

UNIVERSITE TOULOUSE III - PAUL SABATIER
U.F.R Mathématique et Informatique pour la Gestion

THESE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE DE TOULOUSE III

Discipline: Informatique

présentée et soutenue

par

Pierre Gradi

le 21 Décembre 2000.

Spécifications et Conceptions d'Applications Coopératives

Directeur de thèse : Michel Diaz

JURY

| | | |
|-------------------|--|------------|
| Michel Bauderon | Professeur à l'Université Bordeaux I | Rapporteur |
| Claude Kirchner | Directeur de Recherche INRIA | Rapporteur |
| Luc Bougé | Professeur à l'ENS Lyon | Examineur |
| Sergueï Soloviev | Professeur à l'Université Toulouse III | Examineur |
| François Vernadat | Maître de Conférence INSA Toulouse | Examineur |
| Michel Diaz | Directeur de Recherche du CNRS | Directeur |

Introduction

La coopération inter-humaine n'est pas un sujet propre à l'informatique, et il est permis de penser qu'il s'agit d'une activité humaine fondamentale. Ainsi, après la diffusion soudaine et massive des technologies de communication et d'un réseau de communication permettant à tout couple de machines d'échanger des données, quelle que soit la distance les séparant, la question de la coopération sur ce nouveau support est naturellement devenue un sujet de recherche pour la communauté informatique. Dans le cadre du groupe OLC, au sein duquel je travaille depuis mars 1994, la coopération est apparue comme mot clé dans le cadre du projet CNET/CNRS CESAME. Ce projet avait pour objet la conception formelle de systèmes haut-débits multimédias coopératifs. Dans ce projet, la coopération était abordée, du point de vue de la communication, comme la définition de services et de protocoles pour la communication et la gestion de groupes. Par la suite [DGV96], ce type de services a été regroupé dans le terme de "coordination". Dans ce contexte, la problématique de la coopération est devenue *comment concevoir, spécifier et réaliser des activités informatiques coordonnées*. Dans notre travail, nous nous plaçons d'emblée dans un cadre où il n'y a pas de litiges et où l'ensemble des personnes impliquées est considéré de "bonne volonté". C'est dans ce cadre, volontairement restrictif, que nous interpréterons la définition suivante, inspirée de [Mar64]: *lorsque différents utilisateurs participent à un même processus d'élaboration ou à des processus connexes, le travail est de nature coopérative*.

Cette définition contient un terme de nature topologique: "connexe". Ce terme fait écho à un mot important dans le monde du réseau: "connecté". La connexion est nécessaire à la coopération et un certain nombre de protocoles de coordination permettent aux utilisateurs d'accéder à des fonctionnalités d'envoi (smtp) et de publication (http, news), et de se retrouver dans des localités virtuelles (irc, icq). Mais cette coordination n'est pas suffisante pour appréhender des "processus connexes", il faut inclure des formes de connexion temporelle (causalité, planification, historique) et de connexion spatiale ou architecturale (hiérarchie, répartition des ressources). Conceptuellement, le point de vue adopté dans [DGV96] consiste à représenter un système coopératif comme un ensemble dynamique d'agents réactifs reposant sur des services de coordination régulant leurs interactions. Pour pouvoir spécifier ces systèmes, une abstraction des services de coordination est réalisée. Une instance reconnue de ce type de systèmes sont les systèmes multi-agents. Dans cette approche un ensemble d'agents, disposant chacun d'une boîte aux lettres identifiée, échangent des messages. Le protocole de coordination (ici l'échange de message) est réduit, dans ce modèle, à la spécification du service rendu. Ils sont dotés d'un comportement et peuvent créer d'autres agents et se détruire. Des liens de connaissance (appelés accointances) relient les différents agents entre eux. Une spécificité de ces systèmes est la forte interdépendance entre l'architecture et sa possible évolution et le comportement des agents impliqués. En effet, le comportement de chaque agent peut être contraint par sa situation dans le système (e.g. la disponibilité d'une ressource). De même, le comportement d'un agent peut amener des modifications de l'architecture, ne serait-ce que par son

possible départ du système. Notre concept d’“architecture dynamique d’éléments éphémères” formera notre point de départ. Il est présent à tous les niveaux (connexion, coordination, coopération), ce niveau étant défini par les abstractions réalisées sur les services sous-jacents. Dans ce cadre général abstrait, la spécificité critique de la coopération est de ne plus pouvoir accepter comme hypothèse de travail la permanence des éléments impliqués.

La difficulté de ce sujet réside dans sa nouveauté, notre travail suit donc trois axes complémentaires : l’explicitation de concepts spécifiques du domaine, la recherche de formalismes adéquats, et la description et spécification d’exemples paradigmatiques. L’originalité de notre travail est de maintenir ces trois axes en coexistence. Notre travail s’appuie sur l’analyse de quatre sources différentes : les systèmes coopératifs [EGR91, DV94, Aal97, Scu99], les réseaux de Petri [Bra83, Rei85a, Val98, Val78], la réécriture de termes [Mes92, BN98, MT99, Cir00] et les grammaires de graphes [KLG93, CMR⁺97, CH98, Tae96, BC87]. En explorant notre domaine applicatif en suivant les trois axes, notre démarche permet d’importer de différentes littératures des concepts, des exemples et des formalismes. De cette synthèse multi-formalismes, nous dégagerons un cadre général pour concevoir et spécifier les applications coopératives.

Concepts spécifiques Nous proposons le concept d’“*architecture dynamique d’éléments éphémères*” pour décrire la structure des applications coopératives. Comment spécifier spatialement cette structure ? Les formalismes utilisés proposent deux structures complémentaires : terme ou graphe. La différence fondamentale entre ces deux structures réside dans la manière de référencer les éléments constitutifs de l’architecture. Dans les termes, la référence est implicitement définie par la position dans le terme et dans les graphes, elle est explicitée par le nom des sommets du graphe. Pour détecter à la conception quelle structure de spécification est adéquate, nous proposons de caractériser les propriétés devant être associées aux éléments pour réaliser la fonction du système : une propriété exclusive est appelée une identité, une propriété partagée est appelée une localité et nous appelons multiplicité l’effectif d’une localité. A chaque concept, on peut associer une opération critique que l’on doit pouvoir associer aux événements, diffusion pour la localité, apparition pour l’identité, addition pour la multiplicité. Si nous projetons ces trois concepts sur les formalismes que nous étudions, nous remarquons que la multiplicité est traitée par tous les formalismes. Si le problème ne s’exprime qu’avec des multiplicités, les Réseaux de Petri sont adaptés. Ensuite, si on considère aussi des localités, la réécriture de termes est un formalisme adapté. Enfin si en plus de la multiplicité, on utilise des identités alors les grammaires de graphes sont adéquates.

En nous tournant maintenant vers la structure temporelle du système que nous appellerons le comportement du système, nous avons importé de la réécriture les deux qualificatifs de “terminant” et de “confluent” qui s’appliquent aux comportements et défini quatre types de systèmes, de non-terminant/non-confluent à terminant/confluent. Il s’avère que, pour les applications coopératives, les quatre types sont présents. Pour notre cas d’étude, l’édition des actes d’une conférence, la remarque est encore plus frappante, car étant découpé en phases successives, les différentes phases parcourent les quatre types. Les réseaux de Petri sont le plus souvent utilisés pour spécifier des systèmes non “bloquants” (i.e. non terminants). En l’absence de terminaison, la notion de résultat n’a plus de sens, ainsi pour caractériser les propriétés attendues du système, on procède par génération du “graphe des marquages accessibles” (*de facto* une relation étiquetée) de préférence au langage du réseau. Sur ce “graphe”, on utilise des outils de model-checking pour valider les propriétés attendues. La Logique de réécriture, que l’on peut appréhender comme une sémantique de la réécriture de termes, raffine la relation d’accessibilité

en un graphe état/événement où les événements sont des “arêtes” ayant d’ uniques états source et cible.

Cadre Sémantique unifié Nous avons défini pour chaque formalisme une sémantique par entrelacement qui à chaque système associe un graphe état/événement, où chaque événement a un état source et un état cible. Ce type de sémantique est très proche de la Logique de Réécriture, en conséquence l’adaptation que nous proposons est simple : filtrer les événements concurrents ou composés pour obtenir la sémantique souhaitée. Pour les réseaux de Petri nous ajoutons, au nom de la transition tirée, le contexte où le tir de la transition a eu lieu. C’est pour les grammaires de graphes que notre travail a été le plus important. La sémantique originale est une sémantique à base de séquences de dérivation (i.e. de tir) [CMR⁺97]. Nous avons défini une sémantique abstraite où les états sont les graphes abstraits (classe de graphes modulo isomorphisme [BC87]) et les événements des classes de diagrammes d’instanciation des règles. A l’origine motivée par la nécessité d’obtenir un graphe état/événement, cette sémantique présente un avantage essentiel : elle est définie indépendamment de la structure de graphe considérée.

Ainsi pour les trois formalismes :

- un système est la donnée d’une paire $\langle S; R \rangle$ où S décrit une structure considérée et R un ensemble de règles, qui sont des paires d’éléments construits avec la structure.
- la sémantique est une fonction qui à chaque système associe un graphe état/événement.

Muni de ce cadre sémantique unifié on peut poser et résoudre complètement les problèmes relatifs à l’extension d’un formalisme. Il était un fait admis que les réseaux de Petri étaient traduisibles en Grammaires de Graphe [Cor98] ou en Logique de Réécriture [Mes92]. Notre cadre sémantique unifié permet de caractériser la nature de l’équivalence et de la prouver, ce qui n’avait pas été réalisé jusqu’alors. Disposer de ce cadre sémantique nous a ensuite permis de déterminer quelles étaient les théories de réécriture ou les grammaires de graphes pouvant être traduites en des réseaux de Petri à comportement isomorphes. Pour les Grammaires de graphes, cela simplifie la caractérisation de [Cor98]. Pour la réécriture de termes, ceci fournit une caractérisation compacte des réseaux “colorés” [Jen96, Gen86].

Applications Les applications irriguent l’ensemble du document par les besoins qu’elles véhiculent quant à la spécification des systèmes et la caractérisation de leurs propriétés attendues. Les deux premiers exemples sont utilisés dans la phase exploratoire de notre travail, les deux autres sont des jalons permettant d’illustrer le domaine réservé de la Logique de réécriture et des grammaires de graphes étendues.

Le problème des philosophes couvre une bonne partie des difficultés et pouvait être considéré comme faisant partie de la thématique : “Des philosophes pensent et mangent du riz autour d’une table en partageant une baguette avec chacun de leurs deux voisins”. A cet énoncé classique de Dijkstra [Dij71], on ajoutera [KLG93] la possibilité pour un philosophe de quitter la table avec une baguette, ainsi que la possibilité symétrique de s’insérer dans une table . Nous montrerons que nous pouvons découper ce problème (appelé le “gala”) en deux sous-problèmes (appelés respectivement le “pique-nique” et le “bal”) qui sont tous les deux représentables par des Réseaux de Petri Place/Transition, mais le problème complet, le “gala”, passe nécessairement par l’adoption d’un paradigme plus général.

L’organisation d’une conférence est notre cas d’étude. Le protocole décrit correspond aux services actuellement utilisés dans ce contexte. Ce cas d’étude permettra de valider notre stratégie de conception et mettra en valeur des

possibilités intéressantes de la réécriture de termes dans notre domaine d'application et d'interprétation. En outre, la possibilité de travailler sur un support textuel est critique pour la réalisation d'un cas d'étude aussi complexe;

Le `TCPTokenBucket`, un “seau à jetons” dont le débit en sortie est dynamique (à la mode TCP) est spécifiable en réécriture de termes. Il n'admet pas d'expression simple en Grammaires de Graphes. Il n'est pas réalisable en Réseaux de Petri Place/Transition et nécessite d'utiliser des réseaux auto-modifiants [Val78];

La formation d'apartés dans des groupes coopératifs est un problème important de notre domaine d'application. Il a été identifié par [DV94] à la fin du projet CESAME. Ce problème, l'admission dans des sous-groupes dynamiques pouvant se recouvrir partiellement, n'admet pas d'expression simple ni en Logique de Réécriture ni en Grammaires de Graphes. Pour mener à bien cette spécification, nous avons eu recours à notre généralisation des Grammaires de graphes, en l'occurrence l'utilisation de grammaires de “graphes de graphe”. Cette dernière possibilité met en valeur une capacité nouvelle de notre extension des grammaires de graphes : la possibilité de détailler des relations arbitrairement complexes entre des éléments, ce qui est une capacité essentielle dans la spécification d'applications coopératives.

Plan du mémoire Le premier chapitre détaille le cas d'étude et présente les trois formalismes étudiés en utilisant les problèmes des philosophes comme fil conducteur.

Le deuxième approfondit la sémantique des grammaires de graphes pour rendre cette sémantique proche de celles utilisées dans les deux autres formalismes, il détaille ensuite une algèbre de structures dont un terme permet de réaliser la spécification d'apartés dans des groupes coopératifs.

Le troisième chapitre tire partie de ce travail unificateur pour comparer les trois formalismes et détailler les traductions possibles et sans équivoque des uns dans les autres.

Le dernier chapitre regroupe d'une part la spécification du cas d'étude et la construction en MAUDE d'un prototype de simulateur. Ce simulateur permet de simuler “pas à pas” une théorie de réécriture, offrant les fonctionnalités classiques des simulateurs de réseaux de Petri dans le cadre, plus général, de la réécriture de termes.

Chapitre

1

Etat de l'art

Quelle forme la représentation de la coopération peut-elle prendre? Cette question a deux dimensions: d'une part, une dimension spatiale ou architecturale (comment décrit-on un système?), d'autre part, une dimension temporelle ou comportementale (comment représente-t-on l'évolution du système considéré?). Ces deux questions sont étroitement liées mais séparables [Sch93]. Notre premier souci est de qualifier les systèmes coopératifs à la fois spatialement et temporellement. Le système de classification reconnu d'un système coopératif [EGR91, Vil95] s'attache à des définitions spatiales et repose sur deux paires de qualificatifs opposés: Synchrone/Asynchrone qui caractérise le type de communication, Local/Distant qui caractérise la répartition géographique des participants. Nous allons adjoindre à cette qualification deux autres dimensions qui sont de nature comportementale ou causale et proviennent du vocabulaire de la réécriture: la terminaison, qui caractérise qu'un système finit par s'arrêter, la confluence qui stipule qu'aucune divergence (ou branchement) n'est irréversible.

Parmi les formalismes utilisés pour ce type d'approche, nous nous sommes d'abord concentrés sur les Réseaux de Petri. Les réseaux de Petri sont composés de deux parties distinctes, une topologie statique (i.e. le réseau) et un étiquetage dynamique (i.e. le marquage). La règle de tir définit la relation entre cette topologie et la dynamique du marquage. Les places du réseau définissent des localités et nous étudions l'évolution des effectifs d'éléments ayant telle ou telle *localité*, ce que nous appellerons leur *multiplicité*. Ce formalisme ne permet pas de tenir compte du concept d'*identité* car les jetons sont anonymes: ils n'ont pas d'identité [CM92]. Ainsi lorsque l'on souhaite représenter une architecture de composants dotés chacun d'un comportement, on duplique les réseaux des composants, et les liens relatifs à l'architecture sont diffus dans la structure finale du réseau. Cela forme très vite des réseaux dépassant le principal critère de lisibilité: l'absence de croisements de flèches. Il en résulte une confusion, aggravée par le mélange dans une seule représentation de deux notions topologiques: le comportement (i.e. topologie du temps) et l'organisation (i.e. topologie de l'espace).

Les réseaux de Petri de Haut-Niveau permettent de réaliser cette séparation en codant dans le réseau une topologie par l'utilisation de jetons étiquetés par

des paires de références (Cf. les Trains de [Gen86]). Mais ces références ne sont pas des identités : rien n'interdit à deux jetons d'avoir la même "couleur". Si nous considérons ces "couleurs" comme des "identités", alors une personne déjà présente dans le système pourrait y entrer à nouveau. Cette situation d'ubiquité, étrange dans le monde réel, a un sens dans le monde informatique. Cette possibilité est mise en valeur par notre cas d'étude où un membre du comité de lecture peut être un auteur. Pour prendre en compte la diversité des Réseaux de Petri de Haut-Niveau, nous considérons la Logique de Réécriture et montrons qu'elle offre, pour ces Réseaux, un cadre formel unifiant. Dans ce contexte, le problème de l'apparition de "nouvelles références" se confond avec celui des "variables libres à droite" dans une règle de réécriture.

Il existe un formalisme qui ne considère *que* des identités, ce sont les Grammaires de Graphes. Dans ce formalisme, tous les éléments, sommets et arêtes, sont identifiés. En conséquence, même l'apparition d'une arête pose le problème de l'apparition d'une référence. Ce problème est résolu par une construction : la somme amalgamée. La somme amalgamée est une union dont on spécifie l'intersection. Si des éléments ont même nom mais ne sont pas considérés comme "identiques" dans la spécification de l'intersection, ils auront des noms différents dans l'union. Les grammaires de graphes perdent en contrepartie la notion de localité. Une faiblesse reconnue de ce type de formalisme, l'impossibilité de dupliquer des sous-graphes, participe pour nous de la non prise en compte de la localité dans ce formalisme.

Pour illustrer notre problématique, nous développons un cas d'étude et des exemples. Notre cas d'étude développe une situation réelle complète : l'organisation d'une conférence. L'énoncé original [RV99] ouvre notre état de l'art et permettra de présenter les différentes classifications des applications coopératives sur un exemple concret. Sa spécification sera donnée dans la section 4.1. A côté de ce système complexe, les exemples servent d'outils pédagogiques pour éclairer des points techniques de notre propos. Nous avons détaillé deux exemples, le premier sert de fil conducteur de ce chapitre. Il s'agit d'une variation [KLG93] sur le problème des philosophes [Dij71] qui nous permettra d'illustrer la difficulté de représenter conjointement le dynamisme et la topologie. A cet effet, ce problème sera décliné suivant ces deux axes :

Le Dynamisme qui autorise les participants, ici les philosophes, à entrer et à sortir du système. Si on isole ce problème, on obtient une situation où les philosophes peuvent entrer et sortir mais peuvent manger n'importe où, les baguettes étant regroupées dans un sac. Nous appelons cette première situation "le pique-nique des philosophes", elle est représentable en réseau de Petri Place/Transition.

La Topologie qui assigne une place aux participants. Ici, les philosophes sont placés autour d'une table. Nous proposons une variante de cette situation où deux tables peuvent fusionner, et réciproquement une table se "casser en deux". Nous appelons cette deuxième situation "le bal des philosophes". Ici pour chaque situation de départ, qui fixe les effectifs impliqués, on peut associer un réseau de Petri Place/Transition. Un réseau de Petri Haut-Niveau décrit cette situation pour toute configuration de départ.

Le problème à résoudre est de considérer les deux variantes conjointement, ce que nous appellerons "le gala des philosophes". Le deuxième exemple, les apartés [Vil95] dans des groupes coopératifs, est introduit dans la section 2.4 et a pour vocation de montrer un cas où l'utilisation des grammaires de graphes propose de façon native une gestion intéressante du problème du partage coopératif de structures.

La première section détaille le cas d'étude et les premières analyses que nous pouvons réaliser pour mieux l'appréhender. La deuxième section est dédiée aux réseaux de Petri, et se termine par une présentation informelle des réseaux de Petri

de Haut-Niveau. Suit une section qui présente l'essentiel de la logique de Réécriture, c'est à dire la définition d'un état, d'une dynamique et la sémantique concurrente. La quatrième section présente les Grammaires de Graphes en prêtant une attention particulière à la notion de graphe. Enfin nous illustrerons comment les formalismes considérés traitent des concepts d'*identité* et de *localité*.

1.1 Le cas d'étude

Les exemples illustratifs permettent de soutenir l'intuition, de faire passer un message formel mais pas de montrer l'adéquation de nos techniques pour résoudre des cas concrets. Trouver un cas concret n'est pas d'une grande difficulté, le problème adressé est suffisamment partie prenante de la vie en collectivité pour que de tels cas soient légion. Mais nous cherchions un cas d'étude qui soit déjà identifié pour ne pas prêter le flanc à la critique souvent justifiée : "si vous avez pris un clou comme problème, c'est que vous saviez que votre outil était un marteau". Nous avons fini par trouver un cas concret identifié, en tout cas pour la communauté de la recherche : l'*organisation d'une conférence*. De plus, cet exemple est présent deux fois successivement dans les actes de la conférence COORDINATION'99 [Scu99, RV99], et développé selon deux points de vue : d'une part le graphe des tâches ou workflow [Scu99], d'autre part une spécification à base d'espaces de tuples [RV99]. De plus chacun des deux articles commence par un énoncé en langage naturel provenant d'une source commune non citée (un accord entre les auteurs?) qui forme une sorte de cahier des charges ou un guide informel à l'usage de l'apprenti "chairman". Pour nous positionner par rapport à ces deux articles, nous dirons que nous nous plaçons dans un cadre proche, mais théorique. Les techniques à base de *workflow* [Scu99] sont essentiellement des techniques de notation qui gagnent à être formalisées en Réseaux de Petri Place/Transition [Aal97], et peuvent donc être considérées comme adressées dans ce mémoire [AVG98]. La *réécriture de multi-ensembles de tuples* est aussi un sujet central, bien qu'essentiellement théorique et dont les *espaces de tuples partagés* [GT97, Sha97] peuvent former un support applicatif, avec cependant quelques restrictions [Zav98]. Avec une sincère gratitude, c'est à l'énoncé en langage naturel auquel nous accordons une grande valeur. Il est intéressant à plus d'un titre : au-delà de sa simple existence, c'est un témoignage d'une façon de voir ce type de système assez répandue et donc une bonne introduction car le langage utilisé est proche du langage courant.

Cette section commence par une traduction, aussi fidèle que possible de la version la plus détaillée [Scu99]. La sous-section suivante détaille le vocabulaire existant avant ce travail dans le domaine de la spécification coopérative (confluence et terminaison). Ces "éléments de langage" nous permettent d'enrichir notre connaissance du sujet, cette connaissance est résumée par une figure qui donne un qualificatif à chaque phase du processus. A cette occasion nous posons des définitions importantes, comme celle d'un calcul et d'une fonction.

1.1.1 Edition électronique des actes d'une conférence

Comme l'article original, [RV99], notre traduction reprend la séparation proposée entre d'une part la définition de l'architecture du système et d'autre part sa dynamique. Cette séparation se retrouvera dans la section suivante qui traitera d'abord les qualificatifs architecturaux puis ensuite les qualificatifs comportementaux.

L'édition électronique des actes s'appuie sur les services usuels du réseau (mail, www), auxquels les acteurs impliqués ont de plus en plus accès où qu'ils soient (e.g. chez eux, sur leur lieu de travail ...). Il s'agit donc de trouver une solution

permettant de coordonner ces différents outils pour permettre une coopération à la fois synchrone et asynchrone.

Agents et rôles [RV99] Le système que nous cherchons à spécifier contient plusieurs agents. Nous prenons le terme *agent* dans une acception primitive; intuitivement, un agent est une entité qui peut agir de façon autonome; un agent peut émettre/recevoir des messages en utilisant un protocole connu (pas forcément fiable): e.g. courrier physique, courrier électronique, site web, ou autre.

Chaque *agent* a une *identité* unique; par souci de simplicité, ces identités sont définies comme des URL (i.e. Uniform Resource Locator) uniques. Par exemple, un agent est le site de la conférence et son adresse est `url.of.conference`. Un autre agent peut communiquer avec le site de la conférence en utilisant différents protocoles:

- `http://url.of.conference`
- `mailto://url.of.conference`

Certains agents ont des rôles: ils représentent des utilisateurs humains qui peuvent réaliser certaines opérations. Les rôles suivants sont impliqués dans le cas d'étude:

- *Auteur* (i.e. qui soumet un article); si l'article a plusieurs auteurs, seul l'auteur qui a signé la soumission électronique est référencé comme l'"auteur", dans le sens où il est le correspondant attitré pour cet article.
- *Président de Comité de programme* (*PC Chair*);
- *Membre du Comité de Programme* (*PC Member*);
- *Relecteur*;
- *Editeur* des actes.

Chaque *rôle* peut être détenu par plusieurs agents, qui ont des URL distinctes. Par exemple, une conférence typique peut avoir une centaine d'auteurs, deux *PC chairs*, une vingtaine de *PC members*, un nombre suffisant de relecteurs et un éditeur des actes.

Les opérations qu'un agent peut réaliser, selon son rôle, sont décrites dans la sous-section suivante.

Dynamique du cas d'étude [RV99] Nous allons spécifier la dynamique du processus devant être réalisé en définissant un ensemble de tâches supportées par le système de gestion. Ces tâches sont listées ci-dessous dans un ordre arbitraire.

1. Soumission des articles Les auteurs soumettent des articles au site de la conférence. Un formulaire de soumission est disponible à partir du site de la conférence (i.e. les auteurs peuvent demander un tel formulaire à l'agent du site). Il inclut les informations de base sur les soumissions telles que les auteurs, leurs affiliations, le titre de l'article, l'auteur correspondant et le format de l'article soumis.

Une soumission consiste en un formulaire rempli correctement, un fichier ASCII contenant le titre de l'article ainsi qu'un résumé et l'article (ou son URL) dans le format déclaré dans le formulaire (e.g. PostScript).

Si une soumission ne remplit pas les critères ci-dessus, elle est rejetée et son émetteur est invité à soumettre de nouveau. Une soumission conforme est stockée dans le site de la conférence et l'auteur correspondant reçoit un accusé de réception.

2. **Répartition des articles** Chaque *PC member* examine la liste des soumissions et sélectionne un sous-ensemble d'articles "intéressants" pour relecture. Le *PC chair* peut modifier cette sélection pour équilibrer la charge des membres du comité de lecture.
3. **Distribution des articles aux relecteurs** Lorsque la date butoir pour les soumissions a expiré, un agent envoie un e-mail à chaque membre du comité de lecture fixant les articles qui sont assignés à chaque membre et entamant le processus de relecture. Ce message peut même être "actif", dans le sens où il contacte de façon autonome le site à l'URL indiqué, regroupe les papiers et génère à la demande les formulaires de relecture devant être renvoyés ultérieurement au site de la conférence.
On doit se prémunir contre les accès illicites à l'ensemble des articles, la liste des soumissions reste ainsi confidentielle.
4. **Collecte des rapports** Un formulaire WWW est fourni pour la saisie en ligne des rapports des relecteurs, stockés dans le site de la conférence. Ces formulaires peuvent être aussi obtenus en envoyant un e-mail au site de la conférence, et les formulaires remplis hors ligne peuvent être soumis directement par e-mail.
A partir de cette phase, les problèmes de sécurité et d'authentification sont critiques. La confidentialité des informations doit être assurée par des mots de passe qui autorisent l'accès aux seuls membres et en utilisant une adresse unique pour chaque membre vers qui toute la communication est dirigée. Cette phase est gérée par un agent qui répond aux demandes de formulaires et collecte les rapports. L'agent crée un fichier pour chaque article dans lequel il stocke les relectures associées.
5. **Préparation des statistiques** Un agent parcourt tous les rapports pour générer des statistiques et une proposition initiale de classement de tous les articles servant de point de départ à la réunion du comité de programme.
6. **Réunion du comité de programme** Durant cette phase chaque membre du comité de programme doit pouvoir:
 - avoir accès à la liste, aux résumés et aux fichiers des soumissions;
 - avoir accès aux rapports de relecture disponibles à ce moment pour un article spécifique;
 - avoir accès au tableau de classement de toutes les soumissions;
 - ajouter un commentaire à la fin d'un fichier **minutes**;
 - lire le fichier **minutes**;
 - décider du statut final (acceptation ou rejet) et enregistrer sa décision.

Tout ceci est fourni via des liens (protégés par des mots de passe) allant d'une page WWW vers des éléments (actifs) appropriés. En plus, toutes les capacités (features) sont aussi disponibles par e-mail en envoyant des messages ayant un sujet correct à l'agent responsable du serveur de mail. *La situation devient plus complexe si des membres du comité de programme ont la possibilité de soumettre des articles.* Dans ce cas il faut leur interdire l'accès direct ou par inférence, aux informations sur leurs propres soumissions.
7. **Communication des résultats** Une fois que la liste des papiers acceptés est fixée, un agent prend en charge de communiquer les résultats et les rapports de relecture aux auteurs. Un autre agent génère une liste des résumés des articles acceptés (par exemple, en HTML), extrait une liste de relecteurs pour les rapports de relecture et prépare un synopsis du résultat pour le *PC chair*.

- 8. Soumissions des articles camera-ready** Les auteurs des articles acceptés soumettent un article dans sa version finale (*camera-ready*). Un agent contrôle la compatibilité des soumissions avec les standards de soumission *camera-ready* et émet des rappels pour les auteurs en retard dans leur soumission.
- 9. Préparation des actes** Un agent collecte les versions camera-ready, demande et attend une préface de l'éditeur des actes et prépare un brouillon (ang. *draft*) des actes. L'éditeur et les *PC chairs* sont avertis par e-mail lorsque les actes sont prêts.

Ici se termine la traduction de notre cas d'étude. Nous allons maintenant procéder à son analyse, d'une part en utilisant des concepts classiques dans le contexte, puis en utilisant des éléments de langage provenant de la réécriture.

1.1.2 Analyse de la spécification

Le document ci-dessus est une spécification en langage naturel d'un *processus* coopératif, le terme de *processus* signifie ici ensemble d'opérations menant à la réalisation d'un "produit". Ce document a les forces et les faiblesses de l'implicite et du non-dit. Comme nous l'avons dit, sa force réside dans son existence même. Elle est largement suffisante pour permettre une *coopération informelle* [Vil95] entre personnes. Par contre, l'objectif d'une spécification opérationnelle serait de permettre la spécification ultérieure de composants logiciels permettant une *coopération formelle* [Vil95] entre ces agents, c'est à dire où le système lui-même dispose d'une représentation (centralisée ou répartie) de l'état du *processus* et interprète des éléments du contenu des messages pour faire évoluer cette représentation. La spécification contient, comme toute spécification en langage naturel, un grand nombre de prémisses cachées, c'est à dire des axiomes non-dits qui structurent *de facto* le discours. Notre lecture de cette spécification va nécessairement remplir ces vides par nos propres conceptions.

Classifier les architectures Le point de départ de notre conception de la coopération, ou de groupes coopératifs, est la thèse de T.Villemur [Vil95], dont l'état de l'art reste une référence pour l'existant en terme de logiciels et d'approches opérationnelles pour la coopération. Outre la séparation coopération formelle/informelle déjà introduite, il définit deux autres couples d'adjectifs pour ce type d'application.

Synchrone/Asynchrone On parle de coopération synchrone lorsque les différents coopérants doivent être présents simultanément à leur poste de travail pour réaliser la coopération. La réunion est le prototype de la coopération synchrone, et la production/lecture de documents écrits est celui de la coopération asynchrone. La production/lecture de documents peut se raffiner en deux grandes modalités: l'*envoi* et la *publication*:

- avant un *envoi*, le producteur spécifie quelles personnes recevront le document envoyé, par diffusion ou par circulation;
- après une *publication*, le lecteur spécifie la référence d'un document publié pour le recevoir.

L'*envoi* est une modalité plus forte, dans le sens où elle permet de représenter la *publication* par l'ajout d'un contenu de nature algorithmique: un *protocole*. Pour illustrer cette dernière assertion, remarquons que le réseau informatique actuellement déployé s'appuie sur des mécanismes d'envoi/réception de paquets et qu'il supporte deux grands services représentant les deux modalités sus décrites: l'e-mail (*envoi*) et le web (*publication*).

Local/Distant On parle de coopération locale lorsque la coopération synchrone peut avoir lieu sans envoi de messages. Dans le cas de la coopération distante, la coopération synchrone est réalisée par un protocole sur un système à base d'envoi, à l'instar de la publication.

A l'aune de cette classification, notre cas d'étude est essentiellement distant. Il détaille d'ailleurs la phase de coopération synchrone (6. *Réunion du comité de lecture*) par un protocole (i.e. une contribution libre à des **minutes** consultables par tous) reposant sur un système d'échanges de messages. En outre, il identifie volontairement l'envoi et la publication par un procédé que nous détaillerons plus avant. Il se place donc résolument dans un cadre Asynchrone/Distant. Les exemples illustratifs seront eux dans un cadre Synchrone/Local.

Maintenant détaillons comment le système est décrit. Notre point de vue, depuis les travaux de K.Drira sur la coopération [DGV96], caractérise celle-ci comme reposant sur des services de coordination; eux-mêmes reposant sur une communication à base d'envoi. Notre cas d'étude raffine cette perception en décrivant le système en termes de services de coordination rendus par des *agents coordinateurs*, représentant une personne morale (e.g. la conférence), à des *agents mandataires* [Mos94] (*ang.* proxies) représentant chacun une personne physique mandante (e.g. un coopérant). Il stipule en outre que les *agents mandataires* ont un rôle. Un rôle est explicitement défini comme *la possibilité de réaliser certaines actions*. Sachant que la seule action possible est par ailleurs d'envoyer un message, un rôle est défini sans équivoque par l'ensemble des messages qu'un agent doté de ce rôle peut envoyer. Nous verrons que nous doterons les *agents coordinateurs* d'un comportement qui spécifie en plus la causalité entre les différentes émissions/réceptions réalisées. La difficulté de spécifier un tel système provient de la difficulté de décrire le comportement de l'ensemble du service de coordination à partir du comportement des *agents coordinateurs*.

Classifier les dynamiques discrètes

La modalité de classification précédemment décrite ne fait pas intervenir la façon dont le système évolue, ce que nous appellerons sa dynamique. Cependant nous pensons que cet aspect est aussi important et peut permettre de raffiner notre perception du cas d'étude. A cet effet, nous allons utiliser des éléments de langage empruntés à la théorie de la réécriture [BN98] et plus particulièrement les notions de *confluence* et *terminaison*. Ces deux notions vont nous permettre de définir quatre types de systèmes, et cette classification sera utilisée avec profit sur les phases de notre cas d'étude.

Nous introduisons ici les notions d'*ensemble*, de *relation* et, dans un deuxième temps, de *fonction*.

Définition 1.1 (Ensemble) *Un ensemble est une collection d'éléments. Ces éléments sont notés par des lettres minuscules ($a, b, c, d, s, t, u, v, x, y, z, \dots$). Les ensembles seront notés avec des lettres majuscules ($A, B, C, D, E, X, Y, \dots$).*

Définition 1.2 (Relation) *Une relation D de A dans B ($D \subset A \times B$), est une partie de l'ensemble des paires formées d'un élément de A et d'un élément de B . Une relation D sur A est une relation de A dans A .*

Notation misfixe: Une convention souvent adoptée est de noter le signe de la relation de façon *infixe*, $(a, b) \in ({}_D) \equiv (aDb)$. Le fait que D soit compris ici comme un symbole infixé est dénoté par la présence des “ $_$ ” qui permettent de fixer la position des paramètres. Cette notation est appelée *misfixe*, introduite dans OBJ[RB88]. Ceci devient particulièrement lisible si on choisit un symbole fléché

pour la relation, par exemple avec $(_ \rightarrow _) \subset A \times A$, l'équation précédente devient $(a, b) \in (_ \rightarrow _) \equiv (a \rightarrow b)$. Le caractère visuel de cette notation permet de représenter une relation D graphiquement comme des flèches sans étiquette reliant les éléments de l'ensemble A . Plus avant nous différencierons très précisément la notion de graphe de la notion de relation.

On définit un ensemble abstrait A_s des états d'un système s (élément de l'ensemble des systèmes), ce système n'est pas forcément la coopération prise globalement, mais peut être un agent, voire un composant d'un agent. Une dynamique sur un système s est une relation $D \subset A_s \times A_s$. Nous interprétons cette relation comme une dynamique en disant que la présence de la paire (a, b) dans D signifie que si le système est dans l'état a alors il pourra être dans l'état b en un pas. Ici, intervient l'hypothèse très forte que le système que nous considérons est doté de transformations discrètes, il y a un *avant* et un *après* la singularité que constitue la transformation. Pour la réécriture [BN98], cette construction s'appelle *système de réduction abstrait*; dans le domaine des systèmes concurrents, cela s'appelle un *système de transition* [Arn87]. Une fois muni de ces définitions primitives, on va chercher à caractériser les dynamiques discrètes. En fait on va définir trois caractéristiques : le *branchement fini*, la *terminaison* et la *confluence*.

- Le branchement fini : stipule que chaque état n'a qu'un nombre fini de successeurs.
- La terminaison : stipule que le système va inévitablement arrêter de se transformer après un certain nombre, inconnu au départ, de transformations.
- La confluence : stipule que cette dynamique est indépendante des choix effectués lors des branchements : "ils peuvent toujours être rattrapés dans l'avenir du système".

Il est à noter qu'un système peut avoir toutes, certaines ou aucune caractéristiques parmi ces trois. Ceci nous donne 8 types de systèmes possibles. Le premier critère ne sera pas exploité dans la classification finale, d'où la définition ultérieure de quatre types de systèmes.

Définition 1.3 (Branchement fini) Une dynamique D sur A est dite à branchement fini si seulement si pour tout $a \in A$, l'ensemble des b tel que $(a, b) \in D$ est fini.

Définition 1.4 (Terminaison) Une dynamique est dite terminante si et seulement si il n'existe pas de fonction u de l'ensemble des entiers \mathbb{N} dans A vérifiant $(u(n), u(n+1)) \in D$ i.e. strictement croissante.

Un système muni d'une dynamique terminante finit par arriver dans un état sans successeur.

Les deux définitions précédentes sont reliées par une propriété essentielle donnée ci-dessous :

Propriété 1.1 (Fonction de progrès) Une dynamique à branchement fini est terminante ssi il existe une fonction $\text{progres}(_)$ de A dans \mathbb{N} vérifiant $\forall (a, b) \in D, \text{progres}(a) > \text{progres}(b)$.

Pour la démonstration, nous indiquons [BN98] (elle fait intervenir une généralisation du lemme de König) car nous nous intéressons à ce point précis pour souligner la forme du contre-exemple donné dans le même ouvrage.

Considérons la dynamique $(_ \rightarrow _)$ définie sur $\mathbb{N} \times \mathbb{N}$ par les deux règles $(i+1, j) \rightarrow (i, k)$ et $(i, j+1) \rightarrow (i, j)$ pour tout $i, j, k \in \mathbb{N}$. Cette dynamique n'est pas à branchement fini, car le k de la première règle n'est pas "contraint par le membre

gauche de la paire", et donc chaque état à un nombre dénombrable de successeurs indicés par k , plus un pour l'application de l'autre règle. Néanmoins, sa terminaison peut-être obtenue par une construction en considérant un ordre lexicographique. La non-existence de la fonction de **progres**($_$) se prouve par contradiction en considérant $l \stackrel{\text{def}}{=} \text{progres}(1, 1)$ et la suite d'inégalités strictes :

$$l > \underbrace{\text{progres}(0, l) > \text{progres}(0, l-1) > \dots > \text{progres}(0, 0)}_{\text{implique } \text{progres}(0, l) \geq l}$$

Ceci nous permet de statuer de la non-existence de la fonction de progrès, mais ceci prouve par l'exemple qu'il est possible de *raisonner* sur de telles dynamiques. En outre le branchement infini est associé au fait d'avoir une *variable libre à droite* et nous verrons que ce type de construction est essentiel à notre propos.

Pour la confluence, on utilise D^* qui est la clôture transitive et réflexive de D que l'on peut construire comme ceci : on définit par induction la suite de relations D^n : on pose $D^0 = \{(a, a) \in A \times A \mid a \in A\}$ et l'induction $D^{n+1} = \{(a, b) \in A \times A \mid (a, c) \in D^n \text{ et } (c, b) \in D\}$. On définit $D^* = \cup_{i \in \mathbb{N}} D^i$, on vérifie $D^1 = D$.

(a, b) dans D^* signifie que si le système est dans l'état a alors il pourra être dans l'état b après un nombre quelconque de pas. La confluence d'une dynamique spécifie que les branchements de cette dynamique sont sans conséquence à long terme et que l'on peut toujours compléter un branchement divergent par une convergence ultérieure.

Définition 1.5 (Confluence) *Une dynamique est dite confluente ssi pour tout a, b, c éléments de A , si $(a, b) \in D^*$ et $(a, c) \in D^*$ alors il existe d tel que $(b, d) \in D^*$ et $(c, d) \in D^*$. On qualifie deux états b et c de divergents ssi il n'existe pas d tel que $(b, d) \in D^*$ et $(c, d) \in D^*$.*

Nous allons utiliser ces notions pour deux objectifs complémentaires: d'abord, l'approche classique qui s'intéresse aux dynamiques terminantes et confluentes va nous permettre de clarifier des notions relatives aux équivalences entre des énoncés syntaxiquement différents. Ensuite, nous utiliserons les différents qualificatifs pour qualifier la dynamique de notre cas d'étude et chacune de ses phases.

Calculs, équivalence syntaxique et fonctions L'insertion de ce paragraphe théorique correspond à un besoin de clarification de notions importantes relatives au lien qu'entretiennent les ordinateurs avec leur fonction initiale: le *calcul*. Cette fonction n'est plus celle qui nous préoccupe essentiellement, car ces machines sont devenues, grâce au réseau, des *médias*. Ce changement de fonction induit un changement d'interprétation des concepts fondamentaux. Néanmoins, ils conservent leur pertinence. Par exemple, un état sans successeur peut être interprété de façon radicalement différente selon le type de système considéré: pour un système terminant et confluent c'est un *résultat* alors que pour un protocole c'est un *blocage*.

L'informatique s'est très longtemps intéressée aux calculs (i.e. dynamiques terminantes et confluentes) car elles définissent des "fonctions". Ici ce terme, à la différence de ses occurrences antérieures, ce mot n'est pas entendu dans sa notion architecturale ("la fonction des ordinateurs était le calcul") mais comme un objet mathématique associant deux ensembles (cible et source) en associant à chaque élément de la source un élément de la cible. En effet, comme la dynamique termine et que le déroulement est indépendant des branchements, l'état d'arrêt est uniquement fonction de l'état de départ. On appellera ce type de dynamique des *calculs* (ang. *computation*).

Définition 1.6 (Fonction) *Une fonction totale (resp. partielle) $(f(_)) : A \rightarrow B$ est une relation $f(_) \subset A \times B$ telle que pour tout $a \in A$, il existe un et un seul (resp. un ou aucun) b tel que $(a, b) \in f(_)$, on note $f(a) = b$ cette propriété.*

Par la suite, sauf mention explicite, les fonctions considérées seront totales.

Opérateurs et notations Il est essentiel de remarquer que les fonctions sont des ensembles. En conséquence, la fonction vide existe et a un sens, et l'union est une opération partielle : l'union de deux fonction est définie ssi les fonctions sont égales sur l'intersection des sources.

Pour une fonction totale, être vide implique que l'ensemble source est vide. On notera B^A l'ensemble des fonctions de A dans B. On dénote par la formule ci-dessous,

$$(f : \{x, y, z\} \rightarrow \mathbb{N}) = (\text{fn } x \Rightarrow 2 \mid \text{any} \Rightarrow 1)$$

la relation $\{(x, 2), (y, 1), (z, 1)\} \subset \{x, y, z\} \times \mathbb{N}$ qui est aussi une fonction appartenant à $\mathbb{N}^{\{x, y, z\}}$. On utilise ici la notation *any* pour indiquer que la fonction a la même valeur pour tous les éléments de l'ensemble source dont l'image n'a pas été encore spécifiée, cette notation est inspirée de ML [HMM86] bien que le signe $_$, déjà utilisé, ait été remplacé par le mot clé *any*.

Définition 1.7 (Fonction partielle de résultat) On note $\mathcal{D}^\dagger(_)$ la fonction partielle associée à une dynamique D confluente: $\mathcal{D}^\dagger(a)$ est l'unique état d'arrêt atteint par D en partant de a . S'il existe, un tel état est appelé "résultat" de a par D . Si D est terminante, ce qui est en général indécidable, $\mathcal{D}^\dagger(_)$ est totale. La confluence est en général décidable [KB70].

Ce point n'est pas tout à fait déconnecté de notre cas d'étude; en effet, un point important de son propos est l'unification qu'il suppose des moyens de communication asynchrone. Chaque agent a une identité assimilée à une adresse unique à laquelle on peut le joindre. Cette adresse est donnée indistinctement pour le web et pour le mail. Il est précisé plus loin dans la spécification [Scu99] que les requêtes émises vers des pages *www* puissent se faire aussi par e-mail. Comment cette unification peut-elle être réalisée?

Concentrons-nous d'abord sur le problème du traitement des réponses. Soit $\text{reponse}_{\text{mail}}$ et $\text{reponse}_{\text{http}}$ deux réponses "identiques" faites par les deux différents médias; traiter ces deux réponses de façon uniforme implique que l'on dispose d'un calcul C_{reponse} qui traite les deux réponses et les mette sous une forme "normalisée".

Définition 1.8 (Equivalence syntaxique) Soit D une dynamique confluente et terminante (i.e. un calcul) sur A . Deux éléments a et b de A sont D -équivalents (syntaxiquement) ssi $\mathcal{D}^\dagger(a) = \mathcal{D}^\dagger(b)$.

Considérons à présent le problème symétrique, l'émission de formulaires sur les deux médias. On souhaite qu'ils soient "identiques" pour être sûr d'assurer la cohérence des versions entre les voies d'accès. Donc nous définissons un calcul C_{form} nous permettant de statuer cette équivalence. Mais à ce stade, on ne sait pas produire les différents équivalents. Il faut donc se doter de deux calculs qui vont élaborer les différents formulaires, tout en ne perdant pas d'information. C'est-à-dire qu'ils restent dans la classe d'équivalence syntaxique définie par C_{form} . De tels calculs sont appelés *univoques*.

Définition 1.9 (Représentation univoque) Soit C et P deux calculs sur A .

P est une représentation univoque de C ssi

$$\forall a, b \in A, C^\dagger(a) = b \implies C^\dagger(P^\dagger(a)) = b$$

Ainsi la spécification d'un système indépendant du média suppose quatre dynamiques: C_{reponse} pour récupérer les formulaires remplis, deux mises en forme P_{mail} et P_{http} du formulaire vierge, qui sont des représentations univoques du

formulaire a selon C_{form} qui permet de statuer de la validité des deux précédentes. Plus généralement, lorsque nous dirons que deux assertions ou formules sont équivalentes quant à la forme c'est autant que possible en s'étant assuré de l'existence des calculs en question par l'utilisation d'un système de production de documents construits avec ces notions [Knu84, Lam86].

Retour à la classification

Revenons au cas d'étude. On va classer les dynamiques par rapport aux caractéristiques de terminaison et de confluence, la notion de branchement fini n'intervenant pas dans cette classification.

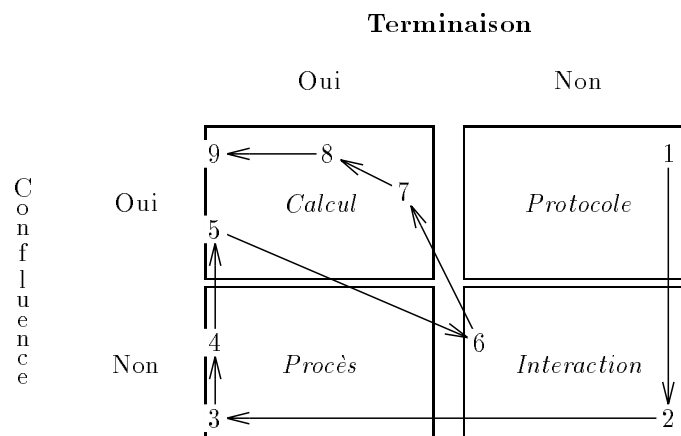
Dans son ensemble, on souhaite qu'il termine; par contre sa raison d'être est justement de ne pas être confluent et de décider quels articles doivent être dans les actes et lesquels seront rejetés. On appellera ce type de dynamique des *processus*, qu'il faut différencier d'autre acceptions de ce même mot [Mil89, CP99]. Comme nous l'avons déjà énoncé, nous appellerons *calcul* une dynamique terminante et confluyente.

Maintenant nous voulons compléter les cases manquantes. Nous pensons qu'une dynamique non terminante mais confluyente recouvre bien l'idée de *protocole*. Pas conçu pour s'arrêter ou calculer un résultat, il doit assurer un service indépendamment des aléas du système sur lequel il s'appuie. Quant à une dynamique non confluyente et non terminante nous l'appellerons une *interaction* en référence à [Weg97].

Muni de notre classification toute fraîche, essayons de l'appliquer aux différentes phases du cas d'étude, la figure 1.1 résume ces différentes caractéristiques des phases :

1. **Soumission des articles** PROTOCOLE. Cette phase n'est pas terminante en elle-même, car elle ne doit sa terminaison qu'à l'utilisation d'un sablier (i.e. un "timer") qui définit une *date butoir* pour les phases 1 & 2, or cette notion, le temps continu, est extérieure à notre approche. Elle doit assurer que tous les auteurs qui veulent soumettre y arrivent, indépendamment de leur ordre d'arrivée. Un point essentiel du point de vue du spécifieur est que l'ensemble des auteurs est inconnu au départ. En outre, cette phase n'est pas finiment branchée : à chaque nouvelle soumission le nombre d'auteurs pouvant soumettre n'est pas borné *a priori* au lancement du système. De plus, on n'en connaît pas *a priori* l'identité. Même en supposant que l'on puisse déterminer au lancement du système un ensemble d'identité de "soumetteur" potentiel, l'adresse d'un nouvel auteur potentiel peut tout à fait être allouée sur le réseau après le lancement du système.
2. **Répartition des articles** INTERACTION. Cette phase n'est pas terminante en elle-même pour les mêmes raisons que celles détaillées pour la phase 1. Elle n'est pas confluyente, les choix des différents membres du comité de programme forment le service rendu par cette phase. Il apparaît ici que la prise de décision est associée ici à un comportement non confluyente du système.
3. **Distribution des articles aux relecteurs** PROCESSUS. Cette phase est terminante et non confluyente car le choix des relecteurs par les membres du comité de programme amène le système dans des états divergents. Encore une fois cette phase n'est pas finiment branchée. Et il apparaît ici que les phases non finiment branchées correspondent à des phases où des agents entrent dans le système, ici les relecteurs.
4. **Collecte des rapports** PROCESSUS. Cette phase est terminante et non

FIG. 1.1 – Classification et application aux phases du cas d'étude



confluente car la décision des relecteurs amène le système dans des états divergents.

5. **Préparation des statistiques** CALCUL. Cette phase terminante et confluente élabore une représentation réduite du système global pour les besoins de la phase suivante. Cette phase est très intéressante car elle permet de dire que le système global a besoin pour être appréhendé de disposer, de façon intégrée, de “réductions” (ou “*vues partielles*”) qui permettent d’en sélectionner des aspects intéressants à un instant donné. On peut d’ailleurs remarquer que cette “vue partielle” a un sens durant les deux phases précédentes pour rendre compte de leur “avancement” qui peut être rapprochée de la valeur d’une fonction de progrès.
6. **Réunion du comité de programme** INTERACTION. Comme pour la phase 2, la terminaison n’est pas un fait de la spécification mais repose sur un *consensus* des participants, dont l’élaboration ne relève pas de cette spécification. On peut par contre la supposer finie. Quant à l’absence de confluence, la *décision*, ici d’accepter ou de rejeter un article, est la définition même d’une transformation irréversible. Même si un recours était possible cela ne mènerait sûrement pas à la confluence, négation même de l’idée de décision.
7. **Communication des résultats** CALCUL. Cette phase, symétrique de la phase 5, élabore différentes “*vues partielles*” qui permettent de réaliser des documents, soit pour les auteurs, soit devant être intégrés aux actes.
8. **Soumissions des articles camera-ready** CALCUL. A la fin de cette phase, tous les auteurs reçoivent les conclusions de la relecture et du comité de lecture.
9. **Préparation des actes** CALCUL. Cette phase est confluente et terminante : à son terme, tous les auteurs acceptés ont remis leur version finale.

Notre approche permet d’enrichir le vocabulaire de description des systèmes coopératifs. Ainsi à côté de définitions qualifiant l’architecture de la coopération déjà reconnues [EGR91] (Local/Distant, Synchrone/Asynchrone), nous avons ajouté deux termes provenant du vocabulaire de la réécriture que nous pouvons grouper par paire pour donner quatre types de dynamiques (Cf. Figure 1.1). Avec ces éléments de langage : notre *cas d’étude* est un *Procès / Asynchrone / Distant* (en faisant

abstraction des phases non terminantes par absence de représentation du temps) et nos *exemples* seront des *Protocoles* / *Synchrones* / *Locaux*.

Nous allons maintenant entamer les développements qui impliquent des formalismes connus de spécification de systèmes. Nous allons considérer trois formalismes: les Réseaux de Petri, La Logique de Réécriture et les Grammaires de Graphes par double-somme (amalgamée). Le point essentiel réside dans les deux phases à branchement infini. Elles correspondent toutes deux à des transformations où un élément inconnu dans le système avant la transformation pénètre dans le système. Ici se situe un problème nodal adressé par notre mémoire. Comment spécifier des systèmes où des éléments sont éphémères, c'est à dire qui peuvent apparaître et disparaître. Nous appelons ce problème le *dynamisme des références*. Les trois formalismes étudiés intègrent cette question et la résolvent par des moyens induisant des interprétations très différentes.

1.2 Réseaux de Petri

Les réseaux de Petri sont un domaine extrêmement actif de la recherche dans le domaine des systèmes concurrents et, chaque année, une conférence internationale lui est dédiée dont les actes contiennent l'ensemble des avancées dans ce domaine. Parmi les applications les plus courantes des réseaux de Petri, on peut citer: les protocoles de communication, la commande d'ateliers de fabrications, l'organisation des flots de travail dans l'entreprise et les interfaces homme-machine [Val86].

Nos premières tentatives pour réaliser une spécification de ce cas d'étude vont s'appuyer sur les Réseaux de Petri Place/Transition. Les réseaux de Petri sont un outil disposant d'une représentation graphique et s'appliquent à un très grand nombre de domaines. Ce formalisme dispose d'une représentation intrinsèque des actions concurrentes, ce qui en fait depuis sa création un outil irremplaçable dans l'étude des systèmes concurrents. En outre, le corps théorique des réseaux de Petri est très fourni et le grand nombre de propriétés démontrées en fait un outil d'analyse puissant.

Définition 1.10 (Réseaux de Petri (Place/Transition)) *Un réseau de Petri Place/Transition est un quadruplet [Reu89] $R \triangleq \langle P; T; \bullet R; R^\bullet \rangle$ où $P \cap T = \emptyset$ et*

- P est un ensemble fini de places;
- T est un ensemble fini de transitions;
- $\bullet R$ est une fonction de $T \times P$ dans \mathbb{N} appelée fonction d'incidence avant;
- R^\bullet est une fonction de $T \times P$ dans \mathbb{N} appelée fonction d'incidence arrière.

Etat *Un marquage est une fonction $(M : P \rightarrow \mathbb{N})$. On note en conséquence \mathbb{N}^P l'ensemble des marquages.*

Transition sensibilisée *Une transition $t \in T$ est sensibilisée par M [dans R] ($M[t >]$) ssi $\forall p \in P, M(p) \geq \bullet R(t, p)$.*

Tir d'une transition *Si $t \in T$ est sensibilisée dans M , le tir de t sur M ($M[t > M']$) donne le nouveau marquage M' tel que:*

$$\forall p \in P, M'(p) = (M(p) - \bullet R(t, p)) + R^\bullet(t, p)$$

On appelle réseau marqué une paire $\langle R; M \rangle$ formée d'un réseau et d'un marquage.

Il existe une autre version des réseaux de Petri élémentaires, les réseaux de Petri Condition/Évènement. Ils sont obtenus en ne considérant comme marquages valides que les éléments de $[0, 1]^P$. De façon similaire, cette contrainte est étendue aux fonctions d'incidence des transitions. D'une certaine façon on les borne structurellement. Leur sémantique est écrite en considérant $[0, 1]^P$ comme étant l'ensemble des parties de P , noté $\mathcal{P}(P)$. On donne la définition suivante:

Définition 1.11 (Réseaux de Petri (Condition/Évènement)) *Un réseau de Petri Condition/Évènement est un quadruplet $R \stackrel{\text{def}}{=} \langle P; T; \bullet R; R^\bullet \rangle$ où $P \cap T = \emptyset$ et*

- *P est un ensemble fini, dont les éléments sont appelés les places;*
- *T est un ensemble fini, dont les éléments sont appelés les transitions;*
- *$\bullet R$ est une fonction de T dans $\mathcal{P}(P)$;*
- *R^\bullet est une fonction de T dans $\mathcal{P}(P)$.*

Etat *Un marquage est une partie de P , ($M \in \mathcal{P}(P)$).*

Transition sensibilisée *Une transition $t \in T$ est sensibilisée par M ssi $M \supseteq \bullet R(t)$ et $(R^\bullet(t) \setminus \bullet R(t)) \cap M = \emptyset$.*

Tir d'une transition *Si $t \in T$ est sensibilisée dans M , le tir t sur M donne le nouveau marquage M' tel que: $\forall p \in P, M'(p) = (M(p) \setminus \bullet R(t, p)) \cup R^\bullet(t, p)$.*

Cette forme est très utilisée pour les problèmes de sémantique du vrai parallélisme [WN95, Maz87]. Les deux formes sont très différentes et n'adressent pas les mêmes problèmes. Nous travaillerons exclusivement sur des réseaux de Petri Place/Transition.

1.2.1 Formes textuelles équivalentes

La prolifération des formes de réseaux de Petri est une caractéristique de ce formalisme simple d'accès mais doté de limitations structurelles qui poussent ses utilisateurs à "customiser" leur propre version pour leurs besoins ponctuels. C'est la rançon du succès, mais cette jungle est bien épaisse pour celui qui essaye de formaliser des propriétés générales.

Même en restant dans le domaine bien délimité des réseaux de Petri Place/Transition, il existe au moins deux autres formes textuelles différentes mais équivalentes des Réseaux de Petri Place/Transition. Ces différences résident dans l'expression des fonctions d'incidence. La forme la plus répandue est aussi la plus difficile à manipuler; elle définit une seule fonction d'incidence, appelée *relation de flot*:

$$(R : P \times T \cup T \times P \rightarrow \mathbb{N})$$

Si P et T sont distincts, alors $T \times P$ et $P \times T$ sont distincts. En considérant les fonctions comme des relations, et en utilisant l'union sur les ensembles comme opérateur partiel sur les fonctions (défini ssi les domaines sont disjoints) on obtient :

$$R = (\bullet R \circ \text{swap}) \cup R^\bullet \text{ où } (\text{swap} : P \times T \rightarrow T \times P) \stackrel{\text{def}}{=} (\text{fn } (p, t) \Rightarrow (t, p))$$

La construction formelle nous permettant d'utiliser l'union comme opérateur partiel sur les fonctions est détaillée dans la partie 2.1. Il est plus technique de retrouver à partir de cette écriture la forme donnée comme référence.

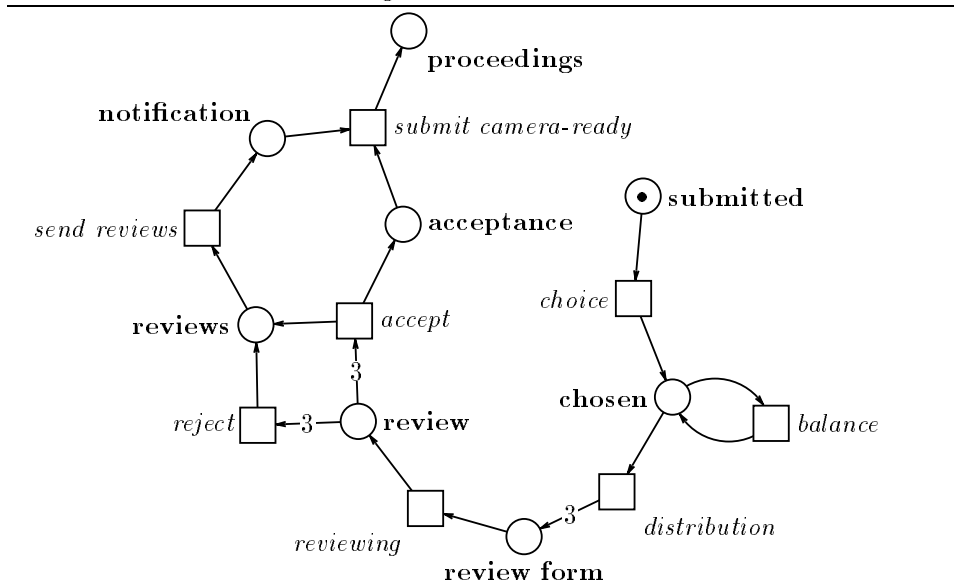
L'autre forme considère aussi une seule fonction d'incidence.

$$(R : T \rightarrow \mathbb{N}^P \times \mathbb{N}^P)$$

L'équivalence s'obtient en deux pas. On currifie les deux fonctions d'incidence, et on obtient une paire de fonction de signature $T \rightarrow P \rightarrow \mathbb{N}$; à partir de ces deux paires de fonctions on réalise de façon univoque une fonction qui donne une paire de marquages. Encore une fois nous faisons appel à une notion non triviale (la *currification*) pour réaliser la passerelle entre les deux écritures. Cette forme est la plus générale et permet de déduire les deux autres sans faire intervenir de notions techniques. Pour les manipulations formelles du chapitre III, la forme currifiée sera la plus adaptée.

1.2.2 Représentation graphique et cas d'étude

FIG. 1.2 – Réseau avec marquage initial d'un article



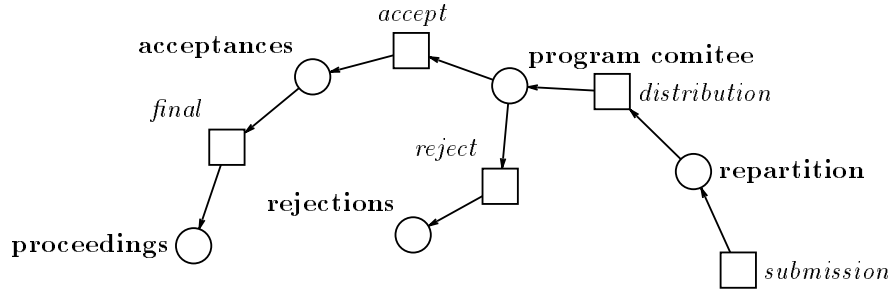
Une caractéristique fondamentale des réseaux de Petri, une des raisons de leur succès et sûrement une des causes de cette multiplicité de notations textuelles équivalentes, est l'existence d'une notation graphique universellement acceptée. On représente les places par des ronds, les transitions par des carrés. La fonction de flot est équivalente à un graphe bipartite complet dont chaque arête est étiquetée par un entier. De façon univoque, on ne représente pas les arêtes de poids nul, et on omet le poids des arêtes de poids 1. On arrive donc à l'équivalence suivante:

Un réseau de Petri est équivalent à un graphe dirigé bipartite dont certaines arêtes sont étiquetées par des entiers supérieurs à 1.

