'E-tâche-graphe G (puisque $G' = G$).

Forme normale unique

Ce système détermine une forme normale unique pour chaque E-tâche-graphe G de $\mathcal{G}$ puisque :

(i) il est *confluent* : Les priorités données aux règles ont éliminé l'existence de paires critiques.

(ii) il est à *terminaison finie* : D'une part, des redondances sont éliminées par l'idempotence (r_1) ; d'autre part, la commutativité de la bifurcation n'engendre pas de boucle car elle n'est utilisée que pour rendre un E-tâche-graphe additif en ordre lexicographique (r_3). De plus, en raison de la récursivité d'un système de réécriture avec priorité, un E-tâche-graphe est irréductible si chacun des ses composants fondamentaux l'est.

La forme normale $n(G)$ d'un E-tâche-graphe G, obtenue à partir de G par le système défini dans cette section, est un E-tâche-graphe en ordre lexicographique, sans redondances (r_1 et r_2) et ayant une forme la plus " compacte" possible représentant des choix réalisés le plus tardivement possible ($r_4 > r_5$).

2.7.3 Propriétés des formes normales

Soient $\mathcal{G}$ un ensemble d'E-tâche-graphes sur une bibliothèque $\mathcal{L}$

et $\mathcal{R}$ la relation de réécriture entre les éléments de $\mathcal{G}$.

Propriété 11 : Si G est une séquence alors $n(S) = S$.

Une séquence est irréductible et est donc sa forme normale (définition 24).

Propriété 12 : Si G est un faisceau en ordre lexicographique non factorisable, alors $n(F) = F$.

Un faisceau F est non factorisable si $\nexists\ G \in \mathcal{G}$ tel que $F \longrightarrow_{r_5} G$ ou $F \longrightarrow_{r_6} G$. Si, de plus, F est en ordre lexicographique alors F est irréductible $\Longrightarrow$ F est une forme normale.

Propriété 13 : Si G est tel que $n(G)$ est un E-tâche-graphe *non déterministe* alors $\exists\ G'$ *irréductible déterministe* $\mid exp(G) = exp(G')$.

Nota : Cette démonstration est illustrée par l'exemple 7.

Dénotons $n(G)$ par NG. Puisque G' est irréductible, il est également une forme normale ; appelons-le NG'.

Si $NG = (V, X, I, O, \lambda, \mathcal{L})$ est non déterministe alors $\exists v \in V,\ x_i,\ x_j \in X$ tels que $x_i = (\overrightarrow{v, u}), x_j = (\overrightarrow{v, w})$ avec $\lambda(x_i) = \lambda(x_j) = \lambda_x$.

Séparons le sous-E-tâche-graphe additif $A' = (V', X', I', O', \lambda, \mathcal{L})$ de NG, tel que $I' = v$. Alors, $A' = \sum_{i=1}^{m} M'_i$.

Pour chaque $M'_i = \prod_{j=1}^{n_i} G'_{ij}$ tel que $\exists\ x'_{ik} = (\overrightarrow{I'_i, v_k})$ avec $\lambda(x'_{ik}) = \lambda_x$ et M'_i n'est pas une séquence, posons $M'_i = G'_{i1} \prod_{j=2}^{n_i} G'_{ij}$

i) si G'_{i1} est un faisceau alors

   $G'_{i1} = \sum_{l=1}^{m_{i1}} S'_{i1l}$ et $\exists k \mid S'_{i1k} = x'_{ik}$ ou $S'_{i1k} = x'_{ik} S''_{i1k}$ (rappelons que $\lambda(x'_{ik}) = \lambda_x$)

   M'_i peut être donc réécrit comme $A'_i = S'_{i1k} \prod_{j=2}^{n_i} G'_{ij} + (\sum_{l=1; l \neq k}^{m_{i1}} S'_{i1l}) \prod_{j=2}^{n_i} G'_{ij}$ (on n'a distribué que la séquence qui commence par λ_x).

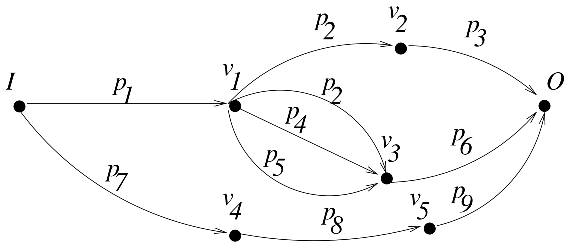
   Si l'on ordonne G' ainsi réécrit et on le factorise à gauche, il n'existe plus qu'un seul arc sortant de I' ayant λ_x comme étiquette.

ii) si G'_{i1} n'est pas un faisceau (il est donc un E-tâche-graphe additif plus complexe) on refait récursivement l'analyse ci-dessus jusqu'à arriver à un niveau où le premier composant fondamental soit un faisceau. Le but est de distribuer, puis factoriser, les composants fondamentaux d'abord.

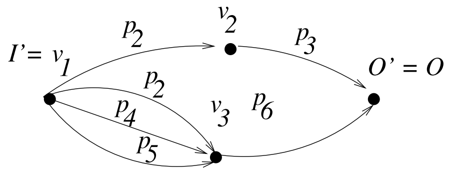
NG' ainsi obtenu de NG décrit le même ensemble de trajectoires que G puisque l'on utilise simplement la propriété distributive et des règles de réécriture (pour ordonner et factoriser). Par conséquent $exp(NG') = exp(NG)$.

Exemple 7 *Déterminisation d'un E-tâche-graphe*

Dans la figure ci-dessous, NG est non déterministe car il existe deux arcs sortant de v_1 ayant la même étiquette $p_2 = \lambda(\overrightarrow{v_1, v_2}) = \lambda(\overrightarrow{v_1, v_3})$. Le sous-E-tâche-graphe additif A' de NG tel que $I' = v_1$ est illustré à droite de la figure.

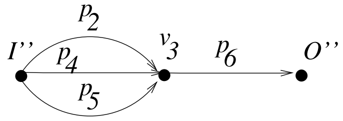


NG

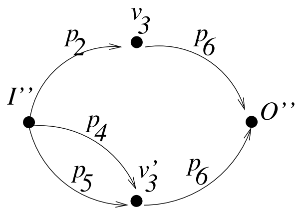


A' = M₁ + M₂

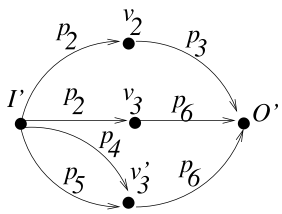
L'arc $(\overrightarrow{I', v_3})$ étiqueté par p_2 en M_2 appartient à un faisceau. Alors M_2 peut être réécrit comme l'E-tâche-graphe additif, illustré au milieu de la figure ci-dessous, en utilisant la distributivité à droite seulement pour la séquence commençant par p_2. En remplaçant M_2 par A_2 dans A', on obtient l'E-tâche-graphe à droite.



M₂

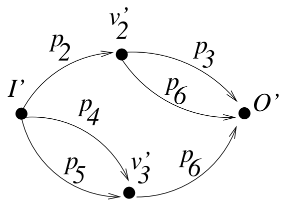


A 2

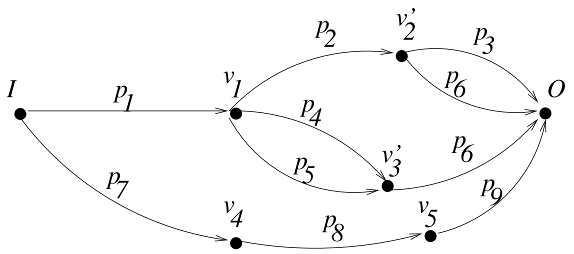


G'

La factorisation de A', réécrit, donne comme résultat l'E-tâche-graphe déterministe illustré à gauche de la figure ci-dessous. Lorsque A', réécrit et factorisé, est remplacé dans NG, il en résulte l'E-tâche-graphe à droite, qui correspond à NG *déterminisé*.



fact(G')



NG

La démonstration de la propriété 13 suggère un processus de *déterminisation* d'une forme normale non déterministe.

Propriété 14 : Si $G = (V, X, I, O, \lambda, \mathcal{L})$ et $G' = (V', X', I', O', \lambda', \mathcal{L})$ sont tels que $n(G)$ et $n(G')$ sont *déterministes* et $exp(G) = exp(G')$ alors $n(G) = n(G')$.

Si $n(G)$ est déterministe alors

$$\forall\, v \in V, \forall\, x_i = (\overrightarrow{v, w}),\ x_j = (\overrightarrow{v, u}) \in X \text{ alors } \lambda(x_i) \neq \lambda(x_j)$$

De même, si $n(G')$ est déterministe alors

$$\forall\, v' \in V', \forall\, x'_i = (\overrightarrow{v', w'}),\ x'_j = (\overrightarrow{v', u'}) \in X' \text{ alors } \lambda'(x'_i) \neq \lambda'(x'_j)$$

Soient $\mathcal{T}$ et $\mathcal{T}'$ les ensembles des trajectoires de $n(G)$ et de $n(G')$, respectivement.

Si $exp(G) = exp(G')$ alors $exp(n(G)) = exp(n(G')) \Longrightarrow \forall \tau \in \mathcal{T}, \exists\, \tau' \in \mathcal{T}'$ tel que $\tau = \tau'$.

Soient τ et τ' trajectoires de $n(G)$ et $n(G')$, respectivement, tel que $\tau = \tau'$. Ces trajectoires peuvent être représentées, respectivement, par

$$\tau = (I, v_1, v_2, \ldots, v_{n-1}, v_n, O) = (\lambda(x_1), \lambda(x_2), \ldots, \lambda(x_n), \lambda(x_{n+1}))$$

où $x_1 = (\overrightarrow{I, v_1}), x_i = (\overrightarrow{x_{i-1}, x_i})$ $(\forall\ 2 \leq i \leq n)$ et $x_{n+1} = (\overrightarrow{x_n, O})$

et $\tau' = (I', v'_1, v'_2, \ldots, v'_{n-1}, v'_n, O') = (\lambda'(x'_1), \lambda'(x'_2), \ldots, \lambda'(x'_n)\lambda'(x'_{n+1}))$

où $x'_1 = (\overrightarrow{I', v'_1}), x'_i = (\overrightarrow{x'_{i'-1}, x'_i})$ $(\forall\ 2 \leq i \leq n)$ et $x'_{n+1} = (\overrightarrow{x'_n, O'})$.

Comme $n(G)$ est déterministe, l'arc $x_1 = (\overrightarrow{I, v_1})$ est le seul arc de G partant de I, étiqueté par $\lambda(x_1)$. De même pour $n(G')$, l'arc $x'_1 = (\overrightarrow{I', v'_1})$ est le seul arc de G' partant de I', étiqueté par $\lambda'(x'_1) = \lambda(x_1)$.

De la même façon, l'arc $x_2 = (\overrightarrow{v_1, v_2})$ est le seul arc de G partant de v_1, étiqueté par $\lambda(x_2)$ ainsi que l'arc $x'_2 = (\overrightarrow{v'_1, v'_2})$ est le seul arc de G' partant de v'_1, étiqueté par $\lambda'(x'_2)$.

Et ainsi de suite, la portée v_i (et sa correspondante v'_i) de chaque sous-trajectoire de τ (et sa correspondante τ') possède *un et un seul* arc x_i partant de v_i (et ses correspondants x'_i et v'_i) étiqueté par $\lambda(x_i)(= \lambda'(x'_i))$.

En faisant cette analyse pour chacune des trajectoires de $n(G)$ et sa correspondante en $n(G')$, on conclut que $n(G) = n(G')$.

La propriété 14 indique que, étant donné deux E-tâche-graphes G et G' tels que $exp(G) = exp(G')$ et $n(G) \neq n(G')$ alors $n(G)$ ou $n(G')$ est *non déterministe*. Toutefois, d'après la propriété 13 pour un E-tâche-graphe sous une forme normale non déterministe il existe toujours un E-tâche-graphe déterministe, également sous forme normale, équivalent au premier. Soient $n_{det}(G)$ et $n_{det}(G')$ les E-tâche-graphes, déterministes et sous forme normale, équivalents à $n(G)$ et (G'), respectivement. D'après la propriété 14, $n_{det}(G) = n_{det}(G')$. Par conséquent, la relation de réécriture $\mathcal{R}$ définie dans la section 2.7.2.2 est confluente modulo la relation d'équivalence *avoir le même ensemble de trajectoires*.

2.7.4 Algorithme

La forme normale d'un E-tâche-graphe $G = (V, X, I, O, \lambda, \mathcal{L})$, on l'a déjà dit, est un E-tâche-graphe $n(G)$ obtenu à partir de G qui représente le même ensemble des trajectoires de G possédant les caractéristiques suivantes :

i) non redondant et en ordre lexicographique ;

ii) une forme la plus compacte possible ;

iii) dans $n(G)$ les choix sont représentés le plus tardivement possible ;

iv) si $n(G)$ est sous une configuration complexe, tous ses composants sont également des formes normales.

La forme normale d'un E-tâche-graphe est déterminée par le moyen du système de réécriture défini pour les E-tâche-graphes. Ci-dessous, les grandes lignes d'un algorithme qui permet de trouver la forme normale d'un E-tâche-graphe G. La complexité de cet algorithme, dont l'analyse est faite dans l'Annexe E, section E.4, est de l'ordre $O(|X|^4)$.

1. Si G est une séquence
    alors $n(G) = G$;   (propriété 11)
2. Sinon, si G est multiplicatif ($G = \prod_{j=1}^{n} G_j$)
    alors $n(G) = \prod_{j=1}^{n} n(G_j)$;
3. Sinon (G est additif, $G = \sum_{i=1}^{m} M_i$)
    mettre G en ordre lexicographique en éliminant les redondances existantes ;
    le factoriser à gauche, si possible ;
    sinon, le factoriser à droite, si possible ;
    si ni la factorisation à gauche, ni la la factorisation à droite est possible et
        si G est un faisceau
            alors $n(G) = G$;   (propriété 12)
        sinon (G est additif, $G = \sum_{i=1}^{m} M_i$)
            $n(G) = n(\sum_{i=1}^{m} n(G_i))$
        fin-si

2.8 Conclusion

Ce chapitre a présenté les *E-tâche-graphes* comme un modèle de la stratégie indiquée par une tâche. Le but de ce modèle est de représenter le lien d'exécution entre des programmes, existants dans une bibliothèque, pour résoudre une classe de problèmes donnée.

Vu que la résolution d'un problème instancié (c'est-à-dire, un problème pour lequel les données ont été bien précisées) revient à l'exécution d'une séquence bien définie de programmes (ce qui dans le vocabulaire des modèles de tâches est appelé *plan d'actions*), la notion de *trajectoire* a été introduite. Une trajectoire représente donc un plan d'actions pour résoudre un problème donné et une stratégie décrit un ensemble de plans d'actions. Par conséquent, un E-tâche-graphe représentant la stratégie indiquée par une tâche, il représente également un ensemble de trajectoires. La représentation " explicite" de cet ensemble de trajectoires est faite par le moyen de la *forme étendue* d'un E-tâche-graphe. La forme étendue d'un E-tâche-graphe est obtenue par une opération unaire (dénommée *extension*) définie à partir d'une des propriétés (la distributivité) des opérations de construction (*concaténation* et *bifurcation*) des E-tâche-graphes. Ces propriétés sont dérivées de la sémantique des E-tâche-graphes.

Cependant, l'évaluation d'une équivalence entre deux stratégies[11] en utilisant la forme étendue de ses E-tâche-graphes correspondants, pourrait être déconseillée dans le cas d'E-tâche-graphes très complexes. Et cela à cause du nombre exponentiel de trajectoires lesquelles peuvent être décrites par un tel E-tâche-graphe. Par conséquent, une *forme normale* d'un E-tâche-graphe est proposée. La forme normale d'un E-tâche-graphe est la forme la plus compacte obtenue à partir de l'E-tâche-graphe d'origine par le moyen des règles de réécriture. Puisque des règles de réécriture préservent la sémantique des entités réécrites, un E-tâche-graphe et sa forme normale décrivent le même ensemble des trajectoires.

Le prochain chapitre est dédié à la comparaison d'E-tâche-graphes en utilisant leurs formes normales respectives.

11. Rappelons que deux stratégies sont équivalentes si elles représentent le même ensemble des plans d'actions.

Chapitre 3

Comparaison d'E-tâche-graphes

L'équivalence entre deux tâches est déterminée par l'égalité des ensembles des plans d'actions, décrits par leurs stratégies respectives. La comparaison de ces stratégies est faite à l'aide des E-tâche-graphes (étudiés dans le chapitre précédent) représentant chacune de ces stratégies. Autrement dit, on cherche à savoir si les E-tâche-graphes qui représentent les stratégies indiquées par deux tâches possèdent le même ensemble de trajectoires. Par extension, deux E-tâche-graphes qui représentent des stratégies équivalentes sont dits *E-tâche-graphes équivalents*. Dû à un probable nombre exponentiel de trajectoires lors de l'extension [1] d'un E-tâche-graphe représentant la décomposition d'une tâche très complexe, cette recherche est faite à l'aide des formes normales respectives des E-tâche-graphes. Si deux E-tâche-graphes possèdent formes normales identiques alors ils sont équivalents, car ils décrivent le même ensemble des trajectoires : celui de leur forme normale. Par ailleurs, il est possible qu'il existe une relation d'inclusion entre les ensembles de trajectoires décrits par deux E-tâche-graphes ou encore qu'ils aient des parties communes. Ceci nous amène à définir une *relation d'ordre partiel* entre les E-tâche-graphes, dénommée *plus général que* (section 3.1). En outre, il est possible que même n'ayant pas de formes normales identiques, deux E-tâche-graphes décrivent le même ensemble des trajectoires. C'est le cas où au moins l'une des deux formes normales est non déterministe [2].

Nous proposons donc de comparer deux E-tâche-graphes sous leurs formes normales respectives en utilisant la notion de *différence* (section 3.2) laquelle indique les modifications nécessaires pour transformer un E-tâche-graphe sous forme normale en un autre également sous forme normale. Ces modifications sont exprimées par des *ajouts* et des *retraits* d'arcs et l'équivalence est vérifiée lorsqu'il n'existe pas d'ajouts ni de retraits.

Un algorithme de différence entre E-tâche-graphes sous formes normales *déterministes* est proposé (section 3.2.2). Cependant, lorsque l'une des formes normales à comparer est non déterministe, d'après la propriété 13, il est toujours possible de trouver un autre E-tâche-graphe, également sous forme normale et déterministe, équivalent au premier.

1. Représentation exhaustive des trajectoires d'un E-tâche-graphe.
2. *Cf.* propriété 14, section 2.7.3.

Après une analyse de la complexité de la comparaison de deux E-tâche-graphes sous forme normale (section 3.3), nous analysons l'utilisation d'arbres de décomposition et celle d'E-tâche-graphes pour la comparaison de deux stratégies de tâches. Finalement, nous montrons un exemple d'utilisation d'E-tâche-graphes dans un système de diagnostic médical (section 3.5).

3.1 Relation d'ordre partiel entre des E-tâche-graphes

La relation d'ordre partiel *plus général que* entre les E-tâche-graphes symbolise le fait que la stratégie représentée par une tâche offre des options de choix de plans d'actions de plus que celle d'une autre tâche. Cette relation se base donc sur la relation d'*inclusion ensembliste*. Elle est définie comme suit :

Définition 30 *Plus général que* : Soient G_i et G_j deux E-tâche-graphes. G_i est dit *plus général que* G_j (noté par $G_i \succeq G_j$) si $\mathcal{T}_i \supseteq \mathcal{T}_j$ où $\mathcal{T}_i$ et $\mathcal{T}_j$ sont les ensembles des trajectoires de G_i et G_j, respectivement. Inversement, G_j est dit *plus spécifique que* G_i (noté par $G_j \preceq G_i$).

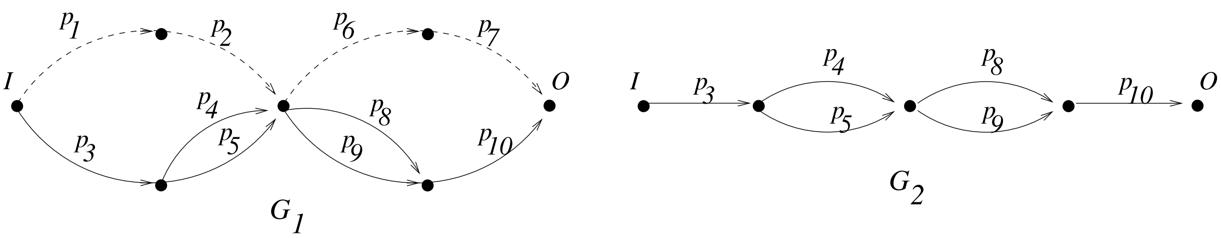


FIG. 3.1 – *Relation plus général que :* $G_1 \succeq G_2$ puisque l'ensemble des trajectoires de G_1 ($\mathcal{T}_1 = \{p_1p_2p_6p_7,\ p_1p_2p_8p_{10},\ p_1p_2p_9p_{10},\ p_3p_4p_6p_7,\ p_3p_4p_8p_{10},\ p_3p_4p_9p_{10},\ p_3p_5p_6p_7,\ p_3p_5p_8p_{10},\ p_3p_5p_9p_{10}\}$) contient celui des trajectoires de G_2 ($\mathcal{T}_1 = \{p_3p_4p_8p_{10},\ p_3p_4p_9p_{10},\ p_3p_5p_8p_{10},\ p_3p_5p_9p_{10}\}$).

