THESE

présentée à

l'Université Louis Pasteur de Strasbourg
Département d'Informatique

par

David CAZIER

pour obtenir le grade de

Docteur de l'Université Louis Pasteur
Mention Sciences
Spécialité Informatique

Construction de systèmes de réécriture pour les opérations booléennes en modélisation géométrique

soutenue le 10 novembre 1997 devant la commisssion d'examen composée de :

M. Jean François Dufourd, directeur de Thèse,
professeur à l'université Louis Pasteur de Strasbourg.

M. Jean Françon, rapporteur interne,
professeur à l'université Louis Pasteur de Strasbourg.

M. Jean-Pierre Finance, rapporteur exterme,
professeur à l'université de Nancy.

M. Bernard Péroche, rapporteur exterme,
professeur à l'école de Mines de Saint-étienne.

M. Pascal Lienhardt, examinateur,
professeur à l'université de Poitiers.

M. Yves Bertrand, invité,
chargé de recherche au LSIIT.

Remerciements

La rédaction d'une thèse est un travail de longue alêne, parsemé de périodes de questionnements et de doutes. Pendant ces quatres années au département d'informatique de l'université Louis Pasteur de Strasbourg, j'ai bénéficié d'un environnement de travail cordial et stimulant, ce dont je doit remercier tous ses membres.

Je tiens particulièrement à remercier Mr. Jean-François Dufourd, professeur à l'université Louis Pasteur, pour avoir accepter de diriger ma thèse, mais surtout pour sa patience à toute épreuve, ses encouragements continus et les nombreuses discussions critiques et pertinantes que nous avons pu avoir.

Je souhaite également remercier Mrs. Jean Françon, professeur à l'université Louis Pasteur, Jean-Pierre Finance, professeur à l'université de Nancy et Bernard Péroche, professeur à l'école de Mines de Saint-étienne, qui ont accepté de juger mon travail et dont les remarques pertinentes m'ont permis d'améliorer ce rapport.

Je veux aussi remercier Pascal Lienhardt et Yves Bertrand, pour l'intérêt constant qu'ils ont porté à mes recherches et pour leur critiques et conseils précieux, notament dans le domaine de la modélisation géométrique. Merci également à Allan Fousse qui m'a stimulé en me proposant des exemples concrets qui m'ont permis d'appliquer mes résultats à des problèmes de raffinement en géologie.

Enfin et surtout, un grand merci à ma femme Sandrine qui a supporté mes abscences et errements, qui m'a encouragé et soutenu pendant ces quatres années et qui comprend maintenant ce qu'avoir un mari chercheur veut dire ...

Table des matières

Chapitre 1

Introduction

Les opérations booléennes – union, intersection, différence, etc. – sur des objets géométriques sont la base de bien des traitements en modélisation géométrique. Elles correspondent à l'envie de manipuler, de manière naturelle, des ensembles de points et sont la contrepartie mathématique de techniques d'usinage concrètes – soudure, fraisage, perçage, etc. – dont chaque individu possède au moins l'intuition. Elles ont été étudiées à de nombreuses reprises [72, 64, 40, 45, 24, 41]. Mais des problèmes se posent toujours quant à leur définition précise et complète et quant à leur robustesse [57, 49, 7].

D'abord, les opérations booléennes ont souvent été définies pour des objets dont la structure topologique n'était pas explicitement représentée, c'est-à-dire des objets uniquement considérés comme des ensembles de points, de segments ou de facettes. Dans ces approches, la topologie n'est introduite souvent que de manière ad hoc, après coup et pour des raisons de complexité algorithmique.

Nous voulons travailler dans un cadre de modélisation à *base topologique*, où les relations d'incidence et d'adjacence entre les parties des objets sont primordiales et doivent être maintenues tout au long des transformations subies par les objets. Ces relations topologiques facilitent les accès et les parcours de cellules, non seulement pour les opérations booléennes, mais dans tous les autres traitements à effectuer par ailleurs, notamment interactifs [12]. Une grande partie de notre travail consiste à reconstituer une topologie correcte pour des objets ayant subi des opérations booléennes, ce qui distingue notre approche de la plupart de celles que nous avons citées.

De plus, comme souvent en géométrie algorithmique, l'analyse, la conception et l'implantation des algorithmes correspondants sont longues et difficiles. Même si une solution naïve peut être décrite en quelques mots dans le cas général, les cas particuliers à prendre en compte et la nature des structures de données utilisées conduisent à des programmes volumineux, difficiles à maintenir et dont la correction est quasiment impossible à prouver.

Ce mémoire de thèse tente d'apporter des réponses à certaines de ces préoccupations, essentiellement par l'utilisation de *méthodes formelles* à tous les stades de la conception et de l'étude des objets, des opérations et des programmes correspondants, c'est-à-dire de l'analyse à l'implantation.

En premier lieu, dans le domaine de la modélisation, nous proposons une extension des *cartes combinatoires plongées* par l'introduction de *labels*, pour modéliser un ensemble d'objets géométriques complexes en dimension 2 ou 3. Nous donnons une description mathématique,

formelle et rigoureuse de ces objets, de leurs propriétés topologiques et géométriques et de leurs contraintes d'intégrité. Ceci nous permet de définir, de manière très générale, les opérations booléennes sur un nombre quelconque d'objets, grâce à la combinaison de deux puissantes opérations appelées *raffinement* et *complétion des labels*.

Deuxièmement, notre travail se situe dans la continuité des recherches menées à Strasbourg sur la *spécification formelle* d'univers d'objets et d'opérations géométriques, ayant notamment conduit à la conception et au développement d'un modeleur à base topologique [30, 32, 12, 14, 13, 34]. Nous décrivons formellement, par un ensemble de spécifications algébriques, les opérations permettant la construction et la manipulation de nos objets géométriques, tout en garantissant la conservation de leurs contraintes d'intégrité.

Troisièmement, grâce à une utilisation nouvelle de la *réécriture*, nous décrivons le processus générique de raffinement et de complétion des labels, à la base des opérations booléennes. A ce niveau, nous étudions et prouvons, en dimension 2, les propriétés de *terminaison* et de *confluence* des transformations élémentaires sous-jacentes.

Quatrièmement, nous dérivons, au sens du génie logiciel, un ensemble de stratégies d'évaluation de plus en plus efficaces. Ce résultat est obtenu par des modifications successives des systèmes de réécriture et par l'introduction de *structures de contrôle* adéquates. Cette approche nous permet d'étudier systématiquement la réduction de la complexité algorithmique jusqu'à l'obtention d'une stratégie optimale. Enfin, elle nous aide à réaliser facilement le passage à un *prototype*, puis à une implantation réelle.

Des approches formelles ont déjà été tentées dans des domaines connexes. Ainsi, elles se sont avérées fructueuses dans des travaux sur la conception de langages pour l'infographie [56], la démonstration automatique en géométrie [17, 18, 4] et la modélisation géométrique [5, 28, 29]. Les travaux strasbourgeois [32, 14, 13] utilisent la structuration horizontale et le raffinement vertical de spécifications algébriques pour des logiciels de grande taille.

Ici, nous combinons ces techniques avec la réécriture pour aborder un problème algorithmique réputé difficile, qui constitue une sorte de *benchmark* pour les questions de robustesse [79] et qui, à notre sens, est aussi un exemple-test dans le domaine des spécifications formelles et du développement de logiciels.

1.1 Les cartes combinatoires plongées et labellées

La représentation par les bords (B-rep) et la géométrie constructive du solide (CSG) sont les modèles les plus utilisés pour la description de solides polyédriques, en dimension 3. La plupart des modeleurs combinent d'ailleurs les deux méthodes pour offrir un plus large éventail d'applications. Dans ce cadre, la définition d'opérations booléennes est un problème crucial, soit pour évaluer la B-rep d'un arbre CSG [66, 3], soit pour calculer l'union, l'intersection ou la différence de deux solides en B-rep.

La définition des opérations booléennes soulève essentiellement des problèmes dans le cas d'objets géométriques décrits en B-rep pour laquelle de nombreux modèles ont été proposés [65, 55]. Il est courrant, dans ce cadre, de distinguer la *topologie* du *plongement*. La topologie définit les cellules de l'objet, c'est-à-dire ses sommets, arêtes, faces et volumes, ainsi que leurs relations d'incidence et d'adjacence. Le plongement définit la position et la forme de ces cellules, le plus souvent coordonnées de points pour les sommets, arcs de Jordan pour les

arêtes et éléments de surface pour les faces.

Dans la méthode que nous développons, il est important que les objets soient définis à partir de subdivisions d'espaces de dimension 2 ou 3, dont la topologie est bien définie et permet de bien sous-tendre le plongement. Aussi avons-nous exclu d'emblée les modèles de [64, 24, 45] où la topologie n'apparaît pas nettement, même s'ils ont servi à la définition d'opérations booléennes.

Pour des raisons de simplicité et d'approche formelle, il est bien sûr plus commode d'avoir un modèle topologique combinatoire et décrit précisément. Ceci nous conduit à écarter les modèles décrits en termes de structures de données concrètes, comme le modèle des arêtes ailées [6], ou d'autres définis de manière trop informelle comme le modèle de GWB décrit par l'intermédiaire des opérateurs d'Euler sous forme de schémas [60]. Ce type d'approche empêche une description précise et complète des contraintes d'intégrité que doivent vérifier les objets modélisés.

Notre approche algébrique nous fait préférer les modèles décrits, complètement, en termes d'algèbres sur des éléments d'un type unique. Aussi, nous excluons le modèle cellulaire des SGC [67] où apparaissent plusieurs types d'objets. Parmi les candidats possibles, nous avons le modèle des facet-edges [27], celui des arêtes radiales [76] ou celui des algèbres d'arêtes [44]. Mais, compte tenu de toutes les études menées jusqu'ici à Strasbourg, notre préférence va bien sûr aux modèles étendant les cartes combinatoires, c'est-à-dire fondés sur les notions de *brins* et de *permutations*.

Nous ne revenons pas, ici, sur la comparaison des différents modèles qui ont été présentés dans ce cadre, comme les cartes elles-mêmes, les cartes généralisées [54] ou les pavages [2, 70]. Une discussion à ce sujet peut être trouvée dans [55, 12]. Les cartes généralisées permettent la représentation de subdivisions de variétés non orientables, ce dont nous n'aurons pas besoin ici. Les pavages quant à eux sont équivalents aux cartes de dimension 3. Finalement, nous avons préféré les cartes de dimensions 2 et 3 que nous avions plus l'habitude de manipuler, mais la conversion dans les autres types de modèles ne présente aucune difficulté.

Les *cartes combinatoires plongées* [51, 74, 73, 19] conviennent bien à nos besoins, notamment parce qu'elles permettent une description commode, précise et concise des subdivisions surfaciques et volumiques, en séparant bien, mais en faisant coexister, les aspects de topologie et de plongement. De plus, les cartes sont définies algébriquement par une description mathématique de leurs propriétés et contraintes, et ainsi indépendamment de toute implantation.

Cependant, même avec les cartes, les objets géométriques sont souvent confondus avec le plongement des subdivisions utilisées pour les modéliser. Cette définition implicite limite l'ensemble des objets modélisables aux objets fermés et réguliers et les opérations booléennes à leurs versions régularisées. Pour le modèle des cartes, cela correspond aux variétés fermées, sans bord. En particulier, ces modèles ne permettent pas la définition d'objets dont le bord est incomplet ou contenant des trous, cette dernière limitation étant rédhibitoire en ce qui concerne les opérations booléennes.

Pour résoudre ce difficile problème, nous introduisons la notion de cartes *labellées*. Celle-ci nous permet de séparer la définition des objets du modèle topologique utilisé pour représenter les subdivisions surfaciques ou volumiques sous-jacentes. Ainsi, nous pouvons définir des objets ouverts ou fermés, avec ou sans bords, contenant des trous, des sommets isolés ou des arêtes et faces pendantes. La définition que nous donnons des opérations booléennes,

essentiellement basée sur cette nouvelle notion, est alors très générale et n'est pas limitée aux versions régularisées.

1.2 Les opérations booléennes en modélisation

En dimension 2, les opérations booléennes robustes faisant notamment intervenir, en même temps, un nombre quelconque d'objets sont très demandées. Dans le cas d'objets fermés, ces opérations apparaissent dans des domaines comme la cartographie [35], la conception de logiciels de dessins [40] ou la manipulation de projections d'objets 3D [41]. De plus, la manipulation d'ensembles ouverts de points est nécessaire pour modéliser les domaines de validité de certaines équations différentielles [67]. Mais leur conception et leur réalisation restent des tâches difficiles, ceci étant dû aux problèmes de robustesse et de bonne définition des objets et des opérations.

Ainsi, dans tous les cas, les problèmes de robustesse proviennent, pour l'essentiel, des erreurs d'approximations et d'arrondis dues à l'utilisation des nombres réels dans les calculs. Pour les résoudre, Hoffmann et Hopcroft [49] proposent de limiter les données géométriques en ne retenant que les équations de plans et ajoutent aux tests géométriques un système de raisonnement symbolique assurant qu'une nouvelle décision n'entre pas en contradiction avec les décisions antérieures. D'autres approches purement numériques visent à contrôler et limiter les erreurs d'arrondis, par exemple par l'utilisation d'intervalles de flottants [43], de rationnels [59], d'arithmétique entière multi-précision [38] ou, comme l'ont proposé Benouamer et al. [7], d'une arithmétique rationnelle paresseuse.

Si de nombreuses études semblent répondre avec succès aux questions d'approximations numériques, en revanche, des doutes sérieux persistent quant au domaine exact d'application des opérations booléennes qui ont été proposées. Plus précisément, il faut connaître le type d'objets effectivement concernés au départ et à l'arrivée et assurer la prise en compte de tous les cas particuliers. Ainsi, il est souhaitable de pouvoir accepter n'importe quelles données en entrée, provenant par exemple d'une saisie sans contrôles préalables, mais de restituer après opération un objet dont on garantit la totale correction. C'est dans ce cadre que se situe notre travail, où les opérations booléennes, notamment le raffinement, apparaissent comme une véritable procédure de contrôle et de restructuration des données sous forme d'un objet topologiquement et géométriquement sain.

Notre approche utilise en priorité la topologie, mais elle n'exclut aucune des solutions évoquées ci-dessus concernant les questions numériques liées au plongement. Au contraire, les informations topologiques récoltées durant la phase de raffinement nous permettent de vérifier la validité des calculs numériques et de corriger les éventuelles contradictions qu'ils contiennent. En effet, la nette séparation entre les aspects topologiques et les problèmes de plongement permet de localiser précisément les endroits où les approximations numériques jouent un rôle crucial.

1.3 Les méthodes formelles

Le raffinement des cartes, qui est au cœur de notre travail, est une extension des problèmes bien connus de recherche des intersections dans un ensemble de segments [8] et d'arrangements ou de cloisonnements de segments ou de plans [16]. Les algorithmes correspondants sont, en général, définis informellement, en termes de structures de données concrètes et pour des cas simplifiés, c'est-à-dire en éludant tous les cas nécessitant un traitement particulier, comme des points alignés par exemple.

De plus, dans ce domaine, la complexité des algorithmes est cruciale. Il est donc courant d'utiliser des structures de données complexes visant à optimiser les calculs [75, 58], comme des files de priorité et des dictionnaires binaires dans les stratégies de balayage du plan [62, 20] ou des graphes d'influence ou de conflits dans les algorithmes randomisés [16], avec parfois une apparition inopinée, dans ces structures, de pointeurs pour accéder à certains composants des objets.

Ces optimisations sont souvent décrites dans des pseudo-langages, voire même dans des langages procéduraux, aujourd'hui souvent les langages C ou C++. Ces descriptions informelles conduisent à des confusions importantes entre les structures de données utilisées pour implanter les modèles d'objets choisis et les structures servant à améliorer la complexité des calculs.

A la complexité intrinsèque des problèmes abordés s'ajoutent ainsi trois difficultés majeures. Premièrement, il est généralement impossible d'exhiber le domaine de validité ou les contraintes d'intégrité des modèles et opérations utilisés, en grande partie à cause de la confusion entre les différentes structures. Deuxièmement, même si la correction des résultats est, à la base, démontrée formellement sur les modèles mathématiques et les cas généraux, celle-ci n'est plus assurée pour les programmes devant prendre en compte tous les cas particuliers. Enfin, le passage de la description d'un algorithme au programme correspondant reste une tâche souvent plus ardue que la simple conception de l'algorithme.

Pour résoudre les trois difficultés susmentionnées, nous avons adopté deux stratégies complémentaires pour l'analyse, la conception et l'implantation des opérations booléennes. La première, qualifiée d'*horizontale*, vise à une description formelle, complète et exhaustive des objets manipulés et des opérations nécessaires à l'évaluation d'opérations booléennes. La seconde, qualifiée de *verticale*, consiste à raffiner, au sens du génie logiciel, cette description formelle, en vue d'obtenir des algorithmes de plus en plus efficaces.

Pour cela, nous employons des méthodes formelles qui sont l'objet d'un intérêt croissant dans la conception de programmes et dont une variété évoluée est formée par les spécifications algébriques [36, 9, 77] alliées à la réécriture [53, 25]. D'une part, cette approche nous permet de nous focaliser sur les aspects logiques et conceptuels du problème à résoudre, en créant des types de données abstraits et des opérations dont le comportement est décrit sous forme d'équations. D'autre part, une orientation correcte des équations en règles de réécriture permet de rendre opérationnelle la spécification, c'est-à-dire de l'exécuter pour débusquer d'éventuelles erreurs d'analyse ou de conception, selon une technique de prototypage logique.

Les stratégies horizontale et verticale sont étudiées depuis plusieurs années à Strasbourg, mais jusqu'ici uniquement dans le cadre de la logique équationnelle. Elles ont conduit à la spécification algébrique de modèles topologiques [32, 31, 33] et à la spécification détaillée et complète d'une hiérarchie d'objets et d'un logiciel de modélisation géométrique, Topofil,

développé dans ce laboratoire [30, 12, 14, 13]. Notre approche est dans la continuité de ces travaux, mais elle introduit de manière intensive la réécriture sur des problèmes algorithmiques complexes, où le plongement joue un très grand rôle pour guider les opérations topologiques.

Nous utilisons les spécifications algébriques qui sont bien adaptées à une description constructive, rigoureuse et homogène des opérations de modélisation géométrique et de raffinement. D'autres méthodes de spécifications ont été développées, comme VDM [52], Z [78, 71] ou B [1]. Elles privilégient les types basés sur la notion d'ensembles au détriment de ceux qui conduisent à des structures de données élaborées et mieux adaptées à notre problématique.

Les algorithmes de raffinement sont décrits sous forme de systèmes de réécriture à différents niveaux. D'abord, les règles de réécriture permettent de reproduire intuitivement les transformations élémentaires successives effectuées sur des ensembles d'objets. En outre, l'indéterminisme inhérent au déclenchement des règles de réécriture permet une description de très haut niveau des processus, conduisant à toute une classe d'algorithmes différents correspondant à diverses stratégies de déclenchement.

Il faut bien sûr s'assurer de certaines bonnes qualités des systèmes de réécriture comme la terminaison et la confluence dont les preuves ont été particulièrement étudiées. De plus, l'utilisation de systèmes de réécriture se montre particulièrement adaptée au raffinement vertical. Ainsi, par des transformations successives des systèmes de réécriture et l'ajout de structures de contrôle, nous avons dérivé une hiérarchie d'algorithmes correspondant à des implantations du raffinement de cartes de plus en plus efficaces.

Enfin, pour faciliter le passage des spécifications formelles des opérations étudiées à leur implantation, nous avons recours à un prototypage logique, réalisé en Prolog, des spécifications et systèmes de réécriture. D'autres langages de prototypage, comme OBJ3 [42] ou LARCH [46, 47], bien que basés sur des spécifications algébriques et de la réécriture, se sont révélés inadéquats dans notre cas, notamment à cause de la complexité des calculs mis en œuvre et de la nécessité d'une interface graphique indispensable pour juger de l'effet d'opérations géométriques.

Traditionnellement, les langages fonctionnels comme Caml permettent une implantation rapide de spécifications formelles. Mais dans notre cas, nous avons préféré Prolog parce que son moteur d'inférence et sa base de faits permettent une traduction immédiate des règles de réécriture. De plus, la version de Prolog utilisée permet un accès facile à une bibliothèque graphique. Les spécifications formelles et le prototypage logique conduisent ensuite à une implantation facile et correcte des algorithmes étudiés.

1.4 Structure et plan du mémoire

Dans le second chapitre, nous présentons le mécanisme de raffinement que nous employons pour l'évaluation d'opérations booléennes, en le comparant avec les méthodes plus classiques de découpage et recollement. Puis, nous rappelons les définitions mathématiques liées au modèle des cartes combinatoires, en dimension 2 et 3, et précisons le modèle de plongement utilisé, lié à l'orientabilité des cartes. Nous définissons alors les objets manipulés grâce, notamment, à la notion de labels et donnons les propriétés et les contraintes d'intégrité de ces objets. Puis, nous proposons une méthode pour modéliser et évaluer, de manière très générale, des expressions booléennes sur ces objets, en faisant le rapprochement avec les méthodes

classiques de modélisation par des arbres CSG.

Dans le troisième chapitre, nous donnons les spécifications algébriques des cartes et des opérations servant à leur manipulation. Nous spécifions notamment un ensemble d'opérateurs topologiques et géométriques nécessaires au raffinement des cartes. Dans ce cadre, nous donnons, dans l'annexe A, les spécifications des objets géométriques utilisés pour le plongement de cartes en dimension 2, tels que les points, vecteurs et segments. Nous spécifions également les opérations permettant la gestion et la manipulation des labels.

Dans le quatrième chapitre, nous présentons le système de réécriture générique réalisant le raffinement en dimension 2. Nous démontrons sa convergence en utilisant des techniques propres à la réécriture, comme la définition d'un *ordre sémantique de réduction* et l'étude systématique des *paires critiques*. Nous y définissons également le mécanisme de complétion des labels permettant l'évaluation d'opérations booléennes. Puis, nous évoquons l'influence des approximations numériques sur les propriétés logiques du système de réécriture et évoquons quelques pistes intéressantes pour leur maîtrise.

Dans le cinquième chapitre, nous décrivons la dérivation de cinq systèmes de réécriture correspondant à différentes stratégies pour l'évaluation du raffinement et de la complétion des labels. Nous y abordons les problèmes liés à la complexité des algorithmes issus de ces systèmes de réécriture, en montrant comment des propriétés géométriques peuvent être utilisées pour optimiser les stratégies de parcours. Nous décrivons, notamment, une stratégie de balayage du plan, par l'utilisation de structures de contrôle évoluées telles qu'une file de priorité et un arbre binaire de recherche.

Le sixième chapitre contient la définition complète du raffinement en dimension 3 qui fait appel à une combinaison de toutes les techniques formelles développées en dimension 2. Nous y détaillons ainsi, sous forme de spécifications algébriques et de systèmes de réécriture enrichis par des structures de contrôle, l'intersection de faces quelconques non coplanaires. Nous y évoquons également les techniques permettant de contrôler la validité et la cohérence des résultats numériques obtenus, par une confrontation de ces résultats avec les données topologiques recueillies durant cette intersection. Nous précisons ensuite le mécanisme d'intersection de faces coplanaires, qui utilise le raffinement déjà défini en dimension 2, et évoquons le principe de conversion d'une 2-carte en faces d'une 3-carte, par l'utilisation des cartes duales.

Dans le septième chapitre, nous proposons une méthode de prototypage des spécifications algébriques et des systèmes de réécriture en Prolog. Nous détaillons le passage de la spécification au prototype et discutons son intérêt et les choix possibles. En particulier, nous expliquons comment les termes modélisant les cartes sont représentés et proposons des méthodes systématiques de transposition des axiomes de la spécification. Puis, nous précisons l'implantation des systèmes de réécriture et particulièrement des structures de contrôle utilisées pour décrire les stratégies de parcours.

Dans ce chapitre, nous abordons également les problèmes liés à l'implantation en C des spécifications et systèmes de réécriture. Nous décrivons les structures de données concrètes utilisées et l'implantation des divers opérateurs définis. Nous montrons comment le travail réalisé durant le prototypage logique simplifie et facilite une implantation propre et rigoureuse des algorithmes décrits formellement.

Dans le huitième chapitre, nous présentons quelques résultats expérimentaux. Nous donnons notamment, en dimension 2, un exemple de raffinement correspondant à l'arrangement

d'un ensemble de segments. Nous proposons quelques exemples significatifs d'évaluation d'opérations booléennes et montrons comment sont gérés les problèmes liés à l'inclusion de composantes connexes. En dimension 3, nous donnons des exemples où le raffinement est utilisé, dans le cadre d'études géologiques, pour la construction de subdivisions volumiques du sol à partir de couches et de failles [48], afin de permettre une localisation efficace en vue de calculs en différences finies [39]. Pour finir, nous donnons des exemples de raffinement sur des objets plus proches de ceux qui sont classiquement utilisés en CAO.

Enfin, dans le dernier chapitre, nous concluons en rappelant les principaux résultats obtenus, en évoquant leur limitation et les difficultés rencontrées. Nous envisageons quelques prolongements possibles de nos travaux, tant au niveau de la modélisation géométrique que de construction de preuves de corrections complémentaires, et proposons des applications des méthodes formelles utilisées à d'autres domaines de la géométrie algorithmique.

Chapitre 2

Opérations booléennes, Cartes combinatoires et Raffinement

Dans ce chapitre, nous abordons les problèmes généraux liés à la définition d'objets modélisés par leur bord et aux opérations ensemblistes (union, intersection, différence) sur de tels objets. Dans la première section, nous rappelons les méthodes classiques d'évaluation d'opérations booléennes, en insistant sur la notion de raffinement qui est la base de ce travail. Dans la seconde section, nous présentons le modèle des cartes combinatoires qui est utilisé pour modéliser la topologie sous-jacente à un objet. Enfin, dans la troisième section, nous définissons précisément la notion d'objet, nous faisons le lien entre l'évaluation d'opérations booléennes et le raffinement des cartes combinatoires.

2.1 Opérations booléennes et raffinement

D'un point de vue général, on peut diviser les méthodes d'évaluation d'opérations booléennes en deux familles étroitement liées au modèle utilisé pour la définition des objets manipulés. Dans le premier cas, les objets de départ sont découpés en différents morceaux qui sont ensuite assemblés pour construire le résultat. Dans le deuxième cas, un seul objet est construit, *le raffinement*, qui contient les éléments nécessaires à l'évaluation de l'opération booléenne.

2.1.1 Opérations booléennes par découpe du bord

Pour des raisons historiques, les objets sur lesquels sont effectuées les opérations ensemblistes, dans les modeleurs, sont souvent représentés par leur bord et, précisément, ce sont des objets dont le bord est une variété. Dans ce cas, en dimension 2, les objets sont des polygones simples dont le bord est une ligne polygonale modélisée, par exemple, par une liste de sommets.

En dimension 3, les objets considérés sont des solides polyédriques dont le bord est une 2-variété, ou une surface fermée sans bord, modélisés par exemple grâce à une structure d'arêtes ailées [6]. Le déroulement d'une opération booléenne entre deux tels objets, A et B, se divise principalement en 4 étapes [66] :

1. Recherche des intersections existant entre les bords de A et B ;
2. Subdivision de A et B par découpage de leurs bords le long des intersections trouvées ;
3. Sélection des morceaux ainsi obtenus qui appartiennent au résultat ;
4. Construction du résultat par assemblage des parties sélectionnées.

Les opérations booléennes sur des objets dont le bord est une variété sont obligatoirement limitées à leur version régularisée. La régularisation d'une opération $*$ entre deux objets A et B, consiste à considérer que le résultat de $A * B$ est la fermeture de l'intérieur du résultat théorique de cette opération [72, 66]. La régularisation implique que le résultat de $A * B$ est encore un objet dont le bord est une variété et assure donc la stabilité de ces opérations par rapport à l'ensemble des objets modélisables.

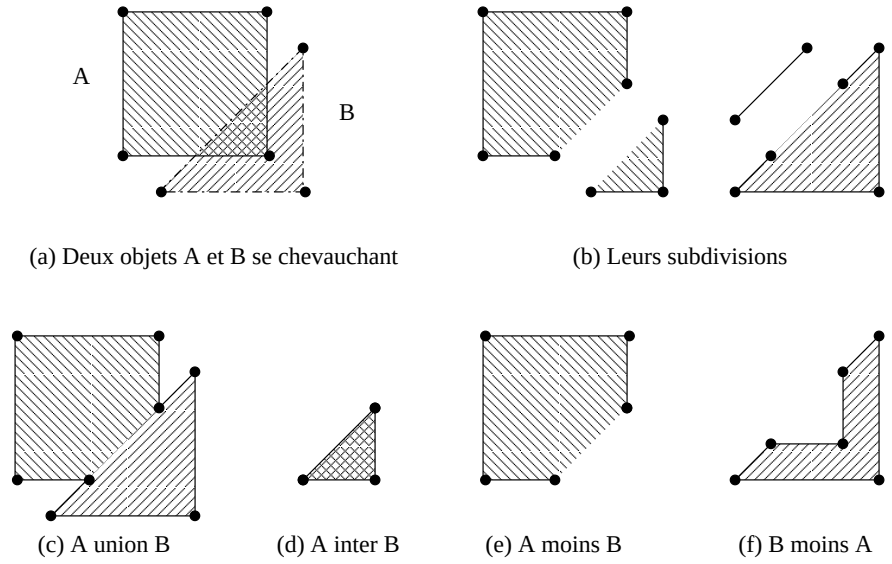


FIG. 2.1: *Exemples d'opérations booléennes en dimension 2*

Exemple 2.1.1 *La figure 2.1 montre un exemple de calcul avec la méthode mentionnée ci-dessus, en dimension 2. Un carré A et un triangle B sont définis par leur bord (a). Après avoir calculé les intersections entre leurs arêtes et sommets, les deux objets sont découpés en morceaux (b). Ces morceaux sont alors utilisés pour construire le résultat des différentes opérations booléennes entre ces objets (c).* □

Ce type de méthodes soulève plusieurs problèmes. D'une part, les opérations booléennes ainsi définies sont limitées à des objets représentés par leur bord et dont le bord est une variété. Ce type d'objets ne permet pas de représenter les cloisons internes que l'on peut souhaiter conserver dans le résultat. De plus, pendant la phase de découpe, les objets intermédiaires ne sont plus des objets représentés par leur bord, puisque ce bord peut être ouvert. Ceci conduit à manipuler plusieurs modèles pour les objets ou à manipuler des objets ne vérifiant pas les contraintes d'intégrité fixées pour le modèle initial.

D'autre part, ces méthodes sont peu robustes. En effet, comme Hoffman et Hopcroft l'ont montré [49], les approximations numériques, notamment lors de la phase de subdivision, peuvent conduire à des bords incompatibles pour les parties de A et B à recoller et

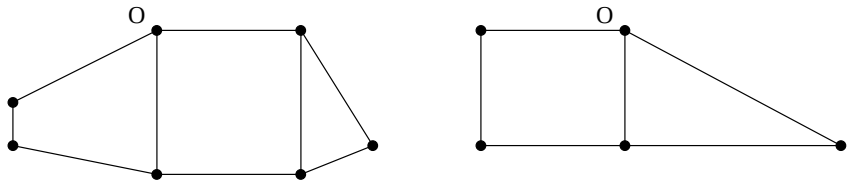
ainsi empêcher une reconstruction correcte. Comme mentionné précédemment, des méthodes numériques ont largement été utilisées pour résoudre ce type de problèmes, mais nous pensons qu'une approche différente, basée sur la topologie et une définition précise des objets manipulés s'impose ici.

2.1.2 Opérations booléennes par raffinement

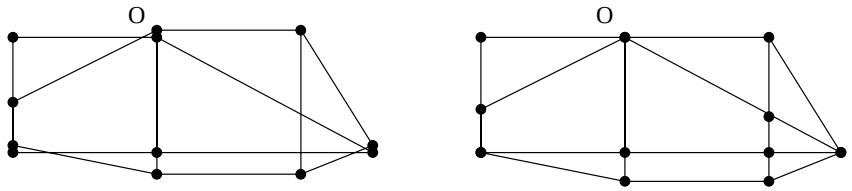
Une autre méthode a été introduite par Weiler [75] pour le 2D et étendue depuis au 3D [24]. Elle se déroule en 2 phases dont la première consiste en grande partie à découper les cellules de A et B en cellules plus fines, compatibles avec les 2 objets, ce que l'on appellera *coraffinement* :

1. Coraffinement de A et B par découpage des cellules sécantes de A et B et reconstruction des relations d'adjacence entre celles-ci ;
2. Sélection des parties du coraffinement appartenant au résultat.

D'un point de vue général, un objet peut être vu comme un ensemble de cellules, c'est-à-dire un ensemble de sommets, arêtes, faces en dimension 2, auquel on ajoute en ensemble de volumes en dimension 3.



(a) Deux subdivisions, O est l'origine d'un repère commun



(b) Leur superposition comme ensemble de sommets, aretes et faces

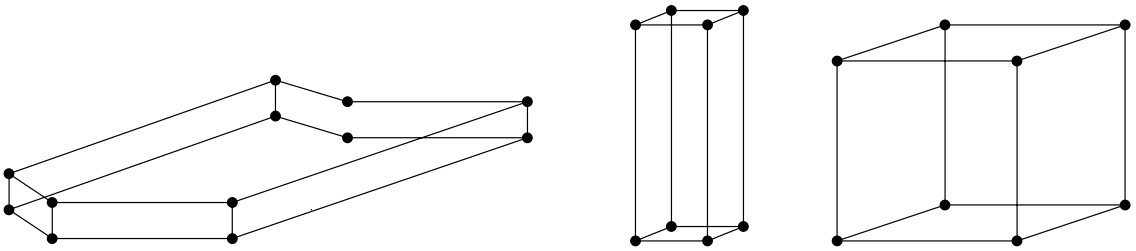
(c) Le raffinement de cet ensemble

FIG. 2.2: *Exemple de raffinement en dimension 2*

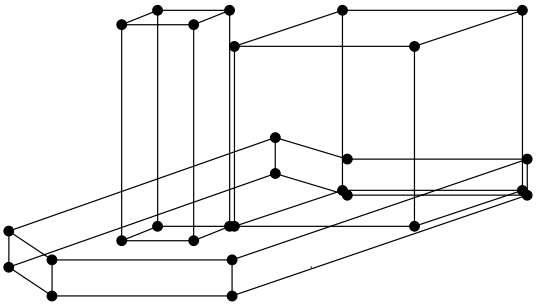
Exemple 2.1.2 *La figure 2.2 montre deux objets en dimension 2 (a) représenté séparément pour la clarté de la figure, mais partageant un point de référence commun O. Ces deux objets sont superposés en un ensemble de sommets, arêtes et faces (b) qui est ensuite auto-raffiné (c). Au total, les sommets, arêtes et faces superposés sont fusionnés, les arêtes et faces sécantes sont coupées en leurs lieux d'intersection qui sont ajoutés à l'ensemble construit en tant que sommets ou arêtes, et finalement les arêtes passant par un ou plusieurs sommets sont coupées aux points d'incidence.* □

Le coraffinement peut être généralisé à un nombre quelconque d'objets, en regroupant ceux-ci, dans un unique ensemble de cellules pouvant contenir des intersections (intersections franches, incidences, superpositions ou recouvrements). Nous parlerons alors de l'*auto-raffinement* ou, plus simplement, du *raffinement* de cet ensemble.

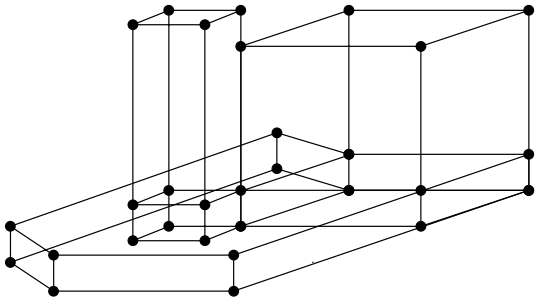
Pratiquement, le raffinement d'un tel ensemble consiste à construire un nouvel objet contenant toutes les cellules de l'ensemble de départ modifiées ou découpées pour prendre en compte les intersections existant entre elles. Ainsi, le résultat du raffinement est un objet dont l'ensemble des cellules ne contient plus aucune intersection. Le résultat de l'opération booléenne désirée est alors obtenu, à partir du raffinement, en y sélectionnant les parties utiles.



(a) Trois solides polyhedriques



(b) Leur superposition comme ensemble de sommets, aretes et faces



(c) Le raffinement de cet ensemble

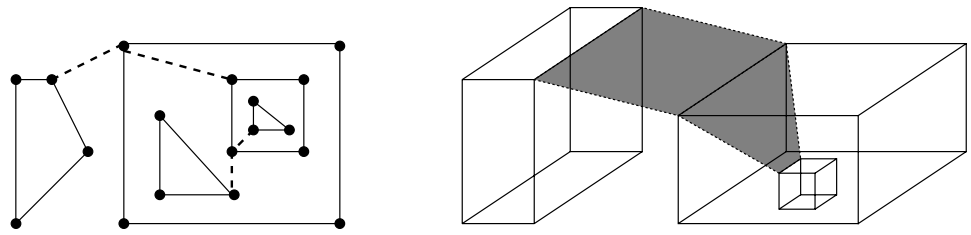
FIG. 2.3: *Exemple de raffinement en dimension 3*

Exemple 2.1.3 *La figure 2.3 montre trois objets en dimension 3 (a) qui sont superposés en un ensemble de cellules (b) qui est raffiné (c). Au total, les sommets, arêtes et faces superposés sont fusionnés, les arêtes et faces sécantes sont coupées en leurs lieux d'intersection qui sont ajoutés à l'ensemble construit en tant que sommets, arêtes ou faces, et finalement les arêtes et faces passant par un ou plusieurs sommets sont coupées aux points d'incidence.* □

Lorsque le résultat du raffinement comporte plusieurs composantes connexes distinctes, et donc ne s'intersectant pas, chacune d'elle est complètement incluse dans une face, en dimension 2, ou dans un volume, en dimension 3, des autres composantes. Diverses techniques permettent de construire et gérer ces relations d'inclusion entre composantes connexes.

La première est basée sur la construction d'un arbre dont chaque arête représente une relation d'inclusion entre deux composantes [37]. Elle impose donc la gestion d'arbres d'inclusion extérieurs au modèle utilisé pour représenter les objets.

La seconde, que nous utilisons, consiste à ajouter à l'objet modélisé des arêtes, en dimension 2, ou des faces, en dimension 3, reliant les différentes composantes connexes. Elle permet l'utilisation exclusive des relations d'adjacence ou d'incidence autorisées par le modèle.



(a) Un objet de dimension 2 contenant 5 composantes connexes

(b) Un objet de dimension 3 contenant 3 composantes connexes

FIG. 2.4: *Exemples d'inclusions de composantes connexes*

Exemple 2.1.4 *La figure 2.4 montre deux exemples d'objets formés de composantes connexes distinctes, en dimension 2 (a) et en dimension 3 (b). Les arêtes ou les faces ajoutées pour représenter les relations d'inclusion y sont représentées par des lignes pointillées ou des faces grisées.* □

Le modèle des cartes combinatoires de dimension n [55] que nous utilisons, permet de modéliser des subdivisions de variétés de dimension n. L'utilisation des cartes de dimension 2 et 3 pour modéliser respectivement des objets de $\mathbb{R}^2$ et $\mathbb{R}^3$, permet de représenter des objets contenant des cloisons internes ou des arêtes et faces pendantes. Cette possibilité est particulièrement intéressante, comme nous le verrons plus loin, dans le cadre d'une utilisation de raffinement. D'autres modèles, comme la structure d'arêtes radiales [76], permettent ce type de représentation. Le modèle des SGC (Selective Geometric Complexes) introduit par Rossignac, basé sur la notion de cellules, est utilisé avec une forme similaire de raffinement dont la trame est donnée dans [67].

L'utilisation du raffinement à l'avantage de conduire à des algorithmes permettant de contrôler la topologie des objets manipulés, ce qui les rend moins sensibles aux erreurs d'approximations. En effet, deux cellules s'intersectant sont découpées simultanément ce qui permet de contrôler, et ainsi conserver, la validité de la topologie de l'ensemble.

Des algorithmes particuliers peuvent être décrits en restreignant les types des ensembles de départ utilisés. Par exemple, l'ensemble de départ peut représenter un objet saisi sans vérifications ou sans contraintes, c'est-à-dire un objet pouvant contenir des intersections ou dont les relations d'incidence sont incomplètes. Le raffinement de cet objet correspond à sa restructuration topologique, au sens où, après son raffinement, l'objet ne contient plus aucune intersection et toutes les relations d'incidence ont été complétées. Si l'ensemble de départ est un ensemble quelconque de faces, le raffinement correspond à la construction d'une subdivision volumique basée sur ces faces. Dans la suite, nous verrons en détail le raffinement, en dimension 2 ou 3, d'objets modélisés par des 2- ou 3-*cartes combinatoires plongées* [55, 30, 13].

2.2 Cartes combinatoires

Un des principaux intérêts des cartes combinatoires est qu'elles permettent de distinguer, pour les objets manipulés, leur topologie de leur plongement. La topologie définit les cellules de l'objet, c'est-à-dire ses sommets, arêtes, faces et volumes, ainsi que leurs relations d'ad-

jacence et d'incidence. Le plongement définit la position et la forme de ces cellules, le plus souvent coordonnées de points pour les sommets, arcs de Jordan pour les arêtes et éléments de surface pour les faces. Les deux aspects coexistent bien dans la notion de carte combinatoire plongée qui permet une description commode, précise et concise des subdivisions planaires ou volumiques. Nous ne revenons pas ici sur la comparaison avec des modèles traditionnels tels que celui des arêtes radiales [76]. On trouvera une discussion à ce sujet dans [55, 13].

2.2.1 Définition de base

Nous présentons ici quelques notions de base sur les *cartes combinatoires* (voir [54, 55, 13] pour de plus amples détails). Les définitions des cartes et des opérations les manipulant font amplement appel aux permutations et à diverses notions qui leur sont liées. Nous commençons donc tout d'abord par quelques rappels mathématiques.

Définition 2.2.1 *Soit E un ensemble fini.*

- *Une permutation sur E est une bijection de E dans lui-même ;*
- *Toute permutation σ sur E peut être représentée sous forme de cycles. Si $x_1, \ldots, x_n$ et y sont des éléments de E, la notation $\sigma = (x_1, \ldots, x_n) \ldots (y)$ signifie que $\sigma(x_i) = x_{i+1}$ pour $i \in [1, n-1]$, $\sigma(x_n) = x_1$ et $\sigma(y) = y$;*
- *Une permutation σ de E est une involution si et seulement si pour tout x de E, $\sigma(\sigma(x)) = x$;*
- *Un élément x de E tel que $\sigma(x) = x$ est appelé un point fixe de la permutation σ.*

Ces quelques préliminaires nous permettent maintenant de définir formellement les cartes combinatoires. Nous nous limitons ici aux cartes de dimension 2 et 3 qui sont utilisées par la suite.

Définition 2.2.2 *Une 2-*carte *est un triplet (B, α_0, α_1) où :*

- *B est un ensemble fini dont les éléments sont appelés* brins *;*
- *α_0 est une involution sur B sans point fixe ;*
- *α_1 est une permutation sur B.*

Définition 2.2.3 *Une 3-*carte *est un quadruplet $(B, \alpha_0, \alpha_1, \alpha_2)$ où :*

- *B est un ensemble fini de brins ;*
- *α_0 et α_1 sont des involutions sur B et α_0 est sans point fixe ;*
- *α_2 est une permutation sur B telle que $\alpha_0 \circ \alpha_2$ soit une involution sur B.*

Le brin est l'élément de base pour la définition d'une carte. Il est possible de lui associer plusieurs sémantiques. La plus courante consiste à considérer un brin comme une demi-arête orientée. Alors, les fonctions α_i correspondent à la *liaison* ou à la *couture* de brins à chaque dimension i. Ainsi, deux brins images l'un de l'autre par α_0 sont dits *liés* par α_0, ou 0-*liés*, et forment une *arête topologique* de la carte. Pour définir des ensembles plus complexes de brins liés les uns aux autres, nous utilisons la notion d'*orbite* décrite ci-dessous.

Définition 2.2.4 *Soient E un ensemble fini, σ une permutation dans E et x un élément de E. On appelle orbite de x par rapport à σ l'ensemble noté $\langle\sigma\rangle(x)$ et défini par :*

$$\langle\sigma\rangle(x) = \{x, \sigma(x), \sigma^2(x), \ldots, \sigma^k(x)\}$$

où k est le plus petit entier naturel tel que $\sigma^{k+1}(x) = x$.

Notons que si σ est une involution, ses orbites contiennent au plus 2 éléments, et que si de plus σ est sans point fixe, ses orbites sont formées par exactement deux éléments.

Dans une carte dont l'ensemble des brins est B, l'orbite $\langle\sigma\rangle(x)$ d'un brin x de B par une permutation σ regroupe les brins liés les uns à la suite des autres par σ, à partir de x. Ainsi dans une 2-carte, une arête $\{x, \alpha_0(x)\}$ est l'orbite par α_0 d'un brin x. De même, $\langle\alpha_1\rangle(x)$, l'orbite de x par rapport à α_1, représente un *sommet topologique* de la 2-carte, c'est à dire un ensemble de brins incluant x et 1-liés entre eux.

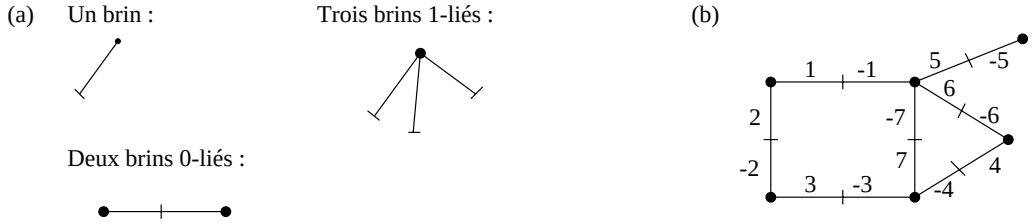


FIG. 2.5: *Représentation graphique et exemple de 2-carte*

Exemple 2.2.1 *La figure 2.5 montre une 2-carte dont l'ensemble des brins est* $\{1, -1, 2, -2, 3, -3, 4, -4, 5, -5, 6, -6, 7, -7\}$ *et les permutations, décrites sous forme de cycles, sont :*

- $\alpha_0 = (-1, 1)(-2, 2)(-3, 3)(-4, 4)(-5, 5)(-6, 6)(-7, 7)$;
- $\alpha_1 = (1, 2)(-2, 3)(-3, -4, 7)(4, -6)(-7, 6, 5, -1)(-5)$.

Ainsi, $\alpha_0(1) = -1$, $\alpha_0(-1) = 1$ et l'orbite $\langle\alpha_0\rangle(1) = \{1, -1\}$ forme une arête. De même, $\alpha_1(6) = 5$, $\alpha_1(5) = -1$, $\alpha_1(-1) = -7$ et $\alpha_1(-7) = 6$. Ainsi, l'orbite $\langle\alpha_1\rangle(6)$ vaut $\{6, 5, -1, -7\}$, ce qui représente l'ensemble des brins du sommet du brin 6. □

La notion de face orientée, dans une 2-carte, peut être définie très simplement. La fonction φ_1, définie par $\varphi_1(x) = \alpha_1(\alpha_0(x))$, permet de parcourir les brins d'une face orientée. Ainsi, dans une 2-carte C et pour tout brin x de C, on définit la *face topologique orientée* de x comme étant l'ensemble des brins de l'orbite $\langle\varphi_1\rangle(x)$.

Exemple 2.2.2 *Dans la figure 2.5, $\varphi_1(-7) = -3, \varphi_1(-3) = -2, \varphi_1(-2) = 1, \varphi_1(1) = -7$, et ainsi $\langle\varphi_1\rangle(-7) = \{-7, -3, -2, 1\}$ est l'ensemble des brins de la face orientée de 7 qui correspond rectangle de la figure. De la même façon, les brins 7, 6 et 4 correspondent triangle et les brins $-4, -6, 5, -5, -1, 2$ et 3 forment la face extérieure.* □

Pour les 3-cartes, on distingue deux types d'arêtes ou de sommets suivant que l'on prend en compte les liaisons par α_2 ou non. Ainsi, deux brins liés par α_0 forment une *arête simple* et une liaison par α_1 correspond à la 1-couture de deux brins dans un *sommet simple*. La figure 2.6 montre les conventions utilisées pour représenter les brins et les liaisons par α_0 et α_1 d'une 3-carte.

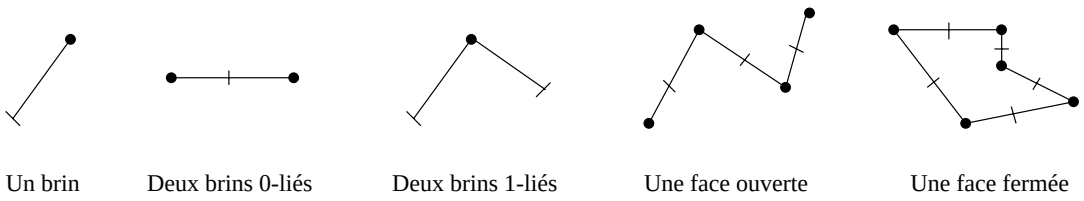


FIG. 2.6: *Exemple de faces pour les 3-cartes*

Comme pour les 2-cartes, la face orientée d'un brin x est l'orbite $\langle\varphi_1\rangle(x)$. Une face orientée est dite *fermée* lorsqu'elle ne contient aucun point fixe pour α_1, et *ouverte* dans le cas contraire. Notons que puisque α_1 est une involution, un ensemble d'arêtes simples 1-cousues deux à deux définit soit une seule face orientée ouverte, soit exactement deux faces orientées fermées qui sont symétriques l'une de l'autre dans le sens où les brins de la seconde sont les images de la première par α_0 et réciproquement. Nous appellerons *face* (non orientée) l'ensemble des brins de deux faces orientées fermées symétriques.

La liaison par α_2 correspond à la 2-couture de deux faces le long d'une arête. Un ensemble d'arêtes simples reliées entre elles par des 2-coutures forme une *arête multiple*, ou plus simplement une *arête*, de la 3-carte. On parle également de *sommets multiples*, ou de *sommets*, par opposition aux sommets simples.

La condition imposant que $\alpha_0 \circ \alpha_2$ soit une involution signifie que si x est 2-lié à y alors $\alpha_0(y)$ et 2-lié à $\alpha_0(x)$, c'est-à-dire que les 2-coutures sont régulières et réalisées sur des arêtes entières. Les faces sont 2-cousues entre elles pour former des *volumes*. Notons que plus de deux faces peuvent être 2-cousues autour d'une même arête.

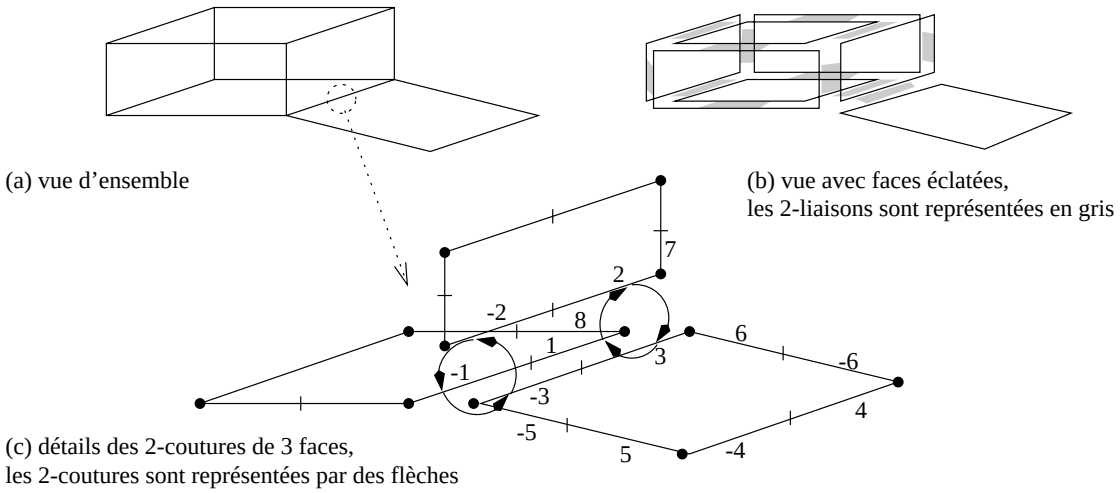


FIG. 2.7: *Exemple de carte et détails de 2-coutures*

Exemple 2.2.3 *La figure 2.7 montre une 3-carte (a), ses faces (b) et le détail des 2-coutures d'une arête (c). Pour les brins numérotés, les permutations sont en notation cyclique :*

- $\alpha_0 = (-1,1)\ (-2,2)\ (-3,3)\ (-4,4)\ (-5,5)\ (-6,6)$;
- $\alpha_1 = (-3,-5)\ (3,6)\ (4,-6)\ (-4,5)\ \ldots$;

– $\alpha_2 = (-4)\ (4)(-5)(5)(-6)(6)\ (-1,-3,-2)\ (1,2,3)$.

Autrement dit nous avons : $\alpha_0(1) = -1$, $\alpha_0(-1) = 1$ *et ainsi l'orbite* $\{1,-1\}$ *est une arête simple ;* $\alpha_1(-3) = -5$, $\alpha_1(-5) = -3$ *et donc* $\{-3,-5\}$ *est un sommet simple. Concernant les 2-liaisons, nous avons* $\alpha_2(-4) = -4$ *et donc l'arête simple* $\{4,-4\}$ *n'est pas 2-cousue ;* $\alpha_2(1) = 2$, $\alpha_2(2) = 3$ *et* $\alpha_2(3) = 1$ *ce qui implique que les faces contenant les brins* $1, 2$ *et* 3 *sont 2-cousues le long d'une même arête.*

On peut vérifier que les autres brins de ces arêtes simples, $-1, -2$ *et* -3*, sont 2-liés ensemble comme le demande la contrainte sur* α_2 *et que intuitivement cette dernière implique que les 2-liens entre* $-1, -2$ *et* -3 *sont réalisés dans le sens inverse de ceux qui existent entre* $1, 2$ *et* 3*. Enfin, les ensembles* $\{1,2,3,6,7,8\}$ *et* $\{1,2,3,-1,-2,-3\}$ *forment respectivement un sommet et une arête triples.* □

Les cartes de dimension n permettent de modéliser des subdivisions de variétés de dimension n *orientables* et *fermées*. Ainsi, une 2-carte est une subdivision de surface *sans bord* et une 3-carte, une subdivision de volume sans bord. Dans le cadre de notre travail, nous n'utilisons les cartes que pour modéliser des subdivisions du plan $\mathbb{R}^2$ ou de l'espace $\mathbb{R}^3$.

2.2.2 Cellules d'une carte et orientations

Les cartes permettent de définir des cellules pour lesquelles existe une orientation implicite. Nous avons déjà vu les notions de sommet, d'arête et de face qui sont les cellules de dimension 0, 1 et 2. Les cartes modélisent des subdivisions de variétés orientable. Ainsi, le choix d'un brin dans une cellule définit une orientation de celle-ci. Vue depuis le brin x, l'arête $\{x, \alpha_0(x)\}$ est orientée de x vers $\alpha_0(x)$. L'orientation d'une face orientée est directement héritée de celle de ses arêtes.

Pour les 3-cartes, une face contient, nous l'avons vu, deux faces orientées symétriques. Elles ont des orientations opposées puisque les brins de l'une sont les images par α_0 des brins de l'autre et appartiennent donc aux mêmes arêtes, mais avec des orientations opposées. Les deux faces orientées d'une face définissent intuitivement ses deux cotés.

Ainsi, si la face orientée d'un brin x correspond à un coté, nous dirons intuitivement qu'elle *pointe* vers le volume adjacent à ce coté, alors la face orientée de $\alpha_0(x)$ pointe vers le volume adjacent à l'autre coté.

La fonction φ_2, définie par $\varphi_2(x) = \alpha_2(\alpha_0(x))$, permet de passer de la face orientée de x poitant vers un volume à la face orientée adjacente à l'arête de x poitant vers le même volume. Ainsi dans une 3-carte, le volume de x est l'orbite $\langle \varphi_1, \varphi_2 \rangle(x)$ qui est l'ensemble des brins que l'on peut atteindre en appliquant un nombre quelconque de fois φ_1 et φ_2.

Intuitivement, le volume orienté de x est formé par les brins des faces orientées pointant vers un même volume et donc reliées deux à deux par φ_2. Les volumes héritent de l'orientation de leurs arêtes ou faces orientées. Les volumes orientés sont définis sans ambiguïté, nous parlerons donc à partir d'ici simplement de volumes.

2.2.3 Carte duale et vue constructive

La notion de carte duale permet de voir les cartes de façon constructive et donc plus intuitive. En changeant la sémantique associée aux brins et aux liaisons, on peut voir une carte comme un assemblage de cellules. Cette aspect nous sera utile par la suite.

Définition 2.2.5 *Le dual de la 2-carte $C = (B, \alpha_0, \alpha_1)$ est la 2-carte C^* définie par le triplet (B, ϕ_2, ϕ_1), avec :*

- $\phi_1 = \alpha_1 \circ \alpha_0$
- $\phi_2 = \alpha_0$

Définition 2.2.6 *Le dual de la 3-carte $C = (B, \alpha_0, \alpha_1, \alpha_2)$ est la 3-carte C^* définie par le quadruplet $(B, \phi_3, \phi_2, \phi_1)$, avec :*

- $\phi_1 = \alpha_1 \circ \alpha_0$
- $\phi_2 = \alpha_2 \circ \alpha_0$
- $\phi_3 = \alpha_0$

La sémantique qui nous intéresse pour les cartes duales est la suivante. Un brin est vu comme une arête orientée. Les brins sont liés par ϕ_1 pour former des faces orientées. Comme ϕ_1 est une permutation, les faces orientées sont des cycles de brins et donc toujours fermées. Les faces orientées sont cousues ensemble le long de deux arêtes d'orientations opposées par la fonction ϕ_2. Enfin pour le dual des 3-cartes, les volumes sont définis par un ensemble de face orientée cousues par ϕ_2. La fonction ϕ_3 permet de coller deux volumes le long de deux faces d'orientations opposées.

Exemple 2.2.4 *La figure 2.8 montre la vue duale de la 2-carte de la figure 2.5. Les brins et les liaisons par ϕ_1 sont représentés par des flèches, les liaisons par ϕ_2 par des traits épais gris. Notons que la face orientée externe apparaît ainsi explicitement et quelle a une orientation inverse à celle des autres faces orientées.* □

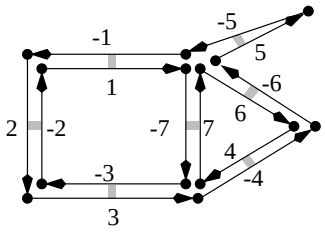


FIG. 2.8: *Représentation duale d'une 2-carte*

2.2.4 Plongements

Les cartes ne décrivent que la topologie de subdivisions. Décrire la géométrie d'un objet dont la topologie est modélisée par une carte, revient à plonger cette dernière. Cette opération consiste à associer à chaque cellule de dimension d d'une carte un objet géométrique de dimension d isomorphe à une boule unité ouverte de dimension d (un point, un segment

ouvert, un disque ouvert ou une sphère ouverte). Nous associons ainsi un point à un sommet, une courbe ouverte à une arête, une surface ouverte à une face et, pour les 3-cartes, un polyèdre ouvert à un volume. Le i-plongement d'un brin est alors l'objet géométrique associé à sa cellule (orientée) de dimension i.

Bien que des plongements plus sophistiqués puissent être définis, nous n'utilisons ici que des plongements linéaires, c'est-à-dire polygonaux pour les 2-cartes et polyédriques pour les 3-cartes. De plus, seuls les 0-plongements sont explicitement définis, les autres étant implicitement définis par leurs bords. Chaque sommet est 0-plongé sur un point de l'espace de plongement, c'est-à-dire un point de $\mathbb{R}^2$ pour les 2-cartes et de $\mathbb{R}^3$ pour les 3-cartes.

Définition 2.2.7 *Une n-carte plongée est un $n+2$-uplet $(B, \alpha_0, \ldots, \alpha_{n-1}, \pi_0)$, où :*

- *$(B, \alpha_0, \ldots, \alpha_{n-1})$ est une n-carte ;*
- *π_0 est une fonction qui à chaque sommet de B associe un point de $\mathbb{R}^n$;*

Ainsi, le 1-plongement d'une arête est l'intérieur du segment joignant les 0-plongements de ses sommets. Le 2-plongement d'une face est l'intérieur du polygone défini par les plongements de ses sommets et arêtes, ce polygone devant être plan pour les 3-cartes. Enfin, le 3-plongement d'un volume est l'intérieur du polyèdre défini par les plongements de ses sommets, arêtes et faces.

Il existe deux exceptions à ces définitions : la face non bornée d'une 2-carte et le volume non borné d'une 3-carte sont respectivement plongés sur l'extérieur d'un polygone et d'un polyèdre.

Définition 2.2.8 *Pour une n-carte plongée, $(B, \alpha_0, \ldots, \alpha_{n-1}, \pi_0)$, les i-plongements d'un brin x de B sont définis par :*

- *$\pi_1(x)$ = l'intérieur du segment orienté $[\pi_0(x), \pi_0(\alpha_0(x))]$;*
- *$\pi_2(x)$ = l'intérieur du polygone orienté $\{\pi_0(y), \pi_1(y)\}_{y \in \langle \varphi_1 \rangle(x)}$;*
- *$\pi_2(x)$ = l'intérieur du polyèdre orienté $\{\pi_0(y), \pi_1(y), \pi_2(y)\}_{y \in \langle \varphi_1, \varphi_2 \rangle(x)}$.*

Comme les cellules d'une carte sont orientées, les plongements de dimension supérieure ou égale à 1 héritent une orientation implicite. Dans une 2-carte, ceci permet de distinguer la droite et la gauche d'une arête et ainsi de définir l'intérieur d'une face comme étant à droite des arêtes qui la constituent.