Le paradigme original de la notation graphique (en apparence simple mais en fait redoutable) réside dans la représentation des marquages. Première simplification, on représente le marquage sur le réseau *lui-même* en étiquetant les places par des entiers. Ceci contient déjà beaucoup de sens implicite. En outre, on représente ces entiers par une accumulation de symboles identiques (\bullet), parcourant à rebours l'abstraction des entiers, ce qui est tout sauf anodin. La représentation finale est donc une sorte d'*Abaque* où l'on aurait indiqué sur le bois les règles du “jeu des jetons”.

Il n'empêche ! Cette représentation est fort pratique et visuellement d'un grand impact car notre reconnaissance intuitive de ce graphe élabore des schémas (cycles, degré...) qui correspondent à des notions théoriques intéressantes (invariants, concurrence...). Nous allons donner deux réseaux de Petri Place/Transition, le

FIG. 1.3 – Spécification par réseau du système



premier, représenté figure 1.2, détaille le cheminement d'un article dans le procès de la conférence. Le second, donné dans la figure 1.3, envisage le système composé de tous les articles : il est non borné, et ceci parce que la transition *submission* n'a pas de pré-condition autre qu'une contrainte de phase.

Or cette contrainte de phase ne peut pas être décrite par un réseau Place/Transition. En effet, les réseaux de Petri Place/Transition sont *monotones*, c'est à dire si $(\mathbf{t} : M \rightarrow M')$ alors pour tout marquage C ($\mathbf{t} : M \oplus C \rightarrow M' \oplus C$) où \oplus est la somme vectorielle. Il en résulte structurellement l'impossibilité de détecter une place vide qui formerait le test d'une fin de phase: les réseaux de Petri Place/Transition n'ont pas de *test à zéro*. On identifie ici une limitation structurelle des réseaux de Petri Place/Transition pour décrire des procès de travail impliquant un nombre inconnu au départ de jetons.

Pour clarifier la notion de test, comparons la situation obtenue avec les réseaux Condition/Evènement. Les réseaux Condition/Evènement ne sont pas monotones puisqu'ils sont bornés par construction. L'équation de tir fixe qu'une transition peut marquer une place uniquement si elle est vide. Néanmoins ceci n'est pas un "test" à zéro, car la place sera marquée après le tir, à la différence d'un test qui la laisserait vide.

Du point de vue du langage, les réseaux de Petri Place/Transition occupent une place singulière : ils étendent les langages réguliers mais sont incomparables avec les langages context-free. Si on permet d'une façon ou d'une autre à un réseau de Petri Place/Transition de "détecter" le vide et de le laisser en l'état alors il devient équivalent à une machine de Turing. Sans cela, les réseaux de Petri Place/Transition sont moins expressifs que ces machines. En contrepartie, plus de propriétés sont décidables.

1.2.3 Analyse structurelle des Réseaux Place/Transition

La grande force des réseaux de Petri *Place/Transition* d'un point de vue théorique est que l'on sait décider des propriétés très fortes relatives à l'accessibilité que l'on ne sait pas énoncer pour d'autres systèmes, en particulier les machines de Turing.

Donnons la définition de base qui sert à l'analyse des réseaux de Petri, la séquence de franchissement. Une séquence de franchissement est un mot (Cf. p.145) de transition tel que l'on pourra tirer successivement les transitions indiquées dans le mot.

Définition 1.12 (Séquence de franchissement) Soit $R = \langle P; T; \bullet R; R^\bullet \rangle$ un réseau de Petri. Soit w une séquence de transitions ($w \in T^*$), cette séquence est dite franchissable à partir de M ($M[w >$), ssi il existe une séquence de marquages

W de longueur $|w| + 1$ ($W \in (\mathbb{N}^P)^{|w|+1}$) telle que $M = W(0)$ et

$$\forall i \in [0..|w| - 1], W(i)[w(i) > W(i + 1)]$$

On note le tir d'une séquence de franchissement $M[w > M']$, où $M' = W(|w|)$.

Il est important de remarquer que l'on a défini séparément, d'abord une paire de propriétés Sensibilisation/Tir, puis dans un deuxième temps, la séquence de franchissement. Cette *présentation* est propre aux réseaux de Petri et, de notre point de vue, aussi importante que le moteur de calcul du franchissement lui-même. Une fois muni de cette définition, on sait énoncer une propriété très forte qui montre l'intérêt de ces structures pour l'analyse des structures dynamiques.

Propriété 1.2 (Décidabilité de l'accessibilité) Soit M et M' deux marquages, le problème de savoir s'il existe une séquence franchissable de transition qui mène de M à M' est décidable.

Cette propriété très puissante, n'est démontrée complètement que dans [Reu89]. Un algorithme permettant de trouver cette séquence est aussi contenu dans ce livre. Un point important est de considérer un réseau de Petri comme opérant dans \mathbb{Z} , puis contraint par le fait d'être positif dans \mathbb{Z} . Ceci permet d'éviter le problème de la *soustraction* dans \mathbb{N} qui figure dans la sémantique opérationnelle (Cf. Chapitre II).

Invariants Un autre outil d'analyse très utilisé [Rei85c, Bra83], et qui s'étend aux réseaux de Haut-Niveau [Rei85b, Gen86, Jen96], est la recherche d'équations invariantes impliquant les marquages accessibles.

- Les *invariants de places* permettent de définir des combinaisons linéaires sur ces marquages qui sont invariantes. S'intéresser aux propriétés invariantes est une approche qui dans le contexte de l'algorithmique répartie permet de résoudre les problèmes de sûreté ("*rien d'indésiré ne sera présent*"); la problématique de la vivacité ("*le désiré sera atteint*") ne peut être traitée par cette technique. En quelque sorte c'est une problématique "hors du temps" non seulement continu, mais aussi discret. La vivacité se traite par des techniques de terminaisons, par identification d'une fonction de progrès.
- Les *invariants de transitions* fournissent des séquences de transitions de la forme $M[w > M]$ qui permettent de calculer les *régimes permanents* de réseaux de Petri Place/Transition.

1.2.4 Extensions des Réseaux de Petri

Il existe deux grandes motivations pour chercher des extensions aux Réseaux de Petri Place/Transition ou Condition/Évènement.

Le test à zéro Toute extension réalisant le test à zéro n'est plus équivalente à un réseau de Petri Place/Transition mais à une Machine de Turing. On peut réaliser cette extension de deux façons équivalentes : en spécifiant un contexte interdit (arc inhibiteurs, *forbidden context*), ou en donnant un ordre de "priorité" aux transitions. Ce type d'extension ne concerne que marginalement notre propos. Nous n'allons pas chercher à résoudre notre cas d'étude par cette voie.

La pauvreté symbolique des marquages On peut réaliser des extensions qui permettent de doter les jetons d'une structure de donnée. On peut même munir

les transitions d'un langage spécifiant des conditions (i.e. gardes). Citons les quatre grandes approches de ce type d'extensions: les réseaux algébriques [MV86], les réseaux colorés [Jen96], les réseaux Prédicat/Transition [Gen86] et les réseaux à jetons identifiés [Rei85a]. Ces approches ont été regroupées dans l'appellation *Réseaux de Petri de Haut-Niveau*.

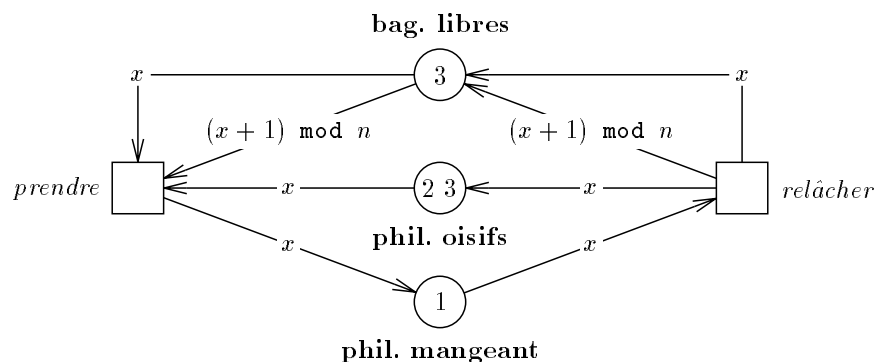
On peut caractériser parmi ces réseaux ceux qui conservent le même pouvoir d'expression que les Réseaux de Petri Place/Transition et aussi leurs propriétés, décidabilité de l'accessibilité, invariants... Ils forment en quelque sorte des "abréviations" ou des "replisages symboliques" de réseaux Place/Transition. C'est ce type spécifique d'extensions sur laquelle nous nous concentrerons et dont nous donnerons une caractérisation précise dans le chapitre III comme extension réciproque des réseaux de Petri Place/Transition en Logique de Réécriture.

Le problème des philosophes

Ce sont des extensions du type des réseaux de Petri de Haut-Niveau que nous allons considérer. Pour ne pas entrer trop dans les détails de ces approches, nous allons les introduire par un exemple, souvent utilisé pour présenter ces formalismes, celui du problème des philosophes. Nous envisageons ce problème dans un cadre non-réparti. Dans ce contexte, une solution pour éviter l'interblocage est d'obliger chaque philosophe à prendre de façon atomique (i.e. indivisible ou "en même temps") ses deux baguettes adjacentes. Dans le cas contraire, si chaque philosophe prend la baguette à sa droite, le système arrive dans un état sans successeur, qui ici n'est pas un résultat mais un blocage que l'on cherche à éviter.

FIG. 1.4 – Enoncé et modèle d'une solution au problème des philosophes à n (marquage pour $n = 3$)

Enoncé : n philosophes pensent et mangent du riz, alternativement. Pour saisir leur pitance, ils doivent être munis de deux baguettes. Les philosophes sont assis à une table ronde et chacun partage une baguette avec chacun de ses deux voisins immédiats.



Note : x prend ses valeurs dans l'ensemble des entiers (références des baguettes et des philosophes).

La figure 1.4 est un modèle d'une solution en *réseaux de Petri Haut-Niveau*[Rei98] de ce problème classique, que les réseaux de Petri résolvent de façon élégante en tirant profit de leur capacité à exprimer la synchronisation. En effet,

une transition peut nécessiter plusieurs jetons “en même temps” pour être tirée. Nous n’allons pas entrer dans les détails de ces extensions, et simplement dire que les jetons sont des tuples et que les arêtes du réseau peuvent être étiquetées par des tuples de variables. Pour être sensibilisée, une transition doit pouvoir prélever autant de jetons qu’indiqué sur les arêtes. Pour indiquer qu’une transition prend (ou dépose) plusieurs jetons dans la même place, on a indiqué plusieurs arêtes de la même source vers la même cible. En outre, comme dans la solution classique de ce problème, on a codé la topologie de la table par une opération arithmétique (**mod**) dont un argument (le nombre de philosophes) est codé “dans le réseau”.

Le marquage représenté décrit l’état d’une table à trois philosophes où le philosophe 1 mange et les philosophes 2 et 3 attendent car il ne reste plus que la baguette 3 de libre.

Deux points essentiels doivent être mentionnés ici, qui forment la base du constat qui motive les améliorations que nous tentons d’apporter à la conception et la spécification de ce type de systèmes:

- Dans la description du marquage, on utilise ostensiblement les symboles 1, 2 et 3 comme les *identités* des philosophes (par l’utilisation de démonstratifs), or rien n’interdit dans le formalisme d’avoir deux “philosophes” qui s’appellent 2. Par exemple, considérer que le marquage de la place **phil.oisifs** est (2 2 3) n’empêche en rien de dérouler le comportement du réseau. Donc le symbole 2 n’est pas “dans le formalisme” une identité, c’est uniquement “dans le réseau” qu’elle est conçue et utilisée comme telle. Et ceci suppose que “dans le marquage”, cette conception est présente. La perception de ces trois niveaux “dans le formalisme”, “dans le réseau” et “dans le marquage” est essentielle à la compréhension de notre problématique. Notre objectif est de réaliser le maximum de travail “dans le formalisme”.
- La situation ainsi modélisée est extrêmement statique : les philosophes sont là de toute éternité et alternativement pensent et mangent. L’aspect statique est aggravé par le fait que la taille de la table, n , est “codée en dur” dans le réseau. Comment parvenir à rendre ce système plus “dynamique” ?

Nous allons considérer deux types d’extensions “dynamiques” à la solution classique : l’une appelée “le pique-nique” n’impose plus de contrainte “spatiale” à la préhension des baguettes, l’autre appelée “le bal” permet de coder les cercles formés par les philosophes comme faisant partie du marquage, ce qui permet de les rendre dynamiques par fusion/séparation.

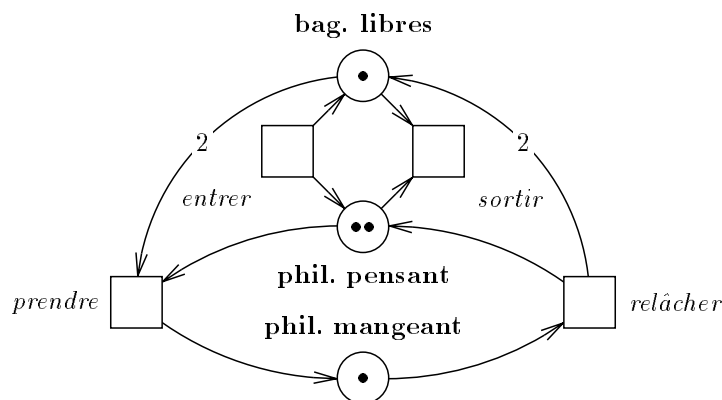
Le pique-nique

La première version que nous donnons de ce problème procède à la fois d’une réduction et d’une extension du problème original: “*des philosophes pique-niquent avec un sac de baguettes. Tant qu’il reste des baguettes dans le sac, ils peuvent en prendre une paire et ensuite les remettre dans le sac*”. La réduction réside dans l’oubli des identités des philosophes et des baguettes représentées par des entiers dans la version de référence. Cet oubli est rendu possible par la possibilité pour un philosophe de prendre ses baguettes dans un sac sans contrainte de localité.

Ceci a une conséquence fondamentale: comme les jetons sont redevenus “anonymes”, le modèle résultant est un réseau de Petri Place/Transition. Et donc les jetons n’existent plus en tant que tels, et sont la représentation d’entiers naturels: ajouter et enlever des éléments ne pose plus guère de difficulté dans ce contexte. Ceci est donc une extension du problème original, des éléments peuvent entrer et sortir du système. La réduction (l’oubli des identités) permet donc d’envisager une extension : le dynamisme des effectifs.

Le réseau de Petri Place/Transition donné dans la figure 1.5 détaille ce système avec les deux transitions respectivement d'entrée et de sortie où l'on fait respectivement entrer et sortir de concert un philosophe et une baguette, pour conserver la parité des effectifs. Le réseau résultant est dit "non borné" : un nombre arbitraire de philosophes et de baguettes peut être dépassé par une séquence de franchissements. Mais cette propriété ne nuit pas au fonctionnement du réseau lui-même. Elle peut poser une difficulté si l'on cherche à évaluer par le calcul l'ensemble des états accessibles qui est lui aussi de cardinal non borné mais dénombrable. En outre le nombre de transitions sensibilisées à chaque instant reste lui borné par le nombre de transitions, ici 4.

FIG. 1.5 – Le pique-nique des philosophes



Le bal

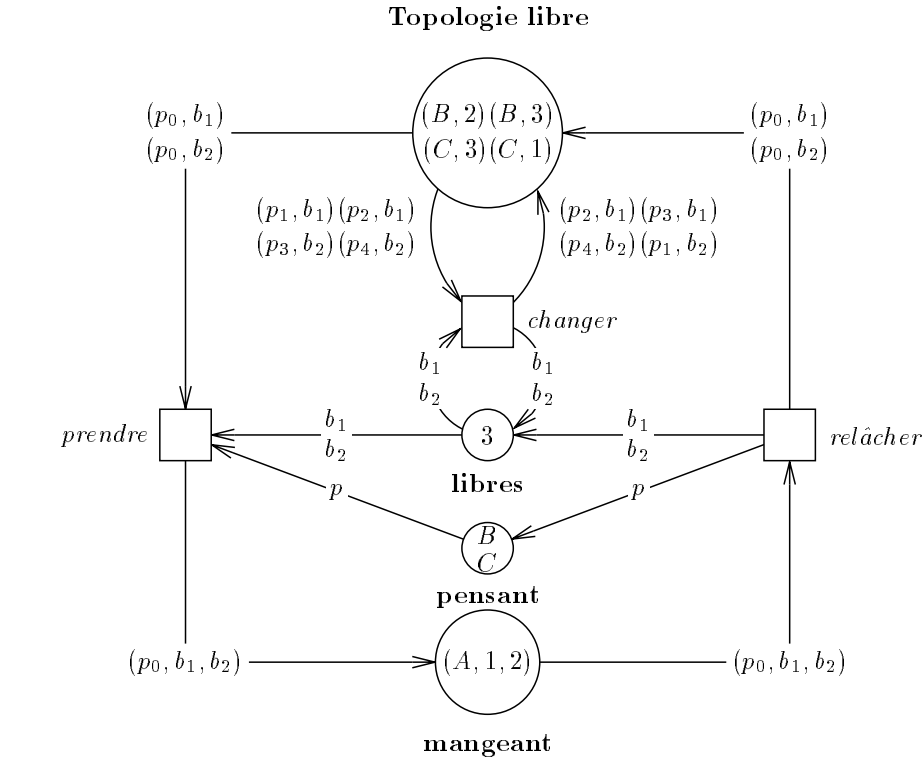
Nous définissons une autre variante du réseau classique, qui utilise l'idée des Trains de [Gen86]: coder un graphe par un multi-ensemble de paires. Ceci va définir une topologie par l'ensemble des paires (philosophe,baguette) contenues dans la place **Topologie Libre**:

- les philosophes, référencés par p_i dans la relation de flot et par des lettres majuscules (A, B, C) dans les marquages,
- les baguettes, référencées par b_i dans la relation de flot et par des chiffres arabes ($1, 2, 3$) dans les marquages.

Ainsi nous nous affranchissons du codage de la topologie dans le réseau par une construction arithmétique comme celle de la version de référence avec la division euclidienne (Cf. figure 1.4). La topologie est codée dans le marquage du réseau, d'une part pour sa partie disponible dans la place du haut et pour sa partie utilisée dans la place du bas. Remarquons ici que la représentation graphique donnée dans la figure 1.6 atteint une limite de lisibilité passée laquelle la représentation graphique n'est plus un gage de simplification. Le marquage donné correspond à une situation équivalente de celle du réseau de référence : Le philosophe A mange avec les baguettes 1 et 2, les philosophes B et C pensent et pourraient, si elles étaient libres, respectivement utiliser les baguettes $\{2, 3\}$ et $\{3, 1\}$.

Si nous considérons la transition *prendre*, elle stipule qu'un philosophe pensant p_0 peut prendre les baguettes b_1 et b_2 si et seulement si ces baguettes sont libres (b_1

FIG. 1.6 – Le bal des philosophes : une topologie dynamique



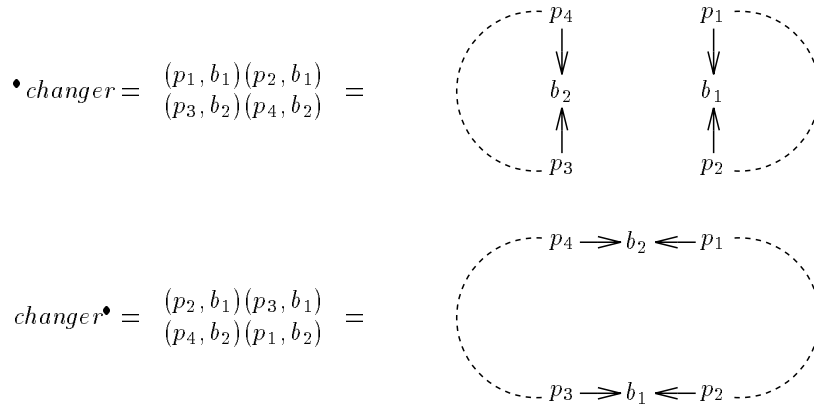
et b_2), et qu'il est en relation avec elles ((p_0, b_1) et (p_0, b_2)). Après le tir de *prendre*, les trois "objets" forment un "complexe" mangeant $((p_0, b_1, b_2))$.

Cette version permet d'envisager une topologie dynamique. Nous introduisons à cet effet une transition qui permet à deux tables de fusionner : la règle *changer*. Cette règle est obtenue par permutation circulaire des indices des philosophes. Nous avons construit la règles à quatre philosophes en partant de l'insertion dans un cycle. On peut construire une règle plus simple à deux philosophes en suivant le même principe de permutation circulaire des indices. La figure 1.7 représente sous forme de "graphe" la partie topologique de la transition, les pointillés sont là comme commentaires pour aider l'intuition du "comment" deux tables fusionnent : si les pointillés complètent de chaque cotés un cycle avant le tir, ils forment un seul cycle après le tir. Il suffit aux pointillés "représentant les tables" de former le cycle avant le tir de la transition ($p_1 - - - p_4$ et $p_2 - - - p_3$) pour appréhender comment la même transition peut séparer une table en deux.

Pour résumer, avec les réseaux de Petri et leurs extensions généralement utilisées, on peut décrire deux types de systèmes. D'une part, des systèmes (de type "pique-nique") à effectif dynamique où les composants ne sont pas identifiés et d'autre part, des systèmes à effectif statique (de type "bal") où les composants sont identifiés.

Notre objectif est de parvenir à spécifier des systèmes à effectif dynamique où les composants sont identifiés. Pour le problème des philosophes, ce système est appelé le "gala". Il s'avère que les extensions Haut-Niveau classiques des réseaux de Petri ne permettent pas de réaliser ce type de systèmes. La situation est un peu plus confuse que cela en réalité, les sémantiques de ces réseaux n'excluent pas formellement cette possibilité mais aucun exemple ne traite de ce type de construction.

Au-delà des extensions dont nous venons de parler, la communauté des réseaux de Petri a produit un grand nombre d'extensions, plus ou moins dédiées à certaines

FIG. 1.7 – Représentation graphique de la transition **change**.

limitations structurelles des réseaux de Petri Place/Transition élémentaires. A titre d'exemple, nous en citerons deux qui nous ont influencés pour leur problématique:

Les réseaux à jeton/objets [Val98] Ce formalisme propose deux niveaux de réseaux de Petri, l'un pour l'architecture et l'autre pour le comportement. Cette construction nous a influencés car elle sépare clairement topologie et comportement comme il nous semble nécessaire de le faire [Sch93]. Mais cette approche souffre d'une écriture trop proche de la structure opérationnelle, ce qui rend sa manipulation formelle malaisée. En outre, la sémantique pose des problèmes liés à l'identification partielle des jetons par leur image en tant que comportement.

Les réseaux auto-modifiants [Val78] permettent à une transition d'être paramétrée par un marquage. Nous verrons que ce type de construction peut être formulé en Logique de Réécriture (cf. 4.1). Les réseaux configurables [BO98] sont une sous-classe de ces réseaux pour lesquels on sait prouver que le réseau reste borné.

L'option de développer notre propre extension correspondant à nos besoins a d'abord été suivie [GV99]. Mais ce type d'approche n'était plus d'actualité, un consensus fort existait sur le fait que les modèles connus suffisaient au besoin. L'échec de cette première tentative consommé, nos recherches nous amenèrent à considérer et à étudier deux formalismes très différents:

La Logique de réécriture: ce formalisme est muni d'une sémantique *compacte* et très puissante et dispose d'un outil permettant de tester opérationnellement les constructions réalisées. Sa grande puissance d'expression en fait un client potentiel pour unifier les différentes extensions des réseaux de Petri. Notre analyse de ce formalisme montrera que si les problèmes liés à la *multiplicité* et la *localité* sont résolus, le problème de l'*identité* reste entier;

Les Grammaires de Graphes (par double somme): ce formalisme est le seul, à notre connaissance, qui traite de façon compacte et intégrée le problème de l'*identité* des objets et propose une construction permettant d'assurer l'unicité d'une référence sans intervention du spécifieur.

Les deux parties suivantes, qui détaillent ces deux formalismes, seront illustrées par le problème du "gala des philosophes".

1.3 La Logique de Réécriture

La Logique de Réécriture est proposée dans [Mes92] comme une sémantique concurrente des systèmes de réécriture de termes où les opérateurs peuvent être associatifs, commutatifs, idempotents ou dotés d'un élément neutre à gauche ou à droite (ou n'importe quelle combinaison de ces propriétés d'opérateurs). Ces possibilités sont écrites dans une sémantique d'une grande simplicité au regard de sa puissance d'expression. Cette construction est aussi une généralisation de la machine chimique [BM91] dont tous les exemples classiques s'écrivent (presque syntaxiquement) en MAUDE [CDE⁺99]. Tous les exemples que nous détaillerons ici sont construits avec un opérateur $(_ \oplus _)$ qui est associatif, commutatif et doté d'un élément neutre (**rien**). En référence à la machine chimique et pour illustrer le caractère non parenthésé et non ordonné d'un terme construit par $_ \oplus _$, la sorte de ce type de terme est appelée **soupe**.

Cette partie est organisée comme suit: d'abord nous introduisons la notion de terme et de théorie, nous donnons ensuite la définition de la dynamique associée et l'expression de la réflectivité de cette logique. Nous développons ensuite la notion de *terme de preuve*, les identifiants des transformations. Nous donnons un exemple détaillé d'une preuve d'accessibilité sous forme de séquents puis sous forme de termes de preuve. Cette preuve est importante en elle-même car elle montre que l'admission d'éléments inconnus dans un système est possible dans cette logique et nous étudierons la modalité de cette admission à la fin de la section.

1.3.1 Termes et théorie

La *Logique de Réécriture* [Mes92] repose sur les définitions des *algèbres universelles* [EM90], en premier lieu la définition d'une signature multi-sortes (i.e. *many sorted*) et d'un terme :

Définition 1.13 (Signature multi-sortes [EM90]) Soient \mathbf{S} un ensemble de sortes, \mathbf{op} un ensemble de symboles fonctionnels et X un ensemble de variables. A chaque variable de X est associée une sorte. A chaque symbole $f \in \Sigma$ est associée un entier naturel n , appelé son arité et une suite de $n + 1$ sortes (s_1, \dots, s_{n+1}) appelée profil du symbole s , on note:

$$f : k_1 \times \dots \times k_n \rightarrow k_{n+1}$$

Un symbole fonctionnel de constante sera noté $f : \epsilon \rightarrow k_1$. Si F est un ensemble de symboles fonctionnels, $F : k_1 \times \dots \times k_n \rightarrow k_{n+1}$ signifie que tous les éléments de F ont le même profil.

A partir de cette définition fondatrice de la théorie dite des “algèbres universelles”, on peut construire deux grands types de structures. On peut soit construire un ensemble de termes, (l'ensemble des “mots parenthésés” où chaque parenthèse ouvrante est précédée d'un opérateur), soit construire une algèbre (qui associe à chaque sorte un ensemble et à chaque opérateur une fonction). Les deux seront exploités dans ce mémoire: cette section repose sur la notion de termes, la section suivante détaillera la notion d'algèbre (Cf. 1.4.6).

Exemple 1.1 La signature de base n'a qu'une seule sorte **soupe**, un symbole fonctionnel binaire $(_ \oplus _)$, une constante **rien** et un ensemble de constantes.

Définition 1.14 (Termes [Cir00]) L'ensemble des termes $\mathcal{T}^\Sigma(X)$ et la sorte k d'un terme t , noté $t : k$, sont définis simultanément par :

- pour toute variable $x \in X$, associée à un symbole de sorte k , $x \in \mathcal{T}^\Sigma(X)$ et $x : k$.

- pour tout symbole $f : k_1 \times \dots \times k_n \rightarrow k_{n+1}$ et pour tous termes $t_1 : k_1, \dots, t_n : k_n$, $f(t_1, \dots, t_n) \in \mathcal{T}^\Sigma(X)$ et $f(t_1, \dots, t_n) : k_{n+1}$.

Un terme peut être vu comme un arbre étiqueté, pour lesquels les fils d'un nœud sont ordonnés. Pour la signature donnée en exemple, la structure peut se percevoir comme un arbre binaire dont les nœuds sont étiquetés par $(_ \oplus _)$ et les feuilles par les constantes. Il s'ensuit que la donnée d'un mot d'entiers permet de localiser de façon univoque un sous-terme [BN98]. Ce mot d'entier est interprété en prenant le i^{me} sous-terme à chaque itération jusqu'à arriver au mot vide où l'on aura atteint le sous-terme recherché. Cette construction classique porte le nom de *position*, c'est une *identité implicite*.

Ceci permet de construire explicitement la classe d'équivalence utilisée dans la définition suivante. En effet, une paire de termes définit une relation plus vaste parce qu'on autorise la réécriture à n'importe quelle position [BN98]. C'est pour des raisons de conflit de noms, qui apparaîtront plus tard, que nous avons choisi le mot "axiome" [Cir00], les mots "équation" [Jan97] et "identité" [BN98] auront un sens différent dans ce mémoire.

Définition 1.15 (Σ -Axiomes [EM90, BN98]) *Un Σ -axiome est une paire de termes $\mathcal{T}^\Sigma(X)$, notée $s \approx t$. Un ensemble \mathcal{E} de Σ -axiomes définit une relation d'équivalence sur $\mathcal{T}^\Sigma(X)$ — en autorisant l'utilisation de l'axiome à n'importe quelle position —. On note $[t]$ la \mathcal{E} -classe d'équivalence du terme t modulo les axiomes dans \mathcal{E} , et $\mathcal{T}_\mathcal{E}^\Sigma(X)$ l'ensemble des classes d'équivalence de termes.*

La plupart des systèmes d'équations utilisés s'applique aux opérateurs binaires (e.g. $(_ \oplus _)$), et quatre équations sont principalement utilisées, ici définies avec x, y et z comme variables de sorte **soupe**:

Associativité $(x \oplus (y \oplus z)) \approx ((x \oplus y) \oplus z)$, cette équation permet de considérer la réécriture de chaînes. Cette structure est fondamentale pour le calcul séquentiel car la plupart des structures de données utilisées mis à part les arbres sont des chaînes (liste, pile, file). Historiquement, la réécriture de chaînes (i.e. de mots) précède les autres formes de réécriture. En 1910, Axel Thue a utilisé des systèmes de réécriture symétrique sur des chaînes de caractères pour poser l'ensemble des problèmes connus dans ce contexte.

Commutativité $(x \oplus y) \approx (y \oplus x)$, conjointement à la précédente ces équations permettent de considérer la réécriture de multi-ensemble. Une conséquence essentielle de ce passage au quotient est que nous ne disposons plus de positions univoques dans ce cas : les sous-termes séparés par un opérateur commutatif ont la même position. Donc après passage au quotient, la position ne correspond plus à une identité mais à une localité. L'adéquation de cette structure à la problématique du calcul réparti et/ou concurrent a fait l'objet de nombreuses publications [Mes92, BB92, CMR94, Eng96], dans lesquelles une convergence entre multi-ensemble, machine chimique et réseaux de Petri Place/Transition a été mise au jour au début des années 90. Mais, dès le début des années 80, les réseaux de Petri Place/Transition étaient qualifiés de système *semi-thuein commutatif* [Bra83], ce qui signifie des systèmes de réécriture sur les équations AC, "semi-" signifiant "orienté" ou "non-symétriques" et "thuein" impliquant l'associativité, puisque Thue travaillait sur les mots.

Élément neutre & Idempotence $(x \oplus \text{rien}) \approx x$ et $(x \oplus x) \approx x$. Ces deux équations posent un certain nombre de difficultés car elles contiennent un membre réduit à une variable et ne préservent pas le nombre de symboles fonctionnels. Leur usage devrait être réservé aux constructions mathématiques mais pas aux structures opérationnelles.

Définition 1.16 (Théorie de réécriture) Une théorie de réécriture est un quintuplet $\mathcal{R} = (\Sigma, \mathcal{E}, \mathcal{L}, X, \mathcal{R})$ formé de :

- une signature Σ
- un ensemble d'axiomes \mathcal{E}
- un ensemble d'étiquettes \mathcal{L}
- un ensemble de variables X
- un ensemble de Règles de réécriture \mathcal{R} qui est une partie de $\mathcal{L} \times (\mathcal{T}_{\mathcal{E}}^{\Sigma}(X))^2$ où X est un ensemble dénombrable [CDE⁺ 99] de variables. On note une règle $(r, ([t(x_1, \dots, x_n)], [t'(x_1, \dots, x_n)]))$ de la façon suivante [CDE⁺ 99]:

$$(\mathbf{r}(\mathbf{x}_1, \dots, \mathbf{x}_n) : [t(x_1, \dots, x_n)] \rightarrow [t'(x_1, \dots, x_n)])$$

Cette dernière écriture nécessite quelques explications : on ne note que les variables intervenant dans la définition des termes gauche et droite, bien que ces variables ne soient pas nécessairement occurrentes dans les deux termes.

De notre point de vue, un intérêt essentiel de la réécriture est présent dans la considération suivante : notre signature-exemple permet de reformuler la limite bien connue entre Machine de Turing et Réseaux de Petri Place/Transition en stipulant, ou non, la présence de l'axiome de commutativité.

Associativité Dotée de l'axiome d'associativité, notre signature d'exemple permet d'associer à toute machine de Turing, un système de réécriture sans variables. Pour la réécriture A sans variables, on sait donc prouver l'indécidabilité de l'arrêt.

Commutativité Tout système de réécriture sans variable sur notre signature dotée des axiomes d'associativité et de commutativité est un réseau de Petri Place/Transition, l'explicitation complète de ce fait forme une partie du chapitre III. Pour la réécriture AC sans variables, on sait donc prouver la décidabilité de l'accessibilité.

Ce n'est donc pas la réécriture elle-même qui permet, ou non, la preuve de propriétés mais les caractérisations que l'on produit sur les systèmes de réécriture qui nous permettent d'inférer des propriétés.

1.3.2 Dynamique et Réflexivité

Définition 1.17 (Dynamique d'une théorie) Soit $\mathcal{R} = (\Sigma, \mathcal{E}, \mathcal{L}, X, \mathcal{R})$ une théorie de réécriture, on définit la dynamique \rightarrow d'une théorie de réécriture comme une relation sur $\mathcal{T}_{\mathcal{E}}^{\Sigma}(X)$ par les quatre déductions suivantes données sous forme de séquents [Plo81, Gir87] dans la figure 1.8.

Liens avec la machine chimique : Le lien de conception entre la machine chimique et la logique de réécriture est accrédité par le fait que [Mes92] et [BB92] se citent mutuellement. En outre [BB92] renvoie à [Mes92] pour “des définitions algébriques très précises de leurs constructions”. En effet, la logique de réécriture voit la “loi chimique” — “une réaction peut intervenir librement dans toute solution” — de [BB92] :

$$\frac{S \rightarrow S'}{S \oplus S'' \rightarrow S \oplus S''}$$

FIG. 1.8 – Dynamique associée à une théorie de réécriture

1. **Réflexivité**: Pour chaque $[t] \in \mathcal{T}_{\mathcal{E}}^{\Sigma}(X)$

$$R \frac{}{[t] \rightarrow [t]}$$

2. **Congruence**: Pour chaque f , symbole fonctionnel d'arité n :

$$C \frac{[t_1] \rightarrow [t'_1], \dots, [t_n] \rightarrow [t'_n]}{[f(t_1, \dots, t_n)] \rightarrow [f(t'_1, \dots, t'_n)]}$$

3. **Application**: Pour chaque règle $(\mathbf{r} : [t(x_1, \dots, x_n)] \rightarrow [t'(x_1, \dots, x_n)])$, où a/b signifie le remplacement dans le terme de $b \in X$ par $a \in \mathcal{T}_{\mathcal{E}}^{\Sigma}(X)$, l'ensemble des remplacements étant atomiques [CDE⁺99]:

$$A \frac{[w_1] \rightarrow [w'_1], \dots, [w_n] \rightarrow [w'_n]}{[t(w_1/x_1, \dots, w_n/x_n)] \rightarrow [t'(w'_1/x_1, \dots, w'_n/x_n)]}$$

4. **Transitivité**:

$$T \frac{[t_1] \rightarrow [t_2], [t_2] \rightarrow [t_3]}{[t_1] \rightarrow [t_3]}$$

comme une instance particulière d'une construction plus générale: la *congruence* (voir Figure 1.8). En effet, si on exprime l'axiome de congruence pour l'opérateur $(- \oplus -)$ avec comme second membre une identité, on obtient :

$$\frac{[t_1] \rightarrow [t'_1], [t_2] \rightarrow [t'_2]}{[(t_1 \oplus t_2)] \rightarrow [(t'_1 \oplus t'_2)]}$$

De la même façon la “loi de membrane” est une congruence sur un opérateur unaire $\{_ \} : \text{soupe} \rightarrow \text{soupe}$. Ce type d'opérateur, que nous appellerons “récurifs” sera développé dans le chapitre IV pour spécifier notre cas d'étude.

Réflexivité : Une caractéristique fondamentale de la logique de réécriture est d'être *réflexive*. Une contribution essentielle de [CM96] est d'avoir caractérisé précisément ce que cela signifiait, avant de le prouver pour la logique de réécriture.

Propriété 1.3 (Réflexivité) *Il existe une théorie finie de réécriture \mathcal{U} dite universelle. Cela signifie qu'il existe*

- Une représentation finie $\overline{\mathcal{R}}$ sous forme de terme de \mathcal{U} de toute théorie finie \mathcal{R}
- une représentation finie \overline{t} sous forme de terme de \mathcal{U} de tout terme t fini de \mathcal{R}
- une représentation $\langle \overline{\mathcal{R}}; \overline{t} \rangle$ de tout couple (\mathcal{R}, t)

vérifiant $(\mathcal{R} : t \rightarrow t') \iff (\mathcal{U} : \langle \overline{\mathcal{R}}; \overline{t} \rangle \rightarrow \langle \overline{\mathcal{R}}; \overline{t'} \rangle)$

Il ne faut pas confondre la “réflexivité” et la “réflexivité”.

Comme \mathcal{U} est finie, on peut la représenter comme un terme d'elle-même et réaliser une “tour réflexive” (ang. *reflective tower* [CDE⁺99]) avec un nombre arbitraire de “réflexions”.

$$(\mathcal{R} : t \rightarrow t') \iff (\mathcal{U} : \langle \overline{\mathcal{R}}; \overline{t} \rangle \rightarrow \langle \overline{\mathcal{R}}; \overline{t'} \rangle) \iff (\mathcal{U} : \langle \overline{\mathcal{U}}; \overline{\langle \overline{\mathcal{R}}; \overline{t} \rangle} \rangle \rightarrow \langle \overline{\mathcal{U}}; \overline{\langle \overline{\mathcal{R}}; \overline{t'} \rangle} \rangle) \dots$$

Le pique-nique des philosophes

Cette version ayant été modélisée par un réseau de Petri Place/Transition, l'écriture de cet exemple va nous permettre de montrer comment traduire un tel réseau en Logique de Réécriture. Les réseaux de Petri Place/Transition sont l'exemple type des réécritures ACI. Le problème de l'équivalence formelle entre les deux écritures fait l'objet d'un développement complet dans le chapitre III de ce mémoire.

Par exemple, pour le pique-nique des philosophes, on a trois constantes qui correspondent aux trois places du réseau donné dans la figure 1.5 : **{Libre, Oisif, Encas}**, où :

Libre représente une baguette libre;

Oisif représente un philosophe pensant;

Encas représente un philosophe mangeant.

Un marquage est une classe de termes modulo associativité/commutativité/élément neutre. Le marquage initial de la figure 1.5 est représenté par la classe de termes suivante :

$$[\text{Libre} \oplus \text{Oisif} \oplus \text{Oisif} \oplus \text{Encas}] = \underbrace{\{(((\text{Libre} \oplus \text{Oisif}) \oplus \text{Oisif}) \oplus \text{Encas}), \dots\}}_{36 \text{ termes}}$$

Une transition peut s'exprimer comme une paire de marquages, et donc le réseau peut s'écrire par l'ensemble suivant de transitions :

get : $[\text{Oisif} \oplus \text{Libre} \oplus \text{Libre}] \rightarrow [\text{Encas}]$

release : $[\text{Encas}] \rightarrow [\text{Oisif} \oplus \text{Libre} \oplus \text{Libre}]$

leave : $[\text{Oisif} \oplus \text{Libre}] \rightarrow [\text{rien}]$

enter : $[\text{rien}] \rightarrow [\text{Oisif} \oplus \text{Libre}]$

Nous retrouvons ici une expression d'un Réseau Place/Transition très proche de ce que [Bra83] appelle la "grammaire associée à un réseau".

Telle quelle, la dynamique de l'exemple est non terminante et confluyente. La confluence s'obtient en remarquant que toutes les règles sont symétriques, revenir au point de départ est donc toujours possible. L'absence de membre gauche dans la règle **enter** implique que la dynamique est non terminante : **enter** est toujours possible. Dans la classification introduite dans la section 1.1, une dynamique confluyente et non-terminante est appelée un *protocole*.

Le bal des philosophes

Cette modélisation va nous permettre de faire intervenir le problème de la représentation des références, d'abord dans le cas où l'ensemble des références reste invariant lors d'un déroulement.

Informellement, comme dans ce cas les jetons sont des tuples, il suffit de préfixer ce tuple par le nom de la place pour obtenir un terme. Chaque place devient alors un opérateur qui à chaque tuple associe un terme de sorte **soupe**. Cette approche nous permet de définir une théorie qui spécifie le "bal des philosophes" conformément au réseau de la figure 1.6 à un renommage des places/opérateurs près. La signature est paramétrée par deux ensembles de constantes, qui forment l'ensemble des références de philosophes (P) et l'ensemble des références de baguettes (B).

Signature $\Sigma(P, B) \stackrel{\text{def}}{=} \langle \mathbf{S}; \text{op} \rangle$ où :

Sortes $\{\text{Philosophe}, \text{Baguette}, \text{soupe}\}$.

| | | | | | | |
|------------|---|----------------------------|---|---|---------------|-------------------|
| Opérateurs | { | rien | : | ϵ | \rightarrow | soupe |
| | | $(- \oplus -)$ | : | $\text{soupe} \times \text{soupe}$ | \rightarrow | soupe |
| | | $(- \hookrightarrow -)$ | : | $\text{Philosophe} \times \text{Baguette}$ | | |
| | { | | | | \rightarrow | soupe |
| | | Libre ($-$) | : | Baguette | \rightarrow | soupe |
| | | Oisif ($-$) | : | Philosophe | \rightarrow | soupe |
| | | Encas ($-, -, -$) | : | $\text{Philosophe} \times \text{Baguette} \times \text{Baguette}$ | | |
| | | | | | \rightarrow | soupe |
| | { | P | : | ϵ | \rightarrow | Philosophe |
| | | B | : | ϵ | \rightarrow | Baguette |

Ensemble de variables

$$X_{\text{Philosophe}} = \{p_i\}_{i \in [0..4]}, X_{\text{Baguette}} = \{b_i\}_{i \in [0..4]}, X_{\text{soupe}} = \{\alpha, \beta, \gamma\}$$

Règles $\mathcal{R} = \{$

get(p_0, b_1, b_2) :

$$[(p_0 \hookrightarrow b_1) \oplus (p_0 \hookrightarrow b_2) \oplus \text{Oisif}(p_0) \oplus \text{Libre}(b_1) \oplus \text{Libre}(b_2)]$$

$$\rightarrow [\text{Encas}(p_0, b_1, b_2)].$$

release(p_0, b_1, b_2) :

$$[\text{Encas}(p_0, b_1, b_2)]$$

$$\rightarrow [(p_0 \hookrightarrow b_1) \oplus (p_0 \hookrightarrow b_2) \oplus \text{Oisif}(p_0) \oplus \text{Libre}(b_1) \oplus \text{Libre}(b_2)].$$

change($p_1, p_2, p_3, p_4, b_1, b_2$) :

$$[(p_1 \hookrightarrow b_1) \oplus (p_2 \hookrightarrow b_1) \oplus (p_3 \hookrightarrow b_2) \oplus (p_4 \hookrightarrow b_2) \oplus$$

$$\text{Libre}(b_1) \oplus \text{Libre}(b_2)]$$

$$\rightarrow [(p_2 \hookrightarrow b_1) \oplus (p_3 \hookrightarrow b_1) \oplus (p_4 \hookrightarrow b_2) \oplus (p_1 \hookrightarrow b_2) \oplus$$

$$\text{Libre}(b_1) \oplus \text{Libre}(b_2)].$$

Comme pour le réseau algébrique équivalent donné dans la figure 1.6, nous avons choisi comme références pour les philosophes les lettres majuscules ($P = \{A, B, C, D, E\}$) et comme références pour les baguettes les représentations des entiers naturels ($B = \{1, 2, 3, 4, 5\}$). Avec ces ensembles de constantes, le marquage donné dans la figure 1.6 est une représentation de la classe de termes clos donnée ci-dessous:

$$\left[\begin{array}{c} (B \hookrightarrow 2) \oplus (B \hookrightarrow 3) \oplus (C \hookrightarrow 3) \oplus (C \hookrightarrow 1) \\ \oplus \text{Libre}(3) \oplus \text{Oisif}(B) \oplus \text{Oisif}(C) \oplus \text{Encas}(A, 1, 2) \end{array} \right] \quad (1.1)$$

Le gala des philosophes

Cette dernière modélisation intègre les deux précédentes. C'est cette version qui servira de support aux analyses ultérieures. En partant du bal des philosophes qui intègre la notion de référence, nous rajoutons les deux règles du pique-nique avec des références.

enter Un philosophe rentre "en tenant deux fois" la même baguette. Ceci permet l'utilisation ultérieure de la règle **change** pour insérer cette paire dans une table. Cette règle rend le système infiniment branché à cause de la variable libre à droite.

$$\text{enter}(p_0, b_0) : [\text{rien}] \rightarrow [(p_0 \hookrightarrow b_0) \oplus (p_0 \hookrightarrow b_0) \oplus \text{Oisif}(p_0) \oplus \text{Libre}(b_0)]$$

leave Symétrique de la première, Un philosophe sort "en tenant deux fois" la même baguette.

$$\text{leave}(p_0, b_0) : [(p_0 \hookrightarrow b_0) \oplus (p_0 \hookrightarrow b_0) \oplus \text{Oisif}(p_0) \oplus \text{Libre}(b_0)] \rightarrow [\text{rien}]$$

L'idée fondamentale contenue dans le système de règles, est que l'on a séparé le problème de l'admission dans le système du problème de l'insertion dans une table déjà formée. Les versions précédentes [KLG93] mélangeaient les deux aspects du problème par une insertion directe dans le cycle des philosophes, rendant l'étude plus ardue. Nous allons maintenant étudier plus précisément la théorie spécifiée pour le gala des philosophes.

1.3.3 Séquents et Sémantique concurrente

Le problème de l'accessibilité s'exprime comme suit: soit \mathcal{R} une théorie de réécriture, soit $(a, b) \in \mathcal{T}_{\mathcal{E}}^{\Sigma}(\emptyset)^2$, a-t-on $a \rightarrow b$ dans la dynamique associée à la théorie \mathcal{R} ? Par exemple,

“un état où un philosophe mange est-il accessible depuis **rien**?”

Nous instancions notre question avec la paire de termes ci-dessous. Cette paire est composée de **[rien]** (car **rien** est l'élément neutre de $(_ \oplus _)$) et d'un terme décrivant la situation où le philosophe A mange avec les baguettes 1 et 2 pendant que B pense.

$$\begin{array}{c} \text{(Cf. figure 1.9)} \\ \hline \left[\begin{array}{c} \text{rien} \\ \oplus \text{rien} \end{array} \right] \\ \rightarrow \left[\begin{array}{c} 0\text{isif}(B) \oplus (B \hookrightarrow 1) \oplus (B \hookrightarrow 2) \\ \oplus \text{Encas}(A, 1, 2) \end{array} \right] \end{array}$$

Prouver que la paire (a, b) est accessible revient à trouver un séquent prouvant $a \rightarrow b$ [Gir97]. Les “feuilles” de ce séquent sont soit des réflexivités soit des applications n'impliquant pas de variables. Comme ici toutes les applications impliquent des variables, les feuilles de “l'arbre de preuve” sont toutes des réflexivités.

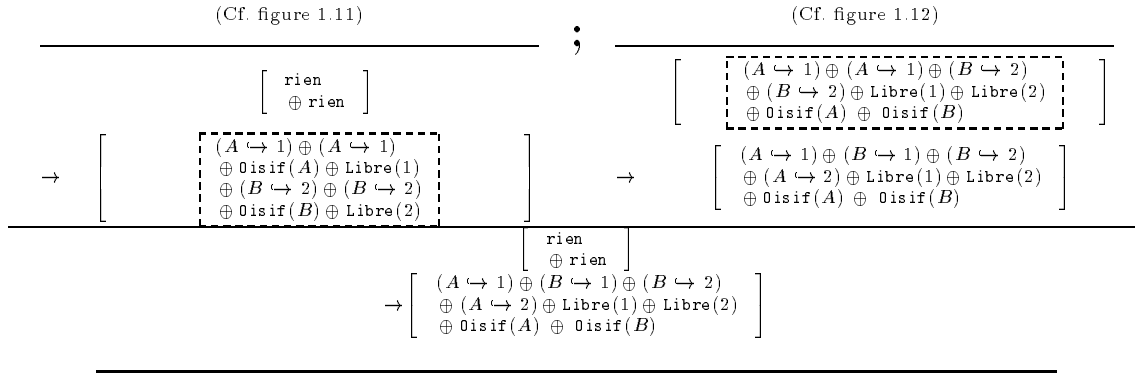
- la première étape de la preuve consiste à dire que l'on va passer par un état intermédiaire où deux philosophes “oisifs” partagent deux baguettes “libres”. Ce terme intermédiaire est donné dans la figure 1.9;

FIG. 1.9 – Un philosophe mange depuis l'état **rien**

$$\begin{array}{c} \text{(Cf. figure 1.10)} \qquad ; \qquad \text{(Cf. figure 1.14)} \\ \hline \left[\begin{array}{c} \text{rien} \\ \oplus \text{rien} \end{array} \right] \qquad ; \qquad \left[\begin{array}{c} 0\text{isif}(B) \oplus (B \hookrightarrow 1) \oplus (B \hookrightarrow 2) \\ \oplus (A \hookrightarrow 1) \oplus (A \hookrightarrow 2) \oplus 0\text{isif}(A) \\ \oplus \text{Libre}(1) \oplus \text{Libre}(2) \end{array} \right] \\ \rightarrow \left[\begin{array}{c} (A \hookrightarrow 1) \oplus (B \hookrightarrow 1) \oplus (B \hookrightarrow 2) \\ \oplus (A \hookrightarrow 2) \oplus \text{Libre}(1) \oplus \text{Libre}(2) \\ \oplus 0\text{isif}(A) \oplus 0\text{isif}(B) \end{array} \right] \qquad \rightarrow \left[\begin{array}{c} 0\text{isif}(B) \oplus (B \hookrightarrow 1) \oplus (B \hookrightarrow 2) \\ \oplus \text{Encas}(A, 1, 2) \end{array} \right] \\ \hline \left[\begin{array}{c} \text{rien} \\ \oplus \text{rien} \end{array} \right] \\ \rightarrow \left[\begin{array}{c} 0\text{isif}(B) \oplus (B \hookrightarrow 1) \oplus (B \hookrightarrow 2) \\ \oplus \text{Encas}(A, 1, 2) \end{array} \right] \end{array}$$

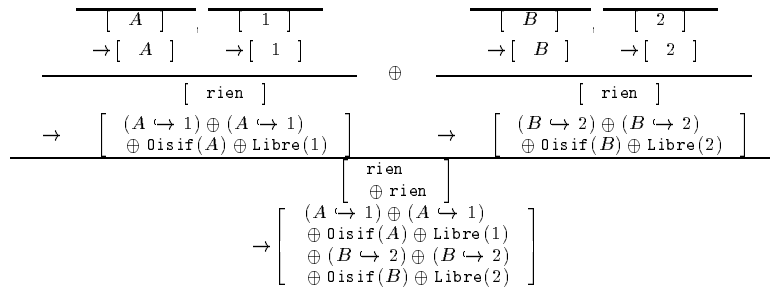
- la figure 1.10 décrit comment atteindre ce terme intermédiaire à partir de **rien**. On décompose cette preuve en deux sous-preuves que l'on met bout à bout par transitivité. Ceci suppose que les deux classes de termes entourées par la boîte en pointillée sont les mêmes; elles peuvent être syntaxiquement différentes, mais il ne faut pas oublier que cette égalité se définit modulo la commutativité de $(_ \oplus _)$;

FIG. 1.10 – Deux philosophes rentrent et forment une table



- une particularité intéressante de notre preuve d’accessibilité apparaît lors de l’admission d’une paire philosophe/baguette (Figure 1.11). Outre le fait que les variables citées à droite ne sont pas citées à gauche, son intérêt réside dans le fait que les deux admissions ne sont pas mises en séquence, mais composées par l’opérateur $(_ \oplus _)$. On interprète ce type de déduction avec le concept de concurrence. On dira, par exemple, que l’apparition des deux paires a été concurrente. Notons au passage qu’il peut en entrer un nombre arbitraire “en même temps”;

FIG. 1.11 – Deux philosophes rentrent



- les figures 1.12 et 1.13 détaillent la fusion des deux tables composées chacune d’un philosophe “tenant sa baguette à deux mains”. La figure 1.12 est une congruence, elle définit le contexte de l’application de la règle **change** donnée dans la figure 1.13

FIG. 1.12 – Deux philosophes forment une table

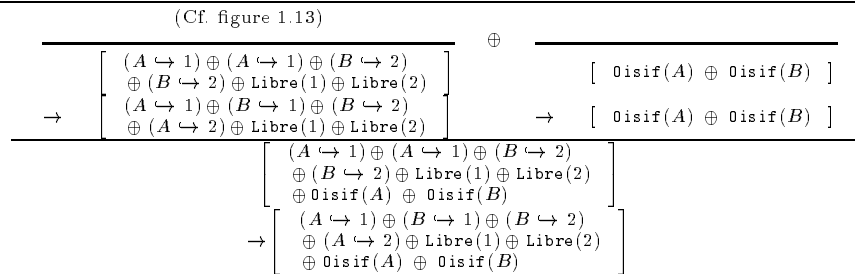


FIG. 1.13 – Une “fusion de table”

$$\begin{array}{c}
\frac{
\begin{array}{c}
\frac{[A]}{\rightarrow[A]} \quad \frac{[A]}{\rightarrow[A]} \quad \frac{[B]}{\rightarrow[B]} \quad \frac{[B]}{\rightarrow[B]} \quad \frac{[1]}{\rightarrow[1]} \quad \frac{[2]}{\rightarrow[2]} \\
\frac{
\begin{array}{c}
(A \hookrightarrow 1) \oplus (A \hookrightarrow 1) \oplus (B \hookrightarrow 2) \\
\oplus (B \hookrightarrow 2) \oplus \text{Libre}(1) \oplus \text{Libre}(2)
\end{array}
}{
\begin{array}{c}
(A \hookrightarrow 1) \oplus (B \hookrightarrow 1) \oplus (B \hookrightarrow 2) \\
\oplus (A \hookrightarrow 2) \oplus \text{Libre}(1) \oplus \text{Libre}(2)
\end{array}
}
\end{array}
}{\rightarrow}
\end{array}$$

- enfin, la dernière règle (Figure 1.14) est une application de la règle classique de prise de deux baguettes par un philosophe.

FIG. 1.14 – Un philosophe prends deux baguettes

$$\begin{array}{c}
\frac{
\begin{array}{c}
\frac{[0\text{isif}(B) \oplus (B \hookrightarrow 1) \oplus (B \hookrightarrow 2)]}{\rightarrow [0\text{isif}(B) \oplus (B \hookrightarrow 1) \oplus (B \hookrightarrow 2)]} \quad \oplus \quad \frac{
\begin{array}{c}
\frac{[A]}{\rightarrow[A]} \quad \frac{[1]}{\rightarrow[1]} \quad \frac{[2]}{\rightarrow[2]} \\
\frac{
\begin{array}{c}
(A \hookrightarrow 1) \oplus (A \hookrightarrow 2) \oplus 0\text{isif}(A) \\
\oplus \text{Libre}(1) \oplus \text{Libre}(2)
\end{array}
}{\text{Encas}(A, 1, 2)}
\end{array}
}{
\begin{array}{c}
0\text{isif}(B) \oplus (B \hookrightarrow 1) \oplus (B \hookrightarrow 2) \\
\oplus (A \hookrightarrow 1) \oplus (A \hookrightarrow 2) \oplus 0\text{isif}(A) \\
\oplus \text{Libre}(1) \oplus \text{Libre}(2)
\end{array}
}
\end{array}
}{\rightarrow}
\end{array}$$

En général pour les systèmes de transitions, appeler ce genre de construction une preuve est déplacé, mais en l’occurrence, nous avons par ce séquent (“éclaté” sur plusieurs figures) prouvé un fait essentiel à notre problématique:

“le système peut arriver, à partir de **rien**, dans un état où un philosophe mange.”

Ce qui implique que la Logique de Réécriture a bien été capable de modéliser l’arrivée dans le système d’un composant. Ceci est possible parce que la sémantique est très précise et donc on peut la pousser hors des limites de son utilisation classique sans pour autant la remettre en cause. Cette limitation non écrite que nous avons “violée” est contenue dans la règle “**enter**” qui a une variable libre à droite. Comme pour la réécriture seule (Cf. 1.1), cette contrainte n’est pas une contrainte propre au formalisme mais à ce que l’on en fait. Il est sûr que pour modéliser un *calcul* (terminant et confluent), il vaut mieux ne pas faire intervenir “l’inconnu” que représente la variable libre à droite. Mais pour un *protocole*, et en particulier un protocole d’admission dans un système, cette possibilité de faire intervenir l’inconnu est indispensable. Notre contribution est donc de montrer que lever cette contrainte de l’absence de variables libres à droite correspond à des applications tangibles dans le domaine des applications coopératives et plus généralement dès que se pose le problème de l’admission d’éléments inconnus.

1.3.4 Termes de preuve

Comment écrire une preuve d’accessibilité qui condense l’information de la transformation? Cette écriture existe, mais avant de la décrire nous allons conditionner sa validité à une condition que doit vérifier l’ensemble des règles [CDE⁺99]:

Définition 1.18 (Théorie univoque) Une *théorie de réécriture* est dite univoque

ssi il existe une bijection ν de \mathcal{L} dans \mathcal{R} vérifiant

$$\forall l \in \mathcal{L}, \exists x \in (\mathcal{T}_{\mathcal{E}}^{\Sigma}(X))^2, \nu(l) = (l, x)$$

Dans ce contexte, on note $\nu l = (l, (\bullet \nu(l), \nu \bullet(l)))$.

On peut toujours rendre une théorie univoque, ce n'est donc pas une restriction de considérer par défaut les théories comme univoques. Cela permet de considérer une théorie de réécriture comme une signature (avec équations) et une famille de paires de classes de termes. C'est cette forme que nous considérerons dans le chapitre III.

Etiquetage univoque de la relation d'accessibilité L'idée fondamentale consiste à étiqueter la relation d'accessibilité (i.e. la dynamique) par le séquent permettant de la prouver [BC88]. La conséquence primitive est que l'on va disposer d'un étiquetage *univoque* de la relation d'accessibilité. Il est essentiel de remarquer dans la figure 1.16 que le terme de preuve contient toute l'information du séquent et donc en particulier sa source et sa destination. En anticipant sur le vocabulaire introduit plus tard (cf. 1.4), la sémantique est un graphe dirigé: un ensemble d'états ($\mathcal{T}_{\mathcal{E}}^{\Sigma}(X)$), un ensemble d'arêtes (les termes de preuve), et deux fonctions **src** et **tgt**. Les déductions suivantes peuvent d'ailleurs parfaitement s'écrire comme des équations inductives sur la structure des termes de preuve calculant deux fonctions **src** et **tgt**. Notons aussi le changement de vocabulaire: Réflexivité et Transitivité (des termes de relations) sont remplacés par Identité et Composition. Ces deux opérations (**Id** $\bar{\cdot}$, ($\bar{\cdot}$; $\bar{\cdot}$)) sont les opérations ajoutées [Cor98] à un graphe dirigé pour qu'il forme une catégorie (Cf Chapitre II).

FIG. 1.15 – Terme de preuve équivalent aux figures 1.9 à 1.14

$$\begin{aligned} & \text{enter}(\text{Id}^{[A]}, \text{Id}^{[1]}) \oplus \text{enter}(\text{Id}^{[B]}, \text{Id}^{[2]}); \\ & \text{change}(\text{Id}^{[A]}, \text{Id}^{[A]}, \text{Id}^{[B]}, \text{Id}^{[B]}, \text{Id}^{[1]}, \text{Id}^{[2]}) \oplus \text{Id}^{[\text{0isif}(A) \oplus \text{0isif}(B)]}; \\ & \text{Id}^{[\text{0isif}(B) \oplus (B \hookrightarrow 1) \oplus (B \hookrightarrow 2)]} \oplus \text{get}(\text{Id}^{[A]}, \text{Id}^{[1]}, \text{Id}^{[2]}) \end{aligned}$$

FIG. 1.16 – Etiquetage univoque des éléments de la relation d'accessibilité

1. **Identité**: Pour chaque $[t] \in \mathcal{T}_{\mathcal{E}}^{\Sigma}(X)$

$$\frac{}{(\text{Id}^{[t]} : [t] \rightarrow [t])}$$

2. **Congruence**: Pour chaque f , symbole fonctionnel d'arité n :

$$\frac{(\mathbf{a}_1 : [t_1] \rightarrow [t'_1]) \dots (\mathbf{a}_n : [t_n] \rightarrow [t'_n])}{(\mathbf{f}(\mathbf{a}_1, \dots, \mathbf{a}_n) : [f(t_1, \dots, t_n)] \rightarrow [f(t'_1, \dots, t'_n)])}$$

3. **Remplacement**: Pour chaque règle $(\mathbf{r} : [t(x_1, \dots, x_n)] \rightarrow [t'(x_1, \dots, x_n)])$:

$$\frac{(\mathbf{a}_1 : [w_1] \rightarrow [w'_1]) \dots (\mathbf{a}_n : [w_n] \rightarrow [w'_n])}{(\mathbf{r}(\mathbf{a}_1, \dots, \mathbf{a}_n) : [t(w_1/x_1, \dots, w_n/x_n)] \rightarrow [t'(w'_1/x_1, \dots, w'_n/x_n)])}$$

4. **Composition**:

$$\frac{(\mathbf{a} : [t_1] \rightarrow [t_2]) (\mathbf{b} : [t_2] \rightarrow [t_3])}{(\mathbf{a}; \mathbf{b} : [t_1] \rightarrow [t_3])}$$

Les séquents donnés dans les Figures 1.9 à 1.14 sont donnés par le terme de preuve donné Figure 1.15. Notons qu'il apparaît ici syntaxiquement que la concurrence de l'apparition des philosophes s'exprime comme la congruence de deux transformations d'apparition.