On sait, de la théorie des ensembles, que si deux ensembles A et B sont tels que $A \supseteq B$ alors $A \cap B = B$. Par ailleurs, dans une relation d'ordre partiel $\succeq$ sur un ensemble $\mathcal{H}$, deux éléments h_i et h_j de $\mathcal{H}$ tels que $h_i \not\succeq h_j$ et $h_j \not\succeq h_i$, h_i et h_j sont dits *éléments non comparables*. Par conséquent, si deux E-tâche-graphes G_i et G_j sont non comparables alors

i) soit $\mathcal{T}_i \cap \mathcal{T}_j = \phi$; ils sont donc *différents* (car ils ne possèdent aucune trajectoire en commun) ;

ii) soit $\mathcal{T}_i \cap \mathcal{T}_j = \mathcal{T}'$, $\mathcal{T}' \neq \mathcal{T}_i$ et $\mathcal{T}' \neq \mathcal{T}_j$; dans ce cas ils possèdent quelques trajectoires communes.

Ce dernier cas signifie que les E-tâche-graphes G_i et G_j représentent des stratégies capables de résoudre quelques problèmes communs. Dans ce cas là, G_i et G_j sont dits *étroitement* *appariés*[3]. Cette relation est définie formellement ci-après :

Définition 31 *Étroitement apparié* : Soient G_i et G_j deux E-tâche-graphes non comparables par la relation $\succeq$. Si $\exists\ \mathcal{T}' \neq \phi$ tel que $\mathcal{T}' = \mathcal{T}_i \cap \mathcal{T}_j$, alors G_i et G_j sont dits *étroitement appariés* (noté par $G_i \approx_{\succeq} G_j$).

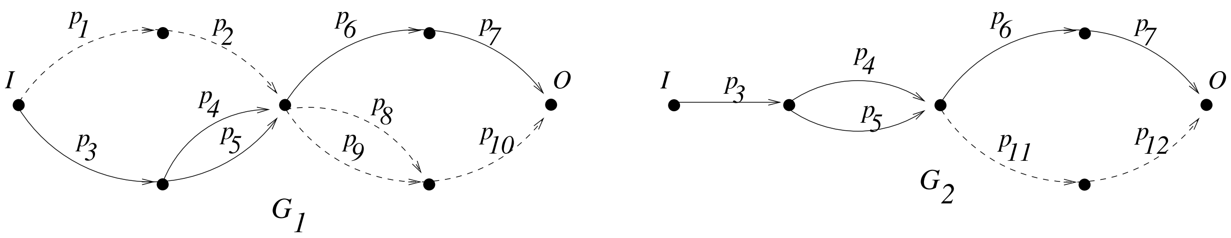


FIG. 3.2 – *Relation étroitement apparié :* $G_1 \approx_{\succeq} G_3$ puisqu'ils possèdent en commun l'ensemble des trajectoires $\mathcal{T}' = \{p_3p_4p_6p_7,\ p_3p_5p_6p_7\}$. G_1 possède sept autres trajectoires : $p_1p_2p_6p_7,\ p_1p_2p_8p_{10},\ p_1p_2p_9p_{10},\ p_3p_4p_8p_{10},\ p_3p_4p_9p_{10},\ p_3p_5p_8p_{10}$ et $p_3p_5p_9p_{10}$. G_2 à son tour possède deux autres : $p_3p_4p_{11}p_{12}$ et $p_3p_5p_{11}p_{12}$.

Même si ces relations sont basées sur l'ensemble des trajectoires des E-tâche-graphes, elles ne seront pas évaluées à l'aide de la forme étendue des E-tâche-graphes[4], à cause d'un probable nombre exponentiel de trajectoires décrites par un E-tâche-graphe. Nous proposons donc d'analyser deux E-tâche-graphes en considérant leurs formes normales respectives à l'aide de la notion de *différence* étudiée dans la section suivante.

3.2 Différence entre des E-tâche-graphes

La notion de différence a pour but d'indiquer quelles modifications doivent être réalisées sur un E-tâche-graphe, sous forme normale, pour le transformer en un autre également sous forme normale. Ces modifications sont exprimées par des *ajouts* et des *retraits* de composants additifs. Un composant additif *ajouté* dans un E-tâche-graphe symbolise une (ou plusieurs) trajectoire(s) *de plus* représentée(s) par cet E-tâche-graphe (figure 3.3). Inversement, un composant additif *enlevé* d'un E-tâche-graphe symbolise une (ou plusieurs) trajectoire(s) *de moins* représentée(s) par cet E-tâche-graphe (figure 3.4). Par conséquent, les relations *plus général que* et *étroitement apparié* entre deux E-tâche-graphes peuvent être déterminées en fonction de l'expression de la différence entre leurs formes normales respectives, c'est-à-dire des *ajouts* et des *retraits* de composants additifs. La relation d'*équivalence* peut également être déterminée par la différence[5]. Dans la section 3.2.1 nous donnons la définition de ces relations entre

3. Dénomination inspirée de *close match*, un concept introduit par Levinson [Lev91] lors d'une étude sur la comparaison de graphes conceptuels.

4. Rappelons que l'interprétation de la forme étendue d'un E-tâche-graphe est l'*union de toutes ses trajectoires* (voir 2.6.3).

5. Pas d'ajouts de trajectoires, pas de retraits de trajectoires signifie pas de différence donc équivalence.

deux E-tâche-graphes en fonction de l'expression de la différence entre leurs formes normales respectives ; la section 3.2.2 est dédiée à la description de l'algorithme de différence.

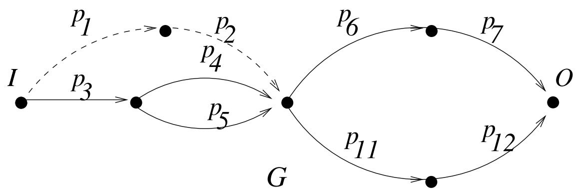


FIG. 3.3 – *Ajout d'arcs :* En ajoutant le composant fondamental p_1p_2 à G on décrit de plus les trajectoires $\{p_1p_2p_6p_7, p_1p_2p_{11}p_{12}\}$.

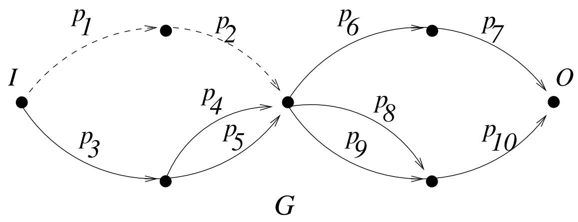


FIG. 3.4 – *Retrait d'arcs :* En enlevant le composant fondamental p_1p_2 de G on enlève les trajectoires $\{p_1p_2p_6p_7, p_1p_2p_8p_{10}, p_1p_2p_9p_{10}\}$.

3.2.1 Définition des relations en fonction de la différence entre formes normales

Notation : La différence entre deux E-tâche-graphes G_i et G_j est notée par $diff(G_i, G_j)$.

Soient G_i, G_j deux E-tâche-graphes et soient $n(G_i)$ et $n(G_j)$ leurs formes normales respectives.

Définition 32 *Plus général que* : $G_i \succeq G_j$ si $diff(n(G_i), n(G_j))$ n'est exprimée que par des *retraits*. Autrement dit, pour transformer $n(G_i)$ en $n(G_j)$ on doit *enlever* des trajectoires décrites *de plus* par $n(G_i)$.

Définition 33 *Plus spécifique que* : $G_i \preceq G_j$ si $diff(n(G_i), n(G_j))$ n'est exprimée que par des *ajouts*. Autrement dit, pour transformer $n(G_i)$ en $n(G_j)$ on doit *ajouter* des trajectoires *manquantes* en $n(G_i)$ par rapport à celles de $n(G_j)$.

Exemple 8 *Différence entre deux E-tâche-graphes G_1 et G_2, où $G_1 \succeq G_2$.*

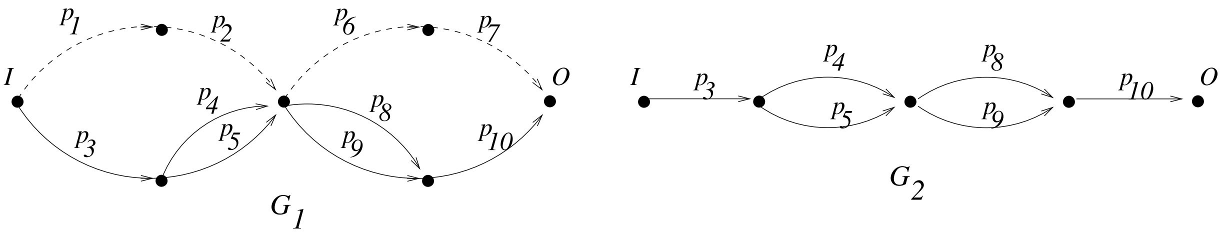


FIG. 3.5 – La différence entre G_1 est G_2 est exprimée par le *retrait* de $\{p_1p_2, p_6p_7\}$. Inversement, la différence entre G_2 est G_1 est exprimée par l'*ajout* de $\{p_1p_2, p_6p_7\}$.

Définition 34 *Étroitement apparié* : $G_i \approx_{\succeq} G_j$ si $diff(n(G_i), n(G_j))$ est exprimée par des *ajouts* **et** des *retraits*. Autrement dit, pour transformer $n(G_i)$ en $n(G_j)$ on doit *ajouter* des trajectoires *manquantes* en $n(G_i)$ par rapport à celles de $n(G_j)$ et *enlever* des trajectoires décrites *en plus* par $n(G_i)$.

Exemple 9 *Différence entre deux E-tâche-graphes G_1 et G_2, où $G_1 \approx_{\succeq} G_3$*

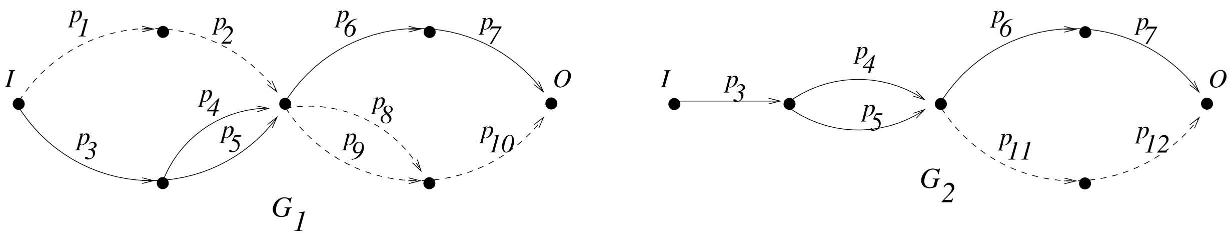


FIG. 3.6 – La différence entre G_1 est G_2 est exprimée à la fois par le *retrait* de $\{p_1p_2, (p_8 + p_9)p_{10}\}$ et l'*ajout* de $\{p_{11}p_{12}\}$.

3.2.2 Algorithme de différence

L'algorithme de différence entre deux E-tâche-graphes $G_i = (V_i, X_i, I_i, O_i, \lambda_i, \mathcal{L})$ et $G_j = (V_j, X_j, I_j, O_j, \lambda_j, \mathcal{L})$, G_i et G_j ***déterministes***, cherche à identifier une liste de trajectoires qui *manquent* à G_i ou qui *excèdent* dans G_i, par rapport à celles de G_j, pour les *ajouter* ou les *enlever*. Pour ce faire, on analyse les arcs sortant des sommets qui sont portées des sous-trajectoires communes aux deux E-tâche-graphes. Ci-dessous, les grandes lignes de l'algorithme de différence. Il est décrit en détails dans l'Annexe D.

1. On crée une liste de paires de sommets qui sont portées de sous-trajectoires communes aux deux E-tâche-graphes. On la crée initialement avec la paire (I_i, I_j). Puis, pour chaque paire (v_i, v_j) de la liste on vérifie les arcs sortant de v_i et v_j. Alors,

2. Pour chaque arc $x_i = (\overrightarrow{v_i, w_i})$ de X_i tel que **il existe** $x_j = (\overrightarrow{v_j, w_j})$ en X_j ayant la même étiquette de x_i, on *marque* x_j et on *inclut* la paire (w_i, w_j) dans la liste de paires de sommets à évaluer.
   • S'il existe dans la liste de paires de sommets à évaluer une paire (w_i, w_j') avec $w_j' \neq w_j$ (ce qui indique que deux arcs partant de v_i atteignent un même sommet w_i de G_i tandis que leurs correspondants en G_j atteignent des sommets distincts) alors on *éclate* w_i en w_i et w_i'.
3. Pour chaque arc $x_i = (\overrightarrow{v_i, w_i})$ de X_i tel que **il n'existe pas** d'arc $x_j = (\overrightarrow{v_j, w_j})$ en X_j ayant la même étiquette de x_i, on *enlève* de G_i le plus grand sous-E-tâche-graphe G_i' de G_i qui ne déconnecte pas le sommet v_i de O_i. On enlève ce sous-graphe car dans ce cas là aucune trajectoire de G_j ne possède $\tau_j' = (I_j \dots v_j w_j) = (I_i \dots v_i w_i)$ comme sous-trajectoire, vu qu'il n'y a pas d'arcs partant de v_j tel que $\lambda_j(\overrightarrow{v_j, w_j}) = \lambda_i(\overrightarrow{v_i, w_i})$.
   • Si cela n'est pas possible, c'est-à-dire si le moindre enlèvement déconnecte v_i de O_i alors $G_i \neq G_j$ car dans ce cas là, G_i et G_j ne possèdent aucune trajectoire commune.
4. S'il existe $x_j = (\overrightarrow{v_j, w_j})$ de X_j *non marqué* (c'est-à-dire, il n'y a pas d'arcs $x_i = (\overrightarrow{v_i, w_i})$ avec la même étiquette de x_j) alors *ajouter* dans G_i, le sous-E-tâche-graphe $G_j' = (V_j', X_j', I_j', O_j', \lambda_j, \mathcal{L})$ de G_j où $I_j' = w_j$, $O_j' = w_j' \in V_j$ tel que w_j' est le premier sommet, atteint par les (sous-)trajectoires passant par x_j, qui peut également être atteint par d'autres (sous-)trajectoires qui ne passent pas par x_j. Si $w_j' = O_j$, on identifie O_j' avec O_i ; sinon, il s'agit d'une sous-trajectoire et on pose $w_i' = O_j'$; dans ce cas là w_i' n'est pas encore connecté à O_i. On inclut donc la paire (w_i', w_j') dans la liste de paires de sommets à évaluer dans le but d'identifier avec quel sommet il faut le fusionner (item 5).
5. Si dans la liste de paires de sommets à évaluer, il existe deux paires (w_i, w_j) et (w_i', w_j') telles que $w_i \neq w_i'$ et $w_j = w_j'$ (ce qui signifie que deux sous-trajectoires de G_i ont des portées différentes tandis que leurs correspondantes en G_j ont la même portée) alors on *fusionne* w_i et w_i'.
   • On note que lorsque l'on fusionne deux sommets, il se peut que des nouveaux chemins entre w_i et O_i et/ou entre w_i' et O_i se créent. Cependant, si après une telle fusion de sommets *aucune* sous-trajectoire n'est enlevée, cela veut dire que les chemins crées *manquaient* effectivement à G_i.
6. Après avoir analysé le sommet v_i, on *enlève* la paire (v_i, v_j) de la liste de paires de sommets à évaluer.
7. L'algorithme s'arrête lorsque la liste de sommets à évaluer est vide *ou* lorsque l'on a identifié que les deux E-tâche-graphes sont différents (voir item 3).

Cet algorithme ne doit être utilisé que pour comparer deux E-tâche-graphes sous forme normale déterministe car dans un E-tâche-graphe non déterministe, il se peut que l'on fusionne des sommets (item 5 de l'algorithme) appartenant à des composants additifs différents. Un telle fusion produirait un digraphe qui n'est pas un E-tâche-graphe (figure 3.7), puisqu'il possède un sous-graphe interdit aux graphes série-parallèle et un E-tâche-graphe est un graphe série-parallèle (voir Annexe B). Cependant, lorsqu'une des formes normales à comparer est non déterministe, d'après la propriété 13 il est toujours possible de trouver un autre E-tâche-graphe, celui-ci déterministe et également sous forme normale, équivalent au premier. Alors, si l'on doit comparer deux E-tâche-graphes G_i et G_j sous forme normale, tels que G_i est *non déterministe* et G_j *déterministe* (ou vice-versa), il suffit de trouver l'E-tâche-graphe déterministe sous forme normale G_i' équivalent à G_i et le comparer G_j. La relation existante entre G_i' et G_j exprime la relation entre G_i et G_j, car si $G_i \sim G_i' \succeq G_j \iff G_i \succeq G_j$ (où $\sim$ symbolise une relation d'équivalence).

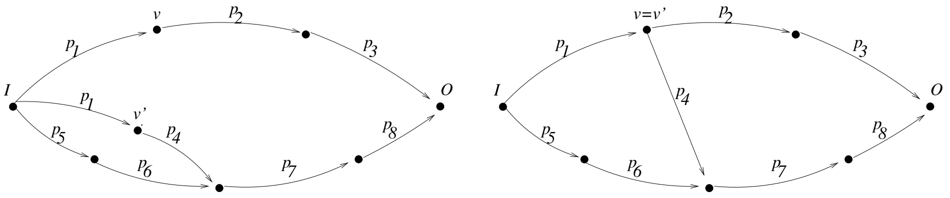


FIG. 3.7 – *Application de l'algorithme de différence à des E-tâche-graphes non déterministe :* On devrait fusionner v et v', de l'E-tâche-graphe à gauche, qui appartiennent à des composants additifs différents. Cette fusion aurait donné comme résultat le digraphe à droite, qui possède un sous-graphe homeomorphe au digraphe W, un sous-graphe interdit aux graphes série-parallèle (voir discussion, section 2.1.)

3.3 Complexité de la comparaison de deux E-tâche-graphes

Nous avons dit précédemment que la comparaison de deux E-tâche-graphes sous leurs formes étendues respectives pourrait être intéressante hormis le fait que, pour des E-tâche-graphes très complexes le nombre de trajectoires décrites pourrait être de l'ordre exponentiel. Dans la section précédente nous avons pu identifier des trajectoires qui *manquent* et/ou qui *excèdent* dans un E-tâche-graphe par rapport à un autre E-tâche-graphe, par le moyen d'*ajouts* et de *retraits* de *composants additifs* de leurs formes normales respectives.

La complexité de la comparaison de deux E-tâche-graphes, $G_i = (V_i, X_i, I_i, O_i, \lambda_i, \mathcal{L})$ et $G_j = (V_j, X_j, I_j, O_j, \lambda_j, \mathcal{L})$, en utilisant leurs formes normales respectives correspond à la somme de la complexité[6] de déterminer la forme normale de chacun des deux E-tâche-graphes (***normal*** (G) est en $\mathcal{O}(|X|^4)$) et celle de la différence entre deux E-tâche-graphes sous forme normale (***différence*** (G_i, G_j) est en $\mathcal{O}(max\{|X_i|^2,\ \ |X_j|^2\})$). La comparaison de deux E-tâche-graphes en utilisant leurs formes normales respectives est donc en $\mathcal{O}(max\{|X_i|^4, |X_j|^4\})$.

Toutefois, pour des E-tâche-graphes relativement "petits", la comparaison de leurs formes étendues respectives peut être encore intéressante, voire même plus intéressante que celle de leurs formes normales.

3.4 Comparaison d'E-tâche-graphe *vs* comparaison d'arbres de décomposition

Rappelons qu'un E-tâche-graphe correspond à la forme canonique d'un arbre de décomposition. Ainsi, deux arbres de décomposition qui possèdent la même forme canonique correspondent au même E-tâche-graphe. En effet, les niveaux intermédiaires d'un arbre (qui représentent des sous-tâches complexes) ne sont pas représentés dans un E-tâche-graphe. De plus, si $T = (V_T, X_T, r_T)$ est un arbre de décomposition d'une tâche et $T_c = (V_c, X_c, r_c)$ est la forme canonique de T, alors $|X_c| \leq |X_T|$.

6. Pour l'analyse de la complexité de chacun des algorithmes, se reporter à l'Annexe E.

Par ailleurs, l'E-tâche-graphe $G = (V, X, I, O, \lambda, \mathcal{L})$ correspondant à T est tel que $|X| \leq |V_c| - 1$, puisque les arcs d'un E-tâche-graphe correspondent aux feuilles de l'arbre. Or, $|V_c| = |X_c| + 1$. Par conséquent, $|X| \leq |X_c|$.

La complexité de la comparaison de deux arbres de décomposition sous leurs formes canoniques respectives correspond à la somme de la complexité de trouver les formes canoniques ($\mathcal{O}(max\{|X_{T_i}|, |X_{T_j}|\})$) et celle de les comparer ($\mathcal{O}(max\{|X_{c_i}|^2, |X_{c_j}|^2\})$). Elle est donc en ($\mathcal{O}(max\{|X_{T_i}|^2, |X_{T_j}|^2\})$).

Dans la section précédente nous avons démontré que la comparaison de deux E-tâche-graphes sous leurs formes normales respectives est en $\mathcal{O}(max\{|X_i|^4, |X_j|^4\})$. Ainsi, la comparaison des arbres de décomposition se montre plus intéressante.

Néanmoins, si deux arbres de décomposition ne possèdent pas la même forme canonique, leur comparaison **ne permet pas** de dire que les stratégies qu'ils représentent sont équivalentes, même si c'est le cas. Nous nous sommes justement fixé le but de comparer deux stratégies lorsque la comparaison de leurs arbres de décomposition respectifs n'est pas suffisante. Deux arbres qui possèdent la même forme canonique représentent en réalité des stratégies identiques. La distinction entre elles est le niveau de détail de chaque étape. Par contre, deux arbres possédant des formes canoniques différentes, qui pourtant décrivent le même ensemble des plans d'actions, représentent des raisonnements différents mais équivalents (comme l'illustre la figure 3.8).