De même, pour les 3-cartes, l'orientation permet de définir la normale d'une face pointant vers le volume auquel elle appartient. Enfin, cette orientation permet de distinguer les faces et volumes internes bornés des faces et volumes externes non bornés.

2.2.5 Partition de l'espace de plongement

Nous aurons besoin dans la suite de vérifier qu'une carte modélise une partition de son espace de plongement. Les notions de genre ou de caractéristique d'Euler peuvent être utilisées pour contrôler formellement si une 2-carte est *planaire* [51], c'est-à-dire plongeable dans le plan, ou si une 3-carte est plongeable sans auto-intersection dans $\mathbb{R}^3$.

Ces notions sont cependant uniquement topologiques et indiquent la possibilité d'un plongement qui réalise une partition. Elles ne suffisent pas à nos besoins car une carte planaire

peut être plongée de manière non-planaire, c'est-à-dire avec un plongement ne vérifiant pas les conditions ci-dessous.

Nous dirons qu'une carte modélise une partition de son espace de plongement lorsque l'ensemble de ses plongements à toute dimension est une partition de $\mathbb{R}^2$ pour les 2-cartes ou $\mathbb{R}^3$ pour les 3-cartes. Ce qui signifie que les plongements recouvrent entièrement l'espace de plongement sans aucune intersection. Nous parlerons alors de cartes bien plongées.

Définition 2.2.9 *Une n-carte plongée, $(B, \alpha_0, \ldots, \alpha_{n-1}, \pi_0)$, est bien plongée si l'ensemble de ses plongements, $\Pi(B) = \{\pi_0(x), \ldots, \pi_n(x)\}_{x \in B}$, vérifie les deux propriétés suivantes :*

- $\bigcup_{p \in \Pi(B)} p = \mathbb{R}^n$;
- $\forall p, q \in \Pi(B), \quad p \neq q \Rightarrow p \cap q = \emptyset$;

La définition implicite des plongements de dimension supérieure ou égale à 1 et l'orientation des cellules permet d'obtenir un nombre restreint de conditions à vérifier pour qu'une carte soit bien plongée.

Proposition 2.2.10 *Une 2-carte est bien plongée si elle vérifie les cinq conditions :*

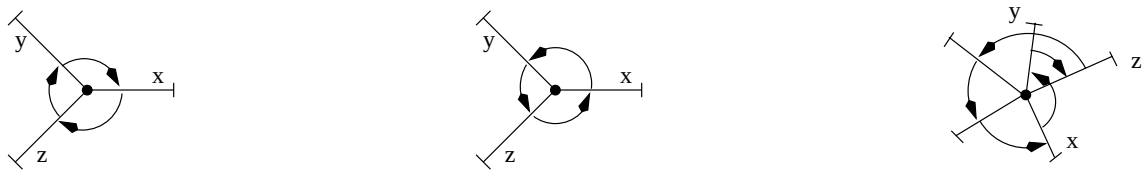
(*i*) *deux sommets distincts sont 0-plongés sur des points distincts ;*

(*ii*) *le 0-plongement d'un sommet n'est inclus dans aucun 1-plongement ;*

(*iii*) *deux arêtes distinctes sont 1-plongées sur des segments non sécants ;*

(*iv*) *tout sommet est bien classé ;*

(*v*) *une arête n'est pas 1-plongée sur un segment de longueur nulle.*



(a) mal classé (b) bien classé (c) mal classé

FIG. 2.9: Sommets bien et mal classés : les flèches représentent les 1-liaisons

La condition (*i*) assure que des brins 0-plongés sur des points égaux appartiennent au même sommet, ce qui implique que toutes les 1-liaisons possibles existent. Les conditions (*ii*) et (*iii*) assurent que les segments et points utilisés pour les plongements ne s'intersectent pas.

La condition (*iv*) signifie que lorsque les arêtes incidentes à un sommet sont parcourues dans le sens des liaisons par α_1, leurs 1-plongements tournent dans le sens trigonométrique autour du sommet (voir figure 2.9). Elle implique intuitivement que les faces ne se replient pas sur elles-mêmes.

Toutes ces conditions assurent que les intersections entre 2-plongements sont vides ce qui n'a donc pas à être vérifié. La condition (*v*), bien qu'elle ne soit pas nécessaire, est ajoutée pour la qualité de la représentation.

En exprimant les choses plus formellement en termes d'involution et en notant $angle(x)$ l'angle du segment $\pi_1(x)$ avec un axe quelconque, fixé dans le plan $\mathbb{R}^2$, on obtient :

Proposition 2.2.11 *Une 2-carte plongée, $(B, \alpha_0, \alpha_1, \pi_0)$, est bien plongée si elle vérifie les cinq conditions qui suivent :*

(i) $\forall x, y \in B, \quad \langle\alpha_1\rangle(x) \neq \langle\alpha_1\rangle(y) \Rightarrow \pi_0(x) \neq \pi_0(y)$;

(ii) $\forall x, y \in B, \quad \pi_0(x) \notin \pi_1(y)$;

(iii) $\forall x, y \in B, \quad \langle\alpha_0\rangle(x) \neq \langle\alpha_0\rangle(y) \Rightarrow \pi_1(x) \cap \pi_1(y) = \emptyset$;

(iv) $\forall x \in B, \quad \exists\, x_0 \in \langle\alpha_1\rangle(x)$ *tel que* $angle(x_0) = \min\{\text{angle}(y)\}_{y \in \langle\alpha_1\rangle(x)}$ *et tel que la suite* $\{\text{angle}(x_0), \text{angle}(\alpha_1(x_0)), \ldots, \text{angle}(\alpha_1^k(x_0))\}$ *soit croissante, k étant le plus petit entier tel que* $\varphi_1^{k+1}(x_0) = x_0$;

(v) $\forall x \in B, \quad \pi_1(x) \neq \emptyset$;

Avec le même principe, pour les cartes de dimension 3, on a :

Proposition 2.2.12 *Une 3-carte est bien plongée si elle vérifie les trois conditions :*

(i) *deux faces distinctes sont 2-plongées sur deux polygones dont les intérieurs sont d'intersection vide et dont les bords ne partagent que des segments non sécants et des points qui sont des 1- ou 0-plongements des brins des deux faces ;*

(ii) *deux arêtes distinctes sont 1-plongées sur des segments distincts ;*

(iii) *toute arête est bien classée ;*

La condition (i) assure que l'intérieur de deux 2-plongements est d'intersection vide. Elle implique, de plus, que les segments et points de leurs bords sont d'intersection vide ou égaux. Dans ce dernier cas, la condition (ii) assure que ce sont les 1-plongements de brins appartenant à une même arête (liés par α_2). Cette condition (ii) implique également que toutes les 2-liaisons possibles existent.

De la même manière que précédemment, la condition (iii) signifie intuitivement que les faces incidentes à une même arête tournent correctement autour de cet axe, ce qui implique que les polyèdres correspondants aux 3-plongements ne se replient pas sur eux-mêmes. Nous détaillerons plus loin, en termes de normale, la définition de cette condition.

La condition (i) implique qu'une 3-carte bien plongée respecte les contraintes fixées à la dimension 2, c'est-à-dire que des faces coplanaires vérifient les contraintes fixées pour les 2-cartes. Comme dans une 3-carte α_1 est une involution, la condition (i) pour deux faces coplanaires s'exprime naturellement de façon plus simple que dans le cas des 2-cartes.

Nous aurions pu écrire les conditions de bon plongement pour les 3-cartes de manière plus formelle, comme dans la proposition 2.2.11. Cependant, une telle description serait trop absconse ici et les détails de ces conditions seront longuement discutés lors de la définition du raffinement en dimension 3.

2.3 Evaluation d'arbres CSG et cartes labellées

2.3.1 Définition des objets et arbres CSG

Les cartes combinatoires que nous venons de définir ne permettent pas une définition explicite des objets. En général, on confond l'objet définit par une carte avec son plongement,

en ne considérant pas la face ou le volume extérieur. Ainsi, un objet modélisé par une carte est usuellement considéré comme étant l'ensemble des points, segments, polygones et polyèdres sur lesquels sont plongées ses différentes cellules.

Cette définition implicite empêche la modélisation de plusieurs objets au sein d'une même carte. De plus, elle limite la notion d'objets aux objets fermés et régularisés. En effet, dans une carte, il est impossible de distinguer une cellule de son bord. Ceci conduit forcément à limiter les opérations booléennes à leurs versions régularisées.

Pour résoudre ces problèmes et séparer la notion d'objet modélisé de la notion de subdivision fournie par les cartes, nous définissons un objet comme étant un ensemble quelconque de cellules d'une carte bien plongée. Cette définition permet de définir toutes sortes d'objets et en particulier des objets non régularisés, avec des faces ou des volumes troués et contenant des sommets ou arêtes isolés, des arêtes ou faces pendantes.

En restreignant cette définition, on peut aisément retrouver les définitions plus classiques, utilisées en CAO. Par exemple, en dimension 3, un objet formé uniquement de volumes internes d'une 3-carte et de toutes les faces, toutes les arêtes et tous les sommets incidents à ces volumes, définit un solide polyédrique fermé et régularisé au sens classique.

Avant de donner les définitions précises des objets, examinons les propriétés intuitives que l'on souhaite obtenir pour ce modèle. Le but final étant la définition d'opérations booléennes, il doit permettre de modéliser, à la fois, des objets désignés par un utilisateur et toute *expression ensembliste* sur ces objets.

Ainsi, nous commençons par définir des objets identifiés, c'est-à-dire auxquels on a donné un nom. Un objet (général) est alors une expression booléenne d'objets identifiés ou non. Il faut garder à l'esprit que de tels objets ne sont désignés, en fait, que par des expressions booléennes d'identificateurs.

Le but essentiel de cette approche est de fournir à notre modèle d'objets la puissance d'expression des modèles CSG, où un objet est définit comme une expression booléenne de *primitives* du modèle. Les objets que nous appelons identifiés joue ici le rôle de primitives. La différence essentielle est que nos objets identifiés peuvent être aussi complexes que possible, puisque formés d'un ensemble quelconque de cellules d'une carte sélectionnées par l'utilisateur. Ceci inclut bien sûr des primitives formées de cartes labellées par avance.

Nous incorporons cette notion d'objet au modèle des cartes en l'étendant par la notion de *label*, pour obtenir ce que nous appelons des *cartes labellées*. Un objet est un ensemble de cellules d'une carte donnée qui doit pouvoir contenir un nombre quelconque de tels objets. Ainsi, chaque cellule d'une carte peut appartenir à un ou plusieurs objets, voire même aucun si cette cellule n'est utilisée que pour en définir d'autres.

Pour représenter cette notion d'appartenance, nous associons à chaque brin d'une carte et pour chaque dimension i un *i-label*. Le i-label d'un brin est l'ensemble des identificateurs des objets auxquels appartient la cellule de dimension i de ce brin.

Définition 2.3.1 *Une n-carte labellée est un $(2n+4)$-uplet $(B, \alpha_0, \ldots, \alpha_{n-1}, \pi_0, I, \rho_0, \ldots, \rho_n)$, où :*

- *$(B, \alpha_0, \ldots, \alpha_{n-1}, \pi_0)$ est une n-carte plongée ;*
- *I est un ensemble d'identificateurs, ou de noms, d'objets ;*
- *$\rho_i : B \longrightarrow \mathcal{P}(I)$, $0 \leq i \leq n$, est une fonction qui, à un brin de B, associe l'ensemble des identificateurs des objets auxquels appartient sa cellule de dimension i ;*

Cette définition non restrictive des cartes labellées autorise la création de cartes avec des labels incomplets, c'est-à-dire que les brins d'une même cellule peuvent avoir des labels différents. De plus, une carte labellée peut être mal plongée et donc contenir des intersections.

Nous verrons que cette possibilité est utile pour la définition du raffinement d'une carte labellée et pour une évaluation locale et uniquement topologique des opérations booléennes. Cependant, pour une définition sans ambiguïté des objets, il est nécessaire de définir la notion de carte bien labellée, pour laquelle tous les brins d'une cellule de dimension i portent le même i-label. Les objets seront alors définit sur des cartes bien labellées et bien plongées.

Définition 2.3.2 *Une n-carte,* $(B, \alpha_0, \ldots, \alpha_{n-1}, \pi_0, I, \rho_0, \ldots, \rho_n)$, *pour* $n \in [2,3]$, *est bien labellée si :*

- $\forall x \in B$ *et* $\forall y \in \langle \alpha_1 \rangle(x)$ *on a* $\rho_0(y) = \rho_0(x)$;
- $\forall x \in B$ *on a* $\rho_1(x) = \rho_1(\alpha_0(x))$;
- $\forall x \in B$ *et* $\forall y \in \langle \varphi_1 \rangle(x)$ *on a* $\rho_2(y) = \rho_2(x)$;
- $\forall x \in B$ *et* $\forall y \in \langle \varphi_1, \varphi_2 \rangle(x)$ *on a* $\rho_3(y) = \rho_3(x)$, *pour* $n = 3$.

Un objet est alors constitué des i-plongements des brins d'une carte dont les i-labels vérifient certaines conditions. Comme les cartes doivent aussi bien représenter des objets explicitement définis que des objets résultant d'opérations booléennes, la définition des objets est réalisée en deux étapes. Pour un objet indentifié par o, la condition sur les labels des brins est qu'ils contiennent o, ce qui formellement donne la définition suivante :

Définition 2.3.3 *Soit* $C = (B, \alpha_0, \ldots, \alpha_{n-1}, \pi_0, I, \rho_0, \ldots, \rho_n)$ *une n-carte bien labellée et bien plongée. Un objet identifié* $\mathcal{O}$, *d'identificateur* o, *de* C *est défini par :*

$$\mathcal{O} = \bigcup_{i=0}^{n} \Big\{ \pi_i(x), \textit{tel que } x \in B \textit{ et } o \in \rho_i(x) \Big\}$$

Un objet est alors une expression booléenne d'objets identifiés. Les objets sont donc définis par induction comme des objets identifiés ou comme union, intersection ou différence de deux objets ou comme le complément d'un objet. La définition d'un objet s'écrit alors formellement :

Définition 2.3.4 *Un objet* $\mathcal{O}$ *est soit :*

- *un objet identifié ;*
- $\mathcal{O} = \mathcal{A} \cup \mathcal{B}$, *où* $\mathcal{A}$ *et* $\mathcal{B}$ *sont des objets ;*
- $\mathcal{O} = \mathcal{A} \cap \mathcal{B}$, *où* $\mathcal{A}$ *et* $\mathcal{B}$ *sont des objets ;*
- $\mathcal{O} = \mathcal{A} \setminus \mathcal{B}$, *où* $\mathcal{A}$ *et* $\mathcal{B}$ *sont des objets ;*
- $\mathcal{O} = \overline{\mathcal{A}}$, *où* $\mathcal{A}$ *est un objet.*

Cette définition des objets est similaire à celle des arbres CSG. Cependant son intérêt réside dans le fait que l'utilisation de la labellisation des cartes permet de transformer les tests géométriques, nécessaires dans le cadre de la CGS, en des tests sur les labels.

En effet, dans une carte C bien plongée et bien labellée, pour tester si le i-plongement d'une cellule appartient à un objet, il suffit de tester une condition sur le i-label d'un brin x de cette cellule. En notant $cond(\mathcal{O}, x, i)$ la condition d'appartenance du i-plongement d'un brin x de C à l'objet $\mathcal{O}$, la définition des objets devient :

Définition 2.3.5 *Soit $C = (B, \alpha_0, \ldots, \alpha_{n-1}, \pi_0, I, \rho_0, \ldots, \rho_n)$ une n-carte bien labellée et bien plongée. Un objet $\mathcal{O}$ de C est :*

$$\mathcal{O} = \bigcup_{i=0}^{n} \{\pi_i(x), \textit{tel que } x \in B \textit{ et } cond(\mathcal{O}, x, i)\}$$

avec :

- $cond(\mathcal{O}, x, i) = o \in \rho_i(x)$, *si $\mathcal{O}$ est un objet de base d'identificateur o ;*
- $cond(\mathcal{O}, x, i) = cond(\mathcal{A}, x, i) \vee cond(\mathcal{B}, x, i)$, *si $\mathcal{O} = \mathcal{A} \cup \mathcal{B}$;*
- $cond(\mathcal{O}, x, i) = cond(\mathcal{A}, x, i) \wedge cond(\mathcal{B}, x, i)$, *si $\mathcal{O} = \mathcal{A} \cap \mathcal{B}$;*
- $cond(\mathcal{O}, x, i) = cond(\mathcal{A}, x, i) \wedge \neg cond(\mathcal{B}, x, i)$, *si $\mathcal{O} = \mathcal{A} \setminus \mathcal{B}$;*
- $cond(\mathcal{O}, x, i) = \neg cond(\mathcal{A}, x, i)$, *si $\mathcal{O} = \overline{\mathcal{A}}$.*

Cette définition transforme ainsi une expression booléenne de primitives en une expression booléenne de conditions à vérifier sur les labels. De plus, elle permet une définition des objets sans ambiguïté. Une carte bien plongée réalisant une partition de son espace de plongement, les cellules de la carte sont bien définies et sans intersection. Un objet formé de ces cellules est donc correctement défini, au sens où chaque cellule est distincte des autres. On empêche, ainsi, que deux cellules se chevauchant puissent simultanément appartenir ou ne pas appartenir à un objet.

Dans la suite, et notamment dans les exemples, nous confondrons, pour simplifier, un objet identifié avec son identicateur. De même un ensemble d'indentificateurs sera représenté par un mot, par exemple A pour l'ensemble $\{A\}$ et AB pour l'ensemble $\{A, B\}$.

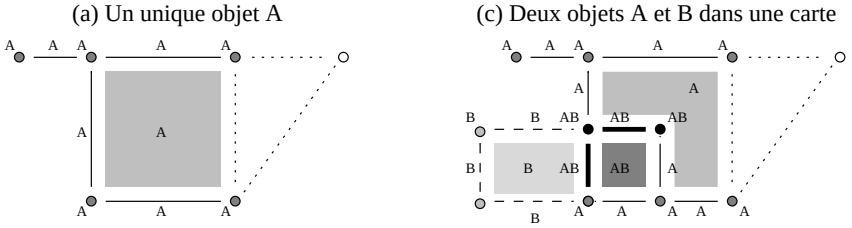


FIG. 2.10: *Exemples d'objets en dimension 2*

Exemple 2.3.1 *La figure 2.10 montre deux cartes labellées modélisant l'une un objet A et l'autre deux objets A et B. Pour plus de clarté, seul un label par cellule est représenté, sous la forme de lettres majuscules placées à coté de chaque cellule. L'objet A est le même dans les deux cartes. Il est formé, dans la carte de gauche (a), d'une face carrée, de 4 arêtes et 5 sommets (a). Dans la carte de droite (b), les cellules de A ont été subdivisées, mais l'union de leurs plongements est la même. L'objet B est formé de deux faces rectangulaires, de 4 arêtes et 4 sommets (b). Remarquons que les cellules appartenant à la fois à A et à B sont marquées AB dans la figure.*

Les conventions graphiques pour les cellules, en fonction de leur label, sont les suivantes : pour les faces et les sommets, ceux de label A sont en gris, ceux de label B en gris clair, ceux de label AB en gris foncé et les autre, i.e. ayant un label vide, sont en blanc. Pour les arêtes, celles de label A sont représentées par des segments pleins, celles de label B par des tirets, celles de label AB par des segments épais et les autres par des pointillés. □

Les objets, tels que nous les avons définis, sont donc des ensembles de points, puisque formés de l'ensemble des plongements de certaines cellules d'une carte sous-jacente. Ces ensembles de points peuvent être non réguliers et ouverts. Avec cette définition, un objet ne possède plus de structure topologique explicite. Cependant, il faut noter qu'un objet possède une topologie induite de sa carte sous-jacente. Les relations d'adjacence et d'incidence entre les cellules d'une carte se reflètent sur les ensembles de points formant les objets qui sont définis par la labellisation de cette carte.

Il était également possible définir les objets par des ensembles de cellules d'une carte, en considérant qu'un objet est une sous-carte de la carte labellisée dont les brins sont ceux qui vérifient les conditions décrites pour les labels. Nous aurions pu ainsi obtenir directement, aux niveaux des objets, les notions topologiques définies pour la carte sous-jacente. Mais dans ce cas, il n'était plus possible de modéliser des objets ouverts ou non réguliers puisque dans une carte une cellule est définie par son bord. De plus, le principe de sélection et d'extraction défini dans la section suivante permet d'obtenir une sous-carte minimale nécessaire à la modélisation d'un objet obtenu par évaluation d'opérations booléennes.

2.3.2 Evaluation d'opérations booléennes

Nous avons vu que pour évaluer un objet défini par une expression booléenne, il était intéressant de calculer le raffinement des objets de départ et que le résultat de l'opération était obtenu en sélectionnant les parties nécessaires du raffinement. Cette idée est généralisée à travers le raffinement des cartes labellées, de façon à prendre en compte la définition des objets donnée ci-dessus.

Considérons une carte labellée, mal plongée et mal labellée. Le raffinement de cette carte consiste à la transformer de manière à ce qu'elle soit bien plongée et bien labellée. Pour bien comprendre cette idée centrale, examinons ce que peuvent être de telles cartes. Supposons que nous avons deux cartes bien plongées et bien labellées modélisant, sans ambiguïté, deux objets A et B.

Si on fusionne naïvement ces deux cartes, on obtient une unique carte contenant deux composantes connexes qui peuvent s'intersecter. Cette carte est donc mal plongée. Le raffinement uniquement géométrique, comme défini précédemment, de cette carte la transforme en une carte bien plongée, mais sûrement mal labellée.

Une deuxième transformation que nous appelons *complétion des labels* et qui utilise uniquement les propriétés topologiques de la carte, c'est-à-dire les relations d'incidence et d'adjacence entre ses brins, la transforme en une carte bien labellée. Dans cette dernière, les cellules appartenant à l'intersection de A et B auront comme label l'ensemble AB.

Exemple 2.3.2 *Dans la figure 2.11, deux composantes connexes d'une carte définissent deux objets A et B. Celles-ci sont représentées séparément en (a) et (b), pour la clarté de la figure, mais partagent un point de référence O commun. Les conventions graphiques sont les mêmes que précédemment.*

Dans le raffinement (c) de cette carte, les cellules ont été subdivisées et les labels complétés. Le résultat de l'opération booléenne $B \setminus A$ est sélectionné (d) en vidant les labels des cellules n'appartenant pas au résultat, c'est-à-dire ici celles dont le label ne contient pas b ou contient a.

Si nécessaire, on peut ensuite extraire une carte plus petite représentant le résultat (e) en

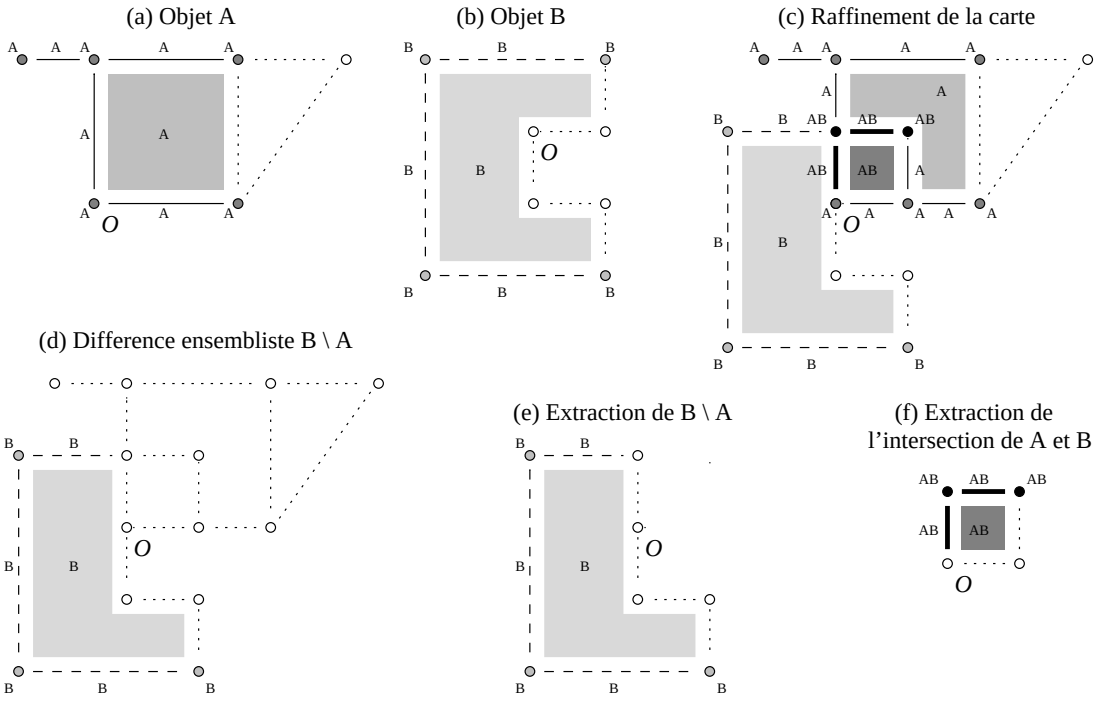


FIG. 2.11: *Exemple de raffinement d'une carte labellée*

supprimant les cellules dont tous les labels sont vides. La figure (f) représente l'extraction de $A \cap B$ *dont les cellules sont celles dont le label est AB.* □

Lorsque le résultat du raffinement contient plusieurs composantes connexes, chacune d'elles est complètement incluse dans une face en dimension 2, ou dans un volume en dimension 3, d'une autre. Ceci est assuré par le fait qu'après le raffinement, les bords de deux cellules distinctes ont une intersection vide. Remarquons qu'une composante connexe peut être incluse dans la face ou le volume extérieur des autres composantes. Ainsi les composantes connexes sont emboîtées les unes dans les autres, ce qui classiquement définit un trou, dans une face ou un volume des premières. Ceci peut empêcher le raffinement de compléter correctement les labels.

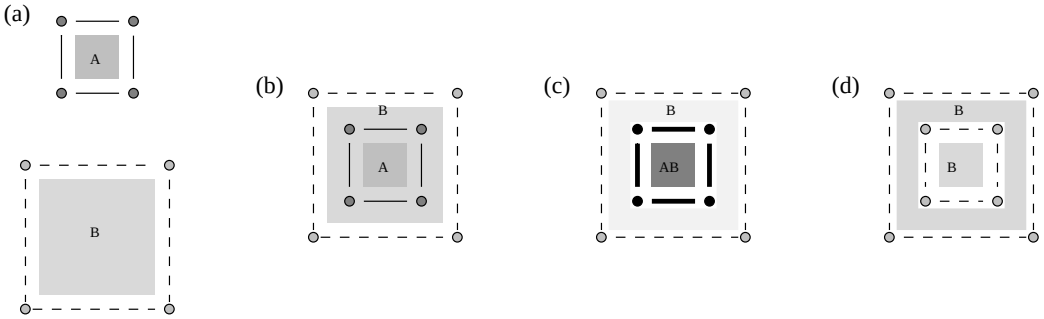


FIG. 2.12: *Exemples avec deux composantes connexes*

Exemple 2.3.3 *Dans la figure 2.12 (a) où seuls les labels de faces sont représentés, deux*

objets A et B sont modélisés. Ils sont superposés, en (b), pour que A représente un trou dans B. Le raffinement ne modifie rien à la carte (b).

En effet, les bords de A et B ne s'intersectent pas. Donc, les propriétés d'incidence ne suffisent pas pour déduire que A est situé à l'intérieur de B. Le résultat correct serait la subdivision (c).

Notons que si B est modélisé avec une face interne, comme en (d), alors le résultat correct est obtenu par le raffinement puisque les bords de A et B s'intersectent alors. □

Ce problème n'est pas lié à la définition que nous avons donnée pour le raffinement, mais est uniquement dû au fait que les cartes combinatoires ne permettent pas de représenter ce type de trous.

Les méthodes classiques pour résoudre ce type de problème passent par la construction d'un arbre d'inclusion pour les composantes connexes. Dans un tel arbre, une arête entre deux composantes indique que la deuxième est un trou dans une face ou un volume de la première. Cette technique implique l'utilisation d'une structure plus complexe, ce qui, à notre sens, n'est ni nécessaire ni satisfaisant.

La solution que nous utilisons consiste à ajouter une arête entre les deux composantes connexes. On peut, en effet, considérer qu'une arête de l'arbre d'inclusion a la même signification qu'une arête reliant deux composantes d'une carte. De plus, ceci permet de résoudre le problème, sans avoir à modifier la définition des objets et du raffinement des cartes labellées. La seule modification à apporter est que le raffinement, outre la subdivision des cellules sécantes, doit relier entre elles les composantes connexes.

2.3.3 Représentation et évaluation d'arbre CSG

Pour simplifier la présentation, nous avons présenté, dans la section précédente, des exemples d'opérations booléennes entre deux objets. Cependant, la définition des objets basée sur les cartes labellées et la définition du raffinement permet d'étendre ces méthodes à la définition et à l'évaluation d'arbres CSG. De plus ces définitions fournissent naturellement la possibilité d'éditer les opérations booléennes à calculer.

Rappelons qu'un arbre CSG est un arbre dont les feuilles sont des primitives et dont les nœuds sont étiquetés par des opérations booléennes. Les primitives d'un tel arbre peuvent être représentées par différentes composantes connexes d'une même carte labellée pouvant s'intersecter. Un arbre CSG peut donc être représenté par une carte mal plongée et mal labellée à laquelle on associe une expression booléenne sur les identificateurs d'objets.

L'évaluation d'un arbre CSG correspond alors au raffinement de la carte, puis à la sélection des cellules du raffinement dont les labels vérifient l'expression booléenne. De plus, le raffinement de l'ensemble des primitives de cet arbre est calculé en une seule fois. Ainsi, tous les arbres CSG ayant les mêmes primitives et des opérations différentes aux nœuds peuvent être évalués par simple sélection dans le raffinement, puisque la phase de sélection et séparée de la phase de raffinement.

Cela rend possible une édition interactive des arbres CSG [24], car la modification de l'expression booléenne n'entraîne qu'une simple modification de la sélection, puisque tous les arbres construits avec des primitives identiques conduisent à un raffinement identique.

Chapitre 3

Spécifications algébriques

Pour formaliser la notion de subdivision et pour éviter les problèmes dus aux structures de données concrètes et à la spécificité des langages de programmation, les cartes et les opérations les manipulant sont définies à travers une spécification algébrique [36]. Celle-ci décrit le comportement d'un ensemble de sélecteurs et constructeurs de cartes à travers une théorie équationnelle du premier ordre. Des détails complémentaires sur les spécifications algébriques des cartes et sur les opérations de base peuvent être trouvées dans [30, 13, 32, 14] et sur l'usage d'algèbres et de spécifications formelles en modélisation géométrique dans [34].

Malgrès nos efforts pour améliorer la lisibilité des spécifications données dans ce chapitre, elles restent parfois absconses. Cependant, notons tout de suite qu'il n'est pas nécessaire de les lire en détail pour comprendre la démarche adoptée. L'essentiel est de percevoir comment s'imbriquent les différents modules et d'examiner les profils des fonctions et parfois leurs préconditions. Les axiomes sont expliqués de manière plus informelle, dans le texte, les plus importants étant schématisés par des figures et des exemples.

3.1 Spécification de base

Nous utiliserons, dans la suite de ce chapitre, un certain nombre de spécifications standard que nous ne redonnons pas ici. Les premières sont BOOL et INT qui définissent respectivement les booléens et les entiers avec les opérations logiques et arithmétiques habituelles. Des exemples de telles spécifications peuvent être trouvées dans [42].

Nous aurons, de plus, besoin de manipuler des données géométriques. Les spécifications des opérations spécifiques à ce travail seront données par la suite et dans l'annexe A. Ces opérations utilisent des nombres, sous formes de rationnels, de réels ou de décimaux à virgules flottantes, et des opérations arithmétiques sur ces nombres. Nous utiliserons, dans la suite, une spécification FLOAT définissant le type Float correspondant, suivant le contexte, à l'un ou l'autre des types de données précédemment cités. Les spécifications classiques des rationnels apparaissent également dans [42] et des explications complémentaires sur l'utilisation des nombres sont données dans l'annexe A.

3.2 Spécification des 2-cartes

Dans cette section, nous spécifions algébriquement les 2-cartes et les opérations topologiques qui leur sont liées. Pour simplifier l'écriture des spécifications, nous utilisons les entiers pour représenter les brins.

3.2.1 Générateurs - Sélecteurs de base - Préconditions

Les 2-cartes sont définies à partir de trois *générateurs de base* $v, l0$ et $l1$ [30, 32]. Intuitivement, ces opérations sont les opérations élémentaires servant à la construction des 2-cartes, à partir d'une carte vide et par insertions et liaisons successives de brins. Plus précisément, l'opération v crée une carte vide. L'opération $l0(C, x, y)$ insère deux nouveaux brins x et y dans la carte C et crée une 0-liaison entre ces deux brins. L'opération $l1(C, x, y)$ crée une 1-liaison depuis le brin x vers le brin y.

Ainsi, dans la carte $l0(C, x, y)$, on a $\alpha_0(x) = y$ et $\alpha_0(y) = x$, le 0-lien étant symétrique puisque α_0 doit être une involution. Et dans la carte $l1(C, x, y)$, on a $\alpha_1(x) = y$, mais pas toujours $\alpha_1(y) = x$, puisque α_1 doit définir une permutation qui n'est pas nécessairement une involution. Nous aurons besoin plus loin que ces générateurs de base commutent entre eux, pour pouvoir considérer comme égales des cartes qui ont été construites en appliquant les générateurs dans un ordre différent.

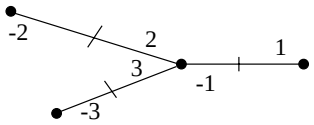


FIG. 3.1: *Une 2carte simple*

Exemple 3.2.1 *La carte de la figure 3.1 est représentée par un des termes suivants :*

$$\begin{aligned} C &= l1(l1(l0(l0(l0(v,-1,1),-2,2),-3,3),-1,2),2,3) \\ ou\ C &= l1(l0(l1(l0(l0(v,-3,3),-2,2),2,3),-1,1),-1,2) \end{aligned}$$

ou par tout autre permutation valide des générateurs de base. □

Le profil des générateurs de bases et les axiomes de permutations sont donnés dans la spécification 2-CARTE-GEN de la table 3.1 où nous utilisons un style d'écriture proche de celui qui est proposé dans [77]. Ainsi dans chaque spécification, le mot clé **Spéc** donne le nom du module de spécification, le mot clé **étend** précise les modules importés et étendus dans le nouveau module, le mot clé **avec** introduit le corps du module. Ce corps est divisé en quatre parties. La première partie, **Sortes**, décrit les nouvelles sortes introduites dans le module. La seconde, **Opérateurs**, donne le profil des opérateurs définis dans le module, alors que la troisième, **Axiomes**, et la dernière, **Préconditions**, expriment respectivement les axiomes et les préconditions précisant le comportement et l'usage de ces opérateurs.

Les sortes correspondent intuitivement à des types abstraits dont les éléments sont construits par les générateurs. Dans la spécification 2-CARTE-GEN, la sorte *Brin* est définie comme étant égale à la sorte *Int*. La sorte $2Carte$, définie dans ce module de spécification,

TAB. 3.1: *Générateurs des cartes*

Spéc *2-CARTE-GEN* **étend** *BOOL, INT* **avec**

Sortes $Brin = Int, 2Carte$ (*renomme la sorte Int en Brin*)

Opérateurs

$v : \longrightarrow 2Carte$	(*carte vide*)
$l0 : 2Carte\ Brin\ Brin \longrightarrow 2Carte$	(*0-liaison*)
$l1 : 2Carte\ Brin\ Brin \longrightarrow 2Carte$	(*1-liaison*)

Axiomes $(C : 2Carte\ ;\ x, y, z, t : Brin)$

$l0(l0(C, x, y), z, t) = l0(l0(C, z, t), x, y)$
$l1(l1(C, x, y), z, t) = l1(l1(C, z, t), x, y)$
$l0(l1(C, x, y), z, t) = l1(l0(C, z, t), x, y)$

Fin

recouvre l'ensemble des termes de type abstrait $2Carte$ et représentant une carte de dimension 2. Ces termes sont construits à partir des générateurs $v, l0$ et $l1$ qui sont les fonctions qui ont une valeur de retour de sorte $2Carte$. Les termes de sortes $2Carte$ sont donc soit v la carte vide, soit un terme composé d'insertions et liaisons successives dans la carte vide.

Les générateurs $v, l0$ et $l1$ permettent la construction d'ensembles de brins 0- ou 1-liés sans contraintes. Pour nous assurer que nous construisons bien des cartes, nous utilisons des préconditions qui définissent des contraintes limitant l'utilisation de chaque opérateur. Afin de pouvoir définir ces préconditions et pour pouvoir observer les cartes construites, nous définissons un ensemble d'*observateurs* ou de *sélecteurs* de base, $\alpha_0, \alpha_1, \alpha_{-1}$ et e, dont les deux premiers correspondent aux fonctions α_0 et α_1 utilisées dans la définition des cartes.

Ces sélecteurs devant permettre l'observation de toute carte, nous leur ajoutons un paramètre de sorte $2Carte$. Ainsi les sélecteurs $\alpha_0(C, x)$ et $\alpha_1(C, x)$ donnent respectivement l'image d'un brin x par les fonctions α_0 et α_1, dans la carte C. Le sélecteur $e(C, x)$ teste l'existence du brin x dans C.

Pour pouvoir définir α_1 comme une permutation totale, même si tous les liens ne sont pas explicitement construits, nous définissons la fonction $\alpha_{-1}(C, x)$ qui est l'inverse de α_1, et les fonctions $ald1, ali1$ et $al1$ qui testent la présence de 1-liens explicites sur le brin x. Nous avons vu que le générateur $l1(C, x, y)$ crée un 1-lien depuis x vers y. Nous dirons que ce lien est un 1-lien *direct* depuis x et un 1-lien *indirect* vers y.

Ainsi, $ald1(C, x)$ test l'absence de 1-lien direct depuis x, et donc, renvoie vrai si x n'a pas été explicitement lié à un brin y, par l'utilisation de $l1(C', x, y)$. Respectivement, $ali1(C, x)$ teste l'absence de 1-lien indirect vers x créé par l'utilisation de $l1(C', y, x)$. Et la fonction $al1(C, x)$ teste l'absence de 1-lien direct ou indirect sur x.

La figure 3.2 schématise l'étude de cas utilisée dans la spécification de α_1. Lorsque l'on cherche l'image, $z' = \alpha_1(l1(C, x, y), z)$, de z par α_1, dans une carte où il existe un 1-lien de x vers y, trois cas ce présentent.

Dans le premier cas (a), z appartient au sommet de x et y, mais il n'y a pas de 1-lien de z vers z'. Alors la suppression du 1-lien entre x et y déconnecte le sommet. Le brin z' recherché est l'image par α_1 de x, dans la carte C où le sommet a été éclaté.

Dans le deuxième cas (b), z appartient au sommet de x et y et il y a un 1-lien de z vers

TAB. 3.2: *Sélecteurs de base et préconditions des générateurs*

Spéc *2-CARTE* **étend** *2-CARTE-GEN* **avec**

Opérateurs

$e : 2Carte\ Brin \longrightarrow Bool$	(*existence*)
$\alpha_0, \alpha_1, \alpha_{\text{-}1} : 2Carte\ Brin \longrightarrow Brin$	(*fonctions d'adjacence*)
$al1, ald1, ali1 : 2Carte\ Brin \longrightarrow Bool$	(*absence de 1-liens,directs,indirects*)

Axiomes $(C : 2Carte\ ;\ x, y, z : Brin)$

$e(v, z) = faux$

$e(l0(C, x, y), z) = (z = x) \vee (z = y) \vee e(C, z)$

$e(l1(C, x, y), z) = e(C, z)$

$\alpha_0(l0(C, x, y), z) =$ **si** $z = x$ **alors** y
sinon si $z = y$ **alors** x
sinon $\alpha_0(C, z)$

$\alpha_1(v, z) = z$

$\alpha_1(l1(C, x, y), z) =$ **si** $z = x$ **alors** y
sinon si $z = \alpha_{\text{-}1}(C, y)$ **alors** $\alpha_1(C, x)$
sinon $\alpha_1(C, z)$

$\alpha_{\text{-}1}(v, z) = z$

$\alpha_{\text{-}1}(l1(C, x, y), z) =$ **si** $z = y$ **alors** x
sinon si $z = \alpha_1(C, x)$ **alors** $\alpha_{\text{-}1}(C, y)$
sinon $\alpha_{\text{-}1}(C, z)$

$ald1(v, z) = vrai$

$ald1(l1(C, x, y), z) =$ **si** $z = x$ **alors** $faux$ **sinon** $ald1(C, z)$

$ali1(v, z) = vrai$

$ali1(l1(C, x, y), z) =$ **si** $z = y$ **alors** $faux$ **sinon** $ali1(C, z)$

$al1(C, z) = ald1(C, z) \wedge ali1(C, z)$

Préconditions

prec $e(C, x) \equiv vrai$

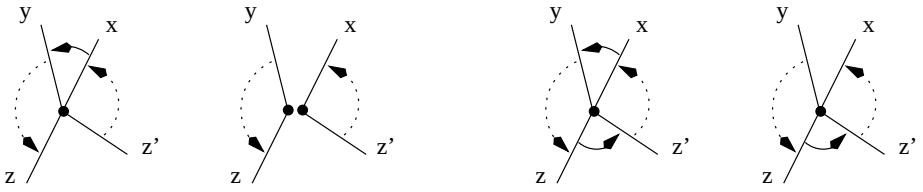
prec $l0(C, x, y) \equiv x \neq y \wedge \neg e(C, x) \wedge \neg e(C, y)$

prec $l1(C, x, y) \equiv x \neq y \wedge e(C, x) \wedge e(C, y) \wedge ald1(C, x) \wedge ali1(C, y)$

prec $\alpha_0(C, x) \equiv$ **prec** $\alpha_1(C, x) \equiv$ **prec** $\alpha_{\text{-}1}(C, x) \equiv e(C, x)$

prec $ald1(C, x) \equiv$ **prec** $ali1(C, x) \equiv$ **prec** $al1(C, x) \equiv e(C, x)$

Fin



(a) S'il n'y a pas de 1-lien depuis z vers z'

(b) S'il y a un 1-lien depuis z vers z'

FIG. 3.2: *Etude de cas pour la recherche de* $z' = \alpha_1(C, z)$

z'. Alors la suppression du 1-lien entre x et y ne change pas la configuration du sommet et la recherche peut continuer dans C. Le troisième cas n'est pas représenté et correspond au cas où z n'appartient pas au sommet de x et y. Ce cas est traité comme le second et la recherche continue dans C.

Les sélecteurs ci-dessus nous permettent de définir précisément les contraintes d'intégrité sur les cartes en donnant des préconditions à l'utilisation de $l0$ et $l1$. Ces contraintes assurent que, pour une carte C, α_0 et α_1 sont respectivement une involution sans point fixe et une permutation de l'ensemble des brins. Elles consistent à vérifier pour $l0(C, x, y)$ que les brins x et y sont distincts et n'existent pas déjà dans C et pour $l1(C, x, y)$ que x et y sont distincts, existent dans C et sont libres de 1-liens respectivement direct et indirect.

Les préconditions sont introduites par le mot clé **prec**. L'expression **prec** $op(x_1, \ldots, x_n) \equiv c(x_1, \ldots, x_n)$ signifie que l'opérateur op peut être utilisé avec les paramètres $x_1 \ldots x_n$ si la condition $c(x_1, \ldots, x_n)$ est évaluée à $vrai$. Les sélecteurs susmentionnés et les préconditions sur les générateurs de base sont définis dans la spécification 2-CARTE de la table 3.2.

Dans les spécifications, lorsque nous définissons un sélecteur, nous exprimons son comportement envers chaque générateur de base sous forme d'équations. En écrivant ces dernières, il apparaît que certaines signifient uniquement qu'un générateur, g, n'a pas d'effets sur le sélecteur, s. C'est le cas, par exemple, pour le troisième axiome définissant e. De telles équations sont de la forme $s(g(C, \ldots), \ldots) = s(C, \ldots)$.

Comme dans [56], pour simplifier la lecture des spécifications, nous omettons certaines de ces équations, dont nous dirons qu'elles sont *implicites*, en sachant qu'elles peuvent être reconstituées automatiquement. Ainsi dans la table 3.2, les axiomes entre, respectivement, α_0 et $l1$, et, α_1, α_{-1} et $l0$, n'apparaissent pas.

La spécification 2-CARTE décrit complètement les cartes de dimension 2 et les opérateurs permettant de les construire. La sorte $2Carte$ ainsi définie vérifie la définition mathématique qui en a été donnée. Il est, en effet, aisé de vérifier que α_0 et α_1 sont respectivement une involution sans point fixe et une permutation. Ceci pourrait être prouvé en utilisant les axiomes et les préconditions définissant ces fonctions, comme cela a été réalisé dans [33].

Une 2-Carte est représentée par une classe d'équivalence de termes de sorte $2Carte$ formés par des applications successives des générateurs de base dont l'ordre d'application est indifférent tant que leurs préconditions sont vérifiées. Une carte peut ainsi être représentée par n'importe lequel des termes de cette classe d'équivalence.

Nous avons défini un ensemble restreint de sélecteurs qui suffissent à observer les cartes. On peut noter que ces sélecteurs observent de la même façon tous les termes représentant une même carte.

3.2.2 Opérateurs topologiques

Pour pouvoir manipuler commodément les cartes et, en fait, les termes les représentant, nous aurons besoin d'opérateurs topologiques de plus haut niveau. Nous séparons ces opérateurs en trois familles : les constructeurs et les destructeurs qui modifient la topologie d'une carte en y ajoutant ou supprimant des éléments, et les sélecteurs, ou observateurs qui permettent d'observer les propriétés topologiques de ses brins.

L'ensemble de ces opérateurs est défini dans la spécification 2-CARTE-TOPO, décrite dans la table 3.3, étend la spécification 2-CARTE. Comme pour les sélecteurs, certains axiomes

implicites, de la forme $op(g(C,\ldots),\ldots) = g(op(C,\ldots),\ldots)$, où op est un opérateur et g un générateur, sont omis pour la description des constructeurs.

TAB. 3.3: *Opérateurs topologiques sur les cartes*

Spéc *2-CARTE-TOPO* **étend** *2-CARTE* **avec**

Opérateurs

$ca : 2Carte\ Brin\ Brin\ Brin \longrightarrow 2Carte$	*(coupe une arête en insérant deux brins)*
$dl1, ds : 2Carte\ Brin \longrightarrow 2Carte$	*(détache un brin de son sommet)*
$da, dai : 2Carte\ Brin \longrightarrow 2Carte$	*(détruit une arête, une arête isolée)*
$eqs, eqa : 2Carte\ Brin\ Brin \longrightarrow Bool$	*(teste l'égalité de 2 sommets, 2 arêtes)*

Axiomes $(C : 2Carte\ ;\ x, y, z, x', y' : Brin)$

$ca(l0(C,x,y),z,x',y') =$ **si** $z = x \vee z = y$
alors $l1(l1(l0(l0(C,x,x'),y,y'),x',y'),y',x')$
sinon $l0(ca(C,z,x',y'),x,y)$

$dl1(l1(C,x,y),z) =$ **si** $z = x \vee z = y$ **alors** $dl1(C,z)$
sinon $l1(dl1(C,z),x,y)$

$ds(C,z) =$ **si** $al1(C,z)$ **alors** C
sinon si $\alpha_{-1}(C,z) = \alpha_1(C,z)$ **alors** $dl1(C,z)$
sinon $l1(dl1(C,z),\alpha_{-1}(C,z),\alpha_1(C,z))$

$dai(l0(C,x,y),z) =$ **si** $z = x \vee z = y$ **alors** C
sinon $l0(dai(C,z),x,y)$
$da(C,x) = dai(ds(ds(C,x),\alpha_0(C,x)),x)$

$eqe(C,x,y) = (y = x) \vee (y = \alpha_0(C,x))$

$eqs(C,x,y) = eqs'(C,x,x,y)$
$eqs'(C,x_0,x,y) =$ **si** $(x = y)$ **alors** $vrai$
sinon si $\alpha_1(C,x) = x_0$ **alors** $faux$
sinon $eqs'(C,x_0,\alpha_1(C,x),y)$

Préconditions

prec $ca(C,z,x',y') \equiv e(C,z) \wedge$ **prec** $l0(C,x',y')$
prec $dl1(C,x) \equiv$ **prec** $ds(C,x) \equiv e(C,x)$
prec $dai(C,x) \equiv e(C,x) \wedge al1(C,x) \wedge al1(C,\alpha_0(C,x))$
prec $da(C,x) \equiv e(C,x)$
prec $eqs(C,x,y) \equiv$ **prec** $eqa(C,x,y) \equiv e(C,x) \wedge e(C,y)$

Fin

Deux sélecteurs supplémentaires, eqa et eqs, testent l'égalité de deux arêtes et celles de deux sommets. Pour cela, la fonction $eqa(C,x,y)$ teste si y appartient à l'arête de x, c'est-à-dire si y est égal à x ou à $\alpha_0(C,x)$. La fonction $eqs(C,x,y)$ teste si y appartient au sommet de x en parcourant ce sommet grâce à la fonction auxiliaire $eqs'(C,x_0,x,y)$, où x_0 est le brin de départ du parcours et x le brin courant.

Le seul constructeur défini ici est la fonction $ca(C,x,x',y')$ qui coupe une arête en deux. Dans une carte C, l'arête formée par les brins x et $y = \alpha_0(C,x)$ est coupée en deux en insérant les brins x' et y' entre x et y pour former deux nouvelles arêtes $\{x,x'\}$ et $\{y',y\}$. La figure 3.3 donne une description graphique de cette fonction. Pour réaliser cette opération, dans la carte

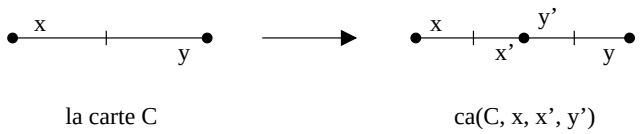


FIG. 3.3: *Illustration du constructeur* $ca(C, x, x', y')$

$ca(C, x, x', y')$ le 0-lien entre x et y est détruit, des 0-liens sont ajoutés entre x, x' et y, y' et des 1-liens sont ajoutés entre x' et y'.

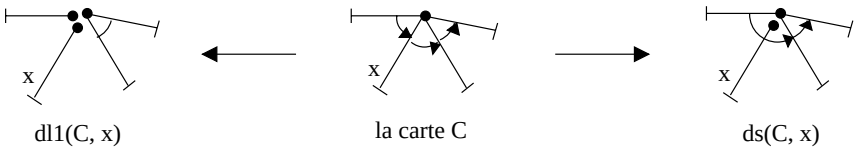


FIG. 3.4: *Illustration des destructeurs* $dl1(C, x)$ *et* $ds(C, y)$

Les premiers destructeurs servent à manipuler les sommets. Les fonctions $dl1(C, x)$ et $ds(C, x)$, représentées figure 3.4, détachent le brin x du sommet auquel il appartient. La différence réside dans le fait que $dl1$ détruit simplement toutes les 1-liaisons de x, tandis que ds, après avoir détruit les 1-liaisons de x, place un 1-lien entre les brins $\alpha_{-1}(C, x)$ et $\alpha_1(C, x)$, évitant ainsi l'éclatement du sommet auquel appartenait x.

Les autres *destructeurs* servent à supprimer des arêtes. La fonction $dai(C, x)$ détruit une *arête isolée* qui est une arête dont les deux brins sont libres de 1-liens. Précisément, $dai(C, x)$ supprime le 0-lien entre les brins x et $y = \alpha_0(C, x)$ et ainsi supprime ces deux brins de C. La fonction $da(C, x)$ détruit une arête quelconque en deux phases. Elle détache d'abord les brins x et y de leur sommet transformant l'arête $\{x, y\}$ en une arête isolée et utilise ensuite $dai(C, x)$ pour supprimer cette arête.

La spécification 2-CARTE-TOPO définit un ensemble d'opérateurs topologiques permettant la manipulation des 2-cartes. Cet ensemble de fonctions est suffisant pour la construction du raffinement qui sera décrite plus loin. Les opérateurs sont définis de telle façon que, s'ils sont appliqués à deux termes distincts représentant une même carte, alors ils donnent le même résultat.

3.2.3 Opérateurs de plongement

Nous décrivons, dans cette section, les opérations de plongement pour les cartes de dimension 2. Nous utilisons pour cela une spécification des objets géométriques de $\mathbb{R}^2$ qui définit les sortes *Point* et *Segment*. Dans celle-ci, un segment est défini par le terme $gs(p, q)$, où p et q sont deux points. La fonction $eqp(p, q)$ teste si les points p et q sont égaux. La spécification GEO-2D concernant les objets géométriques de $\mathbb{R}^2$ et les spécifications POINT-2D, VECTEUR-2D et SEGMENT-2D qu'elle utilise, sont décrites précisément dans l'annexe A.

Le plongement des sommets d'une carte est réalisé par la fonction $gp0(C, x, p)$ qui 0-plonge le sommet du brin x sur le point p, dans la carte C. Cet opérateur est décrit dans la spécification 2-CARTE-PLONGEE de la table 3.4. Notons que, dans cette dernière, un ensemble

d'équations implicites peut être ajouté automatiquement pour spécifier le comportement des opérateurs topologiques précédemment définis vis-à-vis de ce nouveau générateur.

Comme nous souhaitons que le plongement d'un sommet puisse être porté par n'importe lequel de ses brins, un axiome liant $gp0$ et $l1$ est ajouté. Cet axiome stipule que si deux brins sont 1-liés alors le fait de placer le plongement sur l'un ou l'autre, conduit à deux cartes égales.

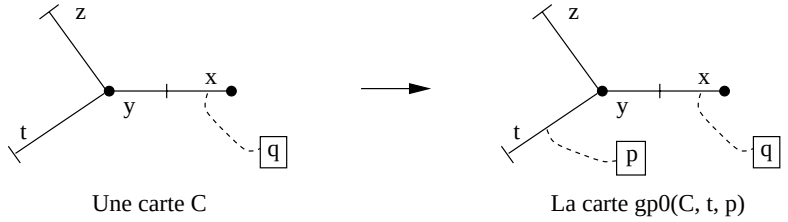


FIG. 3.5: *Exemple de plongement et conventions graphiques*

Exemple 3.2.2 *La figure 3.5 montre un exemple de plongement dans une carte C. Dans la carte $gp0(C, t, p)$ le sommet de t, contenant les brins t, y et z, est plongé sur p. L'utilisation de $gp0(C, y, p)$ ou $gp0(C, z, p)$ aurait conduit au même résultat. Les plongements sont schématisés par un nom de point encadré qui est relié, par une courbe en pointillée, à un des brins du sommet plongé.* □

Des axiomes supplémentaires sont également écrits pour exprimer le fait que le générateur $gp0$ doit pouvoir, pour les mêmes raisons que précédemment, commuter avec les autres générateurs. Ces axiomes et les équations implicites susmentionnées n'apparaissent pas dans la table 3.4.

De même que précédemment, nous définissons un ensemble de sélecteurs et de destructeurs de base pour le plongement. Les fonctions π_0 et π_1 renvoient respectivement les 0- et 1-plongements d'un brin. Ainsi, $\pi_0(C, x)$ donne le point sur lequel est 0-plongé le sommet de x dans C. La fonction $\pi_1(C, x)$ retourne le 1-plongement de x, c'est-à-dire le segment orienté $[p, q]$, représenté par le terme $gs(p, q)$, si x et $\alpha_0(C, x)$ sont 0-plongés sur p et q. Pour écrire les préconditions sur le plongement, nous avons besoin de la fonction $lp0(C, x)$ qui teste si x est libre de 0-plongement dans C, c'est-à-dire non plongé. Enfin, nous définissons un destructeur de plongement $dp0$ qui détruit le 0-plongement d'un sommet.

Les préconditions de plongement stipulent que seuls les brins libres de plongement peuvent être plongés et imposent que les sélecteurs π_0, π_1 et $dp0$ soient appliquées à des brins plongés. Les générateurs de base et les opérations topologiques ont été définies pour des cartes non plongées. Certaines de ces opérations doivent être contraintes pour ne pas agir sur des brins plongés, ce qui revient à ajouter des conditions de non-alignement à leurs préconditions.

Pour cela nous utilisons la syntaxe **prec** $op(x_1, \ldots, x_n) +\!\!\equiv c(x_1, \ldots, x_n)$ signifiant que la condition $c(x_1, \ldots, x_n)$ doit être ajoutée au préconditions précédentes de l'opérateur op. Ainsi, afin d'assurer qu'il n'y a toujours qu'un seul plongement par sommet, la fonction $l1$ ne peut lier deux brins que si l'un d'eux n'est pas plongé et les fonctions $dl1$ et ds ne peuvent agir que sur des sommets non plongés. De même, les fonctions da et dai ne peuvent détruire que des arêtes dont aucun des deux sommets n'est 0-plongé.

TAB. 3.4: *Spécification des opérations de base pour le plongement*

Spéc *2-CARTE-PLONGEE* **étend** *2-CARTE-TOPO,GEO-2D* **avec**

Opérateurs

$gp0 : 2Carte\ Brin\ Point \longrightarrow 2Carte$	(*générateur de 0-plongement*)
$\pi_0 : 2Carte\ Brin \longrightarrow Point$	(*0-plongement d'un brin*)
$\pi_1 : 2Carte\ Brin \longrightarrow Segment$	(*1-plongement d'un brin*)
$lp0 : 2Carte\ Brin \longrightarrow Bool$	(*liberté de 0-plongement*)
$dp0 : 2Carte\ Brin \longrightarrow 2Carte$	(*destruction d'un 0-plongement*)

Axiomes $(C : 2Carte\ ;\ x, y : Brin\ ;\ p : Point)$

$gp0(l1(C,x,y),x,p) = gp0(l1(C,x,y),y,p)$

$\pi_0(gp0(C,x,p),y) =$ **si** $eqv(x,y)$ **alors** p **sinon** $\pi_0(C,y)$
$\pi_1(C,x) = gs(\pi_0(C,x), \pi_0(C,\alpha_0(C,x)))$
$lp0(v,x) = vrai$
$lp0(gp0(C,x,p),y) =$ **si** $eqv(x,y)$ **alors** $faux$ **sinon** $lp0(C,y)$
$dp0(gp0(C,x,p),y) =$ **si** $eqv(x,y)$ **alors** C **sinon** $gp0(dp0(C,y),x,p)$

Préconditions

prec $l1(C,x,y) +\!\!\equiv lp0(C,x) \vee lp0(C,y)$
prec $dl1(C,x) +\!\!\equiv lp0(C,x)$
prec $ds(C,x) +\!\!\equiv lp0(C,x)$
prec $dai(C,x) +\!\!\equiv lp0(C,x) \wedge lp0(C,\alpha_0(C,x))$
prec $da(C,x) +\!\!\equiv lp0(C,x) \wedge lp0(C,\alpha_0(C,x))$
prec $gp0(C,x,p) \equiv e(C,x) \wedge lp0(C,x)$
prec $\pi_0(C,x) \equiv$ **prec** $dp0(C,x) \equiv e(C,x) \wedge \neg lp0(C,x)$
prec $\pi_1(C,x) \equiv e(C,x) \wedge \neg lp0(C,x) \wedge \neg lp0(C,\alpha_0(C,x))$

Fin

3.2.4 Opérations géométriques de base

Dans cette section nous définissons quatre opérations géométriques, c'est-à-dire des opérations qui agissent à la fois sur la topologie et le plongement d'une carte. La première, $dsp(C,x)$, détache le brin x de son sommet plongé et duplique son plongement. La seconde, $dap(C,x)$, détruit l'arête plongée de x. La troisième, $cap(C,x,p)$ coupe l'arête plongée de x au point p. La dernière $n(C)$ fournit deux nouveaux brins pour C. Toutes sont décrites dans la spécification 2-CARTE-GEO1 de la table 3.5. Les deux principales sont schématisées dans la figure 3.6.