Quotientage de l'étiquetage univoque A ce stade, des collections de termes de preuve décrivent la même accessibilité. Ce problème va être résolu en quotientant l'ensemble des termes de preuve par l'ensemble des équations suivantes:

Catégorie Les deux équations suivantes permettent de doter la structure de graphe de deux opérations, identité et composition, qui satisfont les conditions pour que cette structure forme une catégorie. La notion de catégorie est introduite au début du chapitre 2.

- (a) Associativité de la composition: Pour tout terme de preuve α, β, γ , $(\alpha; \beta); \gamma \approx \alpha; (\beta; \gamma)$.
- (b) Identité: Pour tout $(\mathbf{f} : \alpha \rightarrow \beta)$, $f; \text{Id}^\beta \approx f$ et $\text{Id}^\alpha; g = g$.

Fonctorialité de la structure Σ -algébrique L'idée est que l'utilisation d'un opérateur n -aire préserve l'identité et la composition, donc la structure de catégorie. Intuitivement, l'ajout d'un contexte définit un "morphisme de comportement". Ceci est une expression abstraite et concurrente de la notion de monotonie dans les réseaux de Petri, ou de plongement (ang. *embedding*) dans les systèmes de réécriture.

Soit f un opérateur n -aire.

- (a) Préservation de la composition: Pour tout tuple de termes de preuve $\alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_n$

$$f(\alpha_1; \beta_1, \dots, \alpha_n; \beta_n) \approx f(\alpha_1, \dots, \alpha_n); f(\beta_1, \dots, \beta_n)$$

- (b) Préservation de l'identité: $f(\text{Id}^{[t_1]}, \dots, \text{Id}^{[t_n]}) \approx \text{Id}^{[f(t_1, \dots, t_n)]}$

Axiomes de \mathcal{E} Ici on importe les axiomes définis sur les opérateurs de la signature réutilisés dans les congruences de transformation. Pour tout axiome $t(x_1, \dots, x_n) \approx t'(x_1, \dots, x_n)$ de \mathcal{E} , pour tout tuple de termes de preuve $\alpha_1, \dots, \alpha_n : t(\alpha_1, \dots, \alpha_n) \approx t'(\alpha_1, \dots, \alpha_n)$.

Echange Cette équation stipule que l'on peut casser une réécriture en une séquence de deux, et ceci de deux façons possibles: d'un coté l'application elle-même (au "top-level"), de l'autre les réécritures qui ont lieu "dans" les sous-termes. Pour chaque règle $(\mathbf{r} : [t(x_1, \dots, x_n)] \rightarrow [t'(x_1, \dots, x_n)])$:

$$\begin{array}{c} (\alpha_1 : [w_1] \rightarrow [w'_1]) \dots (\alpha_n : [w_n] \rightarrow [w'_n]) \\ \hline \begin{array}{ll} r(\alpha_1, \dots, \alpha_n) & \approx \quad r(\text{Id}^{[w_1]}, \dots, \text{Id}^{[w_n]}); \quad t'(\alpha_1, \dots, \alpha_n) \\ r(\alpha_1, \dots, \alpha_n) & \approx \quad t(\alpha_1, \dots, \alpha_n); \quad r(\text{Id}^{[w'_1]}, \dots, \text{Id}^{[w'_n]}) \end{array} \end{array}$$

La réécriture de termes est un cadre formel adéquat pour spécifier des systèmes dynamiques. Son expressivité est très importante, ses limites se situant bien au-delà des approches classiques étendant les réseaux de Petri. La logique de réécriture fournit une sémantique compacte pour son évaluation. En outre, cette sémantique est concurrente et permet d'appréhender sans ajout les problèmes relatifs au tir concurrent de transitions, qui est une interprétation de l'application d'un opérateur de la signature dans le terme de preuve. Cette logique nous a profondément influencés, et nous sommes convaincus qu'il s'agit d'une approche à considérer en priorité pour traiter de problèmes relatifs à l'extension des réseaux de Petri.

1.3.5 Problème relatif aux variables libre à droite

En réécriture de termes, l'identité d'un élément n'existe pas de façon explicite. Les constantes utilisées dans les termes sont des localités pouvant toujours être partagés par plusieurs sous-termes. Une caractéristique de ce fait est que l'on peut toujours “replier” un terme de preuve sur un ensemble de constantes ne contenant qu'une constante de chaque sorte. Nous allons illustrer cette possibilité de “repliage” sur les références par notre exemple des philosophes. Considérons le terme de preuve suivant:

$$\begin{aligned} & \text{enter}(\text{Id}^{[A]}, \text{Id}^{[1]}) \oplus \text{enter}(\text{Id}^{[A]}, \text{Id}^{[1]}); \\ & \text{change}(\text{Id}^{[A]}, \text{Id}^{[A]}, \text{Id}^{[A]}, \text{Id}^{[A]}, \text{Id}^{[1]}, \text{Id}^{[1]}) \oplus \text{Id}^{[\text{Oisif}(A) \oplus \text{Oisif}(A)]}; \\ & \text{Id}^{[\text{Oisif}(A) \oplus (A \hookrightarrow 1) \oplus (A \hookrightarrow 1)]} \oplus \text{get}(\text{Id}^{[A]}, \text{Id}^{[1]}, \text{Id}^{[1]}) \end{aligned}$$

Sa source est toujours **rien**; quant à sa cible elle est assez paradoxale:

$$\begin{aligned} & \text{Oisif}(A) \oplus (A \hookrightarrow 1) \oplus (A \hookrightarrow 1) \\ & \oplus \text{Encas}(A, 1, 1) \end{aligned}$$

Tout se passe comme si sur chaque “adresse” on avait deux “objets”: sur l’“adresse” A , deux “philosophes” dont l'un mange et l'autre pas; sur l'adresse 1 , deux “baguettes” utilisées par le “philosophe” qui mange. Ceci exclut bien que les variables utilisées comme références se comportent comme des identités. Un élément de comportement (**Libre**($-$) ou **Oisif**($-$)) permettrait d'identifier des composants mais ces éléments sont anonymes comme les jetons d'un Place/Transition.

En quelque sorte, il n'est pas défini de stratégie d'allocation de “nouvelles références” qui correspondent aux variables libres à droite. Néanmoins une stratégie minimale (“*anything*”) est définie qui permet d'associer n'importe quel terme (clos) à une variable libre à droite. Elle diffère d'une stratégie d'allocation de “nouvelles références” (“*new*”). La stratégie “*anything*” peut avoir un sens dans certains contextes comme notre cas d'étude (Cf. section 4.1). Pour le gala des philosophes, elle mène à la situation décrite ci-dessus qui n'est pas satisfaisante puisque tous les philosophes pourraient avoir la même localité.

Plus généralement, et comme nous l'avons souligné au début de la présentation du problème des philosophes, cette contrainte est due à l'utilisation de constantes comme des identités. Cet usage de “constantes individualisées” n'est pas un fait “du formalisme” mais “de la théorie de réécriture” (i.e. “du réseau”). Cette remarque motive notre présentation des grammaires de graphes qui proposent un mécanisme de gestion des références “dans le formalisme”.

1.4 Les grammaires de graphes

Les grammaires de graphes ont été développées à partir des années 60. La recherche à leur sujet était motivée par la reconnaissance de forme, la construction de compilateurs et la spécification par type abstrait [Roz97]. On peut considérer qu'à l'origine, les grammaires de graphes étaient perçues comme une généralisation des grammaires à base de chaînes de caractères [Cou95]. Pour ce type d'utilisations, le choix est de remplacer des nœuds [ER97] ou des arêtes [DHK97], mais ces deux approches sont des approches de type *calcul* et donc ne proposent pas de corps constitué permettant d'en faire des outils de spécification de systèmes dynamiques. En plus d'un remplacement d'un élément non-terminal, il faut pouvoir faire intervenir le contexte et réécrire des motifs. La notion de motif et de collage paramétré de motifs sont centrales dans la théorie développée par Ehrig [Ehr86] qui

stipule la transformation de graphes uniquement à l'aide de morphisme de graphes et d'une construction sur ces morphismes. C'est cette approche que nous présenterons extensivement, en faisant volontairement abstraction de ses énoncés catégoriques tant qu'ils n'auront pas de nécessité intrinsèque (Cf. chapitre 2).

Cette section est organisée comme suit: tout d'abord on définit la notion de graphe dirigé qui sera utilisée dans la suite de ce travail par opposition à d'autres définitions équivalentes (ou pas) qui peuvent être utilisées avec le même nom. Ensuite on introduit la construction fondamentale qui permet de réaliser les transformations de graphes que l'on utilise ici: le "pushout". Pour montrer que la théorie des catégories n'est pas essentielle à ce niveau nous avons volontairement choisi le nom de "somme amalgamée" [Ser80]. Ensuite on définit les différents types de déclinaisons de ce concept de base: d'une part, par les constructions mises en jeu (double ou simple somme), puis dans un deuxième temps pour permettre de "spécialiser" ou "typer" les nœuds du graphe.

1.4.1 Graphes et morphismes

La notion de graphe est tellement intuitive que ce mot nécessite une sorte de décrassage pour que l'on spécifie clairement ce que l'on entend par graphe. Si l'on suit une définition d'un graphe assez couramment acceptée [Ber77] elle dit:

Un graphe $G = (X, U)$ est le couple constitué:

1. *par un ensemble $X = \{x_1, x_2, \dots, x_n\}$*
2. *par une famille $U = (u_1, u_2, \dots, u_n)$ d'éléments du produit cartésien $X \times X = \{(x, y) \mid x \in X, y \in X\}$;*

un élément (x, y) de $X \times X$ peut apparaître plusieurs fois dans la famille U .

Nous utilisons une lettre minuscule pour dénoter "un" graphe parce que c'est "un" élément de l'ensemble des graphes, par contre ses deux premières composantes sont des ensembles donc la lettre directrice est en majuscule: " V " pour *Vertice* et " E " pour *Edges*, mots anglais signifiant Sommets et Arêtes: $g = (V_g, \mathcal{E}_g)$. Il est déroutant de voir un composé de majuscules écrit en minuscules mais cela permet de maintenir notre notation pour les éléments, les ensembles et les fonctions.

Cette définition est ambiguë, car la définition d'une famille n'est pas fournie explicitement. Nous poserons une définition pour le terme de famille sur un ensemble X : une famille \mathcal{A} est une paire $\langle A; \mathbf{a}(_) \rangle$ composée d'un ensemble A et d'une fonction $\mathbf{a}(_)$ associant à chaque élément de l'ensemble A un élément de X . Si on prend cette définition pour une famille, alors la définition de [Ber77], un graphe g est la donnée d'un ensemble de sommets V_g et d'une famille \mathcal{E}_g de paires (source, cible), soit [Ser80]:

$$\langle V_g; \langle E_g; (\mathbf{y}_g(_) : E_g \rightarrow V_g \times V_g) \rangle \rangle$$

Il ressort de cette définition que les sommets et les arêtes sont identifiés. Par contre si on considère que les arêtes sont *interchangeables*, donc que l'identité $u_1 \dots u_n$ des flèches n'est pas une donnée "pertinente" mais que seul le *nombre* de flèches a ayant pour localité par $\mathbf{y}_g(_)$ la paire (x, y) importe, ceci s'écrit:

$$\langle V_g; (\mathbf{u}_g(_) : V_g \times V_g \rightarrow \mathbb{N}) \rangle$$

Cette dernière forme nous permet de tracer une différence claire entre un graphe et une relation. Une relation sur un ensemble A est une partie du produit cartésien

de A avec lui-même, c'est à dire en se souvenant que $D \subset A \times A$ est équivalent à une fonction $(\mathbf{D}(_) : A \times A \rightarrow [0, 1])$:

$$\langle V_g; (\mathbf{u}_g(_) : V_g \times V_g \rightarrow [0, 1]) \rangle$$

D'une façon plus intuitive, dans une *relation* il ne peut y avoir qu'une seule arête au maximum entre deux sommets, ce qui n'est pas le cas dans un *graphe* où ce nombre est non borné. Ce point était d'ailleurs souligné dans la définition donnée comme référence. Notons au passage que dans l'expression "graphe des états accessibles", il s'agit d'une relation étiquetée. Un graphe peut être comparé à une relation étiquetée dont l'étiquetage est univoque : chaque étiquette n'est associée qu'à une arête.

La forme standard des graphes dirigés clôt notre tour d'horizon des différentes définitions données au mot "graphe". Un graphe dirigé est la donnée de deux ensembles (V et E) et de deux fonctions ($\mathbf{src}(_)$ et $\mathbf{tgt}(_)$) associant à chaque arête sa source et sa cible.

Définition 1.19 (Graphe dirigé [Ehr86, Mac71]) *Un graphe dirigé*

$$g = \langle V_g; E_g; \mathbf{src}_g(_); \mathbf{tgt}_g(_) \rangle$$

est formé d'un ensemble de sommets V_g et un ensemble d'arêtes E_g et de deux fonctions $(\mathbf{src}_g(_) : E_g \rightarrow V_g)$ et $(\mathbf{tgt}_g(_) : E_g \rightarrow V_g)$.

Comment représenter le "bal des philosophes" (Cf. 1.2 et 1.3) par des graphes dirigés? On ne dispose pas de la possibilité de définir des sortes comme pour la logique de réécriture. Pour différencier les baguettes de philosophes nous définissons un critère "graphique" (ou topologique). Nous exprimons cette distinction en terme de degré sortant, i.e. le nombre d'arêtes dont la source est le nœud considéré :

- une baguette est de degré sortant 1;
- un philosophe est de degré sortant 2.

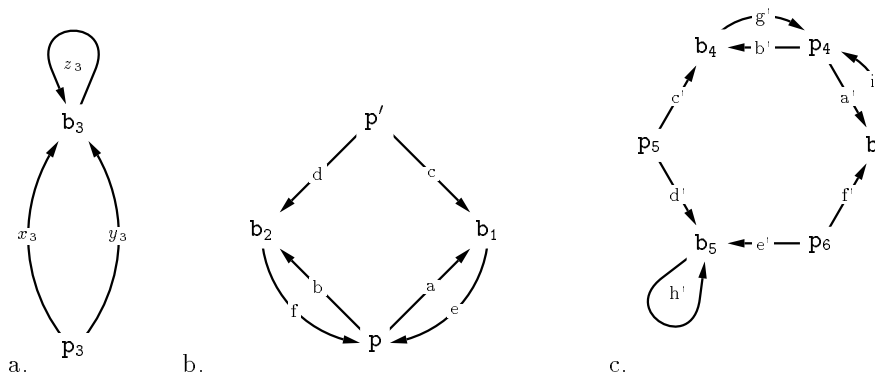
On pose ensuite qu'une arête ayant pour source un philosophe a pour cible une baguette. Enfin, l'évolution comportementale du système sera uniquement décrite par la cible de l'unique arête sortant d'une baguette. Si cette arête a pour cible un philosophe elle est "utilisée" par ce philosophe, si cette arête a pour cible elle-même la baguette est libre. Muni de notre "politique de dessin" nous permettant de représenter l'état d'un "gala de philosophe", nous allons donner quatre exemples de graphe représentant un tel système, trois en extension et un en intention.

Graphes finis en extension La figure 1.17 détaille trois graphes décrivant trois états du système impliquant respectivement une, deux et trois paires philosophes/baguette. Notez que la juxtaposition des trois graphes décrirait encore un état valide du système bien que non "connexe".

Graphes infinis en intention On peut décrire, en intention, un ruban de philosophe/baguette infini, en utilisant la décomposition unique de tout entier sous forme $3q + r$ où $0 \leq r < 3$:

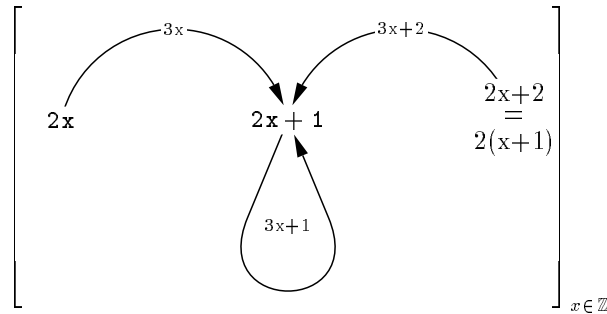
$$\langle \mathbb{Z}; \mathbb{Z}; (\text{fn } 3q + r \Rightarrow 2q + r); (\text{fn } 3q + r \Rightarrow 2q + 1) \rangle$$

FIG. 1.17 – Exemples de graphes décrivant un état du problème des philosophes



- a. Le philosophe p_3 tient “deux fois” la même baguette b_3 .
- b. p et p' , deux philosophes dont un, p , mange en utilisant les baguettes b_1 et b_2 .
- c. p_4 , p_5 et p_6 , trois philosophes, parmi eux seul p_4 mange en utilisant les baguettes b_4 et b_6 .
- abc Trois tables de philosophes, parmi ceux-ci p et p_4 mangent, utilisant les baguettes b_1 , b_2 , b_4 et b_6 .

Cette construction peut être représentée graphiquement de la façon suivante:



Du côté de la logique de réécriture, il faut envisager des termes infinis et ceci suppose une complétion, construction mathématique plus complexe que la donnée par intention.

Morphisme de graphes Quelle que soit la définition choisie, on peut la munir d’une notion de morphisme. Informellement, l’existence d’un morphisme de a vers b signifie que a est présent dans b éventuellement “replié”. Le morphisme est “co-substanciel” à la structure : dès que l’on peut définir une structure, on peut définir des morphismes entre deux instances de ces structures.

Un morphisme ($\mathbf{f} : g \rightarrow h$) de graphe est une “fonction” qui injecte une instance source $g = \mathbf{src}(f)$ dans une instance cible $h = \mathbf{tgt}(f)$ et préserve la structure considérée. Quels que soient cette structure et ses morphismes adéquats, l’ensemble des structures et l’ensemble des morphismes sur ces structures sont munis des opérations suivantes:

- une loi de composition associative ($_ \circ _$).

- une identité : A tout graphe g , on peut associer un morphisme Id^g ayant pour source et cible g tel que pour tout morphisme f tel que $\text{tgt}(f) = g$, on ait $\text{id}(g) \circ f = f$ et inversement pour tout morphisme f tel que $\text{src}(f) = g$, on ait $f \circ \text{id}(g) = f$.

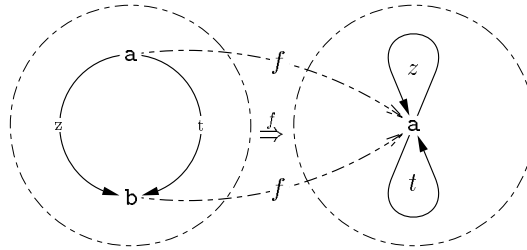
Définition 1.20 (Morphisme de graphes dirigés) *Un morphisme de graphe $(f : a \rightarrow b)$ est une paire de fonctions $((f_V : G_V \rightarrow H_V), (f_E : G_E \rightarrow H_E))$ vérifiant $\text{src}_h(-) \circ f_E = f_V \circ \text{src}_g(-)$ et $\text{tgt}_h(-) \circ f_E = f_V \circ \text{tgt}_g(-)$. La composition est donnée par la composition de fonctions composantes par composantes, et l'identité étant réalisée par une paire d'identités.*

Lorsque l'on considère des ensembles, on peut représenter les fonctions par des diagrammes de Venn. Mais pour représenter un morphisme de graphes, on ne dispose plus de représentation graphique satisfaisante. Nous proposons la politique de dessin suivante pour les morphismes :

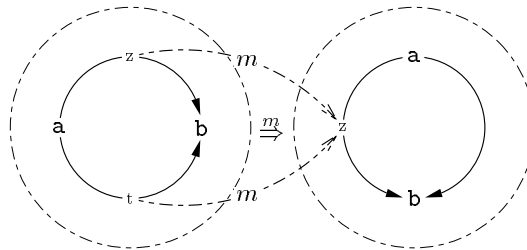
- nous indiquerons uniquement le nom des éléments de la source ainsi que le nom de leur image;
- les éléments appartenant à la cible n'étant pas image d'un élément de la source ne sont pas nommés;
- lorsque la fonction préserve les noms des éléments, elle n'est pas représentée par une flèche : l'identité des noms sera considérée comme une information suffisante.

Cette convention n'est pas univoque dès lors que le graphe n'est pas planaire. La figure 1.18 détaille deux morphismes de graphes relativement simples qui ne seraient pas planaires sans la convention. Ils ne sont pas pour autant injectifs contrairement aux approches uniquement à base de nommage qui le sont souvent par défaut.

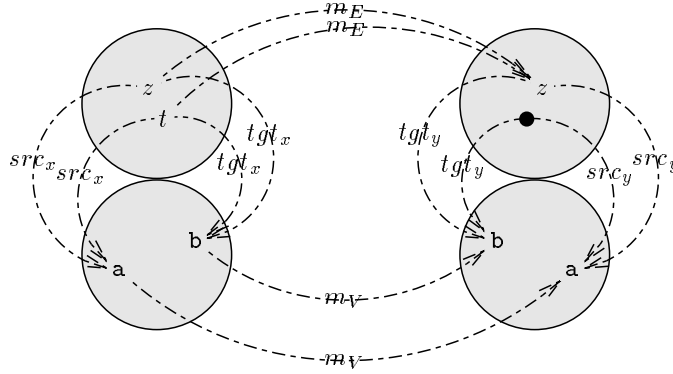
FIG. 1.18 – Deux morphismes de graphes avec les conventions



1. Fusion des noeuds a et b .



2. Fusion des arêtes z et t . Une arête hors de l'image (morphisme non surjectif)

FIG. 1.19 – Diagramme de Venn du morphisme $(m : x \rightarrow y)$ (Cf. Fig 1.18.2)

Néanmoins, on peut tout représenter (y compris $\text{src}(-)$ et $\text{tgt}(-)$) avec des diagrammes de Venn. Cela a un mérite essentiel qui nous fera utiliser cette notation assez fréquemment : c'est la notation la plus proche de la structure mathématique. En contrepartie, on perd tout l'avantage de la notation graphique : la dispersion dans l'espace correspond à un critère d'éloignement dans le graphe. En conséquence nous utiliserons cette notation pour des exemples illustratifs simples.

A titre d'exemple on a représenté le premier morphisme avec des diagrammes de Venn dans la figure 1.19. Dans cette figure, chaque graphe est représenté par deux “bulles” représentant l'ensemble des arêtes (dessus) et l'ensemble des nœuds (dessous). Ces deux ensembles sont reliés chacun par deux fonctions ($\text{src}(-)$ et $\text{tgt}(-)$). Le morphisme est représenté par ses deux fonctions m_E et m_V . Le fait que ces deux fonctions forment un morphisme s'exprime graphiquement en disant que “tous les carrés possibles sont fermés”.

Notez que nous avons utilisé la même convention qui consiste à “oublier” l'identité du nœud qui ne fait pas partie de l'image de la fonction m_E , nous l'avons donc remplacé par un symbole anonyme (\bullet). Dans sa version “graphique” nous avions “oublié” de l'étiqueter.

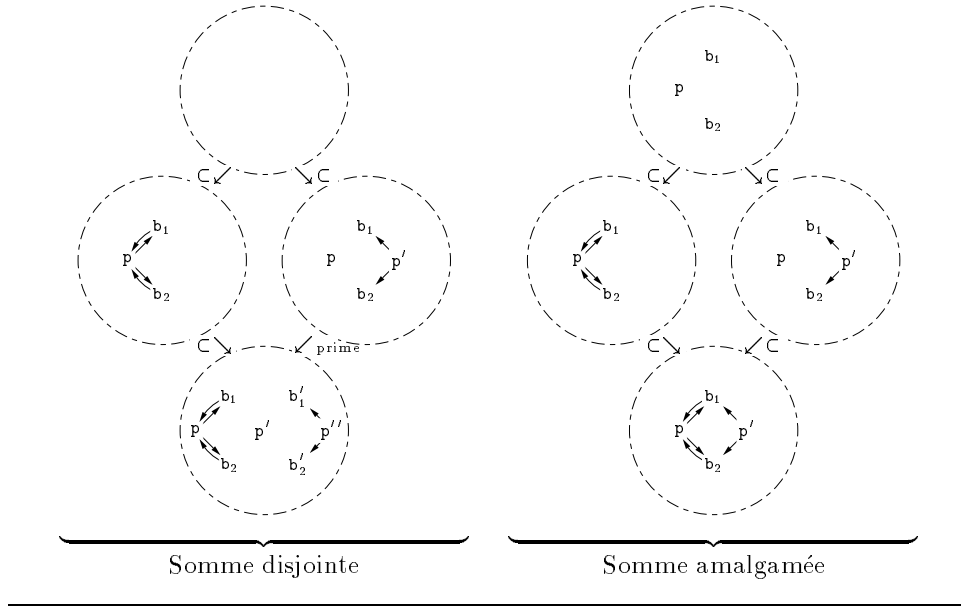
Ceci illustre que ces morphismes sont issus de fonctions sous-jacentes et ont la caractéristique fondamentale d'une fonction : *par une fonction, tout élément de l'ensemble source a une et une seule image*. La réciproque étant fautive : *par une fonction, un élément de l'ensemble cible est image d'un nombre quelconque d'éléments*. Notre exemple illustre qu'un élément de la cible peut, soit n'être image d'aucun élément de la source, soit être image de plusieurs éléments de la source.

1.4.2 Somme amalgamée

La construction ci-dessous, la *somme amalgamée*, permet de transformer toute structure pour laquelle on dispose de morphismes. Il est très important de noter qu'une somme amalgamée existe indépendamment de la façon dont on la construit. On sait caractériser les couples structure/morphisme pour lesquels cette somme existe (Cf 2.1).

Définition 1.21 (Somme amalgamée) *La somme amalgamée d'une paire de morphismes $\langle (f : a \rightarrow b); (g : a \rightarrow c) \rangle$ ayant même source est une paire de morphismes $\langle (g_f : b \rightarrow d); (f_g : c \rightarrow d) \rangle$ ayant même cible vérifiant $g_f \circ f = f_g \circ g$ telle que pour toute paire de morphismes $\langle (g' : b \rightarrow e); (f' : c \rightarrow e) \rangle$ ayant même cible et vérifiant $g' \circ f = f' \circ g$ il existe un unique morphisme $(u : d \rightarrow e)$ tel que $u \circ g_f = g'$ et $u \circ f_g = f'$.*

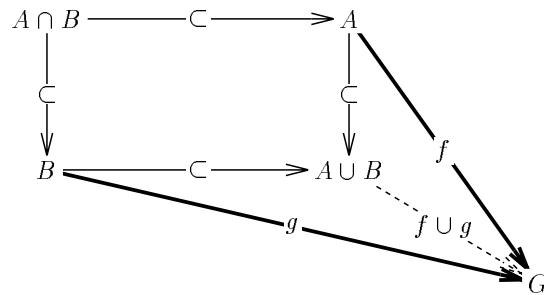
FIG. 1.20 – Somme disjointe et amalgamée



Lorsque l'interface est vide, la somme amalgamée produit deux copies distinctes des graphes cibles. L'éventuel renommage nécessaire pour éviter le conflit de nom est contenu dans les morphismes.

Le calcul de la somme amalgamée de graphes s'obtient en deux phases. On réalise d'abord deux copies disjointes des deux graphes que l'on veut sommer, puis on "amalgame" (i.e. "fusionne") les éléments qui sont des images du même élément de l'interface.

Lien avec l'union ensembliste On peut illustrer cette interprétation en considérant une somme amalgamée dans les ensembles pour laquelle on spécifie l'intersection ensembliste comme interface. Dans ce cas, la somme amalgamée est l'union ensembliste. Les différentes fonctions (sauf f et g qui sont quelconques) sont des injections qui conservent les noms (appelées inclusions).



La propriété universelle de cette somme particulière s'obtient en considérant l'union des fonctions $f \cup g$, valide puisque f et g coïncident sur l'intersection (Cf 1.1), conséquence de la commutation. On peut aussi construire $f \cup g(x)$ par l'expression : si $x \in A \setminus B$ alors prendre $f(x)$ sinon prendre $g(x)$.

Cette construction fonctionne avec l'union sur les graphes qui est une opération partielle assujettie au fait que l'union des fonctions $\text{src}(_)$ et $\text{tgt}(_)$ soient bien des fonctions. Pour cela, elles doivent coïncider sur leur intersection : deux arêtes ayant même nom doivent avoir même source et même cible. L'utilité de cette construction est discutable car c'est un cas très particulier de somme amalgamée, néanmoins une grande partie des sommes réalisées dans le cadre de la théorie classique relève de cette construction.

1.4.3 Système et dérivation

Transformer les graphes par une somme amalgamée peut ajouter des entités (sommets, arêtes) ou en fusionner. On ne peut pas, avec la somme amalgamée définie, enlever. Deux approches ont été proposées pour dépasser cette limitation structurelle, nous les donnons ici par ordre chronologique d'apparition:

Double somme Cette approche [Ehr86] considère une règle comme une paire de morphismes ayant même source, ce sera l'approche suivie dans ce mémoire. On applique cette règle sur un contexte en réalisant deux sommes, une à gauche et une à droite. Une sémantique complète sera donnée au début de la partie 2 dans un cadre structure/morphisme étendu. Cette approche est fortement symétrique, le graphe source est somme amalgamée comme le graphe cible. La section 1.4.4 traitera du problème opérationnel qui consiste à calculer le contexte à partir du graphe source (le *complément*[CMR⁺97]). Un intérêt de cette approche est d'être *réversible* : toute règle dispose d'une règle symétrique. Dans l'exemple donné Figure 1.21, *get* et *release* sont symétriques.

Simple somme Cette approche [Löw93] considère une règle comme un morphisme *partiel*, c'est à dire dans notre exemple comme une paire de fonctions partielles satisfaisant la propriété de morphisme. La dérivation se fait par une somme amalgamée avec la paire graphes/morphismes-partiels. L'élégance de la théorie réside pour partie dans sa conservation de l'analogie transformation/morphisme/flèche. Le principal défaut de cette théorie est sa politique de gestion des références, on peut toujours appliquer une règle (l'approche est dite *complète*), mais toutes les arêtes reliées à des sommets détruits sont aussi détruites. Cette politique, *automatic deletion*, ne correspond à aucune réalité dans notre cadre d'étude. Par contre, le travail de [Löw93] contenait une vraie pépite: la caractérisation des structures/morphismes où la somme amalgamée avec les morphismes partiels existe. Cette caractérisation n'a aucun lien direct avec la politique d'*automatic deletion*, nous l'utiliserons comme première extension pour le typage des nœuds dans la suite.

Nous utiliserons l'*approche par double somme*. Par rapport aux définitions originales[CMR⁺97], nous avons juste changé les noms des graphes impliqués pour les mettre en conformité avec nos notations. Dans cette approche, un système de règles est une famille de paires de morphismes de graphes ayant même source.

Définition 1.22 (Système) *Un système est une paire $\langle L; \mathbf{1}(_) \rangle$ formée*

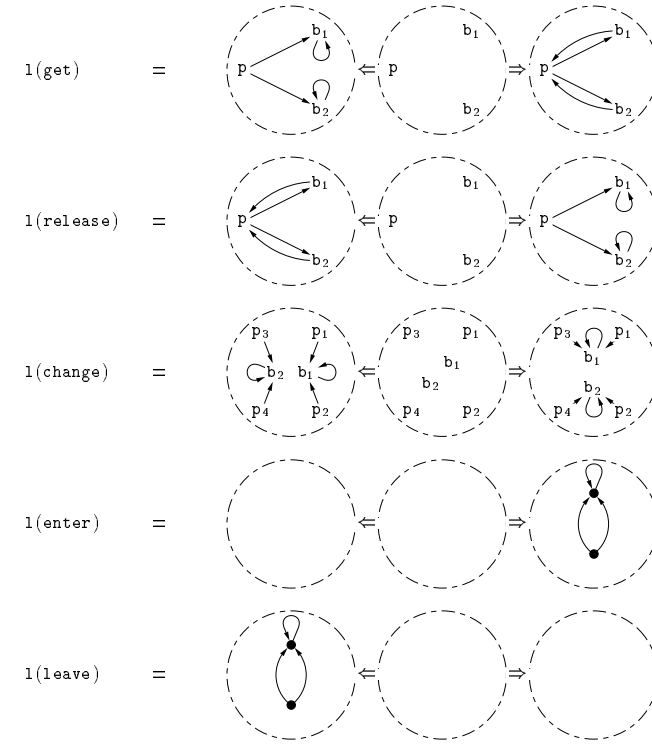
- d'un ensemble L de noms de règles
- d'une fonction $\mathbf{1}(_)$ qui, à chaque nom, associe une paire de morphismes de graphes ayant même source.

$$\mathbf{1}(l) = \langle (\prec \mathbf{1} : \bar{l} \rightarrow \bullet l); (\succ \mathbf{1} : \bar{l} \rightarrow l^*) \rangle$$

Nous pourrions considérer l'ensemble de fonctions $\bullet, \sqsubseteq, \sqsupset, \sqsubset, \sqsupseteq$, et $<, >$ décrivant les différentes composantes de $l(\sqsubseteq)$ et ainsi les adresser séparément. Et on appelle: $\bullet l$ le pré-graphe de l , \bar{l} l'interface de l , l^\bullet le post-graphe de l , $<l$ le morphisme gauche de l , $l>$ le morphisme droit de l .

La figure 1.21 détaille le système de règles pour le “gala des philosophes” en double somme. Les trois premières règles (**get**, **release**, **change**) forment le “bal des philosophes” aux quelles on ajoute les deux règles **enter** et **leave** qui permettent de gérer les admissions et départs de paires philosophe/baguette.

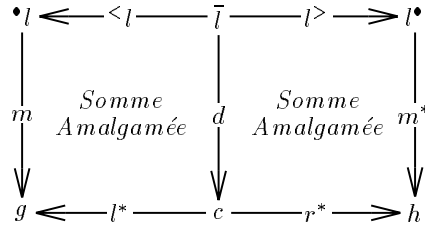
FIG. 1.21 – Système de règles pour le “gala des philosophes”



La sémantique des grammaires de graphes, dans sa forme originale [Ehr86] est très proche de celle des réseaux de Petri. Elle définit une condition de sensibilisation puis un tir. Nous la donnerons dans un deuxième temps car la référence à la somme amalgamée y est diffuse. Nous donnons la définition suivante:

Définition 1.23 (Dérivation directe[CMR⁺97]) Soit g un graphe, $l \in L$ une règle et $(m : \bullet l \rightarrow g)$ un morphisme de graphe, appelé morphisme d'application. Une dérivation directe depuis g vers h utilisant p (basée sur m) existe si et seulement si le diagramme ci-dessous peut être construit, où les deux carrés sont des sommes amalgamées. Dans ce cas c est appelée le contexte, et nous écrirons $g \xrightarrow{p, m} h$, et aussi $g \xrightarrow{p} h$; Nous devrions (sic) écrire $g \xrightarrow{p, m, d, m^*, l^*, r^*} h$

indiquant explicitement tous les morphismes de la double somme.



Soulignons ici que si nous construisons une *relation sur les états*, l'état du système est représenté par un *graphe*.

Une propriété centrale de l'approche par double-somme est qu'elle est réversible, si une règle peut s'appliquer sur un graphe g , alors on peut construire une application de la règle symétrique sur le graphe obtenu qui ramène à g .

Propriété 1.4 (Réversibilité) Si deux règles a et b sont symétriques (i.e. vérifiant $\langle a = b \rangle$ et $\langle b = a \rangle$) alors

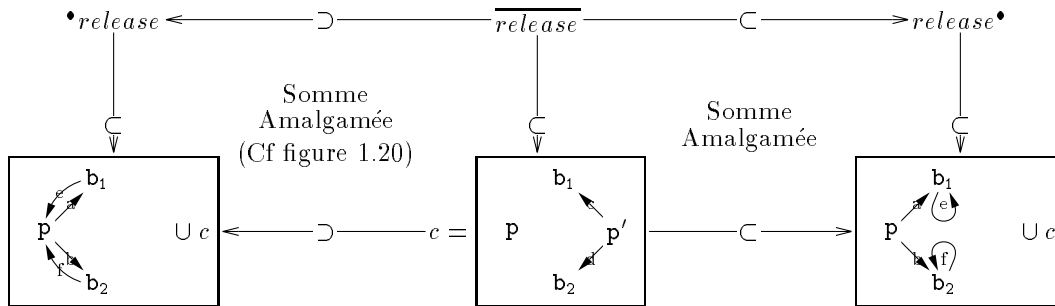
$$(g \xrightarrow{a, m, d, m^*, l^*, r^*} h) \implies (h \xrightarrow{b, m^*, d, m, r^*, l^*} g)$$

Ceci découle de la symétrie du diagramme de la double-somme selon l'axe défini par la flèche d .

Séquence de dérivation

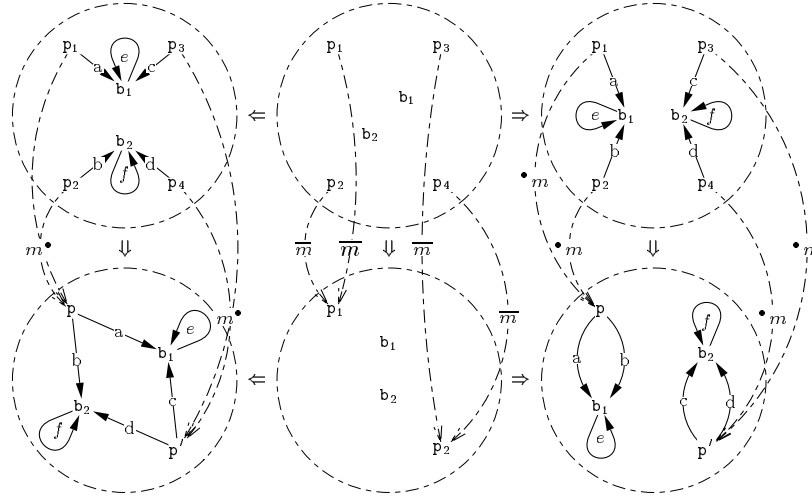
Nous allons détailler une séquence de dérivation. Elle mène d'un état où un philosophe mange (graphe b. de la figure 1.17) au graphe vide. La première dérivation utilise l'union ensembliste comme moyen pour calculer la somme amalgamée, et la seconde sera une somme qui ne peut être décrite par ce moyen car faisant intervenir une "fusion" de nœuds.

- *Relâcher des baguettes* On posant c comme graphe de contexte, on obtient avec les unions la dérivation suivante. On vérifie que l'interface est bien l'intersection du pré-graphe (resp. du post-graphe) et du graphe c .



- *Séparation des deux philosophes* La figure 1.22 détaille la dérivation directe qui montre comment deux philosophes seuls peuvent "casser" une table en deux. Ce séquent contient volontairement une difficulté : le morphisme sur les nœuds n'est pas injectif. Pour aider à lire ce dessin on peut remarquer que

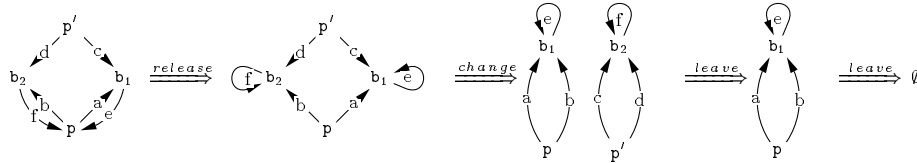
FIG. 1.22 – Exemple de dérivation directe



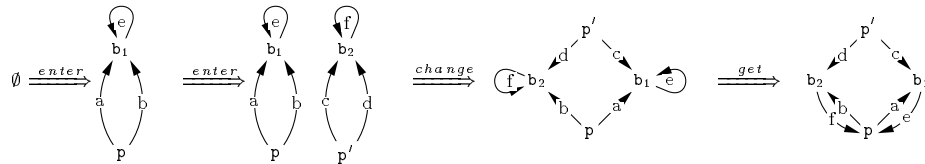
l'ensemble des noeuds est identique sur une ligne et que les morphismes sont tous identiques pour les noeuds (ce qui n'a pas été possible de conserver dans la représentation).

- *Départ des philosophes* Ce départ, en deux étapes, ne met en jeu que des sommes disjointes.

Toutes les dérivation peuvent être regroupées dans la séquence de dérivation ci-dessous:



Comme chaque dérivation est réversible on peut en déduire la séquence de dérivation ci dessous, car les règles ont leur règle symétrique dans le système de règle (pour **change** sa symétrique est elle-même à un renommage près).



Non-unicité du résultat

Tout état isomorphe à un état accessible est accessible. Cette considération découle de la construction de la somme amalgamée pour laquelle tout graphe isomorphe à une solution convient. En conséquence même pour des systèmes triviaux (e.g. conservation d'un unique élément) le nombre de dérivation à partir d'un état donné est non borné. On retrouve ici, sous une autre forme, le branchement infini d'une dynamique où un élément apparaît. On peut rendre ce résultat

unique : il l'est à un isomorphisme près (ou modulo isomorphisme) (ang. *up to isomorphism* [CMR⁺97]). Pour le problème de l'admission on peut le résumer par "peu importe qui entre, ce qui le caractérise c'est qu'il entre".

Le piège réside dans la facilité de dessiner des classes de graphes modulo isomorphisme (des *graphes abstraits* [Cou96, BC87]). Il suffit d'oublier l'identité des noeuds et des arêtes [CMR⁺97] ! Dans un cadre applicatif, c'est la politique par défaut : quel logiciel de dessin vous indique la référence des objets qu'il dessine ? Dans un cadre formel, il faut encore ajouter du formel pour permettre cette ablation ! En l'absence de cet ajout, on risque de simplifier exagérément le problème, voire pire, de le passer sous silence. Cherchant des débouchés applicatifs, nous avons longtemps pris pour argent comptant cette simplification, mais lorsqu'il fallut rédiger définitivement et lui donner un cadre formel, le piège s'est refermé.

Parvenir à trouver une formulation théorique correcte qui utilise cette propriété n'est pas sans difficulté. En effet, on ne peut pas définir de morphisme entre des classes d'équivalence, car on ne sait pas réaliser la composition [CEL⁺94]. Pour en donner une raison intuitive, comment composer puisque tout est indifférencié au point de contact ? La réponse à cette question forme le point de départ du chapitre II.

1.4.4 Sémantique opérationnelle

Nous avons considéré jusque là l'existence de la dérivation sans nous poser le problème de sa construction. Cette question est maintenant abordée suivant la théorie classique [CMR⁺97] par la définition d'une condition de sensibilisation et d'une équation de tir. Cette présentation coïncide avec celle adoptée pour les Réseaux de Petri.

Sensibilisation: Construire le morphisme d'application

Soit un système composé d'une seule règle : comment décider s'il existe un successeur d'un graphe L donné pour ce système ? Décider de l'existence d'un morphisme d'application à partir de l'ensemble des morphismes possibles correspond à la condition de sensibilisation des grammaires de graphes qui se décompose en deux sous-conditions : la *condition de liaison* et la *condition d'identification* [Ehr86].

Propriété 1.5 (Existence d'un morphisme) *Soient deux graphes a et b , l'existence d'un morphisme entre a et b est décidable et NP-complet ainsi que l'énumération des morphismes possibles.*

Le seul morphisme qui appartienne au diagramme final et que l'on peut chercher à ce moment est le morphisme qui reliera dans la somme $\bullet l$ et L . Il faut donc caractériser les morphismes qui pourraient faire partie d'une somme amalgamée. Cette caractérisation est fondamentale et forme en fait la condition de sensibilisation. Pour la définir nous posons la définition de noyau d'une relation et d'un morphisme.

Définition 1.24 (Noyaux) *On appelle noyau $\ker(D)$ d'une relation $(D \subset A \times B)$, le plus petit sous-ensemble de A tel que, il existe $a \neq b \in A$ et $c \in B$ tels que $(a, c), (b, c) \in D \implies \{a, b\} \subset \ker(D)$. Cette notion s'étend aux fonctions qui sont des relations particulières, et aux morphismes de graphes qui sont des paires de fonctions. Soit $(f : g \rightarrow h)$ morphisme de graphe, on note $\ker(f) \subset g$ le noyau du morphisme f , on peut vérifier que c'est un graphe.*

$$\ker(f) \stackrel{\text{def}}{=} \left\langle \ker(f_V); \ker(f_E); \text{src}_{\sqrt{\ker(f_E)}}(-); \text{tgt}_{\sqrt{\ker(f_E)}}(-) \right\rangle$$

Le noyau regroupe tous les éléments de la source pour lesquels le morphisme (ou la relation) ne définit pas de vis-à-vis unique. On dira que des éléments du noyau sont confondus par le morphisme. Un morphisme dont le noyau est le graphe vide est dit injectif.

Propriété 1.6 (Sensibilisation) *Soit le système F formé d'une seule règle f , L a un successeur dans la dynamique associée à F ssi il existe un morphisme $(m : \bullet l \rightarrow L)$ et que ce morphisme satisfasse deux conditions :*

Condition de Liaison *Aucune arête du graphe L hors de l'image du pré-graphe n'est incidente à un nœud détruit par l'application de la règle :*

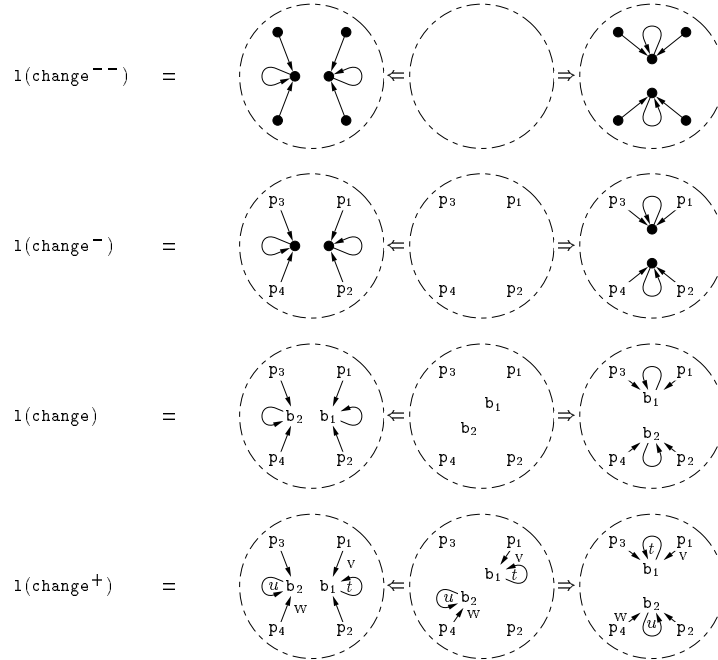
$$(x \xrightarrow{a} y) \in L - m(\bullet l) \implies \{x, y\} \cap m(\bullet l - \prec l(\bar{l})) = \emptyset$$

Condition d'Identification *Le noyau du morphisme m est inclus dans l'image du morphisme $\prec l$: Le morphisme d'application ne peut confondre que des éléments images de l'interface.*

Un morphisme $(m : \bullet l \rightarrow L)$ est un morphisme d'application si et seulement s'il satisfait les conditions de liaison et d'identification.

Exemples de sensibilisation Pour détailler les subtilités de ces conditions on va considérer un ensemble de règles cousines de la règle **change** ordonnées par interfaces croissantes comme données dans la figure 1.23. Comme le morphisme étudié reste constant, seules les conditions de sensibilisation sont concernées.

FIG. 1.23 – Règles **changes** ordonnées par interface croissantes



Montrons que si l'interface est croissante, la règle reste sensibilisée.

Liaison la partie supprimée $(g - m(\bullet l))$ est décroissante donc la condition de liaison est validée.

Identification le noyau reste dans l'image croissante de l'interface, la condition d'identification est validée.

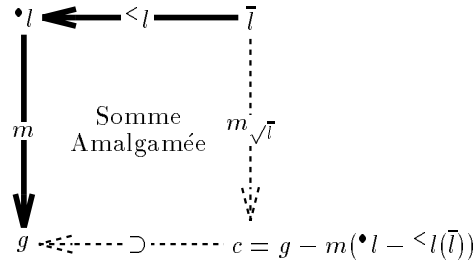
Par exemple considérons les différentes règles **change** données dans la figure 1.23.

- **change** est applicable, le détail de cette application est donné dans la figure 1.22.
- **change**⁺ a une interface plus grande que **change**, donc est applicable et donne le même résultat.
- **change**⁻ est applicable et donne le même résultat car on peut enlever les baguettes et les remettre. La condition de liaison est validée : l'ensemble des arêtes "en contact" avec f_2 sont dans le pré-graphe et l'interface ne contient pas d'arêtes. La condition d'identification est validée car les philosophes, bien que confondus appartiennent à l'interface.
- **change**⁻⁻ a une interface vide. Elle n'est pas applicable; comme tout philosophe a une image, le noyau est forcément non vide ($4 > 2$), il ne peut pas être contenu dans l'image de l'interface qui est vide. La condition d'identification est violée.

Tir: le complément de la somme

Soit une règle l et un graphe L pour lesquels on dispose d'un morphisme d'application ($\mathbf{m} : \bullet l \rightarrow L$). On obtient le contexte en enlevant du graphe la partie du pré-graphe n'étant pas dans l'image du morphisme gauche ($\prec l$) de la règle. L'opération associée calcule le *complément* de la somme, un contexte pour lequel le morphisme d'application, le morphisme gauche, l'injection canonique et la restriction du morphisme d'application forment une somme amalgamée comme indiqué dans la figure 1.24. Ceci ne fonctionne que si le morphisme gauche est injectif et si m vérifie les conditions de *liaison* et d'*identification*.

FIG. 1.24 – Complément de la somme



Notez ici l'apparition d'un signe "−" comme pour la sémantique opérationnelle des réseaux de Petri. Cette opération est définie par extension de la soustraction des ensembles, la validité de son utilisation est subordonnée aux conditions de sensibilisation.

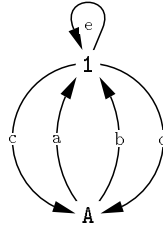
Une fois que l'on a obtenu le contexte, l'obtention du résultat s'obtient par construction d'une somme amalgamée, on choisit en général une formule conservant les références du contexte : on ajoute au graphe contexte une copie disjointe de la partie du post-graphe n'appartenant pas à l'image du morphisme droit. Dans cette optique opérationnelle, comme il existe autant de copies disjointes que l'on veut, on retrouve l'idée que le résultat n'est pas unique.

1.4.5 Comparaison avec la logique de réécriture

Notre motivation pour introduire les grammaires de graphes tenait à l'accessibilité du terme suivant à partir de **rien** avec la théorie de réécriture associée au “gala des philosophes”.

$$\begin{aligned} & \mathbf{Oisif}(A) \oplus (A \hookrightarrow 1) \oplus (A \hookrightarrow 1) \\ & \oplus \mathbf{Encas}(A, 1, 1) \end{aligned}$$

On peut construire informellement un équivalent à ce terme sous la forme du graphe suivant.



Ce graphe n'est pas accessible à partir du graphe vide. Pour s'en convaincre, il faut utiliser un invariant. Un invariant de notre système est que le nombre de sommets multiplié par trois est égal au nombre d'arêtes multiplié par deux. Cette propriété n'est pas vérifiée par le graphe ci-dessus, il n'est donc pas accessible. Dès que l'on s'intéresse à des systèmes avec apparitions/destructions de références, les deux formalismes de Haut-Niveau divergent de façon apparemment irréductible.

Par rapport à la logique de réécriture, les grammaires de graphes souffrent de deux limitations importantes:

- Le typage des noeuds (et des arêtes) du graphe n'est pas une possibilité par défaut de la théorie. Le problème du typage est étudié dans la sous-section suivante, deux solutions peuvent être trouvées dans la littérature à ce sujet, nous les montrerons toutes les deux et nous les comparerons.
- l'impossibilité de dupliquer des sous-graphes[Cou95], restera une limitation structurelle de l'approche suivie. Les solutions pour circonvenir à cette limitation ne sont pas satisfaisantes dans le cadre des approches par somme [KLG93] car elles font intervenir un ensemble dénombrable de règles. Notons ici que les systèmes de réécriture de graphes à base de double-pullback (i.e. double-produit) [Bau95, BJ97] permettraient de dépasser cette limitation. Cette approche n'est pas traitée dans ce mémoire car nous avons rencontré cette technique tardivement. En revanche, elle fait partie intégrante de nos perspectives de travail.

En logique de réécriture, cette duplication ne pose pas de difficulté: la règle ci-dessous en témoigne. Même si elle n'a pas grand sens dans le contexte des philosophes, et plus généralement dans les systèmes à multi-ensemble de tuples, elle duplique un sous marquage quelconque (α est une variable de sorte **soupe**) :

$$\alpha \rightarrow \alpha \oplus \alpha$$

Cette possibilité sera exploitée au début du chapitre IV de façon plus constructive sur une caractérisation de signature orthogonale à celle sous-jacente à celle des philosophes: les arbres commutatifs qui disposent du test à zéro et pour lesquels ce type de réécriture permet d'“initialiser un compteur”. Notons que pour ce type d'application, la condition de liaison permet de réaliser le test à zéro. Tester l'égalité des degrés entrants de deux noeuds est, par contre, hors de portée des grammaires de graphes à double-somme.

1.4.6 Le problème du typage

Les graphes dirigés sont une structure assez pauvre symboliquement. On pourrait néanmoins montrer que l'on peut réaliser la machine de Turing avec des grammaires de graphes dirigés. Cette propriété d'expressivité est aussi suggérée par la remarque que la condition de liaison permet de réaliser le test à zéro. Mais l'expressivité théorique d'un formalisme et le confort qu'il propose au spécifieur sont deux choses différentes. La base du confort moderne est le "typage" : donner aux noeuds du graphe un type qui permet d'éviter de les "confondre" inopinément. Il existe, de notre point de vue, trois grandes approches pour permettre un "typage" des noeuds et des arêtes d'un graphe.

Par invariant [GV97] : c'est l'approche suivie pour le bal des philosophes. Ceci est une technique de typage *a posteriori* utilisable sur toutes les extensions à typage *a priori* qui suivent.

Graphes typés [CM92] : cette approche, introduite dans [CH95] considère un graphe typé comme étant un morphisme de graphe dont la cible est fixée et correspond au graphe typant. Ainsi tout élément de la source du morphisme est typé par son image dans le graphe cible.

Algèbres Unaires [Löw93, BC87] : les algèbres construites sur une signature multi-sortes peuvent être interprétées comme des graphes [Löw93, BC87]. [Löw93] caractérise, de fait, les signatures dont les algèbres disposent d'une condition de sensibilisation : les signatures n'ayant que des opérateurs unaires.

La première technique peut toujours se rajouter sur d'autres techniques de typage *a priori*. Le véritable choix se situe donc entre les deux dernières approches. Pour motiver notre choix de la dernière approche nous allons étudier l'exemple des philosophes. Il a été réalisé par la technique des invariants et nous allons le réaliser avec les graphes typés et les algèbres unaires. Nous pourrions comparer ainsi les deux approches.

Les philosophes typés

Définition 1.25 (Graphe typé) Soit TG un graphe dirigé, on note $\mathcal{G}[TG]$ l'ensemble des morphismes de graphes de cible TG .

Nous voulons exprimer les contraintes suivantes, codées dans le graphe donné figure 1.25.

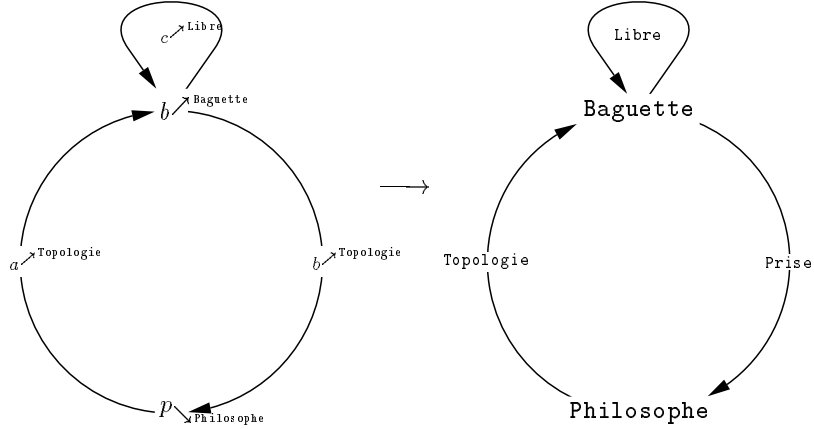
- Deux types de noeuds (**Philosophe**, **Baguette**)
- Trois types d'arêtes (**Libre**, **Prise**, **Topologie**) vérifiant
 - Une arête **Libre** va d'une **Baguette** à une **Baguette**,
 - Une arête **Prise** va d'une **Baguette** à un **Philosophe**,
 - Une arête **Topologie** va d'un **Philosophe** à une **Baguette**.

Les conventions posées pour la représentation des morphismes ne sont plus adéquates pour représenter les graphes typés. En effet, dès que deux éléments ont le même "type" la préservation des noeuds ne suffit plus pour représenter la fonction. Nous avons choisi la convention suivante $b \nearrow^{\text{Topologie}}$ pour indiquer que l'image de b par le morphisme typant est **Topologie**.

Définition 1.26 (Morphisme de graphe typé) Un morphisme de graphes ($\mathbf{f} : a \rightarrow b$) est un morphisme de graphe typé ($\mathbf{f} : (h_a : a \rightarrow TG) \rightarrow (h_b : b \rightarrow TG)$) ssi

$$h_b \circ \mathbf{f} = h_a$$

FIG. 1.25 – Un graphe typé (à gauche) par le graphe typant (à droite)



L'obtention de la somme amalgamée s'obtient par la propriété d'universalité. Quant aux conditions de sensibilisation et aux équations de tir, elles se généralisent sans difficulté avec le même type de restrictions.

Les philosophes unaires

Nous allons maintenant envisager l'autre possibilité pour typer les noeuds et arcs d'un graphe. Soit Σ une signature multi-sortes. Une algèbre sur cette signature est construite en associant à chaque sorte un ensemble, appelé ensemble porteur, et à chaque opérateur une fonction, appelée opération. Ce sont ces algèbres qui vont former l'extension algébrique de la notion de graphe dirigé.

Définition 1.27 (Σ -Algèbres [EM90, BC87]) Une Σ -algèbre est une structure

$$a \stackrel{\text{def}}{=} \left\langle \{A_\sigma\}_{\sigma \in \text{Sortes}_\Sigma} ; \{a_\phi\}_{\phi \in \text{Opérateurs}_\Sigma} \right\rangle$$

avec

- $\{A_\sigma\}_{\sigma \in \text{Sortes}_\Sigma}$ une famille d'ensembles indexée par l'ensemble des sortes. Nous appellerons A_σ ensemble porteur de la sorte σ pour l'algèbre a .
- $\{a_\phi\}_{\phi \in \text{Opérateurs}_\Sigma}$, une famille de fonctions (appelées opérations) indexée par l'ensemble des opérateurs telle que $(a_\phi : A_{k_1} \times \dots \times A_{k_n} \rightarrow A_{k_{n+1}})$.

Parmi toutes ces algèbres, [Löw93] caractérise une famille de signatures pour laquelle "l'approche par simple pushout" qu'il présente va fonctionner. Cette propriété de cette famille de signatures conserve un intérêt opérationnel pour nous car elle permet de disposer d'une condition de sensibilisation: d-complet et d-injectif [Löw93].

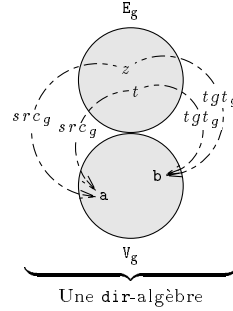
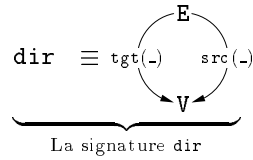
Définition 1.28 (Signature de graphe [Löw93]) Une signature de graphe est une signature multi-sortes ne contenant que des opérateurs unaires.

Extension des graphes dirigés Ensuite, il nous faut montrer que cette définition étend bien la notion de graphe dirigé. Pour ce faire il faut considérer la signature :

Sortes $\{V, E\}$.

Opérateurs $\begin{cases} \text{src}(-) : E \rightarrow V \\ \text{tgt}(-) : E \rightarrow V \end{cases}$

Une remarque qui s'impose ici, sans dévoiler trop ce qui vient, est qu'une signature de graphe est syntaxiquement un graphe dirigé. Par exemple la signature **dir** peut se représenter par le graphe dirigé ci dessous à gauche. Représenter une algèbre sous forme de diagramme de Venn (à droite) permet de conserver la forme du graphe associée à cette signature unaire. Le graphe représenté ici est la source du morphisme m donné figure 1.18.



Morphismes de Σ -algèbres Soit a et b deux Σ -algèbres unaires, un morphisme ($\mathbf{f} : a \rightarrow b$) de Σ -algèbre (un Σ -morphisme) est une famille de fonctions entre les ensembles porteurs de la source vers les ensembles porteurs de la cible préservant la structure d'algèbre.

Définition 1.29 (Σ -Morphismes) Une famille de fonctions f **Sortes_S-indexée** est un Σ -morphisme ($\mathbf{f} : a \rightarrow b$) ssi pour tout opérateur $\phi : x \rightarrow y$, on a :

$$f_y \circ a_\phi = b_\phi \circ f_x$$

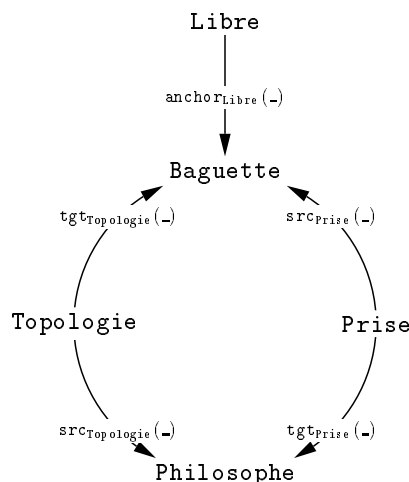
Le couple Σ -algèbres/ Σ -morphisms dispose de sommes amalgamées [Pie91].

Soit $\mathcal{T}^\Sigma(\emptyset)$, l'algèbre des termes sans variables. On construit cette algèbre en prenant comme ensemble porteur de sorte s les éléments de $\mathcal{T}^S(\emptyset)_s$, les opérations étant définies conformément à la structure des termes. Pour toute Σ -Algèbre a il existe un unique morphisme ($!_a : \mathcal{T}^\Sigma(\emptyset) \rightarrow a$) [Pie91].