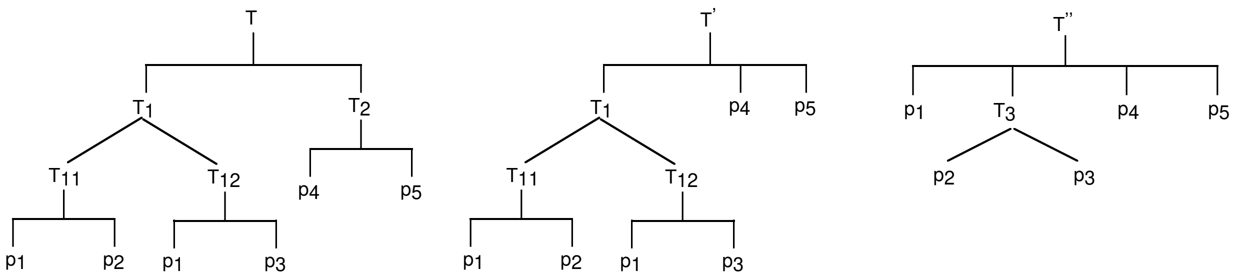


FIG. 3.8 – La distinction entre les arbres T et T' est que dans le premier, l'exécution des programmes p_4 et p_5 ont été regroupées dans une unique étape (complexe) de la stratégie tandis que dans le deuxième, ces deux programmes ont été considérés, dès le départ, comme deux étapes élémentaires à exécuter. Ces deux arbres possèdent la même forme canonique (qui d'ailleurs correspond à T'). Par contre, T'' représente une façon différente de raisonner.

Dans la section suivante nous présentons un exemple d'un cas plus concret. Nous illustrons l'utilisation des E-tâche-graphes pour la comparaison de la stratégie de deux tâches par leur application sur MYOSYS. Ce dernier est un système d'aide au diagnostic des maladies neuromusculaires développé en utilisant SCARP [Wil94], l'environnement générique coopératif de résolution de problèmes, basé sur le modèle de tâches, développé au sein du projet SHERPA.

3.5 Exemple d'application

Dans cette section, nous présentons dans un premier temps le système MYOSYS (3.5.1) et ensuite nous montrons un exemple de comparaison de la structure de deux tâches (3.5.2).

3.5.1 MYOSYS

MYOSYS [Zie93, Mor96] est un système d'aide au diagnostic des maladies neuro-musculaires réalisé en collaboration avec le service d'électromyographie du CHU de Grenoble. MYOSYS a été développé sous SCARP, un environnement générique coopératif de résolution de problèmes basé sur le modèle de tâches. SCARP utilise la représentation de connaissances par objets.

L'électromyographie est une discipline médicale spécialisée à l'intérieur de la neurologie qui, à travers des mesures de diverses caractéristiques électriques dans les muscles et les nerfs, permet d'établir un diagnostic des maladies neuro-musculaires. L'électromyographiste commence toujours son examen avec une évaluation des symptômes du patient. Il se sert également des informations cliniques du patient, telles que le niveau de glycémie, ainsi que de son historique comme un traumatisme ou des facteurs héréditaires. Ces informations guident le médecin praticien vers un certain indice de maladie qu'il vérifiera à travers un ensemble d'examens électromyographiques qui composent un protocole initial. Ces examens permettent d'une part de confirmer une hypothèse de maladie et d'autre part d'exclure d'autres maladies. Finalement, le médecin confronte les résultats de ce protocole à l'hypothèse de départ. Si la première hypothèse n'est pas valable, le médecin doit évaluer une deuxième hypothèse. Ce processus est réitéré jusqu'à ce qu'un diagnostic soit établi.

MYOSYS est divisé en deux étapes : (i) dans un premier temps, à partir des *informations* données par un patient, il fournit un ensemble d'*hypothèses* de maladies, classées selon un critère donné par les médecins ; (ii) dans un deuxième temps, le médecin choisit une des hypothèses (celle qu'il trouve la plus indiquée pour le cas traité) et le système établit donc un *protocole d'examen* pour valider, ou non, cette hypothèse. Nous allons nous intéresser à la première phase, l'*évocation des hypothèses*, dont la description de la tâche qui la représente est illustrée dans la figure 3.9.

Les connaissances manipulées par MYOSYS (c'est-à-dire les connaissances associées aux maladies neuro-musculaires, telles que les symptômes et leur localisation ainsi que les maladies elles-mêmes) sont divisées en trois hiérarchies de classes :

information : c'est la hiérarchie de classe qui contient tous les types d'informations lesquelles peuvent être données par le patient et qui sont importantes pour établir une (ou plusieurs) hypothèse(s) de maladies. Le premier niveau de cette hiérarchie est composé de trois sous-classes :

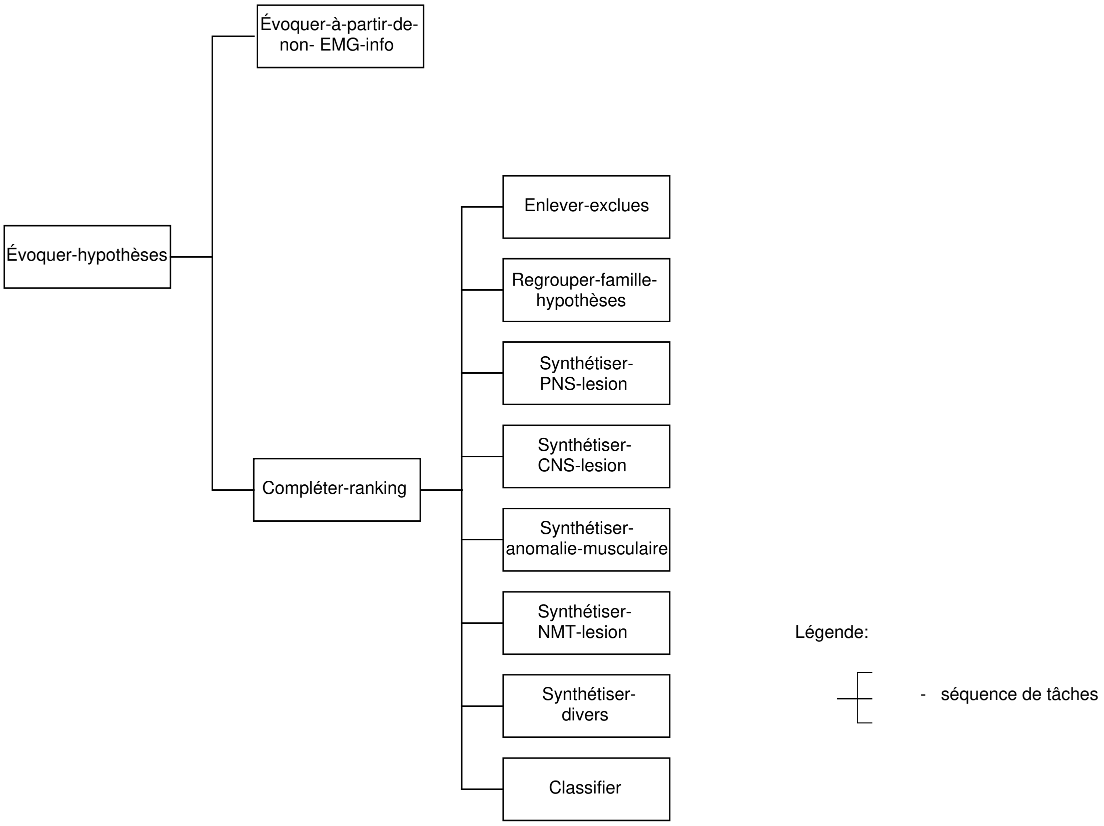


FIG. 3.9 – *Description partielle de la tâche* **Évoquer-hypothèses-EMG** *qui représente la première phase de MYOSYS.*

• *symptômes*, qui représente des symptômes qu'un patient peut présenter, tel que douleur (localisée ou diffuse), crampe, etc... ;

• *clinique-biologie*, qui représente des facteurs cliniques et/ou biologiques, tel que la glycémie, lesquels peuvent être importants pour établir une hypothèse de maladie ; et

• *facteurs de risque*, qui représente l'historique du patient et/ou des facteurs héréditaires.

À chaque sous-classe d'*information* sont associées des maladies qui peuvent être évoquées par la donnée qu'elle représente avec plusieurs niveaux de certitude : *possible, probable* ou *certaine*. De même, pour des maladies qui doivent être exclues par cette information.

localisation : quelques unes des informations nécessitent d'avoir sa localisation précisée ; par exemple une douleur (*symptôme*) ou un plâtre (*facteur de risque*). Le premier niveau de cette hiérarchie de classe est composé de deux sous-classes :

• *neuro-anatomie*, qui représente l'organisation des nerfs et des muscles ; et

• *topographie*, qui associe un nerf ou un muscle à une région topographique du corps, par exemple avant-bras ou coude.

hypothèse : c'est la hiérarchie de classe qui contient la description des maladies neuro-musculaires.

Comme illustre la figure 3.9, la tâche *Évoquer-hypothèses* est composée de deux sous-tâches : *Évoquer-à-partir-de-non-EMG* et *Compléter-ranking*. La première, à partir des informations des patients et les connaissances existant dans la base, "génère des hypothèses plausibles" ; la deuxième se décompose en plusieurs étapes : d'abord elle "sélectionne des hypothèses" en enlevant des exclues selon des critères spécifiques ; ensuite elle les regroupe en six différents groupes ; et finalement elle les classe (dans un ordre établi). *Évoquer-hypothèses* est donc une tâche qui suit la méthode d'*assemblage abductive* (*cf* figure 1.10, section 1.4) où l'étape de sélection d'hypothèses a été enrichie d'une classification, un besoin de l'application.

Analyse de la tâche *Évoquer-à-partir-de-non-EMG-info*

Nous allons maintenant analyser complètement (c'est-à-dire jusqu'aux tâches élémentaires) la tâche *Évoquer-à-partir-de-non-EMG-info*, dont la description est illustrée par la figure 3.10.

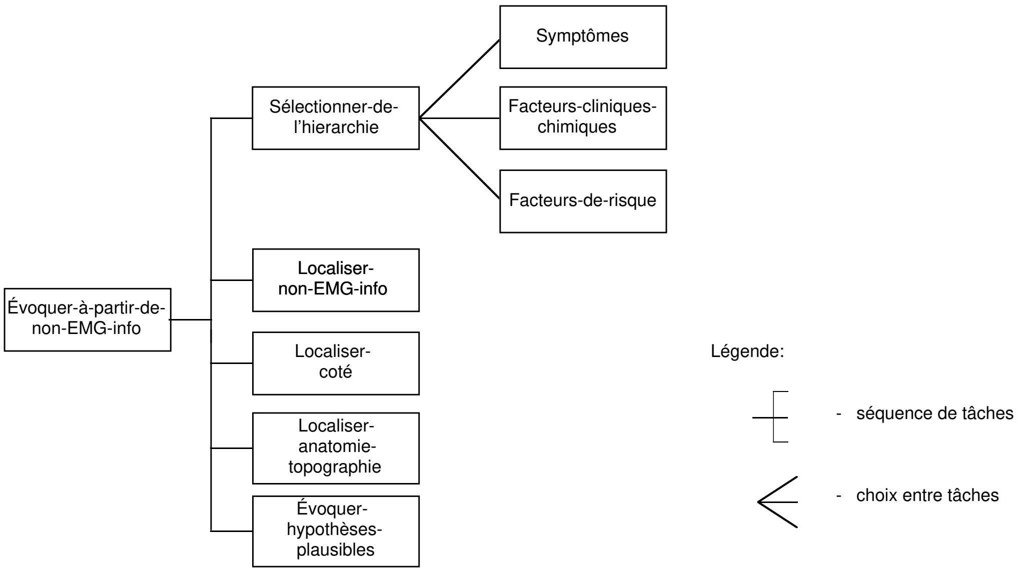


FIG. 3.10 – *Description de la tâche Évoquer-à-partir-de-non-EMG-info.*

La tâche *Évoquer-à-partir-de-non-EMG-info* est composée de cinq étapes: la première, *Sélectionner-de-la-hiérarchie*, permet au médecin d'indiquer l'information donnée par le patient. Pour ce faire, le médecin doit signaler le type d'information (*symptôme, clinique-chimique* ou *facteur de risque*) et sélectionner dans la hiérarchie choisie l'information à indiquer ; les trois suivantes (*Localiser-non-EMG-info, Localiser-coté* et *Localiser-anatomie-topographie*) permettent de bien préciser la localisation des informations données ; finalement, la dernière, *Évoquer-hypothèses-plausibles*, fait l'évocation proprement dite des hypothèses. La première étape ainsi que les étapes de localisation sont représentées par des tâches élémentaires. Par contre, la dernière est représentée par une tâche complexe, *cf* la figure 3.11.

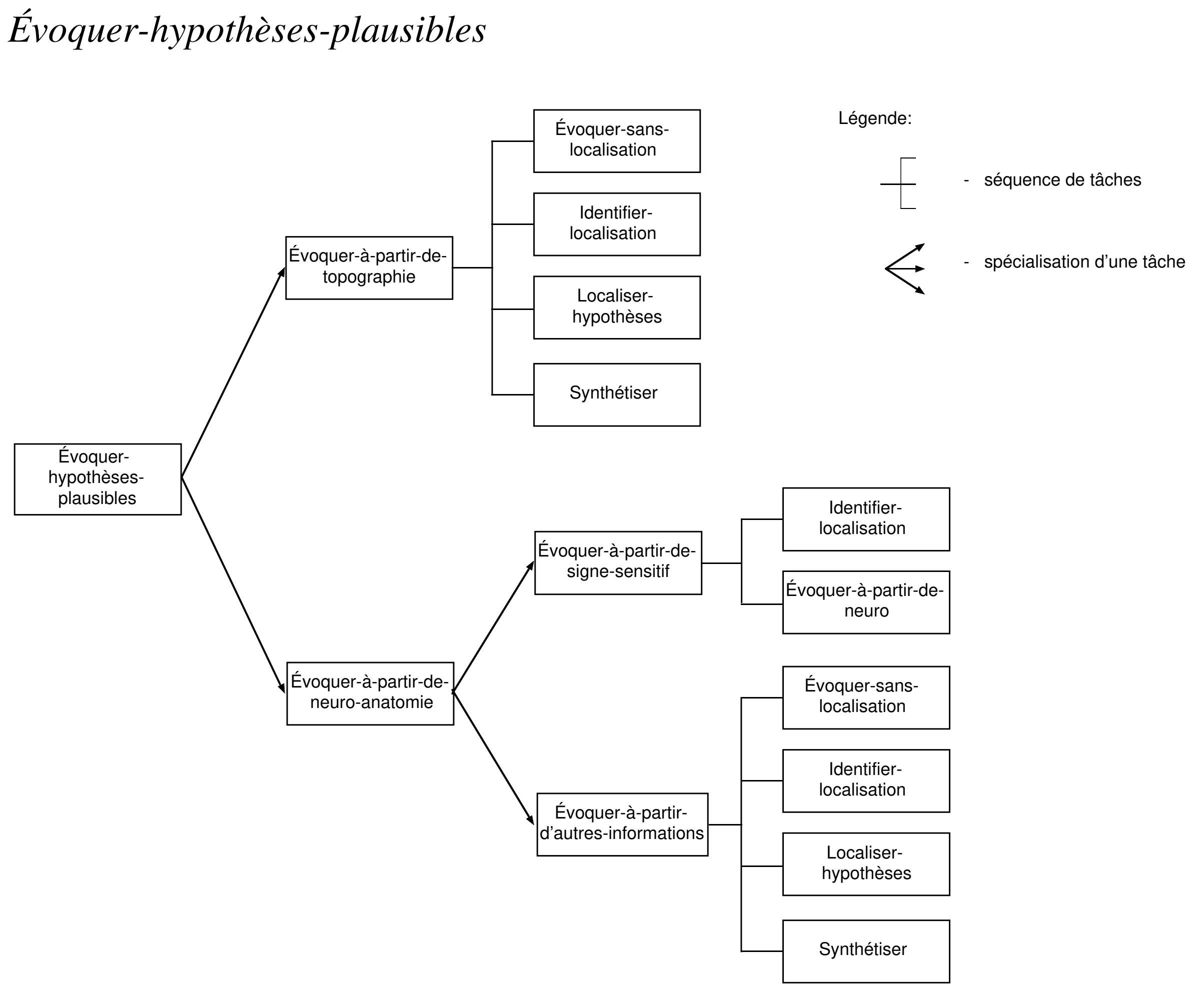


FIG. 3.11 – *Description de la tâche Évoquer-hypothèses-plausibles.*

Évoquer-hypothèses-plausibles est spécialisée en fonction de la localisation des informations données, *topographie* ou *neuro-anatomie* :

i) Si la localisation est une *topographie*, Évoquer-hypothèses-plausibles se décompose en quatre étapes : la première, *Évoquer-sans-localisation*, évoque les hypothèses de maladie indépendamment de la localisation associée à l'information donnée ; la deuxième, *Identifier-localisation* indique les éléments anatomiques qui peuvent être atteints par les maladies évoquées ; la troisième, *Localiser-hypotèses*, précise la localisation des hypothèses ; finalement, la dernière, *Synthétiser*, synthétise les informations collectées.

ii) Si la localisation est une *neuro-anatomie*, un traitement spécial est donnée à des symptômes du type *signe sensitif* : l'hypothèse concernant cette neuro-anatomie est évoquée comme *définitive*, les hypothèses concernant les neuro-anatomies directement liées à celle informée sont évoquées comme *probables*. La tâche se décompose en deux étapes : *Identifier-localisation* et *Évoquer-à-partir-de-neuro*. Pour tous les autres types d'informations, Évoquer-hypothèses-plausibles suit le schéma utilisé pour les topographies.

Les étapes de chacune de ces spécialisations sont représentées par des tâches élémentaires. Cependant, la tâche *Identifier-localisation* est spécialisée en fonction de la localisation associée à l'*information* donnée. La hiérarchie complète de décomposition de la tâche *Évoquer-hypothèses-plausibles* est illustrée dans la figure 3.12.

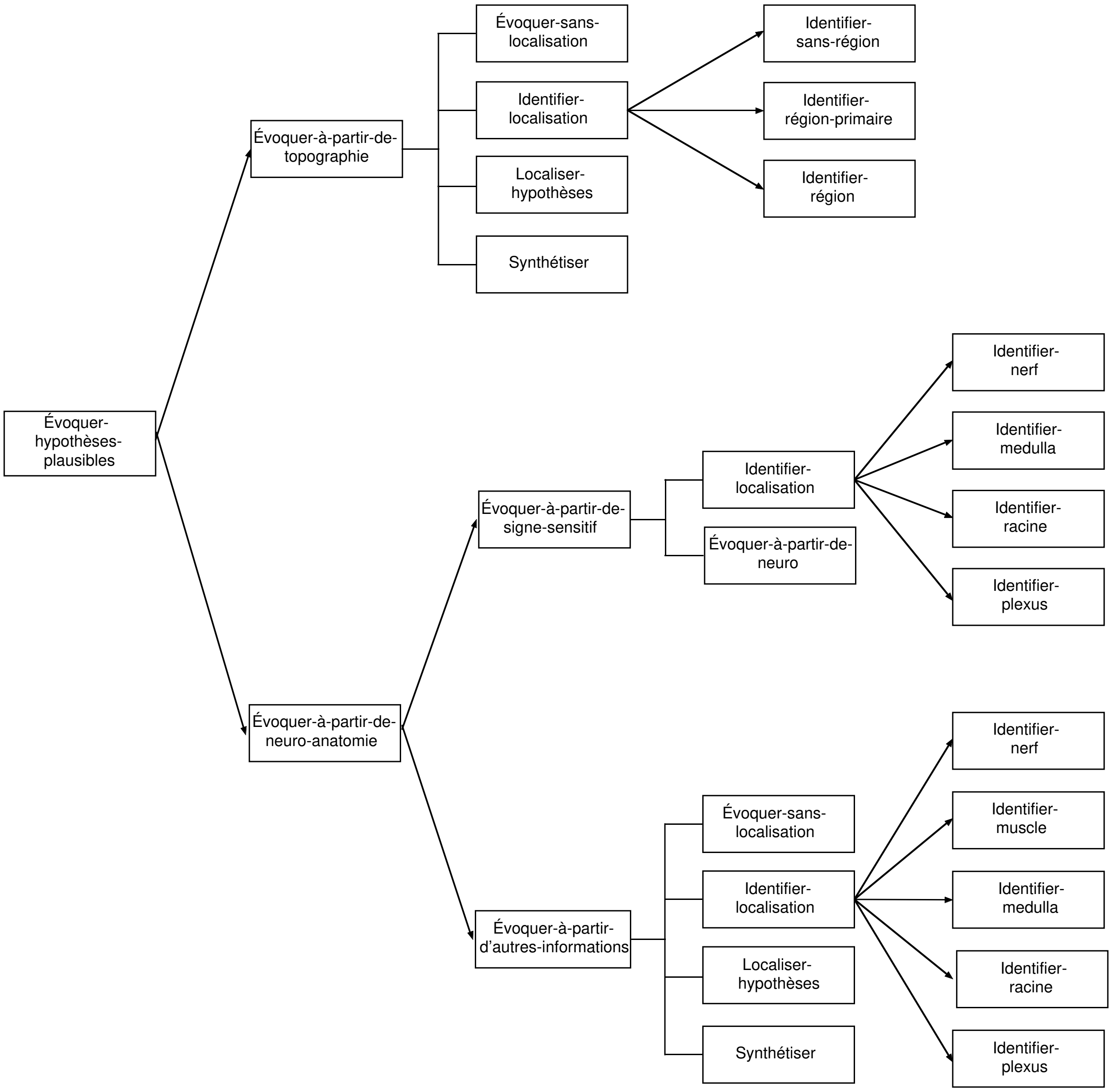


FIG. 3.12 – *Description complète de la tâche Évoquer-hypothèses-plausibles.*

Une autre façon de décrire la tâche Évoquer-hypothèses-plausibles

Vu que seule l'évocation d'hypothèses de maladie concernant des informations de signes sensitifs dont la localisation est une neuro-anatomie possède un traitement spécial, nous proposons de décrire la tâche *Évoquer-hypothèses-plausibles* d'une autre façon, en utilisant les mêmes tâches élémentaires celles-ci réalisées par les mêmes programmes respectifs (*cf* la figure 3.13).