TAB. 3.5: *Spécification des opérations géométriques de base*

Spéc *2-CARTE-GEO1* **étend** *2-CARTE-PLONGEE* **avec**

Opérateurs

$dsp : 2Carte\ Brin \longrightarrow 2Carte$	(*détachement d'un sommet plongé*)
$dap : 2Carte\ Brin \longrightarrow 2Carte$	(*destruction d'une arête plongée*)
$cap : 2Carte\ Brin\ Point \longrightarrow 2Carte$	(*coupure d'une arête plongée*)
$n : 2Carte \longrightarrow (BrinBrin)$	(*couple de nouveaux brins*)

Axiomes $(C : 2Carte\ ;\ x, y, x', y' : Brin\ ;\ p : Point)$

$dsp(C,x) =$ **si** $al1(C,x)$ **alors** C
sinon $gp0(gp0(ds(dp0(C,x),x),x,\pi_0(C,x)),\alpha_1(C,x),\pi_0(C,x))$

$dap(C,x) = da(dp0(dp0(dsp(dsp(C,x),\alpha_0(C,x)),x),\alpha_0(C,x)),x)$

$cap(C,x,p) = gp0(gp0(ca(C,x,x',y'),x',p),y',p)$
avec $(x',y') = n(C)$

$n(C) = (max(C)+1, max(C)+2)$
$max(v) = 0$
$max(l0(C,x,y)) = \max3(max(C),x,y)$

Préconditions

prec $dsp(C,x) \equiv e(C,x) \wedge \neg lp0(C,x)$
prec $dap(C,x) \equiv e(C,x) \wedge \neg lp0(C,x) \wedge \neg lp0(C,\alpha_0(C,x))$
prec $cap(C,x,p) = e(C,x)$

Fin

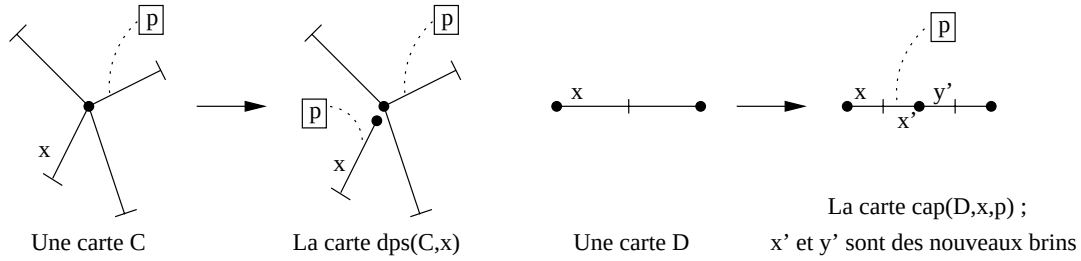


FIG. 3.6: *Représentation graphique des fonctions dsp et cap*

Plus précisément, la fonction $dsp(C,x)$ commence par détruire le plongement de x, puis détache x de son sommet et replonge x et un brin du sommet sur l'ancien plongement de x. La

fonction $dap(C, x)$ détache les sommets de l'arête avec dsp, puis détruit les 0-plongements de ses deux brins et enfin l'arête elle-même avec da. L'opération $cap(C, x, p)$ coupe l'arête $\{x, y\}$ plongée sur le segment $gs(q, r)$ en deux, au point p. Pour cela, elle coupe l'arête topologique avec $ca(C, x, x', y')$, où x' et y' sont deux brins fournis par $n(C)$, et plonge les brins x' et y' sur le point p. On obtient alors deux nouvelles arêtes $\{x, x'\}$ et $\{y', y\}$ plongées respectivement sur les segments $gs(q, p)$ et $gs(p, r)$.

La dernière fonction, $n(C)$, retourne un couple de nouveaux brins pour la carte C, obtenus par la fonction $max(C)$ qui donne le brin le plus grand de la carte C. Rappelons ici que les brins sont en fait des entiers. Les nouveaux brins sont alors simplement $max(C) + 1$ et $max(C) + 2$. La fonction $max3(i, j, k)$ retourne le plus grand des 3 entiers i, j et k. La fonction n retourne un couple de brins que nous avons représenté par la sorte $(Brin\ Brin)$. Nous aurions pu définir une sorte pour les couples de brins et les opérateurs permettant de générer ces couples et d'obtenir séparément les deux membres d'un couple. Cependant, pour simplifier, nous admettons ici les résultats multiples et les produits cartésiens de sortes.

3.2.5 Sélecteurs géométriques

Pour la définition du raffinement, nous aurons besoin de tester les conditions de planarité pour une carte. Dans la spécification 2-CARTE-GEO2 de la table 3.6, un ensemble de sélecteurs géométriques est défini. Il est basé sur un ensemble d'opérations de tests géométriques sur les points et segments dont la définition est donnée dans la spécification GEO-2D et qui est détaillé dans l'annexe A.

La fonction $eqsp(C, x, y)$ teste si les 0-plongements des sommets de x et y sont des points égaux, ce qui est réalisé par la fonction eqp définie dans l'annexe A. La fonction $nullap(C, x)$ teste si une arête est plongée sur un segment nul, c'est-à-dire dont les extrémités sont égales. L'opération $eqap(C, x, y)$ teste si deux arêtes sont plongées sur des segments égaux, c'est-à-dire si le plongement de leurs sommets sont égaux deux à deux.

Les autres opérations de 2-CARTE-GEO2 sont relatives aux recherches d'intersections. La fonction $incident(C, x, y)$ teste si le sommet de x est géométriquement incident à l'arête de y, c'est-à-dire si le point sur lequel est plongé x est à l'intérieur du segment sur lequel est plongé l'arête de y. La fonction $incident2(C, x, y)$ retourne $vrai$ si et seulement si un des sommets des arêtes de x et y est géométriquement incident à l'autre arête. La fonction $secant(C, x, y)$ teste si les arêtes de x et y sont sécantes, c'est-à-dire plongées sur des segments dont les intérieurs ont une intersection non vide. Et enfin, l'opération $intersection(C, x, y)$ retourne le point d'intersection entre deux arêtes sécantes.

3.2.6 Opérations géométriques sur les sommets

Une autre tâche du raffinement est de gérer le problème du bon classement des sommets. Pour cela, nous définissons un ensemble de fonctions permettant, d'une part, de tester si un sommet est bien classé et, d'autre part, de fusionner deux sommets classés. La fonction $class(C, x)$ teste si le sommet de x est bien classé. L'opération $fusion(C, x, y)$ fusionne deux sommets distincts, bien classés, qui sont 0-plongés sur des points égaux. Elle modifie les liaisons par α_1 existant dans ces deux sommets, pour former un unique sommet bien classé.

TAB. 3.6: *Spécifications des sélecteurs géométriques*

Spéc *2-CARTE-GEO2* **étend** *2-CARTE-GEO1* **avec**

Opérateurs

$eqsp : 2Carte\ Brin\ Brin \longrightarrow Bool$	(*égalité des plongements de 2 sommets*)
$nullap : 2Carte\ Brin \longrightarrow Bool$	(*nullité du plongement d'une arête*)
$lgap : 2Carte\ Brin \longrightarrow Float$	(*longueur du plongement d'une arête*)
$eqap : 2Carte\ Brin\ Brin \longrightarrow Bool$	(*égalité des plongements de 2 arêtes*)
$incident : 2Carte\ Brin\ Brin \longrightarrow Bool$	(*incidence d'un sommet à une arête*)
$incident2 : 2Carte\ Brin\ Brin \longrightarrow Bool$	(*co-incidence des sommets d'arêtes*)
$secant : 2Carte\ Brin\ Brin \longrightarrow Bool$	(*arêtes plongées sécantes*)
$intersection : 2Carte\ Brin\ Brin \longrightarrow Point$	(*point d'intersection de deux arêtes*)

Axiomes $(C : 2Carte\ ;\ x, y : Brin\ ;\ p : Point)$

$eqsp(C, x, y) = eqp(\pi_0(C, x), \pi_0(C, y))$
$nullap(C, x) = eqsp(C, x, \alpha_0(C, x))$

$lgap(C, x) =$ **si** $nullap(C, x)$ **alors** 1 **sinon** $lgseg(\pi_1(C, x))$

$$eqap(C, x, y) = [eqsp(C, x, y) \wedge eqsp(C, \alpha_0(C, x), \alpha_0(C, y))] \\ \vee [eqsp(C, x, \alpha_0(C, y)) \wedge eqsp(C, \alpha_0(C, x), y)]$$

$incident(C, x, y) = incident(\pi_0(C, x), \pi_1(C, y))$
$incident2(C, x, y) = incident2(\pi_1(C, x), \pi_1(C, y))$
$secant(C, x, y) = secant(\pi_1(C, x), \pi_1(C, y))$
$intersection(C, x, y) = intersection(\pi_1(C, x), \pi_1(C, y))$

Préconditions

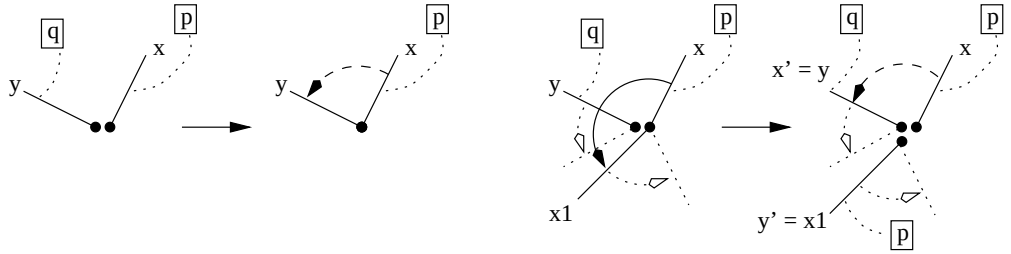
prec $eqsp(C, x, y) \equiv$ **prec** $\pi_0(C, x) \wedge$ **prec** $\pi_0(C, y)$
prec $nullap(C, x) \equiv$ **prec** $\pi_1(C, x)$
prec $lgap(C, x) \equiv$ **prec** $\pi_1(C, x)$
prec $eqap(C, x, y) \equiv$ **prec** $\pi_1(C, x) \wedge$ **prec** $\pi_1(C, y)$
prec $incident(C, x, y) \equiv$ **prec** $\pi_0(C, x) \wedge$ **prec** $\pi_1(C, y)$
prec $incident2(C, x, y) \equiv$ **prec** $\pi_1(C, x) \wedge$ **prec** $\pi_1(C, y)$
prec $secant(C, x, y) \equiv$ **prec** $\pi_1(C, x) \wedge$ **prec** $\pi_1(C, y)$
prec $intersection(C, x, y) \equiv$ **prec** $secant(C, x, y) \wedge secant(C, x, y)$

Fin

Le classement des arêtes autour d'un sommet est basé sur le classement des angles d'incidence de ces arêtes. L'opération $angle(C, x)$ donne l'angle, compris entre 0 et 2π, entre l'axe vertical et le segment orienté sur lequel est plongée l'arête de x. Notons que le classement des sommets est indépendant du choix de l'axe utilisé pour le calcul de l'angle.

Comme nous l'avons vu précédemment, pour tester si un sommet est bien classé, la méthode la plus simple est de vérifier que la suite des angles des arêtes du sommet, débutant par l'arête d'angle minimal, est croissante. Nous utilisons donc une fonction auxiliaire $angle_{min}(C, x)$ qui renvoie le brin du sommet de x dont l'arête est d'angle minimal. La fonction $class$ réalise alors simplement un parcours du sommet à partir du brin obtenu par $angle_{min}$, pour tester si la suite des angles est croissante. Les préconditions de $class$ et de $fusion$ sont définies par la fonction auxiliaire $lp0\_s(C, x)$ qui teste l'absence de plongement sur le sommet de x et sur les sommets adjacents.

La fonction $fusion(C, x, y)$ utilise la fonction $fusion'$ pour interclasser les brins par angles croissants. Cette dernière démarre avec les 2 brins d'angles minimaux des sommets de x et y, puis parcourt ces 2 sommets en réorganisant, au fur et à mesure, les liaisons par α_1 entre leurs brins. La fonction auxiliaire $l1sp(C, x, y)$, utilisée par *fusion-aux* place un 1-lien entre deux brins plongés sur des points égaux, en veillant à ce que l'un des plongements soit supprimé pour satisfaire les préconditions de $l1$.



(a) x et y n'ont pas de 1-lien

(c1) il y a un 1-lien direct depuis x

(b) x n'a pas de 1-lien, il y a un 1-lien direct depuis y

(c2) il y a un 1-lien direct depuis x

FIG. 3.7: *Etude de cas pour la fusion de deux sommets*

La figure 3.7 schématise les différents cas de la définition de la fonction $fusion'(C, x, y)$. Notons que dans cette figure les points p et q sont égaux, mais sont représentés légérement écartés l'un de l'autre pour plus de clarté.

Dans cette figure, les segments et flèches pleines représentent les brins manipulés et les 1-liaisons existantes. Les brins x et y sont les brins courant de la fusion et le brin noté x_1 est l'image de x par α_1 lorsqu'elle est requise. Le cas (a), pour lequel x et y ne possèdent aucune 1-liaison, correspond au cas trivial ou à la condition d'arrêt de la fusion. Le cas (b), pour lequel x n'a pas de 1-liens, correspond à l'arrêt du parcours du sommet de x, la fusion

TAB. 3.7: *Spécifications des opérations géométriques pour les sommets*

Spéc *2-CARTE-GEO3* **étend** *2-CARTE-GEO2* **avec**

Opérateurs

$angle : 2Carte\ Brin \longrightarrow Real$	(*angle d'une arête plongée*)
$class : 2Carte\ Brin \longrightarrow Bool$	(*test du classement d'un sommet*)
$fusion : 2Carte\ Brin\ Brin \longrightarrow 2Carte$	(*fusion de deux sommets*)

Axiomes $(C : 2Carte\ ;\ x, y : Brin\ ;\ p : Point)$

$angle(C, x) =$ **si** $nullap(C, x)$ **alors** 0 **sinon** $angle(\pi_1(C, x))$

$angle_{min}(C, x) = angle'_{min}(C, x, x)$
$angle'_{min}(C, x_0, x) =$ **si** $(\alpha_{\text{-}1}(C, x) = x_0) \vee (angle(C, \alpha_{\text{-}1}(C, x)) > angle(C, x))$ **alors** x
sinon $angle'_{min}(C, x_0, \alpha_{\text{-}1}(C, x))$

$class(C, x) = class'(C, angle_{min}(C, x), angle_{min}(C, x))$
$class'(C, x_0, x) =$ **si** $(\alpha_1(C, x) = x_0)$ **alors** $vrai$
sinon $(angle(C, x) \leq angle(C, \alpha_1(C, x))) \wedge class'(C, x_0, \alpha_1(C, x))$

$l1sp(C, x, y) = l1(dp0(C, y), x, y)$

$fusion(C, x, y) = fusion'(C, angle_{min}(C, x), angle_{min}(C, y))$

$fusion'(C, x, y) =$ **si** $angle(C, x) > angle(C, y)$
alors $fusion'(C, y, x)$
sinon si $al1(C, x)$
alors si $al1(C, y)$
alors $l1sp(C, x, y)$
sinon $l1sp(fusion'(dsp(C, y), y, \alpha_1(C, y)), x, y)$
sinon si $angle(C, \alpha_1(C, x)) > angle(C, y)$
alors $l1sp(fusion'(dsp(C, x), y, \alpha_1(C, x)), x, y)$
sinon $l1sp(fusion'(dsp(C, x), \alpha_1(C, x), y), x, \alpha_1(C, x))$

$lpo\_s(C, x) = lp0(C, x) \wedge lpo'_s(C, x, x)$
$lpo\_s'(C, x_0, x) =$ **si** $(\alpha_1(C, x) = x_0)$ **alors** $lp0(C, \alpha_0(C, x))$
sinon $lp0(C, \alpha_0(C, x)) \wedge lpo'_s(C, x_0, \alpha_1(C, x))$

Préconditions

prec $angle(C, x) \equiv$ **prec** $\pi_1(C, x)$
prec $class(C, x) \equiv e(C, x) \wedge \neg lpo_s(C, x)$
prec $l1sp(C, x, y) \equiv$ **prec** $eqsp(C, x, y) \wedge eqsp(C, x, y) \wedge \neg eqs(C, x, y)$
prec $fusion(C, x, y) \equiv$ **prec** $l1sp(C, x, y) \wedge \neg lpo_s(C, x) \wedge \neg lpo_s(C, x) \wedge class(C, x) \wedge class(C, y)$

Fin

---

continuant alors avec les brins du sommet de y. Les deux cas (c) sont les cas généraux. Pour (c1), l'angle de x_1 est plus grand que l'angle de y et pour le cas (c2) c'est la situation inverse. Pour ces deux cas la fusion continue avec les brins x' et y'.

Les flèches blanches et les segments en pointillés représentent les brins et 1-liaisons qui peuvent éventuellement exister et illustrent l'avancement de la fusion. La fonction $fusion'$ est récursive. Pour schématiser cet aspect, les brins sur lesquels la récursion s'effectue sont notés x' et y'. Après l'appel récursif, une liaison est placée entre les deux premiers brins du futur sommet. Elle correspond à l'utilisation de la fonction $l1sp$ dans les spécifications. Les flèches noires avec des tirets représentent ces 1-liaisons ajoutées *après* l'appel récursif.

3.3 Opérations de manipulation des labels

Dans cette section nous décrivons les opérations liées à la manipulation des labels dans une carte plongée. Pour simplifier, nous utiliserons des entiers comme identificateurs d'objets. Un label est ainsi un ensemble d'entiers. Nous ne détaillons pas les opérations sur les ensembles dont les spécifications classiques peuvent être trouvées dans [42]. Rappelons que pour un brin, son i-label est l'ensemble des objets identifiés auxquels appartient sa cellule de dimension i, avec $i \in \{0, 1, 2\}$. Donc le 0-label correspond au sommet, le 1-label à l'arête et le 2-label à la face du brin.

La table 3.8 décrit les générateurs de label $gl0, gl1$ et $gl2$, pour chaque dimension. Nous n'avons pas détaillé les axiomes de permutations entre ces nouveaux générateurs de base et les générateurs de base précédemment définis. Un ensemble d'axiomes implicites décrit le comportement des opérateurs précédemment définis avec les générateurs de plongement, ainsi que le comportement des opérations sur les labels décrites dans cette section vis-à-vis des générateurs des cartes.

La table 3.8 spécifie un ensemble de sélecteurs, $label0, label1$ et $label2$, permettant d'obtenir le label d'un brin à chaque dimension. Elle définit également un ensemble de fonctions, $sl0, sl1$ et $sl2$, servant à supprimer le label d'un brin. Leurs axiomes ne sont pas donnés car ils sont tout à fait semblables aux sélecteurs correspondants.

La définition des sélecteurs et leurs préconditions imposent qu'il n'y ait qu'un 0-label par sommet et qu'un 1-label par arête. Par contre, chaque brin d'une face possède son propre 2-label. Ceci est nécessaire pour l'évaluation d'opérations booléennes par complétion des labels et sera justifié par la suite.

Pour assurer la préservation des labels pendant le raffinement géométrique, comme cela sera expliqué plus loin, nous aurons besoin d'opérations géométriques prenant en compte les labels. Ces opérations doivent, d'une part, ne pas créer de vide ou de trous. Ainsi, lors des coupures d'arêtes, les nouveaux brins qui sont insérés dans la carte doivent hériter de labels cohérents avec les labels des brins auxquels ils sont reliés. Pour cela nous définissons une opération de coupure d'arêtes labellées $capl(C, x)$ qui coupe l'arête de x et recopie les labels de x et $\alpha_0(C, x)$ sur les nouveaux brins.

Par ailleurs, nous aurons besoin d'opérations permettant d'éviter la perte d'information due à la suppression d'arête. Nous définissons une opération de fusion de deux arêtes labellées $fapl(C, x, z)$ qui détruit l'arête de z après avoir ajouté ses labels à ceux de l'arête de x.

Enfin, comme il ne doit y avoir qu'un 0-label par sommet, nous définissons une opération

TAB. 3.8: *Spécification des opérations de base pour les labels*

Spéc *2-CARTE-LABELLEE* **étend** *2-CARTE-GEO3* **avec**

Sortes $Label = Set[Int]$ (*les labels sont des ensembles d'entiers*)

Opérateurs

$gl0 : 2Carte\ Brin\ Label \longrightarrow 2Carte$	(*générateur de 0-label*)
$gl1 : 2Carte\ Brin\ Label \longrightarrow 2Carte$	(*générateur de 1-label*)
$gl2 : 2Carte\ Brin\ Label \longrightarrow 2Carte$	(*générateur de 2-label*)
$sl0 : 2Carte\ Brin\ Label \longrightarrow 2Carte$	(*suppression d'un 0-label*)
$sl1 : 2Carte\ Brin\ Label \longrightarrow 2Carte$	(*suppression d'un 1-label*)
$sl2 : 2Carte\ Brin\ Label \longrightarrow 2Carte$	(*suppression d'un 2-label*)
$label0 : 2Carte\ Brin \longrightarrow Label$	(*0-label d'un brin*)
$label1 : 2Carte\ Brin \longrightarrow Label$	(*1-label d'un brin*)
$label2 : 2Carte\ Brin \longrightarrow Label$	(*2-label d'un brin*)

Axiomes $(C : 2Carte\ ;\ x, y : Brin\ ;\ L : Label)$

/ axiomes de permutations */*

$label0(v, x) = \emptyset$

$label0(gl0(C, x, L), y) =$ **si** $eqv(x, y)$ **alors** L **sinon** $label0(C, y)$

$label1(v, x) = \emptyset$

$label1(gl1(C, x, L), y) =$ **si** $y == x \vee y = \alpha_0(C, x)$ **alors** L **sinon** $label1(C, y)$

$label2(v, x) = \emptyset$

$label2(gl2(C, x, L), y) =$ **si** $y == x$ **alors** L **sinon** $label2(C, y)$

Préconditions

prec $gl0(C, x, L) \equiv label0(C, x) = \emptyset \wedge L \neq \emptyset$

prec $gl1(C, x, L) \equiv label1(C, x) = \emptyset \wedge L \neq \emptyset$

prec $gl2(C, x, L) \equiv label2(C, x) = \emptyset \wedge L \neq \emptyset$

Fin

TAB. 3.9: *Spécification des opérations géométriques avec labels*

Spéc *2-CARTE-GEOLABEL* **étend** *2-CARTE-LABELLEE* **avec**

Opérateurs

$capl : 2Carte\ Brin\ Point \longrightarrow 2Carte$	(*coupure d'arête labellée*)
$fapl : 2Carte\ Brin\ Brin \longrightarrow 2Carte$	(*fusion d'arêtes labellées*)
$fusionl : 2Carte\ Brin\ Brin \longrightarrow 2Carte$	(*fusion de sommets labellées*)

Axiomes $(C : 2Carte\ ;\ x, y : Brin)$

$fusionl(C, x, y) = gl0(fusion(sl0(sl0(C, x), y), x, y), x, label0(C, x) \cup label0(C, y))$

Préconditions

Fin

de fusion de sommets labellés $fusionl(C, x, y)$ qui fusionne les sommets des brins x et y après avoir regroupé leurs 0-labels. L'ensemble de ces opérations est spécifié dans la spécification 2-CARTE-GEOLABEL, détaillée dans la table 3.9. Seul l'axiome de $fusionl$ est donné. Les axiomes des deux autres opérations sont semblables et sans intérêt. Ces trois opérations suppriment les labels des brins concernés, réalisent l'opération géométrique associée, puis réinsèrent des labels qui sont soit la copie des anciens pour *capl*, soit l'union des anciens en cas de fusion.

3.4 Spécification des 3-cartes

Les 3-cartes sont spécifiées exactement comme les 2-cartes. Le seul changement correspond à l'ajout d'un générateur $l2(C, x, y)$ réalisant une 2-liaison entre les brins x et y.

TAB. 3.10: *Générateurs des 3-cartes*

Spéc *3-CARTE-GEN* **étend** *BOOL, INT* **avec**

Sortes $Brin = Int, 3Carte$	(*renomme la sorte Int en Brin*)
Opérateurs	
$v : \longrightarrow 3Carte$	(*carte vide*)
$l0 : 3Carte\ Brin\ Brin \longrightarrow 3Carte$	(*0-liaison*)
$l1 : 3Carte\ Brin\ Brin \longrightarrow 3Carte$	(*1-liaison*)
$l2 : 3Carte\ Brin\ Brin \longrightarrow 3Carte$	(*2-liaison*)
Axiomes $(C : 3Carte\ ;\ x, y, z, t : Brin)$	
axiomes de permutation	

Fin

Nous ne détaillons pas les spécifications des opérateurs topologiques et géométriques pour les 3-cartes. Ils sont définis de la même façon qu'en dimension 2. La seule différence se situe au niveau des définitions de α_1 et α_2. Pour les 3-cartes, α_1 est une involution et donc définie comme l'est α_0 pour les 2-cartes, alors que α_2 est une involution qui est définie comme l'est α_1 pour les 2-cartes.

Nous présentons, par contre, les seuls opérateurs topologiques différents et qui seront utilisés lors du raffinement 3D. Ces opérateurs sont schématisés dans la figure 3.8.

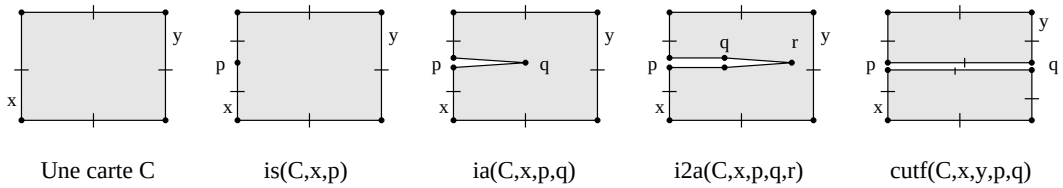


FIG. 3.8: *Exemple d'opérateurs topologiques sur les 3-cartes*

L'opération $is(C, x, p)$ insère un sommet plongé sur p dans l'arête de x. L'opération $ia(C, x, p, q)$ insère une arête double plongée sur le segment $[p, q]$ et telle que le sommet plongé sur p soit inséré dans l'arête de x. L'opération $i2a(C, x, p, q, r)$ est un raccourci pour

insérer deux arêtes doubles, plongées sur $[p, q]$ et $[q, r]$. Elle est utilisée pour insérer l'arête $[q, r]$ à l'intérieur de la face de x, en la rattachant à son bord au point p. La dernière opération, $cutf(C, x, y, p, q)$ coupe localement une face en deux morceaux séparés par une arête double plongée sur $[p, q]$, entre les brins x et y.

Enfin, dans le raffinement 3D, nous utilisons une opération de fusion d'arêtes qui réorganise les 2-liaisons autour de deux arêtes. Cette opération est définie exactement comme la fusion de sommets en dimension 2. La seule différence se situe au niveau du calcul de l'angle permettant l'interclassement. En dimension 3, l'angle utilisé est l'angle que forme le plan de la face d'un brin autour de l'axe formé par son arête.

Chapitre 4

Le raffinement des cartes en dimension 2

L'auto-raffinement d'une carte combinatoire plongée et labellée revient à lui appliquer un certain nombre de transformations, mettant en jeu les opérations géométriques décrites précédemment, jusqu'à ce qu'elle remplisse les conditions de planarité et de bonne labellisation définies dans la proposition 2.2.11 et la définition 2.3.2. Ce raffinement se divise en deux étapes. La première transforme la carte jusqu'à ce qu'elle soit bien plongée. La seconde réalise la complétion des labels. Ces deux étapes sont séparées car la complétion des labels repose sur la comparaison des labels des différentes cellules de la carte et nécessite donc que celles-ci soient complètement calculées.

Tous les algorithmes permettant d'évaluer un tel raffinement doivent appliquer le même type d'opérations. Les seules différences existant réellement entre ces algorithmes se situent dans les stratégies choisies pour l'application de ces modifications. Ainsi, un bon moyen pour décrire formellement ces transformations, en se libérant des problèmes liés au choix d'une stratégie, consiste à utiliser les techniques de réécriture. Chaque transformation élémentaire, et les conditions sous lesquelles elle peut être appliquée, peut être décrite par une *règle de réécriture conditionnelle*. L'ensemble de ces règles constitue un *système de réécriture conditionnel modulo* [10, 26].

Nous utilisons cette méthodologie car elle a l'avantage de permettre une décomposition du raffinement en un ensemble de transformations élémentaires et indépendantes qui est une expression précise et concise du raffinement. De plus, en rejetant à un autre niveau le choix d'une stratégie d'application des transformations, cette méthode conduit à une définition simple et lisible d'une opération a priori complexe. Enfin, cette approche permet, comme nous le verrons dans la seconde section, la démonstration d'un certain nombre de propriétés logiques des transformations décrites et, en particulier, la *terminaison* et la *confluence*.

Le choix de la réécriture pour décrire le raffinement, à la place des spécifications algébriques que nous avons utilisées jusqu'ici, se fonde également sur le fait que, à notre sens, la réécriture permet de rendre plus naturellement la notion de transformations successives. De plus, la forme même des règles de réécriture nous permettra, dans le chapitre suivant, d'étudier diverses stratégies pour l'application des ces règles.

4.1 Le système de réécriture

Dans cette première section, nous décrivons le système de réécriture définissant le raffinement d'une carte combinatoire plongée. Pour simplifier cette première description, nous ferons abstraction des labels de la carte. Les problèmes liés à la manipulation des labels et à leur complétion seront abordés dans une section suivante.

Dans la suite, une règle R du système de réécriture sera écrite suivant la syntaxe suivante :

$$R : \frac{C}{C'} \textbf{ si } \begin{cases} condition_1 \\ \dots \\ condition_n \end{cases}$$

où au numérateur figure la carte initiale C et au dénominateur la carte C' après transformation. Les conditions 1 à n situées après le **si**, lorsqu'elles sont évaluées à *vrai*, permettent le déclenchement de cette règle. Notons que l'ordre d'évaluation des conditions n'a ici aucune importance. Les règles de réécritures sont illustrées graphiquement dans la figure 4.1 et des explications intuitives sont données ci-après. Nous verrons plus loin une description formelle du système de réécriture.

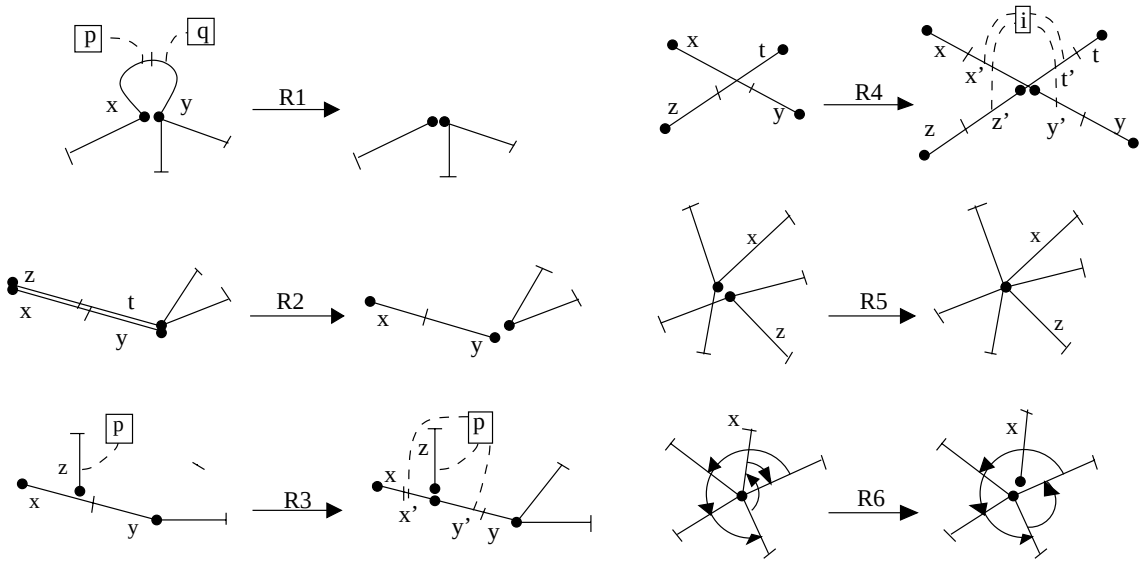


FIG. 4.1: *Illustration graphique des règles de réécriture*

Les règles de réécriture décrivent les transformations à effectuer sur une carte pour qu'elle vérifie les conditions décrites dans la proposition 2.2.11. Ces transformations découlent simplement de ces conditions. En fait, pour chacune d'elles, on définit une règle qui modifie la carte si elle contient un brin ou un couple de brins ne la vérifiant pas.

Ainsi, la première règle R_1 supprime, si elle existe, une arête nulle, représentée par une boucle, plus visible, dans la figure 4.1. Lorsque deux arêtes sont superposées, la règle R_2 supprime l'une des deux. La règle R_3 réalise la coupure par incidence. Lorsqu'un sommet est incident à une arête, celle-ci est coupée en deux. La règle R_4 est la coupure par intersection. Lorsqu'il existe une intersection franche entre deux arêtes, celles-ci sont coupées en leur point d'intersection.

Les deux dernières règles traitent les sommets. La règle R_5 fusionne deux sommets distincts ayant le même 0-plongement. La dernière règle R_6 éclate les sommets non classés. Lorsqu'un brin appartient à un sommet non classé, il est détaché de ce sommet. Cette règle est appliquée jusqu'à ce que le sommet restant soit bien classé ou complètement éclaté. Le sommet sera alors correctement reconstruit par R_5. Dans la figure 4.1, et uniquement pour R_6, les liaisons par α_1 sont représentées par des arcs de cercles.

TAB. 4.1: *Système de réécriture pour le raffinement des cartes plongées*

$$R_1 : \frac{gp0(C,x,p)}{dap(gp0(C,x,p),x)} \textbf{ si } nullap(gp0(C,x,p),x)$$

$$R_2 : \frac{gp0(gp0(C,x,p),z,q)}{dap(gp0(gp0(C,x,p),z,q),z)} \textbf{ si } eqap(gp0(gp0(C,x,p),z,q),x,z)$$

$$R_3 : \frac{gp0(gp0(C,x,p),z,q)}{cap(gp0(gp0(C,x,p),z,q),x,q)} \textbf{ si } \begin{cases} \neg nullap(gp0(gp0(C,x,p),z,q),z) \\ incident(gp0(gp0(C,x,p),z,q),z,x) \end{cases}$$

$$R_4 : \frac{gp0(gp0(C,x,p),z,q)}{cap(cap(gp0(gp0(C,x,p),z,q),x,i),z,i)} \textbf{ si } secant(gp0(gp0(C,x,p),z,q),x,z)$$
$$\textbf{avec } i = intersection(gp0(gp0(C,x,p),z,q),x,z)$$

$$R_5 : \frac{gp0(gp0(C,x,p),z,q)}{fusion(gp0(gp0(C,x,p),z,q),x,z)} \textbf{ si } \begin{cases} \neg eqv(C,x,z) \\ eqp(p,q) \end{cases}$$

$$R_6 : \frac{gp0(C,x,p)}{dsp(gp0(C,x,p),x)} \textbf{ si } \neg class(gp0(C,x,p),x)$$

La table 4.1 montre la définition formelle du système de réécriture décrit ci-dessus. La réécriture est effectuée modulo les équations de la spécification des cartes. Précisément, lorsque l'on cherche à appliquer une règle à une carte C donnée, contenant un sous-terme u, on cherche à unifier ce sous-terme u avec le numérateur de l'une des règles.

Pour cela, on utilise les équations des spécifications, et notamment les équations de permutation, pour remplacer u par un terme lui étant égal et pouvant s'unifier syntaxiquement avec un numérateur. Cette étape correspond, pour l'essentiel, à une commutation des générateurs de base $l0, l1$ et $gp0$ dans u, jusqu'à ce que les générateurs en tête du terme correspondent au motif décrit par un numérateur. Puis u est remplacé dans la carte C par le dénominateur de la règle utilisée. On dit, dans ce cas, que la réécriture est effectuée *modulo les équations* de la spécification.

Exemple 4.1.1 *Dans la règle R_1 de la table 4.1, le numérateur, $N = gp0(C,x,p)$, correspond à une carte N dans laquelle un brin x est plongé. La condition $nullap(gp0(C,x,p),x)$ précise que R_1 peut être déclenchée si l'arête de x est nulle. Le dénominateur $D = dap(gp0(C,x,p),x)$ décrit l'action à effectuer sur N, en l'occurrence la suppression de cette arête. La règle R_1 peut donc se lire de la façon suivante : s'il existe dans N un brin x dont l'arête est nulle, cette dernière est détruite.* □

Les transformations induites par le déclenchement d'une règle de réécriture ne doivent pas pouvoir engendrer des erreurs de précondition dans la carte à raffiner. Pour cela, il faut ajouter des contraintes à la méthode de réécriture que nous avons esquissée ci-dessus. Une règle ne peut être utilisée que si les préconditions des opérations apparaissant dans ses conditions de déclenchement sont remplies et si ces mêmes conditions sont évaluées à *vrai*.

De plus, il faut que le remplacement du sous-terme transformé engendre une carte correcte, c'est-à-dire dont les générateurs vérifient toutes leurs préconditions. Intuitivement, cela signifie que l'on ne peut réécrire un sous-terme d'une carte que s'il contient toutes les informations, les liens et les plongements, concernant les brins sur lesquels portent les transformations.

Pour une définition formelle de la réécriture, nous notons R l'ensemble des règles du système de réécriture, E l'ensemble des équations des spécifications algébriques des 2-cartes et $s \overset{*}{\underset{E}{=}} t$ le fait que deux termes s et t sont égaux modulo un nombre quelconque d'équations de E. Nous notons, de façon plus condensée, $C \xrightarrow[R/E]{} C'$ le fait que la carte C se réécrit en la carte C' en utilisant une des règles de R, modulo les équations de E. Et pour une règle particulière, nous écrivons $c_1 \wedge \ldots \wedge c_m \mid n \rightarrow d$ où n est numérateur de cette règle, d son dénominateur et les c_i ses conditions de déclenchement. La définition formelle de la relation $\xrightarrow[R/E]{}$ est alors la suivante :

Définition 4.1.1 *Soient une carte C représentée par le terme s et une carte C' représentée par le terme t, alors $C \xrightarrow[R/E]{} C'$ si et seulement si :*

- *u est un sous-terme de s à la position p, $u = s\,|_p$;*
- *il existe une règle $c_1 \wedge \ldots \wedge c_m \mid n \rightarrow d$ et une substitution σ tel que u soit égal modulo E à n dont les variables sont associées à celles de u par σ, $u \overset{*}{\underset{E}{=}} n\sigma$;*
- *les conditions de déclenchement c_i et leurs préconditions sont évaluées à vrai ;*
- *t est égal à s dans lequel on a remplacé u par d dont les variables sont substituées de la même façon, $t = s[d\sigma]_p$;*
- *t est un terme correct après ce remplacement.*

Dans la table 4.1, on peut remarquer que les numérateurs sont toujours décrits avec le générateur $gp0$. Ceci est dû au fait que les conditions font toutes appel à des tests géométriques et nécessitent donc des brins plongés. De plus, il faut que les permutations utilisées durant la phase d'unification n'entraînent pas d'erreurs de précondition. Or $gp0$ a la propriété de toujours pouvoir permuter vers l'extérieur du terme. Ainsi son utilisation assure que les autres générateurs, le $l0$ ayant réalisé l'insertion des brins concernés et les $l1$ définissant leurs 1-liaisons, peuvent toujours se trouver à l'intérieur du terme décrivant un numérateur.

La phase de recherche pour unifier s au numérateur d'une règle peut être vu comme un test existentiel, puisque le plongement d'un brin assure son existence. C'est pourquoi, pour condenser l'écriture des règles et pour permettre une lecture plus aisée, nous noterons $x \in C$ dans les conditions d'une règle pour signifier que x est un brin qui apparaît dans l'un des générateurs, et en particulier $gp0$, de la carte C. Ainsi la notation $x \in C$ est un raccourci pour $C = gp0(C', x, p)$ et $x \in C \wedge z \in C$ pour $C = gp0(gp0(C', x, p), z, q)$ modulo les permutations. Notons que cette notation peut donner l'impression que le système simplifié de la table 4.2 n'est pas un véritable système de réécriture au sens de [26], puisque à première vue des variables n'appartenant pas au numérateur sont introduites dans le reste des règles. Ce n'est bien sûr pas le cas.

TAB. 4.2: *Système de réécriture simplifié avec tests existentiels*

$R_1 : \dfrac{C}{dap(C,x)}$ **si** $\begin{cases} x \in C \\ nullap(C,x) \end{cases}$	$R_4 : \dfrac{C}{cap(cap(C,x,i),z,i)}$ **si** $\begin{cases} x \in C \wedge z \in C \\ secant(C,x,z) \end{cases}$ **avec** $i = intersection(C,x,z)$
$R_2 : \dfrac{C}{dap(C,z)}$ **si** $\begin{cases} x \in C \wedge z \in C \\ eqap(C,x,z) \end{cases}$	$R_5 : \dfrac{C}{fusion(C,x,y)}$ **si** $\begin{cases} x \in C \wedge z \in C \\ \neg eqv(C,x,y) \\ eqsp(C,x,y) \end{cases}$
$R_3 : \dfrac{C}{cap(C,x,p)}$ **si** $\begin{cases} x \in C \wedge z \in C \\ \neg nullap(C,z) \\ incident(C,z,x) \end{cases}$	$R_6 : \dfrac{C}{dsp(C,x)}$ **si** $\begin{cases} x \in C \\ \neg class(C,x) \end{cases}$

Le système de réécriture de la table 4.2 décrit complètement et clairement le raffinement d'une carte. Le résultat, lorsque plus aucune règle ne peut être déclenchée, vérifie bien les conditions de planarité décrites dans la proposition 2.2.11. En effet, si l'une des conditions n'était pas remplie, il est aisé de voir que l'une au moins des règles pourrait être déclenchée. Il nous reste ainsi à vérifier, d'une part, que le raffinement est calculable, c'est à dire que le processus de transformation termine, et d'autre part, que le résultat est unique et donc ne dépend pas de l'ordre de déclenchement des règles. Ces démonstrations, données dans les deux sections qui suivent, sont rendues possibles par le formalisme utilisé.

4.2 Terminaison

4.2.1 Généralités

La première qualité que l'on attend d'un système de réécriture, ainsi que de tout calcul informatique traditionnel, est sa terminaison [53]. On dit qu'un système est à *terminaison finie* lorsqu'il n'existe pas de suite infinie d'applications de règles et cela quelles que soient les données initiales. Ainsi, dans notre cas, partant d'une carte donnée et appliquant les règles du système, on arrive, au terme d'un nombre fini de réécritures modulo, à une carte ne pouvant déclencher aucune des règles. Cette carte est une *forme normale* de la carte initiale par rapport au système de réécriture.

La méthode standard pour montrer la terminaison d'un système de réécriture, est la construction d'un *ordre de réduction* sur les termes, c'est-à-dire bien fondé et monotone, tel que l'application d'une règle transforme un terme donné en un terme plus petit pour cet ordre. Les ordres classiques utilisent la syntaxe des termes et un pré-ordre, appelé une *précédence*, sur les symboles de fonctions [53, 10, 25].

Or, dans notre cas la construction d'un tel ordre est difficile car, syntaxiquement, la taille d'une carte augmente lors de son raffinement, des brins et des liaisons pouvant y être ajoutés. Bien sûr, il est possible de construire des ordres de réduction basés sur la syntaxe même dans ce cas. Mais ici, il est clair que le système termine, en fait, parce que le nombre d'intersections ou de sommets superposés dans une carte diminue lors de son raffinement, ce qui est une mesure *sémantique* et non pas syntaxique de l'avancement du processus.

Pour résoudre ce problème délicat, nous nous appuyons sur la technique de [15] qui repose sur l'utilisation d'un *ordre sémantique* $>_{sem}$, plus riche qu'un ordre reposant purement sur la syntaxe, car faisant intervenir le sens que l'on donne aux règles de réécriture. Cet ordre, également basé sur une précédence, fait intervenir, de surcroît, une *mesure* des cartes, notée m. Lorsque deux termes ne peuvent être comparés syntaxiquement en utilisant la précédence, leurs mesures respectives sont comparées.

La mesure m doit faire apparaître l'état de raffinement d'une carte. Pour cela, notons que les règles de raffinement se divisent naturellement en deux groupes. Le premier, formé des règles R_1 à R_4, gère les arêtes. Le second, formé par R_5 et R_6, rectifie les sommets. Nous considérons alors la mesure comme un couple $(m0, m1)$ dont les composantes mesurent l'avancement des transformations réalisées par ces deux groupes.

4.2.2 La composante m_0

La composante m_0 doit décroître strictement en cas de coupure ou de suppression d'arêtes. La première idée serait de considérer comme mesure la somme des carrés des longueurs des arêtes. En effet, lorsque l'on coupe une arête de longueur $l = a+b$ en deux arêtes de longueurs a et b non nulles, on a bien $(a+b)^2 > a^2 + b^2$. De même, la suppression d'une arête non nulle soustrait un réel strictement positif de cette somme. Pourtant, cette mesure ne convient pas car elle ne construit pas un ordre *bien fondé*. Il existe, en effet, des suites de réels positifs décroissantes et infinies.

Cette première idée nous guide cependant vers une solution plus subtile, basée sur le même principe, mais utilisant des entiers. Nous remplaçons la longueur l d'une arête non nulle par l'entier $\lceil l \rceil$ qui lui est immédiatement supérieur ou égal. Une arête nulle étant toujours comptée 1, on a $\lceil l \rceil \geqslant 1$. Montons que pour a et b réels, $\lceil a+b \rceil^2 > \lceil a \rceil^2 + \lceil b \rceil^2$ si et seulement si $\lceil a \rceil \geqslant 2$ et $\lceil b \rceil \geqslant 2$, en utilisant la proposition suivante.

Proposition 4.2.1 $\lceil a+b \rceil = \lceil a \rceil + \lceil b \rceil$ *ou* $\lceil a+b \rceil = \lceil a \rceil + \lceil b \rceil - 1$.

Preuve 4.2.1 *Cette première proposition se montre aisément en utilisant la définition de* $\lceil\ \rceil$ *:*

$$\begin{aligned} \alpha &= \lceil a \rceil - a, \quad \text{avec } \alpha \in [0,1[\\ \beta &= \lceil b \rceil - b, \quad \text{avec } \beta \in [0,1[\\ a+b &= \lceil a \rceil + \lceil b \rceil + \gamma, \quad \text{avec } \gamma = -(\alpha+\beta) \in]-2,0] \\ \lceil a+b \rceil &= \lceil a \rceil + \lceil b \rceil + \lceil \gamma \rceil, \quad \text{avec } \lceil \gamma \rceil = 0 \text{ ou } -1. \end{aligned}$$

□

Proposition 4.2.2 $\lceil a+b \rceil^2 > \lceil a \rceil^2 + \lceil b \rceil^2$ *si et seulement si* $\lceil a \rceil \geqslant 2$*, et,* $\lceil b \rceil \geqslant 2$.

Preuve 4.2.2 *En effet, dans le cas où* $\lceil a+b \rceil = \lceil a \rceil + \lceil b \rceil$*, alors le résultat est immédiat et lorsque* $\lceil a+b \rceil = \lceil a \rceil + \lceil b \rceil - 1$*, alors :*

$$\begin{aligned} \lceil a+b \rceil^2 &> \lceil a \rceil^2 + \lceil b \rceil^2 \\ \Leftrightarrow 2\lceil a \rceil\lceil b \rceil - 2\lceil a \rceil - 2\lceil b \rceil + 1 &> 0 \\ \Leftrightarrow \lceil a \rceil(\lceil b \rceil - 2)\lceil b \rceil(\lceil a \rceil - 2) + 1 &> 0 \\ \Leftrightarrow \lceil a \rceil \geqslant 2 \quad et \quad \lceil b \rceil &\geqslant 2 \end{aligned}$$

□

Il reste alors un problème pour les arêtes de longueur inférieure ou égale à 1. Pour le contourner il suffit de diviser les longueurs des arêtes par la moitié de la longueur minimale des arêtes non nulles de la carte et de compter 2 pour une arête nulle. Les mesures considérées seront alors toujours supérieures ou égales à 2. Notons que cette longueur minimale existe et est non nulle car l'ensemble des brins est fini. Formellement nous obtenons la définition suivante :

Définition 4.2.3 $m_0 = \frac{1}{2} \sum_{x \in C} \lceil 2l(x)/\varepsilon \rceil^2$, *avec :*

- $l(x) = \begin{cases} 1 \text{ si l'arête de } x \text{ est nulle ;} \\ \text{longueur de l'arête de } x \text{ sinon.} \end{cases}$
- $\varepsilon = \min_{x \in C} \{l(x)\}$.

La somme est divisée par 2, car elle est réalisée sur l'ensemble des brins de la carte et donc chaque arête est comptée deux fois.

4.2.3 La composante m_1

La composante m_1 doit rendre compte de l'état de reconstruction des sommets. La règle R_5, fusionnant deux sommets, en supprime toujours un. La règle R_6 ajoute un sommet, mais fait disparaître un brin non classé. Nous pouvons donc définir $m_1(C)$ comme étant le nombre de sommets dans la carte C, plus 2 par brin appartenant à un sommet non classé. Les règles du premier groupe pouvant faire augmenter le nombre de sommets, la composante m_0 doit avoir plus de poids que m_1. La mesure d'une carte sera donc le couple $m(C) = (m_0(C), m_1(C))$.

4.2.4 Spécification de la mesure d'une carte

La mesure m et ses deux composantes sont spécifiées dans la table 4.3 qui nécessite quelques explications. La spécification MESURE importe le module NAT qui décrit les entiers naturels et les opérations arithmétiques usuelles sur ces nombres. La composante m_0 est calculée via une fonction auxiliaire $m'_0(C, \varepsilon)$ qui fait intervenir la valeur de $\varepsilon(C)$ calculé au préalable. Cette fonction auxiliaire additionne les longueurs des arêtes complètement plongées.

Ainsi, lorsque l'on rencontre le générateur $gp0(C, x, p)$, si l'autre brin, $\alpha_0(C, x)$, de l'arête de x n'est pas plongé on ne compte rien et on supprime ce plongement. Sinon, on ajoute la longueur de l'arête de x, obtenue par la fonction $lgap(C, x)$ définie dans la table 3.6, à la mesure de la carte de laquelle on a supprimé l'arête de x, avec la fonction $dap(C, x)$ définie dans la table 3.5.

Le comportement vis-à-vis de $l0$ est implicite, et vis-à-vis de $l1$ consiste à supprimer le 1-lien qui n'intervient pas dans le calcul, en dupliquant correctement les plongements avec dsp.

La composante m_1 compte les sommets et dépend donc des 1-liaisons. Sa définition est basée sur la suppression systématique des 1-liaisons. Ainsi, si on rencontre le générateur $l1$, on supprime cette liaison, ce qui augmente de 1 le nombre de sommets, puis on totalise le nombre de sommets restant dans la carte auquel on soustrait 1. Lorsque les brins rencontrés avec le générateur $l1$ appartiennent à un sommet non classé on ajoute 2 au calcul précédent.

TAB. 4.3: *Spécification de la mesure d'une carte*

Spéc *MESURE* **étend** *2-CARTE-GEO3,NAT* **avec**

Opérateurs

$m : 2Carte \longrightarrow (Nat\ Nat)$	(*mesure d'une carte*)
$m0 : 2Carte \longrightarrow Nat$	(*première composante*)
$m1 : 2Carte \longrightarrow Nat$	(*seconde composante*)

Axiomes $(C : 2Carte\ ;\ x, y, z : Brin\ ;\ p : Point)$

$m(C) = (m_0(C), m_1(C))$

$m_0(C) = m'_0(C, \varepsilon(C))$

$m'_0(v, \varepsilon) = 0$

$m'_0(l1(C, x, y), \varepsilon) = m'_0(dsp(l1(C, x, y), x), \varepsilon)$

$m'_0(gp0(C, x, p), \varepsilon) =$ **si** $lp0(C, \alpha_0(C, x))$ **alors** $m'_0(C, \varepsilon)$
sinon $\lceil 2lgap(gp0(C, x, p), x)/\varepsilon \rceil^2 + m'_0(dap(gp0(C, x, p), x), \varepsilon)$

$\varepsilon(v) = 0$

$\varepsilon(gp0(C, x, p)) =$ **si** $lp0(C, \alpha_0(C, x))$ **alors** $\varepsilon(C)$
sinon $min(lgap(C, x), \varepsilon(dap(gp0(C, x, p), x)))$

$m_1(v) = 0$

$m_1(l0(C, x, y)) = 2 + m_1(C)$

$m_1(l1(l1(C, x, y), y, z)) =$ **si** $x = z$ **alors** $m_1(dsp(l1(l1(C, x, y), y, z), y))$-1
sinon si $class(l1(l1(C, x, y), y, z), y)$
alors $m_1(dsp(l1(l1(C, x, y), y, z), y)$-1
sinon $2 + m_1(dsp(l1(l1(C, x, y), y, z), y))$-1

$m_1(l1(C, x, y)) = m_1(C)$-1 **si** $al1(C, x) \wedge al1(C, y)$

$m_1(gp0(C, x, p)) = m_1(C)$ **si** $al1(C, x) \wedge al1(C, \alpha_0(C, x))$

Fin

Si le générateur en tête est $gp0$ on peut supprimer ce plongement à condition que les brins x et $\alpha_0(C,x)$ soient libres de 1-liens. C'est ce qui est exprimé par le mot clé **si**. Dans le cas contraire, le plongement reste nécessaire aux calculs des angles. Il faut donc appliquer des permutations jusqu'à ce que les 1-liens apparaissent en tête et soient supprimés par la méthode précédente.

Enfin, lorsque l'on arrive au générateur d'insertion $l0$, il n'y a plus de 1-liens ni de plongement et on compte les deux sommets créés par cette insertion d'arête.

4.2.5 L'ordre sémantique

L'ordre sémantique est basé sur une précédence sur les symboles de fonctions, notée $\succcurlyeq$. Une précédence est précisément un pré-ordre bien fondé, c'est-à-dire une relation binaire transitive et réflexive telle qu'il n'existe pas de suite strictement décroissante infinie. La relation d'équivalence $\approx$ associée à $\succcurlyeq$ est définie par : $f \approx g$ si et seulement si $f \succcurlyeq g$ et $g \succcurlyeq f$. L'ordre partiel, strict et bien fondé, noté $\succ$, associé à $\succcurlyeq$ est défini par $f \succ g$ si et seulement si $f \succcurlyeq g$, et non $g \succcurlyeq f$.

La précédence reflète la hiérarchie utilisée pour la définition des opérations. Ainsi, intuitivement, $f \succcurlyeq g$ signifie que f est définie en fonction de g. Deux opérations définies par une récursion croisée sont donc équivalentes pour la précédence. Pour définir $\succcurlyeq$, nous divisons les symboles de fonctions selon leurs profils, ce qui nous donne les groupes suivants :

- Pour les fonctions n'agissant pas sur les cartes :
  - G générateurs des sortes de base ($Bool$, Int, $Point$, Seg, ...) ;
  - O les autres opérations sur ces sortes (opérations géométriques et arithmétiques) ;
- Pour les fonctions agissant sur les cartes :
  - G_c les générateurs des cartes ($v, l0, l1$ et $gp0$) ;
  - S_c les sélecteurs sur les cartes qui sont les fonctions ayant un argument de sorte $2Carte$ et une valeur de retour qui n'est pas de sorte $2Carte$;
  - C_c les constructeurs et destructeurs des cartes qui sont les fonctions ayant un argument et une valeur de retour de sorte $2Carte$.

En effet, tous les opérateurs sont définis en fonctions des générateurs des sortes leur correspondant, et les fonctions sur les cartes étendent les opérations arithmétiques et géométriques. La précédence correspond à l'ordre de ces groupes. Ainsi, pour toute fonction g de G, o de O, g' de G_c, s de S_c et c de C_c, on a $c \succcurlyeq s \succcurlyeq g' \succcurlyeq o \succcurlyeq g$. Et, pour deux fonctions f et g du même groupe, on a $f \succcurlyeq g$ et $g \succcurlyeq f$, c'est-à-dire $f \approx g$.

Après ces préliminaires, nous pouvons maintenant définir l'ordre sémantique :

Définition 4.2.4 *Soient s et t deux termes avec $s = f(s_1, \ldots, s_n)$ et $t = g(t_1, \ldots, t_m)$, alors $s >_{sem} t$ si et seulement si $s >_{sem} t_i$, pour tout i, et une des quatre conditions suivantes est remplie :*

(i) $\exists i$ *tel que* $s_i \geqslant_{sem} t$

(ii) $f \succ g$

(iii) $f \approx g$ *et* NAT $\models m(s) > m(t)$

(iv) $f \approx g$ *et* NAT $\models m(s) = m(t)$ *et* $(s_1, \ldots, s_n) \gg_{sem} (t_1, \ldots, t_n)$

où $\gg_{sem}$ est l'extension lexicographique de $>_{sem}$ [25] et NAT $\models p$ signifie que l'on peut démontrer dans l'algèbre des entiers définie par la spécification NAT que p est un théorème. Si s est une constante alors $n = 0$ et, respectivement, si t est une constante alors $m = 0$. Enfin, des variables ne sont pas comparables par la précédence sauf si elles sont égales.

Les deux premières conditions apparaissant dans la définition de $>_{sem}$, représentent la partie syntaxique de cet ordre basée sur la précédence. Les deux dernières forment la partie sémantique de l'ordre. La quatrième condition ne sera d'ailleurs pas utilisée dans la suite.

Dans [15], on montre que l'ordre sémantique ainsi défini est un ordre de *réduction*. Alors, le système de réécriture R est à terminaison finie s'il est $>_{sem}$-*orientable*, c'est-à-dire si chacune de ses règles l'est. Une règle $c_1 \wedge \ldots \wedge c_m \mid n \to d$ est $>_{sem}$-orientable si l'on a $n >_{sem} vrai$, $n >_{sem} c_i$ pour tout i et $n >_{sem} d$.

Enfin, l'ordre sémantique a la propriété de sous-terme, c'est-à-dire qu'un terme est toujours plus grand qu'un de ses sous-termes, et est stable par substitution, c'est-à-dire que pour tous termes, s et t, et toute substitution σ, $s >_{sem} t \Rightarrow s\sigma >_{sem} t\sigma$.

4.2.6 Démonstration de la terminaison

Pour démontrer la terminaison du système de réécriture, nous allons vérifier que chaque règle est $>_{sem}$-orientable. La première condition, $n >_{sem} vrai$, est remplie grâce à la condition (ii), en raison de la définition de la précédence. En effet, la constante $vrai$ appartient au groupe G et dans toutes les règles le symbole de fonction apparaissant en tête du numérateur n est $gp0$ qui fait partie du groupe G_c. On a donc $gp0(C, x, p) >_{sem} vrai$.

La seconde condition à vérifier est $n >_{sem} c_i$, pour tout i. Les c_i sont tous des termes dont le symbole de tête est un sélecteur du groupe S_c. En utilisant le fait que les préconditions de ses sélecteurs doivent être évaluées à $vrai$ pour pouvoir déclencher une règle, on peut réduire les c_i à des termes dont les symboles de tête sont des opérations géométriques du groupe O. Comme dans le premier cas, n a pour symbole de tête $gp0 \in G_c$. Donc $n >_{sem} c_i$ est vérifié par la condition (ii) et grâce à la précédence. Voyons, pour nous convaincre, la démonstration détaillée de cette condition pour la règle R_1.

Preuve 4.2.3 *Montons que* $gp0(C, x, p) >_{sem} nullap(gp0(C, x, p), x)$. *Les préconditions de la fonction nullap sont* **prec** $nullap(C_0, x) \equiv e(C_0, x) \wedge \neg lp0(C_0, x) \wedge \neg lp0(C_0, \alpha_0(C_0, x))$. *Dans la carte* $gp0(C, x, p)$, *on peut monter formellement, en utilisant directement les axiomes de la spécification, que* x *existe,* $e(gp0(C, x, p), x) = vrai$, *et est plongé,* $\neg lp0(gp0(C, x, p), x) = vrai$. *Il faut donc encore que cette carte vérifie la condition* $\neg lp0(gp0(C, x, p), y) = vrai$, *en remplaçant le terme* $\alpha_0(gp0(C, x, p), x)$ *par* y *pour simplifier l'écriture.*

D'après la définition de $lp0$, *ceci est vérifié si le sommet* y *est plongé par l'utilisation du générateur* $gp0$ *sur un brin du sommet de* y. *En utilisant les axiomes de permutation, on peut amener ce générateur sur le brin* y *et en tête de terme. On peut ainsi affirmer qu'il existe un terme* C' *et un terme* q *tels que* $C = gp0(C', y, q)$. *Cette phase constitue une démonstration inductive, valide dans les algèbres engendrées correspondant à la spécification. Alors :*

$$\begin{aligned} gp0(C, x, p) &= gp0(gp0(C', y, q), x, p), \quad \text{avec } y = \alpha_0(gp0(C, x, p), x) \\ nullap(gp0(C, x, p), x) &= nullap(gp0(gp0(C', y, q), x, p), x) \end{aligned}$$

$$\begin{aligned} &= eqsp(gp0(gp0(C', y, q), x, p), x, y) \\ &= eqp(p, q) \end{aligned}$$

en utilisant les axiomes définissant nullap et eqsp, donnés dans la table 3.6.

Or $gp0(C, x, p) >_{sem} eqp(p, q)$*, puisque l'on a* $gp0(C, x, p) >_{sem} p$ *et* $gp0(C, x, p) >_{sem} q$*, p et q étant des sous-termes de* $gp0(C, x, p)$*, et* $gp0 \succ eqp$*, ce qui permet de remplir la condition (ii). Ceci conclut la démonstration de* $n >_{sem} c_1$*, pour la règle* R_1*.* □

Pour les autres sélecteurs et les autres règles, il est facile de voir qu'en appliquant le même raisonnement et après simplification on arrive au même type de résultat. Il nous reste donc à démontrer que $n >_{sem} d$ pour toutes les règles. Ceci se fait en utilisant la définition de la mesure des cartes pour vérifier la condition (iii), c'est à dire que $\mathrm{NAT} \models m(n) > m(d)$ pour chaque règle. Voyons la démonstration pour la règle R_1.