Comme il n'y a pas de constantes dans une signature unaire, $\mathcal{T}^\Sigma(\emptyset)$ est l'algèbre vide, i.e. dont tous les ensembles porteurs et tous les opérateurs (i.e. des fonctions donc des relations) sont réduits à l'ensemble vide. Il s'ensuit que l'unique morphisme de l'algèbre vide vers toute algèbre est la famille **Sortes_S-indexée** d'ensembles vides. Les ensembles porteurs de la source étant vide, toute relation donc a fortiori toute fonction ayant pour source l'algèbre vide est unique, il s'agit de la relation vide (qui est une fonction dans ce cas). La propriété de morphisme s'exprimant par quantification universelle, elle est ici trivialement satisfaite car opérant sur un ensemble vide.

Retour sur l'exemple On donne dans la figure 1.26 une signature permettant de typer les nœuds du "bal des philosophes". Cette signature est donnée sous forme de graphe comme la remarque précédente détaillant cette notation pour les graphes

FIG. 1.26 – Une signature pour le gala des philosophes



dirigés nous y autorise. Les informations que l'on cherche à coder sont les mêmes que pour les graphes typés. La comparaison des deux approches est faite dans la partie suivante.

La difficulté de représenter une algèbre unaire sous une forme graphique est la même que pour les graphes typés. Il faut adjoindre une information sur les noeuds. Ici cependant, il s'agit d'une sorte, on peut lui associer une commande de dessin, une *layout*. Nous ne développerons pas formellement cette possibilité car les algèbres seront toujours considérées abstraitement ou sous forme de diagramme de Venn. Notre moteur de représentation de graphes dynamiques en JAVA utilise ce type de construction, et place les arêtes comme des objets à part entière. Ainsi les arêtes ayant même source et même cible sont représentées par une accumulation de triangles reliés aux sommets source et cible, accumulation qui rappelle les jetons d'un réseau de Petri.

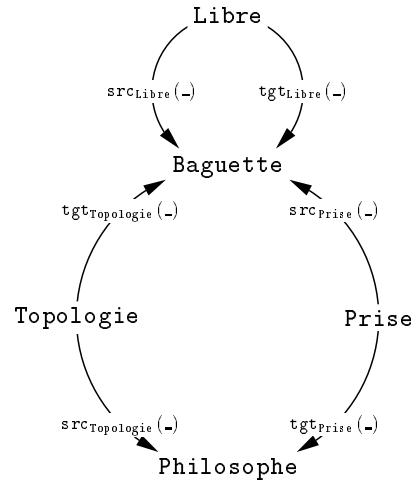
Comparaison des approches

L'approche par signature étend l'approche par graphe typé, on peut représenter tout graphe typé par une algèbre d'une signature. L'idée de la construction consiste à remplacer chaque arête ($j : i \rightarrow j$) du graphe typant par un motif $\langle (src_f(-) : f \rightarrow i); (tgt_f(-) : f \rightarrow j) \rangle$. La figure 1.27 détaille cette signature.

Confrontons les deux signatures données Figures 1.26 et 1.27 : la différence est minime, dans une "Libre" est de degré 2 (Fig. 1.26), dans l'autre "Libre" est de degré 1 (Fig. 1.27). L'avantage induit est de notre point de vue décisif. L'approche par algèbre permet de considérer des "attributs" (comme **libre**), qui sont des "flèches" qui n'ont pas de cible mais qu'une source, que l'on appelle "ancrer".

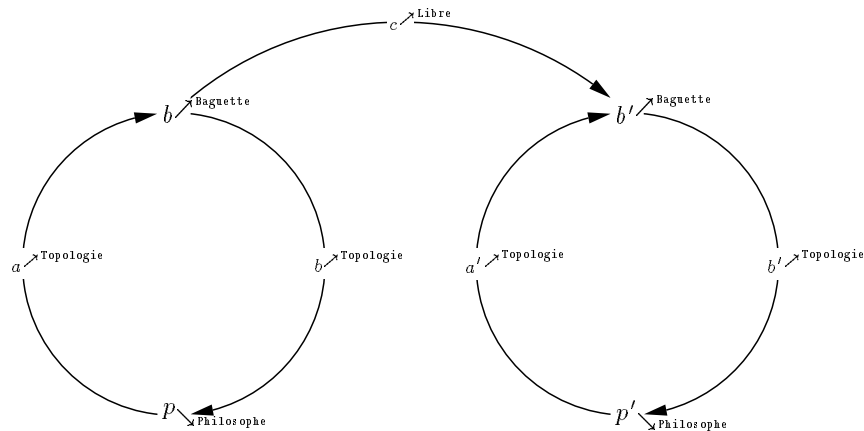
Ceci permet de faire une claire différence entre ce qui est de l'ordre du "comportement" d'un nœud et de l'état de ses "relations" avec les autres noeuds. Pour illustrer cette différence, on construit le graphe typé donné figure 1.28 qui est valide du point de vue du typage mais où deux fourchettes sont reliées par une arête libre. Ce type de configuration est impossible avec les signatures. Plus généralement, un graphe typé par la signature fournit une représentation de toute algèbre de cette signature mais cette représentation n'est pas univoque. L'approche par signature a donc "un grain plus fin" dans sa représentation des systèmes, et la finesse de ce grain

FIG. 1.27 – Signature équivalente au graphe typant



est particulièrement sensible pour pouvoir représenter des attributs ancrés sur des noeuds (comme des jetons dans une place) qui ne sont pas des éléments relationnels et donc ne devraient pas pouvoir être confondus avec eux.

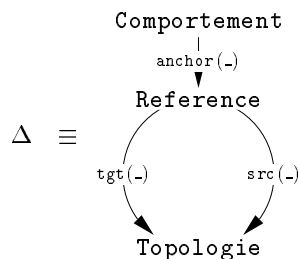
FIG. 1.28 – Un graphe typé “correct”



L’abandon des graphes typés ne se fait pas sans douleur car le contexte de travail développé dans cette théorie est très intéressant. En effet un morphisme de graphe typant permet toujours de transformer les graphes typés et leurs morphismes de façon “cohérente” ce qui n’est pas le cas d’un morphisme de signature (i.e un morphisme de graphe). Nous sommes convaincus que la théorie des graphes typés peut être incluse dans notre approche, dont par exemple, les morphismes permettraient de caractériser des morphismes de signatures qui disposeraient de cette “cohérence” et de ces utilisations prometteuses.

Néanmoins, l’utilisation conjointe des deux techniques est parfaitement possible.

Au début de notre travail, nous avons introduit un modèle de gestion de la coopération à base de graphes qui reposait sur une séparation explicite de la topologie et du comportement [GV99]. Le fait que notre sémantique soit écrite dans une approche ensembliste nous avait valu des commentaires récurrents nous intimant de nous confronter à la théorie que nous venons de présenter. De fait, ce modèle s'exprime très simplement en combinant les deux approches. Il suffit de remplacer dans la définition des graphes typés les graphes dirigés par des algèbres de la signature unaire suivante:



En appliquant le même type de construction que pour les graphes typés, on peut obtenir exactement la signature recherchée. La limitation de cette technique tient au “degré” des éléments ainsi définis, i.e. le nombre de fonctions qui partent d’un élément. Avec seulement 2 comme avec les graphes typés, on confond topologie et comportement, avec $\{1,2\}$ comme notre construction le suggère, nous parvenons à séparer ces deux composantes [Sch93]. Mais pourquoi devrait-on se limiter à un degré borné si on peut utiliser les signatures?

Cette dernière construction montre que le couple structure/morphisme sur laquelle on réalise la somme amalgamée importe peu. Se doter d’opérateurs permettant de propager l’existence de la somme amalgamée pour étoffer les structures utilisées d’une part, et travailler sur le noyau de la sémantique par double somme d’autre part, sont deux choses finalement distinctes. Cet état de fait sera mis en valeur dans notre partie II où nous présenterons la sémantique des grammaires de graphes indépendamment du couple structure/morphisme pour ensuite détailler les différents opérateurs permettant de construire des structures algébriques complexes. Ceci explique le titre du deuxième chapitre “Grammaires à double-somme” car le fait que la catégorie soit une catégorie de structures “graphiques” n’est pas une nécessité. Lever cette contrainte sera ensuite exploité pour traiter un exemple, les apartés (cf. 3.4), qui ne peut se réaliser simplement avec des “graphes” (même étiquetés [CMR⁺97], typés [CM92] ou unaires [Löw93]), mais nécessite de considérer des “graphes de graphes”.

Chapitre

2

Grammaires à double-somme

La sémantique des grammaires de graphes que nous avons exposée présente plusieurs limitations et ce chapitre va s'attacher à les dépasser:

- l'impossibilité de disposer d'un graphe des états accessibles comme pour les Réseaux de Petri ce qui permettrait l'utilisation des techniques connues d'analyse sur ces représentations. Cette impossibilité est une résultante de la non-unicité de la somme amalgamée: tout état isomorphe à un état accessible est accessible (Cf. 1.4.4);
- le fait de travailler sur une structure fixée dans le formalisme. Cette difficulté provient de l'utilisation d'une opération nécessitant d'être définie pour chaque structure: le *complément*. En outre, cette opération est complexe, et montrer que la figure 1.24 (page 51) forme une somme amalgamée relève déjà d'une technicité certaine.

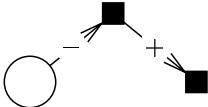
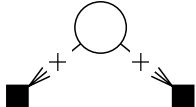
Pour résumer, nous avons un problème de chaque côté, à gauche le *complément* fige la structure utilisée, à droite la somme amalgamée n'est unique qu'à un isomorphisme près.

Il existe une différence importante entre les sémantiques des Réseaux de Petri et des Grammaires de Graphes d'une part, et de la Logique de Réécriture d'autre part. Les sémantiques classiques des réseaux de Petri et des Grammaires de Graphes sont à base de sensibilisation/tir, et permettent de calculer effectivement l'ensemble des possibilités de façon constructive. Nous appelons ces sémantiques *opérationnelles*. Nous lui opposons le terme de sémantique *dénotationnelle* lorsque l'ensemble des possibilités n'est pas construit par la sémantique mais reste du ressort de l'implémentation effective.

Dans les sémantiques opérationnelles, déterminer si une transition (i.e. règle, production...) est "sensibilisée" correspond à s'assurer qu'une soustraction est possible (e.g. dans les Réseaux de Petri, que les places d'entrée soient suffisamment marquées). Le tir correspond à cette soustraction suivie d'un ajout. Or cette

soustraction n'est pas toujours interne. C'est le cas dans les réseaux de Petri et les Grammaires de Graphes. La soustraction n'existe pas dans \mathbb{N} comme loi de composition interne; pour pouvoir considérer cette opération comme interne, il faut se placer dans \mathbb{Z} [Reu89] (i.e. les entiers relatifs ou signés) et, une fois cela réalisé, on peut écrire $(5 - 3 \notin \mathbb{N})$, ce qui signifie en fait $(5 - 3 \in \mathbb{Z} \wedge 5 - 3 \notin \mathbb{N})$. De la même façon l'opération qui consiste à “enlever” un graphe n'existe pas; on ne sait que les “ajouter”, en paramétrant cet ajout en spécifiant comment il se “colle” sur “l'existant”. Le *complément* est cette “soustraction de graphes”. Une différence de taille étant que l'on ne dispose pas dans ce contexte de “sur-ensemble” où cette soustraction est simple.

FIG. 2.1 – Deux types de sémantiques

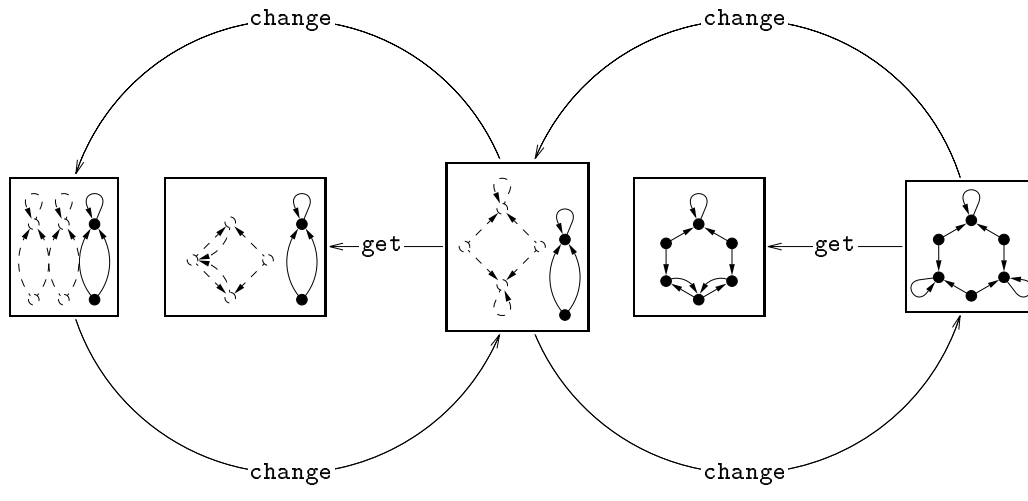
| | Opérationnel | Dénotationnel |
|-----------------------|---|---|
| Schéma abstrait |  |  |
| Logique de réécriture | <i>Outil</i> [CDE+99] | Sémantique [Mes92] |
| Réseaux de Petri | Classique [Bra83, Rei85c] | [MM90, Juh99] |
| Grammaire de Graphes | Classique [Ehr86, CMR+97] | <i>Contribution</i> |

Une approche dénotationnelle permettrait de résoudre le problème en ne considérant que des “additions”. Une contribution centrale de ce mémoire est de proposer une sémantique dénotationnelle pour les grammaires de graphes. Pour faire ce changement de sémantique, l'idée maîtresse est de considérer le morphisme axe de symétrie comme le morphisme directeur et non le morphisme gauche. Le morphisme central étant directeur, la dérivation implique deux sommes amalgamées. Ce glissement qui, graphiquement, recentre l'élément directeur est bien un passage d'une sémantique opérationnelle à une sémantique dénotationnelle. Les deux limitations disparaissent :

- la condition d'unicité modulo un isomorphisme est valable pour les deux volets de la dérivation. Ainsi, la cible *et* la source deviennent unique modulo un isomorphisme. Il devient possible de considérer comme ensemble des états les *graphes abstraits* [Cou96, BC87] (i.e. classes de graphes modulo isomorphisme) pour parvenir à définir une dynamique sur ces *graphes abstraits*. On pourra ensuite appliquer un étiquetage univoque suivi d'un quotient pour obtenir une sémantique par entrelacement satisfaisante. Cette construction permet de considérer la figure 2.2 comme une représentation de notre graphe des états accessibles pour le “bal des philosophes”, sans **enter** et **leave** qui rendent le système non borné. La figure représente la configuration à trois philosophes (et trois baguettes). On a représenté en pointillé le graphe correspondant à la configuration à deux philosophes/baguettes inclus dans le premier. Ceci est la partie visible d'une contribution fondamentale qui permet d'utiliser tous les outils formels et logiciels travaillant sur une relation d'accessibilité étiquetée;

- comme on ne considère plus que des sommes amalgamées, l’existence de cette construction suffira pour définir la sémantique. Or cette construction est une construction catégorique. Nous pouvons donc nous appuyer sur cette théorie sans réserve puisque toutes nos opérations sont maintenant dans ce cadre. Le problème de trouver un contexte d’application d’une règle devient un problème opérationnel qui conserve son importance mais n’intervient plus dans les preuves formelles que l’on pourra réaliser sur ces systèmes.

FIG. 2.2 – Le “bal des philosophes” à 3 philosophes (à deux philosophes en pointillés)



Libéré de la contrainte d’une structure fixe, nous élaborons un contexte formel permettant de doter les structures de deux opérateurs. Le premier est appelé opérateur d’architecture et permet de considérer une architecture statique de composants. Le second est appelé opérateur de famille et permet de considérer une collection dynamique non structurée de composants. La combinaison de ces deux opérateurs est illustrée par la spécification d’un mécanisme d’apartés dans des groupes coopératifs, ceci reposant sur la réalisation d’une architecture dynamique de composants.

La première section détaille les connaissances élémentaires de théories de catégorie pour comprendre l’expression de la sémantique. La deuxième section s’attache à la définition de la sémantique d’une grammaire à double-somme (amalgamée) et propose une expression compacte des identifiants de transformations sous forme de diagrammes de forme T. La troisième section détaille notre algèbre de structure, tandis que la dernière section développe notre spécification des apartés.

2.1 Une théorie de fonctions

Autant nous avons fait l’économie d’un recours au vocabulaire catégorique, il nous faut à présent en introduire les bases nécessaires à notre discours. La difficulté de cette théorie tient à sa structure. La théorie des catégories est une théorie à deux étages : le premier, “la théorie des fonctions”, est accessible et surtout simplificateur et permet les développements du second, “la théorie des catégories”, techniques, mais dont la puissance d’expression est à la hauteur des simplifications réalisées. Présenter la théorie des catégories par ses réalisations, c’est comme prendre un ascenseur sur son toit : ébahi par toute cette technique, mais quel inconfort

tout de même. Cette remarque a amené un certain nombre d'ouvrages pour les "mathématiciens du discret" vantant les mérites de cette théorie. En montrant les applications qui existaient dans ce domaine, ils munissaient l'ascenseur de boutons compatibles avec les préoccupations de cette mathématique. Parmi ces ouvrages, nous citerons [Pie91] qui est une introduction claire à cette théorie, et [RB88] qui permet de manipuler ces constructions en ML.

Le point de départ de cette théorie est la définition d'une catégorie: Une catégorie est un graphe dirigé auquel on ajoute des opérations d'identité et de composition. Il est essentiel de comprendre que ce n'est rien de plus.

Définition 2.1 (Catégorie [Pie91]) Une catégorie \mathbf{C} est

- une collection d'objets appelée $\mathbf{obj}(\mathbf{C})$, dénotés par a, b, c, \dots
- une collection de flèches appelée $\mathbf{arr}(\mathbf{C})$, dénotées par f, g, \dots
- quatre opérateurs :

Domaine qui assigne à chaque flèche f un objet source $a = \mathbf{dom}(f)$.

Co-domaine qui assigne à chaque flèche f un objet cible $a = \mathbf{cdm}(f)$.

Identité qui assigne à chaque objet a une flèche $(\mathbf{Id}^a : a \rightarrow a)$;

Composition qui assigne à chaque paire de flèche $\langle (g : b \rightarrow c); (f : a \rightarrow b) \rangle$ telle que $\mathbf{dom}(g) = \mathbf{cdm}(f)$ une flèche $g \circ f$ appelée leur composition, avec $(g \circ f : \mathbf{dom}(f) \rightarrow \mathbf{cdm}(g))$.

- vérifiant les deux propriétés suivantes:

Associativité Soit des objets et des flèches dans la configuration suivante :

$$a \xrightarrow{f} b \xrightarrow{g} c \xrightarrow{k} d, \text{ on a toujours l'égalité } k \circ (g \circ f) = (k \circ g) \circ f.$$

Loi de l'unité Toute composition des flèches $(f : a \rightarrow b)$ et $(g : b \rightarrow c)$ avec la flèche identité \mathbf{Id}^b donne : $\mathbf{Id}^b \circ f = f$ et $g \circ \mathbf{Id}^b = g$.

Il est très important de remarquer que les noms des objets et des flèches sont uniques, pour un nom donné il n'y a qu'une seule flèche ou qu'un seul objet. L'exemple souvent donné en exemple est la catégorie **Set** qui a pour objet les ensembles et pour flèches les fonctions (totales). Il est important de comprendre que l'on ne définit pas les ensembles avec les catégories mais qu'il s'agit d'une *présentation*, ceci à la différence de l'exemple suivant où l'on va définir un monoïde comme une catégorie particulière.

Exemple 2.1 (Monoïde) Un monoïde libre correspond à une catégorie à un seul objet. Chaque générateur est une flèche sans composition et chaque mot peut être obtenu par composition, l'identité étant l'identité de l'unique objet.

Comme les catégories sont des "graphes dirigés auxquels on ajoute des opérateurs", on peut considérer sans trop se poser de questions des "morphisme de catégories" : on les appelle des foncteurs. On peut donc définir la catégorie **Cat** qui a pour objet les catégories et comme flèches les foncteurs.

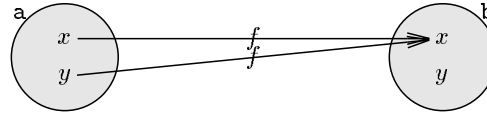
Définition 2.2 (Foncteurs [Mac71]) Un foncteur est un morphisme de catégorie. Plus en détail, un foncteur $(F : \mathbf{A} \rightarrow \mathbf{B})$ est défini par deux fonctions reliées: la fonction des objets qui associe à chaque objet de \mathbf{A} un objet de \mathbf{B} et la fonction des flèches qui associe à chaque flèche f de \mathbf{A} une flèche de \mathbf{B} telle que

$$\begin{aligned} \mathbf{dom}(F(f)) &= F(\mathbf{dom}(f)) & , & \quad \mathbf{cdm}(F(f)) = F(\mathbf{cdm}(f)), \\ F(\mathbf{Id}^a) &= \mathbf{Id}^{F(a)} & , & \quad F(g \circ f) = F(g) \circ F(f) \end{aligned}$$

Nous allons maintenant présenter une catégorie très utile pour la suite de nos développements, la catégorie **Set**. Cette présentation utilisera l'implémentation de cette catégorie telle qu'elle est présentée dans [RB88].

2.1.1 La catégorie Set en ML

Nous allons illustrer la définition fondamentale avec la catégorie **Set** qui servira de point de départ de nos constructions. Vous souvenez-vous de l'approche de la théorie des ensembles par diagramme de Venn que proposaient les Mathématiques Modernes?



Et bien, la théorie des catégories débute par l'observation qu'un grand nombre de propriétés des systèmes mathématiques peuvent être unifiées et simplifiées par une *présentation* avec des diagrammes de flèches. Le schéma précédent est alors représenté par la "flèche":

$$(f : a \rightarrow b)$$

La catégorie **Set** est définie en prenant comme ensemble des objets les ensembles et comme flèches les fonctions (totales). Il s'avère qu'un langage fonctionnel est très adapté pour réaliser des implémentations de constructions catégoriques. Nous avons implémenté en ML un petit moteur de grammaires de graphes sur cette base (suivant en cela l'exercice 11 p.125 de [RB88]). Cela a marqué un tournant dans notre appréhension de cette théorie : elle était opérationnelle.

La première déclaration est la déclaration de la structure de données qui décrit une catégorie. Une catégorie est paramétrée par le type de ses objets et de ses flèches, et sa signature est la signature de ses quatre opérateurs (**dom**(_), **cdm**(_), (**_** **o** **_**), **Id**-). L'exception **nonComposablePair** fait partie intégrante de cette spécification. Cette exception permet de rendre la fonction de composition partielle. En effet la composition est assujettie à la contrainte que la source et la cible des composées doivent être identiques pour que la composition existe.

```
datatype ('o,'a)Cat = cat of ('a->'o)*('a->'o)*('o->'a)*('a*'a->'a) ;
exception nonComposablePair;
```

La difficulté pour coder la catégorie **Set** en ML tient à la difficulté de représenter les ensembles alors qu'on ne dispose que de listes. ML propose un mécanisme permettant de cacher l'implémentation pour ne proposer que des opérations-interfaces qui vont se comporter comme si la structure était un ensemble. Ces fonctions reposent donc sur la librairie des listes; on ne donne volontairement que la signature pour exprimer que l'implémentation importe peu ici. Nous ne conservons de cette signature que la fonction qui nous sera utile ici : l'égalité d'ensembles. La présence de double-quotes ("*a*") dénote en SML l'existence d'une égalité sur le type '*a*'; il s'agit donc de la propagation de cette égalité : des éléments à l'ensemble.

```
abstype 'a Set
```

```
val setEq = -- : ''a Set * ''a Set -> bool
```

Une fois ceci défini, nous sommes dotés des objets de la catégorie cherchée; il nous faut maintenant en donner les flèches:

```
datatype 'a SetArrow = setArrow of ('a Set) * ('a -> 'a) * ('a Set);
```

Les différentes opérations qui forment une catégorie peuvent alors être définies sans grande difficulté:

```
fun setSrc(setArrow(a,-,-)) = a;
```

```
fun setTgt(setArrow(-,-,b)) = b;
```

```

fun setId(a) = setArrow(a,(fn x => x), a);
fun set0(setArrow(c,g,d),setArrow(a,f,b)) =
if setEq(b,c)
then setArrow(a,(fn x => g(f(x))),d)
else raise nonComposablePair;

```

Il est important de noter que l'on a représenté le caractère partiel de la composition par l'usage d'une exception. Nous pouvons à présent déclarer notre catégorie par la construction suivante:

```
val FinSet = cat(setSrc,setTgt,setId,set0)
```

Et l'interpréteur ML nous répond:

```
val FinSet = cat (--, --, --, --) : ('a Set, 'a SetArrow) Cat
```

Nous avons construit ici la catégorie **FinSet** qui est un modèle très fidèle de **Set**. L'idée fondamentale est que les ensembles sont des sous-ensembles des "termes" que l'on peut former comme étant de type '*a*'. Une originalité de la construction réside dans l'expression des flèches; le calcul associé à la fonction ne suffit pas, et il faut spécifier explicitement le domaine et le co-domaine de la fonction. Lorsque l'on indiquera des diagrammes catégoriques avec des expressions fonctionnelles, on oubliera de noter la source et la cible sur les flèches, cette information étant donnée graphiquement par la flèche.

Pour la plupart des expressions fonctionnelles ceci n'est qu'une simplification de notation sauf pour une expression fonctionnelle :

$$\subset = (\text{fn } x \Rightarrow x)$$

La catégorie induite par cette expression (où toutes les flèches ont cette expression) est un *pré-ordre*, des flèches ayant même source et même destination sont identiques (ici par construction, puisque l'on a fixé l'expression fonctionnelle). Cette construction sera utile dans la suite, on appellera cette sous-catégorie *le pré-ordre des inclusions*, et on la notera **Inc**.

La catégorie **Set** est la catégorie de base, et toutes les notations définies ici seront utilisées dans la suite dès que des diagrammes ou des expressions de **Set** seront utiles dans nos développements.

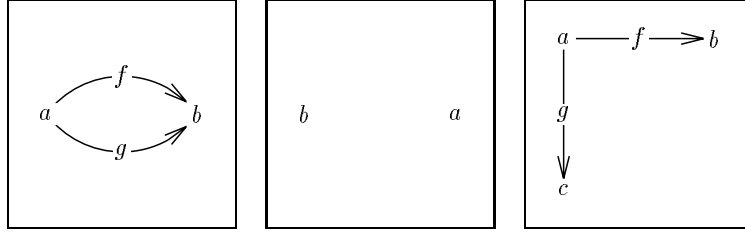
2.1.2 Constructions universelles

Un fondement de la théorie des catégories est l'identification d'éléments universels dans des collections d'objets ou de flèches. Pour illustrer cette notion d'universalité, on prend la notion d'élément initial. Un élément est initial (resp. final) dans une collection si et seulement s'il existe une unique flèche venant (resp. menant) de cet élément vers chaque élément de la collection. Par exemple dans **Set**, l'ensemble vide (\emptyset) est initial, l'unique fonction étant la fonction vide.

Cette démarche est appliquée sur des constructions appelée diagrammes. Un diagramme est un graphe dont les sommets et les arêtes sont étiquetés de façon cohérente par des objets et des flèches de la catégorie étudiée. On peut associer à chaque chemin du diagramme une flèche de la catégorie d'évaluation du diagramme, en vertu de la composition interne à la catégorie. Un diagramme est dit *commutant* ssi à toute paire de chemins ayant même source et même cible correspond la même flèche, et si à tout cycle correspond l'identité.

Nous commençons par définir les *co-limites* associées aux diagrammes donnés dans la figure 2.3. Elles sont données ci dessous dans l'ordre de lecture de cette figure. Pour représenter ces constructions universelles, on adoptera une convention de diagrammes quantifiés inspirée de [BN98]. Les flèches épaisses dénotent le quantifieur universel et les pointillés dénotent le quantifieur d'existence unique.

FIG. 2.3 – Diagrammes fondamentaux.



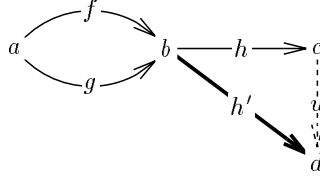
Note: Nous adoptons ici, pour rendre la présentation du co-égaliseur (à gauche) conforme aux exposés standards en la matière [Pie91, Mac71, RB88] la convention suivante : la contrainte de commutation ne s'applique sur les paires de chemins que si au moins un des deux est de longueur supérieure à deux, ainsi f et g ne sont pas forcément identiques dans le diagramme de gauche.

Cette quantification, l'*existence unique*, est le point essentiel des *constructions universelles*.

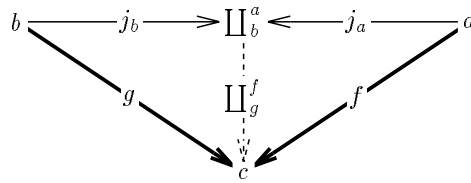
Définition 2.3 (Co-égaliseur) Un co-égaliseur d'une paire de flèches ayant même source et même cible,

$$\langle (f : a \rightarrow b); (g : a \rightarrow b) \rangle$$

est une flèche $(h : b \rightarrow c)$ vérifiant $h \circ f = h \circ g$ telle que pour toute $(h' : b \rightarrow d)$ vérifiant $h' \circ f = h' \circ g$ il existe une unique flèche $(u : c \rightarrow d)$ telle que $h' = u \circ h$.



Définition 2.4 (Co-Produit) Un co-produit d'une paire d'objets $\langle a; b \rangle$ est une paire de flèches $\langle (j_a : a \rightarrow \coprod_b^a); (j_b : b \rightarrow \coprod_b^a) \rangle$ ayant même cible \coprod_b^a (i.e. un résultat du co-produit) telle que pour toute paire de flèches $\langle (f : a \rightarrow c); (g : b \rightarrow c) \rangle$ il existe une unique flèche $(\coprod_g^f : \coprod_b^a \rightarrow c)$ telle que $\coprod_g^f \circ j_a = f$ et $\coprod_g^f \circ j_b = g$



Note aux lecteurs sur la notation choisie: Nous avons choisi un symbole qui en \LaTeX s'écrit `\coprod` et apparaît comme ceci \coprod_b^a pour écrire le coproduit, de préférence au “+” utilisé dans les ouvrages catégoriques pour éviter la confusion avec la somme dans les entiers utilisée par ailleurs et parfois simultanément (Cf. Chapitre III).

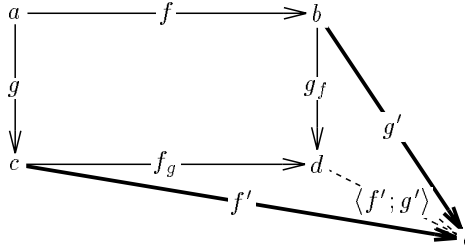
Définition 2.5 (Pushout) Un pushout d'une paire de flèches ayant même source

$$\langle (f : a \rightarrow b); (g : a \rightarrow c) \rangle$$

est une paire de flèches ayant même cible (appelé un résultat du pushout)

$$\langle (g_f : b \rightarrow d); (f_g : c \rightarrow d) \rangle$$

vérifiant $g_f \circ f = f_g \circ g$ telle que pour toute paire de flèches $\langle (g' : b \rightarrow e); (f' : c \rightarrow e) \rangle$ vérifiant $g' \circ f = f' \circ g$ il existe une unique flèche $\langle (f'; g') : d \rightarrow e \rangle$ telle que $\langle f'; g' \rangle \circ g_f = g'$ et $\langle f'; g' \rangle \circ f_g = f'$.

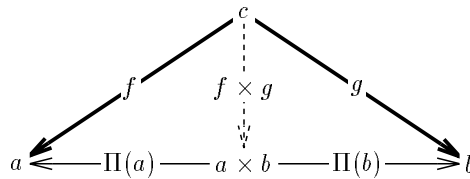


Une catégorie est dite finiment co-complète ssi il existe une co-limite pour tout diagramme fini et un élément initial. Les trois co-limites fondamentales sont des co-limites pour les diagrammes données dans la figure 2.3.

Propriété 2.1 (Co-complétude finie[RB88]) Soit \mathbf{X} une catégorie disposant d'un co-égaliseur, d'un co-produit et d'un élément initial, elle est finiment co-complète, elle dispose donc d'un pushout.

Nous travaillerons dans la suite avec des catégories finiment co-complètes. La sous-section suivante détaillera les constructions universelles pour \mathbf{Set} qui est, entre autre, finiment co-complète.

Constructions duales Ceci n'est que la moitié des constructions universelles fondamentales comme le laissait supposer le préfixe *co*. Les autres constructions, appelée limites, sont obtenues en renversant le sens de toutes les flèches[RB88], le dual de l'élément initial étant l'élément terminal, pour lequel, à tout objet, il existe une unique flèche ayant pour source cet objet et pour cible l'élément terminal. Dans \mathbf{Set} , un élément terminal est un singleton (i.e. un ensemble à un élément), le produit est le produit cartésien, et ses fonctions efférentes (i.e. qui en partent) sont les projections selon les composantes qui sont des injections. Le diagramme quantifié ci-dessous détaille la seule limite dont nous parlerons, le produit :



Notons qu'une approche récente des grammaires de graphes utilise le pullback (co-pushout) pour réaliser des transformations de graphes. Cette approche est très intéressante, et permet d'envisager des constructions hors d'atteinte des grammaires à pushout.

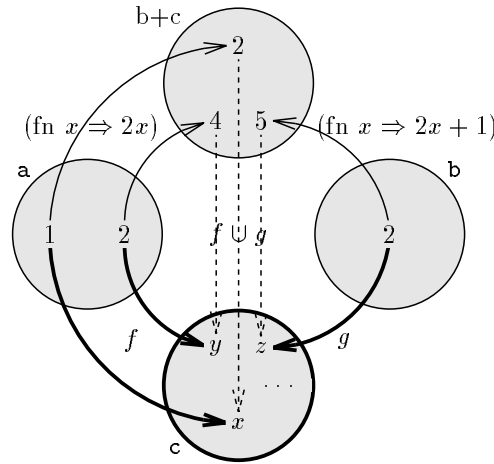
2.1.3 Constructions universelles dans Set

Nous allons maintenant donner les constructions universelles classiques pour la catégorie **Set**. Outre l'élément initial (l'ensemble vide) et un élément terminal (un singleton), nous donnons le co-égaliseur et le co-produit, ce qui nous permet d'affirmer l'existence du pushout comme une construction utilisant les deux précédentes. Enfin, nous considérerons le produit en donnant une application de cette dernière construction.

Le co-produit est une union disjointe et ceci même en cas de conflit de noms au départ entre les ensembles a et b ; le renommage nécessaire pour dépasser le conflit de nom est contenu dans les fonctions partant des ensembles, qui sont des injections. L'implémentation se fait souvent en trouvant un diagramme qui dispose de ces propriétés.

Une version arithmétique (Cf. figure 2.4) du co-produit et de prendre $(j_a : a \rightarrow \coprod_b^a) = (\text{fn } x \Rightarrow 2x)$ et $(j_b : b \rightarrow \coprod_b^a) = (\text{fn } x \Rightarrow 2x + 1)$ et d'en déduire \coprod_b^a en conséquence pour que les deux injections jointes soient surjectives. Pour le calcul de la partie universelle, il suffit de prendre l'union des relations obtenues en composant l'inverse de l'injection afférente à la fonction, cette union est bien une fonction, et son unicité est une conséquence des constructions.

FIG. 2.4 – Un Co-produit dans **Set** avec les entiers (x, y, z ne sont pas forcément distincts)

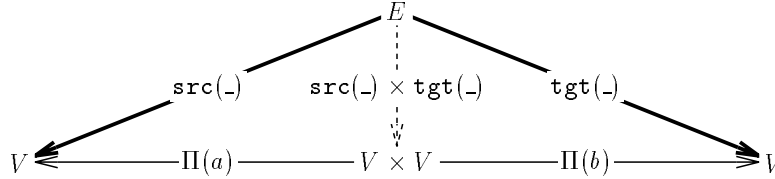


Le co-égaliseur s'obtient par un quotientage. On définit une relation sur b avec $x \sim y$ si et seulement si il existe $z \in a$ tel que $f(z) = x$ et $g(z) = y$, soit \simeq la fermeture symétrique, transitive et réflexive de \sim , et c l'ensemble des classes d'équivalence. La fonction $(h : b \rightarrow c)$ envoie chaque élément x de b sur sa classe d'équivalence $[x]_{\simeq}$, ainsi $h \circ f = h \circ g$. Maintenant, soit c' un ensemble et $(h' : b \rightarrow c')$ une fonction telle que $h' \circ f = h' \circ g$. On définit $(u : c \rightarrow c')$ par $u([x]_{\simeq}) \stackrel{\text{def}}{=} h'(x)$. Ceci est bien une fonction et on a bien $h' = u \circ h$. L'unicité de cette fonction est une conséquence de la construction. A ce niveau, on peut donc affirmer que **Set** est co-complète. Le pushout dans **Set** a été détaillé dans notre état de l'art sous le nom de somme amalgamée. L'existence de cette construction est une conséquence de l'existence du co-produit et du co-égaliseur. Le pushout est d'ailleurs calculé dans **Set** par un co-produit suivi d'un co-égaliseur[CMR⁺97, RB88].

Le pushout s'obtient en considérant le co-produit des cibles (en oubliant la source

commune), puis on co-égalise les injections obtenues en se souvenant de la source commune. Ce type de construction où l'ajout d'un sommet correspond à un co-produit et l'ajout d'une arête à un co-égaliseur permet de prouver par induction la co-complétude finie à partir des deux constructions élémentaires [RB88].

Le produit est le produit cartésien, les fonctions $\Pi(a)$ et $\Pi(b)$ étant les projections usuelles. Cette dernière remarque a une conséquence intéressante pour les graphes. Elle stipule l'équivalence entre la donnée $\langle V; E; (\text{src}(-) : E \rightarrow V); (\text{tgt}(-) : E \rightarrow V) \rangle$ et une forme plus compacte $\langle V; E; (\text{src}(-) \times \text{tgt}(-) : E \rightarrow V \times V) \rangle$. Donc la forme classique [RB88] des graphes dirigés est équivalente à la forme compacte [Ser80].



2.1.4 Isomorphisme

Nous allons maintenant considérer une notion clé pour notre problème : l'isomorphisme. Une flèche est un isomorphisme ssi elle a une flèche inverse qui composée avec elle donne l'identité. Dans **Set**, les isomorphismes sont les bijections, les fonctions dont l'inverse (en tant que relation) est une fonction.

Définition 2.6 (Isomorphisme) *Une flèche ($f : a \rightarrow b$) est un isomorphisme s'il existe une flèche ($f^{-1} : b \rightarrow a$), appelée l'inverse de f telle que $f \circ f^{-1} = \text{Id}^b$ et $f^{-1} \circ f = \text{Id}^a$.*

Deux objets sont dits isomorphes $A \sim B$ ssi il existe un isomorphisme entre eux. Deux objets isomorphes sont dits identiques modulo isomorphisme. Un objet A vérifiant une propriété P est dit unique modulo isomorphisme si tout objet vérifiant P est isomorphe à A .

Propriété 2.2 (Unicité des constructions universelles) *Tout résultat d'une construction universelle (objet initial, objet terminal, produit, co-produit, produit, pushout) est unique modulo isomorphisme.*

C'est une conséquence du quantificateur d'existence unique. Une première proposition est déduite de l'application de l'unicité au résultat : l'unique flèche commutante ne peut être que l'identité. La proposition finale se déduit d'une application à une paire de résultats qui fournit une paire de flèches dont les compositions ne peuvent être que l'identité en vertu de la proposition précédente.

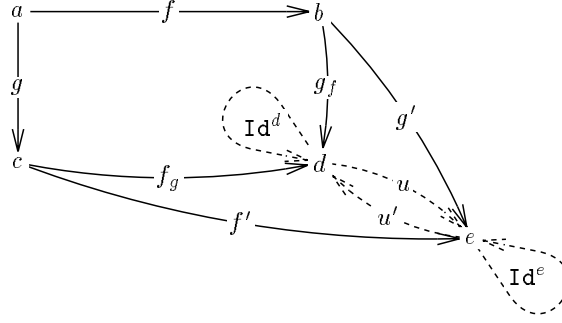
Cette démonstration est vraie dans toute catégorie, pour toute construction universelle. On la détaille par le diagramme ci-dessous pour le pushout pris avec la convention de commutation donnée précédemment. Les flèches en pointillés dénotent l'existence et l'unicité de chaque flèche de ce sous-diagramme et l'unicité des boucles caractérise l'isomorphisme.

Nous allons faire un usage courant de la définition suivante qui, à toute catégorie, associe l'ensemble des classes d'objets modulo isomorphisme. Nous avons essayé dans la suite de réserver les crochets uniquement à cet usage.

Définition 2.7 (Ensemble des classes d'objets) *L'ensemble des classes d'objets modulo isomorphisme sera noté $[X]$,*

$$[X] = \text{obj}(\mathbf{X})_{\sim \text{isomorphisme}}$$

FIG. 2.5 – Unicité du pushout modulo isomorphisme



Remarquons ici à titre d'illustration que $[\mathbf{Set}] \equiv \mathbb{N}$.

Attention ! $[X]$ est un ensemble, et non plus une catégorie. Toute la difficulté provient du fait que la structure de catégorie ne s'exporte pas en prenant comme ensemble d'objets les classes modulo isomorphisme. En effet, on ne peut pas parvenir à composer des flèches de manière compatible avec la composition sur la catégorie originale : au point de contact, tout est mélangé. Cette contrainte, fondamentale [CEL⁺94], induit pour la recherche d'une sémantique des grammaires de graphes une solution différente. Notre contribution est de dire : ce n'est pas de la même catégorie dont il s'agit. Pour illustrer notre démarche, nous dirons que *le* résultat d'un pushout (i.e. somme amalgamée) est la classe d'*un* résultat. Ainsi la catégorie qui a pour objet des classes a pour flèches des diagrammes dont la source et la cible s'obtiennent par des constructions universelles.

2.2 Sémantique dénotationnelle

Pour pouvoir donner une sémantique à un système de règles, il suffit que la catégorie de travail soit finiment co-complète. Nous allons suivre la démarche de la logique de réécriture, donner d'abord la dynamique abstraite, puis essayer de dégager les identifiants uniques des séquents utilisés.

Pour permettre la prise en compte des extensions développées par la théorie classique, nous séparons cette construction en deux parties. Une première partie définit un graphe abstrait de dérivation dans lequel une opération de composition existe, même si nous ne sommes pas parvenus à expliciter cette construction autrement que sur un exemple. Ensuite, une sémantique est définie comme une injection dans ce graphe abstrait permettant une définition unique pour les sémantiques abstraites non constructives et les sémantiques constructives pouvant être obtenues par calcul.

Un système est la donnée conjointe d'une catégorie finiment co-complète, ayant donc un pushout et un élément initial, et d'une famille de paires de flèches ayant même source.

Définition 2.8 (Système et Règles) *Un système $\langle \mathbf{X}; \langle \mathbf{L}; \mathbf{l}(-) \rangle \rangle$ est la donnée d'une catégorie finiment co-complète (appelée parfois catégorie de travail) et d'une famille de règles. Une famille de règles est composée d'un ensemble d'étiquettes L et d'une fonction $\mathbf{l}(-)$ qui, à chaque étiquette, associe une règle, où une règle est une paire de flèches de \mathbf{X} $\langle \langle l; l \rangle \rangle \in \mathbf{arr}(\mathbf{X})^2$ ayant même domaine, i.e. vérifiant :*

$$\mathbf{dom}(\langle l \rangle) = \mathbf{dom}(l \rangle)$$

FIG. 2.6 – Une règle: un diagramme de forme **Span**

$$\bullet_a \longleftarrow \langle a \longrightarrow \bar{a} \longrightarrow a \rangle \longrightarrow a \bullet$$

On note les différents éléments d'une règle comme suit:

$$\langle (\langle 1 : \bar{l} \rightarrow \bullet l \rangle; (1^> : \bar{l} \rightarrow l^{\bullet})) \rangle$$

Dans la suite on notera $\langle \mathbf{X}; \mathcal{L} \rangle$ un tel système, en abrégant la notation de la famille de règles lorsque l'explicitation de l'ensemble des étiquettes de règles n'est pas utile.

L'étape fondamentale est de considérer la sémantique d'un système comme une relation sur les classes d'objets. Cette idée est contenue dans une remarque de [BC87] qui stipule que la relation d'accessibilité est naturellement définie sur les classes d'objets. Pour mettre en valeur cette propriété, comparer la définition suivante avec la définition de dérivation (cf. 1.4) de notre état de l'art. La validité de cette définition repose sur l'unicité modulo isomorphisme de la construction universelle utilisée : le pushout (ou somme amalgamée).

Définition 2.9 (Dynamique abstraite d'un système) *A tout système $\langle \mathbf{X}; \mathcal{L} \rangle$ on peut associer la dynamique $\xrightarrow{\langle \mathbf{X}; \mathcal{L} \rangle} \subset [X] \times [X]$ suivante, induite par les séquents ci-dessous,*

Application pour tout $l \in L$:

$$\begin{array}{c} (\bar{\mathbf{m}} : \bar{l} \rightarrow K) \text{ est un morphisme de graphe} \\ L \text{ résultat du pushout de } \langle (\langle 1 : \bar{l} \rightarrow \bullet l \rangle; (\bar{\mathbf{m}} : \bar{l} \rightarrow K)) \rangle \\ R \text{ résultat du pushout de } \langle (1^> : \bar{l} \rightarrow \bullet l); (\bar{\mathbf{m}} : \bar{l} \rightarrow K) \rangle \\ \hline l \xrightarrow{\langle \mathbf{X}; \mathcal{L} \rangle} [R] \end{array}$$

Une première façon d'enrichir cette relation est de l'étiqueter. Comme le système est à base de règles, on peut considérer la relation étiquetée comme une famille de relations [Mil89] et associer à chaque règle sa dynamique propre.

$$\xrightarrow{\langle \mathbf{X}; \mathcal{L} \rangle} = \left\langle L; (\text{fn } x \Rightarrow \xrightarrow{\langle \mathbf{X}; \{x\}; 1_{\sqrt{\{x\}}}(_) \rangle}) \right\rangle$$

Cette forme de relation étiquetée n'identifie que les règles et donc chaque "arête" n'est pas uniquement identifiée, plusieurs arêtes partageant la même étiquette. Nous allons procéder à un étiquetage univoque de la relation d'accessibilité qui nous permettra de réaliser les preuves formelles d'extension dans le chapitre III.

2.2.1 Les identifiants de transformations

Nous allons réaliser le même type de construction que celle de la logique de réécriture; comme nous disposons d'une définition d'une dynamique sous forme de S.O.S[Plo81], dans les deux cas, l'idée consiste à étiqueter chaque élément de la dynamique avec le séquent qui a permis de le prouver. Par contre, nous ne coderons

FIG. 2.7 – Un identifiant concret: un diagramme de forme T

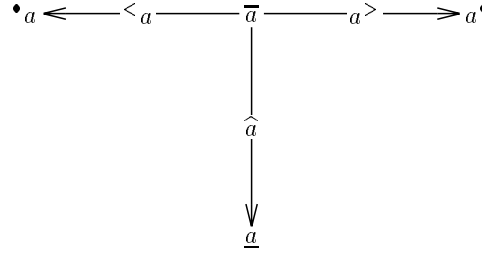
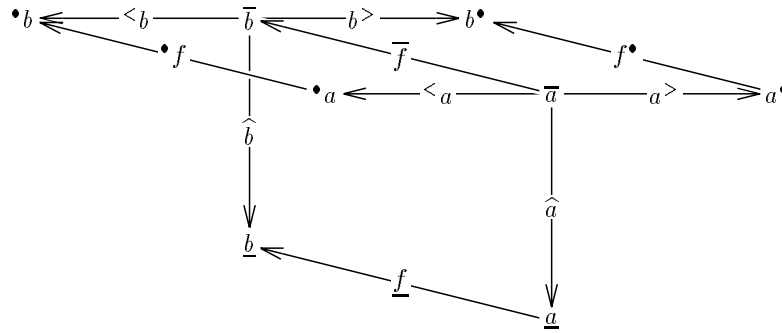


FIG. 2.8 – Diagramme du morphisme de T-diagramme



pas les co-limites impliquées, et nous coderons cet aspect dans les fonctions de source, de cible de l'arête.

Définition 2.10 (T-diagrammes) Un *T-diagramme* est un triplet $\langle \cdot t; \hat{t}; t \cdot \rangle \in \mathbf{arr}(X)^3$ ayant même domaine, i.e. vérifiant: $\text{dom} \cdot t = \text{dom} \hat{t} = \text{dom} t \cdot$. On note les différents éléments d'un T-diagramme comme dans la figure 2.7

Définition 2.11 (Morphisme de T-diagrammes) Un *morphisme de T-diagramme* $(f : a \rightarrow b)$ est un quadruplet de morphismes $\langle \bullet f; \bar{f}; f \cdot; \hat{f} \rangle \in \mathbf{arr}(X)^4$ vérifiant:

$$\bullet f \circ \cdot a = \cdot b \circ \bar{f}, f \cdot \circ a \cdot = b \cdot \circ \bar{f}, \underline{f} \circ \hat{a} = \hat{b} \circ \bar{f}$$

Ceci peut s'exprimer en disant que le diagramme de la figure 2.8 commute. On note $T \Rightarrow \mathbf{X}$ la catégorie des T-diagrammes, (Cf. section 2.3 pour la preuve de validité dans un contexte plus général).

Définition 2.12 (source et cible de T-diagrammes) Soit t un T-diagramme. On appelle *source* de t , l'ensemble des objets $\mathbf{src}(t)$ pouvant être résultat du pushout de $\langle (\cdot \mathbf{t} : \bar{\mathbf{t}} \rightarrow \bullet t); (\hat{\mathbf{t}} : \bar{\mathbf{t}} \rightarrow \underline{t}) \rangle$. On appelle *cible* d'un T-diagramme, l'ensemble des objets $\mathbf{tgt}(a)$ pouvant être résultat du pushout de $\langle (t \cdot : \bar{\mathbf{t}} \rightarrow \bullet t); (\hat{\mathbf{t}} : \bar{\mathbf{t}} \rightarrow \underline{t}) \rangle$

En l'état $\mathbf{src}(_)$ et $\mathbf{tgt}(_)$ ne sont pas des fonctions mais des relations car une infinité d'objets sont de potentiels source et cible d'un T-diagramme donné. L'unicité de la construction universelle modulo isomorphisme permet seulement d'assurer l'unicité de la classe des objets résultats d'un pushout. Ceci est mis en valeur par la propriété suivante :

Propriété 2.3 (Unicité modulo isomorphisme) *Soit t un T-diagramme, pour tout objet x appartenant à $\mathbf{src}(t)$, cet ensemble est la classe de x modulo isomorphisme, $\mathbf{src}(t) = [x]$.*

On en déduit que $\mathbf{src}(_)$ et $\mathbf{tgt}(_)$ sont des fonctions de l'ensemble des T-diagrammes dans l'ensemble des classes d'objets. Mais ce premier pas décisif peut être étendu aux classes de T-diagrammes par la propriété suivante :

Propriété 2.4 (Isomorphisme compatible avec $\mathbf{src}(_)$ et $\mathbf{tgt}(_)$)
Deux T-diagrammes isomorphes ont même source et même cible.

$$a \sim b \implies \mathbf{src}(a) = \mathbf{src}(b) \wedge \mathbf{tgt}(a) = \mathbf{tgt}(b)$$

La démonstration est identique pour $\mathbf{src}(_)$ et $\mathbf{tgt}(_)$; on la fait donc pour $\mathbf{src}(_)$. On va montrer qu'un objet appartenant à la source d'un T-diagramme appartient aussi à la source d'un T-diagramme isomorphe, l'unicité modulo isomorphisme assurant l'égalité des classes.

Soit $(\mathbf{f} : a \rightarrow b)$ un isomorphisme de T-diagrammes, et g le résultat d'un pushout de $\langle (\langle \mathbf{a} : \bar{a} \rightarrow \bullet a \rangle; (\hat{\mathbf{a}} : \bar{a} \rightarrow \underline{a}) \rangle, \text{ montrons qu'il est aussi résultat du pushout de } \langle (\langle \mathbf{b} : \bar{b} \rightarrow \bullet b \rangle; (\hat{\mathbf{b}} : \bar{b} \rightarrow \underline{b}) \rangle. \text{ Soit } \langle (\mathbf{x} : \bullet f \rightarrow g); (\mathbf{y} : \underline{f} \rightarrow g) \rangle \text{ une complétion universelle de la partie } \mathbf{src}(_) \text{ de } a \text{ ayant } g \text{ pour cible commune. } \langle (\mathbf{x} \circ \bullet \mathbf{f}^{-1} : \bullet b \rightarrow g); (\mathbf{y} \circ \underline{f}^{-1} : \underline{b} \rightarrow g) \rangle \text{ est une complétion de la partie } \mathbf{src}(_) \text{ de } b \text{ ayant } g \text{ pour cible commune. Il nous faut montrer qu'elle est universelle.}$

Soit $\langle (\mathbf{x}' : \bullet b \rightarrow g'); (\mathbf{y}' : \underline{b} \rightarrow g') \rangle$ une complétion de la partie $\mathbf{src}(_)$ de b ayant g' pour cible commune. On applique là l'universalité de la complétion de la partie $\mathbf{src}(_)$ de a sur la paire $\langle (\mathbf{x}' \circ \bullet \mathbf{f} : \bullet a \rightarrow g'); (\mathbf{y}' \circ \underline{f} : \underline{a} \rightarrow g') \rangle$: on obtient l'existence d'un unique morphisme $\langle (\mathbf{x}' \circ \bullet \mathbf{f}; \mathbf{y}' \circ \underline{f}) : g \rightarrow g' \rangle$ satisfaisant les propriétés de commutation $\langle (\mathbf{x}' \circ \bullet \mathbf{f}; \mathbf{y}' \circ \underline{f}) \circ x = \mathbf{x}' \circ \bullet \mathbf{f}$ et $\langle (\mathbf{x}' \circ \bullet \mathbf{f}; \mathbf{y}' \circ \underline{f}) \circ y = \mathbf{y}' \circ \underline{f} \rangle$. On a les équivalences suivantes

$$\begin{aligned} u \circ x = \mathbf{x}' \circ \bullet \mathbf{f} &\iff u \circ (x \circ \bullet \mathbf{f}^{-1}) = \mathbf{x}' \\ u \circ y = \mathbf{y}' \circ \underline{f} &\iff u \circ (y \circ \underline{f}^{-1}) = \mathbf{y}' \end{aligned}$$

Ces équivalences nous permettent, dans un sens ($u = \langle \mathbf{x}' \circ \bullet \mathbf{f}; \mathbf{y}' \circ \underline{f} \rangle$) de prouver l'existence du morphisme qui commute avec la complétion de b , et dans l'autre (avec un autre u) son unicité, car s'il existe un u commutant dans b , il convient dans a . Le diagramme 2.9 résume la situation. Dans quelle mesure il se substitue complètement au texte ci dessus n'est pas suffisamment clair pour moi pour omettre la partie textuelle.

Le corollaire de cette propriété est qu'il est valide d'étendre la définition de $\mathbf{src}(_)$ et $\mathbf{tgt}(_)$ comme des fonctions de $[T \Rightarrow \mathbf{X}]$ dans $[\mathbf{X}]$.

Définition 2.13 (Transformations) *On appelle $[T \Rightarrow \mathbf{X}]$ ensemble des transformations de \mathbf{X} . On dispose de deux fonctions ($\mathbf{src}(_) : [T \Rightarrow \mathbf{X}] \rightarrow [\mathbf{X}]$) et ($\mathbf{tgt}(_) : [T \Rightarrow \mathbf{X}] \rightarrow [\mathbf{X}]$).*

Définition 2.14 (Graphe abstrait de dérivation) *Soit \mathbf{X} une catégorie co-complète, soit \mathcal{G}_X le graphe dirigé suivant :*

$$\mathcal{G}_X \stackrel{\text{def}}{=} \langle [\mathbf{X}]; [\mathbf{T} \Rightarrow \mathbf{X}]; \mathbf{src}(_); \mathbf{tgt}(_) \rangle$$

FIG. 2.9 – Diagramme de la preuve de la conservation de la source

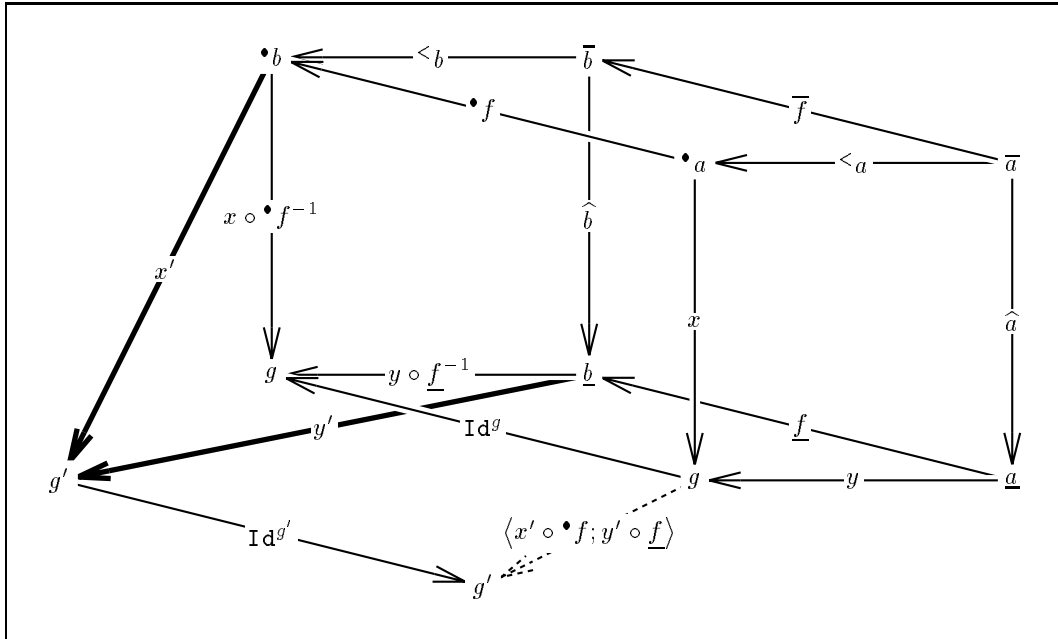
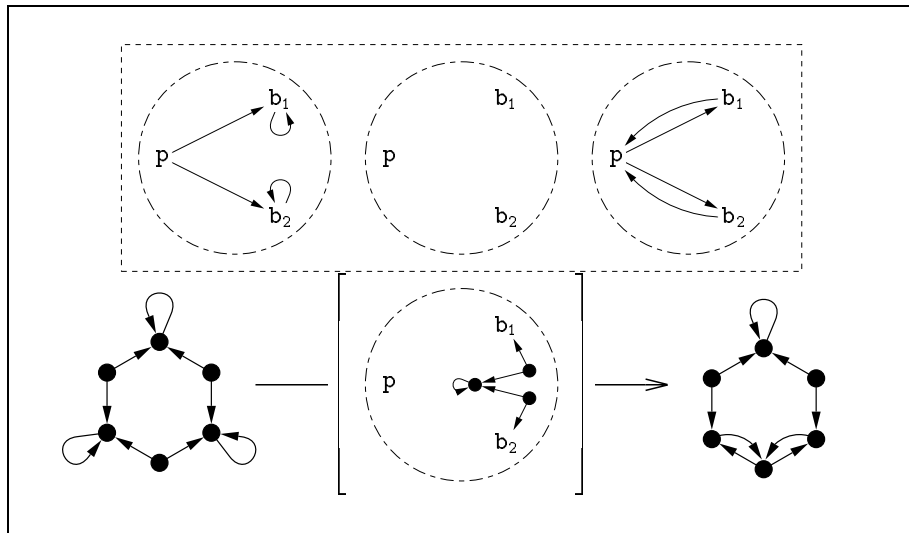


FIG. 2.10 – Une transformation: une classe de T-diagramme



Ce graphe abstrait forme le support de toutes les sémantiques particulières de systèmes qui sont défini comme des sous-graphes de ce graphe abstrait. Ceci n'est pas le cas de la sémantique originale qui permet une infinité de dérivation pour chaque arête du graphe abstrait. Nous allons donc construire une sémantique d'un système qui s'injecte dans le graphe abstrait de dérivation.

2.2.2 Sémantique d'un système

La sémantique d'un système de grammaires à double pushout est un morphisme injectif dans le graphe réflexif abstrait des dérivations. Nous en donnerons deux versions, une version abstraite qui va nous servir pour montrer les équivalences avec les réseaux de Petri dans la partie suivante et une version concrète pour les aspects opérationnels à la fin de cette partie.

Définition 2.15 (T-diagramme d'application) Soit $\langle \mathbf{X}; \mathcal{L} \rangle$ un système de règles par double pushout, Une paire formée d'une étiquette $l \in L$ et d'une flèche m de \mathbf{X} est un T-diagramme d'application ssi $\text{src}(m) = \bar{l}$. On note $\mathcal{T}\langle \mathbf{X}; \mathcal{L} \rangle$ l'ensemble des T-diagrammes d'applications d'un système.

La notion de T-diagramme d'application fixe la partie “horizontale” du T-diagramme en l'identifiant à une règle du système. La notion d'isomorphisme se réduit à la seule partie “mobile” d'un T-diagramme: le contexte d'application.

Propriété 2.5 (Isomorphisme) Deux T-diagramme d'application de la même règle $\langle l; m \rangle$ et $\langle l; m' \rangle$ sont isomorphes ssi il existe un isomorphisme $(u : m \rightarrow m')$ tel que $m' = u \circ m$. Soit $[\mathcal{T}\langle \mathbf{X}; \mathcal{L} \rangle]$ l'ensemble des classes de T-diagrammes d'applications.

Définition 2.16 (Injection canonique) Soit $I_{\langle \mathbf{X}; \mathcal{L} \rangle}(-)$ l'injection canonique qui à toute classe de T-diagramme d'application associe une classe de T-diagramme :

$$(I_{\langle \mathbf{X}; \mathcal{L} \rangle}(-) : [\mathcal{T}\langle \mathbf{X}; \mathcal{L} \rangle] \rightarrow [T \Rightarrow \mathbf{X}])$$

On dispose donc bien d'un ensemble de sommets $([\mathbf{X}])$, de flèches $([\mathcal{T}\langle \mathbf{X}; \mathcal{L} \rangle])$, d'une source $(\text{src}(-) \circ I_{\langle \mathbf{X}; \mathcal{L} \rangle}(-))$ d'une cible $(\text{tgt}(-) \circ I_{\langle \mathbf{X}; \mathcal{L} \rangle}(-))$. Cette structure est donc un graphe dirigé. La sémantique par entrelacement est la partie du graphe abstrait dont les arêtes correspondent à des classes de T-diagrammes d'applications.