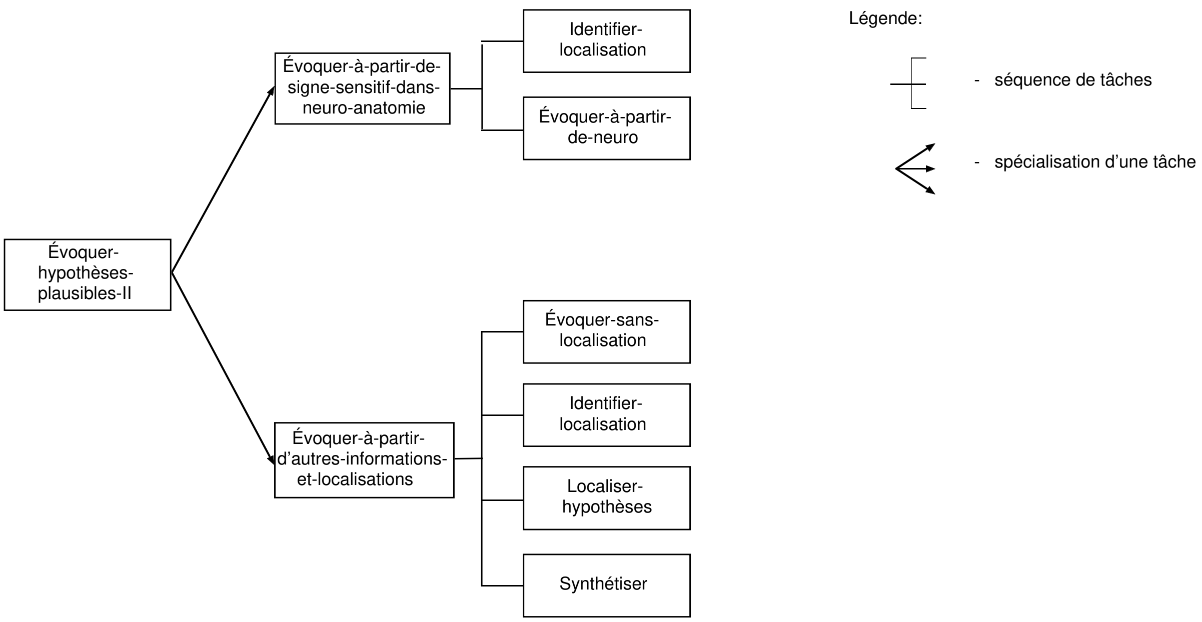


FIG. 3.13 – *Évoquer-hypothèses-plausibles-II:* Une autre façon de décrire la tâche Évoquer-hypothèses-plausibles.

Dans cette spécification, la tâche *Évoquer-hypothèses-plausibles-II* indique le traitement spécial donné aux informations de signes sensitifs localisées dans des neuro-anatomies et décrit le traitement commun donné aux autres informations. La distinction entre ces deux tâches est que la première évolue d'abord en fonction de la localisation du symptôme et ensuite en fonction du type du symptôme (signe-sensitif ou d'autres) tandis que la deuxième distingue simplement le cas spécial d'un *signe sensitif localisé dans une neuro-anatomie* des autres cas. La hiérarchie complète de décomposition de la tâche *Évoquer-hypothèses-plausibles-II* est illustrée en figure 3.14.

Par construction, les deux tâches sont équivalentes. En effet, pour chaque cas précis, le plan d'actions à réaliser reste le même pour les deux tâches. Par exemple, supposons qu'un patient se plaigne d'une douleur localisée au coude gauche. L'évocation d'hypothèses de maladie liées à ce symptôme suit, dans les deux tâches, le plan (*Évoquer-sans-localisation, Identifier-*

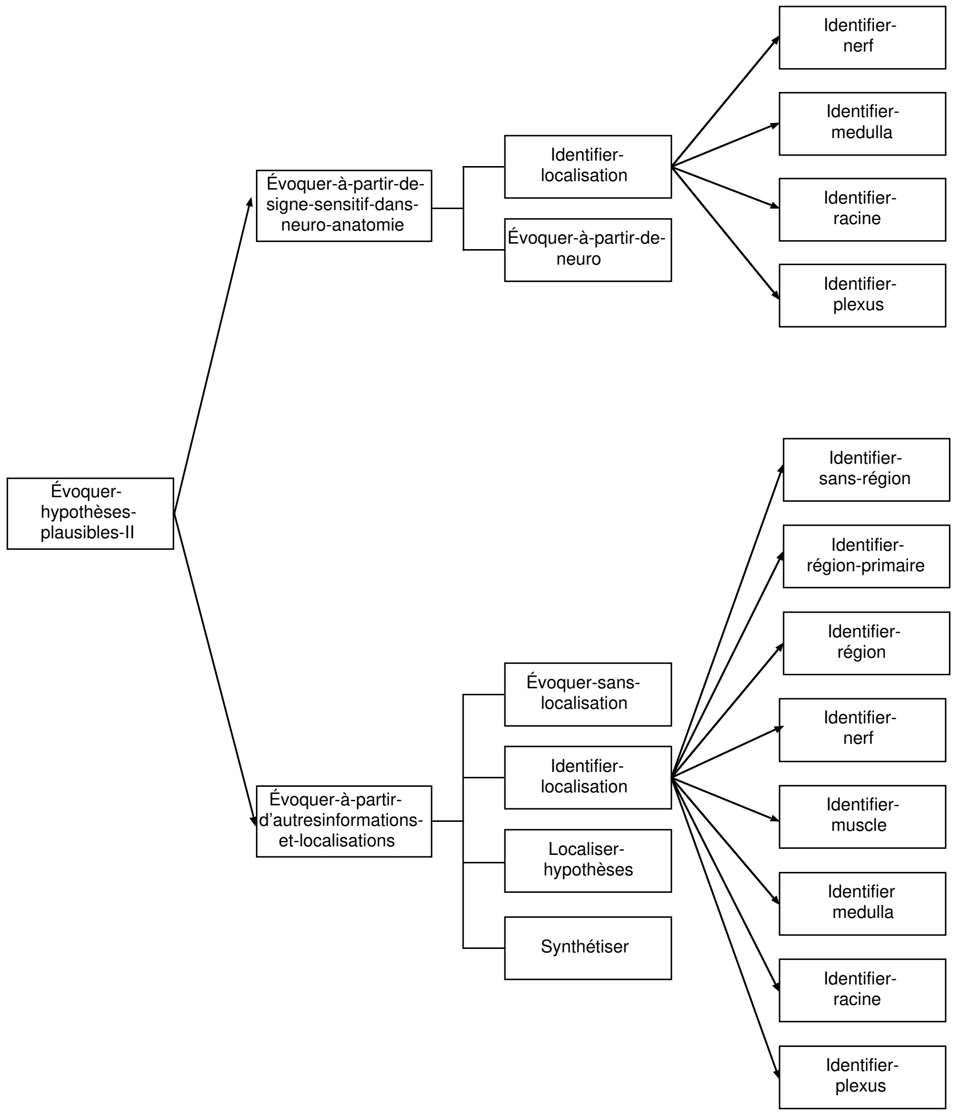


FIG. 3.14 – *Description complète de la tâche Évoquer-hypothèses-plausibles-II.*

région, Localiser-hypothèse, Synthétiser) vu que le coude est classé dans les topographies [7]. Par contre, pour une plainte d'un fourmillement à l'intérieur de la cuisse, interprétée par le médecin comme "une paresthésie (*signe sensitif*) qui atteint le nerf fémoral (*neuro-anatomie*)", l'évocation d'hypothèses de maladie dans ce cas là suit, dans les deux tâches, le plan (*Identifier-nerf, Évoquer-à-partir-de-neuro*). La tâche *Évoquer-hypothèses-plausibles* a d'abord été spécialisée en *Évoquer-à-partir-de-neuro-anatomie* (le fémoral) et cette dernière en *Évoquer-à-partir-de-signe-sensitif* (la paresthésie). *Évoquer-hypothèses-plausibles-II* a été à son tour spécialisée directement en *Évoquer-à-partir-de-signe-sensitif-dans-neuro-anatomie*.

7. Le coude est classé, plus précisément, dans la sous-classe *topographie-motrice*; *Identifier-région* étant la spécialisation de la tâche *Identifier-localisation* associée à cette sous-classe de la localisation.

3.5.2 Comparaison des deux tâches d'évocation d'hypothèses plausibles

Les arbres ET/OU qui représentent les stratégies indiquées par les tâches *Évoquer-hypothèses-plausibles* (telle qu'elle est décrite dans MYOSYS) et *Évoquer-hypothèses-plausibles-II* (notre proposition) sont illustrés respectivement dans les figures 3.15 et 3.16.

Lorsque l'on compare leurs formes canoniques respectives, on vérifie qu'ils ne sont pas isomorphes et, par conséquent, on aurait conclu que les deux tâches ne sont pas équivalentes même si en réalité elles le sont, par construction.

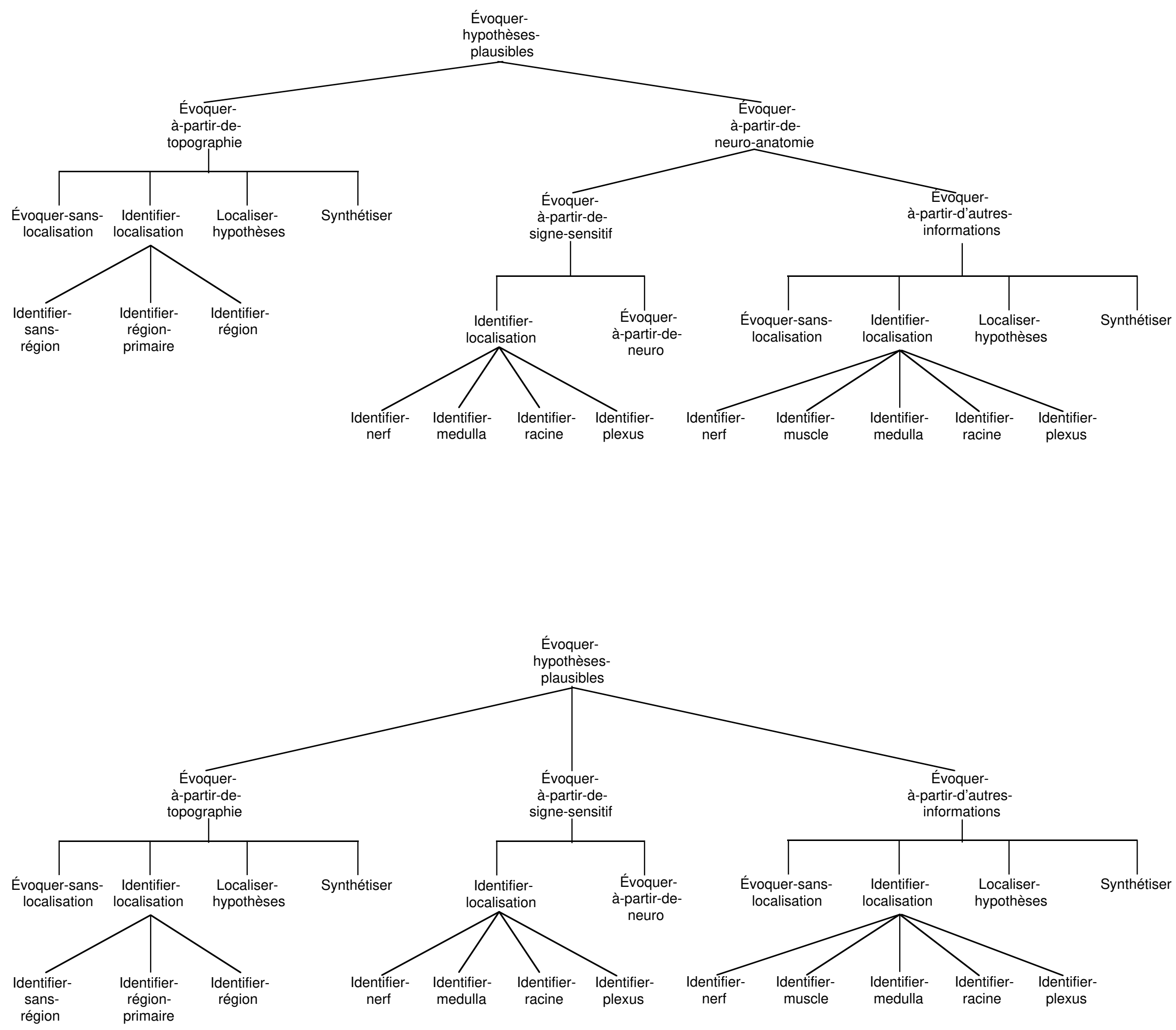


FIG. 3.15 – *Arbre ET/OU (et sa forme canonique) qui représente la tâche Évoquer-hypothèses-plausibles.*

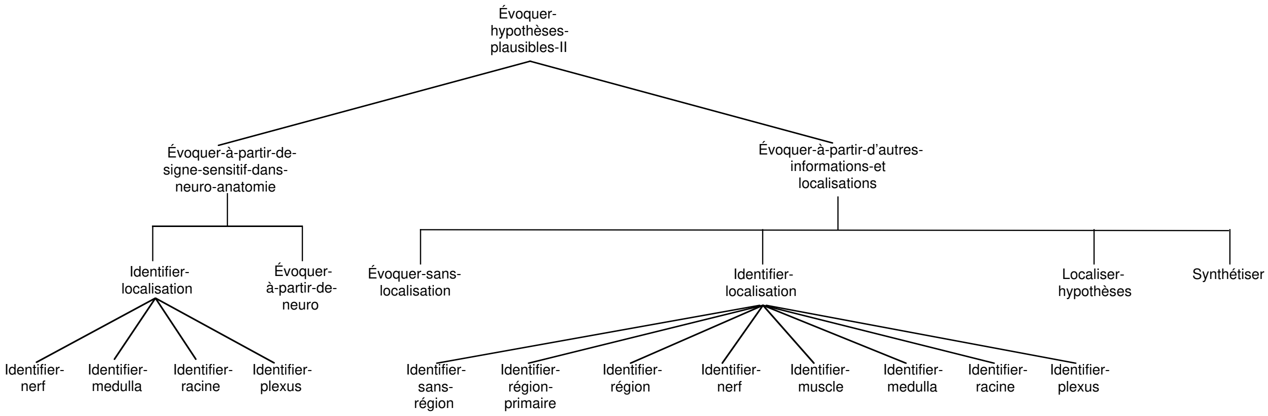
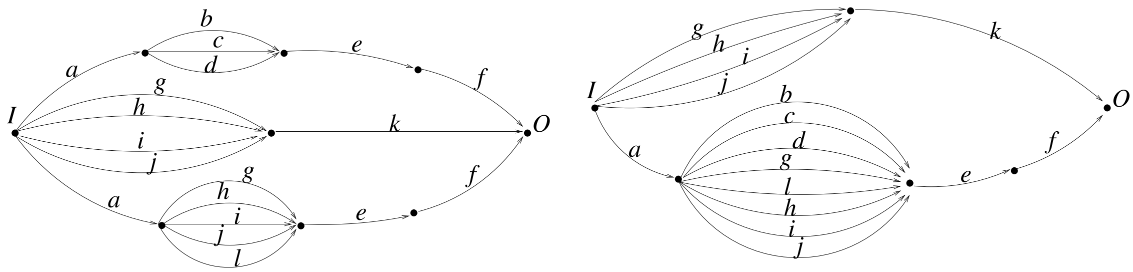
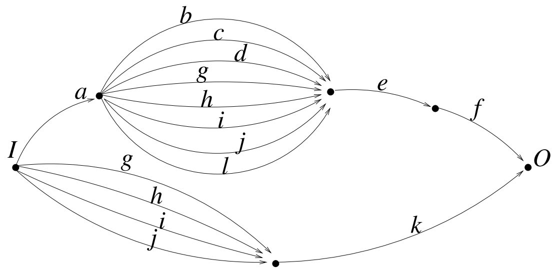


FIG. 3.16 – *Arbre ET/OU qui représente la tâche Évoquer-hypothèses-plausibles-II.* Il est déjà sous une forme canonique.

Par contre, si l'on analyse les E-tâche-graphes correspondants (figures 3.17) on note que leurs formes normales respectives sont identiques.





Légende :
$a \rightarrow$ Évoquer-sans-localisation
$b \rightarrow$ Identifier-sans-région
$c \rightarrow$ Identifier-région-primaire
$d \rightarrow$ Identifier-région
$e \rightarrow$ Localiser-hypothèses
$f \rightarrow$ Synthétiser $g \rightarrow$ Identifier-nerf
$h \rightarrow$ Identifier-medulla
$i \rightarrow$ Identifier-racine
$j \rightarrow$ Identifier-plexus
$k \rightarrow$ Évoquer-à-partir-de-neuro
$l \rightarrow$ Identifier-muscle

FIG. 3.17 – En haut, à gauche, l'E-tâche-graphe qui représente la stratégie indiquée par la tâche *Évoquer hypothèses-plausibles* ; à droite, celui qui représente la stratégie indiquée par la tâche *'Evoquer hypothèses-plausibles-II*. En bas la forme normale des deux E-tâche-graphes.

En conclusion, par l'utilisation des E-tâche-graphes nous avons pu déterminer que les deux tâches sont effectivement équivalentes tandis que par la vérification d'un isomorphisme d'arbres ET/OU de décomposition **nous n'étions pas capable** d'identifier cette équivalence.

Conclusions et Perspectives

L'objectif de ce travail était de comparer des stratégies de résolution de problèmes, existantes dans un Environnement de Résolution de Problèmes, modélisées par des tâches. L'une des principales caractéristiques d'un modèle de tâche est la description d'une stratégie pour résoudre un problème laquelle se base sur la décomposition récursive du problème en sous-problèmes jusqu'à arriver à des problèmes dont la résolution est simple et/ou connue, ce qui conduit à une hiérarchisation de problèmes. Une telle hiérarchie peut donc être représentée par un arbre de décomposition. La simple vérification d'un isomorphisme d'arbres de décomposition de problèmes ne s'est pas montrée complètement efficace puisqu'elle ne permet de vérifier l'équivalence entre deux stratégies que pour un cas particulier : le cas où les stratégies se distinguent simplement par rapport au niveau de détails donné à chaque étape de la décomposition. Dans ce cas là, il est possible de trouver une forme canonique pour les arbres de décomposition de ces stratégies [Ham93]. Ces arbres auront des formes canoniques identiques ou isomorphes.

Nous avons dans ce travail étendu la possibilité d'identifier que deux stratégies sont équivalentes, même si elles sont représentées par des arbres de décomposition non isomorphes. Cette situation se présente lorsque deux stratégies décrivent le même ensemble de plans d'actions en utilisant des raisonnements différents. Pour ce faire, nous avons proposé un modèle de représentation de la stratégie d'une tâche, dénommé *E-tâche-graphe*.

Les E-tâche-graphes représentent la façon dont des actions élémentaires peuvent être intégrées pour résoudre un classe de problème donnée. Ils ne s'occupent pas de la hiérarchie de décomposition du problème en sous-problèmes. Ainsi, différentes façons d'analyser un problème qui, finalement, résultent en un même ensemble de séquences d'étapes directement exécutables, sont identifiées. Nous avons illustré avec un exemple de deux stratégies pour résoudre le problème du diagnostic en électromyographie.

La comparaison d'E-tâche-graphes ne se fait pas par le test d'isomorphisme de graphes puisque d'une part ce problème est NP-complet et d'autre part des E-tâche-graphes non isomorphes peuvent également représenter des stratégies équivalentes. Nous avons recouru à un *système de réécriture conditionnel avec priorité* qui permet de trouver une forme normale pour des E-tâche-graphes équivalents. La comparaison d'E-tâche-graphes se fait donc par la *différence* entre deux E-tâche-graphes sous leurs formes normales respectives (étudié dans le chapitre 3). Cette comparaison est en $\mathcal{O}(max\{|X_i|^4, |X_j|^4\})$; X_i et X_j étant le nombre d'arcs de chacun des E-tâche-graphes comparés. Ce coût est plutôt dû à la recherche de la forme normale d'un E-tâche-graphe. L'algorithme de différence, lui-même, est en $\mathcal{O}(max\{|X_i|^3, |X_j|^3\})$. La comparaison d'arbres de décomposition reste encore plus performante, pour le cas auquel elle s'applique.

La différence entre deux E-tâche-graphes permet également d'identifier si une stratégie est *plus générale* (ou *plus spécifique*) que l'autre. Autrement dit, la différence permet d'identifier si l'ensemble des plans d'actions décrit par une stratégie est inclus dans celui décrit par une deuxième. Cela caractérise une relation d'ordre partiel entre stratégies, dénommée *plus général que*. Une relation, dénommée *étroitement apparié*, qui permet de reconnaître les plans communs à deux stratégies non comparables par la relation "plus général que", peut également être identifiée en utilisant la différence entre des E-tâche-graphes.

Ce travail a été une étude théorique des E-tâche-graphes en tant que modèle de représentation d'une stratégie de résolution de problèmes représentée par une tâche. Il ouvre deux perspectives d'utilisation par des ERP :

i) Dans l'immédiat, un module de comparaison de tâches peut être introduit dans un ERP distribué pour rendre possible la création d'un annuaire qui indiquera sur quels sites il existe des tâches lesquelles permettent de résoudre une même classe de problèmes. Effectivement, lors de la création d'une nouvelle tâche dans l'ERP, si l'on identifie quelles sont les tâches, existants dans tous les sites concernés, qui indiquent des stratégies représentant le même ensemble de plans d'actions, on pourra mettre à jour l'annuaire. Ainsi, soit par exemple un cas de figure où une tâche complexe **T** est définie dans un site s_1 et l'une des sous-tâches (**T**′) qui la composent est définie dans un site s_2. Supposons que le site s_2 soit déconnecté temporairement du réseau. Si le système possède un annuaire de tâches équivalentes, il pourra indiquer à l'utilisateur le fait que le site où la sous-tâche **T**′ est définie n'est pas disponible **et** lui proposer d'utiliser la (ou l'une des) tâche(s) équivalente(s) à **T**′ définie(s) dans d'autre(s) site(s), celui-ci(ceux-ci) disponible(s). Par conséquent l'exécution de la tâche **T** pourra être totalement réalisée.

ii) À plus long terme, l'introduction d'un module de comparaison de tâches dans un ERP *coopératif* rendra possible l'identification de tâches qui permettent de résoudre une même classe de problèmes. Par conséquent, le système pourra offrir à l'utilisateur plusieurs options de tâches qui, même représentant des raisonnements différents, permettent de résoudre son problème. L'utilisateur choisira la tâche qui représente un raisonnement qui lui convient[8]. De plus, il pourra découvrir d'autres façons d'analyser le même problème.