Preuve 4.2.4 *Montons que* $gp0(C, x, p) >_{sem} dap(gp0(C, x, p), x)$*. De même que précédemment, les préconditions impliquent que le brin représenté par le terme* $\alpha_0(C, x)$ *et noté y, est plongé sur un point q et donc qu'il existe un terme* C' *et un terme q tels que* $C = gp0(C', y, q)$*. La démonstration consiste alors en une étude de cas.*

Premier cas : *Supposons que les brins x et y ne sont pas 1-liés, c'est-à-dire que le terme* C' *ne contient aucune occurrence du générateur* $l1$ *faisant intervenir les termes x et y. Alors, le générateur* $l0$ *ayant réalisé l'insertion des brins x et y, et dont une occurrence existe dans* C' *d'après les préconditions, peut être amené en tête de terme, ce qui se démontre encore une fois par induction structurelle. Ainsi, après permutation, on peut affirmer qu'il existe un terme* C'' *tel que :*

$$gp0(C, x, p) = gp0(gp0(l0(C'', x, y), y, q), x, p)$$

Alors, en utilisant les axiomes définissant dap, on obtient :

$$dap(gp0(C, x, p), x) = C''$$

On a donc pour la mesure m_0 *des deux cartes :*

$$\begin{aligned} m_0(gp0(C, x, p)) &= m_0(gp0(gp0(l0(C'', x, y), y, q), x, p)) \\ &= m_0'(gp0(gp0(l0(C'', x, y), y, q), x, p), \varepsilon) \\ &= \lceil 2/\varepsilon \rceil^2 + m_0'(dap(gp0(gp0(l0(C'', x, y), y, q), x, p), x), \varepsilon) \\ &= \lceil 2/\varepsilon \rceil^2 + m_0'(C'', \varepsilon) \\ &= \lceil 2/\varepsilon \rceil^2 + m_0(C'') \\ &= \lceil 2/\varepsilon \rceil^2 + m_0(dap(gp0(C, x, p), x)) \end{aligned}$$

avec $\varepsilon = \varepsilon(gp0(gp0(l0(C'', x, y), y, q), x, p))$*. L'arête de x étant nulle, sa longueur vaut 1, d'après les axiomes de la table 3.6 définissant la fonction lgap. Il est aisé de voir que* $\lceil 2/\varepsilon \rceil^2 > 0$ *est un théorème dans l'algèbre des entiers, c'est-à-dire que* $\mathrm{NAT} \models \lceil 2/\varepsilon \rceil^2 > 0$*. Alors, la condition (iii) est vérifiée :*

$$\mathrm{NAT} \models m_0(gp0(C, x, p), \varepsilon) > m_0(dap(gp0(C, x, p), x), \varepsilon)$$

Second cas : *Supposons que x possède deux 1-liens et que y n'en ait pas. Précisément, supposons qu'il existe deux occurrences de* $l1$ *dans* C' *faisant intervenir x et deux brins* x_1

et x_{-1} tels que $x_1 = \alpha_1(C,x)$ et $x_{-1} = \alpha_{-1}(C,x)$. Le principe de la démonstration reste le même, mais les générateurs $l1$ doivent être amenés en tête de terme avec le générateur $l0$. En effet, ces trois générateurs ne peuvent commuter sans contredire les préconditions de $l1$ qui précisent qu'un brin ne peut être 1-lié que s'il a préalablement été inséré. On a alors, de manière similaire :

$$\begin{aligned} gp0(C,x,p) &= gp0(gp0(l1(l1(l0(C'',x,y),x,x_1),x_{-1},x),y,q),x,p) \\ dap(gp0(C,x,p),x) &= gp0(l1(C',x_{-1},x_1),x_1,p) \end{aligned}$$

la mesure m_0 ne dépendant pas des 1-liens, les mesures des deux termes à comparer sont les mêmes que précédemment et la condition (iii) est encore vérifiée.

Il reste alors à vérifier la condition $s >_{sem} t_i$, pour tout i, soit :

$$\begin{aligned} gp0(gp0(l1(l1(l0(C',x,y),x,x_1),x_{-1},x),y,q),x,p) &>_{sem} l1(C',x_{-1},x_1) \\ gp0(gp0(l1(l1(l0(C',x,y),x,x_1),x_{-1},x),y,q),x,p) &>_{sem} x_1 \\ gp0(gp0(l1(l1(l0(C',x,y),x,x_1),x_{-1},x),y,q),x,p) &>_{sem} p \end{aligned}$$

Les deux dernières inégalités sont immédiatement vérifiées car x_1 et p sont des sous-termes du membre de gauche. La première nécessite une étape supplémentaire. La condition (iii) est encore vérifiée car le membre de droite, $l1(C',x_{-1},x_1)$, contient un plongement de moins que pour la démonstration précédente et donc a une mesure m_0 plus petite. Il reste alors à vérifier les trois inégalités suivantes :

$$\begin{aligned} gp0(gp0(l1(l1(l0(C',x,y),x,x_1),x_{-1},x),y,q),x,p) &>_{sem} C' \\ gp0(gp0(l1(l1(l0(C',x,y),x,x_1),x_{-1},x),y,q),x,p) &>_{sem} x_{-1} \\ gp0(gp0(l1(l1(l0(C',x,y),x,x_1),x_{-1},x),y,q),x,p) &>_{sem} x_1 \end{aligned}$$

Ces inégalités sont immédiatement vérifiées car C', x_{-1} et x_1 sont des sous-termes du membre de gauche.

Les autres cas de 1-liaisons sont traités de la même manière que les seconds cas traités complètement ci-dessus. Il y a au total seize cas possibles, car pour chacun des brins x et y il existe quatre possibilités de 1-liaisons. □

Une démonstration de même nature peut facilement être effectuée sur les autres règles, en utilisant les équations définies dans les spécifications et par étude de cas sur la forme des termes représentant les cartes. La définition informelle de la mesure suffit pour se convaincre que cela est réalisable, bien que les calculs soient, il faut l'avouer, effrayants. Nous concluons donc cette section en affirmant que le système de réécriture R est à terminaison finie.

4.3 Confluence locale et convergence du système

4.3.1 Généralité

La seconde propriété attendue pour un système de réécriture est sa *confluence* [53]. Informellement, un système est *confluent* si, lorsque qu'un terme peut être réécrit de deux manières différentes menant à deux termes distincts, on peut réécrire ces deux termes en un même terme. Intuitivement cela implique que le résultat de l'application d'un nombre quelconque de règles ne dépend pas de l'ordre dans lequel ces règles ont été appliquées. Lorsqu'un système termine, sa confluence est équivalente à sa *confluence locale*. La difficulté vient ici du fait que la réécriture d'une carte se fait modulo la théorie équationnelle E définie par les spécifications.

Précisément, la relation $\xrightarrow[R/E]{}$, définie précédemment, est localement confluente si, lorsqu'une carte peut se réécrire de deux façons distinctes, $C_0 \xrightarrow[R/E]{} C_1$ et $C_0 \xrightarrow[R/E]{} C'_1$, alors il existe deux cartes C_n et C'_m obtenues par des réécritures successives, $C_1 \xrightarrow[R/E]{} \cdots \xrightarrow[R/E]{} C_n$ et $C'_1 \xrightarrow[R/E]{} \cdots \xrightarrow[R/E]{} C'_m$, telles que l'on ait $C_n \overset{*}{\underset{E}{\simeq}} C'_m$.

L'égalité entre les cartes est ici l'égalité modulo les équations de E et à un renommage des brins près, car les numéros des brins donnés par l'opération n dépendent de l'ordre de création de ces brins. Précisément, on a $C_n \overset{*}{\underset{E}{\simeq}} C'_m$ s'il existe deux cartes C et C' telles que $C_n \overset{*}{\underset{E}{=}} C$, $C'_m \overset{*}{\underset{E}{=}} C'$ et $C \sim C'$. La relation $C_n \overset{*}{\underset{E}{=}} C$ signifie qu'on peut montrer que les cartes C_n et C sont égales et ceci en utilisant un nombre quelconque d'équations de E. La relation $C \sim C'$ signifie qu'il existe un isomorphisme de cartes ϕ tel que $C = \phi(C')$, ce qui correspond intuitivement au renommage des brins. Le diagramme suivant qui généralise le *diamant* classique, résume la situation :

$$\begin{array}{ccccccccc} & & C_1 & \xrightarrow[R/E]{} & \cdots & \xrightarrow[R/E]{} & C_n & \overset{*}{\underset{E}{=}} & C \\ C_0 & \nearrow & & & & & & & \| \phi \\ & \searrow & C'_1 & \xrightarrow[R/E]{} & \cdots & \xrightarrow[R/E]{} & C'_m & \overset{*}{\underset{E}{=}} & C' \end{array}$$

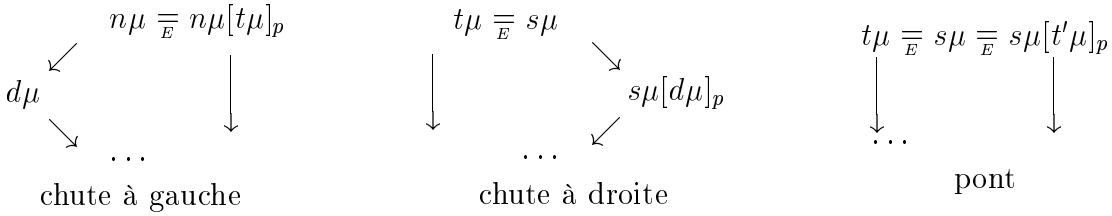
La méthode classique pour démontrer la confluence locale d'un système de réécriture est basée sur l'étude des paires critiques du système [25]. Une paire critique est formée des membres droits de deux règles dont les membres gauches se superposent et qui, intuitivement, correspondent à une ambiguïté du système. Précisément, si $l \to r$ et $s \to t$ sont deux règles du système, p la position d'un sous-terme non variable de s et μ l'unificateur le plus général de $s\ |_p$ et l, alors l'équation $t\mu = s\mu[r\mu]_p$ est une paire critique. Le terme $s\mu$ se réécrit soit $t\mu$ avec la règle $s \to t$, soit $s\mu[r\mu]_p$ en réécrivant le sous-terme à la position p avec $l \to r$. On dit que la paire critique est *joignable*, s'il existe deux suites de réécriture transformant $t\mu$ et $s\mu[r\mu]_p$ en deux termes t' et s' égaux. Le diagramme en losange suivant schématise la situation :

$$\begin{array}{ccccc} & & s\mu & & \\ & \swarrow & & \searrow & \\ t\mu & & & & s\mu[r\mu]_p \\ & \searrow^{*} & & {}^{*}\swarrow & \\ & & t' \overset{*}{\underset{E}{=}} s' & & \end{array}$$

Dans notre cas la réécriture est conditionnelle et s'effectue modulo les équations de E. La conjonction des conditions de déclenchement de deux règles superposables doit être ajoutée à la paire critique qu'elles définissent [26, 10]. De plus, les superpositions entre les équations de E et les règles de R doivent être considérées. Dans ce cas, les paires critiques sont divisées, pour plus clarté, en quatre groupes : les pics, les chutes à droite, les chutes à gauche et les ponts critiques [25].

Définition 4.3.1 *Soient $c_1 \wedge \ldots \wedge c_m \mid n \to d$ et $c'_1 \wedge \ldots \wedge c'_{m'} \mid n' \to d'$ deux règles de R, $s = t$ et $s' = t'$ deux équations de E. S'il existe une position p non variable et une substitution μ telles que :*

- *$n\mu \mid_p = n'\mu$, alors l'équation conditionnelle $c_1 \wedge \ldots \wedge c_m \wedge c'_1 \wedge \ldots \wedge c'_{m'} \mid d\mu = n\mu[d'\mu]_p$ est un pic critique de $R \cup E$;*
- *$n\mu \mid_p = s\mu$, alors l'équation conditionnelle $c_1 \wedge \ldots \wedge c_m \mid d\mu = n\mu[t\mu]_p$ est une chute à gauche critique de $R \cup E$;*
- *$s\mu \mid_p = n\mu$, alors l'équation conditionnelle $c_1 \wedge \ldots \wedge c_m \mid t\mu = s\mu[d\mu]_p$ est une chute à droite critique de $R \cup E$;*
- *$s\mu \mid_p = s'\mu$, alors l'équation $t\mu = s\mu[t'\mu]_p$ est un pont critique de $R \cup E$.*



chute à gauche chute à droite pont

Les pics correspondent aux paires critiques d'un système classique que nous avons évoqué précédemment. Les autres cas sont schématisés par les trois diagrammes ci-dessus, dans lesquels les flèches représentent l'utilisation d'une règle de R et $\underset{E}{=}$ représente l'utilisation d'une équation de E.

Le système est alors localement confluent si toutes les paires critiques sont *joignables*. Pour le démontrer, il suffit, pour chaque paire critique, de trouver les suites de réécriture permettant de fermer le diagramme, correspondant intuitivement aux pointillés des schémas. D'autres méthodes moins exhaustives existent pour des systèmes de réécriture ayant des caractéristiques syntaxiques précises. A notre connaissance, aucune de ces méthodes ne s'applique dans notre cas, essentiellement parce que les paires critiques ne peuvent être étudiées, ici, qu'à travers des considérations sémantiques.

4.3.2 Démonstration de la confluence locale

Pour nous convaincre que le système de réécriture présenté ici est bien localement confluent, nous n'étudions pas toutes les paires critiques possibles car ce serait trop long et rébarbatif. Quelques considérations informelles et un exemple de preuve typique des problèmes à résoudre, pour chaque type de paire critique, nous semblent suffisant.

Dans le cas des pics ou des chutes, une paire critique fait intervenir une, ou deux, équations de E. Les seules équations pouvant être unifiées de cette façon sont les équations de

permutations. Or les opérateurs sont définis dans les spécifications algébriques de manière à être insensibles à ces permutations. De plus, les préconditions des générateurs fournissent des contraintes fortes pour les permutations et limitent ainsi l'effet des paires critiques. Voyons dans le détail, le cas d'une chute critique à gauche, faisant intervenir une équation de permutation et la règle R_2.

Preuve 4.3.1 *L'équation* $gp0(l0(C', x', y'), z', p') = l0(gp0(C', z', p'), x', y')$, *obtenue en renommant les variables, et la règle* R_2 *que nous écrivons ici :*

$$eqap(gp0(gp0(C, x, p), z, q), x, z) \mid gp0(gp0(C, x, p), z, q) \rightarrow dap(gp0(gp0(C, x, p), z, q), z)$$

peuvent s'unifier pour former une chute critique gauche, avec la substitution :

$$\begin{aligned} C &= l0(C', x', y') \\ x &= z' \\ p &= p' \end{aligned}$$

La chute critique s'écrit précisément :

$$\begin{aligned} &eqap(gp0(gp0(l0(C', x', y'), z', p'), z, q), z', z) \mid \\ &\quad gp0(l0(gp0(C', z', p'), x', y'), z, q) = dap(gp0(gp0(l0(C', x', y'), z', p'), z, q), z) \end{aligned}$$

Les préconditions qui englobent le mécanisme de réécriture, impliquent que les brins x' *et* y' *apparaissant comme paramètres du générateur* $l0$ *que l'on permute, sont distincts de* z *et de son image par* α_0, *que nous notons* z_0 *dans la suite. En effet, la condition est* $eqap(\ldots, z', z)$ *dont les préconditions stipulent en particulier que* z *et* z_0 *existent et sont plongés. Si nous avions* $z = x'$ *ou* $z = y'$ *alors* z_0 *serait égal à* y' *ou* x'. *Le générateur* $l0$ *apparaissant dans le terme correspondrait à l'insertion de* z *et* z_0. *Ceci est impossible puisque cela impliquerait que* z_0 *ne soit pas plongé, le plongement d'un brin étant réalisé après son insertion d'après les préconditions.*

De même, les brins x' *et* y' *ne correspondent pas à des brins des sommets de* z' *ou* z_0, *puisque les générateurs* $l1$ *réalisant leurs 1-liens avec* z *ou* z_0 *devraient apparaître après leur insertion, ce qui n'est pas le cas. Ces deux arguments, peuvent être formellement justifiés en utilisant les spécifications des préconditions et des opérateurs entrant en jeu dans le raisonnement.*

La conséquence de ces remarques est que l'on peut appliquer une permutation pour faire passer $l0$ *en tête du terme de gauche et lui appliquer ensuite la règle* R_2. *De même que précédemment, les spécifications peuvent être utilisées pour vérifier que les conditions de déclenchement sont encore évaluées à vrai après ces permutations. On obtient alors une suite de réécritures pour le membre de gauche :*

$$\begin{aligned} & gp0(l0(gp0(C', z', p'), x', y'), z, q) \\ \underset{E}{=} \; & l0(gp0(gp0(C', z', p'), z, q), x', y') \\ \textit{avec la règle } R_2 \; \rightarrow \; & l0(dap(gp0(gp0(C', z', p'), z, q), z), x', y') \\ \overset{*}{\underset{E}{=}} \; & dap(gp0(gp0(l0(C', x', y'), z', p'), z, q), z) \end{aligned}$$

La dernière étape se justifie de la même façon. Comme x' et y' n'appartiennent ni à l'arête de z ni à ses sommets, les axiomes définissant dap nous assurent que les permutations sont possibles. Le dernier membre de droite est égal au terme de droite de la chute critique. Le diagramme correspondant peut donc être refermé et la chute est joignable. □

Le cas de pics critiques est plus délicat, puisque lorsque deux règles peuvent être appliquées les transformations induites modifient la géométrie de la carte et donc l'évaluation des conditions de déclenchement. Un exemple typique de ce problème intervient lorsqu'un sommet est plongé sur le point d'intersection de deux arêtes sécantes, $\{x, y\}$ et $\{x', y'\}$. Si on utilise la règle R_3 de coupure par incidence pour couper l'arête $\{x, y\}$, alors dans la carte en résultant, les arêtes de x et y n'intersectent plus l'arête $\{x', y'\}$. D'autre part, si on utilise la règle R_4 pour découper ces deux arêtes alors le sommet n'est plus incident aux arêtes en résultant. Des considérations géométriques doivent alors être utilisées pour démontrer que le pic critique correspondant est joignable.

Pour rechercher les pics critiques du système, il faut essayer d'unifier les numérateurs des différentes règles. Les termes correspondant à ces derniers ont tous en tête un ou deux générateurs $gp0$. Les unifications possibles sont donc semblables, ce qui nous permet quelques remarques communes à tous les pics critiques. Soient deux règles, notées $c \mid gp0(C, x, p) \to d$ et $c' \mid gp0(C, x', p') \to d'$, la première unification possible consiste simplement à identifier les variables des deux termes de gauche. Alors, un pic critique a la forme générale suivante $c \wedge c' \mid d = d'$.

Les pics de ce type peuvent être écartés très simplement. En effet, la conjonction $c \wedge c'$ des conditions de déclenchement est toujours évaluée à $faux$. Par exemple et intuitivement, deux arêtes sécantes ne peuvent être superposées ou nulles et deux sommets égaux ne peuvent appartenir à des arêtes sécantes. Ceci peut être démontré grâce aux spécifications de tests géométriques d'intersection, d'incidence et d'égalité ou de nullité d'arêtes.

Nous écartons donc toutes ces unifications et ne considérons ainsi que celles qui font intervenir un véritable sous-terme d'un numérateur. Seules les règles dont le numérateur comporte deux occurrences de $gp0$ peuvent être unifiées de cette façon. Les pics sont donc formés par une règle quelconque et une règle faisant intervenir un couple de brins et dont on réécrit un sous-terme strict.

Examinons en détail le pic formé, par exemple, par les règles R_4 et R_3 en notant c_4 et c_3 leurs conditions de déclenchement, pour ne pas alourdir leur expression déjà complexe, et en renommant leurs variables pour plus de clarté :

$$\begin{array}{rcl} c_4(x,z) \mid gp0(gp0(C,x,p_x),z,p_z) & \xrightarrow[R_4]{} & cap(cap(gp0(gp0(C,x,p_x),z,p_z),x,i),z,i) \\ c_3(t,u) \mid gp0(gp0(C',t,p_t),u,p_u) & \xrightarrow[R_3]{} & cap(gp0(gp0(C',t,p_t),u,p_u),t,p_u) \end{array}$$

Preuve 4.3.2 *Le pic critique est formé avec la substitution suivante :*

$$\begin{array}{rcl} C' & = & gp0(C,x,p_x) \\ t & = & z \\ p_t & = & p_z \end{array}$$

Donc le terme $gp0(gp0(gp0(C,x,p_x),z,p_z),u,p_u)$ vérifie les conditions $c_4(x,z) \wedge c_3(z,u)$. Intuitivement cela signifie que x et z appartiennent à des arêtes sécantes, notées $\{x, x'\}$ et

$\{z, z'\}$, et que le sommet de u est incident à l'arête de z. Ce terme peut se réécrire de deux façons distinctes qui conduisent au pic suivant :

$$c_4 \wedge c_3 \mid cap(gp0(gp0(gp0(C, x, p_x), z, p_z), u, p_u), z, p_u) = \\ gp0(cap(cap(gp0(gp0(C, x, p_x), z, p_z), x, i), z, i), u, p_u)$$

Le membre de gauche est obtenu en appliquant R_3 et donc en coupant, au point p_u, l'arête $\{z, z'\}$ en deux arêtes $\{z, z_3\}$ et $\{z'_3, z'\}$, où z_3 et z'_3 sont deux nouveaux brins de la carte. Le membre de droite est obtenu en appliquant R_4 et donc en coupant, en leur point d'intersection i, les arêtes $\{x, x'\}$ et $\{z, z'\}$. Les quatre arêtes obtenues sont notées $\{x, x_4\}$, $\{x'_4, x'\}$, $\{z, z_4\}$ et $\{z'_4, z'\}$. Les deux réécritures possibles et les notations des brins sont schématisées dans la table 4.2.

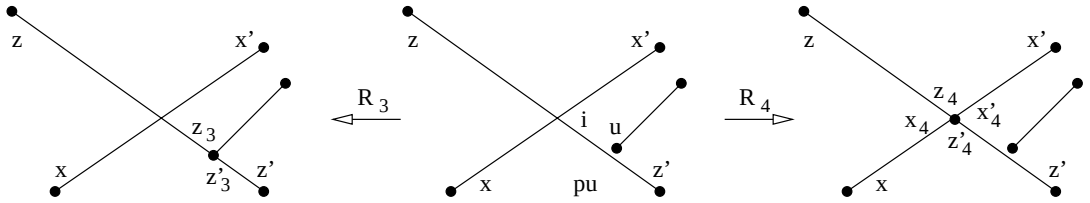


FIG. 4.2: *Notation utilisée pour les brins*

Pour construire les suites de réécritures permettant de fermer le diagramme, il faut alors considérer trois cas, représentés dans la figure 4.3, en fonction des positions relatives, sur l'arête de z, des points i et p_u où ont eu lieu les découpes d'arêtes.

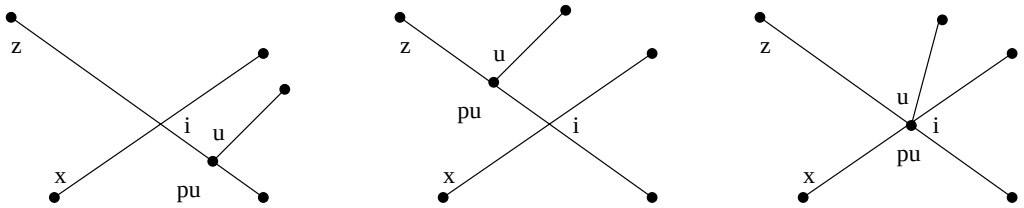


FIG. 4.3: *Les trois positions possibles pour les points i et p_u*

Premier et second cas : *Dans les deux premiers cas, la démonstration consiste à appliquer R_4 au terme obtenu avec R_3 et inversement à appliquer R_3 à celui obtenu avec R_4. Nous allons détailler le premier cas qui est en fait celui qui est schématisé dans la figure 4.2.*

Dans le membre de gauche, obtenu par R_3, les arêtes $\{x, x'\}$ et $\{z, z_3\}$ sont encore sécantes et leur point d'intersection est encore i. Ceci peut être démontré en remplaçant, dans le calcul d'intersection, le point sur lequel est plongé z' par le point p_u sur lequel est maintenant plongé z_3, et en utilisant le fait que p_u était incident à l'arête $\{z, z'\}$. On peut donc lui appliquer R_4.

Par contre, dans le membre de droite, obtenu par R_4, le sommet de u n'est plus incident à l'arête de z, mais à celle de z'_4, ce qui se démontre de la même façon. On peut alors appliquer R_3 aux brins z'_4 et u.

Dans les deux membres, pour pouvoir appliquer la règle déterminée ci-dessus, il faut pouvoir exhiber les générateurs $gp0$ voulus en tête de termes. Cela est réalisé en utilisant les

axiomes de la fonction cap pour réduire les termes à des générateurs de base. Comme pour la démonstration de la terminaison, les préconditions peuvent être utilisées pour déduire la forme que doivent avoir ces termes. Après applications des règles correspondantes et simplification, on obtient deux cartes identiques, mais avec des brins différents, les nouveaux brins $z_3, \ldots, x'_4$ *étant créés avec deux ordres distincts. Il est alors aisé de trouver l'isomorphisme de carte qui est une fonction échangeant simplement les entiers représentant ces brins.*

Troisième cas : *Dans ce cas,* p_u *est égal à* i*. Si on commence par appliquer* R_4*, on coupe les arêtes* $\{x, x'\}$ *et* $\{z, z'\}$ *au point* i*. On obtient donc trois sommets (les deux nouveaux et* u*) dont deux sont plongés sur* i *et un sur* p_u*. Par contre, si on commence par appliquer* R_3*, on coupe l'arête* $\{z, z'\}$ *en* p_u*. Après permutations et simplification, on peut monter que, comme* $i = p_u$*, le sommet* u *est incident à l'arête* $\{x, x'\}$*. On applique alors* R_3 *qui coupe cette arête en* p_u *et on obtient ainsi trois sommets plongés sur* p_u*.*

Pour refermer complètement le diagramme il suffit alors d'appliquer la règle de fusion de sommets sur ces trois sommets jusqu'à n'avoir plus qu'un sommet plongé sur p_u*. Alors, on obtient des cartes égales à un isomorphisme près, qui correspond encore à un renommage des nouveaux brins apparus pendant le processus. Ceci conclut la démonstration du fait que le pic est joignable.* □

Une démonstration de même type peut être effectuée pour les autres pics. Nous affirmons donc que le système est localement confluent. Comme il est à terminaison finie, il est donc *convergent* [25]. Ceci implique que pour chaque terme il existe une *forme normale unique* modulo $\overset{*}{\underset{E}{\simeq}}$. Le système de réécriture peut donc être vu comme une fonction de normalisation des cartes qui associe à une carte quelconque, une unique carte bien plongée : le raffinement de cette première.

4.4 Complétion des labels

4.4.1 Principe de base

Après raffinement, une carte est bien plongée. Les cellules des différents objets qu'elle modélise ont été découpées, en cellules plus petites, ou fusionnées. De plus, toutes les relations d'incidence ou d'adjacence existant entre elles, ont été mises à jour ou complétées. Toutes les intersections ou superpositions qui existaient entre des objets de la carte de départ, ont disparu. Ainsi, si deux objets de la carte de départ se recouvraient, leur intersection a été implicitement calculée et est modélisée, après le raffinement, par un ensemble de cellules.

Pour pouvoir évaluer, une expression booléenne entre des objets modélisés par une carte, il faut pouvoir déterminer à quel(s) objet(s) correspondent les cellules du raffinement de celle-ci. Traditionnellement, cette classification des cellules est réalisée en recherchant pour chaque cellule le ou les objets auxquels elle appartient. Pour cela, on utilise des tests géométriques comme des localisations de points ou l'utilisation de normales indiquant l'intérieur d'une cellule [72, 66], souvent combinés avec des informations obtenues durant le calcul d'intersection [49].

La méthode que nous présentons ici, basée sur la complétion des labels, utilise uniquement les informations topologiques existant dans le raffinement, et en particulier les relations

d'adjacence et d'incidence entre les cellules. Le principe de base est de détecter les cellules partagées par plusieurs objets, après le raffinement, et de compléter leurs labels en tenant compte des nouvelles relations d'incidence ou d'adjacence trouvées. Avant de détailler ce mécanisme de complétion, examinons un exemple simple.

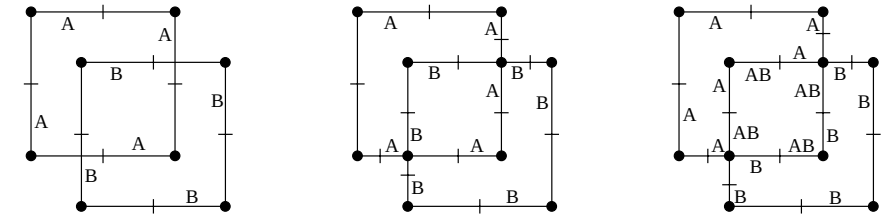


FIG. 4.4: *Complétion des labels pour l'intersection de deux faces*

Exemple 4.4.1 *Considérons deux faces orientées sécantes appartenant à deux objets, d'identificateurs A et B, représentées dans la figure 4.4. Avant le raffinement, les 2-labels des brins de ces deux faces valaient respectivement les ensembles $\{A\}$ et $\{B\}$, notés A et B. Après le raffinement, la face correspondant à leur intersection est formée de morceaux d'arêtes des deux faces initiales. Les brins de ces arêtes ont des 2-labels valant A ou B. On a donc une face dont les brins ont des labels différents. On peut en déduire qu'elle appartient à l'intersection recherchée. Les 2-labels des brins de cette face doivent donc être remplacés par l'union des 2-labels, soit ici l'ensemble $\{A, B\}$, noté AB.*

De plus, les arêtes qui appartenaient à la face B, avant le raffinement, et dont les 2-labels ont été mis à jour, sont à l'intérieur de la face de A. Le 1-label de ces arêtes doit donc également être modifié. Le même raisonnement peut être effectué pour les sommets et leur 0-label. □

4.4.2 Modification à apporter au raffinement

Pour pouvoir appliquer le mécanisme de complétion des labels, il faut que la phase de raffinement préserve l'ensemble des labels d'une carte. D'une part, il ne faut pas créer de vide ou de trous. Ainsi, lorsque des nouveaux brins sont insérés dans la carte, ils héritent de labels cohérents avec les labels des brins auxquels ils sont reliés. D'autre part, il ne faut pas perdre d'information. Ainsi, lorsque deux cellules sont fusionnées, les labels des brins de la nouvelle cellule sont modifiés pour regrouper l'ensemble des labels initiaux.

Pour assurer cette préservation des labels, les opérations géométriques utilisées pour le raffinement sont remplacées par des opérations prenant en compte les labels. Ces opérations sont celles qui sont décrites dans la section 3.3. Les transformations qu'elles induisent sur le système de réécriture sont décrites ci-après.

La première modification apparaît dans la règle R_2 qui correspond à une fusion d'arête. Elle supprime une arête $\{z, t\}$ superposée à une seconde arête $\{x, y\}$. Les labels portés par les brins z et t sont transmis aux brins x et y avant que leur arête ne soit supprimée. La suppression et la transmission des labels sont réalisées par la fonction $fapl(C, x, z)$ qui remplace la fonction

$dee(C, z)$ qui supprimait simplement l'arête de z. Pour $i \in [0, 2]$, le i-label de x est remplacé par l'union des i-labels de x et z. Les labels de t sont transmis de la même façon à y.

Dans les règles R_3 et R_4 qui réalisent des coupures d'arêtes, on remplace la fonction $cap(C, x, p)$ qui coupe l'arête de x au point p, par la fonction $capl(C, x, p)$. Cette fonction utilise *cap* pour couper l'arête et initialise les labels des nouveaux brins. Leurs 1- et 2-labels sont les copies des labels des brins de départ. Le 0-label des nouveaux brins est initialisé avec le 1-label de x, car les objets auxquels appartient ce nouveau sommet sont ceux auxquels appartenait l'arête.

Pour la règle R_5 qui fusionne deux sommets superposés, le seul changement consiste à remplacer le 0-label des brins du nouveau sommet par l'union de leurs 0-labels. Ceci est réalisé par la fonction $fusionl(C, x, z)$ qui fusionne les sommets de x et z avec la fonction $fusion(C, x, z)$ et modifie les 0-labels.

Ces modifications assurent qu'aucun label vide n'est ajouté et que les informations portées par les brins supprimés ou dont les cellules sont fusionnées ne sont pas perdues. Ces changements ne perturbent en rien les transformations géométriques réalisées pendant la phase de raffinement. Il est ainsi aisé de voir que les propriétés démontrées pour le premier système de réécriture restent valables. Une démonstration formelle pourrait en être donnée facilement en prenant en compte dans les démonstrations précédentes la présence des générateurs de base pour les labels.

4.4.3 La règle de complétion des labels

Pendant la phase de raffinement et avec les modifications locales décrites ci-dessus, les 0- et 1-labels sont correctement mis à jour. Seuls les 2-labels peuvent être différents pour deux brins d'une même face (orientée). Ce sont ces différences qui permettent de localiser les endroits où les relations d'incidence ou d'adjacence peuvent être utilisées pour compléter les labels et pour déduire les objets auxquels appartient une cellule nouvellement créée.

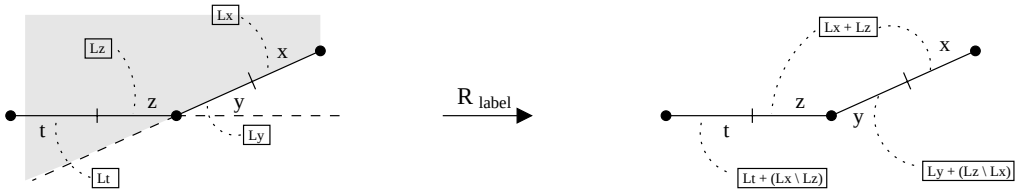


FIG. 4.5: *Illustration de la règle de complétion des labels*

La complétion est réalisée par une unique règle effectuant des modifications locales. Elle n'intervient qu'aux endroits où deux brins adjacents dans une même face ont des 2-labels différents et applique à ces brins la fonction *comp* qui complète localement les labels. Sa description formelle est donnée dans la table 4.4 et schématisée dans la figure 4.5. La fonction *comp* est détaillée ci-après.

Tout d'abord, rappelons que la face du brin x est située à droite de l'arête orientée de x. Pour simplifier, nous parlerons de la droite ou de la gauche d'un brin. Les objets auxquels appartient la face de x, qui sont les objets du 2-label de x, seront simplement appelés les objets à droite de x. Le brin z est l'image par φ_1 de x et donc son successeur dans la face de

TAB. 4.4: *Règle de complétion des labels*

$R_{label} : \dfrac{C}{comp(C,x,z)}$ **si** $\begin{cases} x \in C \wedge z = \varphi_1(C,x) \\ label2(C,x) \neq label2(C,z) \end{cases}$

x. Nous notons y et t les images respectives de x et z par α_0, f_x, f_y, f_z et f_t les faces des brins x, y, z et t et Lx, Ly, Lz et Lt leurs 2-labels.

Si Lx et Lz sont différents, leurs arêtes avant le raffinement appartenaient à des faces distinctes et donc à des objets différents. La face unique à laquelle appartiennent x et z après le raffinement appartient donc à la fois aux objets des ensembles Lx et Lz. Ainsi, les 2-labels de x et z sont remplacés par l'union, notée $Lx + Lz$, de ces ensembles.

De plus, avant le raffinement, la face f_x continuait peut-être sur la gauche de l'arête de z. La zone occupée par f_x avant raffinement est représentée en gris dans la figure 4.5, le bord éventuel de f_x et f_z avant raffinement est représenté en pointillés. Les objets auxquels appartient cette zone sont les objets de Lx dont l'arête de z n'est pas une frontière. Les objets dont l'arête de z est une frontière sont ceux qui sont dans le 2-label de z.

Ainsi, les objets à droite de x continuant à gauche de z sont ceux qui appartiennent à Lx, mais pas à Lz. L'ensemble de ces objets est donc la différence ensembliste $Lx \setminus Lz$. Ainsi, on ajoute au 2-label de t l'ensemble de ces objets et donc Lt et remplacé par $Lt + (Lx \setminus Lz)$. Réciproquement, les objets à droite de z continuant à gauche de x sont ceux qui appartiennent à Lz, mais pas à Lx. Ainsi, le 2-label de y est remplacé par $Ly + (Lz \setminus Lx)$.

Intuitivement, on peut voir ce mécanisme de complétion des 2-labels comme leur propagation générale à travers toutes les faces de la carte. Précisément, pour chaque brin on essaye de transmettre les objets de son 2-label aux faces qui lui sont adjacentes. Les arêtes, qui modélisent les bords des objets initiaux, peuvent être vues comme des filtres empêchant le passage de certains objets. L'utilisation de la différence ensembliste correspond à ce filtrage.

Intéressons nous maintenant aux 1-labels des arêtes de x et z. Lorsque des objets sont transmis d'une face à l'autre cela signifie que l'arête incidente à ces deux faces est à l'intérieur des objets transmis. Les objets transmis de f_x à f_t sont les objets de $Lx \setminus Lz$. On en déduit qu'après le raffinement l'arête $\{z, t\}$ se trouve à l'intérieur de ces objets qui sont donc ajoutés à l'ensemble des objets auxquels appartient cette arête. Ainsi, on ajoute aux 1-labels de z et t l'ensemble $Lx \setminus Lz$. De même, on ajoute aux 1-labels de x et y l'ensemble $Lz \setminus Lx$.

Le même principe est appliqué aux 0-labels de x et t, avec cependant quelques difficultés supplémentaires. Parmi les objets qui sont transmis de f_x à f_t, certains recouvrent le sommet de t. Sur l'exemple de la figure, certains des objets de la zone grise s'étendent au-delà du sommet de t. Les objets qui ne s'étendent pas sont ceux dont le bord possède une arête incidente au sommet de t. Ainsi, on calcule l'union, $ULt = \bigcup_{t' \in \langle \alpha_1 \rangle(t)} label2(C, t')$, des 2-labels des brins du sommet de t. Tous les objets de cette union ont leurs bords qui passent par le sommet de t. Les objets qui s'étendent au-delà de t sont donc ceux de l'ensemble $(Lx \setminus Lz) \setminus ULt$ et sont donc ajoutés au 0-label de t. De même les objets de $(Lz \setminus Lx) \setminus ULx$ sont ajoutés au 0-label de x, avec $ULx = \bigcup_{x' \in \langle \alpha_1 \rangle(x)} label2(C, x')$.

4.4.4 Terminaison de la complétion des labels

Pour démontrer la terminaison du processus de complétion, il est possible de construire une preuve similaire à ce qui a été effectué précédemment. Une mesure basée sur la taille des labels, c'est-à-dire sur le nombre d'objets de ces ensembles, peut être définie. Nous ne détaillons pas ici la définition de cette mesure, ni la démonstration de la terminaison. Cependant, nous donnons les arguments sur lesquels est basée cette preuve.

Lorsque la règle de complétion est appliquée des objets sont ajoutés aux labels des différents brins mis en cause. Le nombre de ces objets est toujours strictement positif puisqu'ils proviennent de la différence de deux ensembles distincts. Or le nombre d'objets contenus dans un label est borné par le nombre total d'objets présents dans la carte. Après un nombre fini d'applications de la règle, plus aucun objet ne peut donc plus être ajouté et le processus termine.

La mesure permettant de prouver cette terminaison consiste à compter le nombre d'objets qu'il est possible d'ajouter, c'est-à-dire la somme des différences entre les tailles des labels et la taille maximale correspondant à un label contenant tous les objets de la carte. Ce nombre d'objets diminue à chaque application de la règle et est positif, ce qui permet de conclure.

Après la complétion des labels, tous les brins d'une même face ont le même 2-label. Ainsi le résultat de la complétion des labels sur une carte raffinée, est une carte bien plongée et bien labellée qui permet donc une définition sans ambiguïté des objets modélisés par la carte.

4.4.5 Exemple de raffinement

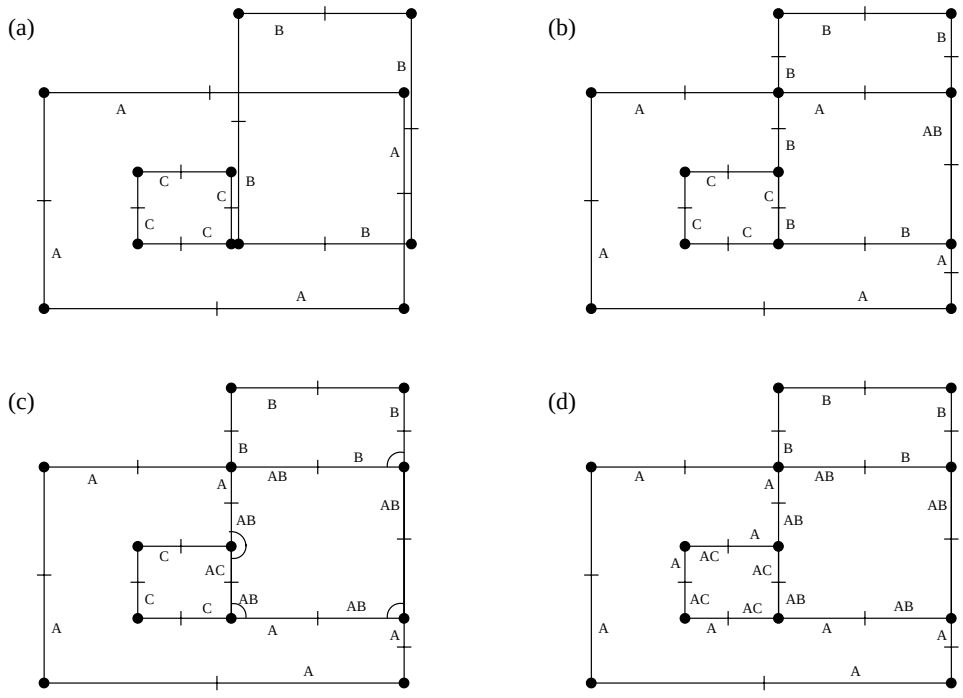


FIG. 4.6: *Exemple de raffinement et de complétion*

Exemple 4.4.2 *Un exemple plus complet de raffinement et de complétion des labels est donné dans la figure 4.6. Pour simplifier le schéma, seuls les 2-labels des brins sont représentés par des lettres situées à droite de leurs arêtes. La figure (a) contient une carte définissant trois objets A, B et C. La figure (b) montre son raffinement et la conservation des 2-labels. Notons que le label de l'arête fusionnée est complété. La figure (c) montre l'état des labels après quatre exécutions de la règle. Enfin, la figure (d) montre le résultat final.* □

Cet exemple limité aux 2-labels correspond en fait au cas d'un raffinement sur des objets régularisés. Le raffinement et la complétion des labels que nous avons définis s'appliquent, en toute généralité, à des objets non régularisés, en particulier, grâce au fait que l'on distingue les labels de différentes dimensions.

4.5 Influence des approximations numériques

Dans le cas où les nombres et l'arithmétique utilisés font intervenir des approximations numériques, des erreurs d'arrondis ou une égalité à un seuil ε près, les calculs numériques influencent le résultat et les propriétés logiques du raffinement. Bien que ces problèmes numériques ne soient pas notre principal centre d'intérêt, les méthodes formelles que nous utilisons nous permettent quelques remarques intéressantes. Tout d'abord les lieux où interviennent ces approximations et l'influence qu'elles peuvent avoir sur le raffinement sont circonscrits à certains endroits stratégiques. Les calculs numériques n'interviennent, en effet, que dans les tests géométriques apparaissant dans les conditions de déclenchement des règles.

En particulier, durant le raffinement, les transformations topologiques ou géométriques appliquées à une carte n'utilisent aucun calcul numérique. Ainsi les problèmes numériques n'ont pas d'influence sur la validité du résultat, dans le sens où les cartes obtenues vérifient toujours leurs contraintes d'intégrités, même si des erreurs numériques ou d'arrondis peuvent conduire à des résultats inattendus du point de vue du plongement. Ce premier point est obtenu par l'utilisation systématique de la topologie, grâce au modèle de représentation des objets et en raison de la nature des opérations utilisées pour la transformation des cartes.

Si les problèmes numériques ne perturbent pas la validité du résultat, ils jouent tout de même un rôle important au niveau des propriétés logiques du système de réécriture. Dans les démonstrations de terminaison et de confluence du système, on peut remarquer deux types d'arguments. Le premier regroupe des arguments apparaissant dans toutes les démonstrations, mais qui sont seuls à intervenir dans la preuve de la terminaison et dans l'étude des chutes et ponts critiques. Ces arguments sont des considérations sur les permutations possibles des générateurs et sur l'utilisation des axiomes définissant les opérateurs et leurs préconditions. Pour ce type d'arguments les approximations numériques ne jouent aucun rôle.

Le second type d'arguments intervient dans l'étude des pics critiques pour la confluence. Ils consistent à montrer que si deux règles peuvent s'appliquer sur un même terme, alors si on applique la première, les conditions de déclenchement de la seconde règle sont encore vérifiées ce qui permet de l'appliquer et réciproquement. Les arguments utilisés se divisent en deux sous-goupes que nous résumons informellement en confondant les cellules avec leurs plongements.

Le premier groupe d'arguments correspond à la transitivité de l'égalité des sommets et donc des arêtes. Elle assure que si trois sommets ou trois arêtes sont superposés alors leur

fusion peut être réalisée dans n'importe quel ordre. Le second groupe d'arguments repose sur le fait que les coupures d'arêtes ne perturbent pas la géométrie de celles-ci. Par exemple, pour la coupure par incidence, si un sommet est incident à une arête et si celle-ci est découpée par une seconde règle, alors après la coupure le sommet est encore incident à une des deux arêtes obtenues.

Ces deux arguments sont les seuls dont la démonstration est basée sur les propriétés des opérations numériques. En cas d'erreur d'approximations ou d'arrondis, leurs preuves ne sont plus valides et entraînent la non-confluence du système. C'est le formalisme utilisé pour ces preuves qui permet de situer précisément les endroits où les approximations numériques ont une influence cruciale. Ceci nous permet sur quelques exemples de donner des pistes possibles pour résoudre ces problèmes. Nous commençons par étudier les problèmes liés à la non-transitivité des tests d'égalité pour les sommets.

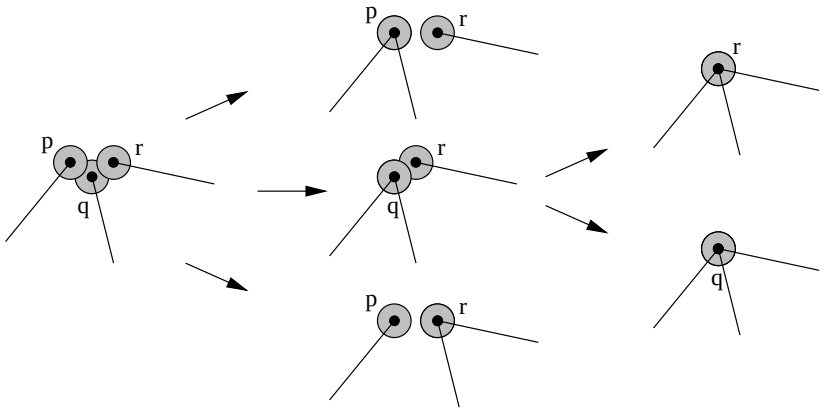


FIG. 4.7: *Exemple de suites de réécritures non transitives*

Exemple 4.5.1 *La figure 4.7 montre un exemple où le fait que l'égalité de sommets ne soit pas transitive, permet quatre suites de réécritures qui ne sont pas confluentes. L'égalité est ici supposée calculée à un ε près. Les cercles en gris ont un rayon de taille ε et représentent les zones d'influence pour l'égalité. La première réécriture fusionne p et q sur le point p (n'oublions pas qu'il n'y a qu'un plongement par sommet) et la dérivation s'arrête puisque p et r sont différents. La seconde et la troisième suites fusionnent p et q sur q, puis q et r sur q ou r. La quatrième fusionne q et r sur r et s'arrête car p et r sont distincts.* □

Cela nous conduit vers deux pistes pour assurer la confluence en cas d'égalité calculée à un ε près. La première est numérique et consiste en l'utilisation d'une grille discrète dans le plan. Deux sommets sont égaux s'ils appartiennent à la même case de la grille. Cela permet d'assurer la transitivité de l'égalité, bien que deux points très proche puissent appartenir à des cases différentes et donc être différents.

La seconde solution est topologique. Au lieu de n'avoir qu'un plongement par sommet, il suffit d'avoir un plongement par brin. Ceci autorise des sommets formés de brins égaux deux à deux, mais globalement différents. L'égalité n'est toujours pas transitive, mais la fusion de sommet le devient, grâce à la transitivité de la relation d'incidence topologique, ce qui assure la confluence à ce niveau.

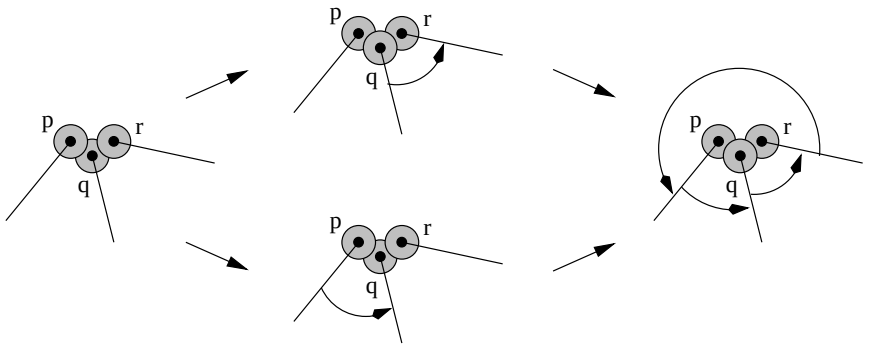


FIG. 4.8: *Exemple de suites de réécritures avec un plongement par brin*

Exemple 4.5.2 *La figure 4.8 montre les suites obtenues sur le même exemple, mais en autorisant un plongement par brin. Les arcs de flèches orientés représentent les liaisons par α_1 qui modélisent la relation d'incidence topologique assurant la transitivité de la fusion de sommets. On peut voir sur cet exemple que la confluence est assurée. Les sommets p et r sont géométriquement distincts, mais la transitivité des liaisons par α_1 assure leur égalité topologique.* □

Le second type de problèmes rencontrés, lorsque des approximations et arrondis sont utilisés, est relatif aux perturbations créées par l'application de transformations géométriques. Précisément, lorsqu'une arête est découpée en un point, deux nouvelles arêtes sont créées. Chacune de ces arêtes est définie par un sommet de l'arête initiale et le point de coupure. Or ce point de coupure est obtenu par des calculs numériques approximés pour l'incidence ou arrondis pour l'intersection. Donc, les deux nouvelles arêtes peuvent avoir des plongements dont l'union est différente du plongement de l'arête initiale.

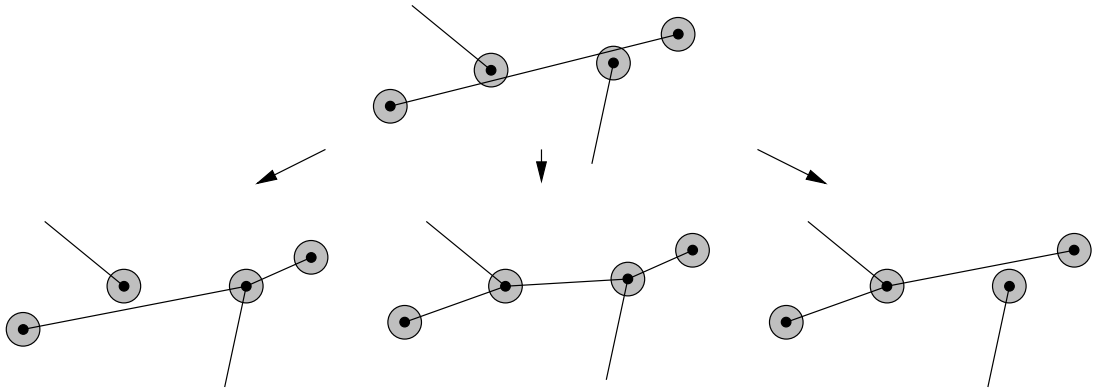


FIG. 4.9: *Exemple de réécritures avec des perturbations par déplacement*

Exemple 4.5.3 *La figure 4.9 montre les suites de réécritures possibles si la coupure d'arête induit des perturbations. Dans le premier et le troisième cas, après coupure de l'arête à un premier sommet le second sommet n'est plus incident aux arêtes résultant de la découpe. Dans le second cas, la perturbation est suffisamment réduite pour que les deux découpes par incidence aient lieu.* □

Une piste éventuelle pour résoudre ce cas est de définir le 1-plongment d'une arête explicitement et non plus implicitement. On peut par exemple associer à chaque arête une droite à laquelle appartient le segment sur lequel elle est 1-plongée. Les deux arêtes résultant de la découpe d'une arête héritent de son 1-plongement. Si les calculs d'incidence ou d'intersection s'effectuent sur ce 1-plongement, les déplacements décrits ci-dessus disparaissent.

Enfin, la dernière remarque que nous faisons a trait à l'argument utilisé pour écarter certains pics critiques. Nous avions fait la remarque que les conditions de déclenchement étaient disjointes et que, par exemple, si deux arêtes sont sécantes, alors un sommet de l'une ne peut être incident à l'autre. Les définitions de l'égalité des sommets et arêtes, de l'incidence et de l'intersection doivent toujours vérifier cette propriété pour assurer la confluence.

Les quelques remarques que nous avons données ici ne remplacent pas une étude des problèmes d'approximation nécessaire en elle-même. Mais elles permettent de recentrer ce type d'études sur les endroits stratégiques où l'influence de ces problèmes est grande.

Chapitre 5

Description de stratégies et algorithmes concrets

Dans le chapitre précédent, nous avons vu que le raffinement, d'une carte plongée peut être décrit formellement par un système de réécriture modulo. Un tel système décrit parfaitement les transformations à effectuer sur une carte pour qu'elle devienne bien plongée et bien labellée. Cependant, ce système ne fournit aucune indication sur l'ordre dans lequel les transformations doivent être effectuées, ni sur la complexité, en terme de nombre de tentatives de réécriture, d'un tel processus. Notons encore que la terminaison et la confluence du système assurent que nous pouvons choisir n'importe quelle stratégie pour l'application des règles. Puisqu'il n'existe pas de suite infinie de réécritures toute stratégie termine et comme le système est confluent, toute stratégie conduit au même résultat.

Notre idée est que tous les algorithmes permettant d'évaluer un raffinement utilisent le même type de règles et se différencient principalement par le choix d'une stratégie pour leur application. Or cette notion de stratégie n'existe pas explicitement dans la syntaxe ou la forme des systèmes de réécriture abstraits. De même, des notions classiques comme l'aspect séquentiel des opérations, le parcours d'une structure telle qu'une carte ou la structuration d'un algorithme par des boucles, inhérentes à la notion d'algorithme, sont difficiles à exprimer dans le contexte de la réécriture.

La réécriture d'une carte est non déterministe. Les permutations des générateurs qui sont utilisées pour déclencher les règles, correspondent en fait à un parcours des brins et des couples de brins de la carte. Le choix de l'ordre dans lequel sont pratiquées ces permutations étant a priori aléatoire, l'ordre de parcours des brins et donc l'ordre d'exécution des règles est également aléatoire, ou plutôt *non déterministe*. Décrire concrètement un algorithme déterministe, à partir du système de réécriture, revient en fait à fixer une stratégie de parcours des brins. L'étude de la classe de tous les algorithmes de raffinement se ramène ainsi à l'étude des stratégies de parcours, et la complexité de ces algorithmes correspond alors au nombre de brins et de couples parcourus.

Dans ce chapitre, nous proposons une approche permettant d'expliciter ces notions. Cette approche est basée sur l'adjonction de *structures de contrôle* aux règles de réécriture. Nous l'utiliserons pour étudier diverses stratégies et pour obtenir une description formelle, mais plus explicite, des algorithmes correspondants. Enfin, nous verrons comment les systèmes de réécriture sont transformés, ou raffinés au sens du génie logiciel, à partir du système

initial, pour décrire des stratégies de plus en plus efficaces, jusqu'à l'obtention d'une stratégie optimale de balayage du plan. Ceci nous permettra de retrouver les algorithmes classiques de raffinement, du plus naïf au plus efficace en terme de complexité.

5.1 Utilisation de structures de contrôle

Une utilisation naïve du système de réécriture que nous avons décrit au chapitre 4 consiste, d'une part, à tester pour chaque brin et chaque couple de brins, les conditions de déclenchement des règles, et, d'autre part, à exécuter celles-ci quand elles sont remplies. L'algorithme correspondant aurait la structure suivante :

```
Répéter
    | Choisir x dans C
    | Essayer d'exécuter les règles 1 et 6 avec x
    | Répéter
    |     | Choisir z dans C
    |     | Essayer d'exécuter les règles 2 à 5 avec x et z
    | Jusqu'à ce qu'aucune règle ne puisse être exécutée avec x
Jusqu'à ce qu'aucune règle ne puisse être exécutée
```

Pour décrire un algorithme concret il faut fixer une stratégie pour le choix les brins, c'est-à-dire une stratégie de parcours de la carte. Pour réaliser ceci, nous ajoutons au système de réécriture des *structures de contrôle* utilisées pour fixer le choix du brin ou du couple de brins pour lequel on essaye d'appliquer une règle. Une règle de réécriture décrit alors à la fois les transformations effectuées sur la carte et celles qui le sont sur les structures de contrôle.

Dans une règle du système de la table 4.2, le choix des brins x et y dans la carte C est effectué par celui de permutations utilisées pour les amener en tête du terme réécrit. Pour séparer ce choix de la forme du terme représentant la carte, nous pouvons également choisir x et y, de manière indéterministe, dans l'ensemble des brins de la carte. C'est cette première idée qui est présentée dans le système R_S de la table 5.1. Ce premier système prépare l'utilisation, dans la suite, de structures de contrôle quelconques.

Dans ce système de réécriture nous utilisons des ensembles dont une spécification détaillée peut être trouvée, par exemple, dans [42]. Les ensembles sont construits à partir de singletons, notés $\{x\}$, et par des unions disjointes et commutatives, notées $\{x\} \cup S$ ou $\{x_1, \ldots, x_n\} \cup S$ pour $\{x_1\} \cup \ldots \cup \{x_n\} \cup S$. Une opération de suppression d'un élément x d'un ensemble S, notée $d(S, x)$, est utilisée dans la suite.

Dans ce système, une carte et un ensemble de brin sont réécrits simultanément. Nous verrons qu'en fait cet ensemble est toujours exactement l'ensemble des brins de la carte. Le choix des brins porte maintenant sur leur appartenance à cet ensemble. Chaque règle de R_S décrit les modifications à apporter à l'ensemble de brins pour que, s'il représentait l'ensemble des brins de la carte avant réécriture, il représente après réécriture l'ensemble des brins de la carte transformée.

Les règles $R_{S_{init}}$ et $R_{S_{fin}}$ décrivent les étapes d'initialisation et de fin de réécriture avec R_S. La réécriture d'une carte C avec R_S débute en initialisant la structure de contrôle avec l'ensemble de ses brins $s(C)$. La réécriture avec le système R_S se termine lorsque les règles R_{S1} à R_{S6} ne peuvent plus être déclenchées, ce que nous notons $\neg R_{S1} \wedge \ldots \wedge \neg R_{S6}$. Cette

TAB. 5.1: *Système de réécriture structuré par des ensembles*

$$R_{S_{init}} : \frac{C}{s(C), C} \qquad R_{S_{fin}} : \frac{S, C}{C} \textbf{ si } \neg R_{S1} \wedge \ldots \wedge \neg R_{S6}$$

$$R_{S1} : \frac{\{x\} \cup S, C}{d(S, \alpha_0(C, x)), dap(C, x)} \textbf{ si } nullap(C, x)$$

$$R_{S2} : \frac{\{x, z\} \cup S, C}{d(\{x\} \cup S, \alpha_0(C, z)), dap(C, z)} \textbf{ si } \begin{cases} \neg eqa(C, x, z) \\ eqap(C, x, z) \end{cases}$$

$$R_{S3} : \frac{\{x, z\} \cup S, C}{\{x, x', y', z\} \cup S, cap(C, x, \pi_0(C, z))} \textbf{ si } \begin{cases} \neg nullap(C, z) \\ incident(C, z, x) \end{cases}$$

$$\textbf{avec } \begin{cases} x' = \alpha_0(cap(C, x, \pi_0(C, z)), x) \\ y' = \alpha_0(cap(C, x, \pi_0(C, z)), \alpha_0(C, x)) \end{cases}$$

$$R_{S4} : \frac{\{x, z\} \cup S, C}{\{x, x', y', z, z', t'\} \cup S, cap(cap(C, x, i), z, i)} \textbf{ si } secant(C, x, z)$$

$$\textbf{avec } \begin{cases} i = intersection(C, x, z) \\ x' = \alpha_0(cap(cap(C, x, i), z, i), x) \\ y' = \alpha_0(cap(cap(C, x, i), z, i), \alpha_0(C, x)) \\ z' = \alpha_0(cap(cap(C, x, i), z, i), z) \\ z' = \alpha_0(cap(cap(C, x, i), z, i), \alpha_0(C, z)) \end{cases}$$

$$R_{S5} : \frac{\{x, z\} \cup S, C}{\{x, z\} \cup S, fusion(C, x, y)} \textbf{ si } \begin{cases} \neg eqv(C, x, y) \\ eqsp(C, x, y) \end{cases}$$

$$R_{S6} : \frac{\{x\} \cup S, C}{\{x\} \cup S, dsp(C, x)} \textbf{ si } \neg class(C, x)$$

condition pourrait facilement être explicitée et réduite en prenant la conjonction des négations des conditions des règles R_{S1} à R_{S6}. Dès qu'elle est remplie, l'ensemble n'est plus nécessaire et $R_{S_{fin}}$ retourne la carte finale.