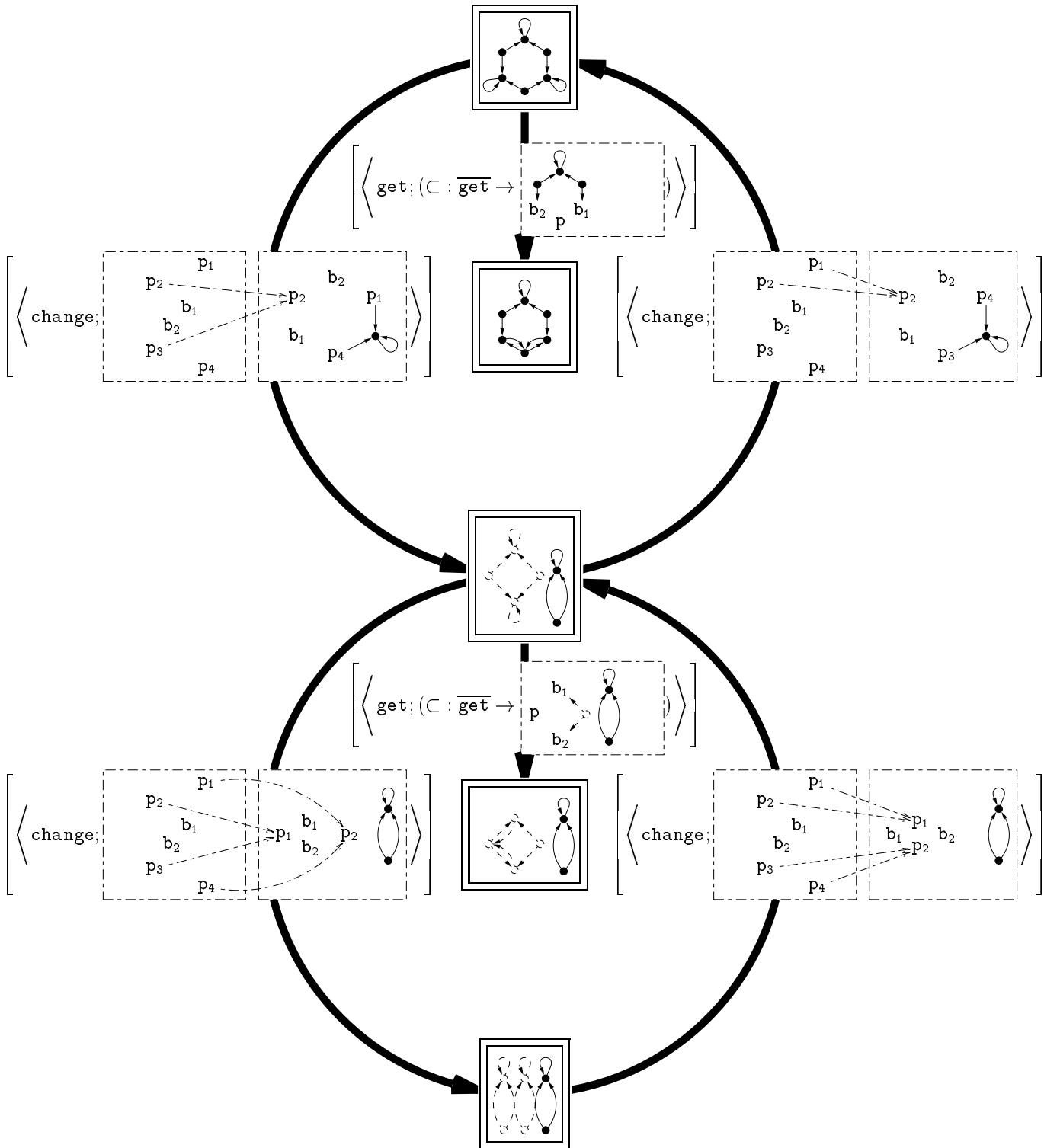
Définition 2.17 (Sémantique par entrelacement) Soit \mathbf{X} une catégorie finiment co-complète, et $\langle \mathbf{X}; \mathcal{L} \rangle$ un système de règles, soit $g(\langle \mathbf{X}; \mathcal{L} \rangle)$ la sémantique par entrelacement du système $\langle \mathbf{X}; \mathcal{L} \rangle$:

$$g(\langle \mathbf{X}; \mathcal{L} \rangle) = \langle [\mathbf{X}]; [\mathcal{T}\langle \mathbf{X}; \mathcal{L} \rangle]; \text{src}(-) \circ I_{\langle \mathbf{X}; \mathcal{L} \rangle}(-); \text{tgt}(-) \circ I_{\langle \mathbf{X}; \mathcal{L} \rangle}(-) \rangle$$

L'injection canonique qui fait de cette structure une sémantique, est donnée par l'injection $I_{\langle \mathbf{X}; \mathcal{L} \rangle}()$.

Le fait d'avoir un graphe des états accessibles est une contribution centrale de notre approche. La figure 2.2 détaille un tel graphe pour le “bal des philosophes” où trois paires de philosophes/baguettes sont impliquées. On a représenté par des pointillés l'inclusion de la dynamique où deux paires de philosophes/baguettes forment un “bal”. Remarquez que la présentation diffère pour les règles pour lesquelles le morphisme d'application est injectif de celles pour lequel il ne l'est pas. Nous avons omis les boucles sur chaque sommet: elles sont étiquetées comme le sommet associé.

FIG. 2.11 – Etiquetage univoque de la Figure 2.2



Perspectives Enfin remarquons que notre approche fournit une base de départ pour les approches à double-pullback[EHL+98], néanmoins un raffinement est nécessaire pour permettre la “loose semantic” (i.e. choisir un contexte universel pour un état et une règle donnés). Ce type de sémantique correspond à une contrainte sur l’injection (prendre un T-diagramme universel parmi ceux qui ont pour source un état donné) : ceci motive le découplage graphe abstrait/sémantique injective.

Enfin notons qu’une “loose semantic” peut être développée pour les double-pushouts. Elle correspond, lorsque $\langle l \rangle$ n’est pas injective, à un remplacement du nœud par son graphe noyau dans l’interface, toutes les arêtes contiguës à ce nœud sont dupliquées pour tous les nœuds du graphe noyau. Ceci correspond à une sémantique interprétable dans la réécriture d’ordre partiels[Fan99, DD93]. Néanmoins cette approche pour le “node-replacement” est beaucoup plus limitée que celle par double-pullback qui permet de spécifier les duplications souhaitées et leurs interfaces, l’intuition étant : le pushout est une somme paramétrée, le pullback est un produit paramétré.

Ce travail est motivé à l’origine par le chapitre III et la nécessité de développer une approche uniforme pour les trois formalismes étudiés. Néanmoins, d’avoir décapé la sémantique nous a libéré de l’appellation grammaires de graphes, pour devenir grammaires à double-somme. Ceci s’exprime dans les notations par le fait que la catégorie de travail n’est plus fixée par le formalisme mais devient un paramètre du système considéré. Que pouvons-nous faire de cette liberté ? Nous répondons à cette question en proposant une algèbre de structures, composé de deux opérateurs. Un exemple de construction réalisable par ces opérateurs, et surtout l’opérateur inédit “famille” sera développé dans la dernière section.

2.3 Une algèbre de structures

On pourrait, à la manière de la théorie classique des grammaires de graphes, choisir une bonne fois pour toute une catégorie finiment co-complète sur laquelle travailler. Ceci ne nous convient pas. En effet, qui peut nier l’utilité pour un spécifieur de pouvoir faire des partitions dans ses objets qu’il sait étanches. Cette contrainte était pour nous essentielle pour parvenir à proposer un outil “réaliste” de spécification de systèmes et permettre des développements ultérieurs.

Nous définissons deux opérateurs. Le premier, l’opérateur d’architecture, est une généralisation des graphes étendus de [Löw93] en remarquant qu’un diagramme de forme G dans **Set** est syntaxiquement une algèbre d’une signature unaire (elle-même syntaxiquement un graphe dirigé) G . Notre extension repose sur l’idée que prendre la catégorie **Set** n’est plus nécessaire en considérant des diagrammes de forme G dans une catégorie finiment co-complète **X**. Le deuxième opérateur est plus original et définit la catégorie des familles des objets d’une catégorie. L’introduction de ce deuxième opérateur a des conséquences importantes en termes de dynamisme des structures considérées. La dernière section tâchera d’illustrer les nouvelles structures envisageables dans le contexte des applications coopératives.

Cette section comporte quatre parties : la première définit l’opérateur d’architecture, la deuxième l’opérateur de famille. La troisième regroupe ces deux opérateurs dans une signature dont les termes sont évaluable en catégories.

2.3.1 L’opérateur d’architecture

Nous avons motivé dans l’état de l’art (Cf. 1.4) l’utilisation de la structure algébrique de [Löw93] par la nécessité de faire cas de la séparation entre les informations relationnelles et celles strictement locales ou comportementales. En outre, nous avons montré que toute signature unaire pouvait être interprétée comme

un graphe dirigé. Lorsque l'on considère une algèbre sur une telle signature unaire, on associe à chaque sommet un ensemble (i.e. un objet de **Set**), et à chaque flèche une fonction (i.e. une flèche de **Set**). Pour considérer le graphe comme opérant sur les catégories, il suffit de lever la contrainte sur **Set**, et remplacer par **X**. C'est exactement ce que fait un diagramme.

Un diagramme dans une catégorie **X** est un graphe dirigé étiqueté par des objets et des flèches de **X** de façon consistante. La consistance signifie que si une flèche du graphe est étiquetée par une flèche de la catégorie ayant comme domaine a et comme co-domaine b alors les extrémités de cette flèche doivent être a et b .

Définition 2.18 (Diagramme) Soit $G \in \mathbf{Graph}$ un graphe dirigé, Soit **X** une catégorie, Une paire de fonctions

$$D = \langle (D_{\text{obj}} : \text{sommets}(G) \rightarrow \text{obj}(\mathbf{X})); (D_{\text{arr}} : \text{aretes}(G) \rightarrow \text{arr}(\mathbf{X})) \rangle$$

est un diagramme de forme G dans **X** ssi $\text{dom}_X \circ D_{\text{arr}} = D_{\text{obj}} \circ \text{src}_G(_)$ et $\text{cdm}_X \circ D_{\text{arr}} = D_{\text{obj}} \circ \text{tgt}_G(_)$. On note $(D : G \rightarrow X)$ un diagramme D de forme G dans **X**.

Définition 2.19 (Morphismes de diagrammes) Une fonction

$$(\mathbf{f} : \text{sommets}_G \rightarrow \text{arr}(\mathbf{X}))$$

est un morphisme $(\mathbf{f} : a \rightarrow b)$ de diagramme de forme G dans **X** ssi

Cohérence pour tout sommet $i \in G$, $(\mathbf{f}(i) : a_{\text{obj}}(i) \rightarrow b_{\text{obj}}(i))$

Préservation pour toute arête $(\mathbf{x} : i \rightarrow j) \in G$, $f(j) \circ a_{\text{arr}}(x) = b_{\text{arr}}(x) \circ f(i)$ les deux membres de l'équation étant des flèches allant de $a_{\text{obj}}(i)$ dans $b_{\text{obj}}(j)$.

Un isomorphisme de diagrammes correspond à une fonction dont l'ensemble d'arrivée est l'ensemble des isomorphismes: pour chaque sommet, la fonction est un isomorphisme (i.e. dans **Set** une bijection).

Définition 2.20 (Catégorie $G \Rightarrow X$) Soit G un graphe réflexif, on définit la catégorie $G \Rightarrow X$ ayant comme objets les diagrammes de forme G dans **X** et comme flèches les morphismes de modèles de S dans **X**.

Identité Soit x un diagramme de forme G dans **X**: $\text{Id}^x \stackrel{\text{def}}{=} (fn\ s \Rightarrow \text{Id}^{x(s)})$

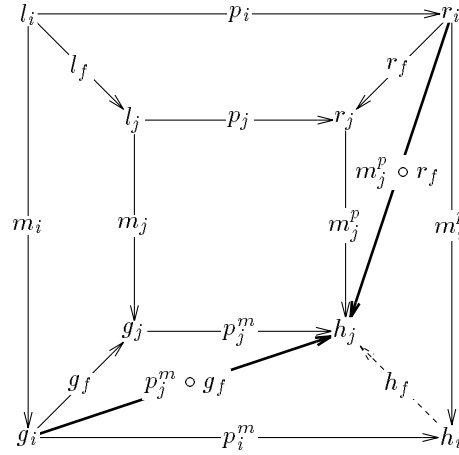
Composition Soit $(\mathbf{f} : a \rightarrow b)$ et $(\mathbf{g} : b \rightarrow c)$ deux morphismes de diagrammes de forme G dans **X** : $g \circ f \stackrel{\text{def}}{=} (fn\ s \Rightarrow g(s) \circ f(s))$

Ici, ces constructions s'appuient sur la catégorie sous-jacente. Il faut vérifier la loi de l'unité et l'associativité, nous le faisons explicitement, la façon de poser nos définitions est essentielle pour mener à bien les démonstrations suivantes.

Loi de l'unité Soit $(\mathbf{f} : a \rightarrow b)$ un morphisme de modèles de S dans **X**,

$$\begin{aligned} f \circ \text{Id}^a &= (fn\ s \Rightarrow f(s) \circ (fn\ s \Rightarrow \text{Id}^{a(s)}(s))) \\ &= (fn\ s \Rightarrow f(s) \circ \text{Id}^{a(s)}) \\ &= (fn\ s \Rightarrow f(s)) \\ &= f \end{aligned}$$

La démonstration est strictement identique pour le cas où l'identité est composée après f .

FIG. 2.12 – Construction du pushout dans la catégorie $G \Rightarrow X$.

Associativité Soit $(\mathbf{f} : a \rightarrow b)$, $(\mathbf{g} : b \rightarrow c)$ et $(\mathbf{h} : c \rightarrow d)$ trois morphismes de modèles de S dans \mathbf{X} :

$$\begin{aligned}
 f \circ (g \circ h) &= (fn \ s \Rightarrow f(s) \circ (g \circ h)(s)) \\
 &= (fn \ s \Rightarrow f(s) \circ (fn \ t \Rightarrow g(t) \circ h(t))(s)) \\
 &= (fn \ s \Rightarrow f(s) \circ (g(s) \circ h(s))) \\
 &= (fn \ s \Rightarrow (fn \ t \Rightarrow f(t) \circ g(t))(s) \circ h(s)) \\
 &= (fn \ s \Rightarrow (f \circ g)(s) \circ h(s)) \\
 &= (f \circ g) \circ h
 \end{aligned}$$

La co-complétude finie de cette catégorie peut se démontrer en prenant comme hypothèse que la catégorie de départ est finiment co-complète. L'idée de la démonstration pour la co-complétude finie réside dans la possibilité de réaliser une construction des co-limites élémentaires. Pour les deux constructions élémentaires, la construction se fait d'abord en considérant la construction de la catégorie de départ pour chaque sommet, puis pour les arêtes on utilise la propriété universelle de cette construction pour la source, qui donne l'opérateur voulu. La figure 2.12 détaille cette partie de la construction pour le pushout. Il s'agit de montrer que l'on peut déduire de façon unique les opérations à partir des ensembles porteurs. Pour toute flèche $(\mathbf{f} : i \rightarrow j)$ du graphe G , cette construction donne le diagramme donné figure 2.12 où :

1. la commutation des trapèzes supérieur et gauche sont des conséquences des morphismes de diagrammes impliqués;
2. les carrés intérieur et extérieur sont des pushouts par construction, d'où on déduit leur commutation et la propriété universelle.

La complétion par la propriété du pushout extérieur donne la construction souhaitée. La commutation s'obtient en passant par le pushout intérieur.

Ceci nous permet d'affirmer la propriété suivante:

Propriété 2.6 (Propagation de la co-complétude finie) Si \mathbf{X} est finiment co-complète alors pour tout graphe réflexif $G \Rightarrow \mathbf{X}$ est finiment co-complète.

D'avoir défini un opérateur qui à tout graphe dirigé et à toute catégorie finiment co-complète associe une catégorie finiment co-complète va nous permettre d'empiler des graphes définissant des signatures pour enrichir notre architecture.

Ceci est aussi un corollaire d'une propriété plus vaste démontrée dans [BW90] qui stipule que cette propriété est vraie même si on adjoint des équations, permettant de définir les graphes symétriques (chaque arête à une arête symétrique) ou les graphes réflexifs (chaque nœud à une boucle).

2.3.2 L'opérateur famille

Cet opérateur est plus original: une famille est une paire formée d'un ensemble et d'une fonction de cet ensemble vers les objets de la catégorie. Cette construction est revenue tellement souvent que nous avons eu le besoin de la formaliser. Sans doute une construction doit exister en uniquement avec des opérateurs catégorique tellement l'utilisation d'index est répandue. Ici, nous utilisons la théorie des ensembles comme moteur pour construire une catégorie et vérifier ces caractéristiques comme pour valider une interface. Une construction uniquement catégorique aurait pour avantage de fournir toutes les propriétés souhaitées dès la construction.

Définition 2.21 (Famille) *Soit E une collection, une famille \mathcal{A} sur E est une paire $\langle A; \mathbf{a}(_) \rangle$ formée d'un ensemble A et d'une fonction $(\mathbf{a}(_) : A \rightarrow E)$. On note $\text{Family}(\mathbf{X})$ l'ensemble des familles des objets de \mathbf{X} .*

Un morphisme de familles est d'abord la donnée de deux familles qui vont être source et cible du morphisme. Ensuite une fonction $\mathbf{f}(_)$ qui relie les ensembles d'indexage et enfin une famille de flèches $\widehat{\mathbf{f}}(_)$ de \mathbf{X} indexée par l'ensemble source. Cette dernière vérifie une propriété de cohérence : que la source de la flèche d'index x est bien l'objet indicé par x , et que la cible de la flèche d'index x est indexée dans la famille cible par $\mathbf{f}(x)$.

Définition 2.22 (Catégorie famille) *Soit \mathbf{X} une catégorie de base, on définit la catégorie $\text{Family}(\mathbf{X})$ ayant pour:*

objet *la collection des objets est la collection des familles d'objets de \mathbf{X} :*
 $\text{Family}(\text{obj}(\mathbf{X}))$.

flèche *Une flèche de cette catégorie est un quadruplet formé de deux familles d'objets de \mathbf{X} une fonction entre les index et une famille de flèches de \mathbf{X}*

$$\langle \mathcal{A}; \mathcal{B}; (\mathbf{f}(_) : A \rightarrow B); \langle A; \widehat{\mathbf{f}}(_) \rangle \rangle \in \text{Family}(\text{obj}(\mathbf{X}))^2 \times \text{arr}(\text{Set}) \times \text{Family}(\text{arr}(\mathbf{X}))$$

$$\text{vérifiant } \forall x \in A, \text{dom}(\widehat{\mathbf{a}}(x)) = \mathbf{a}(x) \wedge \text{cdm}(\widehat{\mathbf{a}}(x)) = \mathbf{b}(\mathbf{f}(x))$$

$$\text{Identité } \text{Id}^A = \langle \mathcal{A}; \mathcal{A}; \text{Id}^A; \langle A; (fn\ x \Rightarrow \text{Id}^{\mathbf{a}(x)}) \rangle \rangle$$

Composition

$$\begin{aligned} & \langle \mathcal{B}; \mathcal{C}; (\mathbf{g}(_) : B \rightarrow C); \langle B; \widehat{\mathbf{g}}(_) \rangle \rangle \circ \langle \mathcal{A}; \mathcal{B}; (\mathbf{f}(_) : A \rightarrow B); \langle A; \widehat{\mathbf{f}}(_) \rangle \rangle \\ &= \langle \mathcal{A}; \mathcal{C}; (\mathbf{g}(_) \circ \mathbf{f}(_) : A \rightarrow C); \langle A; (fn\ x \Rightarrow \widehat{\mathbf{b}}(\mathbf{f}(x)) \circ \widehat{\mathbf{a}}(x)) \rangle \rangle \end{aligned}$$

La loi de l'unité et l'associativité ne posent pas de difficultés mécaniques particulières. Nous considérerons comme suffisant de donner l'équation suivante de "propagation de la composition":

$$\forall x \in A, \text{dom}(\widehat{\mathbf{b}}(\mathbf{f}(x))) = \text{cdm}(\widehat{\mathbf{a}}(x)) = \mathbf{b}(\mathbf{f}(x))$$

Remarque: **Family**(**1**) est une catégorie équivalente à **Set**, où **1** est la catégorie réduite à un objet et une flèche, l'identité de l'objet. Représenter un graphe dirigé par la catégorie $\text{dir} \Rightarrow \mathbf{Family}(\mathbf{1})$ correspond très fidèlement aux diagrammes de Venn utilisés comme support intuitif de notre état de l'art (Cf 1.4).

Ici nous ne pouvons pas nous appuyer sur une construction classique pour montrer l'exportation de la co-complétude finie. Nous remarquons d'abord que la famille vide est toujours l'élément initial de la catégorie des familles sur **X**. Ensuite, pour avoir la co-complétude finie, nous montrons l'existence du co-produit et d'un co-égaliseur [RB88].

Le co-produit ne pose pas de difficulté, c'est la juxtaposition disjointe des index; les fonctions suivent. Avec un léger abus de notation pour pouvoir utiliser le \coprod sur les fonctions qui associent l'indice à l'objet on peut écrire:

$$\coprod_{\langle A; \mathbf{a}(-) \rangle}^{\langle B; \mathbf{b}(-) \rangle} = \left\langle \coprod_B^A; \coprod_{\mathbf{b}(-)}^{\mathbf{a}(-)} \right\rangle$$

Le co-égaliseur lui va se propager sur la catégorie de départ. On commence par réaliser un co-égaliseur sur les index, puis pour les index qui ont été "égalisés", on co-égalise les éléments indexés. Ceci nous permet d'affirmer la propriété suivante:

Propriété 2.7 (Propagation de la co-complétude finie) *Si **X** est finiment co-complète alors **Family**(**X**) est finiment co-complète.*

Avant d'aller plus loin et de définir notre algèbre de structures, nous allons expliciter l'écriture fonctionnelle des diagrammes de **Family**(**X**). En effet comme pour la représentation fonctionnelle des flèches de **Set**, certaines simplifications sont utiles pour rendre ces diagrammes lisibles. En effet, la définition d'une flèche est redondante par rapport à la notation diagrammatique. D'ores et déjà, la notification explicite de la source et la cible est indiquée par la source et la cible dans le diagramme sans avoir à l'indiquer dans la flèche. Ensuite, au lieu de donner les deux fonctions d'index et de flèches de **X**, nous donnerons une suite de triplets composés de l'indice de départ, la flèche de **X** et l'indice d'arrivée. Lorsque l'on représentera des flèches de **Family**(**Set**), on ne donnera comme représentation de la flèche que l'expression fonctionnelle, la source et la cible étant déjà données par les indices. Enfin dans **Family**(**Set**) on compactera l'écriture des familles d'ensembles comme ci-dessous:

$$\langle \{a, b, c\}; (\text{fn } a \Rightarrow \{x, y\} | any \Rightarrow \{x\}) \rangle \equiv \langle \{x, y\}_a, \{x\}_b, \{x\}_c \rangle$$

En outre, nous noterons "!" la flèche partant de l'élément initial: la famille vide ($\emptyset = \langle \emptyset; \emptyset \rangle$). Notez la différence essentielle entre \emptyset (i.e. la famille vide) et $\langle \emptyset_a \rangle$ (i.e. la famille à un élément, vide). Nous étendons la notion d'inclusion (\subset) à la flèche pour laquelle la fonction d'index est l'inclusion et qui associe comme flèche l'inclusion pour tout indice de l'index de la source.

La dernière section développe les constructions à base de familles, les constructions illustratives sont donc à chercher dans la section 2.4.2.

2.3.3 Algèbre et première application

Nous allons maintenant définir notre algèbre de structures et nous en servir comme première application à la définition des éléments de notre sémantique. Ceci commence à devenir une question de commodité opérationnelle. En effet si l'ensemble de la sémantique peut se décrire simplement comme des fonctions entre

des objets structurés comme ceux sur lesquels on opère, cela fait toujours une classe d'objets en moins à définir.

Définition 2.23 (Signature de structure) *Soit la signature (Cf 1.3) suivante:*

Sortes : {Category, Graphs}

Opérateurs $\left\{ \begin{array}{l} _ \Rightarrow _ : \text{Graphs} \times \text{Category} \rightarrow \text{Category} \\ \text{Family}(_) : \text{Category} \rightarrow \text{Category} \end{array} \right.$

L'algèbre d'évaluation de cette signature prend $\text{obj}(\text{Cat})$ (i.e. les catégories) comme ensemble porteur de la sorte **Category** et $\text{obj}(\text{Graphs})$ (i.e. les graphes dirigés) comme ensemble porteur de la sorte **Graphs**, les opérations étant définies dans ce qui précède.

La propriété suivante est montrée par induction sur la structure du terme :

Propriété 2.8 (Propagation de la co-complétude finie) *Tout terme construit à partir d'une catégorie finiment co-complète s'évalue en une catégorie finiment co-complète.*

C'est un corollaire des propriétés correspondantes montrées pour les deux opérateurs introduits précédemment.

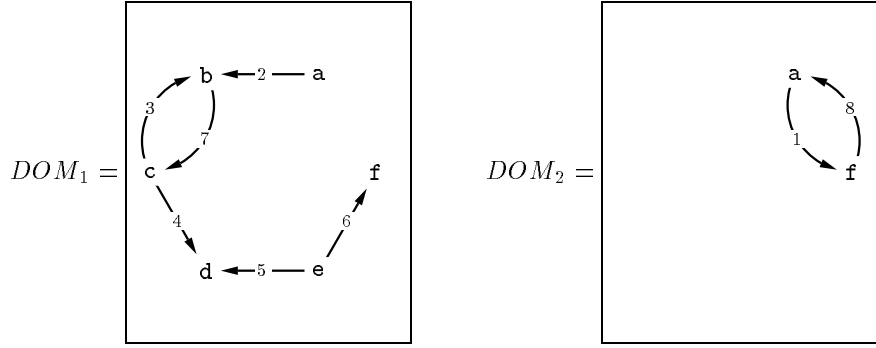
La première application annoncée est de décrire les éléments de notre sémantique à l'aide de termes de notre algèbre de structures. Soit **X** une catégorie finiment co-complète.

Un système est un objet de $\text{Family}(\text{Span} \Rightarrow \mathbf{X})$, Un T-diagramme est un objet de $\mathbf{T} \Rightarrow X$, où **T** est donné dans la figure 2.7 et **Span** est donnée dans la figure 2.6. La notion de morphisme de T-diagramme coïncide exactement; par contre, cela nous permet d'introduire la notion de morphisme de système de règle et de systèmes équivalents. Deux systèmes sont équivalents ssi ils sont isomorphes dans $\text{Family}(\text{Span} \Rightarrow \mathbf{X})$ et deux systèmes isomorphes ont deux sémantiques isomorphes. Cela est dû au fait que tous les T-diagrammes d'instanciation s'injectent sur des T-diagrammes isomorphes, en prenant l'identité pour le contexte. Ceci justifie *a posteriori* notre notation pour les morphismes et par extension pour les systèmes de règles. En effet, on peut omettre de nommer les éléments ajoutés et détruits puisque, à un nommage près, tout système de règle est équivalent. Par contre, l'étiquetage des éléments conservés permet de compacter la définition et d'éviter une surcharge de flèches.

Nous allons maintenant développer une application de notre théorie. Elle concerne la spécification d'apartés [DV94] dans des domaines coopératifs [VVG95]. La construction s'appuie sur l'équivalence entre $\text{Family}(\mathbf{1})$ et **Set**. Si on se donne deux familles de points **V** et **E** et deux morphismes de familles de **E** vers **V**, la construction est "syntaxiquement" le diagramme de Venn d'un graphe dirigé. Nos définitions permettent de substituer **Set** à **1**, et donc de considérer $\text{Family}(\text{Set})$. C'est l'exploration de cette possibilité qui est sous jacente à la partie suivante.

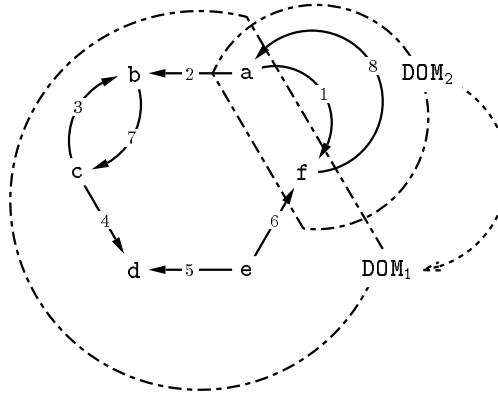
2.4 Distribution et Apartés

Dans [DV94, VVG95] un *domaine de coopération* est un graphe dont les arêtes spécifient que la source peut accéder en lecture à des contenus de la cible. Par exemple considérons les domaines de coopération DOM_1 et DOM_2 :



Nous pouvons considérer que DOM_2 soit un aparté dans DOM_1 . Un *aparté* est défini [DV94, Vil95] comme un sous-domaine privatif et temporaire dans un domaine plus vaste. Nous écrivons $(DOM_2 \rightarrow DOM_1)$ pour indiquer que $(DOM_2$ est un aparté dans $DOM_1)$. On peut coupler les deux informations dans la représentation graphique donnée figure 2.13.

FIG. 2.13 – DOM_2 est un aparté dans DOM_1



D'un point de vue pragmatique et comme nous l'avons représenté en termes de graphes, il doit exister une façon graphique de traiter ce problème. Cette représentation achoppe sur la notion de groupe. Comment représenter un groupe? Graphiquement la première idée consiste à considérer une sorte groupe, une sorte élément et à tracer la fonction d'appartenance. On travaille donc avec une signature dite "flèche":

$$(\text{appartient} : \text{Element} \rightarrow \text{Groupe}) \in \text{Set}$$

L'appartenance sera exclusive à un groupe; pour permettre l'appartenance à plusieurs groupes, on prend une signature de type "span":

$$(\text{est} : \text{appartient} \rightarrow \text{Element})(\text{dans} : \text{appartient} \rightarrow \text{Groupe}) \in \text{Set}$$

Cette signature est celle d'un graphe bipartite, les flèches de ce graphe (de sorte "appartient") ont une source ("est") qui est un "élément" et une cible ("dans") qui est un "groupe". Un élément peut effectivement être dans deux groupes, mais peut aussi être deux fois dans le même groupe, il suffit que deux "appartient" mènent de cet "élément" à un "groupe" donné. La garantie, donnée par la condition de liaison, qu'un "élément" quitte le "groupe" repose sur le départ du système soit du "groupe" soit de l'"élément". Cette garantie est insuffisante pour pouvoir décrire qu'un élément quitte un groupe, car les deux subsistent à ce départ.

L'approche que nous suivons est tout autre, partant du point de départ donné dans la théorie classique, les *graphes distribués* [CMR⁺97], elle développe une technique basée sur l'opérateur famille permettant de considérer des *graphes dynamiques* dont chaque nœud et chaque arête sont des graphes, la source et la cible devenant des morphismes de graphes. Notons que nous ne développons ici qu'un exemple particulier de structure rendu possible par nos opérateurs et qu'en aucun cas nous ne fixons dans le formalisme la structure étudiée.

2.4.1 Des graphes de graphes...

L'approche des grammaires par pushout permet d'envisager ce diagramme sous une forme simplifiée qui est une représentation univoque de la précédente. En effet, si nous considérons les nœuds a et f comme l'intersection non plus subie mais spécifiée des domaines de coopération, alors la situation précédente s'exprime comme deux morphismes de graphes ayant même source: le graphe composé des sommets a et f . La figure 2.13 s'écrit donc

$$DOM_2 \xleftarrow{\text{source}} \boxed{a \quad f} \xrightarrow{\text{cible}} DOM_1$$

Dans ce cas, les deux morphismes (i.e. paires de fonctions Cf. 1.4) sont des inclusions sans renommage (notée \subset Cf. 2.1), nous avons:

$$\subset = \text{source} = \text{cible} = \langle (\text{fn } x \Rightarrow x); (\text{fn } x \Rightarrow x) \rangle$$

Premier point, pour retrouver le graphe complet, il suffit de considérer le pushout de ce diagramme, qui réalisera l'union paramétrée par l'interface, DOM_0 . Dans la théorie classique des grammaires de graphes par double pushout, nous pouvons trouver la définition suivante qui permet de décrire notre système:

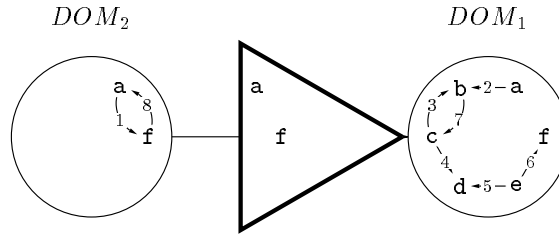
Définition 2.24 (Graphe distribué [CMR⁺97]) *Un graphe distribué consiste en deux graphes locaux, un graphe d'interface, et de deux morphismes de graphes permettant de plonger (i.e. to embed) le graphe d'interface dans les graphes locaux.*

$$DG = \left\langle G_1 \xleftarrow{g_1} G_0 \xrightarrow{g_2} G_2 \right\rangle$$

Une transformation de graphe locale se définit uniquement sur un des graphes locaux, une transformation distribuée va être un triplet de règles relié par des morphismes de règles.

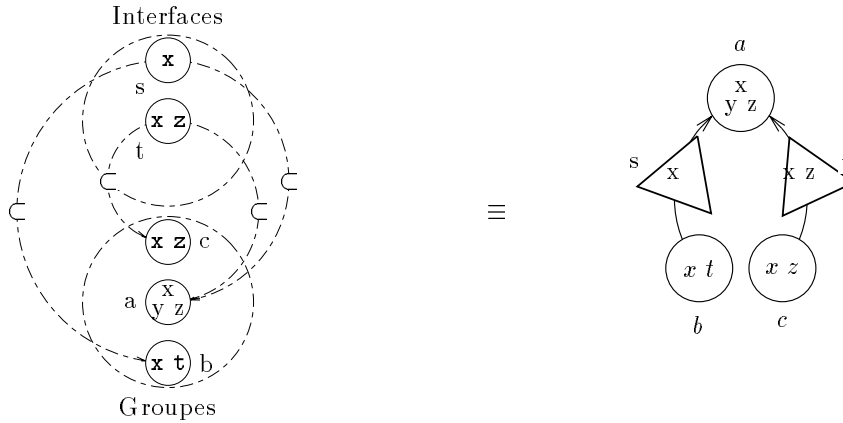
De notre point de vue, la distribution consiste en un changement de la catégorie de travail et à prendre comme catégorie de travail une catégorie de la forme $DG \Rightarrow \text{dir} \Rightarrow \mathbf{Set}$, i.e. considérer des diagrammes de forme “ DG ” dans une catégorie de diagramme de forme “ dir ”. En fait en prenant cette catégorie, on réalise la construction de [Tae96]: des architectures statiques dont les composants sont des graphes. Pour revenir à notre problème, chaque individu peut avoir différentes identités dans les différents groupes, la cohérence est assurée par les portes qui maintiennent la connaissance de “qui est qui” entre deux salles. Si nous considérons l'inclusion par défaut, cette connaissance (“qui est qui”) est assurée par le nommage. Dans ce cas, on peut omettre d'étiqueter les flèches entre l'interface et les graphes source et cible, en posant l'inclusion par défaut. En donnant au graphe interface un symbole orienté (un triangle), la notation suivante décrit de façon univoque le système considéré en permettant d'identifier la source et la cible, pointée par le triangle. Cette représentation est donnée dans la figure 2.14. Elle met en valeur l'idée que nous manipulons là un “graphe de graphe”.

FIG. 2.14 – Représentation par graphe de graphe d’une aparté



Cette notation est beaucoup plus compacte que la figure 2.13. Dans ce “graphe de graphe”, chaque “rond” est un graphe, chaque “triangle” est un graphe. La flèche reliant la pointe du triangle à un rond est le morphisme cible, et le trait reliant la base du triangle à un rond est le morphisme source. Sans indication spécifique, comme ici, ce morphisme est une inclusion préservant les noms. On peut retrouver le graphe global en calculant le pushout DOM_0 du diagramme. Ceci n’a pas vraiment d’intérêt car on perdrait dans cette opération de l’information: l’architecture des apartés disparaîtrait.

FIG. 2.15 – Une architecture d’apartés sous trois formes équivalentes (Venn, Abstrait | Texte)



$$\langle \langle \{x, y, z\}_a, \{x, t\}_b, \{x, z\}_c \rangle; \langle \{x\}_s, \{x, z\}_t \rangle; (s, \subset, b)(t, \subset, c); (s, \subset, a)(t, \subset, a) \rangle$$

On peut aussi considérer deux apartés en même temps comme dans la figure 2.15, voire des arborescences ou des graphes d’apartés. Notons au passage que ceci n’est plus un graphe distribué au sens classique, et que pour chaque architecture il faudrait donc considérer une nouvelle signature d’architecture. Pourtant la figure 2.15 et le premier aparté sont tous les deux des objets d’une catégorie, et cette

catégorie s'appelle:

$$Apartes \stackrel{\text{def}}{=} \text{dir} \Rightarrow \mathbf{Family}(\text{dir} \Rightarrow \mathbf{Set})$$

Ce terme est complexe, nous allons donc considérer d'abord une instance simplifiée du problème pour appréhender comment l'opérateur famille réalise une architecture dynamique.

2.4.2 Architecture dynamique de groupes non-disjoints

La simplification à laquelle nous allons opérer consiste à enlever une structure de graphe. Comme nous avons montré que la donnée d'un graphe interface pouvait être ramenée à une sorte de "super-arête", une structure de graphe apparaît au "top-level"; en conséquence la structure de graphe qui provenait du domaine peut être enlevée. Nous nous intéressons donc au problème de la formation d'apartés, mais sans spécifier les liens de communication entre les éléments. Donc nous cherchons à exprimer la dynamique de groupes ayant des éléments communs, d'où notre appellation: architecture dynamique de groupe non-disjoints. Dans la sous-section suivante, nous reviendrons sur la structure d'apartés comme première étape d'une généralisation.

Groupe Considérons d'abord la catégorie **Set**, point de départ de nos constructions. Un objet de **Set** est un ensemble. Nous l'interprétons ici comme un ensemble d'identités formant un groupe, comme par exemple

$$\{a, b, c\}$$

Si nous considérons le système composé de la seule règle $\text{succ} = \emptyset \xleftarrow{\supset} \emptyset \xrightarrow{\subseteq} \{x\}$, alors sa dynamique abstraite associée est:

$$[\emptyset] \rightarrow [\{a\}] \rightarrow [\{a, b\}] \rightarrow [\{a, b, c\}] \rightarrow [\{a, b, c, d\}] \rightarrow \dots$$

En utilisant notre notation d'oubli de référence (\bullet) nous obtenons:

$$[\emptyset] \rightarrow \{\bullet\} \rightarrow \{\bullet, \bullet\} \rightarrow \{\bullet, \bullet, \bullet\} \rightarrow \{\bullet, \bullet, \bullet, \bullet\} \rightarrow \dots$$

Qui n'est autre que la fonction successeur, en réalisant dans le bon sens l'abstraction fondamentale de l'arithmétique:

$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow \dots$$

En effet, rajouter un élément dans un ensemble où on oublie les identités revient à ajouter 1 au cardinal de cet ensemble.

Groupes disjoints Ici au lieu de prendre l'opérateur d'architecture qui nous mène vers des graphes au sens de [Löw93], nous prenons l'opérateur de famille. Nous considérons donc la catégorie

$$\mathbf{Family}(\mathbf{Set})$$

Soit un objet de cette catégorie: la famille suivante indexée par $\{x, y\}$ conformément aux notations introduites dans la section précédente (Cf. 2.3):

$$\langle \{a, b, c\}_x, \{a, b, c\}_y \rangle$$

Nous considérons ceci comme deux groupes disjoints, en effet le fait qu'ils portent les mêmes noms dans deux ensembles n'a, dans le contexte, aucune importance.

Ceci peut se comprendre si on se rappelle que, lors de la construction de la dynamique d'un tel système, nous considérerons les classes d'équivalences modulo isomorphisme, ici un multi-ensemble d'entiers:

$$[\langle \{a, b, c\}_x, \{a, b, c\}_y \rangle] \equiv \langle \{\bullet, \bullet, \bullet\}_\bullet, \{\bullet, \bullet, \bullet\}_\bullet \rangle \equiv 3 \oplus 3$$

Le fait que nous leur ayons donné les même identités à gauche apparaît bien dérisoire à droite où les identités ont disparues. Si nous considérons le système composé de la seule règle $new = \emptyset \xleftarrow{\quad} \emptyset \xrightarrow{\quad} \langle \emptyset_x \rangle$, sa dynamique abstraite associée est l'ajout d'un nouvel ensemble dans la famille à chaque itération:

$$[\emptyset] \rightarrow [\langle \emptyset_a \rangle] \rightarrow [\langle \emptyset_a, \emptyset_b \rangle] \rightarrow [\langle \emptyset_a, \emptyset_b, \emptyset_c \rangle] \rightarrow \dots$$

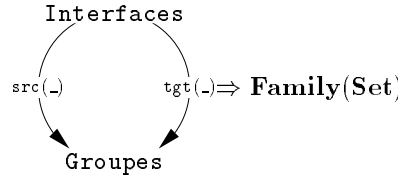
Le résultat est profondément différent de la situation antérieure. On peut aussi trouver un équivalent numérique aux classes d'équivalence, il s'agit d'un multi-ensemble d'entiers. A présent, rajoutons la règle add ,

$$add = \langle \emptyset_x \rangle \xleftarrow{\quad} \langle \emptyset_x \rangle \xrightarrow{\quad} \langle \{a\}_x \rangle$$

Le système formé par les règles new et add , permet soit d'ajouter un nouvel ensemble à la famille (new), soit d'ajouter un nouvel élément (add) dans un élément de la famille d'ensemble.

$$[\emptyset] \rightarrow [\langle \emptyset_a \rangle] \rightarrow [\langle \emptyset_a, \emptyset_b \rangle] \rightarrow [\langle \{c\}_a, \emptyset_b \rangle] \rightarrow [\langle \{c\}_a, \{d\}_b \rangle] \rightarrow [\langle \{c, d\}_a, \{d\}_b \rangle] \dots$$

Groupes non disjoints Comment permettre à nos groupes de partager des éléments? Il faut pour cela exprimer des interfaces et des fonctions de ces interfaces vers les groupes. Donc, une famille de groupes et une famille d'interfaces et pour chaque élément de la famille des interfaces une paire de morphismes vers deux éléments de la famille des groupes permettent de rendre compte de cette structure. C'est exactement un objet de la catégorie suivante:



Pour nous en convaincre, considérons le diagramme de Venn donnée dans la figure 2.15, avec sa notation compacte à droite, qui est une représentation graphique de la formule donnée en bas. Les autres figures de cette partie respectent cette disposition, l'intuition graphique en haut et la forme univoque en bas.

Maintenant, comment transformer ces structures? Nous allons répondre en donnant un système pour lequel la classe de l'objet ci-dessus est accessible depuis un groupe unique à quatre éléments.

Système de groupes non-disjoints Passer une porte va nécessiter deux règles données Figure 2.17: la première consiste à s'engager sur le seuil (i.e. entrer dans l'interface), la deuxième à la passer. Pour réaliser le parcours dans l'autre sens on utilisera les deux règles symétriques.

Graphiquement, ces règles pourraient s'interpréter comme des règles de $Span \Rightarrow \mathbf{Set}$, un système doté d'une architecture à deux salles. Ceci pour souligner qu'elles ne mettent pas en jeu le dynamisme de l'architecture (Remarquons aussi que $Span \Rightarrow \mathbf{Set}$ est une signature de multi-relation). Pour cela il nous faut considérer

FIG. 2.16 – Règle de création d'un sas

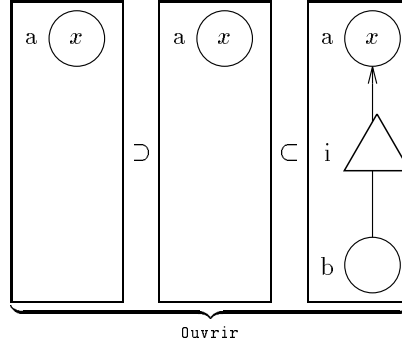
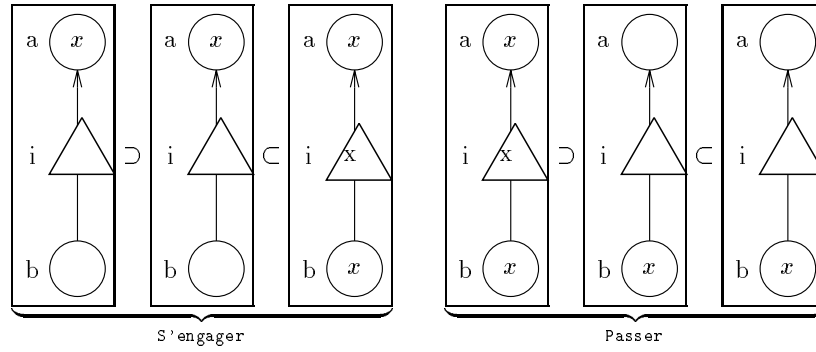


FIG. 2.17 – Règles de franchissement d'un sas



la règle *open* qui permet de créer un nouvel aparté que nous posons vide au départ; par contre nous imposons la présence d'un élément dans la salle de départ. Cette règle est donnée dans la Figure 2.16 sous deux formes “intuitive” et “textuelle”.

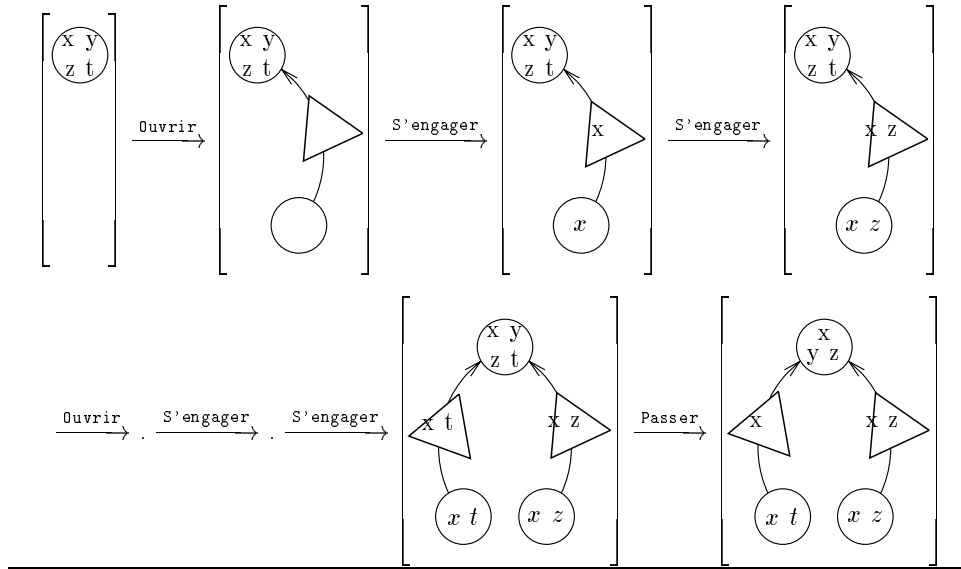
Muni du système composé des règles:

$$\{\text{Ouvrir}, \overline{\text{Ouvrir}}, \text{S'engager}, \overline{\text{S'engager}}, \text{Passer}, \overline{\text{Passer}}\}$$

Nous en donnons une dynamique étiquetée dans la Figure 2.18. Comme chaque règle contient sa symétrique, le parcours peut être réalisé à rebours. Notez que, comme nous représentons des classes d'équivalence, les noms des sommets d'architecture sont omis; par contre à l'intérieur des sommets, ils sont conservés car ils servent à décrire par le nommage les injections canoniques omises sur les traits.

Des signatures dynamiques Une interprétation possible consiste à utiliser cette construction comme possibilité de réaliser des signatures dynamiques. En effet, le graphe détaillé figure 2.15 peut être traduit de façon univoque dans un élément de $b \xrightarrow{s} a \xleftarrow{t} c \Rightarrow \mathbf{Set}$ et le graphe initial dans un objet de $a \Rightarrow \mathbf{Set}$ où a est un graphe réduit à un sommet seul. La signature (i.e. la forme du diagramme) a évolué dans le temps... En contrepartie, puisque l'on travaille à un isomorphisme près on perd l'identification des sortes. Mais on pourrait imaginer repérer des “types” de sortes par invariant comme les baguettes et les fourchettes dans les graphes dirigés. En outre comme rien n'empêche de relier les “arêtes” entre elles (et ces liens d'arêtes entre eux ...) on dispose d'une possibilité de repérage topologique assez riche pour pouvoir spécifier.

FIG. 2.18 – Déroutement menant d'un groupe simple à une structure d'apartés



Généraliser les structures échangées On peut échanger ainsi n'importe quelle structure à travers les interfaces, les éléments d'un ensemble étant la structure élémentaire. Si on partage de cette façon un graphe dirigé, il faudra pour pouvoir passer une arête que la source et la cible soit dans l'interface. On peut complexifier ainsi le structure échangée en considérant le passage de structure par ses éléments constitutants (une règle par sorte).

Une autre perspective intéressante est d'"asymétriser" les échanges selon les types d'arêtes ainsi on aurait une possibilité de représenter des messages dans un réseau. Dans cette optique, les ensembles de structures représenteraient des localités réseau et les interfaces des médias. Cette interprétation conduit à une remarque intéressante: nous disposons d'un modèle des domaines coopératifs.

2.4.3 Spécifier un domaine coopératif

Conservons le même système mais changeons l'interprétation. Les ensembles sont des coopérants, et les éléments des ensembles sont des contenus. Nous rajoutons les deux règles suivantes pour donner un sens aux notions de lecture et d'écriture:

$$\begin{aligned} lire &= \langle \{x\}_a \rangle \xleftarrow{\mathcal{D}} \langle \{x\}_a \rangle \xrightarrow{\mathcal{C}} \langle \{x\}_a \rangle \\ écrire &= \langle \{x\}_a \rangle \xleftarrow{\mathcal{D}} \langle \emptyset_a \rangle \xrightarrow{\mathcal{C}} \langle \{x\}_a \rangle \end{aligned}$$

La différence entre les deux est l'interface, "*lire*" laisse l'élément de l'ensemble considéré dans le contexte, alors qu'"*écrire*" le prélève pour le remettre. En conséquence, un élément cité dans une interface ne peut plus qu'être lu par les deux parties en vertu de la condition de liaison. On modélise l'accès synchrone à une variable en lecture uniquement. La situation modélisée est donc la suivante:

Les coopérants détiennent des contenus, ils peuvent partager en lecture seulement des contenus avec leurs voisins. Cette possibilité de lecture peut se transmettre sur une longueur non bornée. Des coopérants peuvent apparaître et disparaître dans ce système.

La règle permettant de faire apparaître et disparaître des contenus est laissée à la discrétion du lecteur. Cette stratégie est assez frustrante, mais elle correspond à un

système minimal de règles dont la première interprétation s’appliquait au niveau de l’architecture d’apartés. En changeant l’interprétation, on obtient un modèle très proche de [VVG95, DV94] et dont la différence pourrait être comblée en utilisant un système de règles qui lui soit dédié plus spécifiquement. On pourrait donc donner le terme correspondant à des apartés dans des domaines coopératifs avec la catégorie des “graphes de graphes” :

$$\mathit{GraphOfGraphs} \stackrel{\text{def}}{=} \mathit{dir} \Rightarrow \mathbf{Family}(\mathit{dir} \Rightarrow \mathbf{Family}(\mathbf{Set}))$$

Nous remarquons que cette catégorie s’obtient par une alternance d’un opérateur d’architecture et d’un opérateur de famille; ceci montre par l’exemple la pertinence de leur association dans une structure algébrique.

Néanmoins, nous arrivons ici à une limitation de notre approche actuelle, la définition des systèmes est “moins” compositionnelle que celle des catégories, et donc définir un système sur cette catégorie imposerait de l’écrire de bout en bout alors qu’il s’agit de deux images à des niveaux différents du même système. Le premier pas vers cet objectif a été donné dans la section 3.3 sous l’appellation de “système ouvert”.

Ce chapitre a procédé en deux phases. La première phase a été d’appliquer la méthodologie de la logique de réécriture pour définir la sémantique des grammaires à double-somme amalgamée. L’utilisation d’une perspective dénotationnelle nous a permis de nous libérer de la contrainte d’une catégorie fixée par le formalisme. De cette liberté acquise, nous avons pu entamer l’exploration de catégories discrètes par la définition d’une algèbre de structure dotée de deux opérateurs: $\mathit{Architecture}(_ \in _)$ et $\mathbf{Famille}(\mathbf{Family}(_))$. Cette exploration débouche sur une modélisation complète de la structure d’apartés dans des domaines coopératifs proche de l’énoncé de [Vil95]. Cette première application montre la complexité des structures impliquées dans la coopération et la possibilité des graphes à double-somme pour décrire de façon compacte de telles structures.

Chapitre

3

Comparaisons de formalismes

L'un des intérêts majeurs des systèmes formels est de parvenir à définir sans équivoque le comportement d'un système. Nous allons considérer une abstraction fondamentale du système étudié, et nous ne considérons pas le temps comme une donnée continue mais uniquement sous l'angle d'une dynamique (i.e. relation de causalité) donc discret. Le sens que nous donnerons à un système est une succession d'évènements uniques, de singularités, pour lesquels nous disposons d'un *avant* et d'un *après* séparables et uniques. Simplement, par rapport à l'approche courante d'analyse des systèmes dynamiques [Mil89], nous prenons la phrase précédente au pied de la lettre et nous allons effectivement identifier chaque transformation. La structure ainsi définie est syntaxiquement un graphe dirigé, pour lequel on a un ensemble de sommets (les états) un ensemble d'arêtes (les transformations) et deux fonctions source et cible qui, à chaque transformation, associent son état de départ et d'arrivée. Nous montrons que l'approche de la logique de réécriture:

1. définition d'une relation par un ensemble de séquents [Plo81];
2. obtention du graphe par étiquetage des éléments de la relation par le séquent qui le prouve [BC88];

s'applique aux trois formalismes que nous étudions. Ceci nous permet de disposer d'une sémantique pour chaque formalisme dont l'espace d'arrivée est le même et sur lequel nous disposons d'une notion d'équivalence: l'isomorphisme de graphes. Ces propriétés sont mises à profit pour définir formellement qu'un formalisme étend un autre formalisme. Il faut pour cela être capable d'injecter chaque système du premier dans un système du second et de prouver que leurs sémantiques respectives sont équivalentes (ici isomorphes). Ceci induit une mécanique répétitive qui sera utilisée quatre fois:

- ◊ définition et preuve de la bijection des états;

- ◊ définition et preuve de l'injection des systèmes;
- ◊ définition et preuve de la bijection des transformations;
- ◊ preuve de la préservation des sources et cibles.

Cette mécanique est mise à profit une première fois pour montrer que les Réseaux de Petri Place/Transition peuvent être étendus soit par la logique de réécriture, soit par les grammaires à double-somme. Pour pouvoir manipuler ces formalismes dans un contexte commun, nous posons qu'un formalisme définit un ensemble de systèmes, et nous montrerons que les définitions suivantes peuvent convenir pour les trois systèmes étudiés:

$$\begin{aligned}
\mathbb{P} &= \{ \langle P; \mathcal{T} \rangle \mid P \text{ est un ensemble (de places), } \mathcal{T} \text{ est une famille} \\
&\quad \text{de paires de fonctions de } P \text{ dans } \mathbb{N} \text{ (i.e. marquages)} \} \\
\mathbb{R} &= \{ \langle \Sigma_{\mathcal{E}}^X; \mathcal{R} \rangle \mid \Sigma_{\mathcal{E}}^X \text{ est une signature avec variables et équations,} \\
&\quad \mathcal{R} \text{ est une famille de paires de termes } (\mathcal{T}_{\mathcal{E}}^{\Sigma}(X)) \} \\
\mathbb{G} &= \{ \langle \mathbf{X}; \mathcal{L} \rangle \mid \mathbf{X} \text{ est une catégorie finiment co-complète, } \mathcal{L} \text{ est} \\
&\quad \text{une famille de paires de flèches de } \mathbf{X} \text{ ayant même source} \}
\end{aligned}$$

Une fois ces définitions acquises, nous pouvons considérer le diagramme ci-dessous: les flèches représentent des injections qui commutent à un isomorphisme près avec les sémantiques définies sur chacun des ensembles de systèmes, les lettres doublées représentent l'ensemble des systèmes de chaque formalisme comme défini plus haut.

$$\mathbb{R} \longleftarrow \mathbb{P} \longrightarrow \mathbb{G}$$

Mais ceci ne fait que deux flèches et donc deux preuves; les deux autres preuves sont les plus novatrices car elles cherchent à caractériser l'ensemble des théories de réécriture ou de grammaires par double-somme tel que l'on ait le diagramme suivant:

$$\mathbb{R}_{\sqrt{\mathbb{P}}} \longrightarrow \mathbb{P} \longleftarrow \mathbb{G}_{\sqrt{\mathbb{P}}}$$

Les caractérisations de ces ensembles pour lesquels on sait prouver l'extension sont les suivantes:

- $\mathbb{R}_{\sqrt{\mathbb{P}}}$ Théorie de réécriture de multi-ensemble de termes de profondeur 1 sans variables libres à droite. Cette caractérisation recouvre une bonne partie de l'utilisation effective des réseaux de Petri Haut-Niveau, même si dans leur sémantique la nécessité de l'absence de variables libres à droite n'est pas explicitée [Jen96].
- $\mathbb{G}_{\sqrt{\mathbb{P}}}$ Système dont l'interface est l'élément initial, la catégorie de travail (\mathbf{X}) ayant des co-produits libres. On instancie cette caractérisation pour l'opérateur d'architecture sur la catégorie **Set** qui étend algébriquement les structures classiques de graphes. Les générateurs du monoïde libre formé par les classes d'équivalence et le co-produit sont les classes de composantes connexes.

La première partie s'attache à uniformiser les définitions des sémantiques en utilisant la technique initiée dans [BC88], utilisée dans notre état de l'art pour la logique de réécriture. Ces différentes sémantiques sont reliées aux sémantiques originales. La deuxième section montre les deux extensions des réseaux de Petri en logique de réécriture et en grammaires de graphes. Enfin la dernière section détaille les caractérisations des systèmes qui s'étendent en des réseaux de Petri Place/Transition.

3.1 Sémantique discrète de comportement et extension

Pour parvenir à définir des équivalences entre des formalismes, il nous faut faire intervenir une notion de sémantique. Une sémantique de comportement est une fonction totale associant à toute représentation spatiale une représentation temporelle. La sémantique permet ainsi de rendre comparables des systèmes incomparables directement.

Plus précisément, un formalisme \mathbb{A} permet de définir l'ensemble des systèmes \mathbb{A} . Si nous avons deux formalismes \mathbb{A} et \mathbb{B} , nous obtenons deux ensembles de systèmes \mathbb{A} et \mathbb{B} . Les éléments de ces deux systèmes sont en général incomparables; pour pouvoir les comparer, il va falloir les projeter dans le même ensemble qui disposera d'une égalité ou d'une équivalence. En fait, nous les projeterons dans une catégorie **Sens** et considérerons les isomorphismes de cette catégorie pour statuer de cette égalité. On peut considérer qu'on les projette dans $[\mathbf{Sens}]$ (i.e. l'ensemble des classes d'équivalences, Cf. 2.1) pour retrouver notre première intuition de projection dans un ensemble.

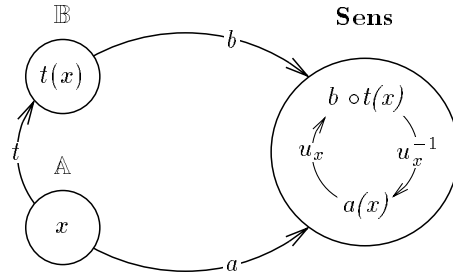
Définition 3.1 (Sémantique discrète) Soit T un formalisme, et \mathbb{T} l'ensemble des systèmes qu'il définit. Une sémantique discrète $(\mathbf{t} : \mathbb{T} \rightarrow \mathbf{obj}(\mathbf{Sens}))$ vers **Sens** est une fonction \mathbf{t} de \mathbb{T} , l'ensemble des systèmes de T , dans $\mathbf{obj}(\mathbf{Sens})$, l'ensemble des objets de **Sens**.

Une sémantique est donc la définition conjointe d'un formalisme de description de système et de la fonction qui associe un sens à chaque système considéré. Deux sémantiques qui ont même source traitent des mêmes systèmes. Si on parvenait à doter les systèmes de morphismes, on pourrait étendre la définition de sémantique et la rapprocher de celle de foncteur. Bien que nous ne considérions pas cette extension, les flèches de **Sens** vont nous intéresser pour la définition suivante. Un formalisme interprété b étend a , ssi on sait associer à tout système de a un système de b tel que leur image par la sémantique (i.e. leur sens) soient isomorphes.

Définition 3.2 (Extension) Une sémantique \mathbf{b} étend \mathbf{a}

$$(\mathbf{a} : \mathbb{A} \rightarrow \mathbf{obj}(\mathbf{Sens})) \longrightarrow (\mathbf{b} : \mathbb{B} \rightarrow \mathbf{obj}(\mathbf{Sens}))$$

ssi il existe une fonction $(\mathbf{t} : \mathbb{A} \rightarrow \mathbb{B})$ vérifiant $\forall x \in \mathbb{A}, a(x) \sim b \circ t(x)$. Ce que l'on peut représenter par le schéma suivant



Remarquons avant toute chose que la notion d'extension est paramétrée par la sémantique. Elle n'est donc pas absolue: prouver une équivalence dépend du langage dans lequel on exprime cette équivalence, ce langage est contraint par les critères

choisis comme pertinents [Ver89]. Seuls deux systèmes identiques sont équivalents pour toutes les sémantiques imaginables. Ensuite, nous posons cette équation sans expliciter l'isomorphisme, pour faire la preuve il faudra expliciter cet isomorphisme. Ceci nous permet de donner à notre extension une forme syntaxique plus générale, en particulier si le signe \sim ne signifie plus un isomorphisme mais une autre forme d'équivalence.