Évidemment, un ERP *coopératif distribué* bénéficiera des deux avantages cités ci-dessus.

Dans un point de vue théorique, considérant que les E-tâche-graphes sont identifiés comme des *graphes séries-parallèle aux arcs étiquetés* (GSP), nous pouvons envisager d'utiliser l'étude sur les E-tâche-graphes aux diverses applications des GSP.

8. Même si l'on propose de comparer des stratégies en utilisant les E-tâche-graphes, cela n'empêche que ce soit encore l'arbre de décomposition d'une tâche en sous-tâches la représentation des tâches exhibée à l'utilisateur, lors d'une résolution d'un problème. La hiérarchie de décomposition d'une tâche représente le raisonnement utilisé pour résoudre une classe de problèmes, en tous ses niveaux d'abstraction. Un E-tâche-graphe permet de représenter l'ensemble des plans d'actions décrit par une stratégie.

Annexe A

Terminologie et notations de la théorie des graphes

Cette annexe fait un rappel de quelques concepts de base de la théorie de graphes et donne les notations utilisées dans ce mémoire.

1. Un graphe $G = (V, X)$ est identifié par l'ensemble V de ses sommets et l'ensemble X de ses arêtes.

2. Un graphe orienté est appelé *digraphe* ("directed graph", en anglais). Dans ce cas X est un ensemble d'*arcs*.

3. Un arc $x = (\overrightarrow{v_i, v_j})$ est un arc qui *sort* de v_i et *arrive* à v_j. Les sommets v_i et v_j sont dits *sommets adjacents*. De plus, v_j est dénommé *successeur* de v_i, et v_i *prédécesseur* de v_j.

4. Soit $G = (V, X)$. Un graphe $G' = (V', X')$ où $V' \subset V$ et X' est tel que $\forall x = (\overrightarrow{v_i, v_j}) \in X$, si $v_i \in V'$ et $v_j \in V', x \in X'$; G' est dénommé *sous-graphe* de G.

5. Le nombre d'arcs qui sortent d'un sommet v est noté $d^+(v)$ (*degré de sortie* de v). De même, le nombre d'arcs qui arrivent à un sommet v est noté $d^-(v)$ (*degré d'entrée* de v).

6. Un sommet dont le *degré d'entrée* est égal à 0 est dénommé *source*. Un sommet dont le *degré de sortie* est égal à 0 est dénommé *puits*.

7. Une *arborescence* est un digraphe sans circuit et qui possède un et un seul sommet v tel que $d^-(v) = 0$. Ce sommet v est dénommé la *racine* de l'arborescence. Une arborescence est identifiée par $T = (V, X, r)$, où r est sa racine. Dans une arborescence,

   - $\forall v \in V, v \neq r$ si $d^+(v) > 0$ alors v est dénommé *sommet interne*, ainsi que si $d^+(v) = 0$ alors v est dénommé *feuille*.

   - $\forall x \in X, x = (\overrightarrow{v_i, v_j}),\ v_i$ est dénommé *père* de v_j ainsi que v_j est dénommé *fils* de v_i.

- $|X| = |V| - 1$.

Nota : *Arborescence* est un concept orienté. Un graphe **non orienté** tel que $|X| = |V| - 1$ est dénommé un *arbre*. Dans ce travail nous parlerons d'*arbres de décomposition*, même s'ils possèdent une "racine", pour être cohérent avec la terminologie utilisée dans d'autres travaux.

8. Soit $\mathcal{L}$ un ensemble d'étiquettes. Une fonction λ d'étiquetage des arcs sur $\mathcal{L}$ est une fonction :

$$\begin{array}{rcl} \lambda : X & \longrightarrow & \mathcal{L} \\ x & \longmapsto & \lambda(x) \end{array}$$

et une fonction λ d'étiquetage des sommets sur $\mathcal{L}$ est

$$\begin{array}{rcl} \lambda : V & \longrightarrow & \mathcal{L} \\ v & \longmapsto & \lambda(v) \end{array}$$

9. Un digraphe G aux *arcs étiquetés* est donc identifié par $G = (V, X, \lambda, \mathcal{L})$ où λ est une fonction sur $\mathcal{L}$ d'étiquetage des arcs. De même, une arborescence T, de racine r, aux *sommets étiquetés* est identifiée par $T = (V, X, r, \lambda, \mathcal{L})$ où λ est une fonction sur $\mathcal{L}$ d'étiquetage des sommets.

10. Un *chemin* c est une séquence de sommets $v_0, v_1, \ldots, v_n$ tel qu'il existe un arc $(\overrightarrow{v_i, v_{i+1}})$ pour tout $0 \leq i \leq n-1$.

- Si $v_0 = v_n$, c est un *circuit*.
- $\forall\ 0 \leq i \leq n-1,\ i+1 \leq j \leq n$, un chemin c' allant de v_i à v_j est un *sous-chemin* de c.
- Dans un digraphe arc étiqueté, un chemin c peut être identifié

  i) soit par ses sommets ($c = v_0 v_1 v_2 \ \ldots \ v_{n-1} v_n$),

  ii) soit par les étiquettes de ses arcs ($c = \lambda(\overrightarrow{v_0, v_1})\lambda(\overrightarrow{v_1, v_2}) \ldots \lambda(\overrightarrow{v_{n-1}, v_n})$).

Annexe B

E-tâche-graphes et graphes série-parallèle

Les graphes série-parallèles (abrégé, par la suite, GSP) ont déjà été utilisés pour modéliser divers problèmes comme par exemple, le problème d'ordonnancement de tâches en calcul parallèle ([Fin96]), le problème de réseau électrique ([Duf65]) ou encore l'analyse dynamique du comportement d'un système distribué ([Bou97]).

Dans la littérature [Val82], on trouve deux façons de construire un GSP $G' = (V, X)$, à partir d'un graphe G composé d'un seul arc $x = (\overrightarrow{v_i, v_j})$ (v_i est donc la source de G et v_j, son puits) : (i) par *décompositions* en série et/ou en parallèle ; (ii) par *compositions* en série et/ou en parallèle. La première remplace un arc x de G soit par une séquence d'arcs (décomposition en *série*), soit par un ensemble d'arcs (décomposition en *parallèle*). La deuxième, étant donné deux GSPs G_i et G_i, permet de construire un nouveau GSP G_k soit en fusionnant le puits de G_i avec la source de G_j (*composition en série*), soit en fusionnant les sources de G_i et G_j aussi bien que leurs puits (*composition en parallèle*).

Valdes $\&$ *al* [Val82] ont donné deux critères pour reconnaître si un digraphe est un GSP. L'un des deux est l'identification de *sous-graphe interdit*. Ils ont démontré qu'*un digraphe avec une seule source et un seul puits est un GSP si et seulement s'il* **ne contient pas** *un sous-graphe homeomorphe au digraphe W* (illustré dans la figure B.1).

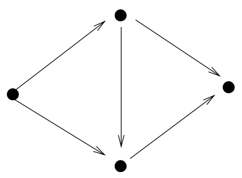


FIG. B.1 – *Sous-graphe interdit pour un GSP*

Nota : Un graphe G contient un sous-graphe *homeomorphe* à un digraphe G' si G' peut être obtenu de G par une séquence des opérations suivantes : i) enlever un arc ; ii) enlever un sommet isolé ; iii) si un sommet v est tel que $d^{-}(v) = d^{+}(v) = 1$, enlever v et remplacer les deux arcs $(\overrightarrow{u,v})$ et $(\overrightarrow{v,w})$ par un arc $(\overrightarrow{u,w})$. L'application récursive de cette dernière opération, en utilisant le vocabulaires des E-tâche-graphes, signifie remplacer une séquence allant d'un sommet u à un sommet v par un arc $(\overrightarrow{u,v})$.

La construction d'un GSP par *décompositions série-parallèles* correspond à la technique de décompositions successives d'un problème en sous-problème, tandis que la *composition série-parallèle* des GSP correspond à liaison de programmes pour le pilotage d'une bibliothèque de programmes scientifiques. Autrement dit, la *décomposition* correspond à l'analyse *top-down* d'un problème tandis que la *composition* correspond à l'analyse *bottom-up*. Les opérations de construction des E-tâche-graphes ne sont donc rien d'autre qu'une *composition en série/parallèle d'E-tâche-graphes*, basée sur les E-tâche-graphes élémentaires.

L'identification des E-tâche-graphes comme digraphes série-parallèle et le critère de reconnaissance d'un GSP nous sont très utile car on a, maintenant, un support pour démontrer qu'un digraphe G obtenu à partir d'un E-tâche-graphe après quelques transformations lors de l'identification de la différence (voir section 3.2.2), est également un E-tâche-graphe.

Annexe C

Borne inférieure du nombre de trajectoires décrites par un E-tâche-graphe

L'extension d'un E-tâche-graphe sous une configuration complexe peut produire un nombre exponentiel de trajectoires. En effet, la forme étendue d'un E-tâche-graphe $G = (V, X, I, O, \lambda, \mathcal{L})$, obtenue par $exp(G)$, est un faisceau qui contient toutes les trajectoires de G. Or, le nombre de trajectoires de G, dénommé *taille de* G, est toujours multiplié à chaque sommet $v \in V$ tel que $d^+(v) > 1$. Dans cet annexe, on évalue une expression qui indique une borne inférieure pour la taille d'un E-tâche-graphe G.

Taille d'un E-tâche-graphe

Définition 35 La *taille* d'un E-tâche-graphe G (noté par $|G|$) est le nombre de trajectoires décrites dans G. Par conséquent, $|G| = |exp(G)|$ puisque $exp(G)$ décrit explicitement toutes les trajectoires de G.

La taille d'un E-tâche-graphe $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij}$ sous une configuration fondamentale est facilement déterminée puisque un tel G est un faisceau[1] ($F = \sum_{i=1}^{m} \prod_{j=1}^{n_i} E_{ij}$) et le nombre de trajectoire de F est donné par m (*cf* propriété 15 ci-après).

Par contre, la taille d'un E-tâche-graphe $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij}$ sous une configuration complexe n'est pas toujours évidente d'être déterminée. Elle dépend de la configuration de chaque G_{ij}, qui peut être à son tour également sous une configuration complexe. Cependant, une borne inférieure pour la taille de G sous une configuration complexe peut être déterminée (*cf* propriété 16 ci-après).

1. Rappelons que dans les configurations fondamentales se trouvent les E-tâche-graphes élémentaires, les séquences et les faisceaux. De plus, $F \supset S \supset E$.

Propriété 15 Si G est sous une configuration fondamentale, c'est-à-dire $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} E_{ij}$ alors $|exp(G)| = m$.

Soit G sous une configuration fondamentale. Alors $G = \sum_{i=1}^{m}(\prod_{j=1}^{n_i} E_{ij}) = \sum_{i=1}^{m} S_{ij}$.

Or, chaque S_{ij} est une trajectoire puisque est une séquence allant de I à O.

Alors G possède m trajectoires et donc $|G| = m$.

En particulier, si $m = 1$, G est une séquence et donc $|S| = 1$.

Propriété 16 Si G est sous une configuration complexe, c'est-à-dire $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij}$ où $\exists\ G_{ij}$ tel que G_{ij} n'est pas élémentaire, alors $|exp(G)| \geq m2^{k_i}$ où k_i est le nombre de G_{ij} non élémentaire de l-ième composant additif de G.

D'après la propriété 9,

$exp(G) = \sum_{i=1}^{m} exp(\prod_{j=1}^{n_i} G_{ij})$ et donc $|exp(G)| = \sum_{i=1}^{m} |exp(\prod_{j=1}^{n_i} G_{ij})|$

D'après la propriété 10,

$exp(\prod_{j=1}^{n_i} G_{ij}) = \prod_{j=1}^{n_i} exp(G_{ij})$ et donc $|exp(\prod_{j=1}^{n_i} G_{ij})| = \prod_{j=1}^{n_i} |exp(G_{ij})|$.

Alors, $|G| = |exp(G)| = \sum_{i=1}^{m} \prod_{j=1}^{n_i} |exp(G_{ij})|$.

Si G sous une configuration complexe, alors $G = \sum_{i=1}^{m} \prod_{j=1}^{n_i} G_{ij}$ où $\exists G_{ij}$ tel que G_{ij} n'est pas élémentaire.

Supposons, d'abord, que G soit multiplicatif (G = M), $m = 1$ et $|M| = \prod_{j=1}^{n} |exp(G_j)|$. La taille de M est donc le produit de la taille de tous ses composants multiplicatifs.

La configuration "assez" simple pour un tel M, est qu'il existe k des G_j tels que G_j sont des faisceaux et les autres $n - k$ soient des E-tâche-graphes élémentaires. Dans ce cas là,

$$|M| = \prod_{\substack{j \mid G_j \text{ est un} \\ \text{faisceau}}} |F_j| \ . \prod_{\substack{j \mid G_j \text{ est} \\ \text{élémentaire}}} |E_j| \geq 2^k$$

puisque $|G_j| = 1\ \forall j \mid G_j$ est élémentaire
et $|G_j| \geq 2\ \forall j \mid G_j$ est un faisceau $(F_j = \sum_{i=1}^{m_j} S_j,\ |F_j| = m_j \geq 2)$.

Alors, $|exp(M)| \geq 2^k$ où k est le nombre de G_j non élémentaires.

Supposons, maintenant, que G soit additif (G = A) et $|A| = \sum_{i=1}^{m} |M_i|$. La taille de A est donc le somme de la taille de tous ses composants additifs.

Or, $|M_i| \geq 2^{k_i},\ \forall i = 1, 2, \ldots, m.$

Alors, $|exp(A)| \geq m2^{k_i}$

Annexe D

Algorithmes

Dans cette annexe nous décrivons les algorithmes indiqués dans ce mémoire.

D.1 Algorithme sous-E-tâche-graphe-additif

sous-E-tâche-graphe-additif $(G, I, anc(I), O)$
$LIST \longleftarrow \{I\}$
Tant qu'il $|LIST| > 1$ ou $LIST = \{I\}$
$v \longleftarrow$ prochain élément dans $LIST$
enlever v de $LIST$ et inclure v dans $LIST - TRAITE$
pour tout $x = (\overrightarrow{v, u})$ sortant de v
$w \longleftarrow$ premier sommet atteint par le premier chemin, dans l'ordre vertical, partant de v
en passant par x tel que $d^-(w) > 1$ ou $d^+(w) > 1$
% w indique la fin ou le début d'un choix ($d^-(w) > 1$ pour la fin et $d^+(w) > 1$ pour le début)
si $d^-(w) > 1$
si w non marqué
marquer w
insérer v dans $anc(w)$ % ($anc(w)$ = liste des ancêtres de w)
insérer w dans $LIST$
sinon
si $v \in anc(w)$
si $(v, w) \notin FF$ % FF = ensemble de faisceaux fondamentaux
inclure (v, w) dans FF
$anc(w) \longleftarrow anc(w) \bigcup anc(v)$
fin si
sinon
insérer v dans $anc(w)$
$anc - commun(v, w) \longleftarrow anc(v) \bigcap anc(w)$
si $anc - commun(v, w) \neq \emptyset$
$w' \longleftarrow$ le dernier élément de $anc - commun(v, w)$
$G' = (V', X', w', w, \lambda, \mathcal{L})$ est un sous-E-tâche-graphe additif de G %$w' = I', w = O'$
fin si
fin si
fin si
sinon
insérer v dans $anc(w)$
sous-E-tâche-graphe-additif $(G, w, anc(w), O)$
$anc(w) \longleftarrow anc(w) \bigcup anc(v)$
si $O \notin LIST - TRAITE$
insérer O dans $LIST$
marquer O
fin si
fin si
fin pour
fin tant que
$O \longleftarrow w$ **fin**

TAB. D.1 – *Algorithme de recherche d'un sous-E-tâche-graphe additif.*

D.2 Algorithme de construction d'un E-tâche-graphe

E-tâche-graphe (**T**, *par*)
Entrée : - description complète de la tâche **T** (décomposition et spécialisation, si c'est le cas) ;
- *par* = "nill" si **T** est une "tâche principale" ;
= **T**$'$, le nom de la "sur-tâche" de **T** si elle fait partie d'une stratégie d'une autre tâche
Sortie : digraphe arc étiqueté D associée à la tâche **T**
début
si *par* = "nill" % **T** est la tâche principale
créer deux sommets (I et O) % Ces deux sommets représentent, respectivement, le début et la fin de l'exécution de **T**
créer un arc $x = (\overrightarrow{I, O})$ étiqueté par le nom de la tâche **T**
fin si
soit $x = (\overrightarrow{v_i, v_j})$ l'arc en D associé à **T**
si T est une tâche élémentaire
remplacer l'étiquette de x par le *nom du programme* qui est associé à l'exécution de **T**
sinon
si T est *spécialisable*
si T est aussi *complexe*
si *par* = "nill" % **T** est la tâche principale
ajouter à x autant d'arcs $(\overrightarrow{v_i, v_j})$ qu'il existe de sous-tâches qui spécialisent **T**
sinon % *par* = **T**$'$ et **T** est une sous-tâche de **T**$'$
ajouter à x autant d'arcs $(\overrightarrow{v_i, v_j})$ qu'il existe de sous-tâches qui spécialisent **T** dont les données sont compatibles avec celles de **T**$'$
fin si
sinon % **T** n'est pas décomposable
si *par* = "nill" % **T** est la tâche principale
remplacer x par autant d'arcs $(\overrightarrow{v_i, v_j})$ qu'il existe de sous-tâches qui spécialisent **T**
sinon % *par* = **T**$'$ et **T** est une sous-tâche de **T**$'$
remplacer x par autant d'arcs $(\overrightarrow{v_i, v_j})$ qu'il existe de sous-tâches qui spécialisent **T** dont les données sont compatibles avec celles de **T**$'$
fin si
fin si
sinon
si T est *complexe*
si T est décomposée en *séquence*
remplacer x par un *chemin*, allant de v_i à v_j, composé d'autant d'arcs qu'il existe des sous-tâches composant **T**
% Chaque arc de ce chemin est étiqueté par le nom de la sous-tâche de **T** qu'il représente
sinon
si T est décomposée en *choix*
remplacer x par d'autant d'*arcs* $(\overrightarrow{v_i, v_j})$ qu'il existe des sous-tâches composant **T**
% Chacun de ces arcs est étiqueté par le nom de la sous-tâche de **T** qu'il qu'il représente
fin si
fin si
fin si
fin si
pour chaque chemin c allant de v_i à v_j
pour chaque arc x_k de c
soit **ST**$_k$ la sous-tâche de **T** représentée par x_k
remplacer x_k par ***E-tâche-graphe*** (ST$_k$,**T**)
fin pour
fin pour
fin si
fin

TAB. D.2 – *Algorithme pour contruire un E-tâche-graphe à partir de la description d'une stratégie.*

D.3 Algorithmes auxiliaires pour déterminer la différence

D.3.1 Algorithme éclater un sommet

éclater$(w_i, v_i, x^-(w_i), G_i, w'_i)$
insérer un nouveau arc $x'_i = (\overrightarrow{v_i, w'_i})$ dans G_i avec $\lambda_i(x'_i) = x^-(w_i)$
% Notez que $x^-(w_i) = \lambda_i(\overrightarrow{v_i, w_i})$
ajouter-sous-graphe$(G_i, G_i, w'_i, w_i, w'_{i1}, w_{i1})$
si $w'_{i1} \neq O_i$
fusionner w'_{i1} et w_{i1}
fin si
enlever l'arc $x_i = (\overrightarrow{v_i, w_i})$ de G_i tel que $\lambda_i(x_i) = x^-(w_i)$
fin

TAB. D.3 – *Algorithme pour éclater un sommet :* algorithme auxiliaire pour déterminer la différence entre deux E-tâche-graphes.

D.3.2 Algorithme fusionner deux sommets

fusionner(G_i, w_i, w'_i)
$X^-(w_i) \longleftarrow \{$ étiquettes des arcs arrivant à $w_i\}$
pour chaque $x^-(w_i) = \lambda(\overrightarrow{v_i, w_i})$
insérer un nouveau arc $x'_i = (\overrightarrow{v_i, w'_i})$ dans G_i avec $\lambda_i(x'_i) = x^-(w_i)$
enlever l'arc $x_i = (\overrightarrow{v_i, w_i})$ avec $\lambda(x_i) = x^-(w_i)$
fin pour
fin

TAB. D.4 – *Algorithme pour fusionner deux sommets :* algorithme auxiliaire pour déterminer la différence entre deux E-tâche-graphes.

D.3.3 Algorithme ajouter-sous-graphe

ajouter-sous-graphe $(G_i, G_j, w_i, w_j, w'_i, w_j)$
si $d^+(w_j) = 1$
insérer un nouveau arc $x'_i = (\overrightarrow{w_i, w'_i})$ dans G_i avec $\lambda_i(x'_i) = \lambda_j(\overrightarrow{w_j, w'_j})$
si $d^-(w'_j) = 1$
ajouter-sous-graphe $(G_i, G_j, w'_i, w'_j, w'_{i1}, w'_{j1})$
fin si
sinon
$G'_j = (V'_j, X'_j, I'_j, O'_j, \lambda_j, \mathcal{L}) \longleftarrow$ ***sous-graphe-additif*** $(G_j, w_j, \emptyset, O'_j)$ $\%O'_j = w_j$
$G'_i = (V'_i, X'_i, I'_i, O'_i, \lambda_i, \mathcal{L}) \longleftarrow G'_j$
identifier w_i avec I'_i
$w'_i \longleftarrow O'_i$
si $w_j = O_j$
identifier O'_i avec O_i
sinon
si $d^-(O'_j) = d^-(w'_j)$
ajouter-sous-graphe $(G_i, G_j, w'_i, w_j, w'_{i1}, w_{j1})$
fin si
fin si
fin si
fin

TAB. D.5 – *Algorithme auxiliaire pour déterminer la différence entre deux E-tâche-graphes G_i et G_j :* Il permet d'ajouter dans G_i un sous-graphe de G_j qui décrit des trajectoires manquantes en G_i par rapport à G_j.