Précisément, si des brins sont supprimés de la carte, comme dans les règles $R1$ et $R2$, ils sont également supprimés de l'ensemble. Ainsi, pour R_{S1}, l'ensemble $\{x\} \cup S$ est remplacé par l'ensemble $d(S, \alpha_0(C, x))$. On a donc supprimé de l'ensemble de départ les brins x et $\alpha_0(C, x)$ qui sont les brins de l'arête nulle supprimée dans $dap(C, x)$. Le même principe est utilisé pour R_{S2}.

Lors d'une coupure d'arête, des brins sont ajoutés à la carte, comme dans les règles R_3 et R_4, et sont donc ajoutés à l'ensemble. Ces nouveaux brins sont obtenus dans la carte transformée en recherchant les nouvelles images par α_0 des brins dont l'arête a été coupée. Ainsi, pour R_{S3}, l'arête $\{x, \alpha_0(C, x)\}$ est coupée, et donc, les nouveaux brins x' et y' sont les images de x et $\alpha_0(C, x)$ dans la carte $cap(C, x, \pi_0(C, z))$. L'ensemble $\{x, z\} \cup S$ est donc remplacé par l'ensemble $\{x, x', y', z\} \cup S$.

Nous pouvons alors affirmer que les deux systèmes de réécriture sont équivalents.

Proposition 5.1.1 *Soit une carte C quelconque, alors :*

$$C \xrightarrow[R/E]{!} C' \Longleftrightarrow C \xrightarrow[R_S/E]{!} C'$$

Cette proposition signifie que la forme normale C', d'une carte C, obtenue par le système R est égale à la forme normale obtenue avec le système R_S. Nous ne donnons pas une démonstration in extenso de la proposition, mais quelques remarques doivent suffire à nous convaincre. Une démonstration formelle consisterait à vérifier par induction structurelle et en utilisant les axiomes définissant $s(C)$, que l'ensemble qui est réécrit est bien égal à l'ensemble des brins de la carte. Cette preuve ne présente aucune difficulté, mais est longue et fastidieuse.

Les règles sont écrites pour vérifier la propriété suivante :

$$C \xrightarrow[R_i/E]{} C' \Longleftrightarrow (s(C), C) \xrightarrow[R_{S_i}/E]{} (s(C'), C')$$

qui signifie que pour toute étape de réécriture de la carte C en la carte C', avec la règle R_i du système R, il est possible de réécrire de manière équivalente C et l'ensemble des brins de C, avec la règle correspondante R_{S_i} du système R_S et réciproquement.

En effet, quels que soient les brins x et z apparaissant en tête du terme représentant C et utilisés pour déclencher la règle R_i, il est possible de construire l'ensemble des brins de C de manière à ce que ces deux brins soient au début de l'ensemble et déclenchent ainsi la règle R_{S_i}. Les deux règles déclenchées par les mêmes brins effectuent les mêmes transformations et l'ensemble obtenu est bien l'ensemble des brins de C'.

Réciproquement, étant donné l'ensemble des brins de C dont les deux premiers brins déclenchent une règle R_{S_i}, il est possible par des permutations successives d'amener le plongement de ces brins en tête du terme C pour qu'ils déclenchent la règle R_i.

En fait, les permutations utilisées pour la réécriture avec le système R sont remplacées, dans le système R_S, par des permutations dans l'ensemble des brins. On pourrait ainsi, pour utiliser R_S et les systèmes que nous verrons par la suite, supprimer les axiomes de permutation des spécifications des cartes.

Cette première étape qui consiste à passer du système R au système R_S ne résout pas le problème du choix d'une stratégie, puisque le choix d'un brin ou d'un couple de brins dans

l'ensemble des brins est toujours non déterministe. Mais elle nous permet d'introduire une structure de contrôle et de séparer ainsi le choix des brins de la forme du terme représentant une carte. L'utilisation de structures de contrôle de plus en plus efficaces va nous permettre dans les sections suivantes d'obtenir les résultats escomptés.

5.2 Formalisation d'un parcours global

L'étape suivante dans la transformation du système de réécriture est de fixer explicitement le parcours des brins. La méthode la plus simple consiste à parcourir tous les brins de la carte. Pour cela, nous remplaçons l'ensemble de brins utilisé comme structure de contrôle, par deux listes de brins. La première liste L est utilisée pour parcourir les brins de la carte et pour fixer le choix du brin x apparaissant dans les règles. La deuxième liste L' permet, pour chaque brin x de L, de parcourir les brins de la carte qui n'ont pas encore été testés avec x et ainsi de fixer le choix des couples de brins.

La réécriture est initialisée avec la liste de tous les brins de la carte C, notée $l(C)$. Un brin est supprimé de L lorsqu'il ne peut plus déclencher aucune règle. Le système termine quand L est vide. A chaque fois qu'un brin est supprimé de L, la liste L' est initialisée avec les brins restants. Les brins x et z sont toujours choisis en tête des deux listes courantes. Lorsque aucune règle utilisant un couple de brins ne peut être déclenchée, le premier brin de la liste L' est supprimé.

Une spécification détaillée des listes peut également être trouvée dans [42]. Dans la suite, nous considérons qu'elles sont construites à partir de la liste vide [ ] et par l'adjonction d'un élément x en tête d'une liste L, notée $[x|L]$ ou $[x_1, \ldots, x_n|L]$ pour $[x_1|[\ldots|[x_n|L]\ldots]]$. L'opération $d(L, x)$ supprime x de la liste L quand x n'est pas le premier élément de L.

La table 5.2 montre le système de réécriture R_L où les règles de R_S ont été transformées pour prendre en compte les deux listes utilisées comme structures de contrôle. Comme avant les brins qui sont supprimés de la carte sont, dans R_{L1} et R_{L2}, également enlevés des deux listes. Ainsi, dans R_{L1}, l'arête $\{x, \alpha_0(C, x)\}$ est détruite. La liste $[x|L]$ est donc remplacée par la liste $d(L, \alpha_0(C, x))$ et la liste L' est remplacée par $d(d(L', x), \alpha_0(C, x))$. Le même principe est utilisé pour R_{L2}.

Lorsque des brins sont ajoutés à la carte, dans R_{L4}, ils le sont aussi à la première liste. On aurait également pu les ajouter à la seconde liste, mais il est facile de voir que ces nouveaux brins ne peuvent pas interagir avec les brins dont l'arête vient d'être coupée. Ainsi, dans R_{L4}, les nouveaux brins x', y', z' et t' sont ajoutés à la liste $[x|L]$ qui est remplacée par $[x, x', y', z', t'|L]$. Les brins sont simplement ajoutés en tête de liste puisque leur ordre dans celle-ci n'a pas d'importance. Le seul impératif est que le premier brin de L reste le même pour ne pas perturber le parcours. Comme dans le système R_S les nouveaux brins sont obtenus en recherchant les images par α_0 des brins dont l'arête est coupée. Les formules utilisées sont les mêmes et ne sont pas réécrites pour éviter d'alourdir l'écriture.

Ce raisonnement est également appliqué à R_{S3} qui nécessite cependant un traitement particulier. En effet, cette règle n'est pas symétrique et les tentatives de déclenchement avec des couples symétriques (x, z) et (z, x) ne sont pas équivalentes. Nous souhaitons décrire, dans le système R_L, un parcours dans lequel une paire $\{x, y\}$ n'est examiné qu'une fois, c'est-à-dire soit avec le couple (x, z), soit avec le couple (z, x). La règle R_{S3} est ainsi dédoublée en

TAB. 5.2: *Système de réécriture structuré par des listes*

$$R_{L_{init}} : \frac{C}{l(C), l(C), C} \qquad R_{L_{fin}} : \frac{[\,], [\,], C}{C}$$

$$R_{L1} : \frac{[x|L], L', C}{d(L, \alpha_0(C, x)), d(d(L', x), \alpha_0(C, x)), dap(C, x)} \textbf{ si } nullap(C, x)$$

$$R_{L2} : \frac{[x|L], [z|L'], C}{d(d([x|L], z), \alpha_0(C, z)), d(L', \alpha_0(C, z)), dap(C, z)} \textbf{ si } \begin{cases} \neg eqa(C, x, z) \\ eqap(C, x, z) \end{cases}$$

$$R_{L3} : \frac{[x|L], [z|L'], C}{[x, x', y'|L], [z|L'], cap(C, x, \pi_0(C, z))} \textbf{ si } \begin{cases} \neg nullap(C, z) \\ incident(C, z, x) \end{cases}$$

avec x', y' *les brins créés*

$$R_{L3'} : \frac{[x|L], [z|L'], C}{[x, z', t'|L], [z, z', t'|L'], cap(C, z, \pi_0(C, x))} \textbf{ si } \begin{cases} \neg nullap(C, x) \\ incident(C, x, z) \end{cases}$$

avec z', t' *les brins créés*

$$R_{L4} : \frac{[x|L], [z|L'], C}{[x, x', y', z', t'|L], [z|L'], cap(cap(C, x, i), z, i)} \textbf{ si } secant(C, x, z)$$

avec $\begin{cases} i = intersection(C, x, z) \\ x', y', z', t' \text{ les brins créés} \end{cases}$

$$R_{L5} : \frac{[x|L], [z|L'], C}{[x|L], [z|L'], fusion(C, x, y)} \textbf{ si } \begin{cases} \neg eqs(C, x, z) \\ eqsp(C, x, z) \end{cases}$$

$$R_{L6} : \frac{[x|L], L', C}{[x|L], L', dv(C, x)} \textbf{ si } \neg class(C, x)$$

$$R_{L7} : \frac{L, [z|L'], C}{L, L', C} \textbf{ si } \neg R_{L2} \wedge \neg R_{L3} \wedge \neg R_{L4} \wedge \neg R_{L5}$$

$$R_{L8} : \frac{[x|L], [\,], C}{L, L, C} \textbf{ si } \neg R_{L1} \wedge \neg R_{L6}$$

deux règles symétriques R_{L3} et $R_{L3'}$. Remarquons que dans $R_{L3'}$ les nouveaux brins z' et t' sont ajoutés à la seconde liste car ils peuvent interagir avec x, notamment pour la fusion de sommet. Pour le reste ces deux règles sont comparables à R_{L4}.

Deux règles qui n'agissent que sur les structures de contrôle, et que nous appellerons règles de contrôle, sont ajoutées au système. Elles décrivent le parcours des brins de la carte et correspondent au passage au brin suivant dans les deux listes. Quand le premier brin z de la seconde liste L' ne peut déclencher aucune des règles nécessitant un couple de brin avec le brin x de la première liste, il est supprimé de L'. Ceci revient à choisir pour z son successeur dans L' et est décrit par la règle R_{L7}. Ainsi, quand les règles 2 à 5 ne peuvent être exécutées, ce que nous notons $\neg R_{L2} \wedge \neg R_{L3} \wedge \neg R_{L4} \wedge \neg R_{L5}$, la seconde liste $[z|L']$ est remplacée par L'. Comme mentionné pour R_S cette condition de déclenchement pourrait facilement être explicitée et simplifiée.

Quand la seconde liste est vide, cela signifie que x, le brin en tête de la première liste, a été testé avec tous les autres brins. La règle R_{L8} supprime alors le brin x de la première liste et initialise la seconde liste avec les brins restants. La condition $\neg R_{L1} \wedge \neg R_{L6}$ qui apparaît dans R_{L8} précise que la suppression de x n'est réalisée que lorsque x ne peut déclencher les règles R_{L1} et R_{L6}.

Les deux dernières règles de contrôle, $R_{L_{init}}$ et $R_{L_{fin}}$, décrivent l'initialisation et la terminaison du système R_L. La réécriture commence avec les deux listes initialisées avec $l(C)$, la liste des brins de C. Elle termine quand les deux listes sont vides. De cette façon, tous les brins et couples de brins sont examinés. On pourrait montrer que chaque brin et chaque couple de brins n'est examiné qu'une fois et que comme précédemment, on a la propriété suivante :

Proposition 5.2.1 *Soit une carte C quelconque, alors :*

$$C \xrightarrow[R/E]{!} C' \Longleftrightarrow C \xrightarrow[R_L/E]{!} C'$$

Le système de réécriture R_L que nous obtenons est une description formelle, précise et complète d'une stratégie de parcours des brins d'une carte, pour le calcul de son raffinement. Cette stratégie n'est pas obtenue par une approche globale. En effet, comme les règles de réécriture sont indépendantes les unes des autres, elle est construite par une étude des différents cas et les changements sur le système de réécriture sont effectués règle par règle. Des règles de contrôle sont simplement ajoutées pour décrire le parcours. Les transformations effectuées pour écrire les règles R_{L1} à R_{L6} sont très simples et pourraient éventuellement être automatisées.

Un algorithme de forme classique peut alors facilement être dérivé du système de réécriture R_L. Sa structure est donnée par les quatre règles de contrôle. Il correspond à un parcours des couples de brins de la carte par deux boucles imbriquées parcourant chacune une liste de brins. Pour le construire nous gérons directement les règles $R_{L_{init}}$, $R_{L_{fin}}$, R_{L7} et R_{L8} et les deux listes de brins. Les brins x et z sont obtenus par la fonction $p(L)$ qui donne le premier élément d'une liste.

L'algorithme produit a alors la structure suivante :

```
L = l(C)
Tant que L ≠ []
    | x = p(L)
    | Essayer d'exécuter la règle 1 avec x
```

```
        Si la règle 1 échoue Alors
            Essayer d'exécuter la règle 6 avec x
            L' = L
            Tant que L' ≠ [ ]
                z = p(L')
                Essayer d'exécuter la règle 2 avec x et z
                Si la règle 2 échoue Alors
                    Essayer d'exécuter les règles 3 à 5 avec x et z
                    L' = d(L', z)
                FinSi
            Fin {tant que L'}
            L = d(L, x)
        FinSi
    Fin {tant que L}
```

Si on compte le nombre de tentatives d'exécutions de règles, ce simple algorithme a une complexité en $O((n+i)^2)$, n étant le nombre de brins de la carte et i le nombre de brins créés qui est proportionnel au nombre d'intersections et d'incidences.

5.3 Optimisation par l'utilisation de propriétés géométriques

Le système R_L décrit le raffinement d'une carte à travers un parcours complet des couples de brins qu'elle contient. Pour optimiser l'algorithme qui en dérive, il faut limiter le nombre de couples parcourus. Pour cela, nous étudions les conditions géométriques nécessaires, mais pas forcément suffisantes, pour que deux brins puissent interagir, c'est-à-dire déclencher une des règles du système. Il faut alors modifier le parcours, et donc les structures et règles de contrôle, pour que ces propriétés soient prises en compte.

Examinons d'abord les conditions nécessaires à l'interaction de deux brins. Si D et D' sont les droites verticales passant par les sommets d'une arête $\{x, y\}$, nous appelons zone d'interaction de l'arête, la région du plan délimitée par D et D'. Les brins qui peuvent interagir avec les brins x et y sont ceux dont les sommets sont dans la zone d'interaction de leur arête.

En effet, si deux arêtes sont sécantes ou superposées, alors un des sommets de ces arêtes appartient à la zone d'interaction de l'autre. Si un sommet est incident à une arête, il appartient à sa zone d'interaction. Et enfin, deux sommets égaux appartiennent chacun aux zones d'interaction des arêtes incidentes à l'autre sommet. Les brins dont le sommet appartient à la zone d'interaction d'une arête, sont dits *actifs* pour cette arête.

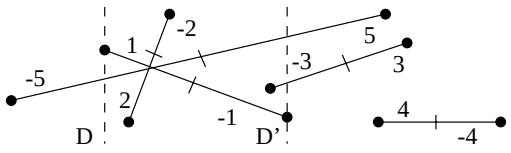


FIG. 5.1: *Région du plan délimitée par l'arête {1, -1}*

Exemple 5.3.1 *Dans l'exemple de la figure 5.1, les brins actifs pour l'arête {1, -1} sont 2, -2 et -3. Remarquons que les brins 5 et -5 ne sont pas actifs pour l'arête {1, -1}, mais qu'inversement les brins 1 et -1 sont actifs pour l'arête {5, -5}.* □

Pour que cette propriété puisse être utilisée, il faut pouvoir vérifier, par le calcul, qu'un brin est actif pour une arête donnée. Dans cette optique, nous définissons un ordre $<_C$ sur les brins d'une carte C donnée.

Définition 5.3.1 *Dans une carte C et pour deux brins x et y 0-plongés de C, les relations $<_C$ et $\leq_C$ sont définies par :*

$$\begin{aligned} x <_C y &\iff \pi_0(C,x) <_p \pi_0(C,y) \\ x \leq_C y &\iff \pi_0(C,x) <_p \pi_0(C,y) \vee eqsp(C,x,y) \end{aligned}$$

où $<_p$ est l'ordre lexicographique sur les coordonnées des points, dont la définition est donnée précisément dans la spécification POINT-2D (cf. annexe A), et $eqsp(C,x,y)$ est le test d'égalité du plongement de deux sommets, défini précédemment.

Alors, formellement, nous avons la définition suivante :

Définition 5.3.2 *Soit C une carte et un brin x fixé. Les brins actifs de C vis-à-vis de l'arête $\{x, \alpha_0(C,x)\}$ sont les brins z tels que :*

$$\begin{cases} x \leq_C z \wedge z \leq_C y & si\ x <_C y \\ y \leq_C z \wedge z \leq_C x & si\ x \geq_C y \end{cases}$$

en notant $y = \alpha_0(C,x)$.

Pour réduire le parcours des couples de brins, il faut que, pour chaque brin x choisi dans la première liste L, nous puissions facilement séparer les brins actifs des autres brins. Le meilleur moyen pour cela est de considérer que les brins de la seconde liste L' sont ordonnés de gauche à droite suivant la relation $<_C$. La liste des brins actifs est alors une sous-liste de L'.

Pour éviter de trier les brins de L' pour chaque x, il suffit de considérer que L est également ordonnée. La liste des brins actifs est alors une sous-liste de L commençant à x. Comme le brin x est toujours choisi en tête de L, L' est égale à L dont la queue est tronquée après le dernier brin actif. Pour éviter la recherche explicite de l'endroit à tronquer, L' est initialisée à L et les brins de la seconde liste sont parcourus tant qu'ils sont actifs. Ainsi, les brins sont parcourus, de gauche à droite, grâce à une liste ordonnée et seuls les couples de brins dont le deuxième membre est actif pour l'arête du premier sont parcourus.

Ce choix de parcours nous permet alors quelques simplifications pour le test d'interaction. Comme les brins sont examinés de gauche à droite, nous avons forcément $x \leq_C z$. En notant $y = \alpha_0(C,x)$, la condition exprimant que z est actif pour l'arête de x devient :

$$\begin{aligned} & \begin{cases} x \leq_C z \wedge z \leq_C y & \text{si } x <_C y \\ y \leq_C z \wedge z \leq_C x & \text{si } x \geq_C y \end{cases} \\ \iff & \begin{cases} z \leq_C y & \text{si } x <_C y \\ z \leq_C x & \text{si } x \geq_C y, \end{cases} \qquad \text{car } x \geq_C y \wedge x \leq_C z \Rightarrow y \leq_C z \text{ est } vrai \\ \iff & \begin{cases} z \leq_C y & \text{si } x <_C y \\ eqsp(C,x,z) & \text{si } x \geq_C y, \end{cases} \qquad \text{car } z \leq_C x \wedge x \leq_C z \Leftrightarrow eqsp(C,x,z) \\ \iff & [x <_C y \wedge z \leq_C y] \vee [x \geq_C y \wedge eqsp(C,x,z)] \end{aligned}$$

La condition d'arrêt pour le parcours de la seconde liste est alors la négation de la condition précédente. Ce qui donne :

$$\begin{array}{rl} & \begin{cases} z <_C x \vee y <_C z & \text{si } x <_C y \\ z <_C y \vee x <_C z & \text{si } x \geq_C y \end{cases} \\ \Longleftrightarrow & [x <_C y \wedge y <_C z] \vee [x \geq_C y \wedge \neg eqsp(C,x,z)] \end{array}$$

Le système R_F de la table 5.3 décrit le raffinement utilisant cette stratégie. Dans certaines de ses règles, comme précédemment, nous serons amenés à supprimer ou insérer des brins dans les structures de contrôle. Ceci est réalisé par deux fonctions, $i(F,x)$ et $d(F,x)$, pour, respectivement, insérer et détruire un brin dans une file. Mais ces opérations ne pourront être implantées de manière efficace si on utilise des listes linéaires ordonnées. Or le but des transformations apportées aux systèmes de réécriture est de décrire, de façon plus explicite, des algorithmes efficaces, même si cette efficacité reste invisible au niveau de la description du parcours, le nombre des couples parcourus ne changeant pas. Pour cette raison, à la place des listes, nous utilisons des *files de priorité*, que nous implantons de manière optimale par des tas et dont les spécifications sont données dans l'annexe B.

TAB. 5.3: *Système de réécriture structuré par des files de priorité*

$$R_{F_{init}} : \frac{C}{f(C), f(C), C} \qquad R_{F_{fin}} : \frac{[\,], [\,], C}{C}$$

$$R_{F1} : \frac{F, F', C}{d(d(F,x), \alpha_0(C,x)), d(d(F',x), \alpha_0(C,x)), dap(C,x)} \textbf{ si } \begin{cases} x = p(F) \\ nullap(C,x) \end{cases}$$

$$R_{F2} : \frac{F, F', C}{d(d(F,z), \alpha_0(C,z)), d(d(F',z), \alpha_0(C,z)), dap(C,z)} \textbf{ si } \begin{cases} x = p(F) \wedge z = p(F') \\ \neg eqa(C,x,z) \wedge eqap(C,x,z) \end{cases}$$

$$R_{F3} : \frac{F, F', C}{i(i(F,x'), y'), F', cap(C,x,\pi_0(C,z))} \textbf{ si } \begin{cases} x = p(F) \wedge z = p(F') \\ \neg nullap(C,z) \wedge incident(C,z,x) \end{cases}$$

$$R_{F4} : \frac{F, F', C}{i(i(i(i(F,x'), y'), z'), t'), F', cap(cap(C,x,i), z, i)} \textbf{ si } \begin{cases} x = p(F) \wedge z = p(F') \\ secant(C,x,z) \end{cases}$$

$$R_{F5} : \frac{F, F', C}{F, F', fusion(C,x,y)} \textbf{ si } \begin{cases} x = p(F) \wedge z = p(F') \\ \neg eqs(C,x,z) \wedge eqsp(C,x,z) \end{cases}$$

$$R_{F6} : \frac{F, F', C}{F, F', dv(C,x)} \textbf{ si } \begin{cases} x = p(F) \\ \neg class(C,x) \end{cases}$$

$$R_{F7} : \frac{F, F', C}{F, d(F',z), C} \textbf{ si } \begin{cases} x = p(F) \wedge z = p(F') \wedge \neg(R_{F2} \ldots R_{F5}) \\ [x <_C y \wedge z \leq_C y] \vee [x \geq_C y \wedge eqsp(C,x,z)] \end{cases} \textbf{ avec } y = \alpha_0(C,x)$$

$$R_{F8} : \frac{F, F', C}{d(F,x), d(F,x), C} \textbf{ si } \begin{cases} x = p(F) \wedge z = p(F') \wedge \neg(R_{F1} \vee R_{F6}) \\ [x <_C y \wedge y <_C z] \vee [x \geq_C y \wedge \neg eqsp(C,x,z)] \end{cases} \textbf{ avec } y = \alpha_0(C,x)$$

Pour l'essentiel le système R_F reprend les règles de la table 5.2. Pour les règles 1 à 6, il n'y a pas de changement notable, excepté le fait que les brins ne sont plus choisis en tête de liste, mais obtenus par la fonction $p(F)$ qui donne le plus petit élément de la file de priorité.

Notons cependant que, comme $x \leq_C z$ pour tout couple choisi (x, z), il n'est plus nécessaire de dédoubler la règle 3.

Les changements les plus importants se situent au niveau des règles de contrôle. Les règles 7 et 8 utilisent les conditions définies précédemment pour limiter le parcours des brins de la seconde file aux brins actifs pour le brin courant de la première file. Ceci est réalisé selon le même principe que pour le système précédent. Les règles d'initialisation sont les mêmes que pour R_L. La règle $R_{F_{init}}$ initialise la réécriture avec la file de priorité $f(C)$ contenant tous les brins de la carte C ordonnés par la relation $<_C$. Enfin, la réécriture se termine avec $R_{F_{fin}}$ lorsque les files de priorité sont vides.

La stratégie définie dans le système R_F correspond à un algorithme de balayage simple du plan. Sa complexité reste en $O((n+i)^2)$, n étant le nombre de brins de la carte et i le nombre de brins créés, dans le pire des cas. En effet, il est possible de construire des cartes telles que, pour chacun de leurs brins, tous les autres brins soient actifs. Cependant en moyenne on obtient une amélioration sensible de la vitesse de calcul, surtout lorsque les brins sont plongés sur des segments de tailles homogènes et régulièrement répartis dans le plan.

5.4 Un algorithme optimal par balayage

L'algorithme de balayage obtenu dans la section précédente n'est pas optimal car, pour un brin x fixé d'une carte C, tous les brins actifs sont examinés. En prenant quelques précautions, il est suffisant d'examiner le brin actif dont l'arête est juste en dessous de l'arête $\{x, \alpha_0(C, x)\}$ et celui dont l'arête est juste au-dessus.

Pour pouvoir rechercher efficacement ces deux brins, l'ensemble des brins actifs pour une arête donnée est trié de bas en haut et ainsi la seconde file de priorité est remplacée par un *dictionnaire binaire*, ou *arbre binaire de recherche*, dont les spécifications sont données dans l'annexe B. Pour éviter de trier l'ensemble des brins actifs pour chaque brin de la première file, le dictionnaire est maintenu par chaque règle et mis à jour par les règles de contrôle lorsque l'on passe au brin suivant. On retrouve ainsi, les structures classiques utilisées pour les algorithmes de balayage et souvent nommées X et Y, comme dans [8, 62, 16].

Ainsi, pour résumer, le système de réécriture débute avec tous les brins de la carte, ordonnés de gauche à droite, et placés dans une file de priorité X et avec un dictionnaire vide A. Le brin x est toujours le premier élément de la file X et z est obtenu en cherchant dans A les deux brins qui sont juste en dessous et au-dessus de x. Puis, lorsque les règles 1 à 6 ne sont plus déclenchables avec x et z, si x n'était pas encore actif, il est inséré dans A sinon il en est enlevé. Après cela, x est supprimé de X et l'on continue avec le brin suivant de la file de priorité.

De plus, comme des sommets égaux sont placés l'un derrière l'autre dans la file X, la fusion de sommets ne peut avoir lieu que pour deux sommets consécutifs. Ainsi, nous ajoutons une structure S qui est un tampon contenant simplement le dernier sommet utilisé et qui sera utilisé par la règle de fusion pour obtenir le brin z. La table 5.4 présente le système de réécriture modifié R_X pour décrire un parcours utilisant ces trois structures de contrôle.

Dans le système de réécriture R_X, seules les règles 1, 4 et 5 subissent des transformations pertinentes, les autres étant similaires. Les opérateurs i et s sont introduits pour respectivement insérer et supprimer un brin dans la structure correspondante. Comme avant, dans la

TAB. 5.4: *Système de réécriture avec stratégie de balayage*

$$R_{X_{init}} : \frac{C}{f(C), v, \{\}, C} \qquad R_{X_{fin}} : \frac{v, v, S, C}{C}$$

$$R_{X1} : \frac{X, A, S, C}{s(s(X, x), \alpha_0(C, x)), A, S, dap(C, x)} \textbf{ si } \begin{cases} x = premier(X) \\ nullee(C, x) \end{cases}$$

$$R_{X2} : \frac{X, A, S, C}{s(s(X, z), \alpha_0(X, z)), A, S, dap(C, z)} \textbf{ si } \begin{cases} x = premier(X) \wedge z = gequal(A, C, X) \\ \neg eqa(C, x, z) \end{cases}$$

$$R_{X3} : \frac{X, A, S, C}{i(i(X, x'), y'), A, S, cap(C, x, \pi_0(C, z))} \textbf{ si } \begin{cases} x = premier(X) \\ z = gincident(A, C, x) \neq nil \end{cases}$$

$$R_{X4} : \frac{X, A, S, C}{i(i(i(i(X, x'), y'), z'), t'), A, S, cap(cap(C, x, i), z, i)} \textbf{ si } \begin{cases} x = premier(X) \\ z = gsecant(A, C, x) \neq nil \end{cases}$$

$$R_{X5} : \frac{X, A, \{z\}, C}{X, A, S, fusion(C, x, z)} \textbf{ si } \begin{cases} x = premier(X) \\ \neg eqs(C, x, z) \wedge eqsp(C, x, z) \end{cases}$$

$$R_{X6} : \frac{X, A, S, C}{X, A, S, dsp(C, x)} \textbf{ si } \begin{cases} x = premier(X) \\ \neg class(C, x) \end{cases}$$

$$R_{X7} : \frac{X, A, S, C}{s(X, x), i(A, x), \{x\}, C} \textbf{ si } \begin{cases} x = premier(X) \wedge gauche(C, x) \\ \neg(R2 \vee \ldots \vee R5) \end{cases}$$

$$R_{X8} : \frac{X, A, S, C}{i(s(X, x), z), s(s(A, \alpha_0(C, x)), z), \{x\}, C} \textbf{ si } \begin{cases} x = premier(X) \wedge droit(C, x) \wedge \neg(R2 \ldots R5) \\ z = gmaxinf(A, C, x) \wedge z' = gminsup(A, C, x) \\ interaction(C, z, z') \end{cases}$$

$$R_{X8'} : \frac{X, A, S, C}{s(X, x), s(A, \alpha_0(C, x)), \{x\}, C} \textbf{ si } \begin{cases} x = premier(X) \wedge droit(C, x) \wedge \neg(R2 \ldots R5) \\ z = gmaxinf(A, C, x) \wedge z' = gminsup(A, C, x) \\ \neg interaction(C, z, z') \vee z = nil \vee z' = nil \end{cases}$$

règle 1, les brins de l'arête détruite sont supprimés de X, ils n'ont pas encore été inclus dans A. Dans la règle 4, l'opérateur $gsecant(A, C, x)$ cherche dans A une arête juste en dessous ou juste au-dessus et sécante à l'arête à laquelle x appartient et retourne un brin de cette arête ou *nil* si une telle arête n'existe pas. Ensuite les nouveaux brins sont insérés dans X. Les derniers changements apparaissent dans la règle 5. Comme mentionné précédemment, le seul brin dont le sommet peut être fusionné avec celui de x est le dernier brin examiné qui a été placé dans S. Ainsi z est le brin contenu dans S.

Les opérations i et s et les opérations de recherche dans l'ensemble des brins actifs sont spécifiées complètement dans l'annexe B. Nous avons utilisé i et s pour alléger l'écriture du système, mais ces deux fonctions sont nommées *insert* et *supprime* dans cette annexe.

FIG. 5.2: *Les deux cas d'inactivation de brins droits*

Les règles 7, 8 et 8′ sont utilisées pour maintenir X, A et S quand la zone de balayage, c'est-à-dire la zone définissant les brins actifs, se déplace vers la droite lorsqu'on passe au brin suivant de X. Ce mécanisme n'est activé que lorsque les règles 1 à 6 ne peuvent être exécutées. Dans ces trois règles le brin x est placé dans S. Si x est un brin gauche, i.e. si x est le sommet de gauche de son arête, alors son arête entre dans la zone de balayage. On est sûr que x n'interagit avec aucun des autres brins de A sinon une des règles aurait été déclenchée. La règle 7 le rend actif en l'insérant dans A et l'enlève de X pour que la réécriture continue avec le brin suivant de X.

A l'opposé, si x est un brin droit, son arête quitte la zone active. De même que précédemment, x n'interagit avec aucun des autres brins de A et doit donc être inactivé en étant supprimé de A et enlevé de X pour le passage au brin suivant.

Dans ce second cas, les arêtes qui sont juste en dessous et au-dessus de x n'ont peut-être pas été testées ensemble et peuvent donc interagir comme montré dans la figure 5.2. Ainsi, si l'une de ces arêtes n'existe pas, i.e. si la fonction de recherche retourne *nil*, ou si elles n'interagissent pas, il n'y a rien de plus à faire (règle 8'). Sinon, le brin z, dont l'arête est juste en dessous de x, est enlevé de A et réinséré dans X afin d'être examiné une fois de plus (règle 8).

Comme avant, un algorithme classique peut être dérivé du système de réécriture. Nous obtenons ainsi un algorithme de balayage [8, 62], dont la complexité est en $O((n+i)\ln(n))$. Cet algorithme est *adaptatif* [16, 61] puisque sa complexité est fonction de la taille des entrées *et* des sorties.

```
X = ensemble des brins de C ordonnés par abscisses croissantes
A = ∅
S = ∅
Tant que X ≠ ∅
    | x = premier(X)
    | Essayer d'exécuter les règles 1 à 6
    | d(X, x)
```

```
        S = {x}
        Si gauche(C, x)
            i(A, C, x)
        Sinon {x est un brin droit }
            d(A, C, α0(C, x))
            z = gmaxinf(A, C, x)
            z' = gminsup(A, C, x)
            Si z ≠ nil et z' ≠ nil et interaction(C, z, z')
                Alors d(A, C, z), i(X, z)
            FinSi
        FinSi
    Fin {tant que}
```

Nous pensons que des algorithmes plus complexes, comme celui de [20], qui ajoute des nouvelles arêtes pour rendre les faces de la subdivision convexes et a une complexité en $O(n\ln(n)+i)$ grâce à cette optimisation, peuvent être rigoureusement conçus en utilisant les mêmes méthodes. L'intérêt de notre approche est la claire séparation entre les structures de données utilisées pour modéliser les cartes et les structures de données utilisées pour améliorer le contrôle et donc la complexité des algorithmes.

Nous avons proposé un mécanisme général pour décrire n'importe quel algorithme concret de raffinement. Différentes structures de contrôle mènent à différentes stratégies et donc différents algorithmes. Une classification des algorithmes de raffinement pourrait ainsi être envisagée, basée sur le type de structures et le type de fonctions de recherche qui sont utilisées. D'autres stratégies pourraient être étudiées formellement, avec les mêmes principes, et notamment les stratégies consistant à *diviser pour régner* ou les stratégies dites de *mariage avant conquête*.

5.5 Complétion des labels

Comme le raffinement, le système de complétion des labels peut être transformé pour inclure une stratégie de parcours des brins de la carte. La complétion s'effectue par l'exécution d'une seule règle nécessitant le choix d'un brin et conduit donc à une stratégie très simple.

Tous les brins de la carte sont placés dans une file FIFO. Le brin x choisi pour essayer d'exécuter la règle est celui qui est en tête de file. Lorsque la règle est appliquée les labels des brins $x, y = \alpha_0(C,x), z = \varphi_1(C,x)$ et $t = \alpha_0(C,z)$ sont modifiés. Ces brins pourraient de nouveau déclencher la règle et sont donc ajoutés en queue de la file. Si la règle n'est pas appliquée, les labels de x ne changent pas et donc x ne peut plus déclencher la règle tant que ses labels n'ont pas été modifiés par l'application de la règle sur un brin qui lui est incident. Le brin x est donc enlevé de la file.

La table 5.5 montre le système R_{label} correspondant à cette stratégie. L'ajout d'un brin x en tête d'une file F est noté $x :: F$ et l'ajout en fin de file est noté $F :: x$. Comme d'habitude, deux règles décrivent, d'une part, l'initialisation du système avec la file $f(C)$ contenant tous les brins de C et, d'autre part, la fin du système lorsque la file est vide.

Nous avons vu que ce processus se termine, puisque lorsque la règle est appliquée à un brin, des objets sont forcément ajoutés à ses labels. Comme la taille des labels est bornée par

TAB. 5.5: *Système décrivant une stratégie pour la complétion des labels*

$$R_{label_{init}} : \frac{C}{f(C), C}$$

$$R_{label} : \frac{x :: F, C}{F :: x :: y :: z :: t, comp(C, x, z)} \textbf{ si } \begin{cases} x \in C \wedge z = \varphi_1(C, x) \\ y = C\alpha_0(C, x) \wedge t = C\alpha_0(C, z) \\ label2(C, x) \neq label2(C, z) \end{cases}$$

$$R_{label_{next}} : \frac{x :: F, C}{F, C} \textbf{ si } \begin{cases} x \in C \wedge z = \varphi_1(C, x) \\ label2(C, x) = label2(C, z) \end{cases}$$

$$R_{label_{fin}} : \frac{[\,], C}{C}$$

le nombre total d'objets, les labels d'un brin ne peuvent être modifiés qu'au plus p fois, si p est le nombre d'objets modélisés par la carte. Donc, un brin ne peut être inséré qu'au plus p fois dans la file. La complexité de l'algorithme issu de ce système est donc en $O(pn)$, où n est le nombre de brins de la carte. La complétion des labels est donc un mécanisme très efficace pour l'évaluation d'un arbre CSG.

Chapitre 6

Raffinement des cartes en dimension 3

Comme dans le cas de la dimension 2, le raffinement d'une 3-carte combinatoire plongée est décrit formellement par un système de réécriture. Les règles de ce système décrivent les transformations à effectuer sur une 3-carte pour qu'elle vérifie les conditions de bon plongement décrites dans la proposition 2.2.12. De plus, un second système de réécriture décrit les transformations permettant la complétion des labels de la carte pour qu'elle vérifie les conditions de bonne labellisation décrites dans la définition 2.3.2.

De nombreux niveaux de formalismes sont possibles pour décrire ces transformations et un système très abstrait conduiraient naturellement à une description plus simple. Celle-ci ne correspondrait cependant pas au but que nous nous sommes fixé qui est de décrire formellement, mais aussi *complètement*, le raffinement.

La description formelle doit être suffisamment *explicite*, pour permettre et faciliter l'obtention d'un prototype et d'une implantation efficace des algorithmes décrits. Le raffinement d'une 3-carte est cependant beaucoup plus complexe qu'en dimension 2 et une description globale et uniforme perdrait tout intérêt si elle était illisible. Ainsi, nous proposons une description du raffinement contenant deux niveaux de lecture.

Le premier niveau décrit le raffinement d'une 3-carte par un système de réécriture qui utilise, pour réaliser l'intersection de faces quelconques, des méta-fonctions qui sont des notations raccourcies pour des transformations complexes. Le deuxième niveau décrit précisément ces transformations par une étude de cas sur la forme de l'intersection. Les différents cas et les modifications qu'ils induisent, pour la carte à raffiner, sont décrits par plusieurs systèmes de réécriture. De même que précédemment, comme nous avons voulu une description complète et détaillée de ces fonctions et des stratégies de recherche et de parcours des différents cas, le système est enrichi par des structures de contrôle.

6.1 Le système de réécriture général

Le raffinement en dimension 3 et essentiellement basé sur la transformation des faces d'une 3-carte. Rappelons qu'une face est composée de deux faces orientées symétriques. Nous travaillons, dans la suite, sur une seule, et peu importe laquelle, des deux faces orientées d'une face, ce qui revient à considérer un brin sur deux. Pour ne pas surcharger notre discours nous confondons la face orientée choisie avec la face (non orientée) à laquelle elle appartient.

Comme pour les méthodes d'intersection classiques, nous supponsons ici que les faces sont planes et sont plongées sur des polygones dont l'intérieur est connexe. C'est-à-dire que le bord d'une face ne peut se situé dans l'intérieur du polygone qu'il définit. Ces contraintes sont classiques est nécessaire pour une définition non ambigue de l'intérieur et donc de l'intersection de deux faces.

Le premier niveau de réécriture contient quatre règles dont la forme est la même que pour les systèmes utilisés en dimension 2. La figure 6.1 schématise les deux principales règles du système. Pour améliorer la lisibilité, dans toutes les figures qui suivent, les arêtes et les faces sont éclatées et les 2-liaisons ne sont pas représentées. Le système est décrit formellement dans la table 6.1. Des explications intuitives sont données ci-après.

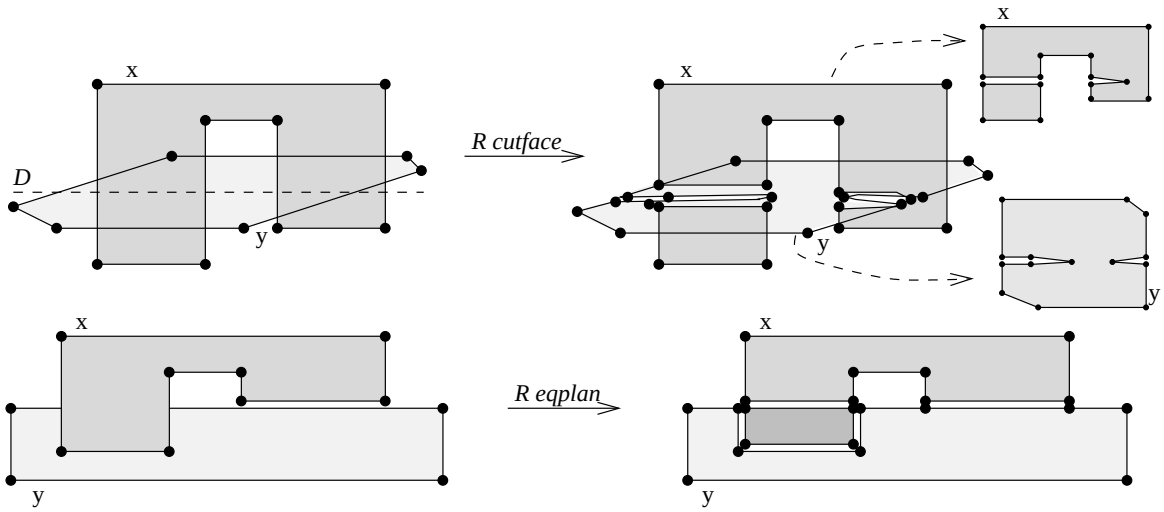


FIG. 6.1: *Illustration graphique des règles de réécriture*

La première règle, $R_{cutface}$, réalise l'intersection de deux faces non coplanaires. Lorsque les plans des faces de deux brins x et y sont sécants ($secant(C, x, y)$), les faces de x et y (dessinées séparément en haut à droite de la figure 6.1) sont découpées le long de la droite d'intersection D. Cette opération consiste d'abord à rechercher des segments de D partagés par les deux faces, puis à insérer des arêtes en fonction des différents cas de placement du segment vis-à-vis du bord de la face. Cette opération est réalisée par un mécanisme complexe, décrit dans la section suivante et que nous résumons ici par la méta-fonction $cutface$.

La deuxième règle, R_{eqplan}, réalise l'intersection de deux faces coplanaires. Lorsque deux brins x et y appartiennent à des faces distinctes ($\neg eqface(C, x, y)$) et plongées dans des plans égaux ($eqplan(C, x, y)$), les deux faces sont subdivisées. Cette opération correspond à l'intersection de deux polygones dans un plan et est réalisée par un mécanisme, résumé par la méta-fonction $refine2d$ et détaillé plus loin. Intuitivement, le principe est de réaliser le raffinement 2D des deux faces, avec le système décrit précédemment. Le résultat, qui est une 2-carte, est ensuite converti en un ensemble de faces d'une 3-carte liées par α_2. La conversion est réalisée par duplication et recollement de la 2-carte duale du résultat.

La troisième règle, R_{eqface}, est exécutée lorsque deux brins x et y appartiennent à des faces distinctes ayant pour plongement deux polygones égaux ($eqpoly(C, x, y)$). Dans ce cas, une des deux faces, ici la face de y, est supprimée par l'opération $delface(C, y)$. Les faces qui

étaient 2-cousues à la face supprimée sont 2-cousues ensuite à l'autre face, par R_{eqedge}.

La dernière règle, R_{eqedge}, complète les 2-liaisons manquantes. Lorsque deux brins x et y ont des arêtes distinctes ($\neg eqedge(C,x,y)$) et plongées sur des segments égaux ($eqseg(C,x,y)$), alors leurs deux arêtes (multiples) sont fusionnées. Précisément, les 2-liaisons des arêtes de x et y sont interclassées de manière à ordonner correctement les faces autour de l'axe formé par ces arêtes. Cette fusion est réalisée par la fonction $merge(C,x,y)$.

TAB. 6.1: *Système de réécriture pour le raffinement des cartes*

$R_{cutface} : \dfrac{C}{cutface(C,x,y)}$ **si** $\begin{cases} x \in C \wedge y \in C \\ secant(C,x,y) \end{cases}$	$R_{eqface} : \dfrac{C}{delface(C,y)}$ **si** $\begin{cases} x \in C \wedge y \in C \\ eqpoly(C,x,y) \\ \neg eqface(C,x,y) \end{cases}$
$R_{eqplan} : \dfrac{C}{refine2d(C,x,y)}$ **si** $\begin{cases} x \in C \wedge y \in C \\ eqplan(C,x,y) \\ \neg eqface(C,x,y) \end{cases}$	$R_{eqedge} : \dfrac{C}{merge(C,x,y)}$ **si** $\begin{cases} x \in C \wedge y \in C \\ eqseg(C,x,y) \\ \neg eqedge(C,x,y) \end{cases}$

Le raffinement d'une carte revient à essayer d'exécuter chaque règle pour chaque couple de brins (i.e. chaque couple de faces ou d'arêtes). Ceci conduit naturellement à un algorithme naïf dont la complexité est en $o(n^2)$, où n est le nombre de brins. Des stratégies classiques, limitant le nombre de couples à tester, peuvent être employées pour améliorer l'efficacité du calcul, comme l'utilisation de boîtes englobantes. Dans ce cas, la recherche des k couples de boîtes sécantes peut être effectuée en $o(k + n.ln^2 n)$ [58].

6.2 Intersection de faces non coplanaires

La partie la plus délicate du raffinement est l'intersection de faces quelconques. Notre manière d'aborder cette question ressemble à celle de [7]. Mais nous bénéficions d'une structure topologique précise, celle des 3-cartes dont l'intégrité est maintenue en permanence, et d'une série d'opérateurs spécifiés algébriquement qui nous permettent de simplifier et contrôler rigoureusement la définition de ce traitement. Lorsque deux faces non coplanaires s'intersectent le long d'une droite D, leur intersection peut être réduite à un problème d'interclassement de segments le long de D.

Chaque face est intersectée avec le plan de l'autre, ce qui fournit un ensemble de segments de D intérieurs à la face ou sur son bord. Deux faces s'intersectent si deux segments de leur intersection respective avec le plan de l'autre face se chevauchent. Un balayage de D est utilisé pour détecter les couples de segments se chevauchant. La configuration de ces segments est utilisée alors pour découper les deux faces.

Une description très formelle de l'intersection de faces, uniquement basée sur l'étude des configurations, a été réalisée pour l'analyse initiale du problème et correspond à la dernière phase de l'intersection. Cependant, une telle description est de peu d'utilité car elle n'explicite pas comment sont obtenus les segments pertinents de D et où sont insérés les arêtes et sommets lors de la découpe des faces. Nous présentons donc un système amélioré par raffinement vertical (au sens du génie logiciel), prenant en compte la stratégie de balayage de D.

La construction de l'ensemble de segments de D pertinents pour une face f se déroule en trois phases. La première est la classification des arêtes de f par rapport à D. Elle consiste à construire la liste des arêtes de f coupant la droite D ou lui étant incidentes. Pour chaque arête, des informations concernant la nature de cette intersection sont collectées.

Dans la seconde phase, cette liste est ordonnée, pour construire la liste des états de ces arêtes. L'état d'une arête indique la position relative (au-dessus, au-dessous, à droite ou à gauche) de l'intérieur de la face par rapport à cette arête. La dernière phase consiste à regrouper ces arêtes par couples, en ne gardant que les couples qui définissent un segment de D à l'intérieur ou sur le bord de la face.

Les trois phases sont détaillées et spécifiées dans les prochaines sous-sections où nous utiliserons les notations suivantes : x est un brin de la face (orientée) f pour laquelle on construit l'ensemble des segments. Le plan de f coupe le plan π d'une autre face, le long d'une droite D munie d'un repère $(O, \vec{\imath})$. Pour ne pas alourdir les figures, dans la suite, le plan π n'est pas représenté explicitement et seule D apparaît.

6.2.1 Obtention des informations géométriques nécessaires

Pour définir la liste des classifications des arêtes de f, on regroupe un ensemble de données géométriques obtenues en recherchant l'intersection des 1-plongements de f avec le plan π. Pour tout brin x de f, les informations suivantes, concernant le segment sur lequel il est plongé, sont recueillies :

- t_x le type du segment ;
- p_x le point d'intersection ou d'incidence du segment avec π ;
- k_x l'abscisse de ce point dans $(O, \vec{\imath})$;
- θ_x la valeur absolue, dans l'intervalle $[0, \pi]$, de l'angle entre le segment orienté de x et le vecteur $\vec{\imath}$, dans le plan de f ;
- i_x un entier déterminant la direction de l'intérieur de f dans $(O, \vec{\imath})$.

Ces informations sont obtenues par la fonction $type\_seg(s, \pi, n)$, où s est un segment, π le plan vis-à-vis duquel sont réalisées les intersections et $\vec{n}$ un vecteur normal de la face. Les types de segments possibles sont : `OUTSEG`, `CUTSEG`, `UPON`, `DOWNON`, `ONUP`, `ONDOWN`, `ON1`, `ON-1` et sont représentés dans la figure 6.2. Pour simplifier nous parlons ensuite du type d'un brin.

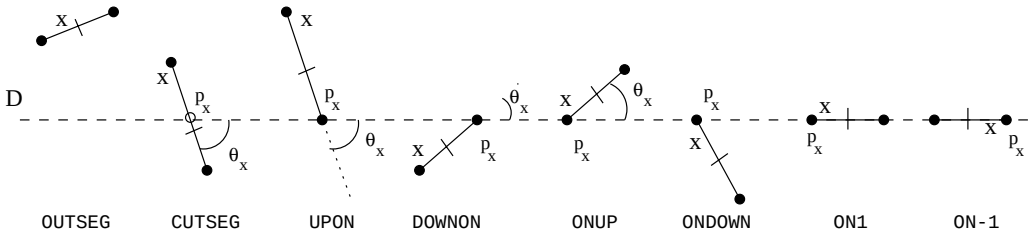


FIG. 6.2: *Type d'un segment par rapport à un plan π*

Dans cette figure, le repère $(O, \vec{\imath})$ n'est pas représenté, mais il est orienté de gauche à droite. Les segments sont représentés à plat, dans le plan de la face f. La direction de la normale

à f détermine deux demi-plans au-dessus et au-dessous de D. Les extrémités des segments sont ainsi situées au-dessus de D, au-dessous de D ou sur D. On peut ainsi différencier les segments qui coupent ou non le plan π (`OUTSEG` et `CUTSEG`), les segments incidents à π suivant leurs orientations de bas en haut ou de haut en bas (`UPON`, `DOWNON`, `ONUP` et `ONDOWN`) et les segments complètement dans π suivant leurs orientations dans le repère $(O, \vec{\imath})$ (`ON1` et `ON-1`)

Le point p_x est le point d'intersection de l'arête de x avec le plan π, pour un brin de type `CUTSEG`, l'extrémité du segment de x située sur D, dans les cas d'incidence, et le 0-plongment de x dans les deux cas d'inclusion dans π. Ce point sera éventuellement utilisé pour l'insertion d'arête ou de sommets dans f.

L'abscisse k_x et l'angle θ_x sont utilisés lors du tri le long de D. La valeur de θ_x est utilisée lorsque deux points ont la même abscisse. Comme nous voulons ordonner les segments de gauche à droite, la valeur absolue de cet angle suffit. En pratique, le cosinus de θ_x est utilisé. Il est obtenu par un produit scalaire entre le vecteur $\vec{\imath}$ et le vecteur défini par l'arête orientée de x.

L'entier i_x, pouvant prendre les valeurs -1, 0 ou 1, permet de repérer la direction $\vec{d}$ de l'intérieur de la face qui est, rappelons-le, situé à droite de l'arête. Le vecteur direction $\vec{d}$ est obtenu par le produit vectoriel entre le vecteur défini par l'arête orientée de x et la normale $\vec{n}$ de la face f. Alors, i_x est le signe du produit scalaire entre $\vec{d}$ et $\vec{\imath}$. Cet entier vaut 1 si la direction $\vec{d}$ est dans le sens de $\vec{\imath}$, c'est-à-dire si l'intérieur de la face est situé après p_x dans le repère, -1 si elle est de sens opposé et 0 si le brin est de type `ON` ou `ON-1`. Nous appelons cet entier l'*indicateur* (d'intérieur) d'une arête.

Ces informations géométriques sont recueillies pour chaque brin de la face f et placées dans une liste par la fonction $info\_face(C, x, \pi, n)$ où x est le brin de f fourni par le système de réécriture, π le plan pour lequel sont calculées les intersections et $\vec{n}$ un vecteur normal à f. Cette fonction est spécifiée dans la table 6.2, qui décrit simplement un parcours de la face de x par la fonction φ_1 et l'appel de la fonction $type\_seg(s, \pi, n)$ pour chaque arête. Notons que le brin est ajouté à l'ensemble des informations car nous en aurons besoin pour savoir où, dans la carte, les insertions d'arêtes ou de sommets auront lieu.

TAB. 6.2: *Spécification de la fonction* $info\_face(C, x)$

Spéc *INFO-FACE* **étend** *3-CARTE-GEO* **avec**

Opérateurs

$info\_face : 3Carte\ Brin\ Plan\ Vect \longrightarrow List[(Brin\ Type\ Point\ Float\ Float\ Int)]$

Axiomes $(C : 3Carte\ ;\ x, y : Brin\ ;\ \pi : Plan\ ;\ n : Vect\ ;$
$t : Type\ ;\ p : Point\ ;\ k, \theta : Float\ ;\ i : Int)$

$info\_face(C, x, \pi, n) = info\_face'(C, x, x, \pi, n)$

$info\_face'(C, y, x, \pi, n) =$ **si** $(\varphi_1(C, y) == x)$ **alors** $[info_y]$
sinon $[info_y | info\_face'(C, \varphi_1(C, y), x, \pi, n)]$

avec $info_y = (y, t, p, k, \theta, i)$

et $(t, p, k, \theta, i) = type\_seg(gem1(C, y), \pi, n)$

Fin

La sorte $List[(Brin\ Type\ Point\ Float\ Float\ Int)]$ représente les listes de sextuplets typés et décrit la sorte de chacun des éléments d'un tel sextuplet. Comme précédemment, nous

aurions pu construire une sorte spéciale pour ces listes avec les générateurs et sélecteurs correspondants. Pour simplifier l'écriture, nous admettons encore une fois l'utilisation de produits cartésiens de sortes et employons une forme paramétrée pour la sorte $List[Elt]$, où Elt décrit la sorte des éléments de la liste.

6.2.2 Classification des brins coupant D ou lui étant incidents

La liste d'informations géométriques obtenue précédemment est utilisée pour construire une liste dans laquelle le type d'un brin est remplacé par la *classe* de son arête ou de son sommet. Cette notion de classe est utilisée pour distinguer les différents types d'intersections avec la droite D et pour collecter des informations plus complètes et plus fiables que les simples données numériques. Dans la suite, seules les arêtes coupant D ou ayant une extrémité incidente à D vont nous intéresser. Les brins de type `OUTSEG` sont donc ôtés de la liste.

Les brins de type `CUTSEG` correspondent à des arêtes qui coupent D. Les informations recueillies pour ces brins sont suffisantes pour décrire l'intersection et sont conservées telles quelles. Les autres brins correspondent à des arêtes incidentes à D. Celles-ci sont regroupées deux par deux, pour former des couples d'arêtes ayant un sommet commun incident à D et partageant certaines informations géométriques.

Ainsi, le brin correspondant à la première arête, par rapport à φ_1, du couple est ôté de la liste. Le second brin est conservé et porte les informations géométriques retenues. Notons que ce brin est toujours celui dont le sommet est sur D. La classe de ce brin permet de distinguer les différents placements possibles des deux arêtes, autour du sommet commun.

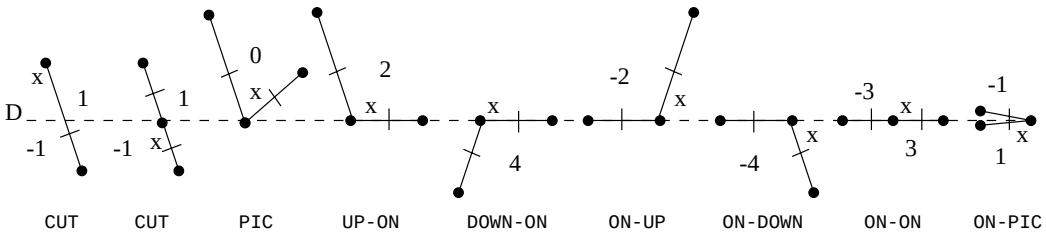


FIG. 6.3: *Les différentes classes d'arêtes et leurs codages*

Les classes d'arêtes sont `CUT`, `PIC`, `UP-ON`, `DOWN-ON`, `ON-UP`, `ON-DOWN`, `ON-ON`, `ON-PIC`. La première, `CUT`, correspond soit à une arête coupant π, soit à deux arêtes traversant le plan. Les autres classes permettent de distinguer les configurations de sommets correspondantes, comme cela est schématisé dans la figure 6.3.

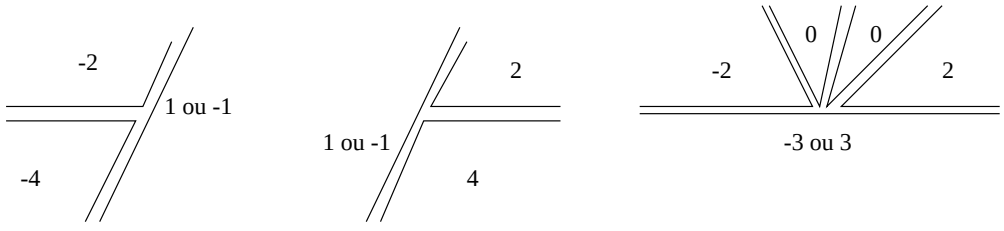


FIG. 6.4: *L'ordonnancement des codes autour d'un sommet multiple*

Pour permettre, durant le tri, la comparaison de deux classes, nous les codons par un entier compris entre -4 et 4. Notons ici que certaines classes ont deux codages différents correspondant aux deux orientations possibles pour les arêtes, soit de bas en haut ou de haut en bas pour la classe `CUT`, soit de droite à gauche ou de gauche à droite pour les classes `ON-ON` et `ON-PIC`.

Les valeurs du code sont affichées à coté de la classe correspondante dans la figure 6.3. Dans le cas où deux valeurs sont possibles, elles sont indiquées de part et d'autre des arêtes. Ce codage permet d'ordonner des arêtes incidentes à un même sommet pour les parcourir de gauche à droite et de bas en haut, comme cela est schématisé dans la figure 6.4. L'intérêt du codage est détaillé, plus loin, en même temps que la présentation du tri.

Notons que pour une arête de classe `CUT`, son code est égal à son indicateur, soit -1 ou 1, et indique si l'intérieur de la face est respectivement avant ou après l'arête dans le repère $(O, \vec{\imath})$. Pour les autres arêtes, le codage dépend de la configuration du sommet correspondant, mais est totalement indépendante de son indicateur. Par exemple, pour une arête de classe `UP-ON` codée 2, l'intérieur de la face peut être dans le quart de plan situé au-dessus et à droite des deux arêtes couplées. Dans ce cas l'indicateur d'intérieur vaut 1. Ou l'intérieur peut se situer dans les trois quarts de plan restant, auquel cas l'indicateur vaut -1.

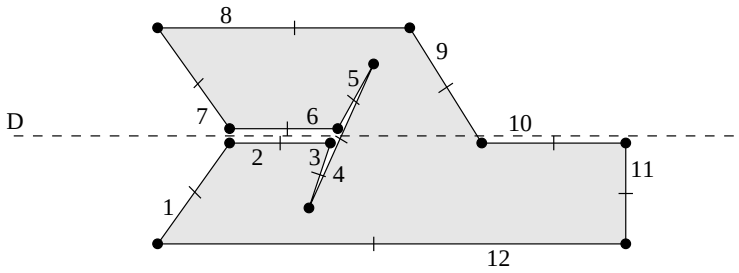


FIG. 6.5: *Exemple de classification des brins des faces*

Exemple 6.2.1 *La figure 6.5 montre une face et la droite d'intersection D. Nous n'avons numéroté qu'un brin sur deux car, rappelons le, seuls les brins d'une des deux faces orientées nous intéressent. Les informations géométriques sont recueillies pour chacun des brins de la face. Pour comprendre le mécanisme de classification, nous partons de la liste obtenue :* $[(1, \texttt{DOWNON}, 1), (2, \texttt{ON1}, 0), (3, \texttt{ONDOWN}, -1), (4, \texttt{CUT}, 1), (5, \texttt{UPON}, -1), (6, \texttt{ON-1}, 0), (7, \texttt{ONUP}, 1), (8, \texttt{OUTSEG}, 0), (9, \texttt{UPON}, -1), (10, \texttt{ON1}, 0), (11, \texttt{ONDOWN}, -1), (12, \texttt{OUTSEG}, 0)]$. *Nous n'avons écrit ici que les numéros des brins, les types des segments et la valeur de l'indicateur d'intérieur, les autres informations étant sans importance pour comprendre le mécanisme.*

Après la classification on obtient la liste $[(2, \texttt{DOWN-ON}, 4, 1), (3, \texttt{ON-DOWN}, -4, -1), (4, \texttt{CUT}, 1, 1), (6, \texttt{ON-UP}, -2, -1), (7, \texttt{UP-ON}, 2, 1), (10, \texttt{UP-ON}, 2, -1), (11, \texttt{ON-DOWN}, -4, -1)]$, *où apparaissent les classes des brins retenus, leurs codages et les indicateurs. Seuls les brins dont l'arête coupe D ou dont le sommet est sur D sont conservés. Les configurations des sommets sont obtenues pour des couples d'arêtes, mais sont portées par un seul brin. Ainsi le brin 1 est supprimé de la liste et son type est regroupé avec le type du brin 2 pour former la classe du brin 2. Les brins 2 et 3 sont conservés et leurs types sont regroupés pour former la classe du brin 3.*

Notons encore l'indépendance entre le codage et l'indicateur d'intérieur. Par exemple, les brins 7 et 10 ont la même classe, `UP-ON` *codée 2, mais des indicateurs différents. Pour le brin*

7, l'indicateur est 1, ce qui signifie que l'intérieur est après l'arête de 7, et donc au-dessus de l'arête du brin 6. A l'opposé, l'indicateur du brin 10 est -1, car l'intérieur est situé avant l'arête du brin 9, et donc sous l'arête de 10. □

La classification des arêtes correspond à la construction d'une liste formée de 7-uplets de la forme $(x, c, p, k, \theta, i, o)$, où c est la classe de l'arête de x et o son code. Les autres membres sont les mêmes qu'auparavant, à savoir le point d'intersection, son abscisse, l'angle d'incidence et l'indicateur d'intérieur.