Notre méthodologie est ici de chercher systématiquement à caractériser un domaine de réduction pour chaque extension. Un domaine de réduction est un sous-ensemble des systèmes de \mathbb{B} (noté $\mathbb{B}_{\sqrt{\mathbb{A}}}$) pour lequel on sait prouver une extension.

$$(\mathbf{b}^\circ : \mathbb{B}_{\sqrt{\mathbb{A}}} \rightarrow \mathbf{obj}(\mathbf{Sens})) \longrightarrow (\mathbf{a}^\circ : \mathbb{A} \rightarrow \mathbf{obj}(\mathbf{Sens}))$$

Nous avons pris soin de changer le nom des deux sémantiques car elles seront plus contraintes pour la réduction. Dans le sens de l'extension, nous considérerons le graphe réflexif complet de tous les états définissables, et dans le sens de la réduction nous ne considérerons que les états accessibles à partir d'un état initial (x°). Ceci pour conserver une caractéristique essentielle des réseaux de Petri : ils sont finis.

3.2 Ensembles de systèmes et sémantique

Nous allons maintenant détailler nos différents ensembles de systèmes et justifier la possibilité de les écrire comme nous le faisons. Cette justification sera ici syntaxique, la justification sémantique sera donnée plus largement dans les sous-sections.

Pour les réseaux de Petri Place/Transition, la définition que nous donnons correspond à l'écriture sous forme curriifiée (cf. 1.3) de ces réseaux.

$$\mathbb{P} = \{ \langle P; \mathcal{T} \rangle \mid P \text{ est un ensemble (de places), } \mathcal{T} \text{ est une famille de paires de fonctions de } P \text{ dans } \mathbb{N} \text{ (i.e. marquages)} \}$$

En ce qui concerne la logique de Réécriture notre simplification correspond à éviter la duplication d'un ensemble des variables. Un ensemble de variables étant défini pour les équations, on le réutilise dans les règles. Cette duplication est de nature opérationnelle il me semble : on ne décrit pas les équations opérationnellement, elles sont codées au plus profond du moteur pour des raisons évidentes de performance. Le fait que l'on puisse indexer l'ensemble des règles correspond au cas de règles univoque, nécessaire pour considérer les termes de preuves.

$$\mathbb{R} = \{ \langle \Sigma_{\mathcal{E}}^X; \mathcal{R} \rangle \mid \Sigma_{\mathcal{E}}^X \text{ est une signature avec variables et équations, } \mathcal{R} \text{ est une famille de paires de termes } (\mathcal{T}_E^\Sigma(X)) \}$$

Enfin pour les grammaires de graphes, nous omettons l'état initial par rapport à la définition classique

$$\mathbb{G} = \{ \langle \mathbf{X}; \mathcal{L} \rangle \mid \mathbf{X} \text{ est une catégorie co-complète, } \mathcal{L} \text{ est une famille de paires de flèches de } \mathbf{X} \text{ ayant même source} \}$$

L'ajout d'un état initial est un problème adressé de façon uniforme pour l'ensemble des formalismes considérés par la définition de *système initialisé* donnée dans cette section.

3.2.1 Sémantiques du comportement

Nous définissons ici notre catégorie **Sens**. Comme indiqué dans l'introduction de ce chapitre nous allons identifier toutes les transitions. Nous raffinons notre premier propos en introduisant ici le fait que nous allons considérer une transformation sans conséquences faisant une boucle sur chaque état (même source et même cible). Notre catégorie **Sens** est donc la catégorie des graphes dirigés: **Graphs**.

Définition 3.3 (Sémantique de comportement) *Une sémantique de comportement est une sémantique dont la cible est $\mathbf{obj}(\mathbf{Graphs})$ (i.e. l'ensemble des graphes dirigés).*

Pour définir ce graphe, nous utiliserons une approche commune. D'une part nous définirons une sémantique dans **Relation** qui, à tout système, associe sa dynamique. Cette sémantique sera définie par un ensemble de séquents [Plo81]. Dans un deuxième temps, nous étiquetterons chaque élément de la relation par le séquent qui a permis de prouver l'existence de cet élément, sans référence à la cible et à la source.

Le fait d'étiqueter la relation fait écho à la notion de relation étiquetée, d'où la question: pourquoi ne pas prendre **Bissimulation** comme cible de la sémantique? Cette catégorie qui a pour objet les relations étiquetées et comme flèches les *bissimulations* [Mil89] est en effet la plus utilisée pour les analyses comportementales. La raison principale est que notre équivalence est plus fine. En effet, nous conservons plus d'informations que le nom de la règle servant de base à la transformation: nous lui accolons le contexte d'application de la règle. La relation étiquetée devient un graphe dirigé puisque les étiquettes sont rendues, par là, uniques. Notre esquisse de preuve repose sur la remarque que tous les formalismes que nous utilisons sont à base de familles de règles. Soit $\langle B; \mathbf{b}(_) \rangle$ un système à base de règles. On peut définir pour tout $x \in B$ le système constitué de la seule règle $\mathbf{b}(x)$ et considérer sa sémantique dans **Relation**, ceci forme la sémantique étiquetée (cf. 2.2). Comme nos traductions se font règle par règle, le graphe associé à chaque règle est isomorphe à celui de sa traduction, nous en déduisons que l'isomorphisme des états (composante de l'isomorphisme de graphe) s'exporte comme un isomorphisme des relations. Comme cet isomorphisme est indépendant du système, il forme une *bissimulation*. Ceci montre que toutes les extensions dans **Graphs** sont vraies dans **Bissimulation**.

La sémantique de comportement est une sémantique abstraite souvent infinie même pour des systèmes triviaux (e.g. un réseau avec une place a pour sémantique l'ensemble des entiers); pour pouvoir considérer ces objets opérationnellement, il faut définir des sémantiques qui renvoient des objets finis. Nous définissons deux sémantiques opérationnelles, *simulation* et *accessibilité*, elles sont toutes les deux basées sur la notion de système initialisé.

Définition 3.4 (Système initialisé) *Une paire d'objet $\langle s; x^o \rangle$ est un système initialisé pour une sémantique de comportement $(\mathbf{a} : \mathbb{A} \rightarrow \mathbf{Graphs})$ ssi le premier est un système ($s \in \mathbb{A}$) et le deuxième est un état de ce système ($x^o \in \mathbf{sommets}_{\mathbf{a}(s)}$). On note \mathbb{A}^o l'ensemble des systèmes marqués pour la sémantique de comportement considérée.*

La sémantique d'accessibilité renvoie le plus petit graphe contenant tous les chemins partant du sommet initial et la sémantique de simulation renvoie les transformations dont l'état initial est source.

Définition 3.5 (Sémantique d'accessibilité) *Soit $(\mathbf{a} : \mathbb{A} \rightarrow \mathbf{obj}(\mathbf{Graphs}))$ une sémantique de comportement; on définit la sémantique d'accessibilité $(\mathbf{a}^o : \mathbb{A}^o \rightarrow$*

$\mathbf{obj}(\mathbf{Graphs})$) comme le plus petit graphe inclus dans $a(s)$ contenant tous les chemins partant de x^o .

Définition 3.6 (Sémantique de simulation) Soit $(a : \mathbb{A} \rightarrow \mathbf{obj}(\mathbf{Graphs}))$ une sémantique de comportement; on définit la sémantique de simulation $(a^> : \mathbb{A}^o \rightarrow \mathbf{obj}(\mathbf{Set}))$ comme l'ensemble des arêtes ayant pour source x^o dans $a(s)$.

Construire un outil informatique qui implémente la sémantique de simulation (une fonction donc un possible calcul) est le premier pas pour pouvoir accéder à la sémantique d'accessibilité. Les problèmes opérationnels relatifs à la minimisation de la taille de ce graphe ne sont pas adressés ici. Néanmoins les gains potentiels de nos abstractions relativement à ce problème concluront les sous-sections relatives à la logique de réécriture et des grammaires de graphes. Pour les grammaires de graphes, la construction de la sémantique a été traitée au chapitre II, il sera donc rappelé les caractéristiques de cette sémantique pour l'objet de ce chapitre.

3.2.2 Sémantique dénotationnelle des P/T

L'ensemble des états d'un réseau de Petri est l'ensemble des marquages, autrement dit \mathbb{N}^P . A tout réseau de Petri, nous pouvons associer la dynamique suivante, qui repose sur le principe de la *congruence monoïdale* [Juh99], nous avons donc un séquent caractérisé par le réseau de Petri lui-même:

$$\langle P; T; \bullet R; R \bullet \rangle \frac{(t \in T) \wedge (M \in \mathbb{N}^P)}{(\text{fn } p \Rightarrow \bullet R(t, p) + M(p)) \rightarrow (\text{fn } p \Rightarrow R \bullet(t, p) + M(p))} \quad (3.1)$$

Cette équation est correcte et ne fait pas intervenir de notations intermédiaires. On peut la rendre plus simple en considérant la forme curriée des fonctions d'incidence; on obtient alors l'équation suivante qui est une congruence pour l'opérateur \oplus qui, ici, est la somme vectorielle ($a \oplus b = (\text{fn } p \Rightarrow a(p) + b(p))$).

$$\langle P; T; \bullet R; R \bullet \rangle \frac{(t \in T) \wedge (M \in \mathbb{N}^P)}{(\bullet R(t) \oplus M(p)) \rightarrow (R \bullet(t) \oplus M)} \quad (3.2)$$

L'identification des arêtes ne pose pas de difficulté, il s'agit d'une paire formée d'une transition t et d'un marquage de contexte M . Compte-tenu de la proximité syntaxique des séquents (Cf 2.2), on obtient la construction suivante comparable à celle donnée pour les grammaires à double-pushout.

Définition 3.7 (Sémantique par entrelacement d'un réseau) Soit p la fonction qui à tout réseau de Petri Place/Transition $\langle P; T; \bullet R; R \bullet \rangle$ associe le graphe dirigé suivant:

$$\langle \mathbb{N}^P; T \times \mathbb{N}^P; \mathbf{src}(-); \mathbf{tgt}(-) \rangle$$

Où $\mathbf{src}(-)$ et $\mathbf{tgt}(-)$ s'expriment comme suit:

- $\mathbf{src}(\langle t; m \rangle) = (\text{fn } p \Rightarrow \bullet R(t, p) + M(p))$
- $\mathbf{tgt}(\langle t; m \rangle) = (\text{fn } p \Rightarrow R \bullet(t, p) + M(p))$

En continuant d'appliquer la démarche graphique originale des réseaux de Petri, ceci correspond à marquer une transition avec un jeton. On peut représenter comme cela une composante connexe du comportement d'un système comme dans la figure 3.1. Il apparaît que notre façon de représenter les transformations consiste à rajouter

le contexte sur l'étiquetage de la transition. Pour différencier les marquages courants des marquages contextes d'une transition, on a représenté ces derniers par des jetons blancs. Cette sémantique n'est pas représentable car infinie, même pour un réseau avec une seule place. Nous avons donc représenté ici une composante connexe de ce graphe pour le système donné par le réseau sous-jacent à tous les réseaux "marqués" représenté.

Cette représentation supporte la concurrence au sens de [MM90]. il suffit d'autoriser le "marquage" d'un nombre quelconque de transitions. Cette possibilité sera détaillée dans la section 3.2.3.

Equivalence avec la forme classique Pour parvenir à représenter le graphe de comportement, on utilisera principalement les deux sémantiques qui utilisent des systèmes marqués: accessibilité et simulation. Ici nous allons montrer que la sémantique de simulation est équivalente à la forme classique de la sensibilisation et du tir d'une transition.

Sensibilisation L'existence d'un marquage de contexte est équivalent à la condition classique car l'ordre utilisé sur les vecteurs est l'ordre induit par la somme vectorielle:

$$a \leq b \iff \exists c \in \mathbb{N}^P \text{ tel que } a \oplus c = b$$

Tir Il faut faire ici assez attention, car la différence de vecteur (\ominus) n'est pas une opération interne: elle est interne dans \mathbb{Z}^P . C'est donc dans \mathbb{Z}^P que l'on écrit l'équation suivante:

$$(\bullet t \leq M) \wedge (M \ominus \bullet t) \oplus \bullet = M' \iff \exists C \in \mathbb{N}^P \text{ tel que } C \oplus \bullet t = M' \wedge C \oplus \bullet = M'$$

Les équations classiques des réseaux de Petri sont donc écrites dans \mathbb{Z}^P puis contraintes [Reu89]: la sensibilisation est l'expression de cette contrainte. En fait, le " (\ominus) " de cette équation correspond exactement au "complément" des grammaires de graphes. Comme décrit au début du chapitre II, l'utilisation d'une sémantique dénotationnelle permet de n'utiliser que des sommes et donc de s'affranchir du problème de la différence.

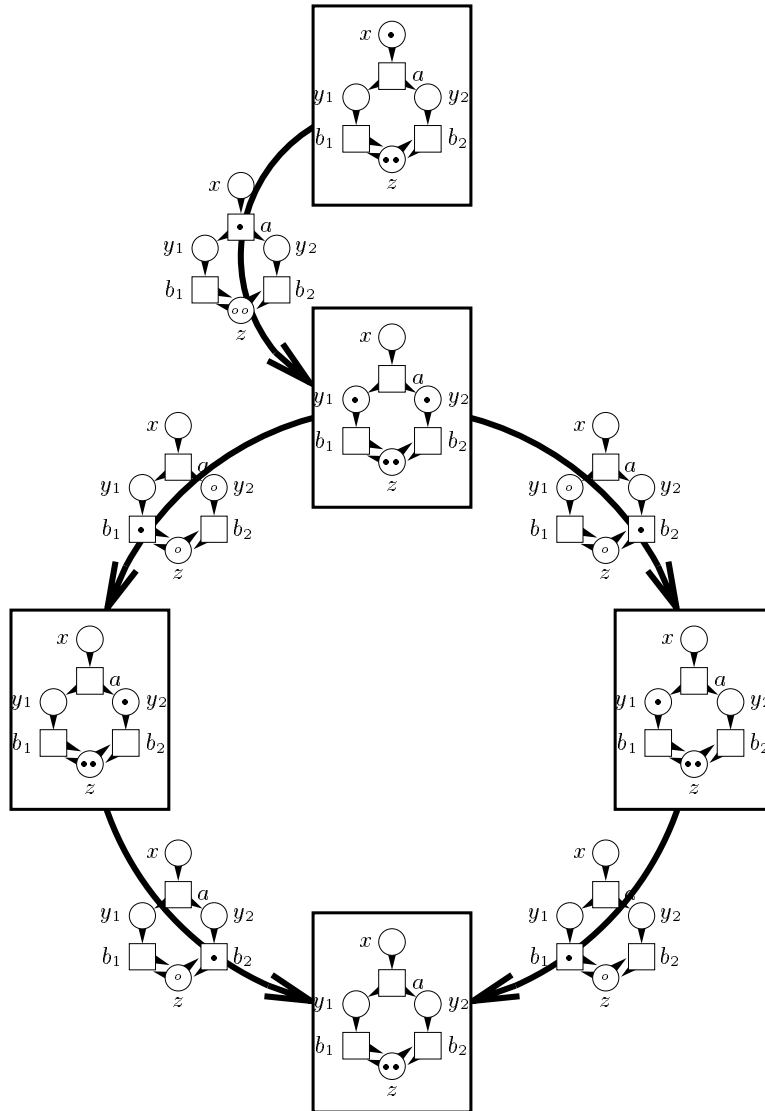
3.2.3 Sémantique sans concurrence de la Logique de réécriture

Nous supposons construite la catégorie $\mathbf{r}(a)$ où $a \in \mathbb{R}$, ayant pour objets les classes de termes et comme flèches les classes de termes de preuves. Nous considérons comme hypothèse de travail que nous pouvons faire des constructions par induction sur la structure des termes de preuves, qui sont des classes d'équivalence de termes modulo les équations de la sémantique.

Montrer l'extension directement entre les sémantiques données est impossible. En effet la catégorie $\mathbf{r}(a)$ n'est pas la catégorie libre engendrée par le graphe dirigé de la sémantique par entrelacement: des transitions sont rajoutées qui correspondent à des applications concurrentes. Ces transformations, bien que représentables avec notre notation de transformations, ne font partie ni de la sémantique par entrelacement ni des séquences de franchissement, ni de la catégorie libre engendrée. Remarquons ici que les deux dernières alternatives représentent la même construction: considérer des mots de transformations comme transformations.

Pour réduire la catégorie $\mathbf{r}(a)$, nous allons construire des "sous-ensembles" de l'ensemble des flèches de cette catégorie qui pourront former soit un graphe dirigé (sémantique sans composition explicite), soit un graphe avec composition (sémantique sans concurrence explicite). L'extension sera montrée avec le graphe dirigé de la sémantique sans concurrence et sans composition.

 FIG. 3.1 – Sémantique d'entrelacement à transitions univoques



Fonctions de filtrage Pour définir ces “sous-ensembles”, nous poserons deux fonctions, la première dénotant le *degré de concurrence*, la seconde le nombre de compositions appelé *longueur*.

On définit ces fonctions sur les termes puis on étudiera l’impact des équations dans un deuxième temps pour étendre ces fonctions aux classes de termes.

Définition 3.8 (Degré d’un terme de preuve) Soit $(\Pi(-) : \text{arr}(\mathbf{r}(a)) \rightarrow \mathbb{N})$ la fonction suivante définie par induction sur la structure des termes de preuves:

1. **Réflexivité:** Pour chaque $[t] \in \mathcal{T}_E^S(X)$

$$\Pi(\text{Id}^{[t]}) = 0$$

2. **Congruence:** Pour chaque f , symbole fonctionnel d’arité n :

$$\Pi(f(a_1, \dots, a_n)) = \sum_{i \in [1..n]} \Pi(a_i)$$

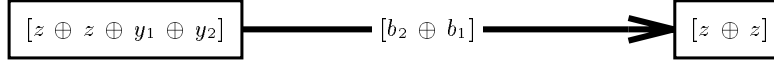
3. **Remplacement:** Pour chaque règle $(\mathbf{r} : [t(x_1, \dots, x_n)] \rightarrow [t'(x_1, \dots, x_n)])$,

$$\Pi(r(a_1, \dots, a_n)) = 1 + \sum_{i \in [1..n]} \Pi(a_i)$$

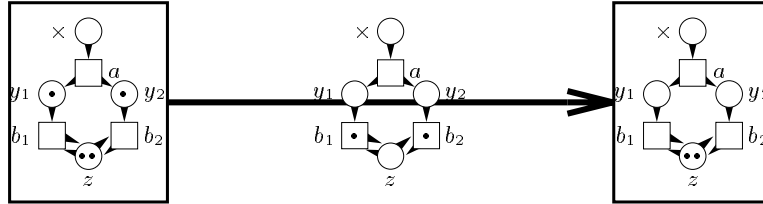
4. **Transitivité:**

$$\Pi(a; b) = \max \langle \Pi(a); \Pi(b) \rangle$$

C’est ce filtre qui élimine l’exemple de tir concurrent de degré 2 donné ci-dessous:



Ici apparaît la notion de concurrence comme l’opérateur de congruence appliqué sur des transformations. Ceci peut être représenté sans difficulté par notre notation d’étiquetage univoque des transitions. il suffit de mettre un “jeton” dans chaque place. Ceci permet de fournir une notation à la concurrence qui étend la notation classique :



Définition 3.9 (Longueur d’un terme de preuve) Soit $(\Lambda(-) : \text{arr}(\mathbf{r}(a)) \rightarrow \mathbb{N})$ la fonction suivante définie par induction sur la structure des termes de preuves:

1. **Réflexivité:** Pour chaque $[t] \in \mathcal{T}_E^S(X)$

$$\Lambda(\text{Id}^{[t]}) = 0$$

2. **Congruence:** Pour chaque f , symbole fonctionnel d’arité n :

$$\Lambda(f(a_1, \dots, a_n)) = \max_{i \in [1..n]} \Lambda(a_i)$$

3. **Remplacement:** Pour chaque règle $(\mathbf{r} : [t(x_1, \dots, x_n)] \rightarrow [t'(x_1, \dots, x_n)])$,

$$\Lambda(r(a_1, \dots, a_n)) = 1 + \max_{i \in [1..n]} \Lambda(a_i)$$

4. **Transitivité:**

$$\Lambda(a; b) = \Lambda(a) + \Lambda(b)$$

Compatibilité des équations Parmi les équations, seule la dernière (Cf. 1.3) pose un problème : échange. Elle abaisse la concurrence de 1 et augmente la longueur de 1. On a donc deux valeurs extrémales pour une classe de termes de preuve. Nous prendrons pour $\Pi(_)$ la valeur du terme qui a le moins de concurrence, et pour $\Lambda(_)$ la valeur d'un terme qui a le moins de compositions. C'est ce choix de construction qui motive l'usage du qualificatif "explicite".

Classification des termes de preuves Les termes *sans concurrence explicite* sont de degrés 1 (i.e. éléments de $\Pi^{-1}(1)$), les termes de longueur 1 (i.e. éléments de $\Lambda^{-1}(1)$) étant nommés *sans composition explicite*. Enfin, nous appellerons termes *nus* les éléments appartenant aux deux classes.

Préservation des fonctions de structure Sur les trois sous-ensembles des termes de preuves, les fonctions $\mathbf{src}(_)$ et $\mathbf{tgt}(_)$ sont définies. Les sous-ensembles étant inclus dans l'ensemble de définition de ces fonctions.

Pour l'identité, par contre, l'image des classes de termes par l'identité n'est incluse dans aucun des trois ensembles car pour chaque $[t] \in \mathcal{T}_E^S(X)$, $\Pi(\mathbf{Id}^{[t]}) = 0$ et $\Lambda(\mathbf{Id}^{[t]}) = 0$. La composition reste interne aux termes *sans concurrence*, et la congruence reste interne aux termes *sans composition*. Par contre pour les termes *nus*, ni la composition ni la congruence ne sont internes.

En conséquence, la sémantique sans concurrence est un graphe dirigé muni d'une opération de composition, les deux autres seront des graphes dirigés. Nous appellerons *graphe composable* la structure de graphe dirigé muni d'une loi de composition.

Trois "Sous-sémantiques" Aux trois ensembles de termes de preuves ainsi filtrés, on définit trois sémantiques en plus de la sémantique existante, qui est la plus générale:

Sémantique nue On note $\mathbf{r}_{CT}(a)$ la fonction qui à toute théorie de réécriture lui associe le graphe dirigé ayant pour sommets les classes de termes et pour arêtes les classes de termes *sans composition* et *sans concurrence*. La structure résultante est un graphe dirigé:

$$\mathbf{r}_{CT}(a) \stackrel{\text{def}}{=} \langle \mathcal{T}_E^S(X); \Pi^{-1}(1) \cap \Lambda^{-1}(1); \mathbf{src}(_); \mathbf{tgt}(_) \rangle$$

Sémantique Concurrente On note $\mathbf{r}_C(a)$ la fonction qui à toute théorie de réécriture lui associe le graphe dirigé ayant pour sommets les classes de termes et pour arêtes les classes de termes *sans compositions*. La structure résultante est un graphe dirigé:

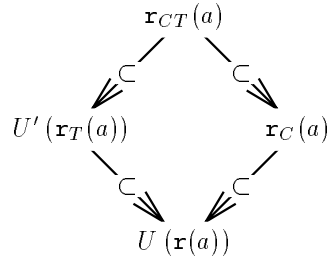
$$\mathbf{r}_C(a) \stackrel{\text{def}}{=} \langle \mathcal{T}_E^S(X); \Lambda^{-1}(1); \mathbf{src}(_); \mathbf{tgt}(_) \rangle$$

Sémantique Transitive On note $\mathbf{r}_T(a)$ la fonction qui à toute théorie de réécriture lui associe le graphe composable ayant pour objets les classes de termes et pour flèches les classes de termes *sans concurrence* ou Transitive. La structure résultante est un graphe composable:

$$\mathbf{r}_T(a) \stackrel{\text{def}}{=} \langle \mathcal{T}_E^S(X); \Pi^{-1}(1); \mathbf{src}(_); \mathbf{tgt}(_); _ \circ _ \rangle$$

Inclusion des "Sous-sémantiques" On peut toujours considérer une catégorie comme un graphe dirigé, il suffit d'oublier l'existence de l'opérateur de composition et d'identité. On appelle U ce foncteur d'oubli qui à toute catégorie associe le graphe dirigé sous jacent. De la même façon, on appelle U' le foncteur d'oubli qui

à tout graphe composable associe le graphe dirigé sous-jacent par oubli de la loi de composition des arêtes. On a alors le diagramme suivant qui commute pour toute théorie de réécriture a dans la catégorie des graphes dirigés.



Préservation de l'accessibilité Cette propriété très importante est déjà contenue dans [MM90]. L'intuition de la preuve est que l'on peut dé-concurrer un terme de preuve, en procédant à l'entrelacement de séquences. Avant de réaliser cette preuve, nous exprimons d'abord le problème avec nos définitions.

Propriété 3.1 (Préservation de l'accessibilité) *Soit a une théorie de réécriture, pour tout $[t] \in \mathcal{T}_E^S(X)$ un sommet de la cible de la sémantique (i.e. une classe de terme). Que l'on considère l'image de la paire $\langle a; [t] \rangle$ par la sémantique accessible issue de la sémantique complète ou par la sémantique la plus petite, l'ensemble des états de cette image est le même.*

$$\text{sommets}_{\mathbf{r}^o}(\langle a; [t] \rangle) = \text{sommets}_{\mathbf{r}_{CT}^o}(\langle a; [t] \rangle)$$

Nous allons découper cette preuve en deux parties, la longueur du chemin entre les deux sémantiques dans le graphe donné ci-dessus.

1. La première partie consiste dans la même propriété pour la sémantique sans concurrence explicite dite sémantique Transitive.

$$\text{sommets}_{\mathbf{r}^o}(\langle a; [t] \rangle) = \text{sommets}_{\mathbf{r}_T^o}(\langle a; [t] \rangle)$$

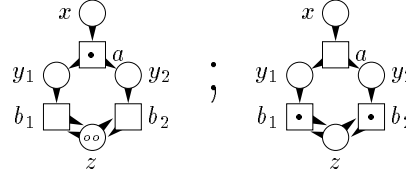
Ceci résulte du fait que tout terme de preuve peut de réduire de façon terminante mais non confluente en un terme de preuve ayant même source et même cible mais de degré 1, ce *système de réduction* est donné ci-dessous, pour tout f opérateur d'arité n de la signature:

$$\begin{aligned} f(a_1, \dots, a_i, \dots, a_n) &\rightarrow f(a_1, \dots, t_i, \dots, a_n); f(t'_1, \dots, a_i, \dots, t'_n) \\ f(a_1, \dots, a_i, \dots, a_n) &\rightarrow f(t_1, \dots, a_i, \dots, t_n); f(a_1, \dots, t'_i, \dots, a_n) \end{aligned}$$

Le degré de concurrence étant une *fonction de progrès*, ce système finiment branché est terminant. Il est structurellement non confluent. Cette propriété souligne le fait que *deux* termes de preuves de concurrence $n-1$ peuvent être pris comme réductions du terme de preuve de concurrence n . L'*explosion combinatoire* résultante est une motivation de la recherche du "vrai-parallélisme" dans l'écriture du graphe des états accessibles [Ver89].

2. La deuxième est une conséquence de la définition de la sémantique connexe de comportement. L'existence d'un chemin est équivalente à l'existence d'une flèche si la composition est définie. Donc l'existence d'un chemin de $\mathbf{r}_T^o(\langle a; [t] \rangle)$ implique l'existence d'une flèche de $\mathbf{r}_T^o(\langle a; [t] \rangle)$ qui elle-même implique l'existence d'un chemin dans $\mathbf{r}_{CT}^o(\langle a; [t] \rangle)$

Perspectives: le système de réécriture est intéressant car il permet de déplier tous les termes de la logique de réécriture avec concurrence. On peut donc définir une sémantique d'exploration qui renvoie pour tout système l'ensemble des termes de preuves qui ont pour source l'état initial, et qui ne sont pas préfixes d'un autre (maximaux pour la composition) et qui ne peuvent être obtenus par dépliage d'un autre (maximaux pour la concurrence). Avec une telle sémantique d'exploration, l'image de l'état initial est le terme de preuve $[a \oplus \text{Id}^{z \oplus z}; b_2 \oplus b_1]$ donné avec sous la forme de transition univoque de réseau Place/Transition ci dessous:



Ce terme de preuve contient en effet toute l'information contenue dans la figure 3.1. La réduction de la combinatoire permise par ce type de technique pourrait prendre place à coté d'autres techniques basées sur les ordres partiels [VAM96] même si son principe est différent. Tracer des ponts entre les deux perceptions de la concurrence (commutation partielle des séquences [Maz87] et utilisation d'une congruence sur des termes de preuves [MM90]) est un objectif recherché [MT99] et une perspective possible de ce travail.

3.2.4 Sémantique des grammaires à double-pushout

Notre sémantique des grammaires à double-pushout est déjà largement détaillée dans le chapitre précédent. Nous allons confronter cette sémantique par rapport à l'existant. Cette sémantique a été réalisée en deux phases:

- réalisation d'un graphe abstrait de dérivation, où tous les T-diagrammes sont considérés et pas seulement ceux qui sont des instanciations de règles;
- définition d'une sémantique pour les grammaires à double pushout comme une injection dans ce graphe abstrait de dérivation.

Ceci a trois avantages par rapport à la conception classique:

- c'est une "sémantique de comportement" et ceci est une importante contribution pour les grammaires à double-pushout. En effet jusque là, la seule construction connue était la séquence de dérivation, analogue à la séquence de franchissement pour les réseaux de Petri. Lorsque les séquences de Petri se sont mises à commuter partiellement [Maz87] pour rendre compte de la concurrence, la construction a été adaptée pour les grammaires de graphes. Notre construction est parallèle, mais pour l'autre interprétation que nous connaissons de la concurrence, celle de *congruence sur les transformations* [MM90, Mes92];
- la possibilité de définir une sémantique concrète qui s'injecte dans le graphe abstrait. Une sémantique concrète est un graphe dirigé où chaque sommet est étiqueté par un graphe et chaque arête par un T-diagramme concret. C'est l'existence d'un pushout qui permet de définir les fonctions **src**(_) et **tgt**(_), l'injection canonique dans la sémantique par entrelacement associant chaque élément à sa classe. Cette façon de faire est une alternative à la notion d'isomorphisme standard [CMR⁺97] de la théorie classique. Opérationnellement, ceci impose un raffinement de l'algorithme classique d'exploration des états accessibles. Tout état obtenu par dérivation doit être

confronté modulo isomorphisme à tous les états atteints précédemment. Ceci a une complexité certaine, mais jusqu'à présent le problème du "matching" (non polynomial) a toujours été considéré globalement, or nos transformations sont locales, il est donc redondant de tout recalculer. Il est probable qu'introduire la propriété de *localité* des transformations permettra des gains de complexité qui rendront ces techniques efficaces. [Dör95] a en outre explicité des motifs pour lesquels rechercher une occurrence a la même complexité que trouver un mot dans une chaîne. Ces deux pistes, localité et motifs simples à rechercher, forment le socle d'une recherche ultérieure sur l'efficacité opérationnelle de ces techniques;

- Par rapport à d'autres formalismes considérant les "localités" explicitement, comme les réseaux de Petri Haut-Niveau, nous travaillons à un isomorphisme près. Comme la sémantique concrète reste une injection, la *réduction de la combinatoire* résultant de l'utilisation des classes de graphes est préservée. Cette réduction peut être comparée aux approches par symétrie [MAV96] qui exploitaient les symétries des données pour réduire les symétries de comportement. Dans cette approche, travailler à un isomorphisme près est natif de la théorie, ici c'est à l'applicatif de suivre.

3.3 Extension des réseaux P/T

L'extension des réseaux de Petri Place/Transition est un fait admis, mais poser les définitions sémantiques permettant de prouver ce fait a fait avancer la perception des différentes sémantiques. La conséquence primitive de notre travail outre la preuve de l'extension qui suit, est de permettre l'utilisation des deux techniques, Logique de Réécriture et Grammaires de Graphes avec les mêmes possibilités d'analyse par simulation ou exhaustive que les réseaux de Petri.

3.3.1 Logique de réécriture

Nous allons montrer que la sémantique de la logique de réécriture $\mathbf{r}_{CT}(_)$ étend la sémantique $\mathbf{p}(_)$; pour cela nous devons construire une fonction d'extension puis construire l'isomorphisme des graphes réflexifs.

Définissons tout d'abord la fonction Σ qui, à tout ensemble P , associe la signature $\Sigma(P)$, c'est une signature qui pose chaque place comme une constante de sorte **soupe**.

◊ **Sortes** : $\{\mathbf{soupe}\}$

◊ **Opérateurs** : $\left\{ \begin{array}{lll} _ \oplus _ : & \mathbf{soupe} \times \mathbf{soupe} \rightarrow & \mathbf{soupe} \\ \mathbf{rien} : & \epsilon \rightarrow & \mathbf{soupe} \\ P : & \epsilon \rightarrow & \mathbf{soupe} \end{array} \right.$

Ensemble de variables $X_{\mathbf{soupe}} = \{\alpha, \beta, \gamma\}$

Axiomes ils ne concernent que l'opérateur $_ \oplus _$ dans l'ordre d'apparition : la commutativité, l'élément neutre et l'associativité:

$$\mathcal{E} = \{(\alpha \oplus \beta) \approx (\beta \oplus \alpha), (\mathbf{rien} \oplus \alpha) \approx \alpha, (\alpha \oplus (\beta \oplus \gamma)) \approx ((\alpha \oplus \beta) \oplus \gamma)\}$$

◊ **Equivalence des états** Pour montrer l'équivalence entre les marquages sur P et les classes de termes de $\Sigma(P)$, on va montrer que les deux forment un monoïde libre commutatif engendré par P , il suffira d'exhiber l'isomorphisme entre les générateurs

pour conclure. Notons ici que tous les monoïdes ne sont pas libres, e.g. $(\mathbb{N}-\{1\}, +, 0)$ est généré par 2 et 3 et on a l'équation $2 + 2 + 2 = 3 + 3$.

Classe de termes La définition donnée coïncide avec la définition algébrique [EM90] d'un monoïde libre commutatif engendré par P . Le monoïde $(\mathcal{T}_E^{\Sigma(P)}(\emptyset), _ \oplus _, \mathbf{rien})$ est le monoïde commutatif libre engendré par P .

Marquage On définit \oplus la somme vectorielle, associative, commutative et dotée d'un élément neutre, le vecteur nul $(\vec{0})$:

$$\forall p \in P, a \oplus b(p) = a(p) + b(p)$$

Les générateurs de ce monoïde sont les éléments x tels que $y \oplus z = x$ implique $y = \vec{0} \vee z = \vec{0}$. Un vecteur dont deux composantes sont non-nulles n'est pas un générateur, donc seul les vecteurs ayant une composante non nulle sont de potentiels générateurs. Parmi ceux-là, sachant que \mathbb{N} est le monoïde libre commutatif engendré par 1, seuls les vecteurs ayant une seule composante non nulle égale à 1 sont les générateurs de ce monoïde. Tout vecteur indexé par P peut s'écrire *de façon univoque* comme une somme de tels générateurs. $(\mathbb{N}^P, \oplus, \vec{0})$ est donc le monoïde libre engendré par les vecteurs ayant une seule composante non-nulle égale à 1.

On a isomorphisme entre l'ensemble P et l'ensemble des éléments de \mathbb{N}^P ayant une seule composante non-nulle égale à 1. Donc les deux monoïdes sont isomorphes. On note $\phi(\mathbb{N}^P, \oplus, \vec{0}) = (\mathcal{T}_E^{\Sigma(P)}(\emptyset), _ \oplus _, \mathbf{rien})$ l'isomorphisme de monoïde correspondant.

◊ **Extension** la fonction d'extension est la fonction qui, à tout réseau de Petri $\langle P; T; \bullet R; R^\bullet \rangle$, associe la théorie de réécriture $\phi(R)$ définie comme ci-dessous, où l'on considère la forme curriifiée (Cf. 1.3 et 3.2) des fonctions d'incidences : il suffit alors de traduire chaque membre de la paire pour obtenir une paire de termes.

$$\Psi(R) \stackrel{\text{def}}{=} \langle \Sigma(P); \langle T; (\text{fn } t \Rightarrow \langle \phi(\bullet R(t)); \phi(R^\bullet(t)) \rangle) \rangle \rangle$$

◊ **Equivalence des arêtes** Le problème n'est pas de définir la fonction, ni son injectivité. Ce sont des conséquences des définitions qui ont été choisies à cet effet, la difficulté provient de la surjectivité. On définit la traduction suivante des arêtes, qui est injective car la traduction des marquages est une bijection.

$$\Phi(\langle t; M \rangle) = [t \oplus \text{Id}^{\phi(M)}]$$

Les crochets signifient que la fonction associe à tout marquage une classe de termes; il nous faut à présent montrer que l'expression donnée comme représentant peut être une forme normale de tout terme de preuve sans composition. Pour cela il nous faut considérer les équations définissant les classes de termes qui ne font pas référence à cet opérateur et les orienter. Deux types d'équations ne font pas intervenir la composition, la *préservation de l'identité* et les axiomes.

Préservation de l'identité On oriente l'équation dans le sens qui minimise le nombre d'occurrences de l'opérateur d'identité:

$$\text{Id}^{[t_1]} \oplus \text{Id}^{[t_2]} \rightarrow \text{Id}^{[t_1 \oplus t_2]}$$

Axiomes On oriente les équations dans le sens permettant à l'application d'être en haut du terme le plus à gauche possible:

$$\begin{aligned}
(x \oplus t) &\rightarrow (t \oplus x) \\
(x \oplus (t \oplus y)) &\rightarrow (t \oplus (x \oplus y)) \\
(x \oplus (t \oplus y)) &\rightarrow ((x \oplus t) \oplus y) \\
((t \oplus x) \oplus y) &\rightarrow (t \oplus (x \oplus y))
\end{aligned}$$

Cette dynamique est confluente et terminante. La confluence est décidable et on peut donc vérifier automatiquement que ce système de réduction est confluente [DM00]; pour la terminaison, le système étant finiment branché, on peut identifier une fonction de progrès qui ajoute à la position de t (exprimé comme un entier avec les pairs et les impairs) le nombre d'occurrences de $id()$. En outre un élément est réductible ssi cette fonction est supérieure strictement à 1, le nombre d'occurrences de t . Ce nombre d'occurrences de t reste constant lors de la réduction, remarquez que le problème de l'élément neutre est traité dans les termes par leurs propres équations après application de la réduction de préservation de l'identité. Ceci nous permet d'affirmer que tout terme sans composition aura pour forme normale un terme de la forme $t \oplus Id^c$. Ceci prouvant la surjectivité de la fonction de traduction Φ . Que ϕ soit une bijection nous permet de conclure que Φ pour les arêtes sera injective donc bijective.

A ce niveau, nous pouvons donner le graphe de comportement équivalent à notre graphe témoin : il est donné dans la figure 3.2.

♦ **Préservation de $\text{src}(-)$ et $\text{tgt}(-)$** Cette partie est la plus connue et ne pose pas de difficulté majeure, les deux diagrammes ci-dessous commutent dans **Set**.

$$\begin{array}{ccc}
\langle t; M \rangle & \xrightarrow{\quad \Phi \quad} & [t \oplus Id^{\phi(M)}] \\
\downarrow \text{src}(-) & & \downarrow \text{src}(-) \\
\bullet t \oplus M & \xrightarrow{\quad \phi \quad} & [\phi(\bullet t) \oplus \phi(M)]
\end{array}$$

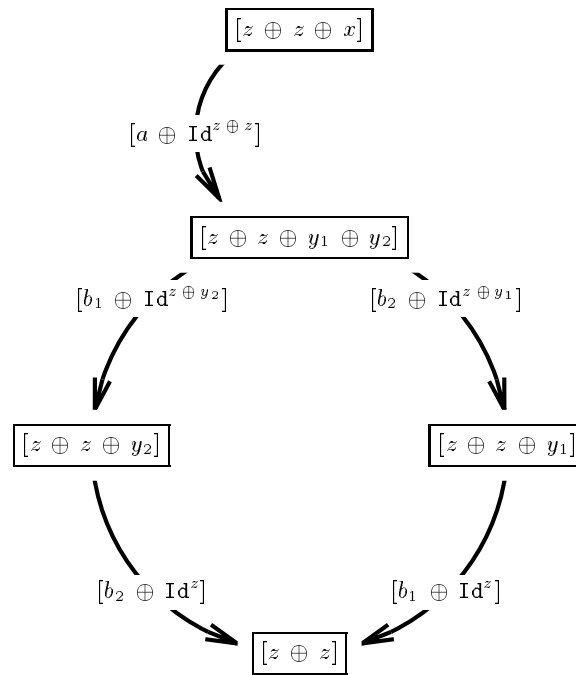
$$\begin{array}{ccc}
\langle t; M \rangle & \xrightarrow{\quad \Phi \quad} & [t \oplus Id^{\phi(M)}] \\
\downarrow \text{tgt}(-) & & \downarrow \text{tgt}(-) \\
t \bullet \oplus M & \xrightarrow{\quad \phi \quad} & [\phi(t \bullet) \oplus \phi(M)]
\end{array}$$

Nous avons donc la propriété suivante :

Propriété 3.2 (Extension des réseaux de Petri) *la fonction Ψ de l'ensemble des réseaux de Petri dans les théories de la logique de réécriture définit une extension de sémantique.*

Ceci clôt notre démonstration que la logique de réécriture forme une extension des réseaux de Petri Place/Transition. Les précisions apportées ici ont une conséquence fondamentale: l'introduction des sémantiques intermédiaires pour la logique de réécriture permet d'utiliser ce formalisme pour étudier les systèmes

 FIG. 3.2 – Equivalent en Logique de réécriture (sémantique nue) de la figure 3.1



avec les outils classiques d'exploration des états accessibles et non seulement par le résultat du calcul associé. Réutiliser la logique de réécriture dans ce contexte est une perspective définitivement ouverte par la démonstration précise de l'extension des réseaux de Petri. Le chapitre IV développe en MAUDE un simulateur qui réalise concrètement cette perspective formelle.

3.3.2 Grammaires à double somme

Le fait que les grammaires de graphes soient une extension des réseaux de Petri est encore un fait acquis [KW86, CM92] bien que les approches citées ici souffrent de n'avoir pas abordé directement le problème qui est le plus technique à résoudre: le problème de la *disparition de l'identité*. Nous avons largement déblayé ce terrain dans la section précédente et ce travail va porter ses fruits ici-même.

Dans [KW86], la problématique est centrée sur la question du vrai parallélisme et l'objectif est de montrer que l'on pourrait munir les grammaires de graphes d'une sémantique du vrai-parallélisme. L'argumentation s'appuie sur une écriture sous forme de grammaires de graphes des transformations que subit un réseau Condition/Évènement marqué (un réseau à places bornées dans le cas général). Les réseaux Condition/Évènement sont les plus utilisés dans cette problématique du vrai-parallélisme car ce sont eux que [WN95] étudie et pour lesquels cette approche a été développée *de facto*. L'extension à des réseaux à places bornées se fait par l'utilisation d'une notation qui esquivé la difficulté: oublier l'identité des nœuds.

[CM92] remarque qu'il n'est pas nécessaire de coder les réseaux comme un graphe, mais que la réécriture de "graphes sans arêtes" est équivalente à un réseau de Petri. L'avancée est réelle car elle découple les réseaux de Petri de leur représentation graphique. Le problème — comme souvent avec cet auteur prolix dont nous ne sommes pour l'instant que le contraire —, c'est que l'intuition est remarquable, le discours ciselé, étayé par des citations couvrant plusieurs communautés, mais aucune construction n'est donnée explicitement. Par exemple, cette section de mathèse correspond à un paragraphe (section 2.1) de l'article en question: *A P/T Petri-Net is a rule base formalism that rewrites labelled sets (the marking) over a fixed set of labels (the places). Graph grammars generalise P/T Petri Nets by replacing labelled sets with labelled graphs, and by allowing a non-empty interface graph in productions. [...] There is a subtle mismatch between the transition [...] and the proposed encoding [...] In fact in the initial marking [...] the tokens in the place A are indistinguishable (i.e. in P/T nets tokens do not have an identity): this is formalised in literature by saying that a marking is a multiset over a set of places P, or equivalently that is an element of the free commutative monoid over P.* L'ensemble des intuitions nécessaires pour comprendre ce chapitre sont contenues dans cette phrase, sauf une: comment ce problème est effectivement résolu par les grammaires de graphes. Il est dit dans ce même paragraphe que le problème leur est apparu lorsqu'ils ont voulu munir les grammaires de graphes d'une sémantique du vrai-parallélisme, renouant ainsi avec les origines du travail de [KW86]. Il faut bien comprendre que ceci ne faisait pas parti de nos préoccupations originales et que cela rend l'approche des grammaires de graphes difficile. Le problème du vrai-parallélisme est une question difficile, surtout lorsqu'il est considéré comme acquis.

Ce problème de l'identité est aggravé par un fait de nature culturelle. Si on souhaite représenter un réseau de Petri graphiquement, la première solution consiste à représenter les places et les transitions de façon graphiquement différenciée et d'indiquer par des entiers sur les arêtes leur multiplicité et par des entiers sur les places le marquage. Pour gagner quelque chose par rapport à la matrice, on adopte la politique suivante: on ne dessine pas les arêtes nulles et on enlève les unités (i.e. 1). Si cette forme d'écriture était la forme admise des réseaux de Petri, les problèmes seraient moindres. Mais l'usage veut que l'on représente le marquage

(i.e. un vecteur d'entiers) par une accumulation de signes identiques: \bullet . Ainsi, en revenant à une forme d'Abaque dynamique, on parcourt à rebord l'abstraction essentielle de l'arithmétique, ce qui est tout sauf anodin. Une fois les jetons dessinés comme des entités distinctes, qu'on le veuille ou non, notre système nerveux les identifie, *puis* les abstrait. Lorsque ce double processus est mis à mal par la recherche d'extensions (qualitative ou quantitative), c'est tout naturellement vers une façon d'identifier des jetons que l'on se tourne (Cf. 1.2).

◊ **Catégorie des marquages à jetons identifiés** Un marquage à jetons identifiés est une fonction qui associe à chaque place un ensemble: tous les jetons sont identifiés. La catégorie pour notre traduction est donc l'ensemble des fonctions de P dans l'ensemble des (petits) ensembles: $\mathbf{obj}(\mathbf{Set})^P$, que l'on notera \mathbf{Set}^P pour simplifier l'écriture et éviter l'empilement des symboles $\mathbf{obj}(_)$. Cette catégorie est co-complète (Cf. préservation des sources).

◊ **Représentation concrète des marquages** La première étape consiste à construire une injection entre N^P et les objets de \mathbf{Set}^P . La base de cette construction est la fonction $(\mathbf{Seg}(_) : \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N}))$ qui, à un entier n , associe le *segment initial* de \mathbb{N} de longueur n (i.e. $[0..n-1]$). On l'étend vectoriellement en la fonction:

$$(\mathbf{Segs}(_) : N^P \rightarrow \mathbf{obj}(\mathbf{Set}^P))$$

◊ **Fonction d'extension** Soit $R = \langle P; T; \bullet R; R^\bullet \rangle$ un réseau de Petri Place/Transition, on définit le système $\mathbf{Trad}(R) = \langle \mathbf{Set}^P; \langle \mathbf{T}; \mathbf{Td}(_) \rangle \rangle$ par la donnée de $\mathbf{Td}(_)$:

$$\mathbf{Td}(t) \stackrel{\text{def}}{=} \langle \langle \mathbf{t} : \emptyset_S \rightarrow \mathbf{Segs}(\bullet R(t)); (\mathbf{t}^> : \emptyset_S \rightarrow \mathbf{Segs}(R^\bullet(t))) \rangle \rangle$$

Chaque règle étant à interface initiale, la donnée des extrémités suffit à donner la règle par unicité du morphisme venant de l'objet initial: $\langle t$ et $t^>$ sont donc uniques par construction. Pour montrer que cette fonction forme une extension, il faut montrer qu'à tout réseau de Petri on peut associer un isomorphisme de la sémantique du réseau de Petri vers la sémantique de sa traduction.

◊ **Bijection des sommets** Il faut montrer que l'on a bijection entre les sommets de $\mathbf{p}(R)$ et $\mathbf{g}(\mathbf{Trad}(R))$. Les sommets de $\mathbf{p}(R)$ sont les marquages du réseau donc les éléments de N^P . Quant aux sommets de $\mathbf{g}(\mathbf{Trad}(R))$, ce sont les classes modulo isomorphisme de \mathbf{Set}^P . On va donc définir une fonction $\mathbf{Sommets}(_)$:

$$\mathbf{Sommets}(M) \stackrel{\text{def}}{=} [\mathbf{Segs}(M)]$$

Si nous considérons notre notation pour les états (ici, des classes de vecteurs d'ensembles), nous obtenons le schéma suivant, qui montre bien que la représentation graphique usuelle des marquages des réseaux de Petri est équivalente à notre notation de classes. Notre approche tend donc à faire de cette notation, non pas une amélioration esthétique, mais une réalité mathématique. Les marquages, représentés par des accumulations de signes identiques dans des ronds, sont des représentations sous formes de diagrammes de Venn de classes d'équivalence de vecteurs d'ensembles indexés par P ! Ceci est le chaînon manquant entre les Grammaires à double pushout et les réseaux de Petri Place/Transition.

$$\begin{aligned}
\text{Sommets}((\text{fn } A \Rightarrow 0 | B \Rightarrow 1 | C \Rightarrow 3 | D \Rightarrow 2)) &= [\text{Segs}((\text{fn } A \Rightarrow 0 | B \Rightarrow 1 | C \Rightarrow 3 | D \Rightarrow 2))] \\
&= \left[\begin{array}{|c|c|c|c|} \hline \text{A} & \text{B} & \text{C} & \text{D} \\ \hline \text{ } & 0 & \begin{smallmatrix} 0 \\ 1 \end{smallmatrix} 2 & \begin{smallmatrix} 0 \\ 1 \end{smallmatrix} \end{array} \right] \\
&= \begin{array}{|c|c|c|c|} \hline \text{A} & \text{B} & \text{C} & \text{D} \\ \hline \text{ } & \bullet & \begin{smallmatrix} \bullet \\ \bullet \end{smallmatrix} & \begin{smallmatrix} \bullet \\ \bullet \end{smallmatrix} \end{array}
\end{aligned}$$

Pour montrer l'injectivité et la surjectivité de la fonction $\text{Sommets}(_)$, on va utiliser une fonction $\text{Card}(_)$ qui, à tout vecteur d'ensemble indexé par P , associe le marquage formé par la famille des cardinaux P -indexée.

Définition 3.10 (Cardinal d'un vecteur d'ensembles) Soit $m \in \text{obj}(\text{Set}^P)$ un objet de Set^P , on définit $(\text{Card}(m) : P \rightarrow \mathbb{N})$ la fonction qui, à chaque composante, associe son cardinal:

$$(\text{Card}(m) : P \rightarrow \mathbb{N}) = (\text{fn } p \Rightarrow \text{Card}(m(p)))$$

Pour montrer l'injectivité, il suffit de remarquer que le cardinal d'une traduction d'un marquage est ce marquage. En effet, le cardinal du segment initial de longueur n est bien n , exprimé sur un vecteur indexé par P on a bien:

$$M = \text{Card}(\text{Segs}(M)) \quad (3.3)$$

Ceci suffit pour montrer que deux marquages ont même image ssi ils sont identiques.

La surjectivité repose sur une application à chaque composante du théorème de Cantor-Berstein qui porte peut-être un nom antérieur pour ce cas non-transfini.

$$\text{Card}(s) = \text{Card}(s') \iff [s] = [s'] \quad (3.4)$$

Pour tout vecteur s d'ensembles indicés par P , l'équation 3.3 appliquée à $\text{Card}(s)$ nous donne $\text{Card}(s) = \text{Card}(\text{Segs}(\text{Card}(s)))$; de l'équation 3.4 on déduit:

$$[s] = [\text{Segs}(\text{Card}(s))] = \text{Sommets}(\text{Card}(s))$$

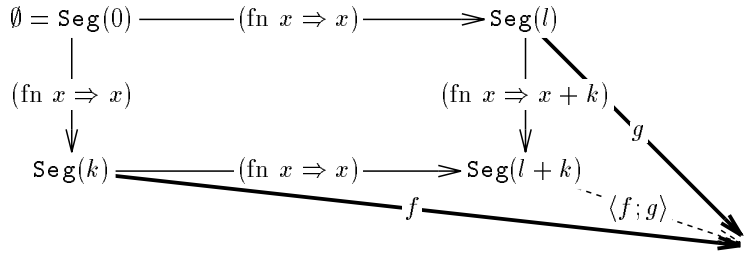
Ce qui exprime que toute classe est bien image par $\text{Sommets}(_)$, donc la surjectivité de cette fonction, qui est donc une bijection.

◇ **Bijection des arêtes** On définit la fonction $\text{Arêtes}(_)$ qui, à toute arête de $\mathbf{p}(R)$, associe une arête de $\mathbf{g}(\text{Trad}(R))$; comme l'ensemble des arêtes de $\mathbf{p}(R)$ est un co-produit (comme celui de $\mathbf{g}(\text{Trad}(R))$) on donne deux équations de traduction relative à chaque composante de la "somme disjointe" dans chacun des deux graphes de comportement.

$$\text{Arêtes}(\langle t; M \rangle) = [\langle t; (! : \emptyset \rightarrow \text{Segs}(M)) \rangle]$$

La bijection des arêtes repose sur deux arguments, chacun pour une des lignes. Pour les applications de règles, comme le contexte est initial, l'équivalence de T-diagramme d'application n'est pas contrainte par le morphisme d'application donc tout isomorphisme convient. La preuve de la bijection des sommets s'applique donc. Ici, l'hypothèse d'interface initiale est centrale.

FIG. 3.3 – Un pushout avec des segments



◇ **Préservation de $\text{src}(-)$ et $\text{tgt}(-)$** Cette question est symétrique pour $\text{src}(-)$ et pour $\text{tgt}(-)$, donc ce qui suit est écrit pour $\text{src}(-)$ et s'écrit par simple substitution pour $\text{tgt}(-)$.

$$\text{Sommets}(\text{src}(\langle t; M \rangle)) = \text{src}(\text{Aretes}(\langle t; M \rangle)) \quad (3.5)$$

En vertu de nos définitions, cette ligne est équivalente à la propriété ci dessous

Propriété 3.3 (Préservation des sources d'applications) $\text{Segs}(\bullet t \oplus M)$ est un résultat du pushout de $(! : \emptyset \rightarrow \text{Segs}(\bullet t))$ et de $(! : \emptyset \rightarrow \text{Segs}(M))$

Notre catégorie de travail est co-complète, le pushout se réalise sur chaque composante, le morphisme universel est construit en prenant le morphisme universel sur chaque composante. Cette preuve se fait donc en deux étapes, d'abord composante par composante puis en montrant que des pushouts sur chaque composante forme un pushout du vecteur d'ensemble.

Lemme 3.1 (Pushout avec des segments) $\text{Seg}(x + y)$ est un résultat du pushout de $(! : \emptyset \rightarrow \text{Seg}(x))$ et de $(! : \emptyset \rightarrow \text{Seg}(y))$

Ce pushout est un co-produit car l'élément initial en est l'interface. Cette remarque est le point de départ de la sous-section suivante. Pour l'instant, considérons la figure 3.3 qui donne deux injections qui, jointes, sont surjectives ayant pour cible commune $\text{Seg}(x + y)$. Ceci caractérise le co-produit dans **Set** [Löw93]. L'analogie entre cette construction et la concaténation de chaînes peut faciliter sa perception.

La preuve du deuxième pas est donnée par le fait que le pushout puisse se décomposer en ses composantes. Le fait que ce pushout par composantes commute ne pose pas de difficulté, la flèche universelle étant donnée par la famille des flèches universelles sur chaque composante. Ceci clôt notre démonstration pour les grammaires à double pushout.

Autres catégories "équivalentes" On peut réaliser cette démonstration avec au moins trois autres catégories de travail qui donneront le même résultat même si elles ne sont pas strictement équivalentes à $\text{Set}^{\mathbf{P}}$.

- $\langle P; \emptyset; \emptyset; \emptyset \rangle \Rightarrow \text{Set}$: Un diagramme dont la forme est composée uniquement de sommets donne une construction syntaxiquement équivalente à $\text{Set}^{\mathbf{P}}$. On peut remarquer qu'il suffit d'"oublier" les opérateurs pour qu'une grammaire d'algèbres unaires (Cf. 1.4) soit un réseau de Petri Place-Transition. C'est par ce type de construction que l'on prouve la non-accessibilité du graphe donné dans la partie 1.5 pour la signature des graphes dirigés;

- la catégorie des flèches de **Set** dont la cible est P . Cette construction pourrait être qualifiée d'ensemble typé (i.e. *labelled set* de [CM92]). On n'a pas équivalence des catégories avec \mathbf{Set}^P . On passe de celle-ci à \mathbf{Set}^P en prenant les ensembles réciproques de chaque élément de P . Mais pour revenir, il faut faire attention aux conflits de noms... C'est l'exemple le plus simple sur lequel explorer le problème de "l'équivalence" avec les graphes typés;
- la catégorie des flèches de **dir** dont la cible est $\langle P; \emptyset; \emptyset; \emptyset \rangle$, la structure originale [CM92]. Nous avons pu nous simplifier l'expression parce que nous n'avons plus la contrainte de la catégorie de travail et prendre ainsi une catégorie beaucoup plus élémentaire que celle-ci.

Cette partie permet pragmatiquement d'envisager la Logique de Réécriture et les Grammaires à Double-Somme comme des alternatives de plus haut niveau que les réseaux de Petri P/T avec exactement les mêmes fonctionnalités d'analyses. Une première application de taille réelle est donnée au début du chapitre IV avec le cas d'étude.

Cette double extension permet d'envisager un certain nombre d'applications pour réduire les représentations de l'espace des états accessibles. Chacun des formalismes, par ces caractéristiques propres, permet d'envisager des réductions significatives de cet espace d'états: pour les grammaires de graphes, la notion d'isomorphisme, et du côté de la Logique de Réécriture, la concurrence définie comme une congruence de transformation qui peuvent se déplier dans leurs différents entrelacements.

3.4 Domaines des extensions réciproques

Cette section cherche à caractériser les systèmes qui sont équivalents à des réseaux de Petri Place/Transition. Il nous paraît très important de caractériser à part les systèmes équivalents à des réseaux de Petri Place/Transition en raison des capacités d'analyse associées à ces réseaux. Pour la logique de réécriture, la caractérisation nous fournit une définition compacte de Réseaux de Petri Haut-Niveau, définie justement par cette propriété d'extension. Pour les Grammaires à Double-Somme Amalgamée, nous raffinons la caractérisation de [CM92] en étendant le type de structures pouvant donner lieu à une extension en réseaux de Petri Place/Transition.

3.4.1 Logique de réécriture

Maintenant, nous allons appliquer notre démarche et donc caractériser les théories de réécriture pour lesquelles la fonction d'extension vers les réseaux de Petri P/T est définie. En présence de variables libres à droite, le système est infiniment branché, aucun réseau de Petri P/T (où P, T sont finis) ne peut rivaliser... Notre caractérisation suit fidèlement l'usage des réseaux de Petri Haut-Niveau: un multi-ensemble de tuples de profondeur 1. Nous rajoutons une contrainte propre à notre codage: absence de variables de sorte **soupe**.

Soit **Refs** un ensemble de sortes et **Motifs** un ensemble de motifs. Soit $(H(-) : \mathbf{Motifs} \rightarrow \mathbf{Refs}^*)$ une fonction qui, à chaque motif, associe un mot de sorte de référence. Soit $\mathbf{Constants}(-)$ une fonction qui à chaque sorte de référence associe un ensemble de constantes disjoints deux à deux.

On construit la signature $\langle \mathbf{Refs}; \mathbf{Motifs}; H(-); \mathbf{Constants}(-) \rangle$ comme suit :

Sortes : $\{\mathbf{soupe}\} \uplus \mathbf{Refs}$

$$\text{Opérateurs : } \left\{ \begin{array}{ll} \text{rien :} & \epsilon \multimap \text{ soupe} \\ _ \oplus _ : & \text{ soupe } \times \text{ soupe } \multimap \text{ soupe} \\ \forall m \in \mathbf{Motifs} & \\ m : & H(m) \multimap \text{ soupe} \\ \forall r \in \mathbf{Refs}, \forall x \in \mathbf{Constants}(r), & \\ x : & \epsilon \multimap r \end{array} \right.$$

Pour illustrer que ce type de signature correspond bien aux réseaux de Petri de Haut-Niveau, nous montrons que la signature de notre exemple du gala des philosophes est de cette forme :

$$\begin{aligned} \Sigma(P, B) &= \langle \{\mathbf{Philosophe}, \mathbf{Baguette}\}; \{(_ \hookrightarrow _), \mathbf{Libre}(_), \mathbf{Oisif}(_), \mathbf{Encas}(_, _, _)\}, \\ & \quad (\text{fn } (_ \hookrightarrow _) \Rightarrow \mathbf{Philosophe} \times \mathbf{Baguette} \\ & \quad | \quad \mathbf{Libre}(_) \Rightarrow \mathbf{Baguette} \\ & \quad | \quad \mathbf{Oisif}(_) \Rightarrow \mathbf{Philosophe} \\ & \quad | \quad \mathbf{Encas}(_, _, _) \Rightarrow \mathbf{Philosophe} \times \mathbf{Baguette} \times \mathbf{Baguette}), \\ & \quad (\text{fn } \mathbf{Philosophe} \Rightarrow P \\ & \quad | \quad \mathbf{Baguette} \Rightarrow B) \rangle \end{aligned}$$

La signature que l'on a utilisée pour décrire le problème des philosophes était donc une signature de multi-ensemble de tuples.

Définition 3.11 (Système extensible) *Un ensemble de règles est extensible ssi les ensembles de constantes sont finis et les règles ne contiennent pas de variables de sorte **soupe**.*

Propriété 3.4 (Extension réciproque) *A tout système extensible R sur une signature $\langle \mathbf{Refs}; \mathbf{Motifs}; H(_); \mathbf{Constants}(_) \rangle$ on peut associer un réseau de Petri Place/Transition.*

- On prend comme ensemble de places les termes de sorte **soupe** ne contenant pas l'opérateur $_ \oplus _$. Tout marquage peut s'exprimer comme un multi-ensemble de ce type de termes. Le fait que cet ensemble soit fini est une conséquence de l'hypothèse de finitude des ensembles de constantes.
- Pour chaque transition, on construit toutes les transitions à une substitution près. Par construction, ces substitutions ne peuvent associer que des variables de sortes dans **Refs** à des constantes de l'ensemble de constantes associé. Pour chaque substitution possible on obtient donc bien une paire de multi-ensemble de termes de sorte **soupe** ne contenant pas d'opérateur $_ \oplus _$. La finitude de cet ensemble est une conséquence de l'hypothèse sur l'absence de variable de sorte **soupe**.
- La préservation de **src**($_$) et **tgt**($_$) est une conséquence directe de la construction.