D.3.4 Algorithme enlever-sous-graphe

enlever-sous-graphe (G_i, w_i, w'_i)
si $d^+(w_i) = 1$
enlever l'arc $x_i = (\overrightarrow{w_i, w'_i})$ partant de w_i
sinon
$G'_i = (V'_i, X'_i, I'_i, O'_i, \lambda_i, \mathcal{L}) \longleftarrow$ ***sous-graphe-additif*** $(G_i, w_i, \emptyset, w'_i)$ $\%O'_i = w'_i$
enlever G'_i de G_i
fin si
si $d^-(w'_i) = 0$
si $w'_i \neq O_i$
enlever-sous-graphe $(G_i, w'_i w'_{i1})$
fin si
fin si
fin

TAB. D.6 – *Algorithme auxiliaire pour déterminer la différence entre deux E-tâche-graphes G_i et G_j :* Cet algorithme permet d'enlever de G_i un sous-graphe qui décrit des trajectoires de plus par rapport à G_j.

D.3.5 Algorithmes prédicats

D.3.5.1 Additif

additif (G)
$A \longleftarrow 1$
si $d^+(I) > 1$
$G \longleftarrow$ ***sous-E-tâche-graphe-aditif*** $(G, I, \emptyset, O')$
si $O' = O$
$A \longleftarrow 0$
fin si
sinon
$A \longleftarrow 0$
fin si
fin

TAB. D.7 – *Algorithme pour déterminer si un E-tâche-graphe est additif.*

D.3.5.2 Faisceau

faisceau (G)
$F \longleftarrow 1$
si $d^+(I) > 1$
$X \longleftarrow \{$ arcs partant de I, dans l'ordre vertical, à partir de $i\}$
Tant qu'il existe $x = (\overrightarrow{I, v})$ non marqué et $F = 1$
si $d^-(w) = 1$
si ***séquence*** (G, v)
marquer x
sinon
$F \longleftarrow 0$
fin si
sinon
si $w = O$
marquer $x^+(I)$
sinon
$F \longleftarrow 0$
fin si
fin si
fin tant que
sinon
$F \longleftarrow 0$
fin si
fin

TAB. D.8 – *Algorithme pour déterminer si un E-tâche-graphe est un faisceau.*

D.3.5.3 Multiplicatif

multiplicatif (G)
  $M \longleftarrow 1$
  si $d^+(I) > 1$
    $G \longleftarrow$ ***sous-E-tâche-graphe-aditif*** $(G, I, \emptyset, O')$
    si $O' = O$
      $M \longleftarrow 0$
    fin si
  fin si
fin

TAB. D.9 – *Algorithme pour déterminer si un E-tâche-graphe est multiplicatif.*

D.3.5.4 Séquence

séquence (G, v)
  $S \longleftarrow 1$
  si $d^+(v) = 1$
    $x^+(v) \longleftarrow (\overrightarrow{v, w})$ % arc partant de v
    si $w \neq O$
      séquence (G, w)
    fin si
  sinon
    $S \longleftarrow 0$
  fin si
fin

TAB. D.10 – *Algorithme pour déterminer si un E-tâche-graphe est une séquence.*

D.3.6 Algorithmes auxiliaires pour déterminer la forme normale d'un E-tâche-graphe

D.3.6.1 Factorisation à gauche d'un faisceau

factorisation-gauche-faisceau (G, i)
  $X^+(I, i) \longleftarrow \{$ arcs partant de I dans l'ordre vertical à partir de i $\}$ % $x_i^+ = (\overrightarrow{I, v_i}),\ x_{i+1}^+ = (\overrightarrow{I, v_{i+1}})$
  si $\lambda(x_i^+) = \lambda(x_{i+1}^+)$
    $x^+(v_{i+1}) \longleftarrow$ arc partant de v_{i+1} % $x^+(v_{i+1}) = (\overrightarrow{v_{i+1}, w})$
    créer un arc $(\overrightarrow{v_i, w})$ étiqueté par $\lambda(x^+(v_{i+1}))$
    enlever l'arc $x^+(v_{i+1})$
    enlever l'arc x_{i+1}^+
    si $i + 1 < d^+(I)$
      factorisation-gauche-faisceau (G, i)
    fin si
  sinon
    si $i + 1 < d^-(I)$
      factorisation-gauche-faisceau $(G, i + 1)$
    fin si
  fin si
fin

TAB. D.11 – *Algorithme pour factoriser un faisceau à gauche.*

D.3.6.2 Factorisation à droite d'un faisceau

factorisation-droite-faisceau (G, i)
$X^-(O,i) \longleftarrow$ { arcs arrivant à O dans l'ordre vertical à partir de i } % $x_i^- = (\overrightarrow{v_i, O}),\ x_{i+1}^- = (\overrightarrow{v_{i+1}, O})$
si $\lambda(x_i^-) = \lambda(x_{i+1}^-)$
$x^-(v_{i+1}) \longleftarrow$ arc arrivant à v_{i+1} % $x^-(v_{i+1}) = (\overrightarrow{w, v_{i+1}})$
créer un arc $(\overrightarrow{w, v_i})$ étiqueté par $\lambda(x^-(v_{i+1}))$
enlever l'arc $x^-(v_{i+1})$
enlever l'arc x_{i+1}^-
si $i + 1 < d^-(O)$
factorisation-droite-faisceau (G, i)
fin si
sinon
si $i + 1 < d^-(O)$
factorisation-droite-faisceau $(G, i + 1)$
fin si
fin si
fin

TAB. D.12 – *Algorithme pour factoriser un faisceau à droite.*

D.3.6.3 Factorisation à gauche

factorisation-gauche $(G, G', i, change)$
$change \longleftarrow 0$
si $G' = 0$
$x_i \longleftarrow (\overrightarrow{I, v_i})$
$x_{i+1} \longleftarrow (\overrightarrow{I, v_{i+1}})$
$w_i \longleftarrow$ le premier sommet atteint par le premier chemin, dans l'ordre vertical, partant de v_i, en passant par x_i, tel que $d^-(w_i) > 1$
$w_{i+1} \longleftarrow$ le premier sommet atteint par le premier chemin, dans l'ordre vertical, partant de v_i, en passant par x_{i+1}, tel que $d^-(w_{i+1}) > 1$
si $w_i = w_{i+1}$
si $\lambda(x_i) = \lambda(x_{i+1})$
$X(w_{i+1}) \longleftarrow$ { arcs partant de w_{i+1} }
pour chaque arc $x(w_{i+1}) = (\overrightarrow{w_{i+1}, u_{i+1}})$
créer un arc $(\overrightarrow{w_i, u_{i+1}})$ étiqueté par $\lambda(x(w_{i+1}))$
enlever l'arc $x(w_{i+1})$
$change \longleftarrow 1$
fin pour
enlever l'arc x_{i+1}
si $i + 1 < d^+(I)$
factorisation-gauche $(G, \emptyset, i, change)$
fin si
sinon
si $w_i = O$
si $i + 1 < d^+(I)$
factorisation-gauche $(G, \emptyset, i + 1, change)$
fin si
sinon
$G'_i \longleftarrow$ ***mirroir*** (***sous-etg-additif*** (***mirroir*** $(G), v_i, \emptyset, O')$)
$j \longleftarrow i + d^+(I'_i) - 1$
si $j < d^+(I)$
factorisation-gauche $(G, G'_i, j, change)$
fin si
fin si
fin si
sinon
si $i + 1 < d^+(I)$
factorisation-gauche $(G, \emptyset, i + 1, change)$
fin si
fin si

sinon
$x_i \longleftarrow (\overrightarrow{I, v_i})$
$x'_1 \longleftarrow (\overrightarrow{I', v'_1})$
si $\lambda(x_i) = \lambda(x'_i + 1)$
$G'_i \longleftarrow$ ***mirroir*** (***sous-etg-additif*** (***mirroir*** $(G), v_i, \emptyset, O'_i))$
si $G' = G'_i$
$w_i \longleftarrow O'_i$
$w_j \longleftarrow O'$
$X(w_i) \longleftarrow$ { arcs partant de w_i }
pour chaque arc $x(w_i) = (\overrightarrow{w_i, u_i})$
créer un arc $(\overrightarrow{w_j, u_{i+1}})$ étiqueté par $\lambda(x(w_i))$
enlever l'arc $x(w_i)$
$change \longleftarrow 1$
fin pour
enlever G'_i
fin si
$j \longleftarrow i + d^+(I') + d^+(I'_i) - 1$
si $j < d^+(I)$
factorisation-gauche $(G, G'_i, j, change)$
fin si
sinon
si $i + 1 < d^+(I)$
factorisation-gauche $(G, \emptyset, i + 1, change)$
fin si
fin si
fin si
fin

TAB. D.13 – *Algorithme pour factoriser à gauche un E-tâche-graphe sous une configuration complexe.*

Nota : **mirroir**(G) consiste en inverser le sens des arcs d'un E-tâche-graphe. Il est en $\mathcal{O}(|X|)$.

D.3.6.4 Factorisation à droite

factorisation-droite $(G, change)$
$change \longleftarrow 0$
$G' \longleftarrow$ ***mirroir*** (G)
$G' \longleftarrow$ ***ordre-lexico*** (G')
$G' \longleftarrow$ ***factorisation-gauche*** (G')
$G' \longleftarrow$ ***mirroir*** (G')
si $G \neq G'$
$change \longleftarrow 1$
fin si
fin

TAB. D.14 – *Algorithme pour factoriser à droite un E-tâche-graphe sous une configuration complexe.*

D.4 Algorithme pour déterminer la forme normale d'un E-tâche-graphe

```
normal (G)
    si sequence(G, I)
      n(G) ⟵ G
    sinon
      si multiplicatif (G)
        pour chaque composant multiplicatif G' de G
          remplacer G' par normal (G')
          si change
            normal (G)
          fin si
        fin pour
      sinon
        ordre-lexico (G)
        si faisceau (G)
          factorisation-gauche-faisceau (G, 1)
          si faisceau (G)
            mirroir (G)
            ordre-lexico (G)
            mirroir (G)
            factorisation-droite-faisceau (G, 1)
            si faisceau (G)
              ordre-lexico (G)
              n(G) ⟵ G
            sinon
              normal (G)
            fin si
          sinon
            normal (G)
          fin si
        sinon
          change ⟵ 1
          Tant que change = 1
              factorisation-gauche (G, ∅, 1, change)
          fin tant que
          factorisation-droite (G, change)
          si change
            normal (G)
          sinon
            si additif (G)
              pour chaque composant additif G' de G
                remplacer G' par normal (G')
                si change
                  normal (G)
                fin si
              fin pour
            fin si
          fin si
        fin si
      fin si
    fin si
fin
```

TAB. D.15 – *Algorithme pour déterminer la forme normale d'un E-tâche-graphe.*

D.5 Algorithme de différence

différence $(G_i = (V_i, X_i, I_i, O_i, \lambda_i, \mathcal{L})\ ,\ G_j = (V_j, X_j, I_j, O_j, \lambda_j, \mathcal{L}))$
$LIST \longleftarrow \{(I_i, I_j, \emptyset)\}$
% $LIST = (v_i, v_j, x^-(v_j))$ représente un ensemble de paires de sommets qui sont portées de sous-trajectoires communes aux deux E-tâche-graphes et $x^-(v_j)$ est l'étiquette du dernier arc d'une de ces sous-trajectoires communes.
Tant qu'il existe $(v_i, v_j, x^-(v_j))$ dans $LIST$
et que G_i n'est pas encore identifié comme différent de G_j
$X^+(v_i) \longleftarrow \{$ étiquettes des arcs sortant de $v_i\}$
$X^+(v_j) \longleftarrow \{$ étiquettes des arcs sortant de $v_j\}$
pour tout $x^+(v_i) = \lambda_i(\overrightarrow{v_i, w_i}) \in X^+(v_i)$
si $\exists\ x^+(v_j) = \lambda_j(\overrightarrow{v_j, w_j}) \in X^+(v_j)$ tel que $x^+(v_j) = x^+(v_i)$
si $\exists (w_i, w'_j, x^-(w'_j))$ dans $LIST$ tel que $w'_j \neq w_j$
éclater $(w_i, v_i, x^-(w'_j), G_i, w'_i)$
substituer $(w_i, w'_j, x^-(w'_j))$ par $(w'_i, w'_j, x^-(w'_j))$ dans $LIST$
fin si
inclure (w_i, w_j) dans $LIST$
marquer $x_j = (\overrightarrow{v_j, w_j})$ avec $\lambda_j(x_j) = x^+(v_i)$
sinon
enlever l'arc $x_i = (\overrightarrow{v_i, w_i})$ de G_i tel que $\lambda_i(x_i) = x^+(v_i)$
si ($d^+(v_i) = 0$ et $v_i = I_i$) *ou* ($d^-(w_i) = 0$ et $w_i = O_i$)
$G_i \neq G_j$
sinon
si $d^-(w_i) = 0$ et $w_i \neq O_i$
enlever-sous-graphe (G_i, w_i, w'_i) %w'_i est le sommet final du sous-graphe qui a été enlevé de G_i
si ($d^-(w'_i) = 0$ et $w_i = O_i$)
$G_i \neq G_j$
fin si
fin si
fin si
fin si
fin pour
si $\exists\ x^+(v_j) = \lambda_j(\overrightarrow{v_j, w_j}) \in X^+(v_j)$ tel que $x_j = (\overrightarrow{v_j, w_j})$ *non marqué* dans G_j
ajouter un nouveau arc $x_i = (\overrightarrow{v_i, w_i})$ dans G_i avec $\lambda_i(x_i) = \lambda_j(x_j) = x^+(v_j)$
si $d^-(w_j) = 1$
ajouter-sous-graphe $(G_i, G_j, w_i, w_j, w'_i, w'_j)$
% w'_i, w'_j sont, respectivement, les sommets finals du sous-graphe de G_j ajouté à G_i
inclure (w'_i, w'_j) dans $LIST$
sinon
inclure (w_i, w_j) dans $LIST$
fin si
fin si
si $\exists$ dans $LIST$ $(w_i, w_j, x^-(w_j))$ et $(w'_i, w'_j, x^-(w'_j))$ tels que $w_i \neq w'_i$ et $w_j = w'_j$ % (item 5)
fusionner w_i et w'_i
fin si
enlever $(v_i, v_j, x^-(v_j))$ de $LIST$
fin tant que
fin

TAB. D.16 – *Algorithme de différence entre deux E-tâche-graphes.*

D.6 Relation d'ordre lexicographique entre deux E-tâche-graphes multiplicatifs

La comparaison de M_i et M_j se fait par l'appel de ***Comparer-sommets***$(\{I_i\}, \{I_j\})$.

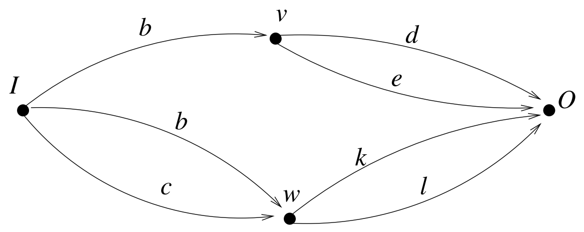
Notations :

Soit un E-tâche-graphe multiplicatif $M = (V, X, I, O, \lambda, \mathcal{L})$. On indiquera par

- $X(V') = \{$ étiquettes des arcs sortant de tous les sommets $v \in V' \mid V' \subseteq V\}$
- Soit $x = \lambda(\overrightarrow{v, w})$, $V(x) = \{$ sommets atteints par des arcs consécutifs, par rapport à l'ordre vertical, partant de v et étiquetés par $x\}$

Par exemple, dans la figure ci-dessous,

- si $V' = \{I\}$ alors $X(V') = X(\{I\}) = \{b, c\}$.
- $\lambda(\overrightarrow{I, v}) = b$ et $\exists$ $(\overrightarrow{I, w}) \mid \lambda(\overrightarrow{I, w}) = b$. Alors, $V(b) = \{v, w\}$.
- si $V' = \{v, w\}$ alors $X(V') = \{d, e, k, l\}$.



Comparer-sommets (V_i', V_j')
(1) $X(V_i') \longleftarrow \{$ étiquettes des arcs sortant de $v_i \in V_i'\}$
(2) $X(V_j') \longleftarrow \{$ étiquettes des arcs sortant de $v_j \in V_j'\}$
(3) **si** $V_i' = \{I_i\}$
(4) $t(I_i) \longleftarrow d^+(I_i)$
(4) % $t(I_i)$ = nombre d'arcs **non examinés** sortant de (I_i)
(5) $t(I_j) \longleftarrow d^+(I_j)$
% $t(I_j)$ = nombre d'arcs **non examinés** sortant de (I_j)
(6) **sinon**
(7) $t(V_i') \longleftarrow \sum_{v_i \in (V_i')} d^-(v_i) - 1$
% $t(V_i')$ = nombre d'arcs **non examinés** arrivant à un des sommets $v_i \in (V_i')$
(8) $t(V_j') \longleftarrow \sum_{v_j \in (V_j')} d^-(v_j) - 1$

% $t(V_j') =$ nombre d'arcs **non examinés** arrivant à un des sommets $v_j \in (V_j')$
(9) **fin-si**
(10) **Tant qu**'il existe $x(v_i) = \lambda_i(\overrightarrow{v_i, w_i}) \in X(V_i')$ non examiné et
qu'aucune relation entre M_i et M_j n'a pas été identifiée
(11) **s**'il existe $x(v_j) = \lambda_j(\overrightarrow{v_j, w_j})$ en $X(V_j')$ non examiné
(12) **si** $V_i' = \{I_i\}$
(13) $t(I_i) \longleftarrow t(I_i) - 1$
(14) $t(I_j) \longleftarrow t(I_j) - 1$
(15) **fin-si**
(16) ***Comparer-arcs*** $(x(v_i), x(v_j))$
(17) **sinon**
(18) **si** $V_i' = \{I_i\}$
(19) $M_i > M_j$
(20) **sinon**
(21) **si** $t(I_j) \neq 0$ ou $t(V_j') \neq 0$
(22) $M_i < M_j$
(23) **sinon**
(24) $M_i > M_j$
(25) **fin-si**
(26) **fin-si**
(27) **fin-si**
(28) **fin-tant-que**
(29) **Si** aucune relation entre M_i et M_j n'a pas encore été identifiée
(30) **s**'il existe $x(v_j)$ en $X(V_j')$
(31) **si** $V_i' = \{I_i\}$
(32) $M_i < M_j$
(33) **sinon**
(34) **si** $t(I_j) \neq 0$ ou $t(V_j') \neq 0$
(35) $M_i > M_j$
(36) **sinon**
(37) $M_i < M_j$
(38) **fin-si**
(39) **fin-si**
(40) **sinon**
(41) **si** $V_i' = \{I_i\}$
(42) $M_i = M_j$
(43) **sinon**
(44) $\forall\, v_i \in V_i'$, marquer v_i et
(45) $\forall\, v_j \in V_j'$, marquer v_j
(46) **fin-si**
(47) **fin-si**
(48) **fin-si**
fin

La condition exprimée par la ligne (18), indique que M_i possède au moins une trajectoire de plus que M_j, puisqu'il existe encore au moins un arc $(\overrightarrow{I_i, w_i})$ en M_i non évalué pour lequel il n'y a pas d'arc correspondant en M_j, vu que tous les arcs $(\overrightarrow{I_j, w_j})$ de M_j ont déjà été évalués. Par conséquent, $M_i > M_j$ (définition 20, item $5.iii$) . Le cas opposé (c'est-à-dire tous les arcs $(\overrightarrow{I_i, w_i})$ de M_i ont déjà été évalués et il existe encore au moins un arc $(\overrightarrow{I_j, w_j})$ en M_j non évalué ; par conséquent M_j possède au moins une trajectoire de plus que M_i), est caractérisé par la condition exprimée dans la ligne (31). Dans l'exemple illustré par la figure D.1, si l'on compare M_1 avec M_2, lors de l'évaluation des arcs partant de I_1, au moment où on évalue l'arc $c = (\overrightarrow{I_1, v_{12}})$, tous les arcs de M_2 partant de I_2 ont déjà été comparés avec les arcs précédants à c et il n'existe donc pas d'arcs en M_2 correspondant à c (ligne (18) de l'algorithme). Cela signifie que M_1 possède au moins une trajectoire de plus que M_2. La prochaine trajectoire commence par l'arc c. Alors $M_1 > M_2$ (définition 20, item $5.iii$). À l'inverse, si l'on compare M_2 avec

M_1, après avoir analysé tous les arcs de M_2 partant de I_2, on vérifie que M_1 possède encore un arc $c = (\overrightarrow{I_1, w_3})$ partant de I_1 sans correspondant en M_2 (ligne (31) de l'algorithme). Alors M_2 possède moins de trajectoires que M_1 et par conséquent, $M_2 < M_1$.