Dans la suite, cette classification n'est pas décrite, comme précédemment par une fonction. En effet, cette transformation est basée sur un parcours de la liste et, pour chacun de ses éléments, sur une étude portant sur 18 cas qui conduirait à une expression fonctionnelle lourde et peu lisible. Nous avons donc choisi de présenté cette transformation par un système de réécriture décrivant juste les changements élément par élément.

Dans une règle du système de la table 6.3, le numérateur et le dénominateur contiennent deux listes séparées par un point virgule. La première, nommée L_i, est la liste d'informations géométriques et la seconde, nommée L_c, la liste obtenue après classification. Pour construire L_c, on examine les types de brins en tête de L_i et on applique la règle correspondant à cette configuration. Chaque règle supprime les informations utilisées en tête de L_i et ajoute les informations calculées en queue de L_c.

Comme d'habitude, deux règles, R_{init} et R_{fin}, décrivent l'initialisation et la fin du processus. La liste L_c est initialisée avec la liste vide et la transformation se termine lorsque L_i est vide. Les règles du système décrivent tous les cas possibles de combinaison de types. Notons que la règle R_1 correspond à l'élimination des arêtes qui n'intersectent pas le plan π, c'est-à-dire des brins de type `OUTSEG`. Les règles R_2 et R_4 décrivent les cas des arêtes de classe `CUT` dont le code est simplement l'indicateur.

Les autres règles correspondent aux configurations présentées dans la figure 6.3 et font toujours intervenir deux arêtes. Elles fournissent la valeur du code en fonction des cas et des orientations. Remarquons simplement que les informations géométriques sont recopiées à partir de la seconde arête de la liste qui est celle qui porte les informations du sommet incident au plan. Il existe, bien sûr, des exceptions que l'on peut remarquer pour les règles R_{11}, R_{12}, R_{15} et R_{16}.

Ces règles correspondent aux cas où la seconde arête est de type `ON` ou `ON-1`. Son identificateur est alors toujours 0 et n'apporte aucune information. L'identificateur conservé est ainsi, pour ces quatre cas, celui de la première arête. Le même principe est valable pour l'angle d'incidence. Ainsi, par exemple, pour une arête de classe `UP-ON`, le point et l'abscisse conservés sont ceux du brin x. Mais, l'angle et l'indicateur qui sont pertinents sont ceux de l'autre arête incidente au sommet de x. Dans ce cas, l'indicateur permet de savoir si l'intérieur est au-dessus ou au-dessous de l'arête de x. Il est au-dessus de x si l'intérieur est à droite de la seconde arête, c'est-à-dire si son indicateur vaut 1.

La dernière règle n'est utilisée qu'au plus une fois, au début de la réécriture, c'est-à-dire lorsque L_c est vide. Elle correspond au fait que le premier brin choisi pour le parcours de la face peut se situer n'importe où. Or, excepté pour les deux premières règles, la transformation utilise toujours un couple d'arêtes dont la première arrive sur le plan et la seconde s'écarte du plan. Si la liste d'information commence par un brin dont l'arête s'écarte du plan, alors la seconde arête nécessaire pour compléter le couple se trouve en fin de liste.

Dans ce cas, c'est-à-dire lorsque les autres règles ne peuvent être déclenchées, la règle R_{loop} effectue une permutation circulaire de la liste en envoyant son premier membre à la fin de celle-ci. Notons qu'excepté ce cas le choix du brin de départ est totalement indifférent et donc que toute permutation circulaire de la liste conduit au même résultat permuté.

Dans les règles de la table 6.3 et pour les alléger, nous avons omis les points et les abscisses. Ces deux informations sont recopiées exactement de la même façon que les brins correspondants. Cela est naturel, car le brin conservé est toujours celui dont l'arête coupe le plan π ou lui est incident. Le point et l'abscisse dont nous aurons besoin sont donc toujours ceux de ce brin. Ainsi, dans les règles, là où on trouve un brin x, il faut voir, à coté, le point p_x et son abscisse k_x. Enfin, rappelons que l'ajout d'un élément x en queue d'une liste L est noté $L :: x$.

TAB. 6.3: *Système de réécriture pour la classification des arêtes*

$$R_{init} : \frac{L_i}{L_i \ ; \ []} \qquad R_{fin} : \frac{[] \ ; \ L_c}{L_c} \qquad R_{loop} : \frac{[(x,t,\theta,i),L_i] \ ; \ []}{L_i :: (x,t,p,\theta,i) \ ; \ []} \ \textbf{si} \ \neg(R_1 \dots R_{18})$$

$$R_1 : \frac{[(x,\texttt{OUTSEG},\theta,i),L_i] \ ; \ L_c}{L_i \ ; \ L_c} \qquad R_2 : \frac{[(x,\texttt{CUTSEG},\theta,i),L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x,\texttt{CUT},\theta,i,i)}$$

$$R_3 : \frac{[(x,\texttt{UPON},\theta,i),(x',\texttt{ONUP},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{PIC},\theta',i',0)} \qquad R_4 : \frac{[(x,\texttt{DOWNON},\theta,i),(x',\texttt{ONUP},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{CUT},\theta',i',i')}$$

$$R_5 : \frac{[(x,\texttt{ON1},\theta,i),(x',\texttt{ONUP},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{ON-UP},\theta',i',-2)} \qquad R_6 : \frac{[(x,\texttt{ON-1},\theta,i),(x',\texttt{ONUP},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{UP-ON},\theta',i',2)}$$

$$R_7 : \frac{[(x,\texttt{UPON},\theta,i),(x',\texttt{ONDOWN},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{CUT},\theta',i',i')} \qquad R_8 : \frac{[(x,\texttt{DOWNON},\theta,i),(x',\texttt{ONDOWN},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{PIC},\theta',i',0)}$$

$$R_9 : \frac{[(x,\texttt{ON1},\theta,i),(x',\texttt{ONDOWN},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{ON-DOWN},\theta',i',-4)} \qquad R_{10} : \frac{[(x,\texttt{ON-1},\theta,i),(x',\texttt{ONDOWN},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{DOWN-ON},\theta',i',4)}$$

$$R_{11} : \frac{[(x,\texttt{UPON},\theta,i),(x',\texttt{ON1},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{UP-ON},\theta,i,2)} \qquad R_{12} : \frac{[(x,\texttt{DOWNON},\theta,i),(x',\texttt{ON1},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{DOWN-ON},\theta,i,4)}$$

$$R_{13} : \frac{[(x,\texttt{ON1},\theta,i),(x',\texttt{ON1},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{ON-ON},\theta',i',3)} \qquad R_{14} : \frac{[(x,\texttt{ON-1},\theta,i),(x',\texttt{ON1},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{ON-PIC},\theta',i',1)}$$

$$R_{15} : \frac{[(x,\texttt{UPON},\theta,i),(x',\texttt{ON-1},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{ON-UP},\theta,i,-2)} \qquad R_{16} : \frac{[(x,\texttt{DOWNON},\theta,i),(x',\texttt{ON-1},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{ON-DOWN},\theta,i,-4)}$$

$$R_{17} : \frac{[(x,\texttt{ON1},\theta,i),(x',\texttt{ON-1},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{ON-PIC},\theta',i',-1)} \qquad R_{18} : \frac{[(x,\texttt{ON-1},\theta,i),(x',\texttt{ON-1},\theta',i') \,|\, L_i] \ ; \ L_c}{L_i \ ; \ L_c :: (x',\texttt{ON-ON},\theta',i',-3)}$$

Durant ces deux premières phases, les informations concernant l'intersection de la face avec le plan π ont été collectées. Nous verrons dans la suite leur utilité, mais auparavant quelques remarques s'imposent, notamment concernant les approximations numériques et les erreurs d'arrondis.

Le seul endroit où sont effectués des calculs numériques se situe dans la fonction *type_seg*. La seconde phase, quant à elle, n'est pas affectée par ces problèmes numériques. Notons également que le formalisme utilisé est relativement simple et fournit ainsi une description claire de la transformation, suffisamment explicite pour permettre de détailler *tous* les cas possibles.

Les informations géométriques et topologiques contenues dans le code de classe et l'indicateur d'intérieur d'une arête seront utilisées ci-après pour contrôler la validité des autres résultats numériques. Nous verrons de plus que ce procédé permet de repérer précisément les conflits numériques et de les gérer au mieux. Il faut cependant garder à l'esprit que, même en cas de problèmes numériques, l'utilisation des opérateurs topologiques et géométriques spécifiés pour les cartes assure que celles-ci satisfont toujours leurs contraintes d'intégrités. En d'autres termes, la gestion des conflits numériques n'intervient que pour contrôler le plongement, c'est-à-dire la forme, du résultat.

6.2.3 Tri des arêtes le long de D

L'étape suivante consiste à trier la liste obtenue après la classification, par abscisses croissantes dans le repère $(O, \vec{\imath})$. La définition du tri revient à définir un ordre sur les 7-uplets formant cette liste. Cet ordre $<$ est spécifié dans la table 6.4. Sa définition va nous permettre de mettre en lumière l'intérêt des informations recueillies, en prenant comme exemple quelques configurations typiques des problèmes à résoudre.

TAB. 6.4: *Spécification de l'ordre sur les classifications*

Spéc *ORDRE-CLASS* **étend** *CLASS-FACE* **avec**

Opérateurs

$< : (Brin\ Type\ Point\ Float\ Float\ Int)\ (Brin\ Type\ Point\ Float\ Float\ Int) \longrightarrow Bool$

Axiomes $(x, x' : Brin\ ;\ c, c' : Class\ ;\ p, p' : Point\ ;\ k, k', \theta, \theta' : Float\ ;\ i, i', o, o' : Int)$

$(x, c, p, k, \theta, i, o) < (x', c', p', k', \theta', i', o') =$ **si** $k \neq k'$
alors $k < k'$
sinon si $c \neq PIC \vee c' \neq PIC$
alors si $o \neq o'$
alors $o < o'$
sinon $i < i'$
sinon si $\theta \neq \theta'$
alors $\theta > \theta'$
sinon $i < i'$

Fin

La spécification ORDRE-CLASS étend une spécification CLASS-FACE, non détaillée, qui elle-même étend la spécification INFO-FACE et contient le système de réécriture décrit dans la section précédente. Elle contient également la définition de la sorte $Class$ et les constantes correspondantes, ainsi que la définition des sortes pour les listes de 7-uplets et leurs éléments.

Pour appréhender les possibilités offertes par cet ordre, il faut garder à l'esprit que le but recherché est l'obtention d'un ensemble de couples d'arêtes formant des segments disjoints et se suivant le long de la droite D. Plus important encore, ces couples d'arêtes doivent autoriser les insertions d'arêtes et de sommets nécessaires à une découpe correcte de la face.

Examinons maintenant, en détail, la définition de l'ordre, en reprenant les notations de la spécification. Si les abscisses k et k' sont distinctes alors les points d'intersection ou d'incidence des deux arêtes avec D sont distincts et l'ordre des arêtes correspond à l'ordre de ces abscisses.

Dans le cas contraire, les arêtes touchent le plan π en des points égaux de D et différents cas apparaissent.

Si l'une des deux arêtes ne forme pas un pic, alors les codes des classes des arêtes, s'ils sont distincts, sont comparés. Comme nous l'avons vu, les entiers choisis pour coder les classes ont été choisis pour que cette comparaison permette de trier les arêtes de gauche à droite et de bas en haut. Si les deux codes sont identiques, alors leurs indicateurs d'intérieur sont comparés. La figure 6.6 schématise l'ordre ainsi obtenu, pour trois exemples typiques de faces.

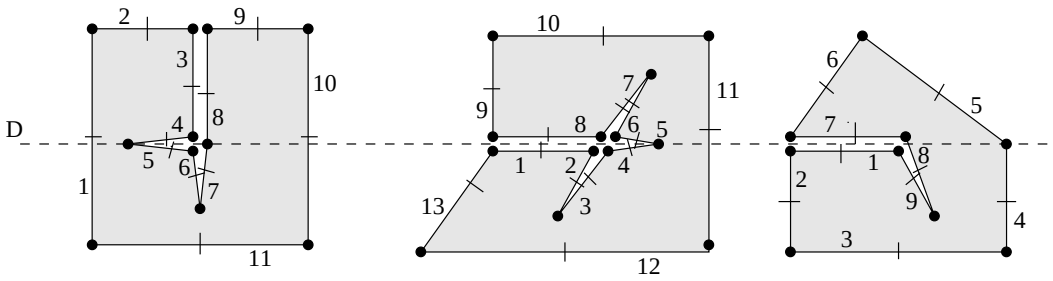


FIG. 6.6: *Exemple d'ordre pour différentes configurations de sommets*

Exemple 6.2.2 *La face de gauche de la figure 6.6 admet, après le tri, la liste de classes* $[(1, \texttt{CUT}, 1), (5, \texttt{ON-PIC}, 1), (6, \texttt{ON-DOWN}, -4), (4, \texttt{ON-UP}, -2), (8, \texttt{CUT}, 1), (10, \texttt{CUT}, -1)]$. *La liste est limitée aux informations pertinentes, c'est-à-dire aux triplets* (x, c, o), *où* x *est le numéro du brin apparaissant dans le schéma, et,* c *et* o *sont respectivement la classe de* x *et son code.*

Les brins 4, 6 et 8 appartiennent à un même sommet, ils sont donc ordonnés grâ ce à leurs codes. Ainsi, les brins 8 et 10 se succèdent dans la liste, et, si une arête doit être insérée à cet endroit, elle le sera correctement. Si, par exemple le brin 4 avait été classé après le brin 8, une insertion entre les brins 4 et 10 conduirait à une face non simple, au sens des polygones. En effet une arête placée entre 4 et 10 couperait les arêtes des brins 7 et 8.

Enfin, notons que pour les brins 1 et 10, le code indique bien que l'intérieur de la face est après l'arête de 1 et avant l'arête de 10, si le repère de D *est orienté de gauche à droite.* □

Exemple 6.2.3 *La face du centre de la figure 6.6 montre une variante du cas précédent. Après le tri la liste est* $[(9, \texttt{UP-ON}, 2), (1, \texttt{DOWN-ON}, 4), (2, \texttt{ON-DOWN}, -4), (8, \texttt{ON-UP}, -2), (6, \texttt{UP-ON}, 2), (4, \texttt{DOWN-ON}, 4), (5, \texttt{ON-PIC}, -1), (11, \texttt{CUT}, -1)]$. *La liste est ici aussi limitée aux informations pertinentes.*

Il faut noter que les segments existants entre les brins 9 et 8 et les brins 1 et 2 sont égaux. Cependant, le codage permet que les intervalles correspondant de la liste ne se croisent pas, c'est-à-dire précisément que le segment défini par les brins 1 et 2 soit, dans la liste, inclus dans celui qui est défini par 9 et 8.

De même, les brins 2, 8, 4 et 6, qui appartiennent au même sommet, sont correctement séparés par le tri. Ces deux remarques sont importantes pour la compréhension des traitements ultérieurs. □

Exemple 6.2.4 *La face de droite de la figure 6.6 montre les limitations du codage et l'intérêt d'utiliser l'indicateur d'intérieur en cas d'égalité. Après le tri la liste est* $[(7, \texttt{UP-ON}, 2, 1), (2, \texttt{DOWN-ON}, 4, 1), (1, \texttt{ON-DOWN}, -4, -1), (8, \texttt{ON-DOWM}, -4, 1), (5, \texttt{CUT}, -1, -1)]$.

Pour ce dernier cas, nous avons ajouté l'indicateur. La liste est ainsi formée de quadruplets de la forme (x, c, o, i), où i est l'indicateur de x, les autres données restant les mêmes.

Les codes des brins 1 et 8 sont les mêmes et c'est donc leurs indicateurs qui servent pour le tri. D'une part, on obtient la même propriété de bonne inclusion pour les segments formés des brins 7 et 8 et des brins 2 et 1. D'autre part, si une arête doit être insérée entre le sommet des brins 8 et 1 et le sommet de 5, le fait de savoir que l'intérieur est après 8 et avant 1 permet de réaliser l'insertion entre 8 et 5, et non pas entre 1 et 5. Ce dernier cas entraînant la création d'une face non simple. □

Si les deux arêtes sont des pics, alors leurs angles d'incidence sont utilisés pour les distinguer. Et, si ces angles sont égaux, c'est l'indicateur d'intérieur qui est utilisé. Le fait que le code d'un pic soit 0 permet de traiter le cas d'un pic isolé comme celui des autres configurations de sommets. Les codes sont en effet négatifs pour les configurations devant être placées à gauche et positifs pour celles devant l'être à droite. Un pic est donc toujours placé correctement au milieu. L'angle d'incidence n'a donc d'utilité que dans la comparaison de deux pics consécutifs.

6.2.4 Liste d'états des arêtes

Avant l'obtention de la liste des segments de D qui nous intéresse, il reste une étape qui est la construction de la liste des états des arêtes de la liste de classifications. L'état d'une arête indique la position de l'intérieur de la face vis-à-vis de celle-ci. Cette information existe déjà partiellement dans l'indicateur d'intérieur, mais n'est pas suffisante pour traiter les cas de segments superposés.

De plus, et c'est là que réside la différence principale avec les algorithmes classiques, cette phase va nous permettre une confrontation entre les données géométriques et les données topologiques obtenues précédemment. En effet, le tri décrit dans la section précédente est très sensible aux approximations numériques, essentiellement en ce qui concerne les abscisses calculées et accessoirement pour les angles. Nous verrons plus loin comment cette phase permet de déceler et de corriger les ambiguïtés produites par le tri.

Les états et leur méthode de construction sont une extension du principe, bien connu, qui consiste à compter le nombre de fois qu'une demi-droite coupe le bord d'une face, pour savoir si l'extrémité de celle-ci est à l'intérieur ou à l'extérieur de la face. Dans notre cas, les différents états possibles sont : `OUT, IN, ON(0,0), ON(1,0), ON(0,1), ON(1,1)`.

Les états `IN` et `OUT` indiquent que la partie de D qui suit l'arête dans le repère $(O, \vec{\imath})$, ou le sommet commun au couple d'arêtes, est respectivement à l'intérieur ou à l'extérieur de la face. Les états de la forme `ON(DESSUS, DESSOUS)` indiquent, pour une arête qui est incluse dans D, si la zone au-dessus de l'arête est à l'intérieur de la face (`DESSUS = 1`) ou à l'extérieur de la face (`DESSUS = 0`) et, de même si la zone sous l'arête est à l'intérieur de la face (`DESSOUS = 1`) ou non (`DESSOUS = 0`).

Le passage de la liste ordonnée des classifications à la liste d'états s'effectue en examinant, au cas par cas, l'état avant l'arête et la classe de l'arête. Ainsi, cette transformation est réalisée par un parcours de la liste, en prenant comme état initial `OUT`. Comme précédemment nous avons choisi de décrire ce parcours et les cas possibles par un système de réécriture, donné dans la table 6.5.

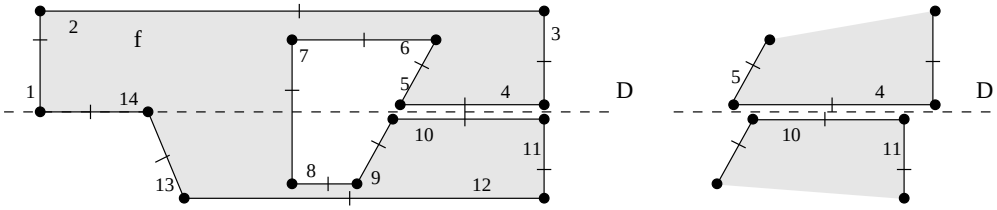


FIG. 6.7: *Exemple d'intersection d'une face avec un plan*

Exemple 6.2.5 *Dans la figure 6.7, la liste des états des arêtes, ou des sommets simples formés par un couple d'arêtes, est la suivante :* $[(1, \texttt{ON}(1,0)), (14, \texttt{IN}), (7, \texttt{OUT}), (5, \texttt{ON}(1,0)), (10, \texttt{ON}(1,1)), (11, \texttt{ON}(1,0)), (4, \texttt{OUT})]$. *Examinons, sur trois exemples, l'information contenue dans cette liste. Le premier segment de D, d'état* `ON(1,0)`*, situé entre les brins 1 et 14, est sur le bord de la face et l'intérieur de celle-ci est situé au-dessus du segment. Le deuxième segment de D, d'état* `IN`*, situé entre les brins 14 et 7, est complètement à l'intérieur de la face. Le dernier segment de D, d'état* `ON(1,1)`*, compris entre 10 et 11, est sur le bord de la face et l'intérieur de celle-ci est à la fois au-dessus et au-dessous de ce segment.*

Notons ici que les différents types d'états `ON` *sont nécessaires. Par exemple si la seule information portée par le brin 1 était* `ON`*, nous ne pourrions pas savoir si le segment situé après le brin 14 est à l'intérieur ou l'extérieur de la face.*

Notons encore que les états des brins 5 et 4 peuvent paraître incorrects à première vue. Ce n'est pas le cas, bien sûr, et cette impression est due au balayage de la liste. Pour avoir une idée plus intuitive de cette situation, on peut imaginer que l'ordre défini sur D produit un léger décalage du brin 5 vers la gauche et du brin 10 vers la droite, de même pour les brins 11 et 4, comme cela est schématisé dans la sous-figure de droite. □

Pour simplifier la lecture de ces différents cas, nous n'avons pas reproduit toutes les informations géométriques contenues dans les listes. En effet, pour cette transformation, la seule information utilisée est la classe d'un brin et le résultat qui nous intéresse est l'état calculé. Ainsi, pour chaque règle du système, le numérateur représente l'état courant et la classe de l'arête traitée, alors que le dénominateur représente l'état courant après le traitement de cette arête.

En fait, les différents états `ON(I, J)` décrits précédemment ne sont utilises que pendant le balayage. Ainsi, dans la liste finale des états, tous les états `ON(I, J)` sont remplacés simplement par `ON`.

La règle R_{init} exprime juste le fait que l'état courant est initialisé à `OUT`, quelle que soit la liste à transformer. La règle R_{fin} exprime, de même, que la transformation est terminée lorsque la liste est vide et qu'alors l'état courant doit être `OUT`, puisque la dernière arête fait forcément partie du bord extérieur de la face.

Les autres règles sont regroupées quatre par quatre. Ainsi chaque ligne indique les changements d'états pour une classe donnée. Le mécanisme de parcours et de construction des listes est le même que dans le système précédent. La classe du premier élément de la liste et l'état courant sont utilisés pour savoir quelle est la règle à appliquer. Puis, le premier élément est enlevé de la liste ordonnée et le résultat est ajouté à la fin de la liste d'états. Ceci n'est plus détaillé ici pour ne pas encombrer un système déjà complexe.

Avant de poursuivre, quelques remarques à propos des erreurs dues aux approximations nu-

TAB. 6.5: *Système de réécriture pour le calcul des états des arêtes*

	$R_{init} : \frac{L}{\texttt{OUT};L}$	$R_{fin} : \frac{\texttt{OUT};[\,]}{[\,]}$	
$R_1 : \frac{\texttt{OUT};\texttt{CUT}}{\texttt{IN}}$	$R_2 : \frac{\texttt{IN};\texttt{CUT}}{\texttt{OUT}}$	$R_3 : \frac{\texttt{IN};\texttt{PIC}}{\texttt{IN}}$	$R_4 : \frac{\texttt{OUT};\texttt{PIC}}{\texttt{OUT}}$
$R_5 : \frac{\texttt{ON(0,0)};\texttt{ON-ON}}{\texttt{ON(0,0)}}$	$R_6 : \frac{\texttt{ON(0,1)};\texttt{ON-ON}}{\texttt{ON(0,1)}}$	$R_7 : \frac{\texttt{ON(1,0)};\texttt{ON-ON}}{\texttt{ON(1,0)}}$	$R_8 : \frac{\texttt{ON(1,1)};\texttt{ON-ON}}{\texttt{ON(1,1)}}$
$R_9 : \frac{\texttt{OUT};\texttt{UP-ON}}{\texttt{ON(1,0)}}$	$R_{10} : \frac{\texttt{IN};\texttt{UP-ON}}{\texttt{ON(0,1)}}$	$R_{11} : \frac{\texttt{ON(0,1)};\texttt{UP-ON}}{\texttt{ON(1,1)}}$	$R_{12} : \frac{\texttt{ON(1,0)};\texttt{UP-ON}}{\texttt{ON(0,0)}}$
$R_{13} : \frac{\texttt{OUT};\texttt{DOWN-ON}}{\texttt{ON(0,1)}}$	$R_{14} : \frac{\texttt{IN};\texttt{DOWN-ON}}{\texttt{ON(1,0)}}$	$R_{15} : \frac{\texttt{ON(1,0)};\texttt{DOWN-ON}}{\texttt{ON(1,1)}}$	$R_{16} : \frac{\texttt{ON(0,1)};\texttt{DOWN-ON}}{\texttt{ON(0,0)}}$
$R_{17} : \frac{\texttt{ON(1,0)};\texttt{ON-UP}}{\texttt{OUT}}$	$R_{18} : \frac{\texttt{ON(0,1)};\texttt{ON-UP}}{\texttt{IN}}$	$R_{19} : \frac{\texttt{ON(1,1)};\texttt{ON-UP}}{\texttt{ON(0,1)}}$	$R_{20} : \frac{\texttt{ON(0,0)};\texttt{ON-UP}}{\texttt{ON(1,0)}}$
$R_{21} : \frac{\texttt{ON(0,1)};\texttt{ON-DOWN}}{\texttt{OUT}}$	$R_{22} : \frac{\texttt{ON(1,0)};\texttt{ON-DOWN}}{\texttt{IN}}$	$R_{23} : \frac{\texttt{ON(1,1)};\texttt{ON-DOWN}}{\texttt{ON(1,0)}}$	$R_{24} : \frac{\texttt{ON(0,0)};\texttt{ON-DOWN}}{\texttt{ON(0,1)}}$
$R_{25} : \frac{\texttt{ON(0,0)};\texttt{ON-PIC}}{\texttt{OUT}}$	$R_{26} : \frac{\texttt{ON(1,1)};\texttt{ON-PIC}}{\texttt{IN}}$	$R_{27} : \frac{\texttt{IN};\texttt{ON-PIC}}{\texttt{ON(1,1)}}$	$R_{28} : \frac{\texttt{OUT};\texttt{ON-PIC}}{\texttt{ON(0,0)}}$

mériques s'imposent encore une fois. Nous avons annoncé que les différentes données recueillies pouvaient être confrontées pour repérer d'éventuelles erreurs dues aux calculs numériques et pour les corriger. Deux cas se présentent en fait.

D'une part, les règles du système de la table 6.5 décrivent tous les cas possibles. Aussi, lorsque, pour un état courant donné, on arrive sur une arête dont la classe ne correspond à aucun des cas décrits, on peut immédiatement conclure qu'une interversion s'est produite dans le tri. De telles interversions peuvent avoir lieu pour des points d'intersection ou d'incidence sur D très proches les uns des autres, mais dont les abscisses ont été mal arrondies. Or, d'après l'étude des cas possibles, pour un état donné, on connaît exactement les classes possibles.

Il suffit donc de rechercher dans la suite de la liste un brin ayant la bonne classe et de permuter les deux éléments. Bien sûr, cette recherche est limitée aux éléments de la liste dont l'abscisse est égale, à un ε près, à celle de l'élément qui pose problème. Si un tel élément n'est pas trouvé dans la suite de la liste, c'est que l'erreur a eu lieu avant. Il faut donc effectuer un retour en arrière et recommencer avec les permutations possibles.

D'autre part, l'information brute contenue dans l'indicateur d'intérieur doit être cohérente avec l'état trouvé. En cas d'incohérence, le même principe d'interversions dans la liste peut être utilisé pour résoudre le problème.

Comme le traitement de ces erreurs n'est pas notre sujet principal, nous ne détaillons pas plus les traitements évoqués ci-dessus. Mais, on notera tout de même que le formalisme utilisé, qui permet de détailler explicitement tous les cas possibles, et l'emploi simultané d'informations géométriques et topologiques, conduisent à des descriptions restant simple, tout en permettant un contrôle rigoureux de la validité des résultats numériques obtenus.

6.2.5 Liste des segments de D

La dernière étape est la construction, à partir de la liste des états, d'une liste de segments de D disjoints, ordonnés et étant soit dans la face soit sur son bord. Le principe de ce dernier calcul est de prendre, dans la liste d'états, les couples d'arêtes consécutives qui nous intéressent. Il convient notamment d'éliminer les arêtes dont l'état est `OUT`, et de choisir uniquement deux arêtes en cas de superpositions.

La liste des segments est constituée d'éléments de la forme (x, p_x, y, p_y, e), où x et y sont des brins, p_x et p_y sont les points d'intersection ou d'incidence de leurs arêtes respectives avec le plan et e l'état calculé pour l'arête de x. En plus des points, nous aurons besoin, ci-après, de l'abscisse de ces points. Mais, encore une fois, pour éviter de surcharger inutilement les formules qui suivent, nous les oublions pour l'instant.

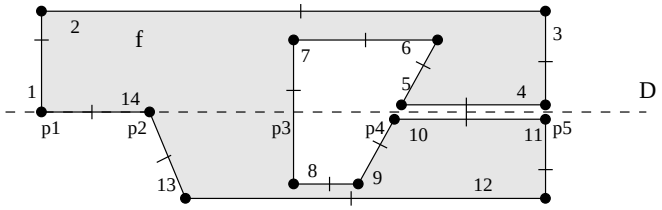


FIG. 6.8: *Exemple d'intersection d'une face avec un plan*

Exemple 6.2.6 *Reprenons l'exemple précédent dont le schéma est rappelé dans la figure 6.8. La liste des segments de la face est* $[(1, p1, 14, p2, \texttt{ON}), (14, p2, 7, p3, \texttt{IN}), (10, p4, 11, p5, \texttt{ON})$. *On peut remarquer que les brins choisis pour le dernier segment sont consécutifs dans la face et donc du même côté de D. Ainsi les brins retenus permettront des insertions d'arêtes ou de sommets correctes.* □

Nous ne donnons pas le détail de l'algorithme permettant d'obtenir la liste des segments, car il est assez simple et sans intérêt ici. Le point essentiel est que, lorsque des segments sont superposés, alors dans la liste des états il existe une suite d'arêtes dont l'état est `ON` et qui possèdent des points égaux correspondant à l'extrémité gauche du segment. Juste après cette suite, on trouve une suite d'arêtes dont les points sont également égaux et correspondent à l'extrémité droite du segment. Il suffit de choisir un brin dans chacune de ces suites tel que les deux brins soient consécutifs dans la face, c'est-à-dire que l'un soit image de l'autre par φ_1.

Les problèmes numériques ayant été résolus dans la phase précédente, ils n'interviennent plus du tout ici.

6.2.6 Découpe des faces

Les étapes précédentes doivent être réalisées pour chaque couple de faces, notées f et g, à intersecter. Leur intersection est réalisée par interclassement de leurs deux listes de segments le long de leur droite d'intersection D. Si les listes de segments de f et g ont respectivement pour premiers éléments $(x, p_x, x', p'_x, etat_x)$ et $(y, p_y, y', p'_y, etat_y)$, les opérations topologiques à effectuer pour découper f et g sont obtenues par une étude de cas sur l'ordre respectif de p_x, p'_x, p_y et p'_y sur la droite D.

Plus précisément, c'est l'abscisse calculée pour ces points qui est utilisée pour les comparaisons. Mais, pour simplifier les notations, nous oublions les abscisses et nous écrivons par la suite $p_x = p_y$, ou $p_x < p_y$, pour exprimer que, dans le repère $(O, \vec{\imath})$ de la droite D, l'abscisse du point p_x est égale à celle de p_y, ou strictement inférieure à celle de p_y. Notons que l'égalité peut être stricte ou à ε près suivant le type de géométrie utilisée, mais que dans tous les cas $p_x < p_y \Rightarrow \neg(p_x =_\varepsilon p_y)$ pour éviter les conflits entre ces relations.

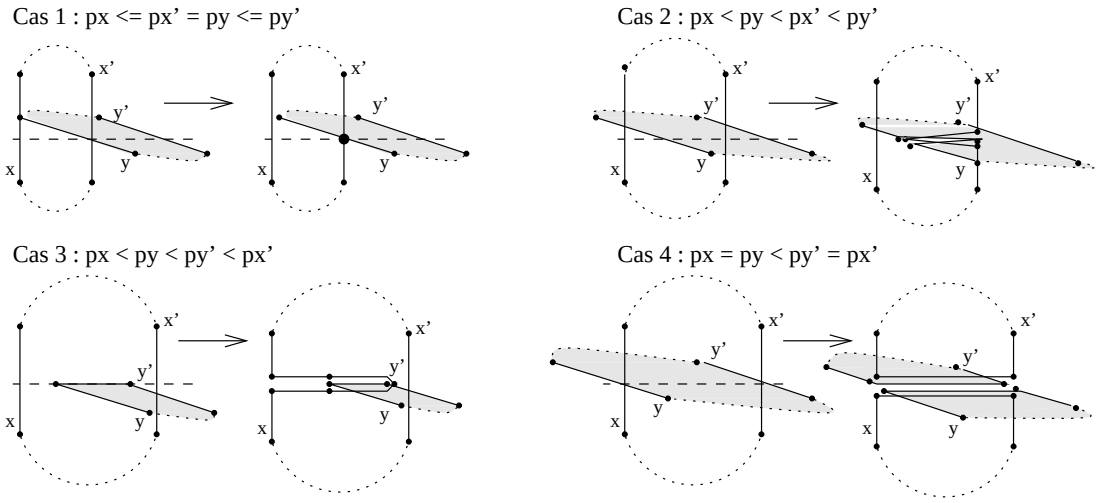


FIG. 6.9: *Représentation de 4 cas d'intersection de faces*

Pour décrire formellement les différents cas et les opérations correspondantes, nous utilisons un système de réécriture qui prend en entrée la carte C et les deux listes de segments. Ainsi, le déclenchement et l'initialisation de l'opération $cutface(C, x, y)$ sont réalisés par la règle R_{init}. La notation utilisée pour cette règle signifie que la carte C et les deux listes de segments sont transmises, en quelque sorte comme des paramètres, au système réalisant l'intersection des faces de x et y. Ce dernier système est détaillé dans la table 6.6. Les principaux cas sont dessinés dans la figure 6.9.

L'intersection des deux faces est terminée lorsque l'une des deux listes de segments est vide. La règle R_{fin} le détecte et retourne au premier système de réécriture la carte C dans laquelle l'intersection a été effectuée. Les règles $R_{symetrie1}$ et $R_{symetrie2}$ échangent les deux listes de segments pour diviser le nombre de cas grâce aux symétries. Dans les autres règles, on a donc toujours $p_x < p_y$ ou $(p_x =_\varepsilon p_y \wedge p'_x \leq p'_y)$.

La règle R_{nulle} passe à l'élément suivant dans la première liste lorsque aucune intersection n'existe entre les deux segments. La règle R_{cas1} insère un nouveau sommet dans les arêtes de x' et y qui se coupent sur D au point p'_x, égal à p_y. Le système continue avec le segment suivant de la première liste. Nous notons $[p, q]$ le segment d'extrémités p et q.

Dans le cas de la règle R_{cas2}, le segment $[p_y, p'_x]$ est dans l'intérieur des deux faces. Une nouvelle arête est donc insérée dans les deux faces, avec ses sommets respectivement plongés sur les points p'_x et p_y. Ceci est réalisé par l'opération ia.

Pour la règle R_{cas3}, $[p_y, p'_y]$ est dans l'intérieur de la face de x. Ce segment étant déjà inclus dans le bord de la face de y, rien n'est à faire pour cette face. Par contre, il est intérieur à la face de x, mais n'est pas incident à une arête de cette face. Deux arêtes doubles sont donc

TAB. 6.6: *Système de réécriture pour l'intersection de faces*

$$R_{init} : \frac{cutface(C,x,y)}{C \ ; \ liste\ (C,x) \ ; \ liste\_segment(C,y)} \qquad R_{fin} : \frac{C \ ; \ L_x \ ; \ L_y}{C} \ \textbf{si} \ L_x = vide \vee L_y = vide$$

$$R_{symetrie1} : \frac{C \ ; \ [(x,p_x,x',p'_x,etat_x)|L_x] \ ; \ [(y,p_y,y',p'_y,etat_y)|L_y]}{C \ ; \ [(y,p_y,y',p'_y,etat_y)|L_y] \ ; \ [(x,p_x,x',p'_x,etat_x)|L_x]} \ \textbf{si} \ p_y < p_x$$

$$R_{symetrie2} : \frac{C \ ; \ [(x,p_x,x',p'_x,etat_x)|L_x] \ ; \ [(y,p_y,y',p'_y,etat_y)|L_y]}{C \ ; \ [(y,p_y,y',p'_y,etat_y)|L_y] \ ; \ [(x,p_x,x',p'_x,etat_x)|L_x]} \ \textbf{si} \ p_y = p_x \wedge p'_y < p'_x$$

$$R_{nulle} : \frac{C \ ; \ [(x,p_x,x',p'_x,etat_x)|L_x] \ ; \ [(y,p_y,y',p'_y,etat_y)|L_y]}{C \ ; \ L_x \ ; \ [(y,p_y,y',p'_y,etat_y)|L_y]} \ \textbf{si} \ p_x \leq p'_x < p_y \leq p'_y$$

$$R_{cas1} : \frac{C \ ; \ [(x,p_x,x',p'_x,\texttt{IN})|L_x] \ ; \ [(y,p_y,y',p'_y,\texttt{IN})|L_y]}{is(is(C,x',p'_x),y,p_y) \ ; \ L_x \ ; \ [(y,p_y,y',p'_y,\texttt{IN})|L_y]} \ \textbf{si} \ p_x \leq p'_x < p_y \leq p'_y$$

$$R_{cas2} : \frac{C \ ; \ [(x,p_x,x',p'_x,\texttt{IN})|L_x] \ ; \ [(y,p_y,y',p'_y,\texttt{IN})|L_y]}{ia(ia(C,x',p'_x,p_y),y,p_y,p'_x) \ ; \ L_x \ ; \ [(y,p_y,y',p'_y,\texttt{IN})|L_y]} \ \textbf{si} \ p_x < p_y < p'_x < p'_y$$

$$R_{cas3} : \frac{C \ ; \ [(x,p_x,x',p'_x,\texttt{IN})|L_x] \ ; \ [(y,p_y,y',p'_y,\texttt{ON})|L_y]}{cutf(i2a(C,x,p_x,p_y,p'_y),y,y',p_y,p'_y) \ ; \ L_x \ ; \ [(y,p_y,y',p'_y,\texttt{ON})|L_y]} \ \textbf{si} \ p_x < p_y < p'_y < p'_x$$

$$R_{cas4} : \frac{C \ ; \ [(x,p_x,x',p'_x,\texttt{IN})|L_x] \ ; \ [(y,p_y,y',p'_y,\texttt{IN})|L_y]}{cutf(i2a(C,x,p_x,p_y,p'_y),y,y',p_y,p'_y) \ ; \ L_x \ ; \ L_y} \ \textbf{si} \ p_x = p_y < p'_x = p'_y$$

insérées par l'opération $i2a(C,x,p_x,p_y,p'_y)$.

Dans le dernier cas, R_{cas4}, le segment $[p_x,p'_x]$, égal au segment $[p_y,p'_y]$, sépare les deux faces. Des arêtes doubles sont donc insérées respectivement entre x et x', et entre y et y', séparant localement les deux faces. Les trois derniers cas, que nous ne détaillons pas ici, sont de même nature et utilisent les mêmes opérations d'insertion d'arêtes et de découpe de faces. Ils correspondent aux autres possibilités de classements respectifs des points p_x, p'_x, p_y et p'_y.

Chacun des 7 cas de classement se subdivise en 4 cas pour chaque valeur de $etat_x$ et $etat'_x$. En fait, si $etat_x = \texttt{IN}$, les opérations décrites sont bien effectuées. En revanche, si $etat_x = \texttt{ON}$ comme dans le cas 3, alors l'arête de x est sur la droite D, l'insertion de sommets ou d'arêtes n'est plus nécessaire et rien n'est fait. La même chose est valable pour $etat_y$ et la face de y. Au total ce sous-système comprend 32 règles $(4 + 4 * 7)$.

Intuitivement, la *terminaison* du processus ainsi décrit est due au fait que la somme des longueurs des listes diminue au moins de un à chaque étape de réécriture. Enfin, il est facile de prouver que chaque application de règle sur une 3-carte fournit une 3-carte, ce qui assure la *correction* du processus d'intersection. La question de la *confluence* du système de réécriture ne se pose pas du fait de la disjonction des conditions des règles.

6.3 Intersection de faces coplanaires

Comme nous l'avons dit auparavant, la définition de l'intersection de faces coplanaires ne demande pas de travail supplémentaire. En effet, une face d'une 3-carte est une 2-carte et

donc l'intersection de deux faces coplanaires est réalisée par le raffinement 2D d'une 2-carte contenant ces deux faces.

La seule chose à préciser est la conversion du résultat du raffinement 2D, qui est une 2-carte, en un ensemble de faces à inclure dans la 3-carte concernée. Cette conversion a déjà été définie [55] et spécifiée [14, 13] dans de nombreux travaux. Les opérations nécessaires à celle-ci, comme la duplication et le recollement, sont décrites notamment dans la thèse d'Yves Bertrand [12]. Aussi, nous contentons-nous de décrire le mécanisme de conversion sur un exemple.

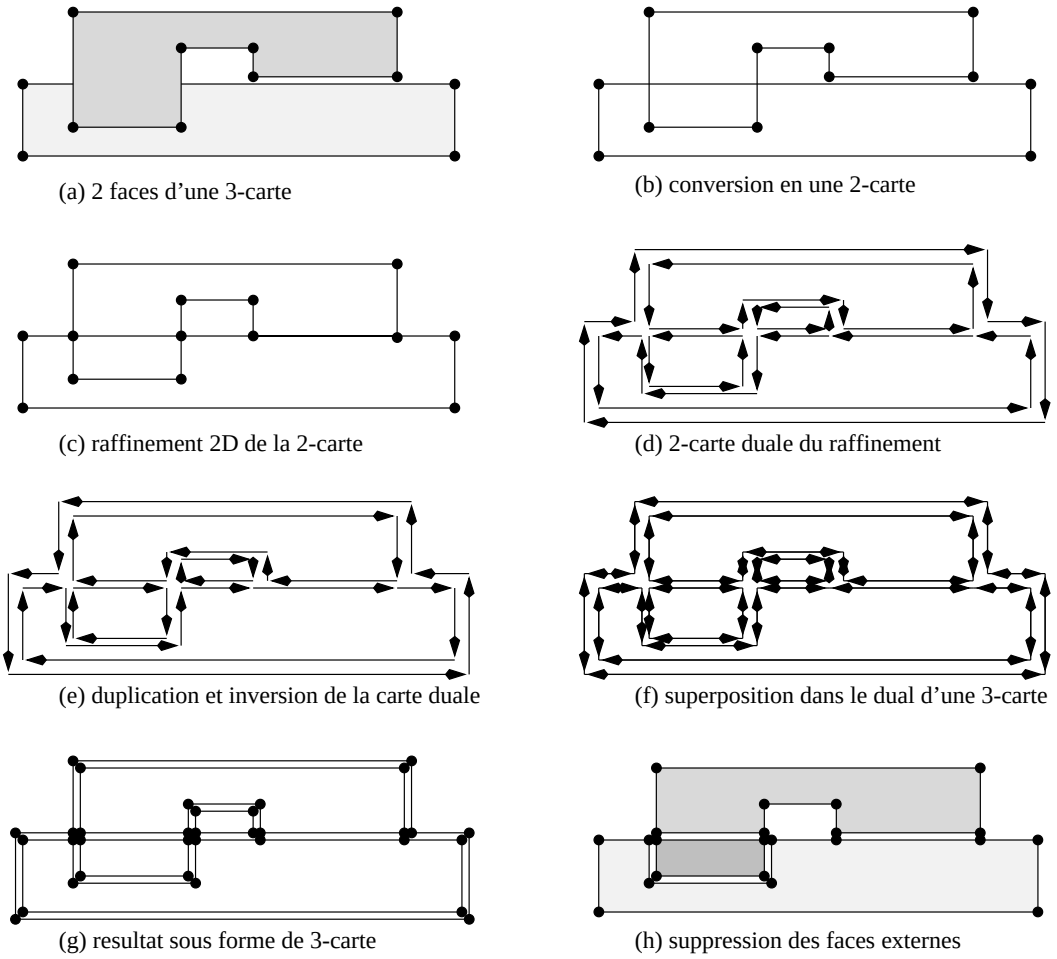


FIG. 6.10: *Conversion d'une 2-carte en un ensemble de faces d'une 3-carte*

Exemple 6.3.1 *La figure 6.10 schématise les différentes étapes de l'intersection de deux faces coplanaires, dessinées en (a). Les deux faces sont converties en une 2-carte contenant deux faces sécantes (b). Cette conversion consiste simplement à oublier les 2-liaisons existant sur les brins des deux faces. La 2-carte est alors raffinée en dimension 2 (c), avec le système du chapitre 4.*

La figure (d) représente la carte duale du résultat du raffinement. Cette 2-carte duale contient le résultat désiré. On voit clairement apparaître les faces voulues mais celles-ci ne contiennent qu'un brin par arête. Pour y remédier, la 2-carte duale est dupliquée et les liaisons

par α_1 de la nouvelle 2-carte sont inversées (e).

Les deux 2-cartes duales sont alors fusionnées en plaçant des liaisons par ϕ_3 entre chaque brin et le brin lui correspondant dans la carte dupliquée. Ces liaisons sont schématisées (f) en représentant deux brins liés par ϕ_3 par une unique double flèche. On obtient ainsi, la 3-carte duale de la 3-carte recherchée (g).

Dans la 3-carte obtenue, les diverses faces sont correctement liées par α_2, car ces liaisons correspondes aux liaisons par ϕ_2 de la 2-carte duale du raffinement. Il ne reste plus alors qu'à supprimer les faces extérieures dont on a pas besoin (h). Les faces restantes peuvent alors être insérées dans la 3-carte à laquelle appartenaient les deux faces initiales.

Notons que le passage au dual est une vue de l'esprit utile pour comprendre les mécanismes mis en jeu, mais qu'en pratique aucun calcul n'est effectué. De même un marquage des brins initiaux permet de repérer les brins des faces extérieures et de retrouver facilement les endroits où les faces obtenues doivent être 2-liées, lors de leur réinsertion dans la 3-carte initiale. □

6.4 Complétion des labels

Après son raffinement, une 3-carte ne contient plus d'intersection ni de superposition et définit ainsi, sans ambiguïté, une subdivision volumique de $\mathbb{R}^3$. Comme dans le cas de la dimension 2, une complétion des labels est alors utilisée pour permettre une évaluation des objets définis par des expressions booléennes sur les primitives de la 3-carte initiale.

La complétion des labels est réalisée exactement de la même façon qu'en dimension 2, avec juste un changement de dimension des cellules et labels concernés. Ainsi au lieu de diffuser les 2-labels au travers des faces, les arêtes jouant le rôle de filtres, ce sont les 3-labels qui sont diffusés au travers les volumes et les faces qui filtrent ces 3-labels.

Comme auparavant, quelques précautions doivent être prises lors du raffinement. Les brins qui sont créés lors de l'insertion de sommets ou d'arêtes doivent hériter des labels des brins auxquels ils sont reliés. De plus, lors des fusions de faces ou d'arêtes, les labels des brins concernés sont réunis pour ne pas perdre d'information, en suivant le même principe que pour les fusions de sommets ou d'arêtes en dimension 2.

La complétion est également réalisée par une unique règle de réécriture identique à celle que nous avons vue en dimension 2. Précisément, lorsqu'un brin x et son image z par φ_2 ont des 3-labels différents, une fonction de transmission des labels est appelée. Rappelons que deux brins liés par φ_2 appartiennent à deux faces liées par α_2 et pointant vers un même volume.

Les 3-labels des brins de x et z sont remplacés par leur union, puisque, après le raffinement, x et z appartiennent à un même volume. Les objets de l'union de ces deux 3-labels sont ensuite transmis aux volumes adjacents. Avec le même principe qu'en dimension 2, pour chaque face du volume, on enlève de cet ensemble le 2-label de ses brins, car cette face fait partie du bord des objets de ce 2-label. Les objets restants sont ajoutés aux 3-labels des brins du volume adjacent, incident à cette face.

De plus, à chaque 0 ou 1-label des sommets et arêtes d'une face du volume de x et z, on ajoute les objets de l'union des 3-labels du bord desquels la face, les sommets ou les arêtes, ne font pas partie.

Ce mécanisme est relativement simple, une fois compris le fonctionnement en dimension

2. Mais, il est très pénible à détailler, car il nécessite le parcours des faces d'un volume, ce qui n'est pas simple à décrire dans le formalisme autorisé par les spécifications algébriques que nous utilisons. Ceci est principalement dû au fait que ce mécanisme fait appel à de nombreux parcours qui sont facile à décrire dans un langage itératif, alors que les spécifications algébriques impliquent une description plus fonctionnelle inadéquate dans ce cas précis.

De même, il est quasiment impossible de faire un schéma simple en 3D représentant à la fois les brins d'une 3-carte et leurs labels à chaque dimension. C'est pourquoi la complétion des labels n'est pas détaillée plus avant.

Chapitre 7

Prototypage et implantation

Le but de notre étude est d'obtenir une implantation correcte et robuste des opérations booléennes en dimension 2 et 3. La première étape a été de définir précisément, à l'aide de cartes combinatoires plongées et labellées, les objets manipulés et ce que nous entendions par l'évaluation d'une expression booléenne sur ces objets. La seconde étape a consisté en la spécification algébrique des cartes et des opérateurs nécessaires à leur manipulation. L'évaluation d'opérations booléennes, et les stratégies de parcours qui leur sont liées, ont été décrites par des systèmes de réécriture enrichis par des structures de contrôle.

Dans les chapitres précédents, les spécifications ont été données dans leur ensemble et dans leur version finale. Bien sûr, elles n'ont pas été construites d'une pièce. En fait, il est souvent utile de faire des aller-retours entre la spécification et le programme. Cela permet de vérifier que toutes les fonctionnalités nécessaires à l'implantation existent et sont décrites. De plus, en implantant un module de spécification il arrive que l'on découvre une façon plus simple d'exprimer les axiomes qu'il contient.

Pour faciliter ces aller-retours, nous avons utilisé un *prototypage logique* de la spécification. Un tel prototype est une implantation de la spécification dans un langage de suffisamment haut niveau pour qu'on puisse éviter, durant la phase de conception, les problèmes liés à l'utilisation de structures de données comme des tableaux, des structures ou des pointeurs. Le prototypage doit pouvoir être réalisé simplement et rapidement pour que les aller-retours restent efficaces. Notons encore que les deux écritures manipulées, la spécification et le prototype, peuvent être confrontées pour débusquer un grand nombre d'erreurs de conception au niveau de la formalisation ou de l'analyse du problème.

Les problèmes liés à l'implantation sont résolus durant cette phase de prototypage où l'on peut s'assurer que les opérateurs définis sont exprimables dans un langage de programmation. Ce travail de conception effectué, le passage à une implantation efficace en C est largement facilité. Nous expliquons, dans ce chapitre, la technique de prototypage utilisée et la transcription en C du prototype obtenu.

7.1 Prototypage logique en Prolog

Nous avons choisi d'implanter le prototype en Prolog. D'une part, ce langage permet la manipulation de termes et permet ainsi une transcription quasiment immédiate de la spécifi-

cation. D'autre part, le Prolog que nous avons utilisé comprend une interface graphique qui facile grandement le travail. Les termes représentant une carte sont, en effet, très grands dès que des exemples non triviaux sont envisagés, ce qui les rend inexploitables syntaxiquement et empêche ainsi une interprétation géométrique facile.

D'autres langages de spécification et prototypage existent comme B [1], VDM [52], LPG [11], LP [46, 47] ou OBJ3 [42]. Mais, aucun d'entre eux ne contient d'interface graphique. Il est également courant de réaliser les prototypes dans des langages fonctionnels de haut niveau comme Caml qui n'a pas été retenu parce que Prolog convient mieux pour le prototypage des systèmes de réécriture.

De plus, comme nous l'avons vu, l'exécution de nos systèmes de réécriture nécessite la possibilité d'effectuer de nombreuses permutations dans les termes. Les langages classiques de spécification algébriques permettent de définir de telles permutations entre les générateurs. Mais, lors de l'exécution, elles induisent une complexité telle que la manipulation de grands termes est impossible en pratique. Nous verrons que l'implantation en Prolog permet de résoudre facilement et élégamment ce problème.

Dans les spécifications, les opérateurs sont tous définis en décrivant leur comportement vis-à-vis des générateurs de base. Le choix d'une implantation se résume ainsi, pour l'essentiel, au choix de l'implantation des générateurs de base et à la représentation des termes dans le langage choisi. Les autres opérateurs sont alors implantés par des fonctions faisant appel aux générateurs de base.

Avant d'aller plus loin, revenons sur quelques notions de base du langage Prolog. C'est une implantation de la logique du premier ordre avec des prédicats et des variables. Les règles d'inférence y sont des clauses de Horn, c'est-à-dire de la forme $p_1 \wedge \ldots \wedge p_n \vdash p$ où les p_i sont soit des prédicats, soit une disjonction de prédicats.

Dans la syntaxe de Prolog une clause s'écrit par exemple `P :- P1, (P2 | P3), P4.` où `P` et les `Pi` sont des prédicats avec variables et les signes `","` et `"|"` représentent respectivement le "et" et le "ou" logique. Elle se lit `P` est vrai s'il est possible de démontrer que l'expression `(P1` $\wedge$ `(P2` $\vee$ `P3)` $\wedge$ `P4)` est vraie. Les prédicats sont évalués successivement de gauche à droite. Un programme Prolog est, pour résumer, constitué d'un ensemble de faits connus, sous forme de prédicats, et d'un ensemble de règles d'inférence, sous forme de clauses.

Le moteur d'inférence de Prolog permet alors de vérifier si un prédicat est vrai en utilisant les règles et les fait connus. En particulier, celui-ci utilise le *back-tracking* pour envisager toutes les règles applicables à l'ensemble des faits connus. Cette propriété s'avère très utile pour une implantation simple, mais efficace, de nos systèmes de réécriture.

7.1.1 Implantation des spécifications des cartes

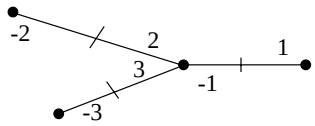
Comme nous l'avons remarqué précédemment, manipuler simplement les termes représentant une carte n'est pas envisageable en pratique à cause des permutations nécessaires. Nous devons donc envisager une autre solution. Dans les systèmes de réécriture, nous avons déjà fait remarquer que les permutations étaient équivalentes, d'un point de vue sémantique, à des tests existentiels.

Or, le moteur d'inférence de Prolog réalise quelque chose de similaire, puisqu'il recherche, et donc teste l'existence, des faits et des règles permettant de valider un prédicat. De plus,

l'ordre des faits dans un programme Prolog est indifférent. Ainsi, un prédicat valide pour un ensemble de faits, l'est également pour toute permutation de cet ensemble.

Ces similitudes de fonctionnement nous ont encouragé à utiliser le langage Prolog et conduisent alors à une implantation formelle qui reste simple et relativement efficace. Un terme représentant une carte est formé d'occurrences des générateurs de base. En Prolog, nous représentons un tel terme par un ensemble de faits dont chaque élément représente une occurrence d'un générateur. Une carte est donc représentée par une base de faits.

Le langage Prolog que nous utilisons autorise la définition de plusieurs bases de faits distinguées par un identificateur. Ainsi, dans notre prototype, plusieurs cartes peuvent cohabiter et sont distinguées par le nom de leur base de faits. Cependant, pour simplifier l'exposé et ne pas rentrer dans les détails de syntaxe, nous considérons par la suite que nous ne manipulons qu'une carte et omettons donc le paramètre de sorte *Carte* dans les différents opérateurs présentés.




```
l0(1,-1)
l0(2,-2)
l0(3,-3)
l1(-1,2)
l1(2,3)
l1(3,-1)
```

FIG. 7.1: *Une carte et la base de faits correspondant*

Exemple 7.1.1 *La figure 7.1 montre un exemple simple de base de faits pour une carte représentée par le terme* $l1(l1(l1(l0(l0(l0(v,1,-1),2,-2),3,-3),-1,2),2,3),3,-1)$. □

Chaque occurrence d'un générateur de base, $l0, l1, l2, gp0, gl0, gl1, gl2$ ou $gl3$, est transposée en un fait correspondant, `l0, l1, ..., gl3`. L'occurrence du générateur v, qui crée une carte vide correspondant à une base de faits vide, est implantée par le prédicat `abolish` qui vide, par effet de bord, la base de faits.

L'application, à une carte, d'un générateur de base, en tant que fonction utilisée par les autres opérateurs, correspond à l'ajout d'une occurrence de ce générateur dans le terme représentant une carte. Les générateurs de base sont donc implantés par des prédicats ajoutant, à la base de faits, le fait correspondant à son occurrence dans le terme, par effet de bord.

Les destructeurs de base sont implantés selon le même principe. L'application d'un destructeur correspond à la suppression d'occurrences de générateurs dans le terme représentant la carte sur laquelle il est appliqué. Ils sont donc implantés par des prédicats enlevant le fait correspondant de la base.

Exemple 7.1.2 *Dans la spécification CARTE-TOPO, l'axiome non implicite définissant la fonction* $ca(C,x,x',y')$ *qui coupe l'arête de* x *en insérant les brins* x' *et* y' *est le suivant :*

$$\begin{aligned} ca(l0(C,x,y),z,x',y') = \ & \textbf{si } z = x \vee z = y \\ & \textbf{alors } l1(l1(l0(l0(C,x,x'),y,y'),x',y'),y',x') \\ & \textbf{sinon } l0(ca(C,z,x',y'),x,y) \end{aligned}$$

Il est implanté de la façon suivante :

```
ca(Z,X',Y') :- l0(X,Y), (Z == X | Z == Y),
    retract(l0(X,Y)),
    assert(l0(X,X')), assert(l0(Y,Y')),
    assert(l1(X',Y')), assert(l1(Y',X')).
```

L'implantation de `ca(Z,X',Y')` *en Prolog se lit de la façon suivante : s'il existe un fait* `l0(X,Y)` *tel que* `Z` *soit égal à* `X` *ou* `Y` *alors le fait* `l0(X,Y)` *est supprimé, par le prédicat* `retract`*, et les autres faits sont ajoutés à la base, par le prédicat* `assert`*. Ceci correspond bien au fait que la fonction ca supprime une occurrence de* $l0$ *et la remplace par quatre autres applications de générateurs.*

Les axiomes implicites, comme nous l'avons vu, expriment le fait qu'un opérateur n'a pas d'influence sur un générateur. Ils n'ont donc pas besoin d'être implantés. □

En Prolog, seuls des prédicats sont manipulés. Une expression du type $y = f(x_1, \ldots, x_n)$ est alors implantée par un prédicat de la forme `f(X1,...,Xn,Y)` qui est vrai si `Y` est l'image par `f` des variables `X1,...,Xn` et si les préconditions de `f` sont remplies. En particulier, cette technique est utilisée pour les sélecteurs de base qui testent l'existence du fait correspondant aux générateurs de base intervenant dans leurs axiomes non implicites. Les fonctions revoyant un booléen sont simplement implantées par des prédicats.