Nous sommes passés assez rapidement car, mis à part l'expression en Logique de Réécriture, le reste est très étudié [Jen96, Gen86]. La combinatoire de cette transformation est énorme, et valide la conception selon laquelle les réseaux à base de “tuples plats” sont une notation compacte pour exprimer des réseaux Place/Transition. Une conséquence importante est que toutes les constructions structurelles (invariants de place, de transitions...) et les propriétés (décidabilité de l'accessibilité) restent valides dans ce contexte.

3.4.2 Grammaires à double-somme

Pour les grammaires à double-pushout, par contre, il n'existe rien d'équivalent dans la littérature. Pourtant nous avons donné une caractérisation du domaine de la réduction dans l'introduction de notre partie. Elle comporte deux parties:

Règles à contexte initial Ceci va transformer le pushout en co-produit. La dérivation d'une règle à contexte initial est ainsi équivalente à la congruence avec le monoïde commutatif formé par les classes d'objets et le co-produit. La structure résultante est un monoïde commutatif, mais il n'est pas forcément libre.

Co-produits libre Cette hypothèse nous garantit que la structure résultante est un monoïde commutatif. Nous montrerons cette propriété pour les graphes étendus [Löw93], l'intuition étant que la dérivation d'une règle à contexte initial va prélever en bloc la source et ajouter en bloc la destination. En conséquence, les composantes connexes forment l'alphabet dans lequel écrire de tels systèmes.

Nous allons donc d'abord montrer que le co-produit forme un monoïde avec les classes d'objets puis, avec l'hypothèse de liberté du co-produit nous pourrions conclure. Cette hypothèse sera prouvée pour les graphes étendus [Löw93] dans un second temps.

Extension

Dans un premier temps, nous considérons notre caractérisation (i.e. coproduits libres) vérifiée et nous montrons l'extension d'un tel système dans les réseaux de Petri Place/Transition.

Co-produit sur les classes On étend le co-produit comme une opération sur les classes d'objets $\coprod_B^A = \coprod_{[B]}^{[A]}$. Pour pouvoir poser cette définition il faut montrer que \coprod forme un opérateur sur les classes d'équivalence; il nous faut donc montrer que $A \sim A'$ et $B \sim B'$ impliquent $\coprod_B^A = \coprod_{B'}^{A'}$. Ceci ne pose pas de difficulté.

Propriété 3.5 (Monoïde commutatif du co-produit) *Soit \mathbf{X} une catégorie co-complète, et \emptyset un élément initial de cette catégorie. $([\mathbf{X}], \coprod, [\emptyset])$ est un monoïde commutatif.*

Le fait que ce soit un monoïde est une conséquence de l'existence de la généralisation du co-produit à n (pour l'associativité), du fait qu'un co-produit \coprod_A^B est isomorphe à \coprod_B^A (commutativité) et enfin que la classe de l'objet initial est élément neutre.

Catégorie avec des co-produits libres. Soit \mathbf{X} une catégorie. Elle est dite avec des co-produits libres ssi le monoïde du co-produit est isomorphe au monoïde libre engendré par un ensemble de classes d'objets appelés les composantes. On note $\mathbf{Abstract}(_)$ cet isomorphisme de monoïde

$$(\mathbf{Abstract}(_) : ([\mathbf{X}], \coprod, [\emptyset]) \rightarrow \mathbb{N}^{\text{Composantes}(\mathbf{X})}, \oplus, \vec{0})$$

Un objet x est dit finiment engendré ssi $\sum_{c \in \text{Composantes}(\mathbf{X})} \mathbf{Abstract}([x])(c) < \infty$.

Définition 3.12 (Caractérisation) $\langle \mathbf{X}; \mathcal{L} \rangle$ est dit simplifiable ssi \mathbf{X} est une catégorie avec des co-produits libres, et $\langle \mathbf{X}; \mathcal{L} \rangle$ est un système à interface initiale

(i.e. pour tout $l \in L$, \bar{l} est initial dans \mathbf{X}), finiment engendré (i.e pour tout $l \in L$, $\bullet l$ (resp. l^\bullet) est finiment engendré).

Comme pour la réduction de la logique de réécriture l'extension inverse utilise la sémantique connexe de comportement. La raison est similaire, c'est le système de règle et l'état initial qui bornent conjointement l'ensemble des générateurs, le futur ensemble de places. Nous allons donc montrer l'extension de notre caractérisation dans l'ensemble des réseaux de Petri dotés d'un ensemble infini de places. On procédera à l'utilisation de la sémantique d'accessibilité dans un deuxième temps.

Equivalence des états La fonction $\mathbf{Abstract}(_)$ fournit cette équivalence.

Extension A tout système appartenant à notre caractérisation, nous associons le réseau de Petri infini ci-dessous:

$$R_G = \langle P; L; (\text{fn } l \Rightarrow \mathbf{Abstract}(\bullet l); (\text{fn } l \Rightarrow \mathbf{Abstract}(l^\bullet))) \rangle$$

Equivalence des Arêtes On construit la fonction $\mathbf{Tabs}(_)$ qui, à toute flèche de $\mathbf{g}(\langle \mathbf{X}; \mathcal{L} \rangle)$, associe une flèche de $\mathbf{p}(R_G)$:

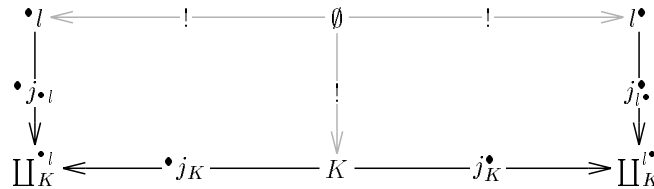
$$\mathbf{Tabs}(\langle l; (! : \emptyset \rightarrow K) \rangle) = \langle l; \mathbf{Abstract}(K) \rangle$$

L'équivalence des arêtes est une conséquence de l'équivalence des états, et surtout du fait que l'on a pris un contexte initial. L'information contenue dans le morphisme d'instanciation est ainsi intégralement contenue dans sa cible.

Préservation des sources et cibles C'est une conséquence de la caractérisation concernant l'interface initiale qui rend la source et la cible des co-produits du pré-graphe et du contexte. La propriété que l'on cherche à prouver ne pose pas de difficulté pour les identités où le pre-graphe est initial; en conséquence notre preuve consiste à prouver les équations suivantes.

$$\begin{aligned} \mathbf{Abstract}(\text{src}(\langle l; (! : \emptyset \rightarrow K) \rangle)) &= \mathbf{Abstract}(\bullet l) \oplus \mathbf{Abstract}(K) \\ \mathbf{Abstract}(\text{tgt}(\langle l; (! : \emptyset \rightarrow K) \rangle)) &= \mathbf{Abstract}(l^\bullet) \oplus \mathbf{Abstract}(K) \end{aligned}$$

Ceci est prouvé en deux pas, d'abord concrètement par le diagramme suivant qui illustre la simplification opérée par l'utilisation d'une interface initiale



Ensuite, en utilisant les différentes équations déjà prouvées, on obtient la preuve

recherchée.

$$\begin{aligned}
 \text{Abstract}(\text{src}([\langle l; (! : \emptyset \rightarrow K) \rangle])) &= \text{Abstract}(\coprod_{K}^{\bullet l} [\cdot]) \\
 &= \text{Abstract}(\coprod_{[K]}^{[\cdot l]}) \\
 &= \text{Abstract}([\cdot l]) \oplus \text{Abstract}([K])
 \end{aligned}$$

C'est la dernière ligne qui fait intervenir l'hypothèse de "liberté du co-produit". La démonstration est la même pour $\text{tgt}(-)$, ce qui conclut notre démonstration.

Passage à la sémantique d'accessibilité C'est ici que l'hypothèse "finiment engendré" va nous servir. Soit P l'ensemble des composantes qui servent à la définition des règles. La classe d'objets qui sert de sommet de départ n'est pas forcément écrite uniquement avec ses composantes, mais on peut retrancher à la sémantique connexe l'excédent systématiquement, la difficulté étant que l'on ne sait qu'ajouter !

Soit $[x]$ le sommet de départ. On peut décomposer de façon unique $[x]$ en $[x] = [x_P] + [r]$ où pour toute composante c de P , $\text{Abstract}([r])(c) = 0$ et pour toute composante \bar{c} de $\text{Composantes}(X) - P$, $\text{Abstract}([x_P])(\bar{c}) = 0$. L'extension nous donne que la sémantique connexe de $\langle \langle \mathbf{X}; \mathcal{L} \rangle; [x_P] \rangle$ est isomorphe à $\langle R_G; \text{Abstract}([x_P]) \rangle$. Maintenant il nous faut montrer que $\mathbf{g}^\circ(\langle \langle \mathbf{X}; \mathcal{L} \rangle; [x_P] \rangle)$ est isomorphe à $\mathbf{g}^\circ(\langle \mathbf{X}; \mathcal{L} \rangle; [x])$.

On va d'abord montrer que la tranche $\mathbf{g}^\circ(\langle \langle \mathbf{X}; \mathcal{L} \rangle; [x_P] \rangle)$ s'injecte dans $\mathbf{g}^\circ(\langle \mathbf{X}; \mathcal{L} \rangle; [x])$ par monotonie, puis on montrera qu'il n'existe pas de flèches sortantes de l'image.

Propriété 3.6 (Monotonie) *Soit \mathbf{X} une catégorie co-complète, à toute classe d'objets $[x]$, on peut associer un morphisme $\mathcal{M}([x])$ de graphe réflexif injectif dans le graphe abstrait de dérivation.*

Ce morphisme ajoute $[x]$ (par co-produit) à tous les sommets et $[x]$ à tous les contextes d'applications. Le fait que ce soit un morphisme ne pose aucune difficulté. C'est cette propriété que l'on a utilisée dans la présentation des figures relatives à l'exploration des états accessibles du chapitre II. On a représenté en pointillé l'injection du graphe des états accessibles à deux paires dans celui donné, à trois paires.

Etendre cette propriété aux T-diagrammes d'application se fait sans difficulté:

$$\mathcal{M}([x])(\langle t; (\mathbf{m} : \bar{t} \rightarrow K) \rangle) = \langle t; (\mathbf{j}_\mathbb{N} \circ \mathbf{m} : \bar{t} \rightarrow K + x) \rangle$$

Le fait qu'aucune flèche ne sorte de l'image de $\mathbf{g}^\circ(\langle \langle \mathbf{X}; \mathcal{L} \rangle; [x_P] \rangle)$ par $\mathcal{M}([r])$ est une conséquence du fait que P donne l'ensemble des composantes citées dans les règles, dont toutes les composantes de $[r]$ sont préservées par l'application.

Graphes et connexité

La catégorie des diagrammes de forme D ($D \Rightarrow \mathbf{Set}$) admet des co-produits libres. Les générateurs de ce monoïde sont les classes de composantes connexes. Pour définir ce qu'est une composante connexe, il nous faut étendre la notion de chaîne [Ber77]. Une chaîne est une séquence d'éléments de différentes sortes qui peuvent reliés en séquence par des opérateurs, le sens de l'opérateur étant variable. On

procède en deux phases, on définit d'abord un mot sur le co-produit des ensembles images des sommets, puis on fournit une expression pour retrouver les éléments originaux.

Définition 3.13 (Co-produit fini[Pie91]) *Un co-produit d'une famille $\{A_i\}_{i \in I}$ d'objets de \mathbf{X} indexé par un ensemble I est un objet $\coprod_{i \in I} A_i$ et une famille de flèches d'injection $\{(j_i : A_i \rightarrow \coprod_{i \in I} A_i)\}_{i \in I}$ telle que pour tout objet c et pour toute famille de flèches $\{(f_i : A_i \rightarrow c)\}_{i \in I}$, il existe une unique flèche $((f_i)_{i \in I} : \coprod_{i \in I} A_i \rightarrow c)$ telle que $\forall i \in I, (f_i)_{i \in I} \circ j_i = f_i$, i.e que la famille suivante de diagrammes commute :*

$$\left\{ \begin{array}{ccc} A_i & \xrightarrow{j_i} & \coprod_{i \in I} A_i \\ & \searrow f_i & \downarrow (f_i)_{i \in I} \\ & & c \end{array} \right\}_{i \in I}$$

Définition 3.14 (Connexité) *Soit $(G : D \rightarrow \mathbf{Set})$ un diagramme de forme D .*

- *Soit w un mot de $\coprod_{s \in \text{sommets}_D} G_s$, le co-produit (i.e. l'union disjointe) des ensembles associés aux sommets. On note $s(n)$ l'élément unique de sommets_D tel qu'il existe $\overline{w}(n) \in s(n)$ vérifiant $w(n) = j_{s(n)}(\overline{w}(n))$.*
- *Un mot d'éléments w est une chaîne ssi pour tout $n \in [0..|w| - 2]$, il existe une arête $(f(n) : s(n) \rightarrow s(n+1))$, vérifiant $f(n)(\overline{w}(n)) = \overline{w}(n+1)$.*
- *G est connexe ssi pour toute paire d'éléments (a, b) de $\coprod_{s \in \text{sommets}_D} G_s$, il existe une chaîne w tel que $w(0) = a$ et $w(|w| - 1) = b$.*

Soit $\mathbf{Composantes}(D \Rightarrow \mathbf{Set})$ l'ensemble des classes de diagrammes de forme D connexes.

Tout graphe peut être écrit comme l'union de ses composantes connexes. La propriété de connexité implique que cette union est disjointe, donc forme un co-produit [RB88]. On déduit de cette construction qu'il existe une fonction B des diagrammes de forme D dans $\mathbf{Composantes}(D \Rightarrow \mathbf{Set})$.

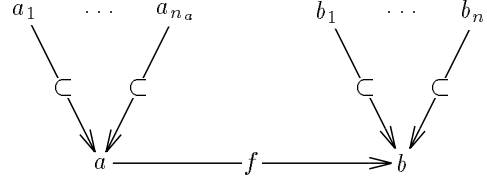
C'est dans l'extension de cette fonction aux classes de graphes que se situe la preuve centrale, et propre à notre problématique.

Propriété 3.7 (Importation des isomorphismes) *l'existence de $(f : a \rightarrow b)$ un isomorphisme de graphe implique que a et b ont même image par B .*

◇

On prouve cette propriété en montrant que l'on peut associer deux à deux, par des isomorphismes, les composantes connexes de a et de b . Nous allons d'abord montrer qu'un morphisme implique de façon unique l'existence de morphismes reliant deux à deux les décompositions; le fait que ce soit un isomorphisme nous permettra de statuer sur l'existence d'une collection bijective d'isomorphismes ce qui nous permettra de conclure.

Lemme 3.2 (Importation des morphismes) *l'existence d'un morphisme $(f : a \rightarrow b)$ implique l'existence et l'unicité d'une fonction $(\overline{f} : [1..n_a] \rightarrow [1..n_b])$ et d'une famille de fonctions indexées par I telles que $\{(\phi_i : A_i \rightarrow B_{\overline{f}(i)})\}_{i \in I}$.*



L'argument central est le suivant et forme une caractérisation du co-produit dans **Set** : les inclusions formant les co-produit, jointes, sont surjectives [Löw93].

Soit $i \in [1..n_a]$, on obtient la fonction en prenant un élément α_i dans l'ensemble a_i . Soit x_i son image par l'inclusion associée suivie de f . x_i appartient donc à l'image d'un seul $b_{\overline{f}(i)}$, soit β_i l'élément de $b_{\overline{f}(i)}$ qui a pour image x_i par l'inclusion canonique.

Prenons maintenant un autre élément α'_i de a_i , soit w la chaîne qui le relie à α_1 . On montre par induction sur la longueur de la chaîne que tous les éléments de cette chaîne peuvent être envoyés par un morphisme sur $b_{\overline{f}(i)}$.

Supposons la propriété vraie pour tout élément de la chaîne jusqu'à l'indice k . Soit β_i^l les éléments images de $\overline{w}(l)$ pour tout $l \in [0..k]$. Considérons l'image β_i^{k+1} de $\overline{w}(k+1)$ par l'inclusion suivi de f . La propriété de préservation nous indique, quel que soit le sens du morphisme que β_i^{k+1} doit appartenir à la même composante que β_i^k , son prédécesseur. Donc β_i^k appartient à $b_{\overline{f}(i)}$.

◇

On déduit de cette propriété que l'on peut construire la fonction **Abstract**($_$). Il nous faut montrer qu'**Abstract**($_$) est bien un morphisme. **Abstract**($_$) envoie bien le graphe vide sur l'élément neutre, il reste à montrer que

$$\mathbf{Abstract}\left(\coprod_{[b]}^{[a]}\right) = \mathbf{Abstract}([a]) \oplus \mathbf{Abstract}([b])$$

On prend a et b des représentants distincts des classes $[a]$ et $[b]$ et leur co-produit \coprod_b^a élément de $\coprod_{[b]}^{[a]}$. Le fait que le co-produit de leurs décompositions soit isomorphes à leur co-produit est une conséquence de l'associativité de l'union, qui permet de dire qu'ils sont égaux.

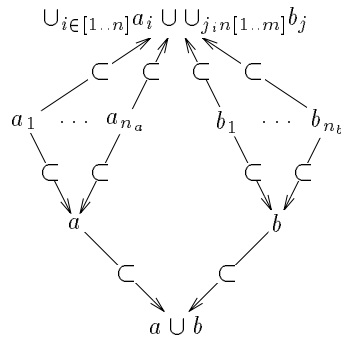
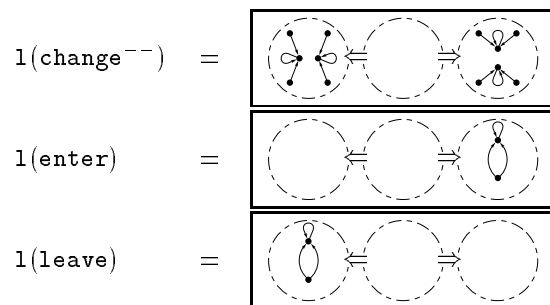
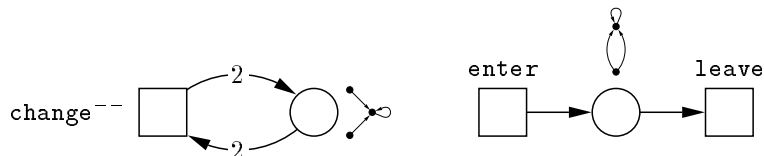


Illustration et perspective Ceci termine notre preuve de la réduction. Pour l'illustrer avec un exemple concret, considérons le système de règles suivant dans les graphes dirigés construit avec toutes les règles à contexte vides définies dans ce mémoire:



Nous avons prouvé que cette grammaire de graphe avait le même comportement par entrelacement et avec état initial que le réseau de Petri suivant, où l'on étiquette les places par les composantes connexes cités dans les règles:



Maintenant, si nous considérons ce réseau de Petri, il dispose d'une sémantique plus vaste que celle par entrelacement si on admet la concurrence. Peut-on réaliser une sémantique équivalente en grammaires de graphes? La réponse est positive si on considère le séquent suivant de congruence basé sur le co-produit:

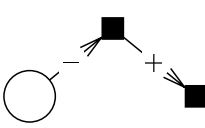
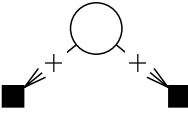
Congruence :

$$\frac{[L_0] \rightarrow [R_0], [L_1] \rightarrow [R_1]}{[\coprod_{L_1}^{L_0}] \rightarrow [\coprod_{R_1}^{R_0}]}$$

Un terme de preuve qui spécifie l'arrivée concurrente de deux philosophes est par exemple:

$$\coprod_{\langle \text{enter}, (C:\emptyset \rightarrow \emptyset) \rangle} \langle \text{enter}, (C:\emptyset \rightarrow \emptyset) \rangle$$

On retrouve ici une expression de la concurrence comme une "congruence" de transformation: le co-produit de T-diagrammes réalisé dans le graphe abstrait de dérivation (cf. 2.2). L'équivalence repose sur l'isomorphisme des monoïdes sous-jacents et les interfaces initiales. On peut donc rajouter une ligne à notre tableau donné au début du chapitre II (Figure 2.1) qui énumérait pour les différents formalismes les versions opérationnelles et dénotationnelles, cette ligne concerne la forme prise par les sémantiques concurrentes dans les deux contextes :

| | Opérationnel | Dénotationnel |
|-----------------|---|--|
| Schéma abstrait |  |  |
| Concurrence | Classes de séquences[Maz87] | Congruence de transformations[MM90, Mes92] |

Perspectives: Parvenir à relier ces deux formes de représentation de la concurrence est une perspective importante dans cette problématique. La question est déjà adressée dans [MT99] mais d'autres pistes sont possibles; en particulier en remarquant qu'avec une sémantique concurrente, le co-produit (ici deux copies disjointes) de deux réseaux a pour comportement le produit catégorique des comportements.

Il ne manque qu'un embranchement dans notre système d'équivalences, une passerelle directe entre les deux formalismes sur un domaine incluant le bal des philosophes. Une comparaison est possible entre les signatures de multi-ensembles et les diagrammes de forme B où B est bipartite. La séparation du graphe en deux ensembles correspond aux motifs (uniquement source) et aux références (uniquement cible). Mais cette intuition se heurte à une incompatibilité des deux formalismes: les Grammaires de Graphes (au sens de [Löw93]) considèrent un isomorphisme global sur la structure, alors que la structure de multi-ensemble est comparable à un isomorphisme (cf. 3.3) mais uniquement sur les arêtes. En conséquence, l'extension est impossible avec ces sémantiques. On ne peut pas affaiblir celle des grammaires de graphes en essayant de conserver les références par une dynamique concrète particulière car on perdrait l'injection dans le graphe abstrait. Bref, cette équivalence n'en est pas une avec nos définitions.

En outre, utiliser ces formalismes pour les cas où des éléments sont éphémères reste notre priorité. Dans cet ordre de perspective, peut-on trouver une théorie de réécriture qui soit équivalente au "bal des philosophes" en double-somme? La réponse est positive en considérant un opérateur "cyclique" ($cycle(x.1) \approx cycle(1.x)$) où l'on parvient à coder par une localité dans une classe de termes la caractéristique topologique du cycle et à réécrire. Dans ce contexte, les perspectives de recherche intéressantes se situent au niveau des passerelles que l'on peut établir entre des localités dans des classes de termes et les caractérisations topologiques de classes de graphes.

Chapitre

4

Implémentation et Perspectives

Ce chapitre détaille les aspects opérationnels afférents à nos développements formels. Nous avons réalisé plusieurs prototypes et nous n'en développerons qu'un seul ici, basé sur MAUDE [CDE⁺99]. Nous avons aussi développé un moteur de grammaires de graphes en ML [RB88] mais il n'est pas doté de recherche automatique de morphismes ce qui le limite à une simulation aidée par l'utilisateur et empêche pour l'instant la construction du graphe des états accessibles que notre théorie a proposé. Enfin nous disposons d'un prototype en JAVA permettant de visualiser des graphes dynamiques et qui positionne les nœuds à l'aide d'heuristiques dynamiques, un résultat de ces heuristiques est donné dans la figure 1.2.

Essentiellement le problème adressé dans cette partie sera de rendre opérationnelle notre approche en logique de réécriture. Cette approche permet d'utiliser cette logique pour spécifier des systèmes dont on analysera l'ensemble du comportement et non seulement la fonction partielle de calcul associée. En effet, la logique de réécriture dans sa sémantique originale est difficilement accessible en MAUDE ce qui semble attester du fait que la logique est postérieure à l'outil. Historiquement, la notion de réécriture est très liée à la notion de calcul, et c'est un des mérites du travail de [Mes92] que d'avoir montré comment cette approche contenait aussi des formalismes considérés comme différents, comme les réseaux de Petri ou la machine chimique.

Néanmoins cette précedence du calcul sur d'autre forme d'appréhension d'une dynamique est présente en MAUDE. Par défaut, pour une théorie de réécriture, MAUDE cherche à réduire un terme jusqu'à une forme normale. Si MAUDE en restait là, l'utilisation de ce moteur ne nous permettrait pas d'analyser une théorie avec les outils normalement appliqués aux réseaux de Petri ou aux algèbres de processus, qui reposent sur l'obtention du graphe des états accessibles, ou tout au moins sur la possibilité de réaliser un seul pas de réécriture. Les possibilités offertes par la réflexion permirent de dépasser la difficulté et d'obtenir la sémantique de base (sans concurrence), qui est celle pour laquelle on a montré les extensions.

Néanmoins, l'expression d'un simulateur, qui applique la stratégie de simulation des réseaux de Petri à toute théorie de réécriture nécessite un travail assez important.

Pour motiver l'utilisation de MAUDE comme outil de spécification d'application coopérative, nous avons détaillé la spécification de notre cas d'étude dans une version implémentable dans ce langage. Pour des raisons de lisibilité, les expressions parenthésées ont été écrites sous forme d'arbres, mais une version MAUDE de la spécification peut être produite à partir du même code.

La première section détaille une spécification de notre cas d'étude en logique de réécriture et détaille une caractérisation de signature orthogonale aux multi-ensembles de tuples : les arbres commutatifs. La deuxième section s'attache à montrer comment on peut réaliser un programme en MAUDE qui en explore la sémantique de simulation et ses limites actuelles.

4.1 Spécifier le cas d'étude

Où prend place notre cas d'étude ? Deux questions se posent qui correspondent à la caractérisation de sa position.

Pourquoi la réécriture ? Comme le souligne le texte donné en référence, le cas que doit explorer la spécification est le cas où un PC member soumet un article, ou avec les grammaires de graphes, ce genre de cas doit être prévu par le spécifieur. Cette assertion est suffisante mais il en est une autre plus pragmatique. On arrive avec ce cas d'étude à un niveau de complexité où la représentation textuelle devient indispensable pour pouvoir spécifier en un temps raisonnable ce type de structure. Pour donner un ordre de grandeur, nous avons passé une journée pour "écrire" notre spécification et une autre pour la "debugger", dessins compris (car ils sont générés automatiquement en \TeX). A moins de fournir une interface textuelle performante aux grammaires de graphes ceci est hors de portée dans des temps comparables, à titre d'exemple l'ensemble des dessins relatifs aux problèmes des philosophes a pris plus d'une semaine — mise au point comprise —.

Pourquoi pas des multi-ensembles de tuples ? A cause des phases de type *processus* qui nécessitent de détecter la terminaison et donc de réaliser d'une façon ou d'une autre un test à zéro inaccessible avec les multi-ensembles de tuples, puisqu'ils s'injectent dans les Réseaux Place/Transition (cf. 3.4).

En conséquence, nous sommes obligés pour réaliser notre cas d'étude de trouver une signature différente. A l'opérateur binaire $_ \oplus _$ de **soupe.soupe** dans **soupe**, on spécifie des opérateurs unaires de **soupe** dans **soupe** et pour chaque sorte de référence un ensemble de constantes.

Soit $\langle \mathbf{Motifs}; \mathbf{Refs}; \mathbf{Constants}(_) \rangle$ un tuple formé d'un ensemble de motifs, un ensemble de références, et une fonction qui à toute sorte de référence associe un ensemble de références. On construit la signature associée à ce tuple comme suit, avec les équations AC usuelles pour $_ \oplus _$ et **rien**:

Sortes : $\{\mathbf{soupe}\} \uplus \mathbf{Refs}$

$$\mathbf{Opérateurs} : \left\{ \begin{array}{lll} \mathbf{rien} : & \epsilon \multimap & \mathbf{soupe} \\ _ \oplus _ : & \mathbf{soupe} \times \mathbf{soupe} \multimap & \mathbf{soupe} \\ \mathbf{Motifs} : & \mathbf{soupe} \multimap & \mathbf{soupe} \\ \mathbf{Refs} : & r \multimap & \mathbf{soupe} \\ \forall r \in \mathbf{Refs}, \forall x \in \mathbf{Constants}(r), & & \\ x : & \epsilon \multimap & r \end{array} \right.$$

Nous avons appelé cette structure “arbre commutatif” car un terme peut s'interpréter comme un arbre dont les branches ne sont pas ordonnées et dont on ne maîtrise pas le degré : sauf à contrôler les admissions dans un “sous terme”, on ne fixe pas le nombre de “fils” d'un nœud. L'intérêt réside dans le fait que toute classe de termes a une représentation graphique univoque. Cette représentation graphique est un arbre dont les branches ne sont pas ordonnées à la différence d'un terme. La structure ainsi définie est donc orthogonale aux Réseaux de Petri de Haut-Niveau mais toujours dans le cadre de la Logique de Réécriture. Le contrôle des admissions va nous permettre de spécifier ce processus, mais avant de le réaliser, nous allons donner deux exemples simples permettant de se familiariser avec la structure tout en goûtant sa puissance d'expression.

4.1.1 Un premier exemple

Il nous faut donc montrer simplement comment réaliser le test à zéro.

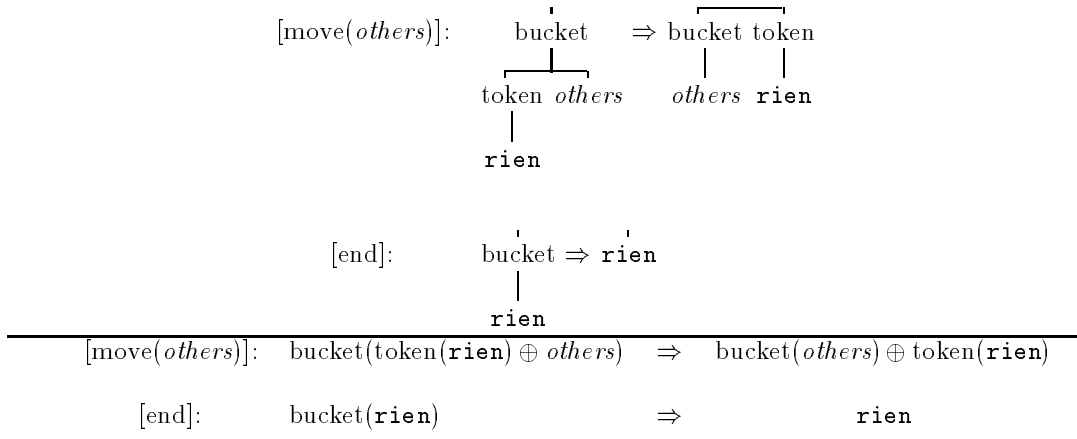
Considérons l'exemple suivant, où un “seau à jetons” se vide, et l'on détecte la terminaison de ce processus :

$$\text{Motifs} = \{\text{token}, \text{bucket}\} \quad \text{Refs} = \emptyset$$

L'ensemble de règles est décrit graphiquement et textuellement par la figure 4.1. Pour réaliser un système comparable avec des Réseaux P/T ou Haut-Niveau, il faut se munir d'un test à zéro.

Note: On peut remarquer que pour toute classe de terme prise comme état initial, le système termine dans un état où tous les jetons sont à la racine. Dans ce cas la terminaison globale n'est pas détectée par notre jeu de règle.

FIG. 4.1 – Détection de la terminaison avec des arbres commutatifs



Nous raffinons cet exemple (Figure 4.2) avec un seau qui se vide à la mode TCP, en incrémentant sa vitesse chaque fois qu'il peut émettre des jetons et la divisant par deux pour terminer (deux règles pour les cas pair et impair).

$$\text{Motifs} = \{\text{token}, \text{bucket}, \text{control}\} \quad \text{Refs} = \emptyset$$

Ceci pour souligner l'expressivité des systèmes à bases d'arbres commutatifs. Ce type de systèmes où le contenu d'une “place” influence le tir d'une transition peut être rapproché des réseaux dynamiques de [BO98]. Néanmoins réaliser un moteur d'inférence qui supporte ce type de règles n'est pas évident, c'est ce problème qui est adressé dans la section suivante consacrée à l'outil MAUDE.

FIG. 4.2 – Le “seau TCP à jeton” (*TCP token bucket*)

| | |
|---|--|
| $[\text{odd}(\text{others}, \text{speed})]:$ | $\text{control}(\text{speed} \oplus \text{others} \oplus \text{speed} \oplus \text{others} \oplus \text{token}(\mathbf{rien})) \oplus$ $\text{others} \oplus \text{others}$ |
| \Rightarrow | $\text{control}(\text{speed} \oplus \text{others}) \oplus$ $\text{others} \oplus \text{others}$ |
| $[\text{even}(\text{others}, \text{speed})]:$ | $\text{control}(\text{speed} \oplus \text{others} \oplus \text{token}(\mathbf{rien}) \oplus \text{speed} \oplus \text{others} \oplus \text{token}(\mathbf{rien})) \oplus$ $\text{others} \oplus \text{others} \oplus \text{token}(\mathbf{rien})$ |
| \Rightarrow | $\text{control}(\text{speed} \oplus \text{others} \oplus \text{token}(\mathbf{rien})) \oplus$ $\text{others} \oplus \text{others} \oplus \text{token}(\mathbf{rien})$ |
| $[\text{move}(\text{others}, \text{speed})]:$ | $\text{bucket}(\text{control}(\text{speed}) \oplus \text{speed} \oplus \text{others})$ |
| \Rightarrow | $\text{bucket}(\text{control}(\text{speed} \oplus \text{token}(\mathbf{rien})) \oplus \text{others}) \oplus \text{speed}$ |
| $[\text{end}(\text{speed})]:$ | $\text{bucket}(\text{control}(\text{speed}) \oplus \mathbf{rien})$ |
| \Rightarrow | \mathbf{rien} |

4.1.2 Procès de la conférence

Compte-tenu de la complexité du système de soumission — du câble aux écrans — le spécifier complètement n’est pas un objectif raisonnable. Il nous faut définir un niveau où nous allons arrêter notre investigation: le “grain” de notre spécification. Plutôt que de le fixer à un niveau arbitraire, nous allons proposer une approche par raffinement successif sans détailler les dessous formels d’une telle approche mais dans un objectif de rendre la présentation accessible. La contribution ici est de parvenir à décrire ce cas d’étude. Outre la puissance d’expression nécessaire (i.e. le test à zéro), le traitement des apparitions (i.e. variables libres à droite), on peut noter parmi les contributions à ce problème d’avoir doté notre caractérisation d’une forme graphique adéquate permettant de jongler avec des termes difficiles à appréhender sous forme parenthésée. Ici la perspective graphique est un moyen et non une fin, à la différence des Grammaires de Graphes.

Le processus de la conférence évacue le problème central : la décision. Il décrit la soumission, et le retour des versions finales. Ce qui fonde la décision est caché dans une variable de sorte **soupe**: *context*, qui indique que ce sont des éléments présents dans le descriptif de l’article qui forment cette décision sans détailler son pourquoi. Après cette décision on lui accole une autre variable, libre à droite, qui se nomme *decision* et qui pourra dans un raffinement futur contenir les motivations de cette décision, en plus des relectures.

Les motifs utilisés sont au nombre de 10. Six concernent le comportement de la conférence et quatre concernent le comportement des articles, le motif Status est le marqueur de progrès de la phase en cours.

$$\begin{aligned} \mathbf{Motifs} &= \{\text{CFP}, \text{Prepare}, \text{Accept}, \text{Reject}, \text{End}, \text{Status}\} \\ &\uplus \{\text{Article}, \text{Accepted}, \text{Final}, \text{Rejected}\} \end{aligned}$$

Nous utilisons deux types de références : la première concerne les personnes, la deuxième les contenus échangés. Pour illustrer sans remettre en cause la possibilité de considérer ses modes de communications comme unifiables (Cf. 1.1.3), les “identités” sont des adresses de type envoi (e-mail), les “contenus” sont des adresses de type publication, le problème de la sécurisation des échanges n’est pas modélisé.

$$\mathbf{Refs} = \{\text{Identity}, \text{Content}\}$$

La spécification est donnée sous forme graphique dans la figure 4.3. La double ligne horizontale sépare informellement les règles qui relèvent de la dynamique de la conférence des règles qui relèvent d’un article. Le comportement de la conférence

est décrit par une dynamique à base de seau à jetons donnée dans la partie supérieure: cette partie traite essentiellement du test à zéro et de sa spécification. Le comportement d'un article, plus complexe car réalisant les connections entre les différents éléments, est donné dans la partie inférieure.

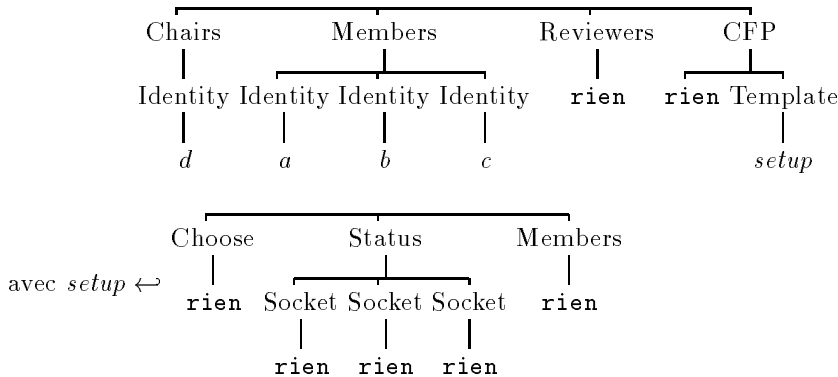
Cette première spécification fait l'impasse sur deux aspects essentiels, d'une part la gestion des relectures et la gestion de la réunion. Nous allons traiter ces questions par l'utilisation d'un raffinement de la première spécification, ce raffinement est basé sur le principe de l'inclusion de cette théorie dans la raffinée et une politique de gestion des conflits de noms sur les règles.

4.1.3 Le processus des relectures

Le processus des relectures concerne un aspect qui est concurrent au premier processus et dont nous verrons qu'il ne nécessite qu'un point de synchronisation. Ce processus, plus complexe que le premier, traite de la lecture depuis le choix par les membres jusqu'au retour des relectures et la réunion des membres du comité de lecture.

Pour spécifier le point de connexion, nous considérons une approche très élémentaire du raffinement, *une règle a' raffine explicitement une règle a si et seulement si il existe une substitution de a dans a' , ainsi tout terme de preuve impliquant a' peut être transformé en un terme de preuve impliquant a* . Enfin, nous considérons une approche par inclusions comme en MAUDE, la précédente règle de raffinement intervenant sur les conflits de nom entre transitions.

Raffinement implicite Ceci est une extension d'un "raffinement implicite" qui peut intervenir lorsque l'on a des variables qui préemptent un contexte pouvant être vaste. L'exemple de raffinement implicite est donné par la règle "submit". Considérons l'état initial suivant où a, b, c et d sont des termes clos de sorte **Entier**:



Tout tir de "submit" verra la substitution assigner comme valeur à: *infos* un terme qui est une substitution de *setup*.

Raffinement explicite Le point de synchronisation est la fin "maîtrisée" du processus de relecture: l'envoi aux relecteurs. Le comportement est donné dans la figure 4.4. Comme indiqué, ce comportement contient un raffinement de "Prepare". On a spécifié ici qu'un papier était "prêt" lorsque l'on connaissait ses relecteurs. La substitution pour le raffinement spécifie cela :

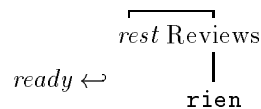
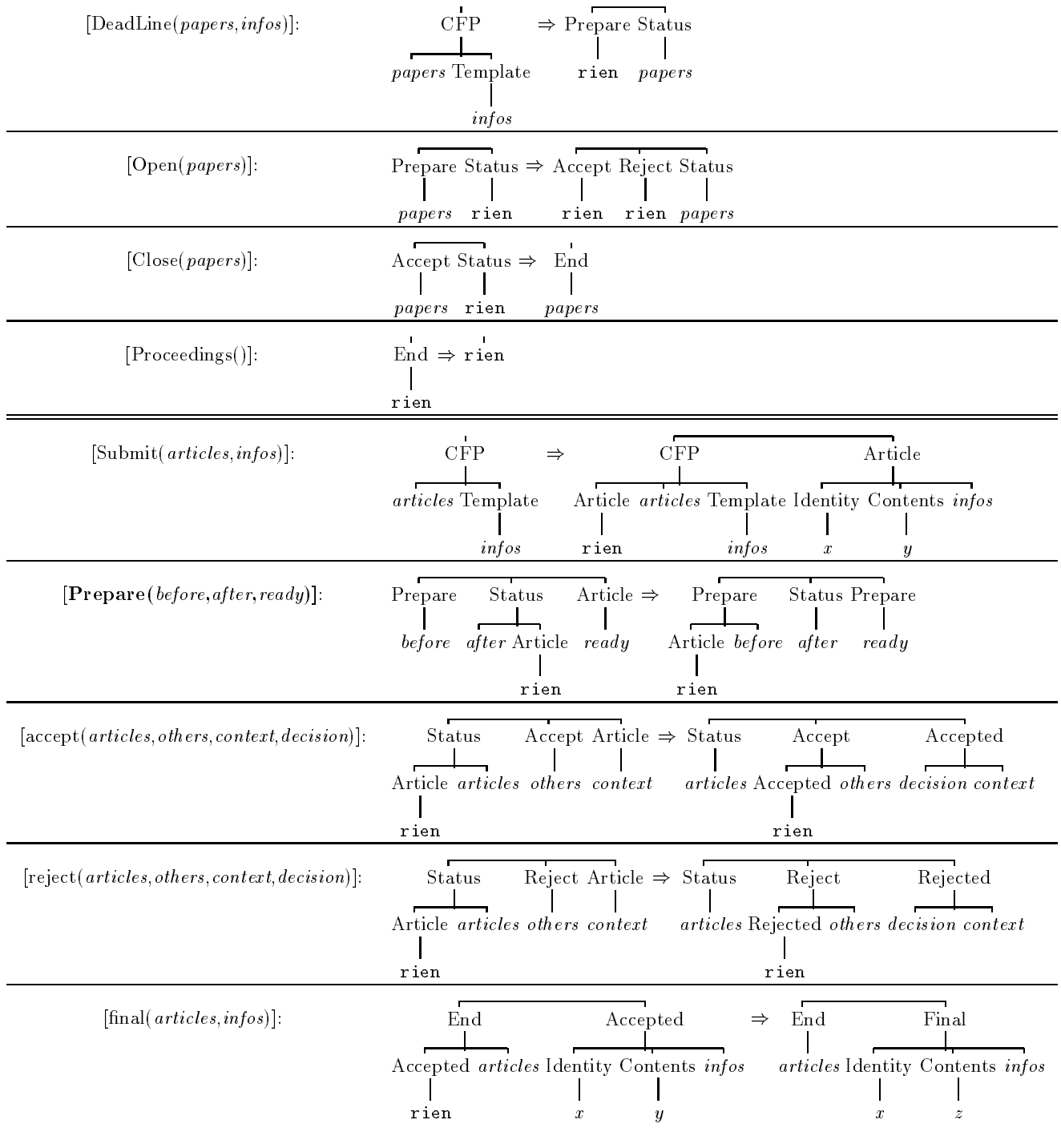


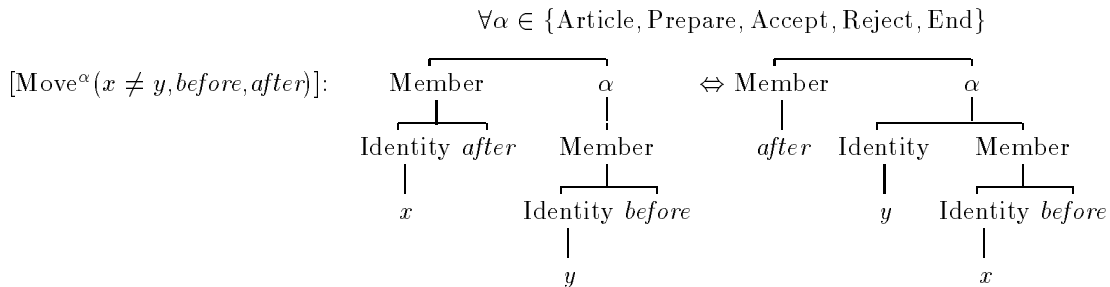
FIG. 4.3 – Spécification de notre cas d'étude en Logique de Réécriture



La règle “Prepare” marque la terminaison du comportement observé de la relecture pour un article, la règle “Open” signant la fin du processus pour l'ensemble des articles. Que la “fin des relectures” ne soit pas détectée est un point important apporté par l'analyse formelle. A l'analyse du texte original, il apparaît que la réunion du comité de lecture peut se tenir sans que l'ensemble des relectures soient connues. Compte-tenu du caractère asynchrone (et non fiable) du système, ceci est une nécessité. C'est par cette connexion causale que les relecteurs coopèrent avec les membres du comité de lecture. Mais cette connexion est non bloquante, une relecture peut être intégrée après l'acceptation ou le rejet. Dans le même ordre d'idée, la phase de répartition des articles entre les membres du comité n'est pas affectée par le DeadLine, comme un graphique donné dans un des articles consacré à ce cas d'étude [RV99] le spécifie dans son ordre partiel (i.e. son “graphe”) de dépendance. Ce chevauchement de phases est la raison principale de la phrase sibylline “les tâches ci-dessous sont listées dans un ordre arbitraire”. Cet ordre de présentation était une extension linéaire [DD93] de l'ordre causal partiel entre les tâches, l'autre ordre possible était 3 avant 2. Notre spécification résout ce problème en identifiant les deux points de contact entre le processus de la conférence et le processus de la relecture. Le premier point de contact est implicite et se réalise par le truchement d'une substitution contrainte par le contexte d'application, le deuxième met en jeu notre procédé informel de raffinement, qui consiste à prendre une règle qui est une possible substitution de son ancêtre. Nous aurions pu utiliser le procédé explicite pour le *setup* mais cela aurait eu pour effet de coder le *setup* dans la spécification; il en aurait résulté une perte évitable de généralité, généralité qui nous permet, comme dans la spécification originale, de ne pas coder le nombre de relecture que nécessite un article dans la spécification mais de laisser cette information dans le *setup*. Par contre une autre information n'a pas sa place dans le *setup* mais nous l'y avons laissée pour avoir l'opposition raffinement implicite/explicite, il s'agit de la présence du terme “Member(**rien**)”. Ce germe, dupliqué sur chaque article, va permettre de réaliser des réunions synchrones locales à chaque article.

Spécifications de la communication Maintenant, nous allons aborder notre manière de spécifier les interactions dans notre système. Ces interactions sont de deux types:

Communication Synchrone Toutes les identités de membres du comité de programme sont considérées en communication synchrone (ou “simulée synchrone” par échange de messages courts) dès qu'ils ont le même opérateur “Members” les regroupant. Chaque terme “Member(*group*)” représente une réunion. Ces réunions peuvent être locales à un article. En effet nous avons muni notre *setup* d'un sous-terme “Member” qui permet cette possibilité. Il nous suffit alors de munir le système de règles permettant à des membres du comité de programme de se “déplacer” ceci est défini par un jeu de règles (symétrique) défini pour tous les états possibles de l'article.



Ceci permet de définir quel membre accède à quel article et on ne dispose pas d'un accès global aux contenus de tous les articles, ni aux identités des auteurs; on conserve ainsi une certaine confidentialité. On peut définir un système de règles du même type mais dans un seul sens (ascendant) pour récupérer la liste des relecteurs, mais ceci n'est plus un objectif de communication mais tient plutôt des phases de "vues partielles" (4 & 7) que nous ne considérons pas; ce sont des raffinements qui lisent le contenu des éléments de la conférence.

Notons que cela définit une dynamique de groupes pouvant être comparée avec les apartés (cf. 2.4). En se rappelant que les adresses sont des localités et que rien n'interdit d'avoir plusieurs membres avec la même adresse, ceci forme une sorte de dynamique de groupe non-disjoints, une même adresse pouvant être présente dans deux groupes à la fois, la différence avec les apartés étant que ceci échappe complètement au contrôle de la spécification, à la différence des apartés pour lesquels chaque interface identifie "qui est qui" dans les groupes en relation.

Asynchrone Nous n'avons pas représenté l'extérieur du système considéré; en conséquence les interactions asynchrones avec cet extérieur apparaissent comme des événements spontanés. Nous avons conçu notre spécification du point de vue d'un pilote abstrait d'un site de conférence, dans ce point de vue, les arrivées ou les départs de messages sont perçues comme des synchronisations avec le service concerné. La problématique de la composition forme la perspective naturelle d'une telle démarche.

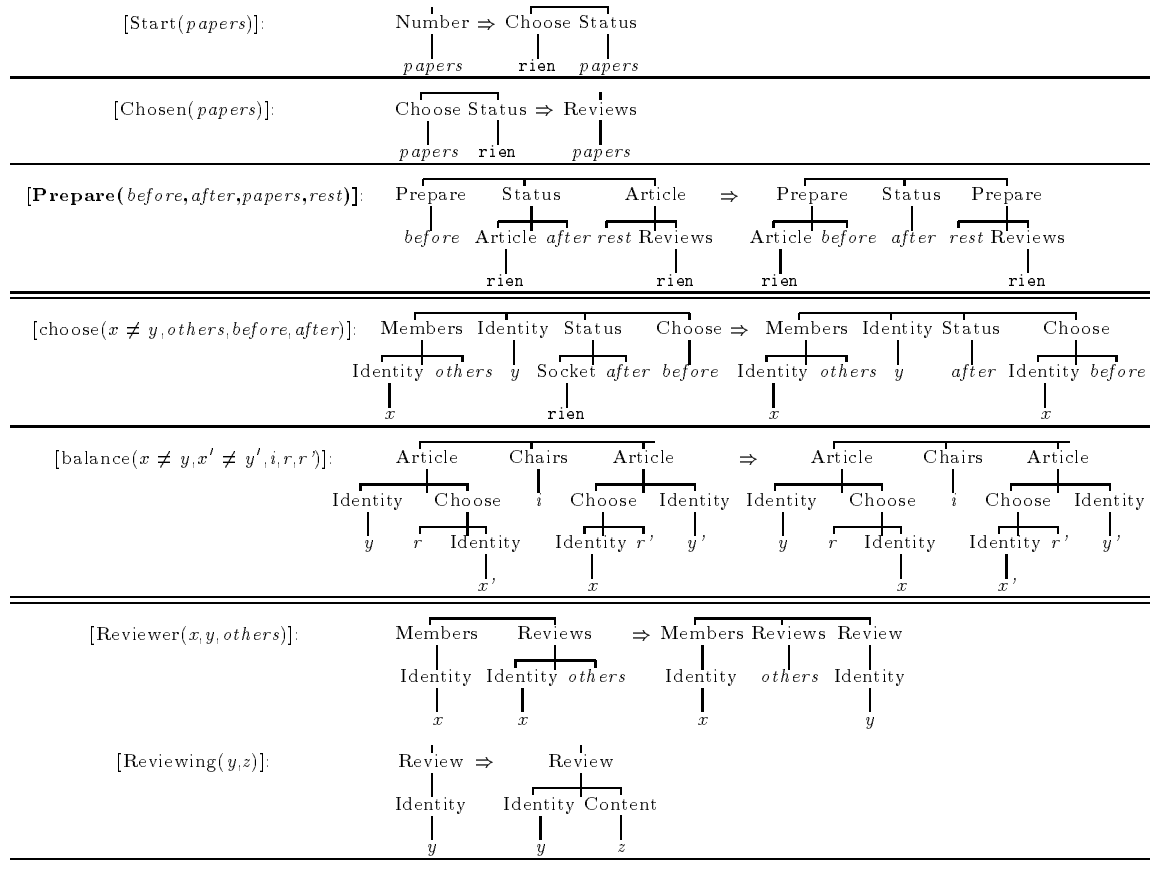
Interdiction Formellement, bien qu'ayant fait partie de nos motivations, la question de la possibilité pour un membre de soumettre n'est pas détectable. Cela supposerait de pouvoir faire un test sur le contenu des variables autre que l'égalité et ceci n'est pas réalisable en l'état. On a tout de même indiqué les conditions sous forme de garde dans l'énoncé des variables utilisées car c'est une réalité applicative. En MAUDE, l'égalité de deux sous-termes passe aussi par l'expression d'une garde contrôlant le franchissement de la règle. Considérer des égalités ou des différences relève donc de la même complexité applicative. Par contre même avec cette inflexion, la détection de deux adresses identiques dans un groupe reste impossible à assurer. On peut en dire autant pour l'exigence qu'un membre ne soit pas capable "d'inférer une information le concernant en tant qu'auteur". Ceci est un objectif presque absurde dans un cadre où les membres pourraient "comploter", il suffit d'évoquer le grésillement mélodique de la radio anglaise pour statuer l'inviolabilité intrinsèque de ce type de code par des machines.

Notre cas d'étude est spécifié jusqu'à un niveau qui est un compromis: d'une part il faut que le cas d'étude ne soit pas réduit au statut d'exemple illustratif et donc une certaine complexité est requise, d'autre part cette complexité est limitée par le support papier utilisé et ses possibilités de navigation. Le compromis trouvé repose sur une remarque fondamentale, deux processus connectés en un seul "point" coexistent dans la spécification: le processus de la conférence et le processus de la relecture. Ce point de connexion est exprimable par la causalité suivante: pour que la réunion du comité de programme puisse commencer, toutes les demandes de relectures doivent avoir été émises. Mis à part cette contrainte, les deux ensembles de règles coexistent sur le même terme sans interactions. Seule une spécification poussée pouvait arriver à cette conclusion.

Interpréter cette spécification

Ces variables qui se "promènent" un peu partout doivent retenir notre attention. Si l'on interprète cette spécification avec le point de vue d'un "réécrivain", même

FIG. 4.4 – Raffinement pour la relecture



si c'est syntaxiquement et sémantiquement correct, ce type de spécification ne correspond à aucune réalité car ce n'est pas calculable. En effet, mais ce n'est pas son objectif. L'objectif de cette spécification est de fixer les bases d'un protocole et non d'un calcul. C'est pour permettre une spécification incomplète que nous avons proposé notre démarche de raffinement. Maintenant quelle réalité opérationnelle peut-on donner à cette spécification?

Pour répondre à cette question, nous allons considérer une notion de *stratégie* [BKK⁺98] de simulation: parmi les termes de preuves sans concurrence et sans composition dont la source est l'état courant, lequel ou lesquels choisir? La réponse à cette question implique nécessairement de faire intervenir le type de propriétés que l'on souhaite valider sur le système [Ver89]. La stratégie que nous proposons ici correspond à une exploration du comportement décrit à l'aide d'hypothèses minimales sur ce qui n'est pas connaissable:

Variables de sorte soupe Ce type de variable résulte d'une porte ouverte au raffinement. Cette variable peut s'instancier différemment sans changer le comportement (i.e. elle préempte du contexte). On choisit son instantiation minimale (**rien**).

Variables de sortes Refs libres à droite De fait, la stratégie induite de la sémantique est "n'importe quoi" (i.e. une référence connue ou inconnue). Le point essentiel concerne l'ensemble des constantes associées à cette sorte, si cet ensemble est fini alors l'exploration est possible, et bien que d'une combinatoire énorme, elle reste finie. Dans le cadre du pilotage cette hypothèse n'est plus possible, aussi bien pour les adresses que pour les contenus et il faut considérer des ensembles de constantes infinis, ou raffiner leur définition pour parser, par exemple, les adresses de mail.

Une fois cette interprétation posée, nous allons détailler comment réaliser un simulateur sur ce type de structures en MAUDE. Nous n'avons pas détaillé quelle logique d'expression de preuve était compatible avec notre stratégie [Ver89]. Implémenter est la manière la plus simple de spécifier; le formel prend plus de temps et d'énergie, en conséquence le développement précède toujours un peu la formalisation: c'est une source d'inspiration.

Ici apparaît une limitation générale de notre approche. On ne sait pas réutiliser des fragments de spécification. Cette dernière problématique n'est pas du tout adressée dans ce travail où nos spécifications sont "monolithiques" (i.e. d'un bloc), la démarche proposée pour le raffinement ne dépasse pas le cadre de la notation informelle, introduite pour limiter la taille d'un bloc de spécification. C'est une perspective de travail de parvenir à rendre des fragments de spécifications réutilisables, comme le principe du seau à jeton qui est dupliqué pour la conférence. Cette limitation est déjà apparue pour les apartés dans le cadre des grammaires à double-somme (Cf. 2.4), dans notre approche il aurait fallu proposer un mécanisme unique pour les deux formalismes. La seule opération qui soit définissable dans les deux formalismes est la somme amalgamée [DM00]! La possibilité de définir des morphismes entre les théories permet en effet d'utiliser les co-limites comme outil de composition, comme les modules de [EM90], utilisés par l'outil MOKA [MW97] pour réaliser la composition de signatures.

4.2 Simuler en MAUDE

Le premier moteur d'inférence que l'on va développer est écrit en MAUDE. Cette section va se décliner en trois parties, la première va présenter succinctement l'outil et ses principales caractéristiques; ensuite on s'intéressera au problème

des systèmes décrits par la logique de réécriture, en particulier les réseaux de Petri Place/Transition sur lesquels on élabore le simulateur; nous resterons volontairement dans le cadre de la logique de réécriture seule avec les possibilités offertes par CORE-MAUDE[CDE⁺99].

4.2.1 Réécrire des structures en MAUDE

Un module se compose d’une signature, d’un ensemble de réécriture silencieuse (des “*équations*” [CDE⁺99], mot que nous réservons aux équations définis par les propriétés des opérateurs) et un ensemble de réécritures étiquetées. Considérons le module suivant correspondant au pique-nique des philosophes

```
mod PIQUE-NIQUE is
  sort soupe .
  op rien : -> soupe .
  op  $\oplus$ _- : soupe soupe -> soupe [assoc comm id: rien] .
  ops dehors dedans libre occupée : -> soupe .
  rl[prends] pensant  $\oplus$ libre  $\oplus$ libre => mangeant .
  rl[relache] mangeant => pensant  $\oplus$ libre  $\oplus$ libre .
  rl[entre] rien => pensant  $\oplus$ libre .
  rl[sort] pensant  $\oplus$ libre => rien .
endm
```

Comme sémantique opérationnelle native d’une spécification, MAUDE fournit le calcul. Or, comme le calcul est une fonction partielle(Cf. 1.1), il n’est pas une sémantique comportementale des réseaux de Petri (i.e. une fonction totale, Cf. 3.1). L’argument est abstrait, on peut en donner une version plus concrète: le fait qu’un réseau de Petri ait une dynamique terminante ou pas peut avoir une importance, mais ce n’est pas la seule question qui ait une importance. La sémantique d’un réseau marqué, c’est tout le graphe d’accessibilité et non un chemin choisi arbitrairement. Finalement la seule question à laquelle va répondre le moteur d’inférence à la question “**rew rien .**” concerne son implémentation, **enter** est-il le minimum de l’ordre arbitraire sur les transitions? Si oui, alors le moteur s’arrête sur un dépassement de capacité mémoire, sinon il ne répondra rien et “moulinera” jusqu’à sa disparition sur le fait que si un philosophe prend et relâche ses baguettes alors il peut prendre et relâcher ses baguettes et donc il peut prendre...

4.2.2 le simulateur primitif

Ce conflit de notions clés peut être insoluble. Il le fut pour le moteur PROGRESS[Sch90] pour lequel je ne suis pas arrivé à écrire que je ne voulais “faire qu’un pas” ! Tout le reste était parfait, sauf que je ne pouvais pas faire sortir de la machine le moindre résultat correspondant à la sémantique que j’utilisais sur ces systèmes. Lorsque le même phénomène s’est reproduit avec MAUDE, j’ai cru que je n’arriverais jamais à donner un contenu opérationnel à mon travail (ML est arrivé bien après pour les constructions catégoriques).

Mais MAUDE n’est pas un moteur d’inférence comme les autres: il est réflexif. Comme défini formellement dans l’état de l’art, cela signifie qu’il existe un codage des termes et des modules. La base de ces codages est la notion de *quoted identifier* (“Qid”), un “Qid” est une chaîne de caractère préfixée par une quote ('). C’est le fait que l’ensemble des *quoted identifier* soit potentiellement dénombrable et donc infini qui permet l’implémentation de la réflexion.

Module Pour les modules, nous utiliserons une construction qui permet de faire appel à la représentation interne d’un module déjà défini. Pour la syntaxe

complète de la représentation d'un module se référer à [CDE⁺99].

```
sort Module .
op [] : Qid -> Module .
```

Termes Pour les termes, nous donnerons la signature complète car elle est intéressante par sa simplicité relative :

```
sorts Qid Term TermList .
subsort Term < TermList .
op _[] : Qid TermList -> Term .
op { }_ : Qid Qid -> Term .
op _,- : TermList TermList -> TermList [assoc] .
op error* : -> Term .
```

La représentation de $B \oplus C \oplus D$ est ' \oplus [_{'B}'soupe,{'C}'soupe,{'D}'soupe}]. Cette simplicité est grandement due à la construction **subsort** qui permet à tout élément d'une sorte d'être aussi d'une autre sorte.

Une fois parvenu à ce niveau, on peut appliquer les *fonctions de descente* qui sont les implémentations de l'équation de réflectivité. Parmi ces fonctions nous n'en utiliserons qu'une:

```
sorts Substitution ResultPair .
op _<- : Qid Term -> Substitution .
op none : -> Substitution .
op _,- : Substitution Substitution -> Substitution [assoc comm id: none] .
op { ,_- } : Term Substitution -> ResultPair .
op meta-apply : Module Term Qid Substitution MachineInt -> ResultPair .
```

L'opération **meta-apply**($\overline{R}, \overline{t}, r, \sigma, n$) est évaluée comme suit [CDE⁺99]:

- le terme t est d'abord réduit par les règles silencieuses du module. Il est pour cela souhaitable que ces règles soient confluentes et terminantes;
- On essaye ensuite d'instancier les règles d'étiquettes r avec le terme résultat. Les substitutions doivent considérer la substitution passée en paramètre comme point de départ obligatoire. Il est important de remarquer ici, que seule la règle d'application est appliquée éventuellement avec une congruence, ce qui correspond à une réécriture dans le *top-level* exclusivement;
- les n premières instanciations sont éliminées, s'il existe une $n + 1^{me}$ instanciation, la règle est appliquée en utilisant cette dernière instanciation.
- le résultat de l'application est réduit par les règles silencieuses du module;
- la paire {résultat de l'application réduit ; substitution utilisée} forme le terme de résultat, qui est retourné. Toute information relative à la congruence est perdue.