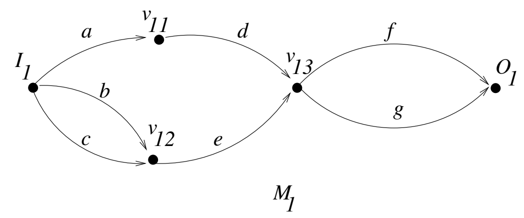


FIG. D.1 – *Exemple d'un cas exprimé par la ligne 18 de l'algorithme (et la situation inverse exprimée par la ligne 31)*

La ligne (21) indique que M_i possède au moins une trajectoire τ_i' de plus ayant $c = (I_i \ldots v_i) = (I_j \ldots v_j)$ comme sous-trajectoire tandis que M_j n'en a plus, puisqu'il n'y a plus d'arcs non examinés partant de v_j. Si M_j possède encore des trajectoires non traitées, soit il existe encore au moins un arc non examiné partant de I_j, ce qui est caractérisé par $t(I_j) \neq 0$; soit il existe encore des sous-trajectoires, non traitées, de portée $v_j \in V_j'$, ce qui est caractérisé par $t(V_j') \neq 0$). La prochaine trajectoire de τ_j' de M_j est telle que $\tau_j' > \tau_i'$ puisque M_i et M_j sont en ordre lexicographique (figure D.2Lorsque $t(I_j) = 0$ et $t(V_j') = 0$ (ligne (23) de l'algorithme), M_j n'a plus aucune trajectoire à évaluer et donc $M_i > M_j$ (définition 20, item $5.iii$). Les situations opposées sont indiquées, respectivement par les conditions exprimées par les lignes (34) et (36). Dans l'exemple illustré par la figure D.2, si l'on compare M_1 avec M_2, lorsque l'on compare les arcs partant de $V_1' = \{v_{12}\}$ (atteint une première fois, dans l'ordre vertical, par la sous-trajectoire $\tau_1' = (gj)$) avec ceux partant de son correspondant en M_2, c'est-à-dire $V_2' = \{v_{23}\}$ (également atteint une première fois, dans l'ordre vertical, par la sous-trajectoire $\tau_2' = (gj)$), on note qu'il existe un arc de plus partant de v_{12} par rapport à ceux partant de v_{23} (l'arc $f = (\overrightarrow{v_{12}, v_{14}})$). Alors, M_1 possède encore au moins une trajectoire $\tau_1'' = (gjf \ldots)$ tandis que M_2 n'a plus de trajectoire, ayant $\tau_2' = (gj)$ comme sous-trajectoire, à examiner. Or, M_2 possède encore des trajectoires à évaluer car, d'une part l'arc $(\overrightarrow{v_{22}, v_{23}})$ n'a pas encore été examiné (ce qui est représenté par $t(V_2') = 1$) et d'autre part les arcs $k = (\overrightarrow{I_2, v_{22}})$ et $u = (\overrightarrow{I_2, v_{22}})$ n'ont pas encore été évalués (ce qui est représenté par $t(I_2) = 2$). Comme M_1 et M_2 sont en ordre lexicographique, la prochaine trajectoire τ_1'' de M_1 est telle que $\tau_1'' < \tau_2''$ où τ_2'' est la prochaine trajectoire de M_2(en l'occurrence $\tau_1'' = (gjfv) < \tau_2'' = (kzbe)$).

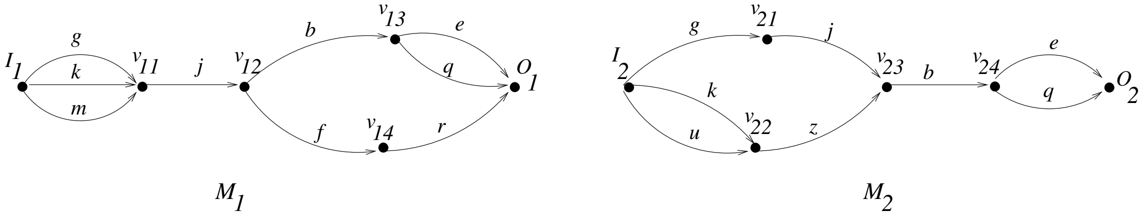


FIG. D.2 – *Exemple d'un cas exprimé par la ligne 21 de l'algorithme (et la situation inverse exprimée par la ligne 34).*

La ligne (40) indique que tous les arcs partant de tous les sommets de V_i' ainsi que ceux partant de tous les sommets de V_j' ont été parcourus. Si V_i' ne contient que le sommet initial I_i (et par conséquent V_j' ne contient que le sommet initial I_j) les E-tâche-graphes sont égaux (ligne (41) de l'algorithme). Dans d'autres cas, cela signifie que les sous-graphes de M_i et M_j partant, respectivement, des sommets appartenant aux ensembles V_i' et V_j' et allant, respectivement, à O_i et à O_j sont égaux. Pour ne plus examiner ces sous-graphes, les sommets $v_i \in V_i'$ et $v_j \in V_j'$ sont marqués. Dans l'exemple illustré par la figure D.3 (basé sur M_1 et M_7 avec un arc de plus, $z = (\overrightarrow{v_{12}, v_{13}}) = (\overrightarrow{v_{22}, v_{23}})$), si l'on compare M_1 avec M_2, lorsque l'on compare les arcs partant de $V_1' = \{v_{13}\}$ avec ceux partant de $V_2' = \{v_{23}\}$ (c'est-à-dire, des sommets atteints une première fois dans l'ordre vertical par une sous-trajectoire $\tau_1' = \tau_2' = (gjb)$ dans les respectifs E-tâche-graphes), on vérifie que le sous-graphe de M_1 partant de v_{13} et allant à O_1 est égal au sous-graphe de M_2 partant de v_{23} et allant à O_2. Alors, on marque les sommets v_{13} et v_{23} pour ne plus les analyser lorsqu'ils seront atteints à nouveau par d'autres sous-trajectoires (par exemple, $\tau_1'' = \tau_2'' = (gjz)$, dans les graphes de la figure D.3, lors de l'analyse du prochain arc partant respectivement de v_{12} et v_{22}).

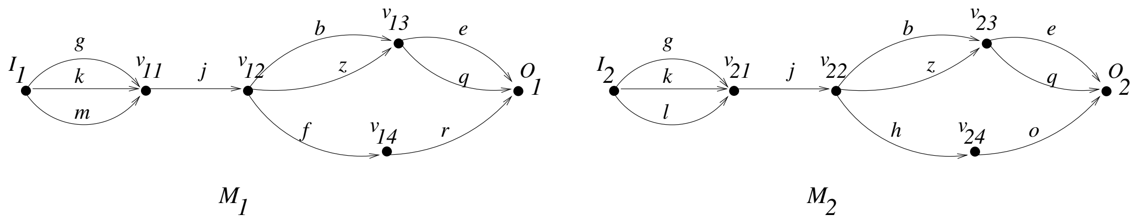


FIG. D.3 – *Exemple d'un cas exprimé par la ligne 40 de l'algorithme.*

Comparer-arcs $(x(v_i), x(v_j))$
(1) **si** $x(v_i) < x(v_j)$
(2) $M_i < M_j$
(3) **sinon**
(4) **si** $x(v_i) > x(v_j)$
(5) $M_i > M_j$
(6) **sinon** % $x(v_i) = x(v_j)$
(7) $V(x(v_i)) \longleftarrow \{$ sommets **non marqués** atteints par des arcs **non examinés**,
(8) consécutifs, par rapport à l'ordre vertical,
(9) partant de v_i et étiquetés par $x(v_i)\}$
(10) $V(x(v_j)) \longleftarrow \{$ sommets **non marqués** atteints par des arcs non **examinés**,
(11) consécutifs par rapport à l'ordre vertical,
(12) partant de v_j et étiquetés par $x(v_j)\}$
(13) **si** $V(x(v_i)) \neq \emptyset$ et $V(x(v_j)) \neq \emptyset$
(14) **si** $O_i \in V(x(v_i))$ et $O_j \notin V(x(v_j))$
(15) $M_i < M_j$
(16) **sinon**
(17) **si** $O_i \notin V(x(v_i))$ et $O_j \in V(x(v_j))$
(18) $M_i > M_j$
(19) **sinon**
(20) ***Comparer-sommets*** $(V(x(v_i)), V(x(v_j)))$
(21) **fin-si**

(22) **fin-si**
(23) **fin-si**
(24) **fin-si**
(25) **fin-si**
fin

Lorsque l'on compare $x(v_i) = \lambda_i(\overrightarrow{v_i, w_i})$ avec $x(v_j) = \lambda_j(\overrightarrow{v_j, w_j})$, en réalité on compare les (sous-)trajectoires $c_i = (I_i \ldots v_i w_i)$ et $c_j = (I_j \ldots v_j w_j)$. Si $x(v_i) < x(v_j)$ alors $c_i < c_j$ et, par conséquent $M_i < M_j$; alors que si $x(v_i) > x(v_j),\ \ c_i > c_j$ et, par conséquent $M_i > M_j$. Par contre, si $x(v_i) = x(v_j)$, les (sous-)trajectoires c_i et c_j sont égales. Alors, il faut analyser les sommets atteints pour ces (sous-)trajectoires dans les respectifs E-tâche-graphes.

La ligne (14) indique que c_i est en réalité un trajectoire de M_i alors que c_j est une sous-trajectoire de M_j. Autrement dit, $c_i = \tau'_i = (I_i \ldots v_i O_i)$; $\ c_j = (I_j \ldots v_j w_j)$ et il existe une trajectoire τ'_j de M_j telle que $\tau'_j = (I_j \ldots v_j w_j \ldots O_j) = (I_i \ldots v_i w_j \ldots O_j)$ alors $\tau'_i < \tau'_j$, par conséquent $M_i < M_j$ (définition 20, items $5.ii$ et $5.iii$). Le cas opposé est caractérisé par la condition exprimée par la ligne (17).

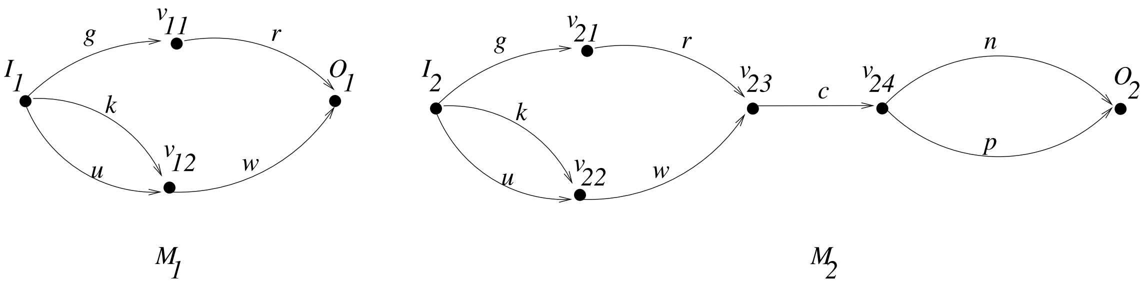


FIG. D.4 – *Exemple d'un cas exprimé par la ligne 14 de l'algorithme.*

Dans l'exemple illustré par la figure D.4, si l'on compare M_1 avec M_2, lorsque l'on compare les arcs partant de $V'_1 = \{v_{11}\}$ avec ceux partant de $V'_2 = \{v_{21}\}$ (tous les deux atteints, dans les respectifs E-tâche-graphes, par une sous-trajectoire $\tau'_1 = \tau'_2 = (g)$), on note que l'arc étiqueté par r partant de v_{11} atteint le sommet O_1 et donc $\tau''_1 = (gr)$ est en fait une trajectoire. Par contre, l'arc r partant de v_{21} atteint un sommet différent de O_2, ce qui signifie que $\tau''_2 = (gr)$ est une sous-trajectoire de M_2. Alors, il existe au moins une trajectoire en M_2 commençant par (gr). En l'occurrence, la première trajectoire de M_1 est $\tau_{11} = \tau''_1 = (gr)$ et celle de M_2 est $\tau_{21} = \tau'''_2 = (grcn)$ et $(gr) < (grcn)$. Par conséquent, $M_1 < M_2$ (définition 20, item $5.i$).

Lorsque c_i et c_j sont soit toutes le deux des sous-trajectoires, soit toutes les deux des trajectoires (condition exprimée par la ligne (20) de l'algorithme), il faut comparer les sommets atteints par ces (sous-)trajectoires. Le sommet atteint par une trajectoire est bien évidemment le somme O d'un E-tâche-graphe. La comparaison de O_i avec O_j a pour but de les marquer (ligne (40) et (41) de l'algorithme ***Comparer-sommets***, une fois qu'ils sont atteints.

Annexe E

Complexité des Algorithmes

E.1 Relation d'ordre lexicographique entre deux E-tâche-graphes multiplicatifs

La comparaison de deux E-tâche-graphes multiplicatifs $M_i = (V_i, X_i, I_i, O_i, \lambda_i, \mathcal{L})$ et $M_j = (V_j, X_j, I_j, O_j, \lambda_j, \mathcal{L})$ se fait par des appels récursifs croisés des algorithmes ***Comparer-sommets*** et ***Comparer-arcs*** ; le premier appel étant ***Comparer-sommets***(I_i, I_j). Cette comparaison utilise la technique d'exploration d'un graphe dénommé *recherche en profondeur d'abord* car on va aussi loin que possible à partir d'un sommet initial. Cette technique consiste en "visiter" un sommet v en G, parcourir un arc $(\overrightarrow{v,w})$ partant de v ; si w a déjà été visité, on choisi un autre arc partant de v. Par contre, si w n'a pas encore été visité, on le visite (on marque qu'il a été visité) et on recommence le processus récursivement sur w. Lorsque tous les arc partant de w sont visités, on prend le prochain arc $(\overrightarrow{v,w'})$ partant de v. La boucle est finie lorsque tous les arcs qui partent du premier sommet visité ont été parcourus. Le premier sommet visité par notre algorithme est le sommet I, qui correspond au début d'une trajectoire.

La visite à un sommet v n'est réalisée qu'une seule fois puisque lorsque l'on visite, on le marque et un sommet visité n'est plus exploité. La boucle de l'algorithme ***Comparer-sommets*** est contrôlée par les sommets du premier graphe, c'est-à-dire M_i. ***Comparer-sommets*** prend alors un temps $\mathcal{O}(|V_i|)$, où $|V_i|$ est le nombre de sommets de M_i. Dans le pire des cas, tous les arcs partant d'un sommet v_i de M_i sont également examiné (sauf s'il n'existe plus d'arc partant de v_j en M_j, v_j correspondant à v_i). Par conséquent, dans les pire des cas, tous les arcs de M_i sont également examiné. ***Comparer-arcs*** prend donc un temps total de $\mathcal{O}(|X_i|)$, où $|X_i|$ est le nombre d'arcs de M_i.

La comparaison de deux E-tâche-graphes multiplicatifs, quant à l'ordre lexicographique, prend donc un temps, dans le pire des cas, de l'ordre de $\mathcal{O}(|V_i| + |X_i|)$. Comme dans un E-tâche-graphe $|V| \leq |X| - 1$ (l'égalité pour une séquence), en utilisant la règle de sommation de $\mathcal{O}$, cet algorithme est en $\mathcal{O}(|X_i|)$.

E.2 Recherche d'un sous-E-tâche-graphe-additif

Cet algorithme analyse le sommet v de $G = (V, X, I, O, \lambda, \mathcal{L})$, donné comme paramètre d'entrée, et chaque sommet w de G qui est portée d'une sous-trajectoire passant par v tel que $d^+(w) > 1$ ou $d^-(w) > 1$ (ligne (7)). Pour chaque sommet w analysé, l'algorithme parcourt tous les arcs partant de v (ligne (6)), et chaque sommet n'est évalué qu'une seule fois puisqu'il est inclu dans la liste de sommets à visiter ($LIST$) qu'à la première fois qu'il est atteint (lignes (12) et (33)) et une fois inclu dans la liste il est marqué. Alors, l'algorithme parcourt chaque sommet et chaque arc du sous-E-tâche-graphe $G' = (V', X', I', O', \lambda', \mathcal{L})$ de G qu'il reconnaîtra. Par conséquent, ***Sous-E-tâche-graphe-additif*** prends un temps de l'ordre $\mathcal{O}(|V'| + |X'|) \leq \mathcal{O}(|V| + |X|)$. ***Sous-E-tâche-graphe-additif*** est donc en $\mathcal{O}(|X|)$.

Nota : Il n'est pas toujours évident, voire même impossible, identifier la valeur de $|V'|$ et de $|X'|$ d'un sous-E-tâche-graphe G' de G. Par conséquent, lorsque la complexité d'un algorithme est mesurée en fonction du nombre d'arcs d'un sous-E-tâche-graphe de G, nous avons décidé de l'exprimer en fonction du nombre d'arcs de G. Cela peut pénaliser la complexité de l'algorithme, certes, cependant la valeur $|X|$ est sans aucun doute connue.

E.3 Algorithme de différence

L'algorithme de différence entre deux E-tâche-graphes, $G_i = (V_i, X_i, I_i, O_i, \lambda_i, \mathcal{L})$ et $G_j = (V_j, X_j, I_j, O_j, \lambda_j, \mathcal{L})$, sous leurs formes normales respectives, évalue chaque arc partant de chaque sommet v_i de G_i, et ceux de son correspondant v_j en G_j ; v_i et v_j étant portées, dans les E-tâche-graphes respectifs, de sous-trajectoires communes aux deux E-tâche-graphes. Pour chaque arc $x_i = (\overrightarrow{v_i, w_i})$ de G_i, l'un des cas suivants est observé :

i) il existe un arc $x_j = (\overrightarrow{v_j, w_j})$ en G_j partant de v_j, étiqueté par la même étiquette de x_i, auquel cas l'algorithme *marque* simplement l'arc x_j en G_j (coût = 1) ;

ii) il n'existe pas d'arcs partant de v_j étiqueté par la même étiquette de x_i, auquel cas il faut *enlever* le plus grand sous-E-tâche-graphe (additif) $G'_i = (V'_i, X'_i, I'_i, O'_i, \lambda_i, \mathcal{L})$ de G_i ayant v_i comme sommet initial qui ne déconnecte pas v_i de O_i. Le coût de l'opération est égal au coût d'identifier G'_i et de l'enlever. Il vaut $2(\mathcal{O}(|X'_i|)) \leq 2(\mathcal{O}(|X_i|))$. L'opération est donc en $\mathcal{O}(|X_i|)$;

iii) Il existe un (ou plusieurs) arc(s) x_j partant de v_j en G_j lequel(lesquels) il n'y a pas d'arcs x_i partant de v_i en G_i tel(s) que $\lambda_i(x_i) = \lambda_j(x_j)$. Dans ce cas là, il faut *ajouter* dans G_i le sous-E-tâche-graphe (additif) $G'_j = (V'_j, X'_j, I'_j, O'_j, \lambda_j, \mathcal{L})$ de G_j ayant v_j comme sommet initial tel que O'_j est le premier sommet, atteint par les (sous-) trajectoires passant par x_j, qui peut également être atteint par d'autres (sous-)trajectoires qui ne passent pas par x_j. Le coût de l'opération est égal au coût d'identifier G'_j et de l'insérer dans G_i, en identifiant I'_j avec v_i. Il vaut $2(\mathcal{O}(|X'_j|)) \leq 2(\mathcal{O}(|X_j|))$. L'opération est donc en $\mathcal{O}(|X_j|)$;

iv) Il existe deux arcs x_i et x'_i partant de v_i et arrivant à un même sommet w_i tels que leur correspondants[1] dans G_j partent de v_j mais arrivent à des sommets distincts w_j et w'_j. Dans ce cas là, il faut *éclater* le sommet w_i en w_i et w'_i. L'éclatement d'un sommet w_i consiste en diriger l'arc $x_i = (\overrightarrow{v_i, w_i})$ vers un nouveau sommet w'_i et ajouter le sous-E-tâche-graphe $G'_i = (V'_i, X'_i, I'_i, O'_i, \lambda_i, \mathcal{L})$ de G_i ayant w_i comme sommet initial, G'_i est en réalité un produit de sous-E-tâche-graphes, additifs ou élémentaires, ($G'_{ij} = \prod_{l=1}^{k} G'_{il}$) tel que $O'_i = O'_{ik}$ est le premier sommet, atteint par les (sous-)trajectoires passant par x_i, qui peut également être atteint par d'autres (sous-)trajectoires qui ne passent pas par x_i ni par les arcs appartenant à G'_{i1}. Le coût de l'opération est égal au coût d'identifier G'_i et de l'insérer dans G_i, en identifiant I'_i avec v_i. Il vaut $\mathcal{O}(|X'_i|)) \leq \mathcal{O}(|X_i|)$. L'opération est donc en $\mathcal{O}(|X_i|)$.

Alors, pour chaque sommet v_i de G_i, l'algorithme est en $\mathcal{O}(d^+(v_i)max\{|X_i|, |X_j|\}) < \mathcal{O}(|X_i|\ max\{|X_i|, |X_j|\})$. Et la détermination de la différence entre deux E-tâche-graphes G_i et G_j prends donc un temps de l'ordre de $\mathcal{O}(|V_i|\ |X_i|\ max\{|X_i|, |X_j|\})$

Or, $|V_i|\ |X_i|\ max\{|X_i|, |X_j|\} = max\{|V_i|\ |X_i|^2,\ |V_i|\ |X_i|\ |X_j|\}$

Et $max\{|V_i|\ |X_i|^2,\ |V_i|\ |X_i|\ |X_j|\} \ <\ max\{|X_i|^3,\ |X_i|^2\ |X_j|\}$;

si $|X_i| < |X_j|$ alors $X_i{}^3 <\ |X_i|^2\ |X_j|$ et

$max\{|X_i|^3,\ |X_i|^2\ |X_j|\} = |X_i|^2\ |X_j| < |X_j|^3$;

si au contraire, $|X_i| > |X_j|$ alors $X_i{}^3 > |X_i|^2\ |X_j|$ et

$max\{|X_i|^3,\ |X_i|^2\ |X_j|\} = |X_i|^3$.

Alors, $max\{|V_i|\ |X_i|^2,\ |V_i|\ |X_i|\ |X_j|\} \ <\ max\{|X_i|^3, |X_j|^3\}$.

L'algorithme de différence est donc en $\mathcal{O}(max\{|X_i|^3, |X_j|^3\})$.