Exemple 7.1.3 *Dans la spécification CARTE, les axiomes de* α_0 *et e sont :*

$$\begin{aligned}\alpha_0(l0(C,x,y),z) &= \textbf{si } z = x \textbf{ alors } y \\ &\quad \textbf{sinon si } z = y \textbf{ alors } x \\ &\quad\quad \textbf{sinon } \alpha_0(C,z) \\ e(l0(C,x,y),z) &= (z = x) \vee (z = y) \vee e(C,z)\end{aligned}$$

Ces deux opérations sont implantées par :

```
alpha0(X,Y) :- l0(X,Y).
alpha0(X,Y) :- l0(Y,X).

e(X) :- l0(X,Y) | l0(Y,X).
```

Le prédicat `alpha0(X,Y)` *est vrai si* `Y` *est l'image par* α_0 *de* `X`*. Il est défini par deux règles correspondant au deux cas décrits, par le mécanisme du* **si ... alors ... sinon ...***, dans l'axiome de* α_0*. Le prédicat* `e(C)` *est vrai si* `X` *appartient à la carte, c'est-à-dire si il existe un fait assurant que* `X` *a été inséré.* □

Dans les spécifications, nous utilisons souvent des compositions de fonctions qui correspondent, bien sûr à des applications successives de celles-ci. Ainsi, une expression du type $y = f(g(\ldots), \ldots)$ est implantée par l'évaluation successive des deux prédicats correspondants où le résultat intermédiaire, la valeur de $g(\ldots)$, est transmis de l'un à l'autre par une variable `X`. Ceci s'écrit `g(...,X), f(...,X,...,Y)` en Prolog.

Les parcours, qui sont décrits dans les spécifications par l'utilisation de fonctions auxiliaires et de la récursivité, sont implantés de la même façon en Prolog. Ces deux aspects sont détaillés dans les exemples suivants. Les autres opérateurs sur les cartes sont implantés de la même manière par transcription directe des axiomes de la spécification.

Exemple 7.1.4 *Examinons l'implantation des opérateurs topologiques $dl1, ds, dai$ et da dont les axiomes sont rappelés ci-après.*

$$\begin{aligned}
dl1(l1(C,x,y),z) &= \textbf{si } z = x \vee z = y \textbf{ alors } dl1(C,z)\\
&\qquad \textbf{sinon } l1(dl1(C,z),x,y)\\
ds(C,z) &= \textbf{si } al1(C,z) \textbf{ alors } C\\
&\qquad \textbf{sinon si } \alpha_{-1}(C,z) = \alpha_1(C,z) \textbf{ alors } dl1(C,z)\\
&\qquad\qquad \textbf{sinon } l1(dl1(C,z),\alpha_{-1}(C,z),\alpha_1(C,z))\\
dai(l0(C,x,y),z) &= \textbf{si } z = x \vee z = y \textbf{ alors } C\\
&\qquad \textbf{sinon } l0(dai(C,z),x,y)\\
da(C,x) &= dai(ds(ds(C,x),\alpha_0(C,x)),x)
\end{aligned}$$

Leur implantation est quasiment immédiate. Les clauses que nous avons placées en commentaire sont celles qui découlent de la spécification, mais qui ne sont plus nécessaires dans le prototype. En effet, comme les règles d'inférence sont évaluées les unes à la suite des autres en Prolog, il est possible de simplifier les tests à effectuer dans certains cas.

```
dl1(Z) :- l1(X,Y), (Z == X | Z == Y),
    retract(l1(X,Y)), dl1(Z).
/* dl1(Z) :- not l1(X,Z), not l1(Z,Y). */
dl1(Z).
```

La clause commentée de `dl1` *correspond au cas* **sinon** *de l'axiome de $dl1$ et exprime le fait que le parcours de la carte est terminé si* `Z` *ne possède plus de 1-liens. Ce test est inutile car si* `Z` *possède encore un 1-lien la première clause est appliquée avant. La deuxième clause de* `dl1` *simplifiée correspond aussi à l'arrêt de la récursion qui est défini par l'axiome implicite liant $dl1$ et le générateur de base v. C'est le seul cas où un axiome implicite doit être implanté.*

```
/* ds(Z) :- al1(Z).*/
ds(Z) :- alpha_1(Y,Y_1), alpha1(Y,Y1),
    dl1(Y), (Y_1 == Y1 | assert(l1(Y_1,Y1))).
ds(Z).

dai(X) :- retract(l0(X,Y)) | retract(l0(Y,X)).

da(X) :- alpha0(X,Y), ds(X), ds(Y), dai(X).
```

Les implantations de `ds`, `dai` *et* `da` *ne présentent pas de difficulté. Leurs clauses permettent d'illustrer les principes de composition, en particulier pour* `da`*, et la conservation de résultats intermédiaires dans des variables liées, comme* `Y1` *et* `Y_1` *pour* `ds`*.* □

Exemple 7.1.5 *Cet exemple illustre l'implantation d'un parcours avec une fonction auxiliaire récursive. Les axiomes de eqs décrivent un tel parcours :*

$$\begin{aligned}
eqs(C,x,y) &= eqs'(C,x,x,y)\\
eqs'(C,x_0,x,y) &= \textbf{si } (x = y) \textbf{ alors } vrai\\
&\qquad \textbf{sinon si } \alpha_1(C,x) = x_0 \textbf{ alors } faux\\
&\qquad\qquad \textbf{sinon } eqs'(C,x_0,\alpha_1(C,x),y)
\end{aligned}$$

L'implantation est très simple. Il faut encore noter qu'une clause est ajoutée pour terminer la récursion de `eqs_aux`*, prédicat correspondant à eqs'. Elle correspond à l'axiome implicite liant eqs' à v.*

```
eqs(X,Y) :- eqs_aux(X,X,Y).

eqs_aux(X0,X,Y) :- X == Y.
eqs_aux(X0,X,Y) :- alpha1(X,X1), X1 =\= X0, eqv_aux(X0,X1,Y).
```

Par contre, le cas qui mène à $faux$, dans la spécification, n'a pas besoin d'être implanté puisque, si les deux clauses ne peuvent être appliquées, alors l'évaluation du prédicat `eqs_aux` *échoue. L'évaluation de* `eqs` *échoue également et donc ce prédicat est évalué à faux.* □

En ce qui concerne les opérations sur les objets qui ne sont pas de sorte *Carte*, comme les points, les segments ou les labels, l'implantation en Prolog est encore plus simple. Nous avons découpé les termes de sorte *Carte* en un ensemble de faits, à cause de leur taille et pour faciliter les permutations. Pour les autres termes ce problème n'existe pas et nous manipulons donc directement les termes correspondant à leurs générateurs.

Exemple 7.1.6 *Les points et les vecteurs sont manipulés sous forme de termes du type* `gp(X,Y)` *et* `gv(X,Y)`*. Ainsi, la fonction $gvp(p,q)$ qui fabrique un vecteur à partir des points p et q (cf. annexe A) est implantée par :*

```
gvp(gp(PX,PY), gp(QX,QY), gv(VX,VY)) :-
    VX is QX-PX, VY is QY-PY.
```

Le prédicat `gvp(P,Q,V)` *est donc vrai si* `V` *est le vecteur fabriqué à partir de* `P` *et* `Q`*. La syntaxe* `VX is QX-PX` *signifie que la valeur* `QX-PX` *est placée dans la variable* `VX`*, cette valeur étant effectivement calculée. Le signe* `-` *est réservé en Prolog pour un calcul formel, sans évaluation numérique.* □

Exemple 7.1.7 *Considérons une opération plus complexe comme le test d'incidence d'un point sur un segment dont la spécification est :*

$$incident(m, seg(p,q)) = \textbf{si } nullv(V) \textbf{ alors } faux \textbf{ sinon } (t > a) \wedge (t < 1-a) \wedge nullv(U - t.V)$$

$$\textbf{avec } V = gvp(p,q), U = gvp(p,m), t = \frac{ps(V,U)}{ps(V,V)}, a = \frac{\varepsilon}{nv(V)}$$

Son implantation est directe. Le test de nullité de V, $nullv(V)$, est simplement remplacé par un test de nullité pour la norme de `V`*,* `nulln(T2)`*, et sa norme est calculée directement à partir du produit scalaire, par* `sqrt(T2)`*, pour accélérer un peu les calculs.*

```
incident(M, seg(P,Q)) :-
    gvp(P,Q,V), gvp(P,M,U),
    ps(V,V,T2), not nulln(T2),
    ps(V,U,T1), T is T1/T2,
    A is epsilon/sqrt(T2),
    T > A, T < (1-A),
    scale(-T,V,TV),
    add(U,TV,N),
    nullv(N).
```

On peut noter également que les méta-variables que nous avons utilisées pour simplifier l'écriture des axiomes, sont remplacées par des variables liées contenant les calculs numériques intermédiaires. □

7.1.2 Implantation des systèmes de réécriture

Chaque règle de réécriture est implantée par un prédicat qui est vrai si la règle a pu être appliquée et qui est évalué à faux dans le cas contraire. Le moteur d'inférence de Prolog permet d'essayer d'appliquer toutes les règles jusqu'à ce qu'aucune ne puisse être déclenchée. Pour les règles du système de base, sans structure de contrôle, l'implantation est immédiate comme on peut le voir sur un exemple.

Exemple 7.1.8 *La règle de coupure d'arêtes sécantes :*

$$R_4 : \frac{gp0(gp0(C,x,p),z,q)}{cap(cap(gp0(gp0(C,x,p),z,q),x,i),z,i)} \textbf{ si } secant(gp0(gp0(C,x,p),z,q),x,z)$$
$$\textbf{avec } i = intersection(gp0(gp0(C,x,p),z,q),x,z)$$

est implantée par le prédicat :

```
regle4 :-
    gp0(X,P), gp0(Z,Q), X =\= Z,
    secant(X,Z),
    intersection(X,Z,I),
    cap(X,I), cap(Z,I).
```

Cette clause recherche deux brins distincts plongés dont les arêtes sont sécantes. Si aucun couple d'arêtes sécantes n'est trouvé, le moteur d'inférence de Prolog assure que tous les couples ont été testés et la règle échoue. Dès qu'un couple est trouvé, la règle s'applique, les arêtes sont découpées et la règle réussit. Il faut donc ressayer d'appliquer les règles jusqu'à ce que toutes échouent. □

Pour assurer que le raffinement est réalisé complètement, il faut ajouter quelques clauses forçant l'échec du raffinement tant que des règles peuvent être déclenchées. Nous ne détaillons pas cet aspect qui nous obligerait à détailler la syntaxe du Prolog à un niveau qui ne nous intéresse pas ici.

Les règles des systèmes qui utilisent des structures de contrôle sont implantées avec le même principe, mais les structures de contrôle sont, dans ce cas, des paramètres des prédicats correspondant aux règles.

Exemple 7.1.9 *La règle de coupure d'arêtes sécantes, pour le système décrivant un balayage du plan :*

$$R_{X4} : \frac{X,A,S,C}{i(i(i(i(X,x'),y'),z'),t'),A,S,cap(cap(C,x,i),z,i)} \textbf{ si } \begin{cases} x = premier(X) \\ z = gsecant(A,C,x) \neq nil \end{cases}$$
$$\textbf{avec } \begin{cases} i = intersection(C,x,z) \\ x' = \alpha_0(cap(cap(C,x,i),z,i),x) \\ y' = \alpha_0(cap(cap(C,x,i),z,i),\alpha_0(C,x)) \\ z' = \alpha_0(cap(cap(C,x,i),z,i),z) \\ z' = \alpha_0(cap(cap(C,x,i),z,i),\alpha_0(C,z)) \end{cases}$$

est implantée par le prédicat :

```
regle4(CX,CA,CS) :-
    premier(CX,X),
    gsecant(CA,X,Z), Z \== nil,
    alpha0(X,Y),
    alpha0(Z,T),
    intersection(X,Z,I),
    cap(X,I), cap(Z,I),
    alpha0(X,XX), alpha0(Y,YY),
    alpha0(Z,ZZ), alpha0(T,TT),
    i_tas(CX,XX), i_tas(CX,YY), i_tas(CX,ZZ), i_tas(CX,TT).
```

□

Les systèmes de réécriture, utilisés pour décrire l'intersection de faces en dimension 3, sont implantés par des fonctions dont les paramètres sont les différentes listes de classifications. Les études de cas décrites sous forme de système de réécriture sont implantées par un ensemble de clauses, avec une clause par cas.

Exemple 7.1.10 *Voici, par exemple, les 8 premières clauses qui définissent les transitions d'un état courant en fonction des classes des arêtes telles quelles sont définies dans le système de la table 6.5.*

```
transit_class_etat(cut  , out    , in     ).
transit_class_etat(cut  , in     , out    ).
transit_class_etat(pic  , out    , out    ).
Transit_class_etat(pic  , in     , in     ).
transit_class_etat(up_on, out    , on(1-0)).
transit_class_etat(up_on, in     , on(0-1)).
transit_class_etat(up_on, on(0-1), on(1-1)).
transit_class_etat(up_on, on(1-0), on(0-0)).
```

Et voici les clauses définissant le prédicat `etat_class(LClass,LEtat)` *qui implante le système de réécriture transformant une liste de classification,* `LClass`*, en une liste d'états,* `LEtat`*, pour les arêtes d'une face. Notons que les listes de termes font partie du langage Prolog et qu'elles s'écrivent sous la forme* `[X|L]` *où* `X` *est un terme et* `L` *une liste. La liste vide s'écrit* `[]`*. Les 7-uplets formant les éléments de ces listes sont écrit sous la forme* `(X-Class-GeoInfo)` *où le terme* `GeoInfo` *regroupe les informations géométriques (point, abscisse, angle, indicateur et orientation) non utilisées ici. L'utilisation de la virgule est ici impossible à cause de la syntaxe de Prolog.*

```
etat_face(LClass,LEtat) :-
    etat_face(LClass,out,LEtat).
etat_face([(X-Class-GeoInfo)|LClass], Etat, [(X-Etat2-GeoInfo)|LEtat]) :-
    transit_class_etat(Class,Etat,EtatSuivant),
    functor(EtatSuivant,Etat2),
    etat_face(LClass,EtatSuivant,LEtat).
etat_face_plan([],_,[]).
```

La première clause correspond à la règle d'initialisation du système qui démarre avec un état courant égal à `OUT`*. La seconde clause décrit le parcours de la liste. Le prédicat* `functor` *permet*

de ne conserver que le foncteur `ON` *d'un terme de type* `ON(I,J)`. *La dernière clause indique que la réécriture est terminée lorsque la liste est vide. Le symbole* `"_"` *indique une valeur quelconque.* □

7.2 Implantation en C

7.2.1 Générateurs de base

Le passage à une implantation en C est facilité par le travail déjà effectué pour le prototypage en Prolog. Comme nous l'avons vu, les choix d'implantation se résument, pour l'essentiel, aux choix d'implantation des termes et des générateurs de base des différentes sortes. Les sortes simples comme les booléens, les entiers, les nombres réels sont implantées directement par les types classiques offerts par le langage C.

Les objets pour lesquels les générateurs sont uniques, comme les points, les vecteurs, les segments sont implantés par des pointeurs sur des structures dont les membres sont les paramètres de leurs générateurs. Ainsi, par exemple, un segment, de générateur $gs(p,q)$, devient un pointeur sur une structure dont les membres sont deux points. Voici, par exemple, l'implantation des points 3D et de leur générateur $gp(x,y,z)$:

```
typedef struct point
{    double x,y,z;
} *Point;

Point gp(double x,double y,double z)
{   Point p = (Point) calloc(1,sizeof(struct point));
    p->x = x;
    p->y = y;
    p->z = z;
    return p;
};
```

De même, les structures de contrôle, comme les listes, les files FIFO, les arbres ou les tas sont implantées de manière classique. Les générateurs correspondent aux opérations de base sur ces structures, comme par exemple, pour les listes, la création d'une liste vide et l'ajout d'un élément en tête de liste. Parfois, lorsque leurs tailles sont connues par avance et fixes, les listes sont implantées par des tableaux. Nous ne détaillons pas plus ces opérations classiques.

Le point crucial de l'implantation est le choix d'une représentation pour les cartes et les brins. Une carte est un ensemble de brins, pour lequel les seuls opérations essentielles sont l'ajout et la suppression d'éléments. Dans les spécifications, les brins sont représentés par des entiers, mais ce choix n'avait pour but que de permettre de tester l'égalité de deux brins. La représentation des brins n'est pas importante en elle-même et le seul impératif est qu'un brin puisse être fourni en paramètre à une fonction. Par contre, les informations associées à un brin sont essentielles pour l'implantation. A chaque brin on doit pouvoir associer ses images par les différentes fonctions d'adjacence α_i, son plongement et ses labels.

Ainsi, les brins sont implantés par des pointeurs sur une structure contenant trois ou quatre brins, suivant la dimension, qui sont ses images par les α_i, un point et un label. Une carte est

alors implantée comme une liste doublement chaînée de brins. Pour cela, nous ajoutons à la structure d'un brin deux brins représentants le brin suivant et le brin précédent dans la carte. Voici les structures utilisées pour les brins d'une 3-carte :

```
typedef unsigned int indice;
typedef struct dart *Dart;

typedef struct dart
{   indice ind;
      Bool m;
      Dart a0,a1,a2,a_2;
      Dart succ,pred;
     Point e0;
      Plan e2;
      Bbox bb;
   GeoInfo info;
} dart_struct;
```

Les quatres brins `a0, a1, a2` et `a_2` sont les images par α_0, α_1, α_2 et α_{-2}. Le dernier est ajouté pour faciliter les fusions d'arêtes. Les brins `succ` et `pred` sont les brins permettant le chaînage dans une carte. Le point `e0` est le 0-plongement du brin. Il faut noter que nous utilisons un 0-plongement par brin, pour accélérer les traitements et éviter de parcourir les sommets pour rechercher le 0-plongement d'un brin. Cependant, pour respecter la spécification, nous assurons que le point est le même pour tous les brins d'un sommet.

De plus, quelques champs sont ajoutés pour faciliter les traitements. L'indice `ind` est utilisé pour numéroter les brins lors de la sauvergarde sur fichier d'une carte. Le booléen `m` est un marqueur facilitant les parcours de sommets ou volumes.

D'autres champs sont ajoutés pour stocker des données géométriques. Le plan `e2` est le plan sous-jacent au 2-plongement de la face du brin. Pour éviter de calculer ce plan de multiples fois, un prétraitement est effectué, avant le raffinement, durant lequel les équations des plans de chaque face sont calculées une fois pour toutes. Pendant ce prétraitement, nous calculons également une boîte englobante de la face, conservée pour chaque brin dans le champ `bb`. Enfin, le champ `info` reçoit, lors de l'intersection de faces, les informations géométriques calculées pour ce brin.

7.2.2 Implantation des opérations topologiques et géométriques

Les sélecteurs de base sont implantés très simplement par des macros permettant la consultation des champs d'un brin. Notons que des compositions de sélecteurs simples sont implantées de la même manière, comme φ_1 par `phi1`. Voici ces implantations :

```
#define alpha0(x)  x->a0
#define alpha1(x)  x->a1

#define phi1(x)    alpha1(alpha0(x))

#define gem0(x)    x->e0
```

```
Dart alpha2(Dart x)
{   Dart x_2 = x;

    if (x->a2 != NULL) return x->a2;
    while (x_2->a_2 != NULL) x_2 = x_2->a_2;
    l2(x,x_2); /* complétion de la 2-liaison */
    return x_2;
};
```

Il faut noter l'exception existant pour `alpha2`. En cas de 2-liaisons incomplètes, il est nécessaire de parcourir les brins du sommet pour rechercher l'image par α_2. Dans les spécifications, cette recherche est définie par une récursion croisée entre α_2 et α_{-2}, ce qui n'est pas efficace. Nous avons donc remplacé cette récursivité par une boucle. De plus, la fonction `alpha2` complète, par effet de bord, les 2-liaisons trouvées lors de ces parcours, pour éviter de les effectuer une nouvelle fois.

Les générateurs de base, comme $l0, l1, l2$ ou $gp0$, sont implantés par des macros ou des fonctions qui modifient les champs des brins correspondants. Ils ne modifient bien sûr pas la liste chaînée qui modélise la carte contenant les brins impliqués. Ainsi, comme le paramêtre de sorte *Carte* n'est pas utile pour l'implantation de ces opérations topologiques, nous l'avons systématiquement oté. Comme par exemple :

```
#define l0(x,y) x->a0 = y; y->a0  = x
#define l1(x,y) x->a1 = y; y->a1  = x
#define em0(x,p) (x->e0) = copypoint(p)

void l2(Dart x,Dart y)
{   if (x != y)
    {   x->a2  = y;
        y->a_2 = x;
    }
    else
    {   x->a2  = NULL;
        x->a_2 = NULL;
    };
};
```

Les cartes sont implantés par des listes chaînées de brins dont la structure est donnée ci-après. Les champs `first` et `last` sont respectivement le premier et le dernier brin de la carte. Nous avons, de plus, ajouté deux champs `current` et `current2` pour permettre deux parcours distincts de la liste des brins.

```
typedef struct map
{   Dart first,last;
    Dart current,current2;
} *Map;
```

Le générateur v qui crée une carte vide est implanté très simplement. Il initialise les champs d'une structure de type `Map` :

```
Map v(void)
{   Map m = (Map) calloc(1,sizeof(struct map));

    m->nbr_dart = 0;
    m->first    = NULL;
    m->last     = NULL;
    m->current  = NULL;
    m->current2 = NULL;

    return m;
};
```

Le seul point qui nécessite un traitement particulier est la création d'un nouveau brin, réalisé par la fonction `ndart`. Cette fonction retourne un nouveau brin dont la structure est initialisée. De plus, par effet de bord, cette fonction met à jour les pointeurs de chaînage de la carte :

```
Dart ndart(Map m)
{   Dart x = (Dart) calloc(1,sizeof(struct dart));

    x->m    = FALSE;
    x->a0   = x->a1   = x;
    x->a2   = x->a_2  = NULL;
    x->pred = x->succ = NULL;
    x->e0   = NULL;
    x->e2   = NULL;
    x->bb   = NULL;
    x->info = NULL;

    (m->nbr_dart)++;
    if (m->first == NULL)
    {   m->first    = x;
        m->last     = x;
        m->current  = x;
        m->current2 = x;
    }
    else
    {   m->last->succ = x;
        x->pred = m->last;
        m->last = x;
    };
    return x;
};
```

Les opérateurs de plus hauts niveaux sont alors implantés naturellement, comme dans le prototype, par des appels aux générateurs de base. Nous ne détaillons pas toutes ces opérations car ce serait fastidieux. Mais illustrons juste le principe avec l'exemple de la coupure d'arête déjà détaillée pour le prototype. La fonction `cutee` coupe l'arête du brin `x` au point `p`. Ici le paramêtre de type `Map` est implanté, puisque l'insertion des nouveaux brins nécessite des modifications dans la structure de la carte.

```
void cutee(Map m,Dart x,Point p)
{   Dart y  = alpha0(x);
    Dart xx = ndart(m);
    Dart yy = ndart(m);

    l0(x,xx);
    l0(y,yy);
    l1(xx,yy);
    em0(xx,p);
    em0(yy,p);
};
```

La fonction fait appel à `ndart`, pour créer les nouveaux brins. Puis, elle appelle les générateurs permettant de créer les nouvelles liaisons topologiques et les 0-plongements de brins insérés.

Toutes les opérations topologiques, de plongement ou géométriques sont implantées pour être réalisées en temps constant. Les seules exceptions sont, bien sûr, les fonctions nécessitant un parcours de sommet, de face ou de volume, dont la complexité en temps est proportionnelle au nombre de brins de la cellule. Cette remarque nous permet d'affirmer que les méthodes formelles utilisées ne sont, en aucun cas, un frein à l'obtention de programmes efficaces, après implantation.

De plus, il faut noter que le travail effectué pendant les aller-retour entre la spécification et le prototype facilite grandement le travail d'implantation. Les seuls problèmes à résoudre se situent dans le choix des structures de données et des générateurs de base. Une fois ce choix effectué, l'implantation des opérations de haut niveau est quasiment immédiate et suit à la lettre les définitions données dans les spécifications.

Enfin, notre approche à l'avantage de nettement séparer la conception de l'implantation. Les problèmes de conception sont résolus principalement pendant la phase de spécifications et en partie pendant le prototypage logique, en faisant abstraction des problèmes d'implantation. L'implantation se fait lorsque tous ces problèmes sont résolus, ce qui permet de se concentrer sur les problèmes d'optimisation.

7.2.3 Implantation de l'intersection de faces en dimension 3

Les systèmes de réécriture définissant l'intersection de faces non coplanaires décrivent en fait un parcours de chaque face pour obtenir les informations géométriques, puis des parcours de listes de brins. Le premier parcours est simplement implanté par une boucle sur les brins d'une face. Puis le nombre de brins restant constant, les brins de la face qui coupent le plan de l'autre face sont placés dans un tableau, après leur classification.

```
TDart class_face_plan(Dart x,Vect nx,Repere1D rep,Plan py)
{   Dart y = x,y_1;
    Seg s;
    Type t;
    Point i;
    double k,a;
    int n = 0;
    short in;
    TDart tab;

    /* parcours des brins de la face et obtention des informations gémétriques */
    do
    {   s = gem1(y); /* segment sur lequel est 1-plongé y */
        if (type_seg(s,nx,py,rep,&t,&i,&k,&a,&in))
            new_info(y,t,i,k,a,in); /* remplit le champ info du brin */
        free(s);
        y = phi1(y); /* brin suivant de la face */
    } while (y != x);

    /* classification des arêtes par un deuxième parcours de la face */
    y_1 = phi1_1(y);
    do
    {   switch(get_type(y))
        {   case CUTSEG : set_class(y,CUT,get_inside(y)); n++; break;
            case ONUP   : switch(get_type(y_1))
            {   case UPON   : set_class(y,PIC    , 0); n++; break;
                case DOWNON : set_class(y,CUT    , get_inside(y)); n++; break;
                case ON1    : set_class(y,ON_UP  ,-2); n++; break;
                case ON_1   : set_class(y,UP_ON  , 2); n++; break;
            }; break;
            case ONDOWN : switch(get_type(y_1))
            {   case UPON   : set_class(y,CUT    , get_inside(y)); n++; break;
                case DOWNON : set_class(y,PIC    , 0); n++; break;
                case ON1    : set_class(y,ON_DOWN,-4); n++; break;
                case ON_1   : set_class(y,DOWN_ON, 4); n++; break;
            }; break;

            /* autres cas ... */
        };
        y_1 = y;
        y = phi1(y);
    } while (y != x);

    /* les brins qui coupent le plan sont placés dans un tableau */
    tab = gtdart(n);

    n = 0;
```

```
    do
    {   if (get_class(y) != NIL)
        {   set_elt(tab,n,y); /* place le brin y dans le tableau */
            n++;
        }
        else
            free_info(y); /* si le brin ne coupe pas la droite
                             on libère les informations géométriques */
        y = phi1(y);
    } while (y != x);

    /* tri des arêtes le long de la droite d'intersection */
    /* la fonction cmp_info est l'implantation de l'ordre sur les 7-uplets */
    qsort(tab->tab,tab->nbr,sizeof(Dart),cmp_info);

    return tab;
};
```

Les autres systèmes sont alors implantés par un parcours des éléments du tableau. Les différents cas décrit par les règles des systèmes sont implantés par l'intermédiaire de `switch`. Par exemple, la fontion `etat_face_plan` calcule les états de chaque brin. Le traitement des incohérences numériques provoquées par le tri a été supprimé pour faciliter la lecture.

```
void etat_face_plan(TDart t)
{   indice i;
    Dart x;
    V_Etat e = OUT; /* état courant */

    /* parcours du tableau */
    for (i = 0;i<get_size(t);i++)
    {   x = get_elt(t,i); /* ième élément du tableau */
        /* calcul du l'état courant après le brin courant */
        switch(get_class(x))
        {   case CUT     : switch(e)
            {   case OUT   : e = IN   ; break;
                case IN    : e = OUT  ; break;
                default    : error("CUT");
                /* mise en évidence d'incohérence numériques */
            }; break;
            case ON_DOWN : switch(e)
            {   case ON0_1 : e = OUT  ; break;
                case ON1_0 : e = IN   ; break;
                case ON1_1 : e = ON1_0; break;
                case ON0_0 : e = ON0_1; break;
                default    : error("ON_DOWN");
                /* mise en évidence d'incohérence numériques */
            }; break;
```

```
        /* autre cas ... */
        };
        /* états retenus pour le brin */
        switch (e)
        {   case OUT : set_etat(x,OUT); break;
            case IN  : set_etat(x, IN); break;
            default  : set_etat(x, ON); break;
        };
    };
};
```

7.2.4 Implantation des systèmes de réécriture

Les systèmes de réécriture définissant les raffinements sont implantés de la même façon que pour le prototype en Prolog. Chaque règle est implantée par une fonction ayant en paramètre les structures de contrôle éventuelles. Cette fonction teste les conditions de déclenchement et réalise, si nécessaire, les transformations décrites par la règle.

Les systèmes de réécriture, quant à eux, sont implantés par des boucles réalisant les parcours nécessaires. Lorsque des stratégies de parcours sont fixées par des structures de contrôle, le système consiste juste à appeler les fonctions correspondant aux règles jusqu'à ce que la condition d'arrêt soit remplie.

Les stratégies de parcours et les algorithmes en découlant, comme l'algorithme de balayage du plan, ont déjà été abordées complètement en dimension 2. Nous ne revenons donc pas dessus. Par contre, en dimension 3, quelques optimisations ont été implantées. D'une part, comme cela a été abordé précédement, des boîtes englobantes ont été ajoutées à la structure des brins. Les fonctions correspondant aux règles du raffinement 3D ne sont appelées que si les deux brins courants du parcours ont des boîtes englobantes qui se coupent.

D'autre part, le système de réécriture 3D décrit un parcours de tous les couples de brins. Or pour les règles d'intersection de faces, un parcours des couples de faces est suffisant. Nous avons donc implanté ce système en deux étapes. La première réalise un parcours des couples de faces durant lequel on appelle, pour chaque couple, les fonctions d'intersection de faces. Pour cela, nous gérons une structure de contrôle supplémentaire contenant la liste des faces de la carte. La seconde étape est un parcours de tous les couples de brins durant lequel on appelle les règles de fusion de faces et d'arêtes.

7.2.5 Implantation des labels

Dans les spécifications, les labels sont définis par des ensembles d'identificateurs d'objets. En pratique le nombre d'objets est souvent limité. Nous avons donc choisit d'implanter les labels par des champs de bits, c'est-à-dire en fait des entiers non signés. Ceci limite le nombre d'objets à 32, dans notre implantation. Cependant cela simplife l'implantation des oprérations sur les labels. Les unions et différences utilisées pour la complétion des labels sont implantées par des opérations logiques bits à bits.

L'évaluation des labels a été prototypée et implantée en dimension 2. Bien que complètement spécifiées, les opérations sur les labels n'ont pas été implantées en dimension 3. La manipulation des labels sur des objets en 3D aurait impliqué la réalisation d'un modeleur 3D permettant de gérer plusieurs objets, leurs labels et des expressions booléennes correspondant à des arbres CSG sur ces objets. Or nous avons manqué de temps pour cela et l'implantation d'un tel modeleur sortait quelque peu du cadre de notre étude.

En fait, les objets manipulés sont des 3-cartes qui sont créées par le modeleur Topofil, développé par Yves Bertrand à Strasbourg [12], et sauvergardées dans des fichiers distincts. Notre implantation réalise le raffinement d'une carte correspondant à l'importation d'un nombre quelconque de fichiers de cartes au format de Topofil. Le résultat du raffinement est sauvergardé dans un fichier lisible par ce modeleur.

Chapitre 8

Expérimentation

Dans ce chapitre, nous présentons quelques exemples d'expérimentations des raffinements, en dimension 2 et 3. Nous montrons, en premier lieu, comment le raffinement généralise les problèmes d'arrangements de segments dans le plan. Ainsi, en dimension 2, nous donnons des exemples de reconstruction topologiques de subdivisions planaires, à partir de données incomplètes. Puis, nous abordons des exemples d'évaluations d'opérations booléennes faisant intervenir les labels. Nous concluons, pour la dimension 2, en montrant comment sont gérés les cas d'inclusions de composantes connexes multiples.

En dimension 3, nous montrons comment nous avons appliqué le raffinement 3D à des problèmes de constructions de subdivisions volumiques en géologie, par des arrangements de plans ou de surfaces. Nous terminons enfin par quelques exemples de raffinement sur des solides polyédriques plus complexes.

8.1 Raffinement en dimension 2

Nous commençons par présenter, dans la figure 8.1, l'interface graphique réalisée pour le prototypage logique en Prolog. La fenêtre de contrôle nous permet de gérer les cellules et les labels correspondants, pour chaque dimension, c'est-à-dire pour les sommets, arêtes et faces. L'interface propose également les quatre stratégies de raffinement que nous avons étudiées et quelques expressions booléennes basiques souvent utilisées.

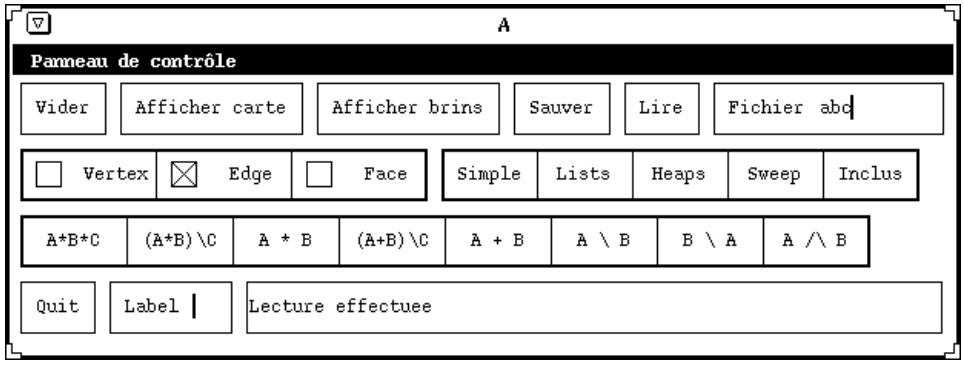


FIG. 8.1: *Fenêtre de contrôle du prototype*

La figure 8.2, montre un ensemble de segments du plan qui sont, en pratique, affichés dans une seconde fenêtre. Les sommets sont représentés par des petits carrés et les segments sont légèrement éclatés pour distinguer les sommets topologiques des intersections de segments. Cette première figure correspond à une carte mal plongée.

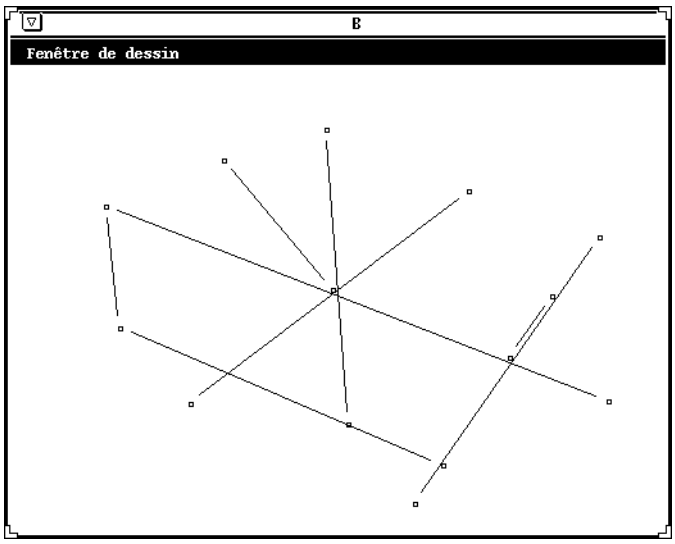


FIG. 8.2: *Un ensemble de segments*

Notons que, dans cette figure, tous les brins sont points fixes de α_1, c'est-à-dire qu'il n'existe pas de 1-liaisons. La figure comprend plusieurs configurations qui sont habituellement traitées comme des cas particuliers [16]. On peut remarquer 4 arêtes s'intersectant en un même point et une arête partiellement superposée à une autre.

La figure 8.3, montre le raffinement 2D de cet ensemble de segments. Les liaisons par α_1 autour des sommets sont représentées par des petits tubes reliant un segment au sommet auquel il est incident. La carte représentée ici et maintenant bien plongée et réalise une partition du plan. Notons qu'un des sommets contient 7 brins et que les arêtes superposées ont été découpées et fusionnées.

Comme on peut le voir sur cet exemple le raffinement 2D, dans le cas d'un ensemble de segments, correspond à l'*arrangement* de ces segments dans le plan [16]. L'exemple suivant montre que le raffinement est en fait une extension de l'arrangement. L'ensemble de départ peut être constitué de composantes connexes de cartes superposées. De plus, dans l'exemple de la figure 8.4, les liaisons par α_1 sont incomplètes.

La représentation des 1-liaisons par des tubes permet de visualiser facilement celles qui manquent. Remarquons encore la présence d'une arête verticale traitée exactement comme les autres. Après le raffinement, toutes les liaisons sont correctement calculées. Cela montre que notre raffinement est réellement un mécanisme de reconstruction topologique, puisque les relations d'incidence ou d'adjacence fournies au départ peuvent être incomplètes.

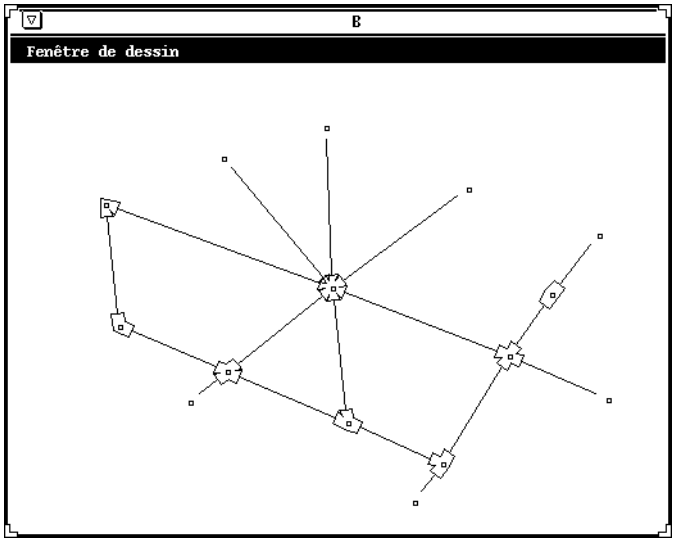


FIG. 8.3: *Le raffinement réalise l'arrangement des segments*

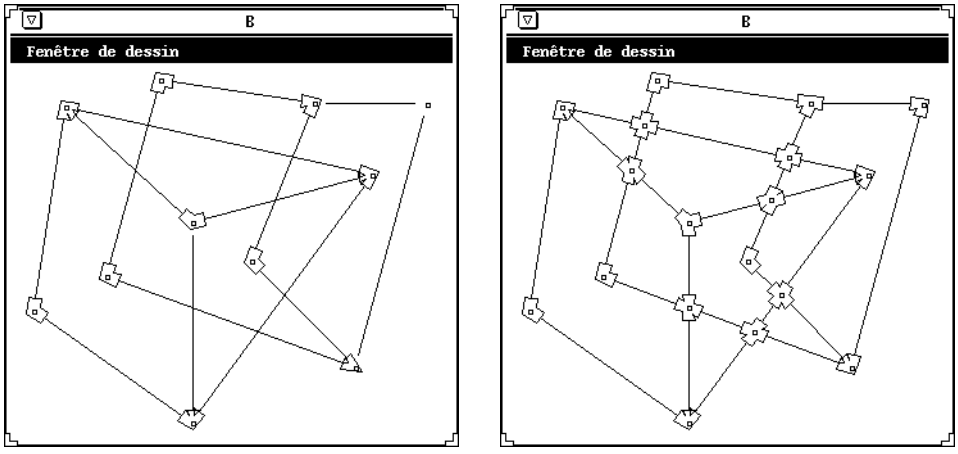


FIG. 8.4: *Une carte et son raffinement*

8.2 Evaluation d'opérations booléennes en dimension 2

La figure 8.5 représente une carte avec deux composantes connexes modélisant trois objets. Les 2-labels des faces sont représentés par une lettre placée à gauche des arêtes de la face.

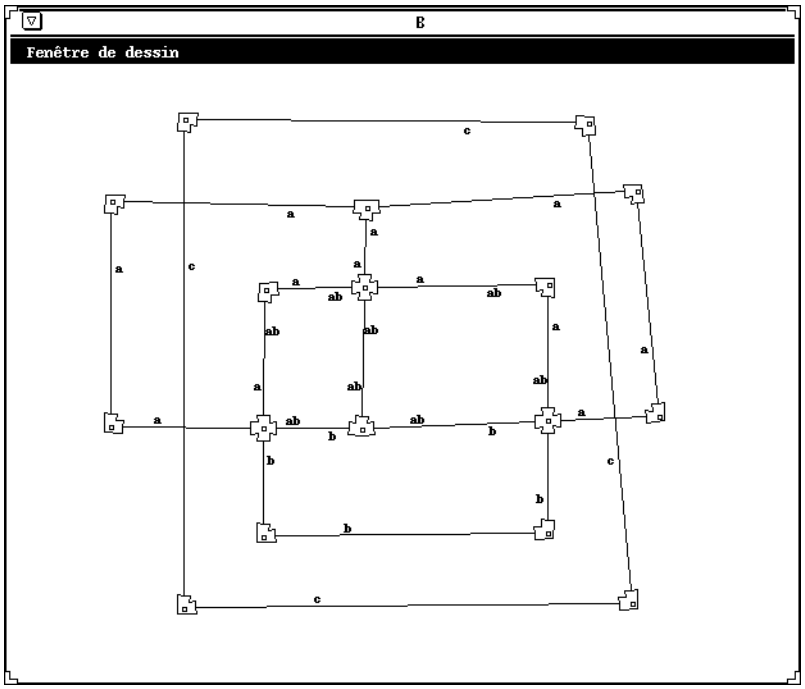


FIG. 8.5: *Trois objets et leurs 2-labels*

Dans cette section, nous considérons des objets réguliers, c'est-à-dire dont les 0 et 1-labels sont identiques aux 2-labels pour chaque brin. L'objet C est formé par l'intérieur du grand carré. Les objets A et B sont modélisés par la deuxième composante connexe et sont formés respectivement de quatre et de trois carrés. Notons que A et B ont deux carrés en commun.

Le résultat du raffinement et de la complétion des labels est donné dans la figure 8.6. Les différentes faces de la carte raffinée ont été subdivisées. Leurs 2-labels ont été propagés partout. On peut noter que la transmission locale réalisée lors de la complétion permet de propager les labels de C dans les faces modélisant B, alors qu'il n'y avait pas d'intersection entre ces deux objets. Seules les relations topologiques sont ainsi utilisées pour déduire que l'objet B est complètement à l'intérieur de l'objet C. Ainsi aucun calcul numérique de localisation n'est nécessaire, ce qui fait une différence majeure avec les méthodes classiques.

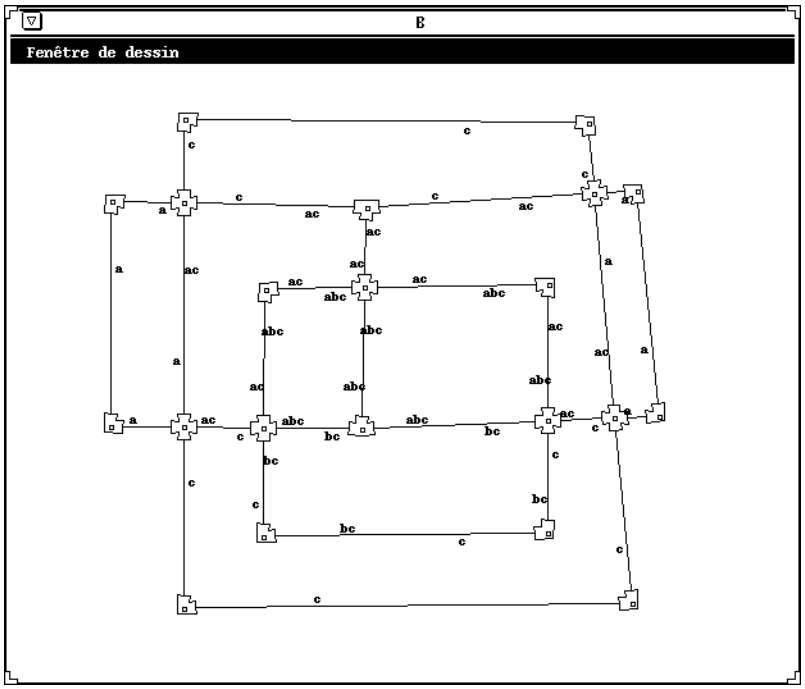


FIG. 8.6: *Après raffinement et complétion des labels*

Les deux figures suivantes, 8.7 et 8.8, montrent un exemple un peu plus complexe de raffinement sur une carte modélisant toujours 3 objets, avec des 2-labels uniquement. Dans cette carte, on peut noter des arêtes verticales partiellement superposées. Les figures suivantes, 8.9 et 8.10, montrent quelques exemples d'extractions d'opérations booléennes à partir du raffinement. On peut noter, encore une fois ici, qu'après le raffinement et la complétion des labels, toutes les expressions booléennes sur les objets identifiés au départ peuvent être réalisées par de simples sélections.

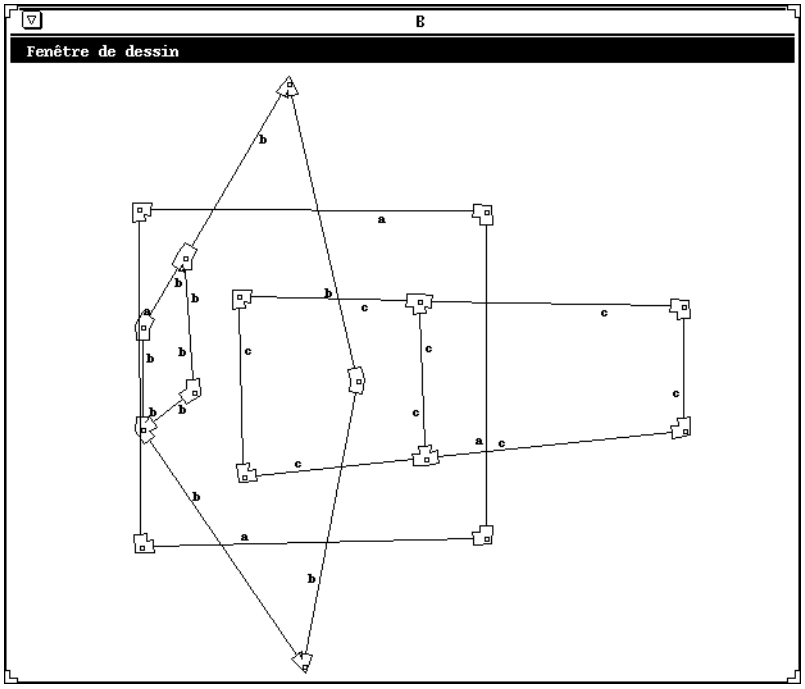


FIG. 8.7: *Une carte modélisant trois objets*

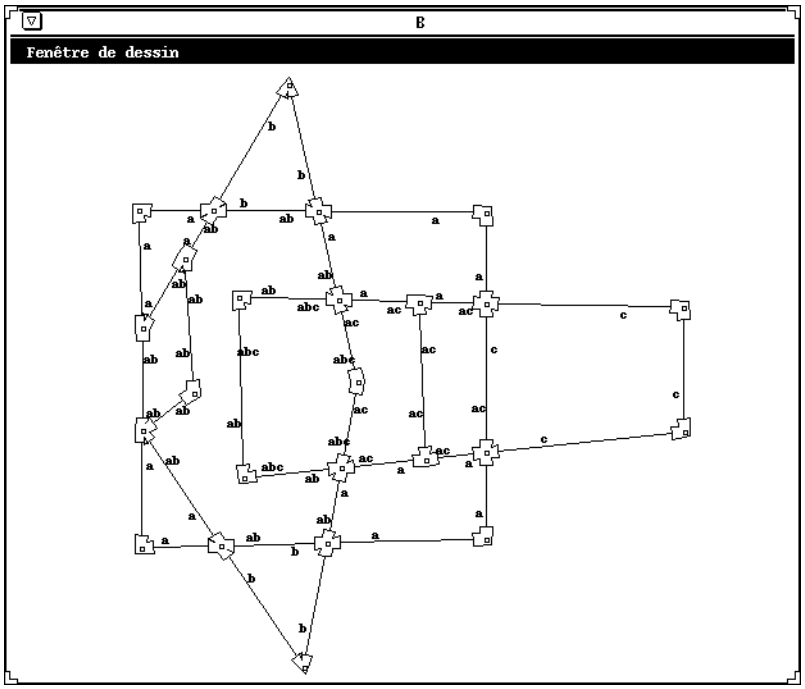


FIG. 8.8: *Le raffinement de cette carte, après complétion des labels*

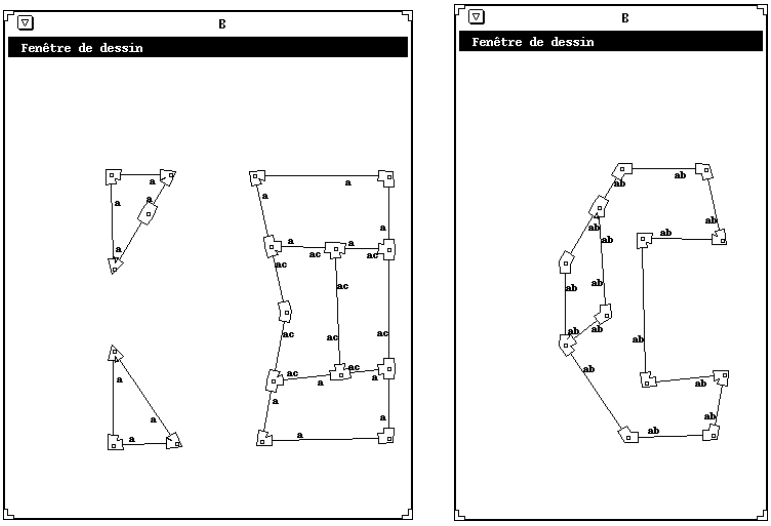


FIG. 8.9: *Extraction de* $A \setminus B$ *et* $(A \cap B) \setminus C$

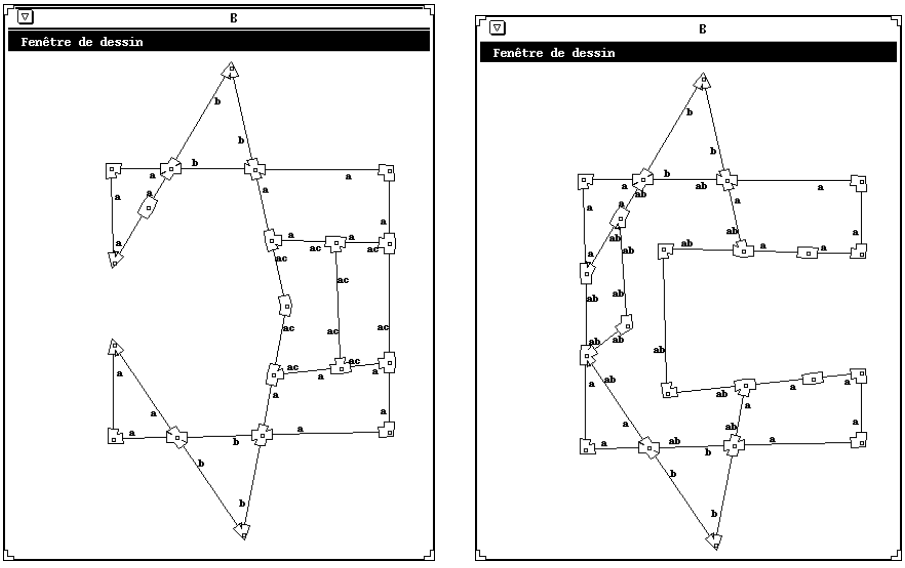


FIG. 8.10: *Extraction de* $(A \setminus B) \cup (B \setminus A)$ *et* $(A \cup B) \setminus C$

8.3 Gestion des relations d'inclusion

Les figures 8.11 à 8.13 représentent un exemple de carte comportant plusieurs composantes connexes incluses les unes dans les autres. La première figure 8.11 montre la carte avant raffinement. Elle définit quatre objets A, B, C et D.

La figure 8.12 montre cette carte après le raffinement. Des arêtes ont été ajoutées pour représenter les liens d'inclusion entre les différentes composantes. Notons que ces arêtes sont des arêtes topologiques et n'ont pas de plongement. Elles sont représentées ici par des segments qui peuvent intersecter les arêtes initiales de la carte. Il faut considérer que ces arêtes sont déformables à volonté et qu'il serait possible de leur donner un plongement qui contourne les cellules initiales sans les intersecter.

Ces arêtes sont ajoutées lors du balayage du plan dès qu'un brin gauche d'une arête est rencontré et si ce brin n'est lié à aucun autre à gauche de la droite de balayage. La figure 8.13 représente deux états de la carte lors du balayage du plan. Elle schématise la droite de balayage et l'ensemble des brins actifs qui sont affichés en gras.

Les arêtes représentant les liens d'inclusion sont ajoutées avec des labels vides et permettent donc la transmission de tous les labels entre les diverses composantes connexes. Avec ce principe, il est possible d'insérer trop d'arêtes, ce qui n'est pas gênant pour le raffinement. En effet, après la complétion des labels, les arêtes ajoutées peuvent être supprimées ou pas suivant les besoins. De plus, il est toujours possible d'obtenir un nombre minimal d'arêtes en vérifiant après le raffinement que chacune d'elles connecte bien deux composantes. Pour cela, il suffit de tester si les deux brins de l'arête appartiennent à la même face.

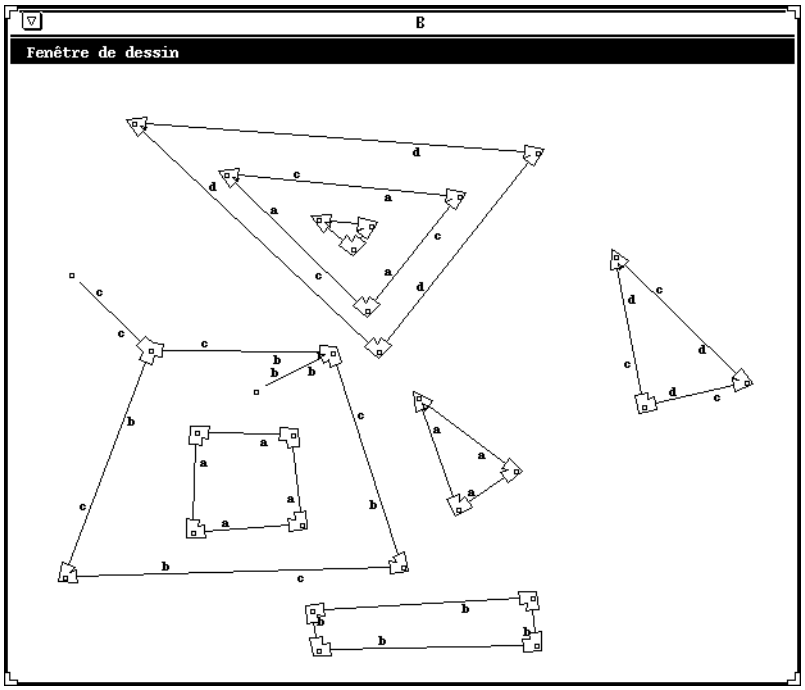


FIG. 8.11: *Une carte comportant plusieurs composantes connexes*

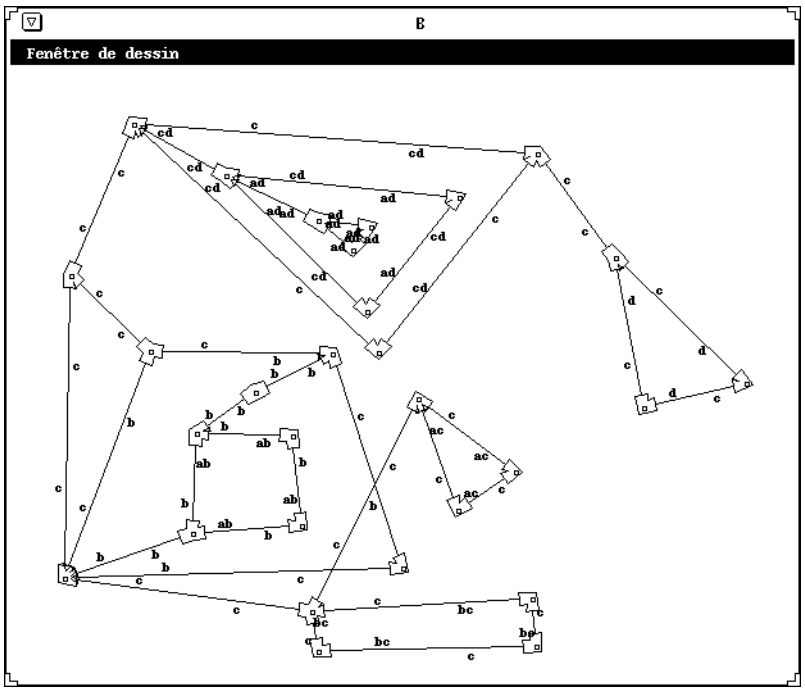


FIG. 8.12: *Leur arbre d'inclusion après raffinement*

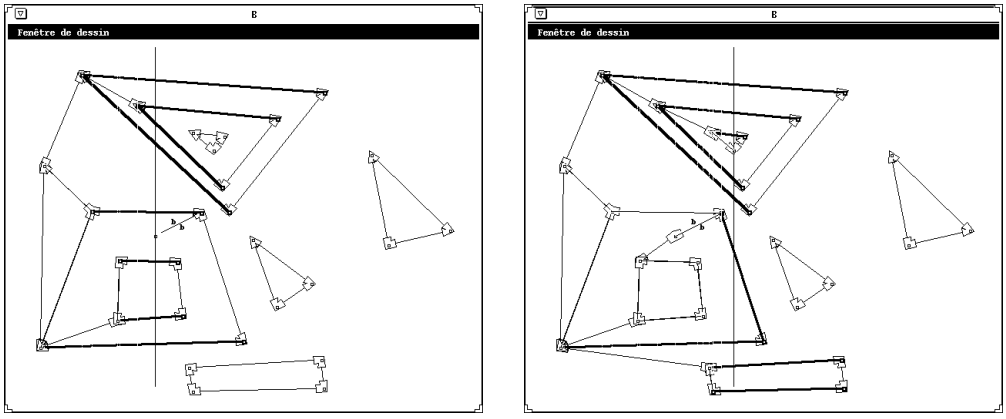


FIG. 8.13: *Ensemble des segments actifs pour deux étapes du balayage*

8.4 Exemple d'objets non réguliers

Les figures suivantes représentent des objets non régularisés, c'est-à-dire dont les labels sont quelconques. Nous reprenons ici l'exemple de la figure 2.11 du chapitre 2 dont le schéma est redonné dans la figure 8.14. Il n'est pas aisé de représenter tous les labels en même temps à cause des limitations du graphisme autorisé en Prolog. Ainsi, les cartes sont représentées ici par trois figures distinctes. La première représente les 2-labels comme précédemment. Les deux autres représentent respectivement les 0 et 1-labels des brins à côté de leurs sommets et de chaque côté de leurs arêtes.

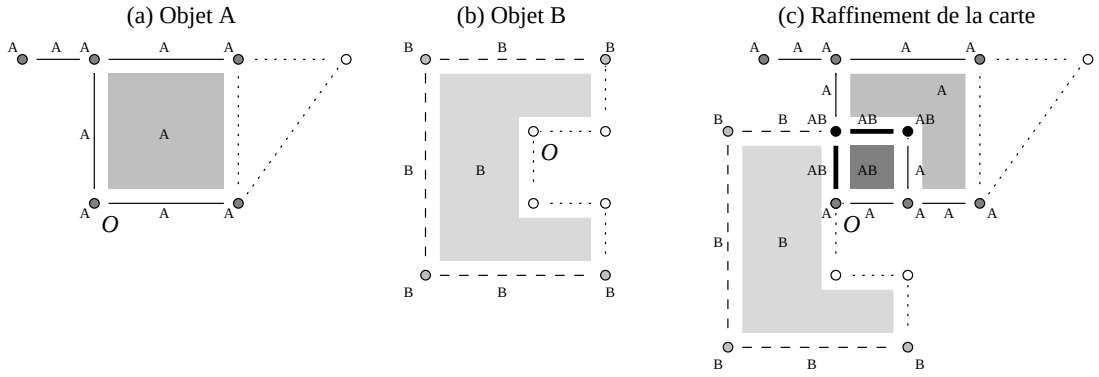


FIG. 8.14: *Schématisation des labels avant et après le raffinement*

Les figures 8.15, 8.16 et 8.17 représentent les labels des brins des cartes avant et après le raffinement, pour chaque dimension.