Et ce niveau nous avons presque notre simulateur primitif, il suffit de parcourir l'ensemble des règles et, pour chaque règle, énumérer les entiers. On s'arrête pour chaque règle lorsque l'on obtient {**error***, **none**}. On donne ci dessous le résultat de l'énumération décrite ci-dessous en ne donnant pas les résultats nuls.

```
Maude> red in META-LEVEL : meta-apply(['PLACE/TRANSITION],
                                     '⊕-[_{'B}'soupe,{'C}'soupe,{'D}'soupe], 'b, none, 0) .
result ResultPair : { '⊕-[_{'C}'soupe,{'D}'soupe], none }
Maude> red in META-LEVEL : meta-apply(['PLACE/TRANSITION],
                                     '⊕-[_{'B}'soupe,{'C}'soupe,{'D}'soupe], 'c, none, 0) .
result ResultPair : { '⊕-[_{'B}'soupe,{'D}'soupe], none }
```

Tout semble pour le mieux dans le meilleur des mondes possibles jusqu'à ce que nous essayons la ligne suivante:

```
Maude> red in META-LEVEL : meta-apply(['PLACE/TRANSITION],
'⊕-['A]'soupe,{ 'D}'soupe], 'a, none, 0) .
result ResultPair: error*, none
```

Comment est-ce possible? Pour comprendre, il faut bien intégrer la stratégie du **meta-apply**: pour réaliser sa réécriture il faut que les opérateurs de “top-level” soient identiques, ce qui n'est pas le cas dans cette configuration! La représentation de la règle *a* est en effet:

```
r1 ['a] : {'A}'soupe => '⊕-['B]'soupe,{ 'C}'soupe] .
```

Cette bizarrerie, qui permet de penser que **meta-apply** est un *patch* ultérieur à la conception du noyau de MAUDE, nous a plongé dans des journées entières de “bidouilles” pour parvenir à le dépasser. Ce faisant, nous réalisons pour le même prix la fonction “simulation” qui, pour une représentation de module et de terme, retourne tous les termes résultats de l'application d'une règle étiquetée par un **Qid** donné. Mais donner les termes et les substitutions ne suffisent que dans le cas où la réécriture a lieu au “top-level”, sinon un calcul est nécessaire pour réaliser l'équation de congruence. Nous allons réaliser ce calcul en propageant une structure de termes de preuves qui contient une information univoque sur l'application de la règle.

4.2.3 Les termes de preuves

Donc avant d'étudier le cas d'un moteur d'inférence permettant la simulation de théories de réécriture, on se pose la question de savoir si on peut définir des termes de preuves, dédiés à MAUDE. Pour partir de l'objectif, on cherche à équiper cette signature:

```
fmod OBJECTIVE-PROOF-TERM is
including META-LEVEL .
sorts ProofTerm .
ops source target : Module ProofTerm -> Term .
endfm
```

Il existe une solution relativement canonique, qui est donnée dans la figure 4.5. Les déclarations de variable ne sont pas données, car elles peuvent être déduites de la spécification, elles portent toujours le nom de leur sorte avec une minuscule en tête, éventuellement suivies de quotes (') pour différencier des instances différentes. Cette écriture montre que l'information que l'on peut extraire du moteur est insuffisante pour construire des termes de preuves concurrents, en effet on ne récupère pas la partie du contexte laissé invariant par l'application d'une règle au “top-level”.

4.2.4 Le module STEPPER

La première constatation pour simplifier l'écriture du simulateur pas à pas est de casser en trois les arguments de **meta-apply** en remarquant que des paramètres sont de l'ordre de la définition abstraite de la transformation, et ensuite on a le terme et l'entier.

```
mod STEPPER-SIGNATURE is
including IMPLEMENTATION-PROOF-TERM .
sort Actif Steps .
subsort ProofTerm < Steps .
op pit* : -> Steps .
op +_ : Steps Steps -> Steps [assoc id: pit*] .
op Actif : Module Qid Substitution -> Actif .
op Steps : Actif Term -> Steps .
endm
```

 FIG. 4.5 – Implementation des termes de preuves en MAUDE

```

fmod IMPLEMENTATION-PROOF-TERM is
including OBJECTIVE-PROOF-TERM .
sort ProofTermList .
subsort ProofTerm < ProofTermList .
op ProofTerm : Term Qid Substitution MachineInt -> ProofTerm .
op Identity : Term -> ProofTerm .
op Congruence : Qid ProofTermList -> ProofTerm .
op _ : ProofTermList ProofTermList -> ProofTermList [assoc] .

ops rootSource rootTarget : Module ProofTerm -> Term .
ops expandSource expandTarget : Module ProofTermList -> TermList .
op extract : ResultPair -> Term .

[...] Variable declaration

eq source(module,proofTerm) = meta-reduce(module,rootSource(module,proofTerm))
.
eq target(module,proofTerm) = meta-reduce(module,rootTarget(module,proofTerm))
.

eq rootSource(module, ProofTerm(term,qid,substitution,machineInt))
= term .
eq extract({term,substitution}) = term .
eq rootTarget(module, ProofTerm(term,qid,substitution,machineInt))
=
extract(meta-apply(module,term,qid,substitution,machineInt)) .

eq rootSource(module,Identity(term)) = term .
eq rootSource(module,Congruence(qid,proofTermList)) =
qid[expandSource(module,proofTermList)] .
eq expandSource(module, (proofTerm proofTermList)) =
rootSource(module,proofTerm), expandSource(module,proofTermList) .
eq expandSource(module, proofTerm) = rootSource(module,proofTerm)
.

[...] rootTarget symmetric computations

endfm

```

On commence par écrire le SURFACE-STEPPER qui ne fait que déplier les itérations successives de **meta-apply**. Puis on écrit le DEEP-STEPPER, qui implémente l'algorithme suivant :

- si une réécriture est possible au “top-level”, auquel cas on récupère une suite de “steps”, auquel on adjoint les “steps” possibles des sous-termes préfixés par une représentation d'opérateur (i.e. `[_]`);
- sinon, aucune réécriture n'est possible au “top-level” : on ajoute les “steps” possibles à partir de tous les sous-termes, y compris les représentations de constantes (i.e. `{_}`);
- enfin on rajoute systématiquement les réécritures possibles à partir de l'élément neutre de l'opérateur au “top-level”.

Il faut ensuite à la remontée du calcul, déplier les alternatives qui ont eu lieu “dans” un sous-terme. Structurellement, on ne traite pas le cas de réécriture concurrente. A ce propos, il faut noter que toute la concurrence n'est pas détectable. En effet lorsqu'une réécriture est réalisée au “top-level” par **meta-apply** on ne sait pas sur quelle fraction du terme elle est effective, et donc on n'est dans l'impossibilité de produire automatiquement le terme de preuve donné en exemple. Pour pouvoir réaliser un tel simulateur avec concurrence intégrée, le **ResultTerm** devrait avoir la forme suivante, où le dernier terme donne la partie non-affectée par la règle.

```
op {_,_;-} : Term Substitution Term ->ResultTerm .
```

Pour retrouver un simulateur classique de Réseaux de Petri, On écrit le petit module suivant qui implémente une façon de lire les pas:

```
fmod PETRI-SIMULATOR is
including STEPPER .
sort Alternatives .
subsort Term < Alternatives .
op none* : -> Alternatives .
op _+_ : Alternatives Alternatives -> Alternatives [assoc id:
none*] .
op enable : Module QidList Term -> Alternatives .
op targets : Module Steps -> Alternatives .
[...]Variables declaration
eq enable(module,nil,term) = none* .
eq enable(module,qid qidList,term) = targets(module,Steps(Actif(module,qid,none),term))
+ enable(module,qidList,term) .
eq targets(module,pit*) = none* .
eq targets(module,proofTerm + steps) = target(module,proofTerm) +
targets(module,steps) .
endfm
```

On remarque que l'on est obligé de fournir le noms des règles. En effet aucune commande ne permet de les récupérer automatiquement. Dans le même ordre d'idée, aucune commande ne permet de savoir si un opérateur est associatif/commutatif, ce qui est important pour ne pas donner des termes de preuves équivalents. Donc, pour pouvoir réaliser un simulateur “self-contained” il manque deux commandes:

```
op RuleList: Module ->QidList .
op AC: Module Qid ->Bool .
```

On implémentera ces manques par le rajout d'un module permettant de récupérer toute l'information nécessaire au déroulement du simulateur.

4.2.5 Normalisation des variables d'état

Le problème pour réaliser des systèmes où des conditions sur les références sont posées nécessite de pouvoir extraire d'un terme arbitraire l'ensemble de

ces références. On souhaite donc réaliser une fonction qui renvoie l'ensemble des références d'une sorte donnée. Comme illustré dans notre expérimentation, une convention non formulée dans la définition stipule qu'un terme peut être un **Qid**, ce qui signifie que c'est une variable. Mais on ne parvient pas à retrouver sa sorte, à moins de pouvoir de mettre cette information préventivement sous une forme récupérable. En outre, on se limiterait à un ensemble borné par construction de l'ensemble des variables, ce qui est dommage.

Pour dépasser cette limitation opérationnelle, on impose de déclarer, pour toute sorte de référence **Reference**, un opérateur de la forme:

```
op Reference : X -> Reference .
```

La sorte **X** étant la sorte de nommage, les chaînes de caractères (**Qid**) ou les entiers (**MachineInt**) en formant les deux principales instances. On a aussi développé des sortes de références qui étaient des commandes **L^AT_EX** pour des questions de visualisation, cette sorte de référence imprévue lors de la première version a montré l'intérêt de considérer cette donnée comme étant laissée libre au spécifieur. Cette déclaration prend place dans le module d'informations complémentaires qui sera détaillé à la fin de cette section.

Une fois cette normalisation achevée, on souhaite réaliser une fonction de calcul dont l'état de départ contient un terme et un ensemble de noms de sorte de référence et dont l'état unique d'arrêt contient en plus l'ensemble des références des sortes passées en paramètres. Il inclut le module fonctionnel **BELONGS** qui contient les deux opérateurs d'appartenance dans une liste pour les **Qids** et les **Terms**. Ceci est un parcours (non concurrent) en profondeur du terme.

```
fmod PROFONDEUR is
including BELONGS .
sort References .
op References : TermList -> References .
op Get : QidList Term References -> References .
op Explore : QidList TermList References -> References .
[.../ Variables declaration
eq Get(qidList, {qid}qid', references) = references .
eq Get(qidList, qid[termList], References(termList')) =
if qidIn(qid,qidList)
then if termIn(qid[termList],termList')
then References(termList')
else References((qid[termList],termList'))
fi else Explore(qidList, termList, References(termList'))
fi .
eq Explore(qidList, term, references) = Get(qidList, term,
references) .
eq Explore(qidList, (term,termList), References(termList')) =
Explore(qidList, termList, Get(qidList, term,
References(termList')))) .
endfm
```

Ceci permet de récupérer toutes les références citées dans un terme et donc de faire la différence entre des références connues et inconnus, ce qui est le fondement d'une politique d'allocation de références.

4.2.6 Contraindre la substitution

Le fait que le **meta-apply** de MAUDE permette de spécifier une contrainte sur la substitution est une capacité déterminante pour réaliser notre prototype. Cette caractéristique m'a obligé à conserver mon travail sur ce moteur. En effet, il existe

un concurrent à MAUDE nommé ELAN[BKK⁺98] et développé par le LORIA. Ce moteur est plus spécialisé sur la notion de stratégie. En conséquence le STEPPER en ELAN est beaucoup plus simple et compact, mais comme on n'a pas accès à la substitution, on est coincé. Ceci dit, ayant découvert ce logiciel très tardivement, peut-être qu'une solution procédant complètement différemment m'a échappé.

Pour permettre de travailler avec des spécifications en logique de réécriture contenant des variables non contraintes, La politique est la suivante :

Variable non contrainte lorsque c'est possible on choisit une constante canonique, comme `rien` pour la sorte `soupe`. Lorsque ce n'est pas possible, on considère être dans le cas suivant :

Variable de référence libre à droite pour chaque variable libre à droite, on choisit de l'instancier sur n'importe quelle "variable" connue ou sur une variable nouvelle.

Ceci donne une combinatoire énorme mais cela a aussi un sens de vérifier qu'un système résiste à cet empilage sur une référence de termes venant de règles a priori différentes. On ne filtre pas les termes de preuves, ils sont tous corrects. On peut, pour permettre une exploration exhaustive, empêcher l'allocation de nouvelles références.

Réécrire de graphes concrets en MAUDE Réécrire des graphes en MAUDE se heurte à la différence de nature entre les formalismes. On ne pourra pas directement réécrire des classes de graphes. Néanmoins, comme nous avons du déjà contrôler la réécriture pour réaliser le cas d'étude pourtant spécifier en Logique de Réécriture. Il s'agit d'adapter cette technique informellement pour obtenir des dérivations concrètes de graphes. Exprimé en terme de contrainte sur les termes de preuves, la politique est la suivante,

Variable libre à droite pour chaque variable libre à droite on l'instancie sur une variable nouvelle (i.e. non citées dans l'état courant).

Substitution injective pour les libres Un terme de preuve correct vérifie : Aucune variable ne partage son image par la substitution avec une variable libre (à gauche ou à droite).

Variable libre à gauche Un terme de preuve correct vérifie : Toute image d'une variable libre à gauche est absente de la cible du terme de preuve.

Avec cette politique la spécification du "bal des philosophes" écrit en MAUDE a un comportement comparable à une grammaire de graphe. Comme indiqué à la fin du chapitre III, ce n'est pas une équivalence de comportement, mais le moteur réalise des dérivations concrètes. Cette possibilité ouvre une voie intéressante, si on a les moyens d'accéder aux termes de preuves, même sans concurrence, on peut choisir ceux qui correspondent à une dérivation concrète d'une grammaire de graphes en représentant les graphes par des multi-ensemble de tuples. Pour parvenir à réaliser le graphe des états accessible tel que définit dans le chapitre II, il faut munir le moteur d'une recherche d'isomorphisme (ici un renommage de variable de références) et ainsi on pourrait prouver une injection de la structure calculée dans le graphe abstrait de dérivation.

4.2.7 Module d'information

Pour parvenir à réaliser ces politiques, il faut indiquer à la machine virtuelle qui simule les différentes informations dont il a besoin pour réaliser les contraintes. Le

module d'information contient toutes ces informations nécessaires au déroulement de la politique de gestion de références et de la simulation pas à pas. Il doit contenir les informations suivantes:

- Le nom des règles
- Les identifiants des opérateurs Associatif-Commutatif
- Les identifiants des opérateurs élément neutres d'opérateurs Associatif-Commutatif
- Les identificateurs des sortes et parmi elles les sortes de références et donc les noms d'opérateurs définissant les "variables"
- Les références des deux membres de chaque règles

Ce cahier des charges correspond dans l'implémentation actuelle à un module qui contient toutes ces informations. Elles peuvent être calculées à partir de la spécification originale. En outre, nous supposons que ceci pourrait former des opérateurs à la manière de `[_]` ce qui permettrait sans grand frais d'étendre le champ d'action de MAUDE. Une solution techniquement réalisable serait de tout développer en FULL-MAUDE. Nous avons longtemps rejeté cette possibilité car nous craignons pour les performances de notre engin, et puis cela contrevenait à un principe d'économie de moyens. Peut être dans une optique de faisabilité, cette option devrait être reconsidérée.

Ce dernier point permet de construire un moteur basé sur MAUDE qui développe en l'état des séquences qui peuvent être associées à des chemins dans le graphe abstrait. Deux éléments dans la séquence peuvent être envoyés sur le même état (i.e. classes de graphe ou *graphes abstraits*). Pour circonvenir à cette limitation, le moteur doit proposer une recherche d'isomorphisme sur les états déjà connus et inclure dans le terme de preuve une représentation de la permutation nécessaire. Ceci peut être implémenté en MAUDE mais ne permet pas de traiter une construction clé : les architectures dynamiques (Cf. 2.4). En conséquence, cette perspective n'est plus une perspective centrale.

Permettre à MAUDE (ou ELAN) de proposer une représentation univoque des termes de preuves est une perspective prioritaire. La première application serait de pouvoir construire le graphe des états accessibles dont les arêtes sont des termes de preuve et les sommets des termes. Pour mettre ce graphe en conformité avec la théorie, il faudrait pouvoir extraire non seulement la substitution mais aussi le contexte d'application d'une règle. En l'état, MAUDE ne permet pas de retirer suffisamment d'information du **meta-apply** à cet effet. Ceci permettrait aussi d'en faire un moteur capable non seulement de concevoir mais aussi d'exprimer la concurrence comme une congruence de transformations car on pourrait continuer à chercher de nouvelles applications dans le contexte laissé invariant par la première application. Une fois parvenu à retirer d'un terme l'ensemble des termes de preuves avec concurrence qui ont pour source ce terme, on peut élaborer des stratégies visant à réduire les redondances dans cet ensemble de termes de preuve dans un objectif de réduction de l'explosion combinatoire.

Conclusion

La spécification d'applications coopératives est un problème nouveau et complexe. Pour résumer ces spécificités, nous avons proposé de définir une application coopérative comme une "architecture dynamique d'éléments éphémères". Cette définition pourrait s'appliquer à d'autres types de systèmes, comme les systèmes répartis. Mais deux hypothèses simplificatrices classiquement admises pour la spécification d'un protocole, la connaissance a priori des effectifs impliqués et de leur intention, ne sont plus des hypothèses acceptables.

En l'absence de connexions physiques entre les acteurs de cette coopération, leur appartenance au même processus coopératif est assurée par des connexions logiques (causale ou asynchrones). La simplification conceptuelle initiale [GV97] consiste à considérer la connexion comme objet quelque soit sa modalité : physique ou logique. En ce sens, il faut comprendre la terme d'architecture non seulement dans une acception spatiale (comme les connexions) mais aussi temporelle (comme les dépendances). Le dynamisme de la topologie abstraite de connexion est une conséquence de la non-permanence des éléments impliqués.

Nous avons montré que la spécification d'applications coopératives, ou de "techniques de coopération" n'était réalisable en l'état que par une "coopération de techniques" formelles choisies à cet effet. Cette "coopération de techniques" a été rendue possible par un vaste travail préparatoire qui a permis d'unifier le contexte de définition et d'interprétation d'une spécification, quelle soit en Réseaux de Petri Place/Transition, en Logique de Réécriture ou en Grammaire à Double-Somme.

Comparaison expérimentale: Notre approche est de nature expérimentale, partant des problèmes nous cherchons à les spécifier dans des formalismes adéquats. Cette adéquation repose sur un principe abstrait d'économie syntaxique et une approche multi-modèles ou multi-paradigmes. Dans cette démarche, une contribution de notre travail est d'explicitier des exemples pour lesquels on propose différentes spécifications et d'en déduire les caractéristiques-clés des formalismes utilisés pour spécifier notre contexte d'étude. Ces formalismes sont au nombre de trois :

- Les réseaux de Petri Place/Transition servent de point de départ méthodologique. Le cadre général de l'analyse des systèmes provient de ce corps théorique. D'une part, on s'intéresse à l'ensemble du comportement du système et d'autre part, on utilise des déductions que l'on peut réaliser structurellement sur la spécification. Ces déductions permettent d'extraire des informations d'une spécification même lorsque l'exploration des états accessible est impossible ou trop coûteuse.
- La Logique de Réécriture, "sémantique" de la réécriture de termes modulo un système d'équations, est d'abord introduite comme un cadre d'écriture compact et rigoureux des réseaux de Petri de Haut-Niveau, comme les réseaux colorés ou les réseaux Prédicats/Transitions. Dans un deuxième temps, les

caractéristiques propre des spécifications dans ce système formel sont plus finement exploitées. Nous avons contribué de manière importante à faire de ce cadre formel un outil complet de spécification d'applications coopératives.

- Les Grammaires de Graphes sont un formalisme qui utilise une opération catégorique, la somme amalgamée (ou pushout), pour exprimer le remplacement d'un motif dans un contexte. Cette opération propose une gestion native du problème des identités. Pour ce formalisme, permettre son utilisation dans un cadre de spécification d'applications coopératives nécessite un travail plus important que pour les deux précédents et forme le chapitre II de notre mémoire.

Pour pouvoir positionner les différents formalismes, nous avons défini trois exemples :

- le “gala des philosophes” sert de fil conducteur au premier chapitre et permet de comparer (cf 1.5) la Logique de Réécriture et les Grammaires de Graphes sur un domaine où elle permettent toutes deux de décrire le système considéré. Le “gala” est décomposé d'abord en deux sous-problèmes, spécifiés en réseaux de Petri Place/Transition :
 - “la pique-nique des philosophes”, où les philosophes prennent leurs baguettes dans un sac et peuvent entrer et sortir du système. Ce système est à effectif dynamique et à topologie statique.
 - “le bal des philosophes” le problème classique auquel on adjoint la possibilité pour les tables de philosophes de fusionner ou de se séparer. Ce système est à effectif statique et à topologie dynamique.

Seul la combinaison des deux problèmes, une topologie dynamique à effectif dynamique nécessite un formalisme pouvant décrire la machine de Turing.

- le TCPTokenBucket, un seau à jetons dont le débit en sortie est dynamique, est spécifiable en Logique de Réécriture mais n'a pas d'expression simple en Grammaires de Graphes. Il n'est pas réalisable en Réseaux de Petri Place/Transition et nécessite d'utiliser des réseaux auto-modifiants [Val78]. Cet exemple montre la possibilité de définir des événements paramétrés par des sous-structures.
- la formation d'apartés dans des groupes coopératifs est un problème important de notre domaine d'application. Le problème de l'admission dans un sous-groupe n'admet pas d'expression simple ni en Logique de Réécriture, ni en Grammaires de Graphes. Pour mener à bien cette spécification, nous avons eu recours à la généralisation des Grammaires de graphes proposée en 2.2, en l'occurrence l'utilisation de grammaires de “graphes de graphe”. D'avoir pu appliquer cette structure dans la définition d'un groupe coopératif et aussi dans la définition des apartés dans ces groupes est un fait significatif de la pertinence de cette structure dans ce cadre. Ceci met en valeur une possibilité insoupçonnée des grammaires à double somme : celle de pouvoir décrire des relations arbitrairement complexes entre des éléments.

A l'issue de cette confrontation pragmatique, chacun conserve des caractéristiques propres. Les réseaux de Petri ont comme avantage leur capacité d'analyse, et chacun des deux autres formalismes a un domaine spécifique dans lequel l'autre ne peut s'exprimer au mieux qu'avec de grandes difficultés (étant tous deux équivalents aux machines de Turing). Pour résumer, la Logique de Réécriture permet de décrire des éléments complexes et des événements de même complexité, tandis que notre généralisation des Grammaires de Graphes, les Grammaires à Double-Somme, permet de décrire des relations complexes entre des éléments : des flèches d'une catégorie.

Comparaison à l'aide de concepts spécifiques : Nous avons identifié trois concepts spécifiques de notre domaine d'application qui caractérisent les propriétés que l'on peut associer à chaque élément. Chaque propriété peut être soit exclusive d'un élément, soit partagée par plusieurs éléments. Nous appelons *identité* une propriété exclusive, et *localité* une propriété partagée. Ces concepts permettent de séparer les formalismes utilisés selon leur adéquation à spécifier telle ou telle caractéristique et donc de choisir au plus tôt le formalisme adéquat.

L'explicitation de ces concepts d'identité et de localité, est motivée par les problèmes-clés associés à chaque type de propriété :

- Manipuler des identités suppose que l'on garantisse structurellement la persistance du caractère exclusif de la propriété considérée.
- Manipuler des localités implique que l'on fournisse un moyen d'en faire une réalité pour le système par la définition d'évènements impliquant tous les membres d'une localité. La possibilité de dupliquer une structure [Cou95] participe de ce type d'opérations.

Les spécifications d'applications coopératives font souvent apparaître ces deux types de propriétés. Il serait donc nécessaire de pouvoir les gérer simultanément ce que les formalismes existants ne parviennent pas à réaliser. Les deux formalismes équivalent aux machines de Turing que nous avons utilisés, la Logique de Réécriture et les Grammaires de Graphes, font soit l'un soit l'autre :

- La Logique de Réécriture permet de définir des termes d'éléments, et de définir des évènements sur des groupes d'éléments en permettant d'importer la structure de l'état dans la structure des évènements. La notion d'identité n'a pas de sens en logique de réécriture, car la duplication de structure est toujours possible.
- Les Grammaires de Graphes reposent sur la définition explicite des identités des éléments et des connexions entre éléments. Elles proposent donc un mécanisme pour gérer les identités. Par contre, il n'est pas possible de définir des évènements sur une localité, par exemple modifier simultanément tous les voisins d'un noeud donné ou de dupliquer un sous-graphe.

Pour illustrer la capacité de ces concepts à aiguiller le choix du formalisme adéquat, nous avons appliqué cette approche sur notre cas d'étude : l'organisation d'une conférence (cf 1.1 et 4.1). Le protocole décrit correspond aux services actuellement utilisés dans cette optique. La possibilité pour un membre du comité de lecture d'être aussi un auteur amène différents agents mandataires du même utilisateur final à partager une propriété : l'identité de cet utilisateur. On parle dans ce cas d'ubiquité possible des participants. Dans ce contexte, l'identité des utilisateurs est un leurre puisque cette identité correspond en fait à une localité des agents, seuls considérés dans la spécification. Par suite, il n'y a pas d'identité dans cette application, seulement des localités et en conséquence une spécification en Logique de Réécriture est tout à fait satisfaisante. Plus pragmatiquement, l'utilisation d'un support textuel devient critique pour pouvoir écrire une spécification d'une telle taille.

Cadre sémantique unifié : La contribution formelle de ce travail est de proposer un cadre sémantique unique pour les trois formalismes ce qui permet de dépasser la simple juxtaposition de formalismes pour permettre d'envisager un réelle coopération de techniques. Notre approche sémantique permet de couvrir deux besoins essentiels de la spécification de systèmes coopératifs :

- de pouvoir gérer des *architectures dynamiques d'éléments éphémères*, ces éléments pouvant être dotés d'un comportement et eux-mêmes structurés.

Ceci suppose que la sémantique associée propose une gestion native de la dynamique des références. Les Grammaires de Graphes proposent une gestion native de ces références par la définition de conditions de sensibilisation (liaison et identification, cf.1.4). Notre généralisation de ces grammaires, (les grammaires à double-somme) permet de paramétrer l’“architecture” des objets de la catégorie de travail. D’une façon symétrique, nous avons montré que la Logique de Réécriture, disposant d’une notion de signature, fournissait un moyen de gérer le dynamisme des références mais en induisant des interprétations différentes de celles des grammaires de graphes.

- d’appréhender l’*ensemble du comportement d’un système* (e.g. le graphe des marquages d’un Réseau de Petri, cf. 3.1) et pas uniquement le calcul associé (cf. 1.1). Ceci nous a amené à reconsidérer les sémantiques associées aux formalismes que nous étudions. Nous développons une démarche, issue de la Logique de Réécriture, unifiant l’expression de sémantique à base de séquents et d’étiquetage univoque de la relation d’accessibilité. Ce type de sémantique associe à tout système un graphe de comportement. Ce graphe de comportement est le point de départ des techniques d’analyses exhaustives des systèmes (i.e. Model Checking), qu’ils soient formalisés par des Réseaux de Petri, des Algèbres de Processus ou, maintenant, par une Grammaire à Double-Somme ou une Théorie de Réécriture. En outre, nous avons montré que le vocabulaire (cf 1.1) de la réécriture et des systèmes de réduction abstraits, bien qu’originellement dédié à la spécification de calculs, pouvait enrichir la taxonomie des systèmes coopératifs.

Une fois la démarche unifiée pour les différents formalismes, notre contribution formelle est de pouvoir comparer rigoureusement ces formalismes et d’étudier les traductions de l’un dans l’autre. On prouve l’extension possible de Réseaux de Petri P/T en grammaires à double-somme et en logique de réécriture. De plus, nous identifions des classes de systèmes, dans chacun des deux formalismes, s’étendant en réseaux de Petri Place/Transition. Si, pour la Logique de Réécriture, cette caractérisation revient à une redéfinition compacte des réseaux de Petri Haut-Niveau, la contribution est plus originale pour les Grammaires de Graphes. En effet, toute Grammaire de Graphe dont les interfaces sont réduites au graphe vide est équivalente à un réseau de Petri dont les places sont les classes de composantes connexes citées dans les règles.

Perspectives de réduction du graphe des états accessibles Les techniques de Haut-Niveau comme la Logique de Réécriture ou les Grammaires de *Graphes de Graphes*, devront être outillées pour permettre l’exploration de leur graphe des états accessibles. Cette étape est en effet un prérequis pour permettre l’utilisation de techniques de vérification telles que model-checking. Une fois ceci achevé, elles devront être appliquées sur des cas concrets d’une taille importante pour valider leur utilisation réaliste. Ceci rendra critique le problème d’explosion combinatoire du graphe des états. Pour rendre possible l’exploration exhaustive de systèmes complexes, il faut permettre l’utilisation de techniques de réduction du graphe des états accessibles. Il s’avère que chacun de ces formalismes dispose dans sa sémantique de possibilités d’applications originales de techniques connues de réduction du graphe des états accessibles :

- La logique de réécriture propose une expression compacte de la concurrence comme une congruence de transformation. Ceci permet d’envisager l’emploi précis de techniques de réduction à base d’ordres partiels par expression de transformations concurrentes, sans décrire tous les entrelacements possibles, source d’une combinatoire exponentielle.

- De façon symétrique, la sémantique que nous avons donnée aux grammaires à double-somme, généralisation des grammaires de graphes à n’importe quelle catégorie de travail comme celle des graphes de graphes, est définie sur les graphes abstraits (classes modulo isomorphismes). La construction du graphe abstrait de dérivation peut donc se comparer aux techniques de réduction par symétries.

Ici apparaît encore la remarquable complémentarité des deux formalismes qui, une fois leurs sémantiques définies dans le cadre commun, proposent chacun un point d’entrée aux deux techniques principales de réduction de l’explosion combinatoire.

Perspectives des systèmes évolutifs : Nous retiendrons finalement une autre perspective qui, outre son ouverture sur des travaux à venir, permet de synthétiser les forces des différents formalismes utilisés. Cette perspective concerne les réseaux de Petri Place/Transition à synthèse et déroulement concurrents, c’est à dire dont la construction ne précède pas causalement le déroulement mais forme deux facettes concurrentes d’une seule exécution.

- L’utilisation des réseaux de Petri P/T comme support de la représentation répond aux qualités de ce formalisme : représentation graphique connue et synthétique de systèmes concurrents, possibilité de décider formellement des propriétés fondamentales comme le caractère borné ou l’accessibilité.
- Si on tente de spécifier le problème posé en Logique de Réécriture, il apparaît que l’équation de tir – activation de l’élément complexe “transition” — ne pose aucune difficulté. Chaque transition est un terme contenant deux marquages, et comme pour le *TCPTokenBucket* l’utilisation de variables de sorte marquage (**pre**, **post**) permet de coder l’équation de tir :

$$\text{rl}[\text{tir}] : \text{trans}(\text{pre}, \text{post}) \oplus \text{pre} \implies \text{trans}(\text{pre}, \text{post}) \oplus \text{post}.$$

Créer de nouvelles transitions est possible car elles sont référencées implicitement, mais par contre la possibilité de créer de nouvelles places nécessite un mécanisme propre qui n’est pas interne au formalisme.

- D’une façon remarquablement symétrique, les problèmes sont inversés en Grammaires de Graphes où l’apparition de nouveaux éléments (place ou transitions) ne pose pas de difficulté. Par contre, l’équation de tir — le degré des transitions étant non borné — nécessite un nombre non borné de règles.

Dans les deux cas, on peut “forcer” et parvenir à un résultat opérationnellement acceptable, mais la solution ainsi construite présente des faiblesses formelles rédhibitoires. Ce problème, et sa solution, forment pour nous un exemple tout à fait intéressant et fondamental des problèmes à venir, et qui justifie la recherche d’un formalisme manipulant conjointement localité et identité.

Notations

Petits Ensembles Les petits ensembles sont notés en italiques avec une simple lettre (A, B, C, \dots).

Grands ensembles Les grands ensembles sont notés avec de doubles lettres majuscules

\mathbb{N} Ensemble des entiers naturels.

\mathbb{Z} Ensemble des entiers relatifs (ou signés).

\mathbb{P} Ensemble des Réseaux de Petri Place/Transition.

\mathbb{R} Ensemble des Théories de Réécriture.

\mathbb{G} Ensemble des Grammaires à Double-Somme.

Fonctions Les symboles fonctionnels sont notés en **true type**, cette notation s'étend aux flèches des catégories, et aux arêtes des graphes. Lorsque l'on note le symbole fonctionnel seul, on donne son arité par l'utilisation de la notation *misfixe* [CDE⁺99] **src**($_$). L'image par une fonction d'un élément est notée **src**(a). L'ensemble des fonctions de A dans B est noté B^A .

Eléments Les éléments d'un ensemble sont notés en *italique* (a, b, c, \dots), cette notation s'étend aux objets des catégories et aux sommets des graphes.

Monoïde libre Soit S un ensemble, on définit l'ensemble des *mots* sur S comme l'union des ensembles de fonctions des *segments initiaux* de \mathbb{N} dans S .

$$S^* = \bigcup_{i \in \mathbb{N}} S^{[1..i]}$$

- soit $x \in S^*$, on note $|x|$ l'élément de \mathbb{N} tel que $x \in S^{[1..|x|]}$ (i.e. $(\mathbf{x} : [1..|x|] \rightarrow S)$);
- soit $(_ _ : S^+ \times S^+ \rightarrow S^+)$ la fonction de concaténation définie comme suit

$$(\mathbf{x.y} : [1..|x| + |y|] \rightarrow S) \stackrel{\text{def}}{=} (\text{fn } i \Rightarrow \begin{array}{ll} \mathbf{si} & (i \leq |x|) \\ \mathbf{alors} & x(i) \\ \mathbf{sinon} & y(i - |x|) \end{array})$$

Cette opération est associative et admet un élément neutre à gauche et à droite: l'unique élément de S^\emptyset , la fonction vide notée ϵ . On note $(S^*, _ _, \epsilon)$ le monoïde des mots sur S avec la concaténation. Le triplet $\langle S^*; _ _, \epsilon \rangle$ est appelé *monoïde libre* engendré par S .

Catégories Une catégorie est notée en **gras**.

Set Catégorie des ensembles, les objets sont les ensembles et les flèches les fonctions totales.

Graphs Catégorie des graphes dirigés, les objets sont les graphes dirigés, les flèches les morphismes de graphes dirigés.

Cat Catégorie des catégories, les objets sont les catégories, les flèches les foncteurs.

$D \in \mathbf{X}$ Catégorie des diagrammes de forme D dans \mathbf{X} .

Family(\mathbf{X}) Catégorie des familles d'objets de \mathbf{X} .

Bibliographie

- [Aal97] W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azema and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
- [Arn87] André Arnold. *Système de transitions finis et sémantique des processus communicants*. Masson, 1987.
- [AVG98] Pierre Azéma, François Vernadat, and Pierre Gradiat. A workflow specification environment. In W.M.P Van der Aalst, G De Michelis, and C.A Ellis, editors, *Workshop Workflow Management: Net-based Concepts, Models, Techniques, and Tools (WFM'98)*, pages 5–21, 1998.
- [Bau95] Michel Bauderon. A uniform approach to graph rewriting: The pullback approach. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, volume 1017 of *Lecture Notes in Computer Science*, pages 101–115, 1995.
- [BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, April 1992.
- [BC87] Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20((2-3)):83–127, 10 April 1987.
- [BC88] Gérard Boudol and Ilaria Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informaticae*, XI:433–452, 1988.
- [Ber77] C. Berge. *Graphes et Hypergraphes*. Dunod, Paris, 1977.
- [BJ97] M. Bauderon and H. Jacquet. Node rewriting in hypergraphs. *Lecture Notes in Computer Science*, 1197:31–??, 1997.
- [BKK⁺98] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An overview of ELAN. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the International Workshop on Rewriting Logic and its Applications*, volume 15 of *Electronic Notes in Theoretical Computer Science*, Pont-à-Mousson, France, September 1998. Elsevier Science.
- [BM91] J. P. Banâtre and D. Le Métayer. Introduction to Gamma. In J. P. Banâtre and D. Le Métayer, editors, *Research Directions in High-Level Parallel Programming Languages*, pages 197–202. Springer Lecture Notes in Computer Science 574, June 1991.

- [BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, 1998.
- [BO98] Eric Badouel and J Oliver. Reconfigurable nets, a class of high-level petri nets supporting dynamic changes. In W.M.P Van der Aalst, G De Michelis, and C.A Ellis, editors, *Workshop Workflow Management: Net-based Concepts, Models, Techniques, and Tools (WFM'98)*, pages 5–21, 1998.
- [Bra83] G (nom collectif) Brams. *Réseaux de Petri: Théorie et Pratique*. Masson, 1983.
- [BW90] Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice Hall, New York, 1 edition, 1990.
- [CDE⁺99] Manuel Clavel, Francisco Durn, Steven Eker, Patrick Lincoln, Narciso, Mart-Oliet, José Meseguer, and José Quesada. Maude: Specification and programming in rewriting logic. Technical report, SRI International, March 1999.
- [CEL⁺94] A. Corradini, H. Ehrig, M. Lowe, U. Montanari, and F. Rossi. Note on standard representation of graphs and graph derivations. *Lecture Notes in Computer Science*, 776:104–??, 1994.
- [CH95] Andrea Corradini and Reiko Heckel. A compositional approach to structuring and refinement of typed graph grammars. In A. Corradini and U. Montanari, editors, *SEGRAGRA'95, Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation*, volume 2. Elsevier, 1995.
- [CH98] A. Corradini and R. Heckel. A compositional approach to structuring and refinement of typed graph grammars. *Theoretical Computer Science*, 194(1–2):241–??, March 1998.
- [Cir00] Horatiu Cirstea. *Calcul de réécriture: fondements et applications*. PhD thesis, Unversité Henri Poincaré (Nancy I), 2000.
- [CM92] A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its applications to logic programs. *Theoretical Computer Science*, 103(1):51–106, August 1992.
- [CM96] Manuel Clavel and José Meseguer. Reflection and strategies in rewriting logic. In *Proceedings of the First International Workshop on Rewriting Logic and its Applications (RWLW96)*, Pacific Grove, CA, 3–6 September 1996.
- [CMR94] A. Corradini, U. Montanari, and F. Rossi. An abstract machine for concurrent modular systems: CHARM. *Theoretical Computer Science*, 122(1-2):165–200, January 1994.
- [CMR⁺97] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. Algebraic approaches to graph transformation - part I: Basic concepts and double pushout approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapter 3, pages 163–246. World Scientific, 1997.

-
- [Cor98] A. Corradini. Concurrent computing: from Petri nets to graph grammars. *Theoretical Computer Science*, 194(1-2):241–242, March 1998.
 - [Cou95] Bruno Courcelle. Réécriture de graphes: orientation bibliographique. <http://dept-info.labri.u-bordeaux.fr/~courcell/Font-Romeu.ps>, 1995.
 - [Cou96] Bruno Courcelle. Basic notions of universal algebra for language theory and graph grammars. *Theoretical Computer Science*, 163(1-2):1–54, 30 August 1996. Tutorial.
 - [CP99] J.-M. Couvreur and D. Poitrenaud. Detection of illegal behaviours based on unfoldings. *Lecture Notes in Computer Science*, 1639:364–??, 1999.
 - [DD93] Philippe Darondeau and Pierpaolo Degano. Refinement of actions in event structures and causal trees. *Theoretical Computer Science*, 118(1):21–48, 13 September 1993.
 - [DGV96] K. Drira, P. Gradiat, and F. Vernadat. Graph-based coordination of cooperative agents. In *Workshop RWTH Aachen “Trends in Distributed Systems” (TreDS’96)*, pages 133–146, 1996.
 - [DHK97] Frank Drewes, Annegret Habel, and Hans-Jörg Kreowski. Hyperedge replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapter 2, pages 95–162. World Scientific, 1997.
 - [Dij71] Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1(2):115–138, October 1971. Reprinted in *Operating Systems Techniques*, C.A.R. Hoare and R.H. Perrot, Eds., Academic Press, 1972, pp. 72–93. This paper introduces the classical synchronization problem of Dining Philosophers.
 - [DM00] Francisco Durn and José Meseguer. A church-rosser checker tool for maude equational specifications. *Manuscript. Universidad de Mlaga and SRI International*. <http://maude.csl.sri.com/papers/crc>, 2000.
 - [Dör95] Heiko Dörr. Bypass strong V-structures and find an isomorphic labelled subgraph in linear time. In W. Mayr, Ernst, Gunter Schmidt, and Gottfried Tinhofer, editors, *Graph-Theoretic Concepts in Computer Science*, volume 903 of *Lecture Notes in Computer Science*, pages 305–318, 1995.
 - [DV94] Michel Diaz and Thierry Villemur. Formation of private conversation subgroups in a cooperative group of processes. *Journal of the Brazilian Computer Society*, 1(1):46–58, July 1994.
 - [EGR91] C.A Ellis, S.J Gibbs, and G.L Rein. Groupware, some issues and experiences. *Communications of the ACM*, 34(1):39–58, JanuaryAugust 1991.
 - [EHL⁺98] Hartmut Ehrig, Reiko Heckel, Mercè Llabrés, Fernando Orejas, Julia Padberg, and Grzegorz Rozenberg. Double-pullback graph transitions: A rule-based framework with incomplete information. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Theory and Application of Graph Transformations, 6th International Workshop, TAGT’98*, volume 1764 of *Lecture Notes in Computer Science*, pages 85–102. Springer-Verlag, 1998.
-

- [Ehr86] H. Ehrig. Tutorial introduction to the algebraic approach of graph-grammars. In A. Rosenfeld H. Ehrig, M. Nagl, G. Rozenberg, editor, *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, volume 291 of *LNCS*, pages 3–14, Warrenton, VA, December 1986. Springer Verlag.
- [EM90] H. Ehrig and B. Mahr. Fundamentals of Algebraic Specifications. In *Monographs on Theoretical Computer Science 1 (2)*, volume Volume 6 (21) of *EATCS*. Springer-Verlag, 1985 (1990).
- [Eng96] J. Engelfriet. A multiset semantics for the pi-calculus with replication. *Theoretical Computer Science*, 153(1–2):65–94, January 1996.
- [ER97] J. Engelfriet and G. Rozenberg. Node replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1: Foundations. World Scientific, 1997.
- [Fan99] Jean Fanchon. A syntactic congruence for the recognizability of pomset languages. Rapport N99008, LAAS/CNRS, 1999.
- [Gen86] Genrich. Predicate/transition nets. In *Advances in Petri Nets 1987*, ed. Grzegorz Rozenberg, *LNCS 266*; *Petri Nets: Central Models and Their Properties*, *Advances in Petri Nets 1986*, *LNCS 254-255*, 1987, *LNCS 188 (1984)*, *LNCS 340 (1988)*, *LNCS 483 (1991)*. Springer-Verlag, Berlin, 1986.
- [Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.
- [Gir97] François Girault. *Formalisation en logique linéaire du fonctionnement des réseaux de Petri*. PhD thesis, Université Paul Sabatier (Toulouse III), 1997.
- [GT97] A. Gibaud and P. Thomin. Une approche objet pour la réalisation d'applications distribuées basée sur le modèle linda typé. In *NOuvelles TEchnologies de la REpartition '97*, Pau, November 1997.
- [GV97] Pierre Gradiat and François Vernadat. Les grammaires de réécriture de graphes pour la vérification d'algorithmes distribués. In *NOuvelles TEchnologies de la REpartition '97*, Pau, November 1997.
- [GV99] P. Gradiat and F. Vernadat. Layered δ -net specification of a workshop. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '99)*, June 1999.
- [HMM86] R. W. Harper, D. B. MacQueen, and R. Milner. Standard ML. Report ECS-LFCS-86-2, Department of Computer Science, University of Edinburgh, Edinburgh, UK, 1986. Also CSR-209-86.
- [Jan97] Matthias Jantzen. Basics of term rewriting. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages Vol. III: Beyond Words*, pages 269–337. Springer-Verlag, 1997.
- [Jen96] K. Jensen. *Coloured Petri Nets. Basic concepts, analysis methods and practical use*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1996.
- [Juh99] G. Juhas. Reasoning about algebraic generalisation of Petri nets. *Lecture Notes in Computer Science*, 1639:324–??, 1999.

-
- [KB70] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297, Oxford, 1970. (Oxford, 1967), Pergamon Press.
 - [KLG93] Simon Kaplan, Joseph Loyall, and Steven Goering. Specifying concurrent languages and systems with δ -GRAMMARS. In G Agha, P Wegner, and P Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, volume 99. MIT Press, 1993.
 - [Knu84] D. E. Knuth. *The T_EXbook*. Addison-Wesley, Reading, Massachusetts, 1984.
 - [KW86] H.-J. Kreowski and A. Wilharm. Net processes correspond to derivation processes in graph grammars. *Theoretical Computer Science*, 44(3):275–305, 1986.
 - [Lam86] Leslie Lamport. *A Document Preparation System L^AT_EX*. Addison-Wesley, 1986.
 - [Löw93] Michael Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109(1–2):181–224, March 1993.
 - [Mac71] Saunders MacLane. *Categories for Working Mathematicians*. Springer-Verlag, New York, 1971.
 - [Mar64] Karl Marx. *Le Capital (in Chap.13)*. , 1864.
 - [MAV96] François Michel, Pierre Azéma, and François Vernadat. Permutable agents in process algebras. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 187–206. Springer-Verlag, 1996.
 - [Maz87] Antoni Mazurkiewicz. Trace theory. In W. Brauer et al., editors, *Petri Nets, Applications and Relationship to other Models of Concurrency*, number 255 in *Lecture Notes in Computer Science*, pages 279–324, Berlin-Heidelberg-New York, 1987. Springer.
 - [Mes92] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.
 - [Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989. SU Fisher Research 511/24.
 - [MM90] José Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, October 1990.
 - [Mos94] Achour Mostefaoui. *Conception et expérimentation d’algorithmes pour la coopération répartie*. PhD thesis, Université Rennes I, 1994.
 - [MT99] José Meseguer and Carolyn Talcott. A partial order event model for concurrent objects. In Sjouke Mauw Jos C. M. Baeten, editor, *CONCUR ’99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands*, volume 1664 of *LNCS*, pages 415–430. Springer-Verlag, 1999.
-

- [MV86] G Memmi and J Vautherin. Analysing nets by the invariant method. In *Advances in Petri Nets 1987*, ed. Grzegorz Rozenberg, LNCS 266; *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986*, LNCS 254-255, 1987, LNCS 188 (1984), LNCS 340 (1988), LNCS 483 (1991). 1986.
- [MW97] Pierre Michel and Virginie Wiels. A framework for modular formal specification and verification. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313 of *Lecture Notes in Computer Science*, pages 533–552. Springer-Verlag, September 1997. ISBN 3-540-63533-5.
- [Pie91] B. C. Pierce. *A Taste of Category Theory for Computer Scientists*. MIT Press, 1991.
- [Plo81] Gordon D Plotkin. A Structural Approach to Operational Semantics. Tech. Rep. FN-19, DAIMI, Univ. of Aarhus, Denmark, September 1981.
- [RB88] D. E. Rydeheard and R. M. Burstall. *Computational Category Theory*. Prentice-Hall, New York, 1988. ISBN 0-13-162736-8.
- [Rei85a] W. Reisig. Petri nets with individual tokens. *Theoretical Computer Science*, 41(2-3):185–213, 1985.
- [Rei85b] W. Reisig. Petri nets with individual tokens. *Theoretical Computer Science*, 41(2-3):185–213, 1985.
- [Rei85c] Wolfgang Reisig. *Petri Nets, An introduction*, volume 4 of *EATCS, Monograph on theoretical Computer Science*. Springer Verlag, 1985.
- [Rei98] Wolfgang Reisig. *Elements of distributed algorithms*. Springer Verlag, 1998.
- [Reu89] Christophe Reutenauer. *Aspects mathématiques des réseaux de Petri*. Masson, Paris, 1989.
- [Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph transformation, Volume 1: Foundations*. World Scientific, 1997.
- [RV99] Davie Rossi and Fabio Vitali. Internet-based coordination environments and document-based applications: A case study. In *Proceedings of the Third International Conference on Coordination Models and Languages (COORDINATION'99)*, number 1594 in *Lecture Notes in Computer Science*, pages 259–274. Springer-Verlag, April 1999.
- [Sch90] Andy Schuerr. PROGRESS: A VHL-language based on graph grammars. Technical Report 90-16, Technical University of Aachen (RWTH Aachen), 1990.
- [Sch93] H Schneider. Graph grammars as a tool to define the behavior of process systems: From petri nets to linda. In *Fourth International Conference on Graph Grammars*, 1993.
- [Scu99] Adriano Scutellà. Simulation of conference management using an event-driven coordination language. In *Proceedings of the Third International Conference on Coordination Models and Languages (COORDINATION'99)*, number 1594 in *Lecture Notes in Computer Science*, pages 243–258. Springer-Verlag, April 1999.

-
- [Ser80] Jean-Pierre Serre. *Trees*. Springer Verlag, 1980.
 - [Sha97] Apu Shah. State of the art: JavaSpaces. *Java Report: The Source for Java Development*, 2(5):16–??, May 1997.
 - [Tae96] Gabriele Taentzer. Hierarchically Distributed Graph Transformation. In Janice E. Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg, editors, *Proc. 5th Int. Workshop on Graph Grammars and their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 304–320. Springer-Verlag, 1996.
 - [Val78] Rüdiger Valk. Self-modifying nets, a natural extension of Petri nets. In Giorgio Ausiello and Corrado Böhm, editors, *Automata, Languages and Programming, Fifth Colloquium*, volume 62 of *Lecture Notes in Computer Science*, pages 464–476, Udine, Italy, 17–21 July 1978. Springer-Verlag.
 - [Val86] R. Valette. Nets in production systems. In G. Rozenberg, editor, *Petri Nets: Applications and Relationship to Other Models of Concurrency, Advances in Petri Nets 1986, PART II, PO of an Advanced Course*, volume 255 of *LNCS*, pages 191–217, Bad Honnefs, September 1986. Springer-Verlag.
 - [Val98] R. Valk. Petri nets as token objects — an introduction to elementary object nets. *Lecture Notes in Computer Science*, 1420:1–25, 1998.
 - [VAM96] F. Vernadat, P. Azema, and F. Michel. Covering step graph. *Lecture Notes in Computer Science*, 1091:516–??, 1996.
 - [Ver89] François Vernadat. *Vérification formelle d'applications réparties: Caractérisation logique d'une équivalence de comportement*. PhD thesis, Université Paul Sabatier (Toulouse III), 1989.
 - [Vil95] Thierry Villemur. *Conception de services et de protocoles pour la gestion de groupes coopératifs*. PhD thesis, Université Paul Sabatier (Toulouse III), 1995.
 - [VVG95] François Vernadat, Thierry Villemur, Pierre Gradiat, and Michel Diaz. Distributed management of dynamic membership of agents in cooperative group. In *European Research Seminar on Advances in Distributed Systems*, pages 236–241, 1995.
 - [Weg97] P. Wegner. The Paradigm Shift from Algorithms to Interaction. *Communication of the ACM*, May 1997.
 - [WN95] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*. Oxford University Press, 1995. To appear.
 - [Zav98] G. Zavattaro. On the incomparability of gamma and linda. Technical Report SEN-R9827, CWI - Centrum voor Wiskunde en Informatica, October 31, 1998.

Table des matières

| | |
|---|-----------|
| Introduction | 1 |
| 1 Etat de l'art | 5 |
| 1.1 Le cas d'étude | 7 |
| 1.1.1 Edition électronique des actes d'une conférence | 7 |
| 1.1.2 Analyse de la spécification | 10 |
| 1.2 Réseaux de Petri | 17 |
| 1.2.1 Formes textuelles équivalentes | 18 |
| 1.2.2 Représentation graphique et cas d'étude | 19 |
| 1.2.3 Analyse structurelle des Réseaux Place/Transition | 20 |
| 1.2.4 Extensions des Réseaux de Petri | 21 |
| 1.3 La Logique de Réécriture | 27 |
| 1.3.1 Termes et théorie | 27 |
| 1.3.2 Dynamique et Réflexivité | 29 |
| 1.3.3 Séquents et Sémantique concurrente | 33 |
| 1.3.4 Termes de preuve | 35 |
| 1.3.5 Problème relatif aux variables libre à droite | 38 |
| 1.4 Les grammaires de graphes | 38 |
| 1.4.1 Graphes et morphismes | 39 |
| 1.4.2 Somme amalgamée | 43 |
| 1.4.3 Système et dérivation | 45 |
| 1.4.4 Sémantique opérationnelle | 49 |
| 1.4.5 Comparaison avec la logique de réécriture | 52 |
| 1.4.6 Le problème du typage | 53 |
| 2 Grammaires à double-somme | 59 |
| 2.1 Une théorie de fonctions | 61 |
| 2.1.1 La catégorie Set en ML | 63 |
| 2.1.2 Constructions universelles | 64 |
| 2.1.3 Constructions universelles dans Set | 67 |
| 2.1.4 Isomorphisme | 68 |
| 2.2 Sémantique dénotationnelle | 69 |
| 2.2.1 Les identifiants de transformations | 70 |
| 2.2.2 Sémantique d'un système | 74 |
| 2.3 Une algèbre de structures | 76 |
| 2.3.1 L'opérateur d'architecture | 76 |
| 2.3.2 L'opérateur famille | 79 |
| 2.3.3 Algèbre et première application | 80 |
| 2.4 Distribution et Apartés | 81 |
| 2.4.1 Des graphes de graphes... | 83 |
| 2.4.2 Architecture dynamique de groupes non-disjoints | 85 |
| 2.4.3 Spécifier un domaine coopératif | 88 |

| | |
|---|------------|
| 3 Comparaisons de formalismes | 91 |
| 3.1 Sémantique discrète de comportement et extension | 93 |
| 3.2 Ensembles de systèmes et sémantique | 94 |
| 3.2.1 Sémantiques du comportement | 95 |
| 3.2.2 Sémantique dénotationnelle des P/T | 96 |
| 3.2.3 Sémantique sans concurrence de la Logique de réécriture . . . | 97 |
| 3.2.4 Sémantique des grammaires à double-pushout | 102 |
| 3.3 Extension des réseaux P/T | 103 |
| 3.3.1 Logique de réécriture | 103 |
| 3.3.2 Grammaires à double somme | 107 |
| 3.4 Domaines des extensions réciproques | 111 |
| 3.4.1 Logique de réécriture | 111 |
| 3.4.2 Grammaires à double-somme | 113 |
| 4 Implémentation et Perspectives | 121 |
| 4.1 Spécifier le cas d'étude | 122 |
| 4.1.1 Un premier exemple | 123 |
| 4.1.2 Procès de la conférence | 124 |
| 4.1.3 Le processus des relectures | 125 |
| 4.2 Simuler en MAUDE | 130 |
| 4.2.1 Réécrire des structures en MAUDE | 131 |
| 4.2.2 le simulateur primitif | 131 |
| 4.2.3 Les termes de preuves | 133 |
| 4.2.4 Le module STEPPER | 133 |
| 4.2.5 Normalisation des variables d'état | 135 |
| 4.2.6 Contraindre la substitution | 136 |
| 4.2.7 Module d'information | 137 |
| Conclusion | 139 |
| <i>Notations</i> | 145 |
| Notes | 159 |

Table des figures

| | | |
|------|--|----|
| 1.1 | Classification et application aux phases du cas d'étude | 16 |
| 1.2 | Réseau avec marquage initial d'un article | 19 |
| 1.3 | Spécification par réseau du système | 20 |
| 1.4 | Enoncé et modèle d'une solution au problème des philosophes à n (marquage pour $n = 3$) | 22 |
| 1.5 | Le pique-nique des philosophes | 24 |
| 1.6 | Le bal des philosophes : une topologie dynamique | 25 |
| 1.7 | Représentation graphique de la transition change | 26 |
| 1.8 | Dynamique associée à une théorie de réécriture | 30 |
| 1.9 | Un philosophe mange depuis l'état rien | 33 |
| 1.10 | Deux philosophes rentrent et forment une table | 34 |
| 1.11 | Deux philosophes rentrent | 34 |
| 1.12 | Deux philosophes forment une table | 34 |
| 1.13 | Une "fusion de table" | 35 |
| 1.14 | Un philosophe prends deux baguettes | 35 |
| 1.15 | Terme de preuve équivalent aux figures 1.9 à 1.14 | 36 |
| 1.16 | Etiquetage univoque des éléments de la relation d'accessibilité | 36 |
| 1.17 | Exemples de graphes décrivant un état du problème des philosophes | 41 |
| 1.18 | Deux morphismes de graphes avec les conventions | 42 |
| 1.19 | Diagramme de Venn du morphisme $(m : x \rightarrow y)$ (Cf. Fig 1.18.2) | 43 |
| 1.20 | Somme disjointe et amalgamée | 44 |
| 1.21 | Système de règles pour le "gala des philosophes" | 46 |
| 1.22 | Exemple de dérivation directe | 48 |
| 1.23 | Règles changes ordonnées par interface croissantes | 50 |
| 1.24 | Complément de la somme | 51 |
| 1.25 | Un graphe typé (à gauche) par le graphe typant (à droite) | 54 |
| 1.26 | Une signature pour le gala des philosophes | 56 |
| 1.27 | Signature équivalente au graphe typant | 57 |
| 1.28 | Un graphe typé "correct" | 57 |
| 2.1 | Deux types de sémantiques | 60 |
| 2.2 | Le "bal des philosophes" à 3 philosophes (à deux philosophes en pointillés) | 61 |
| 2.3 | Diagrammes fondamentaux. | 65 |
| 2.4 | Un Co-produit dans Set avec les entiers (x, y, z ne sont pas forcément distincts) | 67 |
| 2.5 | Unicité du pushout modulo isomorphisme | 69 |
| 2.6 | Une règle: un diagramme de forme Span | 70 |
| 2.7 | Un identifiant concret: un diagramme de forme T | 71 |
| 2.8 | Diagramme du morphisme de T-diagramme | 71 |
| 2.9 | Diagramme de la preuve de la conservation de la source | 73 |
| 2.10 | Une transformation: une classe de T-diagramme | 73 |
| 2.11 | Etiquetage univoque de la Figure 2.2 | 75 |

| | | |
|------|---|-----|
| 2.12 | Construction du pushout dans la catégorie $G \Rightarrow X$ | 78 |
| 2.13 | DOM_2 est un aparté dans DOM_1 | 82 |
| 2.14 | Représentation par graphe de graphe d'une aparté | 84 |
| 2.15 | Une architecture d'apartés sous trois formes équivalentes (Venn, Abstrait Texte) | 84 |
| 2.16 | Règle de création d'un sas | 87 |
| 2.17 | Règles de franchissement d'un sas | 87 |
| 2.18 | Déroulement menant d'un groupe simple à une structure d'apartés | 88 |
| 3.1 | Sémantique d'entrelacement à transitions univoques | 98 |
| 3.2 | Equivalent en Logique de réécriture (sémantique nue) de la figure 3.1 | 106 |
| 3.3 | Un pushout avec des segments | 110 |
| 4.1 | Détection de la terminaison avec des arbres commutatifs | 123 |
| 4.2 | Le "seau TCP à jeton" (<i>TCP token bucket</i>) | 124 |
| 4.3 | Spécification de notre cas d'étude en Logique de Réécriture | 126 |
| 4.4 | Raffinement pour la relecture | 129 |
| 4.5 | Implementation des termes de preuves en MAUDE | 134 |

Notes

Auteur : Pierre Gradiat
Titre : Spécifications et Conceptions d'Applications Coopératives.
Directeur de thèse : Michel Diaz
Lieu de la soutenance : Laboratoire d'Architecture et d'Analyse des Systèmes.

Les applications coopératives permettent à des groupes d'utilisateurs distants de réaliser des objectifs communs. Une première étude nous permet de poser trois concepts clés pour l'analyse de ces applications: l'identité (une référence exclusive), la localité (une référence partagée) et la multiplicité (cardinalité d'une localité). Une caractéristique de ces applications est d'utiliser des architectures dynamiques d'éléments réactifs éphémères. Pour ce type d'architecture, les formalismes de spécification existants ne proposent pas d'approches précises intégrant ces trois concepts sans intervention du spécifieur.

Parmi les formalismes utilisés pour ce type d'approche, nous nous sommes d'abord concentrés sur les Réseaux de Petri Place/Transition (P/T) et les Grammaires de Graphes. Celles-ci permettent de réaliser une séparation entre l'architecture (dynamique) et le comportement (réactif) au lieu de confondre ces notions dans un seul réseau. Pour prendre en compte la diversité des Réseaux de Petri de Haut-Niveau, nous considérons la Logique de Réécriture et montrons qu'elle offre, pour ces réseaux, un cadre formel unifiant. Pour étendre cette unification, nous montrons que l'approche dénotationnelle suivie pour définir la sémantique de cette logique, déjà appliquée aux réseaux de Petri P/T, peut s'appliquer aux Grammaires de Graphes. Dans ce cadre sémantique unifié, tout Réseau de Petri P/T peut être traduit en une Théorie de Réécriture ou une Grammaire de Graphes équivalentes.

La recherche de cette unification a motivé une sémantique des Grammaires à Double-Pushout affranchie de contraintes opérationnelles. Pour cette démarche dénotationnelle la structure transformée devient un paramètre: il devient possible, par exemple, de considérer des graphes de graphes (graphes dynamiques dont les nœuds et arêtes éphémères sont des graphes dynamiques). Nous appliquerons cette construction à la spécification d'apartés dynamiques dans des groupes coopératifs.

Mots-Clés : Coopération, Réseaux de Petri, Logique de réécriture, Théories des Catégories, Equivalence de comportement, Extension de formalismes.

Cooperative applications allow group of distant users to reach common goals. A preliminary study hits three key concepts to analyse these applications: Identity (an exclusive reference), Locality (a shared reference) and Multiplicity (number of identities in a locality). These applications rely on dynamic structures of reactive and transient components. For that kind of structures, no existing formalism embeds our three concepts in a precise and native way.

Among formalisms used for cooperative application design, we focus on Place/Transition (P/T) Petri Nets then on Graph Grammars. These grammars allow identifying dynamic architecture with respect to reactive behaviour, instead of merging these notions in a single Petri Net. To take into account High-Level Petri Nets diversity, we study Rewriting Logic and assess how this logic can offer a powerful unifying frame for those Nets. To extend this unification, we show that denotational approach used to define Rewriting Logic semantics, and already applied over P/T Petri Nets, can be also applied to Graph Grammars. In such unified semantic frame, any Place/Transition Petri Nets can be translated in an equivalent Rewriting theory or Graph production rules.

Seeking that unification motivates the proposal of a Double-Pushout Grammar semantics free of operational constraints. Through that denotational approach, transformed architectures are parameters: considering dynamic graph of graphs becomes possible. Dynamic graphs are composed of transient edges and vertices that are both labelled with dynamic graphs. This construction finds a straightforward application in the specification of dynamic private subgroups in cooperative domains.

Keywords : Cooperation, Petri Nets, Rewriting Logic, Category theory, Behaviour equivalences, Formalism extensions.