E.4 Forme normale

La forme normale d'une séquence S est S (propriété 11). Le coût de la détermination de la forme normale d'une séquence équivaut donc au coût de vérifier si un E-tâche-graphe G est une séquence. Or, le prédicat ***séquence*** (annexe D.3.5.4) dans le pire des cas parcourt tous les arcs d'un E-tâche-graphe G lorsqu'il est effectivement une séquence. Alors $n(S)$ est en $\mathcal{O}|X|$.

1. Les correspondants de x_i et x'_i dans G_j sont des arcs x_j et x'_j partant de v_j et étiquetés par $\lambda_j(x_j) = \lambda_i(x_i)$ et $\lambda_j(x'_j) = \lambda_i(x'_i)$.

Un E-tâche-graphe G qui n'est pas une séquence est soit un faisceau, soit un E-tâche-graphe sous une configuration complexe. Dans les deux cas, il faut tout d'abord mettre G en ordre lexicographique et ensuite le factoriser à gauche, tant qu'il est possible, puis le factoriser à droite. Nous allons évaluer ces trois étapes séparément pour les faisceaux et pour les E-tâche-graphes sous une configuration complexe. Et cela parce qu'on a des traitements différents, pour chacune de ces étapes, selon que l'E-tâche-graphe soit un faisceau ou un E-tâche-graphe sous une configuration complexe. Alors,

i) Si $G = (V, X, I, O, \lambda, \mathcal{L})$ est un faisceau, $G = F = \sum_{i=1}^{m} S_i$, où $S_i = (V_i, X_i, I, O, \lambda, \mathcal{L})$; $V_i \subset V$ et $V_j \subset V$; $X_i \subset X$ et $X_j \subset X$

• *Ordonner lexicographiquement un faisceau :* Mettre un faisceau en ordre lexicographique équivaut à ordonner lexicographiquement m séquences. Le coût de cette opération est donc $m^2(comparer\ \ S_i\ \ et\ \ S_j)$. Et comparer deux séquences S_i et S_j est en $\mathcal{O}(max\{|X_i|, |X_j|\}) < \mathcal{O}(|X|)$. Ainsi, mettre un faisceau en ordre lexicographique est en $\mathcal{O}(m^2|X|)$

Or $m = d^+(I) < |X|$. Ainsi l'opération d'ordonner lexicographiquement un faisceau est en $\mathcal{O}(|X|^2\ |X|) = \mathcal{O}(|X|^3)$.

• *Factoriser un faisceau à gauche* : L'algorithme ***factoriser-gauche-faisceau*** (D.3.6.1) ne compare que les arcs partant de I, dans l'ordre vertical. Alors, si x_i et x_{i+1} sont deux arcs consecutifs partant de I,

Pour le premier appel, les arcs x_i et x_{i+1} sont comparés et,

- soit $\lambda(x_1) = \lambda(x_2)$, auquel cas il faut fusionner x_1 et x_2 et faire un appel récursif à partir du premier arc partant de I. Cependant, lors de la fusion de deux arc, $d^+(I)$ se réduit d'une unité.

- soit $\lambda(x_1) \neq \lambda(x_2)$, auquel cas il faut appeler récursivement l'algorithme à partir du deuxième arc partant de I. Cela correspond à factoriser le sous-faisceau F' de F tel que $d^+(I') = d^+(I) - 1$.

Lors du deuxième appel, il n'y a que $d^+(I) - 1$ arcs à comparer. Et, encore une fois,

- soit les deux premiers arcs possèdent la même étiquette et il sont donc fusionnés, ce qui réduit la valeur de $d^+(I)$ d'une unité ;

- soit ils possèdent des étiquettes différentes et l'on factorise le sous-faisceau F' dont $d^+(I) = d^+(I) - 1$.

Or, à chaque appel le nombre d'arcs à comparer se réduit donc d'une unité. Et l'algorithme n'est appelé récursivement que si le deuxième arc comparé n'est pas le dernier arc partant de I. L'algorithme est donc appelé récursivement $n - 1$ fois. Si l'on considère que fusionner deux sommets a un cout égal à 1, ainsi que comparer deux arcs, l'algorithme ***factoriser-gauche-faisceau*** est donc en $\mathcal{O}(d^+(I) - 1)$. Comme $d^+(I) - 1 < |X|$ l'algorithme est en $\mathcal{O}(|X|)$.

• *Factoriser un faisceau à droite* : Pour factoriser un faisceau à droite, avant d'utiliser l'algorithme ***factoriser-droite-faisceau*** (D.3.6.2), il faut d'abord mettre le faisceau en ordre lexicographique dans le sens inverse. Autrement dit,

- ***mirroir***(G) inverse le sens des arcs d'un E-tâche-graphe et transforme le sommet I en O et vice-versa. Cet opération est donc en $\mathcal{O}(|X|)$. Si F' est le faisceau résultat de l'opération ***mirroir*** (***lexicographique*** (***mirroir*** (F))) alors F' correspond à F ayant les séquences ordonnées lexicographiquement dans le sens O vers I. Sachant que mettre un faisceau en ordre lexicographique est en $\mathcal{O}(|X|^3)$, l'obtention de F' est en $\mathcal{O}(|X|^3)$.

- l'algorithme ***factoriser-droite-faisceau*** suit le même raisonnement que ***factoriser-gauche-faisceau***. Il est donc en $\mathcal{O}(|X|)$.

Ainsi, factoriser un faisceau à droite est donc en $\mathcal{O}(|X|^3)$.

ii) Si $G = (V, X, I, O, \lambda, \mathcal{L})$ est sous une configuration complexe, $G = \sum_{i=1}^{m} \prod_{i=1}^{n_i} Gij = \sum_{i=1}^{m} M_i$ où $G_{ij} = (V_{ij}, X_{ij}, I_{ij}, O_{ij}, \lambda, \mathcal{L})$; $V_{ij} \subset V$ et $X_{ij} \subset X$

• *Ordonner lexicographiquement un E-tâche-graphe G sous une configuration complexe :* Si G est additif (c'est-à-dire $m > 1$) d'après la définition 19, G est en ordre lexicographique si chacun de ses composants l'est et $M_i < M_{i+1}\ \forall\ i = 1, 2, \ldots, m-1$.

Alors, mettre un E-tâche-graphe en ordre lexicographique équivaut à comparer m E-tâche-graphes multiplicatifs, eux-mêmes en ordre lexicographique. Le coût de cette opération est donc $m^2(comparer\ M_i\ et\ M_j)$. Et comparer deux E-tâche-graphes multiplicatifs M_i et M_j est en $\mathcal{O}(|X_i|) < \mathcal{O}(|X|)$ (annexe E.1). Par conséquent, mettre un E-tâche-graphe en ordre lexicographique est en $\mathcal{O}(m^2|X|)$

Or $m = d^+(I) < |X|$. Ainsi l'opération *ordonner lexicographiquement un E-tâche-graphe* est en $\mathcal{O}(|X|^2\ |X|) = \mathcal{O}(|X|^3)$.

De plus, un E-tâche-graphe multiplicatif M_i est en ordre lexicographique si chacun des G_{ij} l'est. Ainsi, mettre un E-tâche-graphe multiplicatif M_i en ordre lexicographique correspond à ordonner lexicographiquement chacun des $n_i\ G_{ij}$. Cet opération est dans le pire des cas pour chaque G_{ij} en $\mathcal{O}(|X_{ij}|^3)$. Comme $|X_{ij}| < |X|$, alors ordonner lexicograflquement un est E-tâche-graphe multiplicatif M_i est en $\mathcal{O}(n_i|X|^3)$. Finalement, $n_i < |X|$. Alors, ordonner lexicographiquement un E-tâche-graphe multiplicatif est donc en $\mathcal{O}(|X|^4)$.

• *Factoriser à gauche un E-tâche-graphe G sous une configuration complexe* : L'algorithme ***factoriser-gauche*** (D.3.6.3) ne compare que les arcs partant de I, dans l'ordre vertical. Alors, si x_i et x_{i+1} sont deux arcs consecutifs partant de I,

Pour le premier appel, les arcs $x_1 = (\overrightarrow{I, v_1})$ et $x_2 = (\overrightarrow{I, v_2})$ sont comparés. Tout d'abord on vérifie s'ils font partie d'un même composant additif, c'est-à-dire si le premier sommet v, portée de la première trajectoire passant par x_1 tel que $d^-(v) > 1$ est également le premier sommet v portée de la première trajectoire passant par x_2 tel que $d^-(v) > 1$. Si c'est le cas, on compare ensuite les étiquettes de x_1 et x_2 ; et :

- soit $\lambda(x_1) = \lambda(x_2)$ auquel cas il faut d'abord fusionner ces deux arcs. Cette opération est en $\mathcal{O}(2d^+(v_2) + 1)$ (car fusionner les deux arcs $x_1 = (\overrightarrow{I, v_1})$ et $x_2 = (\overrightarrow{I, v_2})$ correspond à remplacer chaque $(\overrightarrow{v_2, \vec{u}})$ par un nouveau arc $(\overrightarrow{v_1, \vec{u}})$ et à la fin enlever $(\overrightarrow{I, v_2})$). Ensuite faire un appel récursif à partir du premier arc partant de I. Cependant, lors de la fusion de ces deux arcs, $d^+(I)$ se réduit d'une unité.

- soit $\lambda(x_1) \neq \lambda(x_2)$ et

◦ si $v = O$, c'est-à-dire aucune des sous-trajectoires passant par x_1 et x_2 possèdent des portées communes (alors aucun fusionnement n'est possible). Dans ce cas là, on fait appel récursif de l'algorithme à partir du deuxième arc partant de I. Cela correspond à factoriser un sous-graphe G' de G tel que $d^+(I') = d^+(I) - 1$.

◦ si au contraire $v \neq O$, c'est-à-dire v est portée de deux sous-trajectoires distinctes (l'une passant par x_1 et l'autre par x_2), alors on cherche dans G' (le "mirroir" de G) le sous-E-tâche-graphe additif G'_1 ayant v comme sommet initial. Cette opération est en $\mathcal{O}(|X|)$. Ensuite on fait un appel récursif de l'algorithme, cette fois-ci indiquant qu'il faut considérer le sous-E-tâche-graphe additif G'_1.

Si par contre les arcs $x_1 = (\overrightarrow{I, v_1})$ et $x_2 = (\overrightarrow{I, v_2})$ ne font pas partie du même composant additif, on fait un appel récursif de l'algorithme à partir du deuxième arc partant de I, ce qui correspond à la factorisation d'un sous-graphe G' de G, tel que $d^+(I') = d^+(I) - 1$.

Pour le deuxième appel,

$a)$ si $G'_1 \neq \emptyset$, il faut comparer x'_1, le premier arc partant de I'_1, avec l'arc $x_{d^+(I')+1}$ partant de I.

- si $\lambda(x'_1) = \lambda(x_{d^+(I')+1})$ alors identifie le sous-E-tâche-graphe additif G' de G qui contient $x_{d^+(I')+1}$ et on le compare avec G'_1 ;

◦ s'ils sont égaux, il faut les fusionner. Le coût de ces opérations est :

mirroir $\rightarrow \mathcal{O}(|X|)$ +

sous-E-tâche-graphe-additif $\rightarrow \mathcal{O}(|X|)$ +

mirroir $\rightarrow \mathcal{O}(|X|)$ +

comparer G' et $G'_1 \rightarrow \mathcal{O}(|X|)$ +

fusionner G' et $G'_1 \rightarrow \mathcal{O}(|X|)$ +

enlever $G' \rightarrow \mathcal{O}(|X|)$.

Par la règle de sommation, le côut total est $\mathcal{O}(|X|)$.

◦ s'ils sont différents, aucun fusionnement n'est possible et on fait un appel récursif de l'algorithme en considérant les arcs suivants, dans l'ordre vertical, partant de I.

$b)$ si $G'_1 = \emptyset$, la procédure est identique au premier appel.

Alors, le premier appel de ***factoriser-gauche***, ainsi que les autres appels, est en $\mathcal{O}(|X|)$. Dans le pire des cas, on fait $d^+(I)$ appels récursifs. Ainsi ***factoriser-gauche*** est en $\mathcal{O}(d^+(I)\ |X|)$. Comme $d^+(I) < |X|$, ***factoriser-gauche*** est donc en $(|X|^2)$.

• *Factoriser à droite un E-tâche-graphe G sous une configuration complexe* : L'algorithme ***factoriser-droite*** (D.3.6.4) fait appel à ***factoriser-gauche*** en utilisant le "mirroir" de G ordonné lexicographiquement dans le sens O vers I. Ordonner lexicographiquement est l'algorithme le plus coûteux parmi ceux que ***factoriser-droite*** fait appel, (il est en $\mathcal{O}(|X|^4)$; ***mirroir*** est en $\mathcal{O}(|X|)$ et ***factoriser-gauche*** est en $\mathcal{O}(|X|^2)$. Alors, ***factoriser-droite*** est en $\mathcal{O}(|X|^4)$.

Ainsi, parmi les algorithmes auxiliaires de ***normal*** (G), le plus coûteux est ***factoriser-droite*** qui est en $\mathcal{O}(|X|^4)$. Pour chaque appel de ***normal*** (G), ***factoriser-droite*** n'est appelé qu'une seule fois. Par conséquent, chaque appel de ***normal*** (G) est, dans le pire des cas, en $\mathcal{O}(|X|^4)$.

Les appels récursifs de ***normal*** (G) se fait sur des E-tâche-graphes plus petits que G, à chaque fois. Ainsi, ***normal*** (G) est en $\mathcal{O}(|X|^4)$.

Bibliographie

[All93] D. Allemang et G. Heijst, Generic task in KEW. In *Knowledge Acquisition for Knowledge-Based Systems (7^{th} European Workshop, EKAW'93)*, édité par N. Aussenac, G. Boy, B. Gaines, M. Linster, J. Ganascia et Y. Kodatroff, LNAI 723, Septembre 1993, pp. 139–158.

[Bae87] J. Baeten, J. Bergstra et J.W.Klop, Term rewriting systems with priorities. In *Rewriting Techniques and Applications*, édité par G. Goos et J. Hartmanis, Spring-Verlag LNCS 256, Berlin, Mai 1987, pp. 83–94.

[Bou97] V. Bouchitté et J.-X. Rampon, On-line algorithms for orders. *Theoretical Computer Science*, vol. 175, 1997 pp. 225–238.

[Cha92] B. Chandrasekaran, T. Johnson et J. Smith, Task-structure analysis for knowledge modeling. *Communications of the ACM*, vol. 35, no. 9, Septembre 1992 pp. 124–137.

[Cha93a] M. Chaillot, Une architeture de contrôle réactif pour la résolution coopérative de problèmes. Thèse de Doctorat, INPG (Institut National Polytechnique de Grenoble), Septembre 1993.

[Cha93b] B. Chandrasekaran et T. Johnson, *Generic Tasks and Task Structures: History, Critique and New Directions*. Springer-Verlag, 1993, pp. 232–272.

[Che93] F. Chevenet et J. W. F. Jean-Marie, Slot: a cooperative problem-solving environment in explanatory data analysis. In *Actes 49^{th} session of the International Statistical Institute*, Firenze, 1993, pp. 225–256.

[Che94] F. Chevenet, Environnement coopératifs de résolution de problèmes pour l'analyse statistique: représentation objets et analyse de donées multivariées en écologie. Thèse de Doctorat, Université Claude-Bernard, Lyon, 1994.

[Cle90] V. Clément, Raisonnements cognitifs appliqués au pilotage d'algorithmes de traitement d'images. Thèse de Doctorat, Université de Nice, Janvier 1990.

[Cra94] I. Crampé, Sémantique des tâches décomposables et spécialisables. Rapport de DEA d'informatique, Université Joseph Fourier, Grenoble, Juin 1994.

[Cle82] M.-A. Clemente-Salazar, Études et algorithmes liés à une nouvelle structure de données en T.A: Les E-graphes. Thèse de Doctorat, Université Scientifique et Médicale de Grenoble, Mai 1982.

[Del93] I. Delouis, LISA - un langage réflexif pour la modélisation du contrôle dans les systèmes à base de connaissances. - Application à la planification des réseaux életriques. Thèse de Doctorat, Université de Paris-Sud - Centre d'Orsay, Paris, Juin 1993.

[Der90] N. Derschowitz et J. Jouannand, *Rewrite Systems*, vol. B. Handbook of Theoretical Computer Science, 1990, pp. 243–320.

[Duf65] R. J. Duffin, Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, vol. 10, 1965 pp. 303–318.

[Eri95] H. Eriksson, Y. Shahar, S. Tu, A. Puerta et M. Musen, Task modeling with reusable problem-solving methods. *Artificial Intelligence*, vol. 79, no. 2, , Décembre1995 pp. 293–326.

[Fin96] L. Finta, Z. Liu, I. Milis et E. Bampis, Scheduling uet-uct serie-parallel graphs on two processors. *Theoretical Computer Science*, vol. 162, 1996 pp. 323–340.

[Had96] A. Jadj-Kacem, Les SBC coopératifs, vers une architecture fondée sur une formalisation du contrôle. In *RFIA'96 - Reconnaissance des Formes et Intelligence Artificielle*, vol. II, Rennes, Janvier 1996, pp. 964–972.

[Hai98] P. Haïk, F. Charpillet et A. Jaussellin, Modèles de tâches et bibliothèques de programmes intelligentes pour l'interprétation de signaux & le traitement du signal. In *La Lettre de L'IA*, no. 134-135-136, , mai juin juillet août 1998, pp. 16–20.

[Ham93] R. Hamadi, Similarité de raisonnement à l'aide de graphes. Rapport de DEA en recherche opérationnelle, Université Joseph Fourier, Grenoble, Juillet 1993.

[Lev91] R. Levinson, Pattern associativity and the retrieval of semantics networks. In *Special Issue of Computers an Mathematics on Semantic Networks in Artificial Intelligence*, édité par F. Lehmann, Pergamon Press, 1991.

[Med95] C. Médigue, T. Vermat, G. Bisson, A. Viari et A. Danchin, Cooperative computer system for genome sequence analysis. In 3^{rd} *ISMB*, 1995, pp. 249–258.

[Moi97] S. Moisan, R. Vincent et M. Thonnat, Program supervision: from knowledge modeling to dedicated engines. Tech. Rep. 3324, INRIA - Sophia Antipolis, Décembre 1997.

[Mol95] M. Molina et J. Cuena, Knowledge oriented and object oriented design: The experience of KSM. In *Proceedings of the* 9^{th} *Knowledge Acquisition for Knowledge-based Systems Workshop (KAW'95)*, 1995. http://www.isys.dia.fi.upm.es/Publications.html.

[Mol96] M. Molina, Y. Shahar, J. Cuena et M. Musen, A structure of problem-solving methods for real-time decision support: Modeling approaches using PROTÉGÉ-II and KSM. In 10^{th} *Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, edité par B. Gains et M. Musen, Canada, 1996.

[Mon96] F.-R. Monclar, ELICO pour le développement de systèmes à base de connaissance coopératifs. In *RFIA'96 - Reconnaissance des Formes et Intelligence Artificielle*, vol. II, Rennes, Janvier 1996, pp. 973–982.

[Mor96] J.-Y. Morneau, Conception et mise en œuvre d'interfaces adaptables pour un système à base de connaissances: application au diagnostic en électromyographie. Mémoire d'ingénieur CNAM, Grenoble, 1996.

[Par98] T. Parmentier, D. Ziebelin et F. Rechenmann, Environnement de résolution de problèmes distribué. In *RFIA'98 - Reconnaissance des Formes et Intelligence Artificielle*, vol. II, Clermont Ferrand, Janvier 1998, pp. 265–299.

[Pie91] C. Pierret-Golbreich, Vers une nouvelle architecture pour l'explicitation du raisonnement au niveau connaissances: TASK. Tech. Rep. RR674, Universtité Paris-Sud (Orsay) - LRI, Juin 1991.

[Pre91] M. Prévosto et F. Rechenmann, Said: a knowledge-based system in signal processing. In *Actes de la conférence européene d'automatique*, Grenoble, Juillet 1991.

[Pue93] A. Puerta, S. Tu et M. Musen, Modeling tasks with mechanisms. *International Journal of Intelligent Systems*, vol. 8, no. 1, 1993 pp. 129–152.

[Rem82] J. Rémy, Étude des systèmes de réécriture conditionnels er applications aux types abstraits algébriques. Thèse de Doctorat, Nancy, Juin 1982.

[Rou88] B. Rousseau, Vers un environnement de résolution de problèmes en biométrie. Apport des techniques de l'intelligence artificielle et de l'interaction graphique, Université Claude Bernard - LYON I, 1988.

[Sch94] G. Schreiber, B. Wielinga, R. Hoog, H. Akkermans et W. Velde, CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert (Intelligent Systems and their Applications)*, vol. 9, no. 6, Décembre 1994 pp. 28–37.

[Val82] J. Valdes, R. Tarjan et E. Lawler, The recognition of series parallel digraphs. *SIAM Journal of Computation*, vol. 11, no. 02, Mai 1982 pp. 298–313.

[Tor98] L. Torres et C. Frydman, Verifying and validiting specifications of knowledge-based systems. In *ECAI'98 (13^{th} European Conference on Artificial Intelligence)*, Août 1998.

[Wie92] B. Wielinga, W. V. de Velde, G. Schreiber et H. Akkerman, The KADS knowledge modelling approach. In *Proceedings of the 2^{nd} Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*, édité par R. Mizoguchi, H. Motoda, J. Boose, B. Gaines et R. Quinlan, Hitachi, Advanced Research Laboratory, Hatoyama, Saitama, Japan, 1992, pp. 23–42. http://swi.psy.uva.nl/projects/CommonKADS/abstracts/Wielinga:92c.html.

[Wil94] J. Willamowski, Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur. Thèse de Doctorat, Université Joseph Fourier, Grenoble, 1994.

[Zie93] D. Ziébelin, MYOSYS: système de diagnostic électromyographique utilisant une représentation de connaissance par objets et tâches. In *Bulletin de l'AFIA*, vol. 14, Juillet 1993.

Index