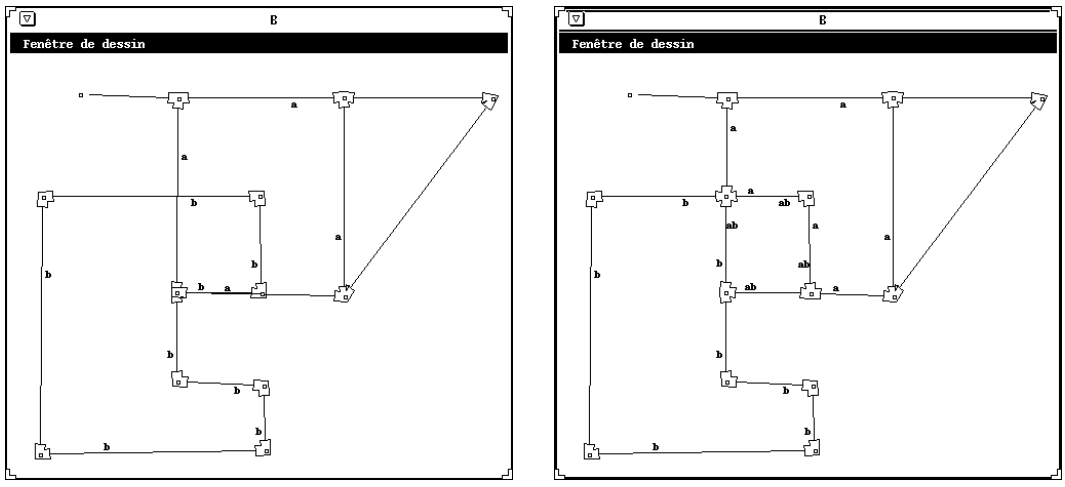


FIG. 8.15: *2-labels des faces avant et après raffinement*

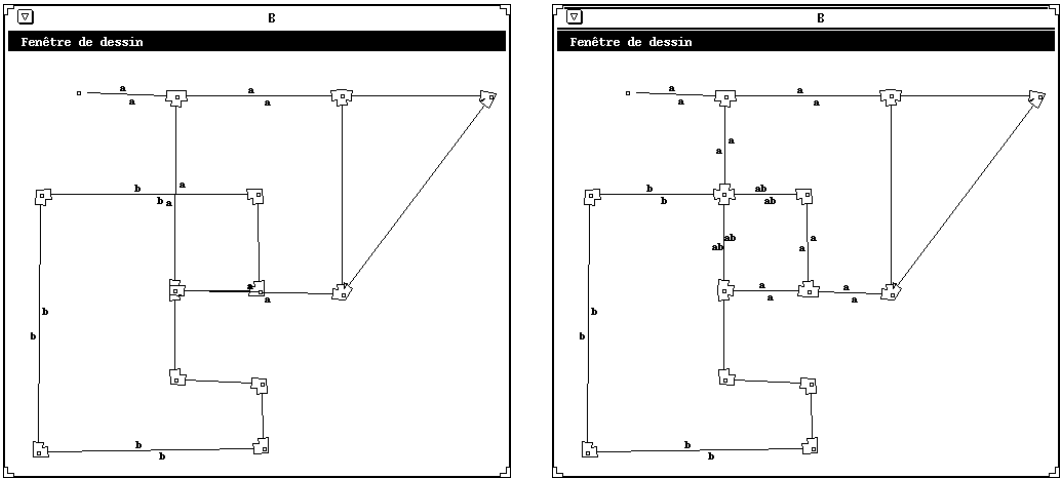


FIG. 8.16: *1-labels des arêtes avant et après raffinement*

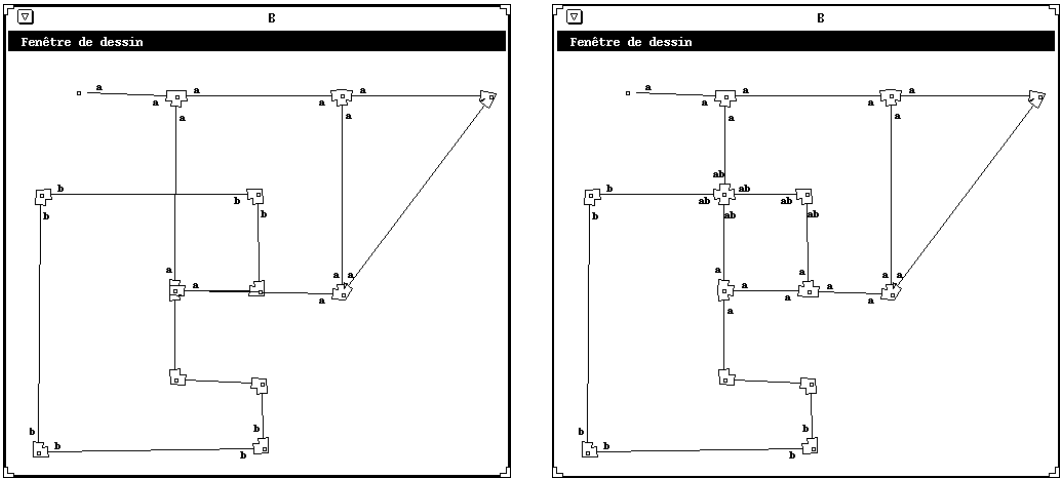


FIG. 8.17: *0-labels des sommets avant et après raffinement*

8.5 Raffinement en dimension 3

Le premier exemple en dimension 3 est issu d'une expérimentation du raffinement en géologie, dans le cadre d'études menées dans notre équipe avec le Laboratoire de Détection et Géophysique du CEA (Bruyères-le-Châtel) et le consortium GOCAD (Nancy). Un ensemble de failles du sous-sol est modélisé par des faces sécantes d'une 3-carte. Elles proviennent de données réelles d'un échantillonnage effectué sur le terrain, comme dans la figure 8.18. Elles correspondent à des séparations entre couches géologiques et à des failles. Grâce au raffinement, on obtient une subdivision volumique du sous-sol correspondant en première approximation aux différentes masses géologiques réelles. Cette subdivision permet, ensuite, d'utiliser les relations topologiques construites, pour réaliser la localisation efficace d'un très grand nombre de points [39].

Le raffinement apparaît dans la figure 8.19, où on peut noter un grand nombre de faces pendantes qui font, en fait, partie du volume extérieur. Les volumes intérieurs déterminés après le raffinement sont mis en évidence par trois vues distinctes à droite de la figure. L'utilisation du raffinement réalisée sur cet exemple est une généralisation de travaux similaires déjà réalisés dans le domaine de la géologie [48].

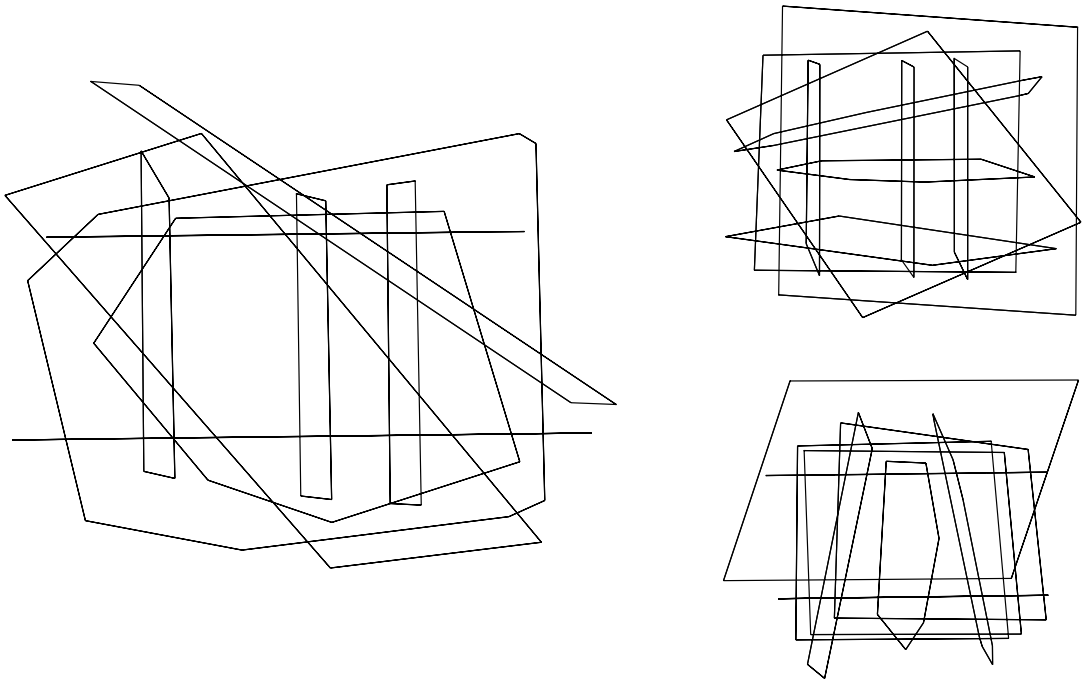


FIG. 8.18: *Un ensemble de faces quelconques : trois points de vue distincts*

La figure 8.20 montre le raffinement où nous avons mis en évidence les volumes internes par l'utilisation de niveaux de gris distincts.

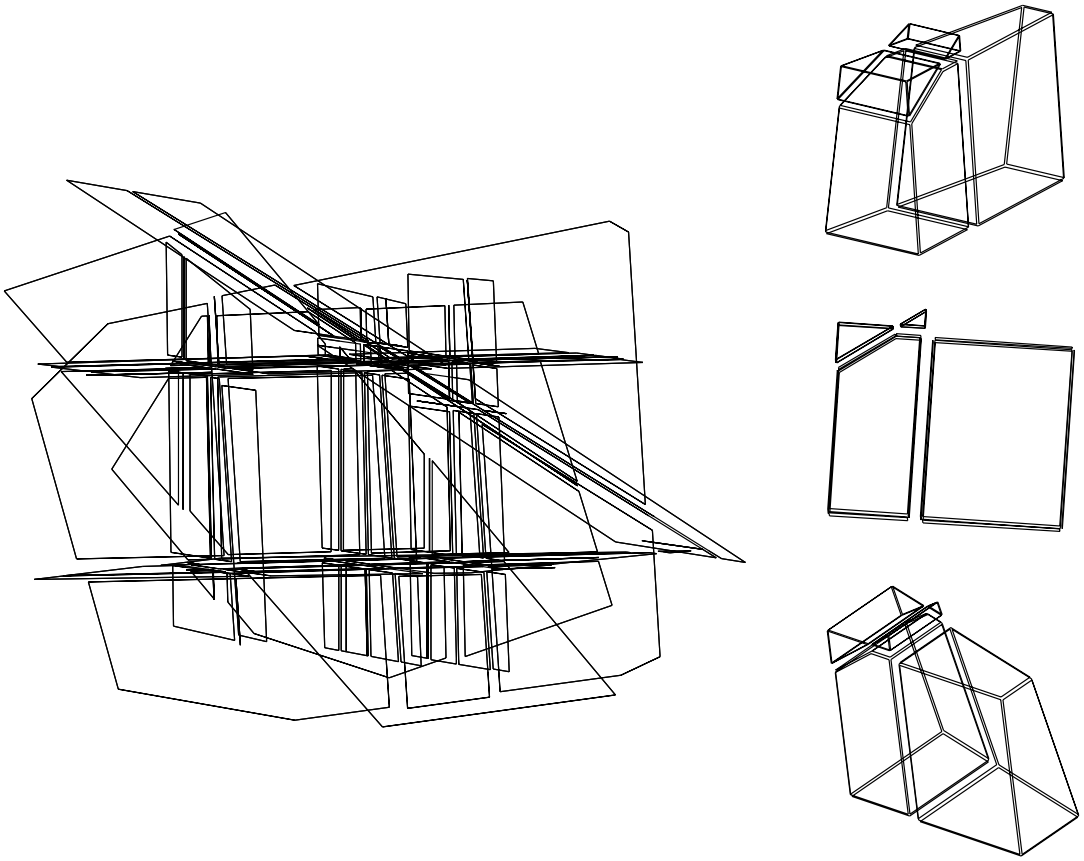


FIG. 8.19: *Leur raffinement et les vues des volumes définis*

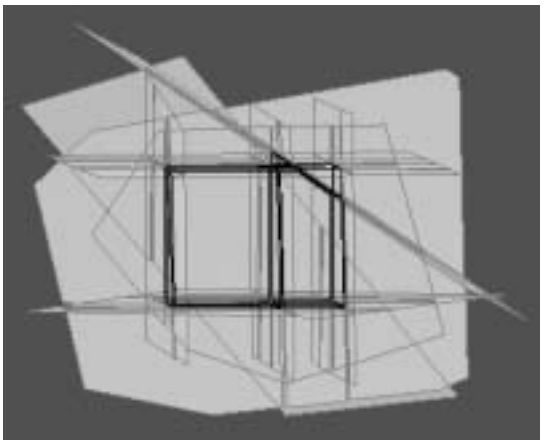


FIG. 8.20: *Une autre vue du raffinement*

Les deux figures suivantes montrent un exemple de raffinement plus complexe, toujours dans le domaine de la géologie. Les couches du terrain sont modélisées par un ensemble de surfaces obtenues par interpolation de données réelles. Un cube englobant a été ajouté pour fermer la portion du sous-sol étudiée. Le raffinement permet de construire les volumes définis par ces différentes couches géologiques. Notons que la robustesse du raffinement est cruciale ici, puisque le cube englobant doit être découpé le long du bord d'un grand nombre de faces. La figure 8.22 montre les volumes ainsi obtenus, en niveau de gris. Nous avons effacé le volume extérieur modélisé par le cube pour clarifier la figure.

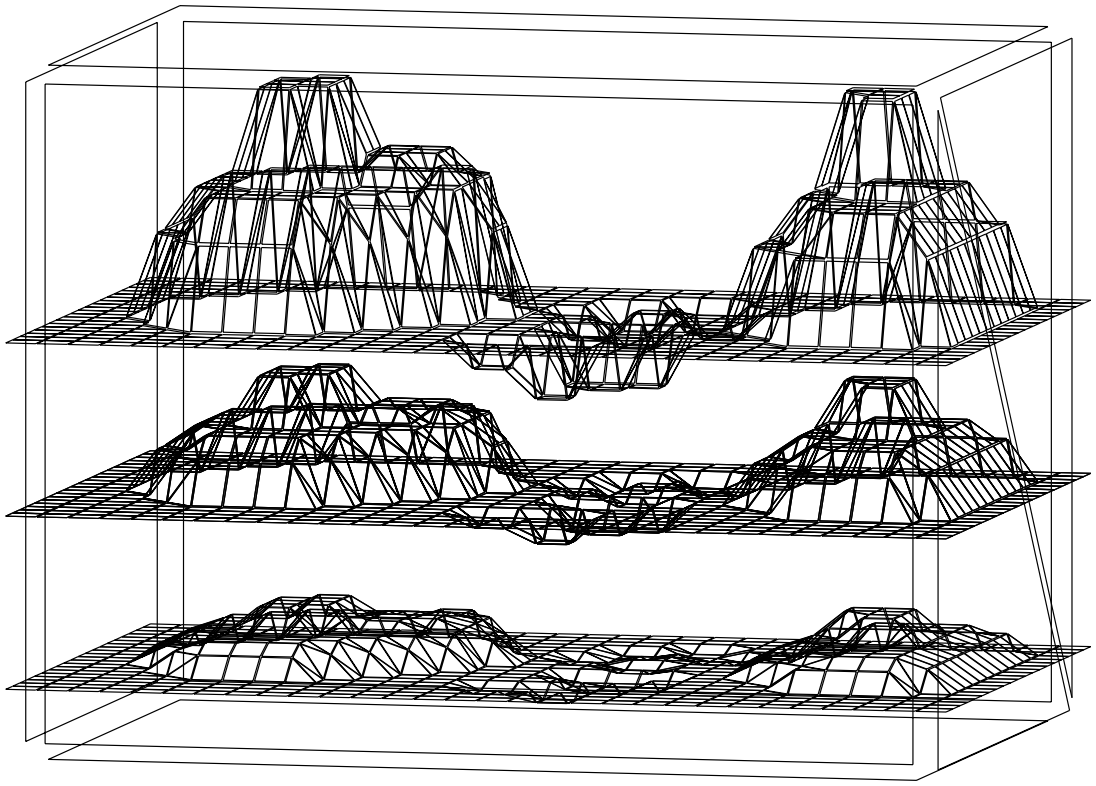


FIG. 8.21: *Trois couches géologiques modélisées par des surfaces*

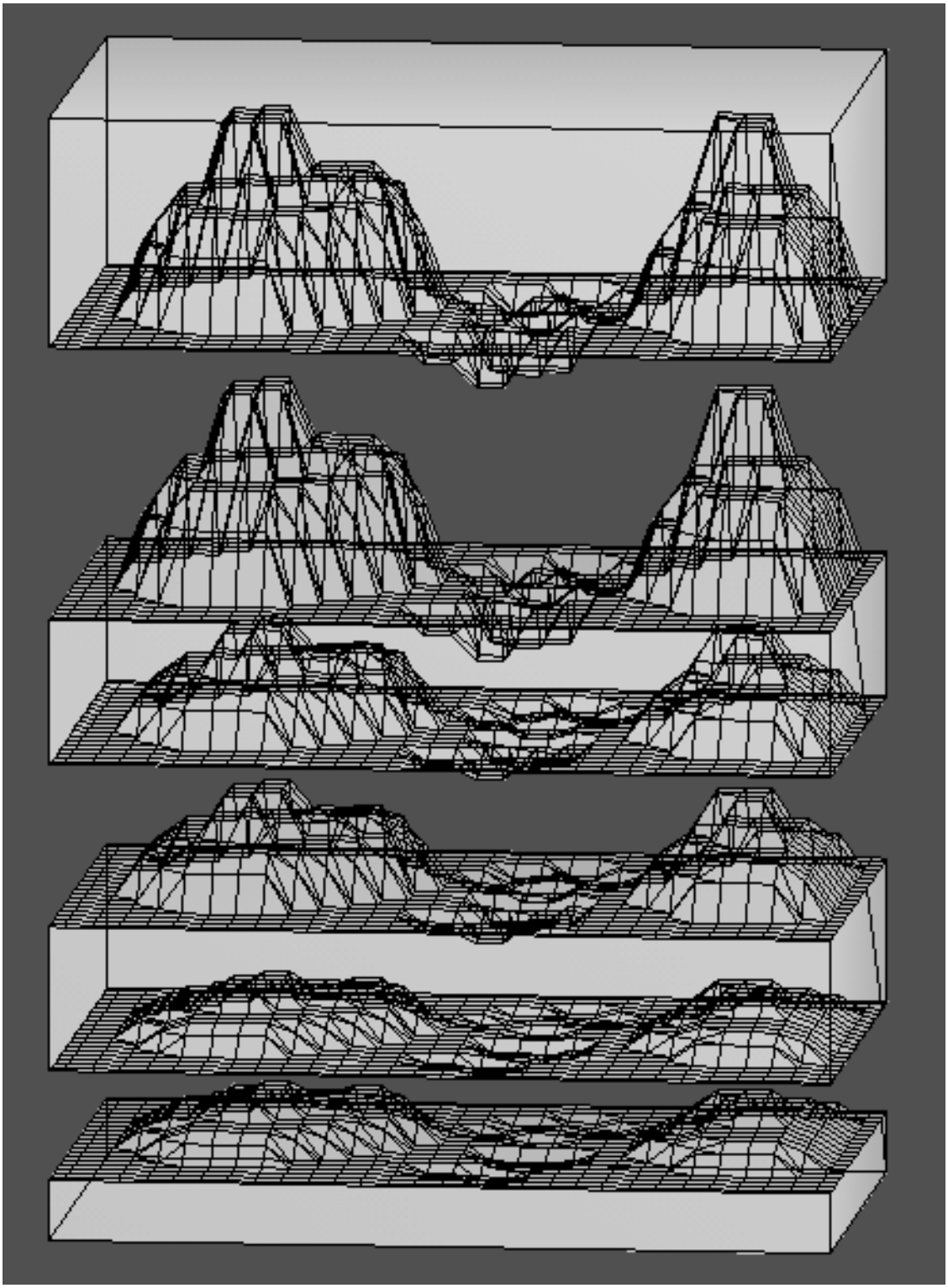


FIG. 8.22: *Les quatre volumes obtenus après raffinement*

8.6 Raffinement de solides polyédriques

Dans cette seconde série d'exemples, nous montrons le raffinement de solides polyédriques classiquement utilisés en CAO. La figure 8.23 représente trois parallélépipèdes sécants, avec une vue en perspective et deux vues de coté. Il faut noter que cette 3-carte comporte de nombreuses faces superposées et que le pavé long passe à l'intérieur de l'un des deux cubes sans couper le bord de ses faces.

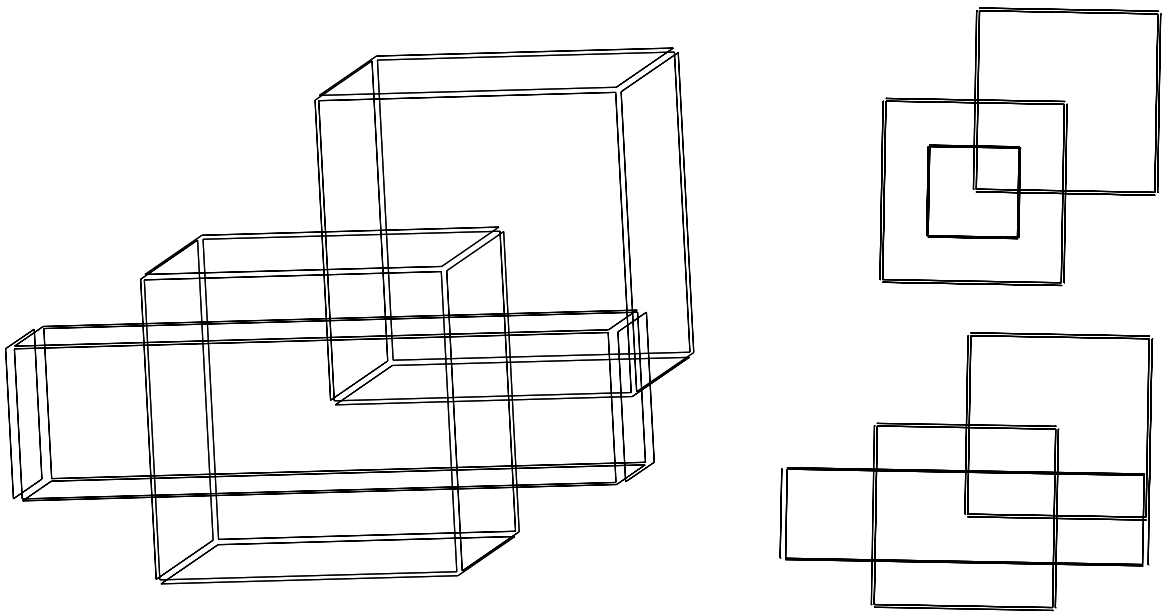


FIG. 8.23: *Une carte comportant plusieurs composantes connexes polyédriques*

Les deux figures suivantes, 8.24 et 8.25, montrent le raffinement 3D correspondant. La première est une vue filaire dans laquelle les faces sont légèrement éclatées. Dans la seconde, nous avons coloré les faces et laissé apparaître par transparence les volumes intérieurs. Il convient de noter que les trous dans les faces du cube mentionné précédemment sont modélisés grâce à l'ajout d'arêtes liant le bord extérieur d'une face au bord du trou situé à l'intérieur de celle-ci.

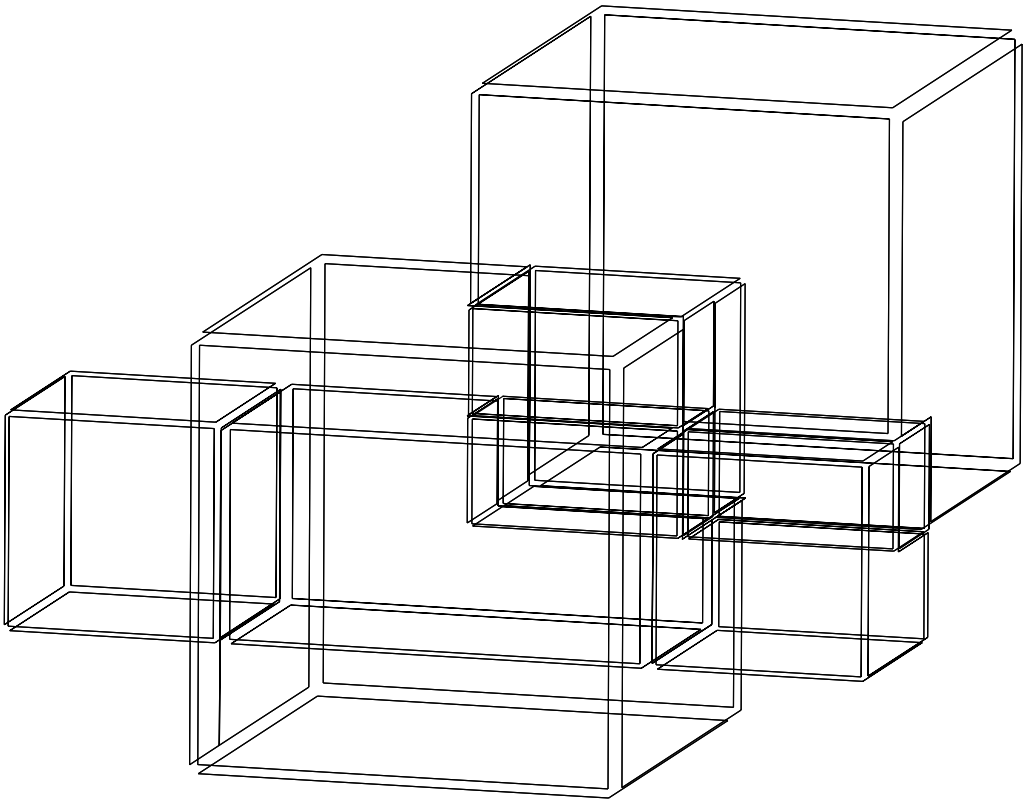


FIG. 8.24: *Son raffinement*

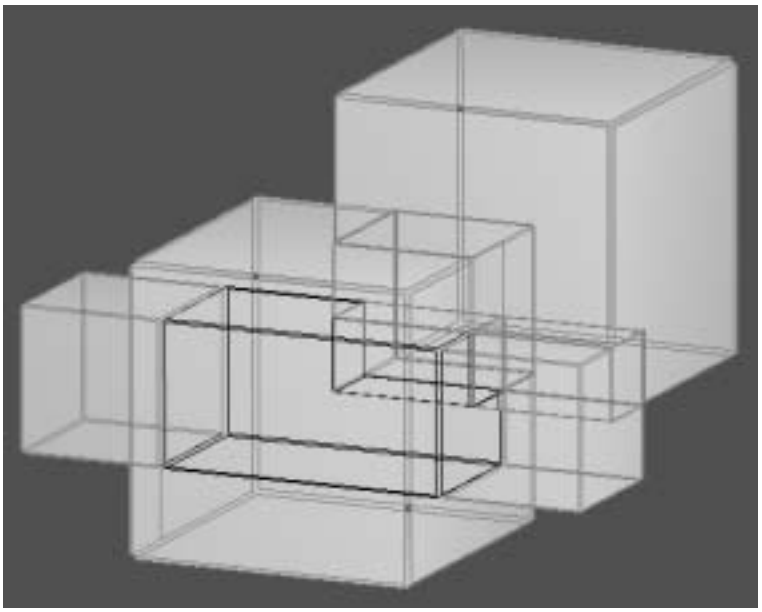


FIG. 8.25: *Une autre vue du raffinement*

La figure 8.26 représente le volume extérieur que nous avons ouvert et colorié pour mettre en évidence les faces qui le composent. La figure 8.27 montre les volumes intérieurs définis par le raffinement et déplacés pour qu'ils soient tous visibles. On peut noter que le gros volume cubique apparaissant en bas de la figure contient un trou qui a été formé par le pavé long mentionné plus tôt. L'intérieur du trou est rempli par le volume plus petit dessiné juste au-dessus et qui est un bout du pavé long qui a été découpé. Pour ces deux figures nous avons effectué une rotation qui procure un meilleur angle de vue.

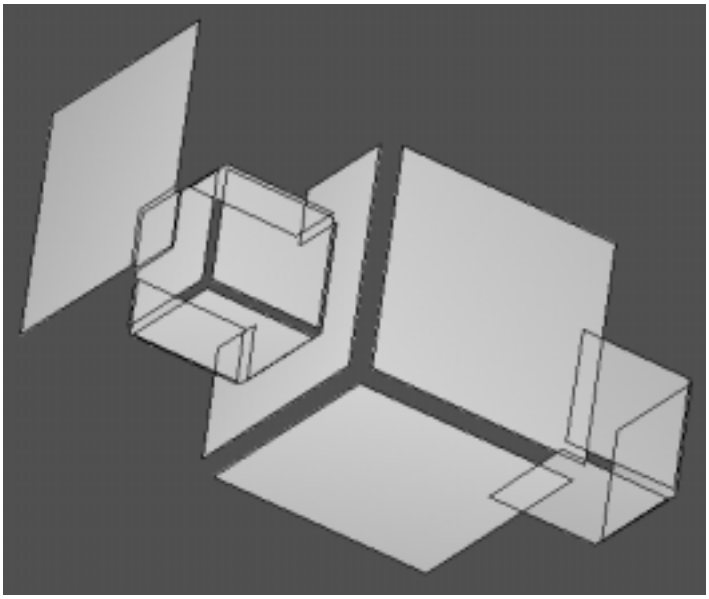


FIG. 8.26: *Vue du volume extérieur éclaté et ouvert*

Les quatre dernières figures représentent le raffinement d'une carte contenant une sphère et un tore modélisés par un très grand nombre de faces. Nous donnons trois vues différentes du raffinement dans lesquelles nous avons mis en évidence les volumes internes. Dans la dernière figure, nous montrons séparément les quatre volumes définis dans le raffinement.

Dans le tableau ci-dessous, nous avons indiqué les temps de calcul pour les différents exemples de raffinement en dimension 3, ainsi que le nombre de brins avant et après le raffinement. Ces temps de calcul nous semblent corrects étant donné que nous n'avons pas, pour l'instant, implanté d'optimisation du parcours des brins. Cela ce voit peu pour des objets contenant beaucoup d'intersections par rapport au nombre de faces, mais devient génant dans le cas du raffinement de surfaces dont les faces ne se coupent pas. Nous utilisons des boîtes englobantes pour éviter de rechercher l'intersection de deux faces dont les boîtes ne se coupent pas. Il suffirait, pour réduire les temps de calcul, d'effectuer un tri de ces boîtes et de n'examiner que les couples de faces dont les boîtes ne sont pas séparées par ce tri.

	nombre de brins		temps
	avant	après	(en secondes)
parallélépipèdes	340	452	0.5
failles	138	1098	2.7
tore & sphère	12720	13360	15
couches surfaciques	16380	19444	112

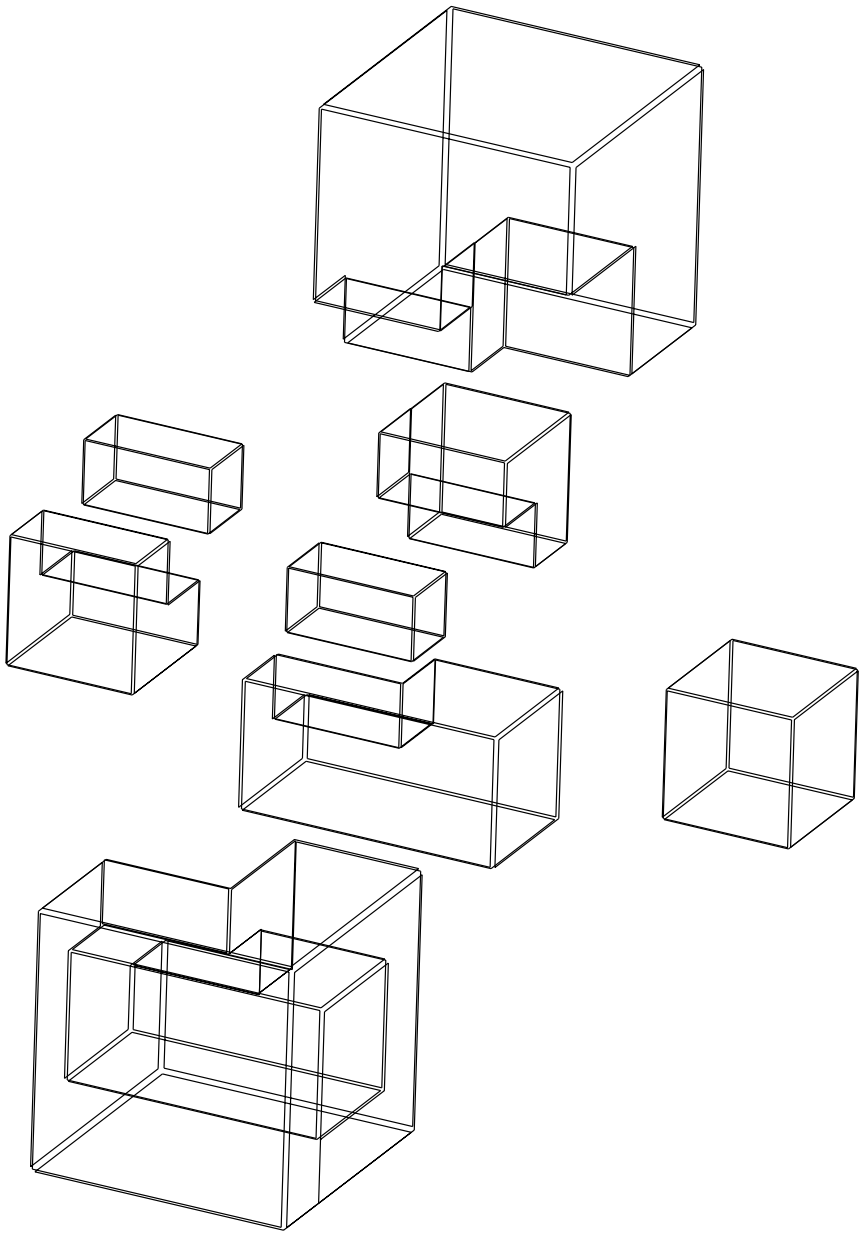


FIG. 8.27: *Vue éclatée des volumes intérieurs*

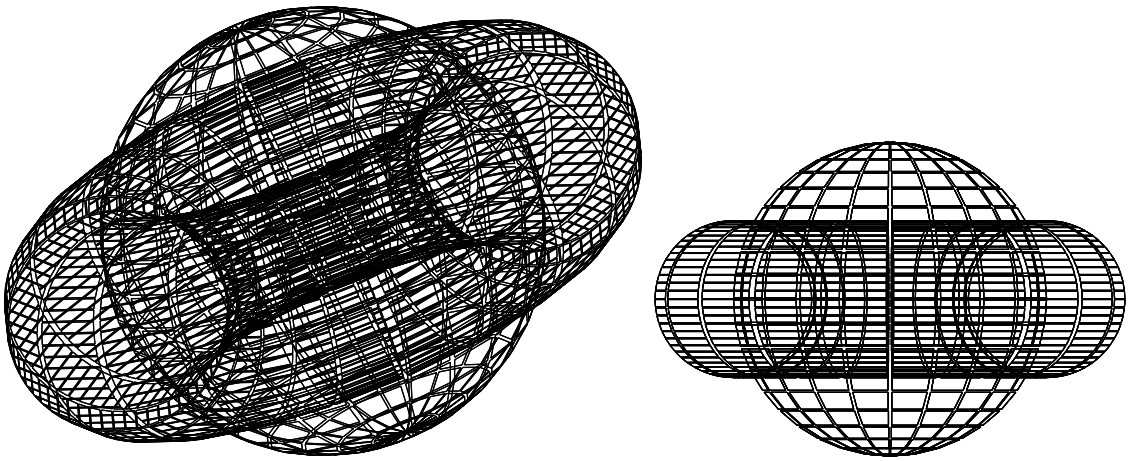


FIG. 8.28: *Une carte modélisant un tore et une sphère*

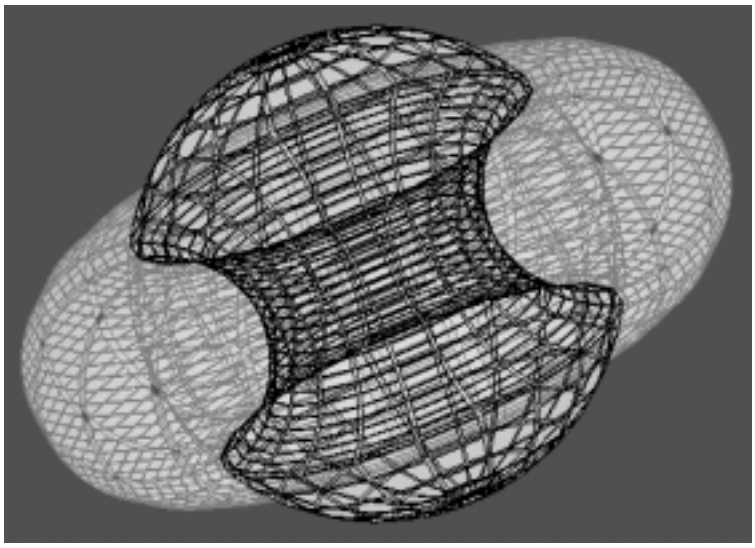


FIG. 8.29: *Une vue du raffinement*

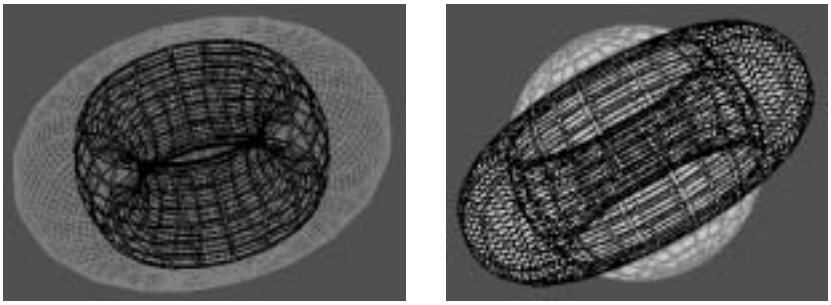


FIG. 8.30: *Deux autres vues du raffinement*

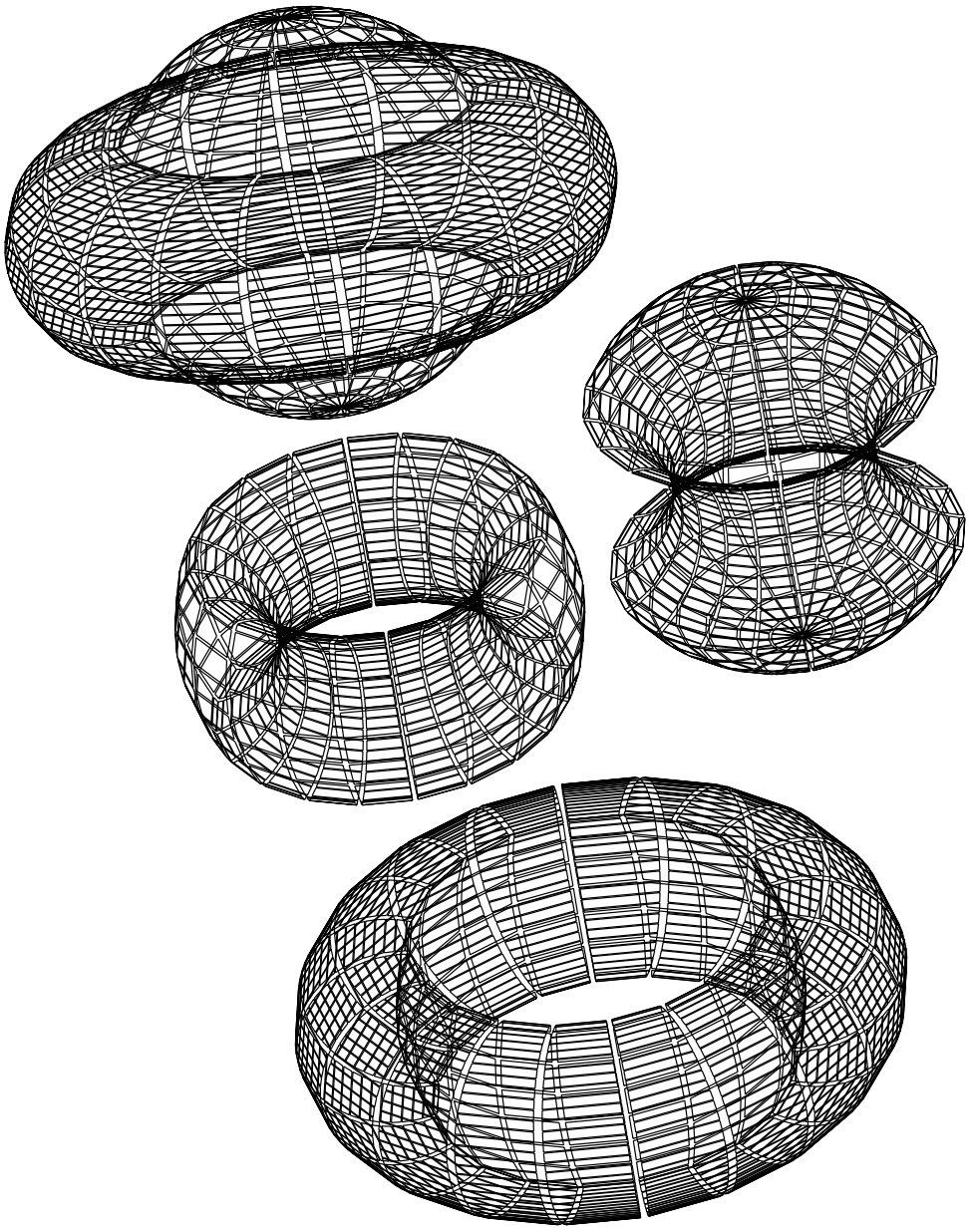


FIG. 8.31: *Vue éclatée des volumes du raffinement*

Chapitre 9

Conclusion

Le résultat principal de ce mémoire se situe dans l'expérimentation et la combinaison de méthodes formelles, notamment les spécifications algébriques et la réécriture, appliquées à la modélisation mathématique des objets géométriques et à toutes les phases de la conception de programmes réalisant des opérations booléennes sur ces objets. Les méthodes formelles utilisées recouvrent l'analyse mathématique et statique du problème, les preuves de certaines propriétés logiques des processus étudiés, la prise en compte des problèmes de complexité des calculs et l'étude d'optimisations des stratégies de parcours, jusqu'au prototypage et à l'implantation des programmes correspondants. Dans ce qui suit, nous montrons l'intérêt et les limitations de cette étude, ainsi que les perspectives qu'elle suggère, en examinant les différents domaines abordés.

9.1 Modélisation géométrique

Nous avons proposé une extension des cartes combinatoires par l'ajout de labels. Celle-ci nous a permis de modéliser au sein d'une même carte un ensemble d'objets géométriques ouverts ou fermés, dont les bords peuvent être incomplets, contenant des trous ou des cellules isolées et pendantes. A partir de ce modèle, nous avons défini des opérations booléennes très générales et dans leur version non régularisée.

Cette extension qui hérite naturellement des avantages liés au modèle topologique des cartes possède de plus la puissance d'expression propre à la modélisation par des arbres CSG. Nous avons, en effet, montré qu'un arbre CSG dont les primitives sont modélisées par des cartes, peut être représenté par une carte labellée et une expression booléenne sur les identificateurs de ces primitives. De plus, ce modèle est suffisamment général pour permettre la représentation d'arbres CSG avant et après évaluation des opérations booléennes, et nous avons montré que l'on pouvait extraire par une simple sélection des cellules concernées le résultat d'une évaluation.

En pratique cependant, l'utilisation complète de ce modèle pose des problèmes liés à l'ergonomie et aux limites de nos interfaces graphiques. En effet, nous avons eu des difficultés pour représenter graphiquement l'ensemble des labels d'une carte et nous avons été contraints de réaliser un affichage distinct pour chaque dimension de label. Par contre, ces problèmes disparaissent si l'on se limite aux objets régularisés qui ne nécessitent qu'un unique label.

Il est ainsi possible de limiter l'utilisation de tous les labels à des cas bien spécifiques où ils s'avèrent pertinents. Dans ce cadre, il serait intéressant d'étudier des représentations plus intuitives des labels, peut-être par l'utilisation de couleurs ou d'autres modes d'affichage tels que des pointillés ou des hachures.

Une perspective importante de notre approche basée sur le raffinement de cartes labellées serait l'étude et le développement d'un modeleur à base topologique, offrant les opérations de haut niveau que l'on trouve en général dans les logiciels à base de CSG, comme la gestion d'ensembles d'objets et la manipulation intuitive et l'édition d'expressions booléennes sur ceux-ci. Un tel logiciel devrait bénéficier des avantages du modèle topologique de représentation par les bords fourni par les cartes, et notamment des possibilités de sélections interactives, de modifications locales ou de rendus rapides comme la gestion des faces cachées.

9.2 Evaluation d'opérations booléennes

Du point de vue des opérations booléennes, l'utilisation de ce modèle topologique apporte deux avantages essentiels. Le premier lié directement à l'utilisation des labels est la possibilité d'effectuer le raffinement simultané d'un nombre quelconque d'objets et d'évaluer, après coup, toute expression booléenne sur ces objets. Le deuxième avantage, moins visible, mais peut-être plus intéressant, apparaît dans le mécanisme de complétion des labels.

Dans les méthodes classiques, après l'évaluation des cellules du résultat, que ce soit par raffinement ou par découpe et recollage du bord, l'évaluation d'une opération booléenne implique une classification des cellules permettant de savoir si chacune d'elles est à l'intérieur, sur le bord ou à l'extérieur du résultat [72]. Cette classification nécessite l'utilisation de structures supplémentaires telles que des représentations de voisinages [66] ou des normales [64], pour savoir où se situe l'intérieur des objets. Parfois, elle implique également des calculs de localisations géométriques. De plus, elle se révèle souvent difficile à mettre en œuvre lorsque plus de quatre arêtes (ou faces) sont incidentes à un même sommet (ou à une même arête), car elles restent souvent définies par des heuristiques approximatives ou incomplètes.

Dans notre cas, les relations topologiques sont mises à profit pour réaliser une propagation des labels à travers toute la carte. Cette transformation globale est réalisée à partir de transmissions locales des labels autour des sommets, en dimension 2, ou des arêtes, en dimension 3, et sans aucun calcul numérique. Comme nous l'avons vu sur les exemples, cette méthode permet de déduire si un objet est situé à l'intérieur d'un autre, uniquement d'après les relations d'incidence et d'adjacence et grâce à l'utilisation de l'orientation des cellules. Cette technique s'avère très intéressante, puisqu'elle s'applique même dans le cas où les bords de ces objets ne s'intersectent pas.

Nous avons ainsi décrit complètement et de manière formelle le raffinement des cartes plongées et labellées. Dépassant le problème du raffinement de subdivisions planaires ou volumiques, cette opération est en fait un procédé général de normalisation de subdivisions volumiques. Nous avons vu, par des exemples, que le raffinement réalise une extension des problèmes bien connus de cloisonnements ou d'arrangements de segments ou de plans.

Nous avons notamment utilisé le raffinement en dimension 3, pour construire des subdivisions volumiques à partir d'un ensemble de faces ou de surfaces. Cet aspect du raffinement s'est révélé intéressant pour aborder efficacement des problèmes propres à la géologie. Une

étude est d'ailleurs en cours, à Strasbourg en collaboration avec le Laboratoire de Détection et Géophysique du CEA (Bruyères-le-Châtel) et le consortium GOCAD (Nancy), pour mettre à profit les informations topologiques obtenues durant le raffinement afin de réaliser des localisations rapides d'un très grand nombre de points dans une subdivision volumique du sous-sol [39].

Le raffinement tel que nous l'avons vu est réalisé sur une unique carte modélisant un nombre quelconque d'objets, souvent dans des composantes connexes distinctes. Cet aspect est particulièrement utile dans le cadre de restructurations topologiques, où les objets initiaux sont quelconques, voire même incomplets. Mais cela peut parfois être un handicap lorsque l'utilisateur sait par avance qu'un nombre important d'objets initiaux ne se coupent pas ou ne contiennent pas d'intersections.

Imaginons, par exemple, que l'on veuille réaliser un grand nombre de trous dans une plaque. Les trous sont modélisés par des volumes cylindriques et le perçage correspond à la différence ensembliste entre la plaque et ces cylindres. Il est probable que les cylindres ne se chevauchent pas et que seules les intersections entre chaque cylindre et la plaque sont pertinentes.

Il serait intéressant de pouvoir gérer ce type d'information permettant de limiter les calculs durant le raffinement. Nous utilisons déjà en pratique des boîtes englobantes sur les faces, pour éviter de rechercher l'intersection entre chaque couple de faces. On peut imaginer une hiérarchie de boîtes englobantes rendant compte des connaissances que peut avoir un utilisateur des intersections possibles.

Il serait également intéressant d'utiliser d'autres connaissances topologiques, comme l'ensemble des faces d'un volume, pour hiérarchiser par défaut les boîtes englobantes. Dans le cadre d'un modeleur utilisant le raffinement, les objets modélisés par des composantes connexes d'une carte sont soit des primitives, soit obtenus par des raffinements précédents. Dans ces deux cas, on peut assurer qu'il n'existe plus d'intersection au sein de ces objets et donc gérer cette nouvelle information par un mécanisme de boîtes définies également par défaut.

9.3 Spécifications algébriques et systèmes de réécriture

Nous avons donné une spécification algébrique complète des opérations permettant la construction de cartes combinatoires et leur manipulation, notamment en ce qui concerne leur plongement et la gestion des labels. Cette spécification a été définie par des extensions successives de petits modules. Bien que les axiomes apparaissant dans les spécifications restent parfois difficiles à lire, il nous semble que cette approche modulaire facilite la compréhension générale et intuitive des fonctionnalités décrites.

De plus, en pratique, cette approche se montre particulièrement efficace lors du prototypage et, plus encore, de l'implantation. En effet, les générateurs de base et les sélecteurs correspondants forment un noyau de base du programme manipulant directement les structures de données. Les autres opérations sont définies par couches successives réalisant des appels au noyau, puis aux couches précédentes, ce qui est bien adapté à une programmation propre, surtout pour un logiciel de grande taille.

La partie la plus originale reste cependant l'utilisation de la réécriture. Celle-ci nous a permis d'exprimer les raffinements, en dimension 2 et 3, qui sont des problèmes algorithmiques complexes, sous forme de transformations élémentaires et indépendantes. Ce résultat

est obtenu grâce au fait que le formalisme choisi permet une abstraction totale des structures de données utilisées et des problèmes propres à l'implantation efficace des algorithmes correspondants. La simplicité apparente de la description naïve du raffinement en dimension 2 en témoigne.

En dimension 3, la simplicité est moins évidente à première vue, car le cheminement de la définition de l'intersection de faces reste long et quelque peu fastidieux. Cette complexité est bien sûr due aux difficultés propres à ce problème, mais elle est surtout due au fait que nous avons décrit rigoureusement et complètement tous les cas possibles, avec un grand souci du détail. Ainsi, si cette définition formelle reste complexe, en revanche, le passage à une implantation devient, lui, immédiat, puisque toutes les zones d'ombre dues aux cas particuliers à prendre en compte sont résolues avant celui-ci.

Pour le raffinement en dimension 2, nous avons démontré la terminaison et la confluence du système de réécriture. De plus, grâce à l'ajout de structures de contrôle, nous avons pu dériver un ensemble de systèmes de réécriture décrivant des stratégies de plus en plus efficaces pour le parcours des brins d'une carte. Nous avons tout d'abord montré comment des structures de contrôle pouvaient être utilisées pour fixer une stratégie. Puis, en utilisant des propriétés géométriques correspondant aux conditions de déclenchement des règles, nous avons enrichi pas à pas les systèmes de réécriture, jusqu'à l'obtention d'une stratégie bien connue de balayage du plan.

Ainsi, nous pensons avoir montré que le formalisme des spécifications allié à l'expressivité de la réécriture facilite la conception propre et rigoureuse d'algorithmes abstraits. Ce formalisme est aussi un point de départ sain pour la conception d'algorithmes concrets efficaces, où la part est faite nettement entre les questions de logique et celles de contrôle. Notre démarche n'est donc en aucun cas un handicap pour l'obtention d'une complexité optimale.

Des méthodes formelles similaires ont déjà été utilisées pour étudier des algèbres de graphes [21] ou définir des grammaires sur des hypercartes et des algorithmes de graphes [69]. La réécriture de graphe a également été abordée dans ce cadre [23, 22] et il serait intéressant d'étudier les implications possibles de ces approches dans le cadre de la modélisation géométrique. Il faut noter cependant que ces travaux sont basés sur des analyses combinatoires et algébriques, alors que dans notre cas les problèmes liés aux plongements sont essentiels puisque ceux-ci servent à guider la réécriture.

Deux types de prolongement de l'utilisation des méthodes formelles nous semblent intéressants. En premier lieu, nous avons montré que les méthodes proposées ici s'appliquent bien à l'étude de problèmes complexes en géométrie algorithmique. Il serait intéressant d'aborder avec une approche similaire d'autres types d'optimisation comme les stratégies du type *diviser pour régner* ou *mariage avant conquête* [61].

Ces méthodes s'annoncent prometteuses pour d'autres domaines de la géométrie algorithmique, comme la recherche d'enveloppes convexes [61], la construction de diagrammes de Voronoï [44] ou les problèmes de localisation. En effet, tous ces problèmes utilisent des modèles géométriques et des opérations similaires à ceux développés pour ce travail. L'analyse des algorithmes correspondants gagnerait certainement en clarté et en rigueur, si elle était abordée par le biais de la réécriture et de l'utilisation de structures de contrôle bien séparées des objets géométriques traités.

En deuxième lieu, nous avons donné les preuves de certaines propriétés logiques des opé-

rations et processus décrits. Ces preuves ont été réalisées à la main et sont, nous l'avons vu, longues, fastidieuses et souvent répétitives. Il serait intéressant d'utiliser des systèmes d'aide à la preuve, comme Coq [50], pour les réaliser, dans la lignée de [68, 63].

De plus, nous n'avons pas abordé les problèmes liés aux preuves de correction des spécifications algébriques vis-à-vis des spécifications mathématiques. Il serait utile, surtout pour les opérations topologiques de base, de démontrer plus formellement que les modèles définis par les spécifications algébriques vérifient les contraintes et propriétés définies mathématiquement pour les cartes.

Enfin, à l'autre bout de la chaîne, nous avons vu que les méthodes formelles permettaient une implantation finale propre et rigoureuse des programmes étudiés. Cependant, le problème de la correction du programme vis-à-vis de sa spécification sous forme de systèmes de réécriture reste un problème ouvert.

9.4 Approximations numériques

Nous n'avons pas utilisé de méthodes numériques spécifiques pour gérer les questions d'approximations et d'arrondis, puisque tel n'était pas notre but premier. Cependant, par quelques remarques, nous avons montré que ces problèmes étaient bien localisés. Précisément, les problèmes numériques n'interviennent que lors de l'évaluation des conditions de déclenchement des règles et, pour le raffinement en dimension 3, dans la définition du tri le long de la droite d'intersection de deux faces non coplanaires. Autrement dit, l'utilisation d'une arithmétique différente n'affecterait que ces parties et serait bien sûr sans effet sur la partie topologique.

De plus, nous avons montré que des informations topologiques pouvaient être mises à profit pour résoudre les incohérences dues aux approximations numériques. Il faut cependant avouer que cette approche à des limitations. Nous avons constaté, en pratique, que le choix du seuil ε est crucial. Un ε trop grand implique des confusions importantes dues au fait que des cellules petites ou fines se trouvent aplaties ou réduite à un amalgame de sommets et limite ainsi les possibilités de déduction à partir des propriétés topologiques. Au contraire, un ε très petit provoque une trop grande sensibilité aux limites de la précision numérique des nombres flottants.

Pour nos expérimentations, les techniques de rectification des incohérences numériques proposées se sont révélées suffisantes. Mais, dans le cadre du développement d'un logiciel de plus grande envergure, d'autres solutions devraient être envisagées en complément de l'utilisation de la topologie. Par exemple, l'arithmétique rationnelle paresseuse de [7] semble séduisante, même si dans le cadre d'un modeleur, son utilisation peut soulever d'autres difficultés pour certaines opérations, comme la définition de rotations rationnelles ou la désignation précise de points d'incidence.

9.5 Prototypage logique et implantation

Les opérations sur les cartes et les règles de réécriture ont été implantées en Prolog. Ce prototypage logique nous a permis de vérifier rapidement par une mise en pratique, mais sur des jeux de données réduits, la validité de nos spécifications. Le prototype nous a également

permis d'étudier, en pratique, différents types de stratégies pour une exécution efficace des systèmes de réécriture.

Durant la phase d'analyse et de conception, nous avons développé en même temps les spécifications et le prototype, et réalisé de nombreux allers et retours entre ces deux écritures. Cette méthodologie a tout d'abord facilité la formalisation, tout en nous assurant que les spécifications restaient suffisamment explicites. En effet, pour formaliser un problème, il est utile de se confronter à des cas concrets, car cela permet d'appréhender plus intuitivement les difficultés à résoudre. De plus, le prototypage nous a aidé à hiérarchiser l'ensemble des opérations et à diminuer leur nombre, pour obtenir un ensemble cohérent et réduit aux fonctionnalités strictement nécessaires.

Finalement, nous avons réalisé l'implantation en C du raffinement de cartes combinatoires en dimension 2 et 3. Comme nous l'espérions, le passage des spécifications formelles à leur implantation s'est avéré très facile et quasiment immédiat. En effet, les difficultés théoriques ont été complètement résolues et abordées dans le détail durant la phase de spécification. Les problèmes liés à l'implantation, comme le choix des structures de données, l'introduction de variables intermédiaires ou les combinaisons d'appels de fonctions ont été résolus presque totalement lors du prototypage.

Ainsi lors de l'implantation, nous avons pu nous concentrer sur les difficultés propres à la programmation en C, tels que la gestion des pointeurs ou les problèmes d'allocation de mémoire. Un résultat notable est que les programmes obtenus n'ont nécessité pratiquement aucun déboggage.

Nous nous sommes limité à l'implantation des différents raffinements et, faute de temps et parce que ce n'était pas notre but premier, nous n'avons donc pas réalisé d'interface graphique permettant la saisie et la gestion d'un ensemble d'objets. En pratique, nous avons utilisé le modeleur Topofil pour créer des cartes que nous avons utilisées ensuite par l'intermédiaire de fichiers.

Comme nous l'avons déjà mentionné, il serait intéressant de concevoir un logiciel de modélisation complet permettant notamment la gestion des labels d'un ensemble d'objets, d'expressions booléennes sur ces objets et des extractions de résultat à partir du raffinement. Un tel logiciel devrait proposer des fonctionnalités similaires à celles qui sont proposées dans le cadre de logiciels basés sur le modèle CSG.

Annexe A

Spécifications des objets géométriques en dimension 2

Les objets géométriques et les opérations définis ici font, bien sûr, appel à des calculs numériques. Différents types d'arithmétiques sont possibles en ce qui concerne, d'une part, la nature des nombres retenus pour les coordonnées et, d'autre part, la façon de définir les tests de nullité ou d'égalité. Nous avons voulu présenter une spécification des objets géométriques indépendante du type d'arithmétique choisi. Ainsi, les spécifications qui suivent étendent toutes une spécification FLOAT contenant la définition des nombres et des opérateurs pour une arithmétique choisie.

Les nombres définis dans FLOAT peuvent être de différentes natures : des réels classiques, des rationnels, des nombres décimaux, des nombres à virgule flottante ou toute autre arithmétique plus évoluée comme les intervalles de nombres flottants [43] ou l'arithmétique rationnelle paresseuse décite dans [7].

Pour pouvoir évoquer les problèmes liés aux approximations numériques et aux erreurs d'arrondi, nous définissons deux types d'extensions de la spécification FLOAT suivant que l'arithmétique est exacte ou utilise un seuil ε pour l'égalité. Deux spécifications, 0-FLOAT et ε-FLOAT, peuvent ainsi être utilisées par les spécifications géométriques. Elles décrivent les tests de nullité pour un nombre x, noté $null(x)$, et pour le carré d'une norme n, noté $nulln(n)$. Notons que cette fonction $nulln$ est utlisée pour nous éviter l'utilisation de racine carrée qui allourdirait l'écriture.

Cette différenciation nous permet dans les spécifications suivantes d'utiliser une unique spécification x-FLOAT, où $x = 0$ ou $x = \varepsilon$, en faisant abstraction de la façon dont est définie la nullité. Les spécifications des objets géométriques qui suivent utiliseront donc l'une ou l'autre indifféremment, le choix étant repoussé au niveau du prototypage ou de l'implantation.

La spécification 0-FLOAT définit alors des nombres pour lesquels un test de nullité exact existe. Dans ce cas, les tests d'égalité peuvent être effectués sans approximation. Cette spécification s'applique en particulier aux rationnels et aux vrais réels.

La spécification ε-FLOAT définit, quant à elle, les nombres pour lesquels la nullité, et donc l'égalité, n'est vérifiable qu'à ε près. Elle s'applique en particulier aux décimaux et aux nombres à virgule flottante. Dans cette spécification, une constante ε est définie, mais notons que sa valeur exacte, indiquée ici à titre d'exemple, n'a pas besoin d'être spécifiée explicitement.

TAB. A.1: *Spécifications des nombres avec tests de nullité exacts*

Spéc *0-FLOAT* **étend** *FLOAT* **avec**

Opérateurs

$null : Float \longrightarrow Bool$	(*nullité d'un réél*)
$nulln : Float \longrightarrow Bool$	(*nullité du carré d'une norme*)

Axiomes $(x : Float)$

$null(x) = (x == 0)$

$nulln(x) = (x == 0)$

Fin

TAB. A.2: *Spécifications des nombres avec nullitée à ε-près*

Spéc ε*-FLOAT* **étend** *FLOAT* **avec**

Opérateurs

$\varepsilon : \longrightarrow Float$	(*seuil d'erreur pour l'égalité*)
$null : Float \longrightarrow Bool$	(*nullité d'un réél*)
$nulln : Float \longrightarrow Bool$	(*nullité du carré d'une norme*)

Axiomes $(x : Float)$

$\varepsilon = 0.001$

$null(x) = (|x| < \varepsilon)$

$nulln(x) = (x < \varepsilon^2)$

Fin

La spécification POINT-2D définit les points du plan engendrés par deux coordonnées, leur égalité et une relation d'ordre sur ceux-ci. Dans cette spécification, l'égalité des points est définie par le biais de la fonction *nulln*, de manière à être indépendante des nombres choisis. De la même façon, tous les tests d'égalité sont définis par ce biais dans les autres spécifications. Enfin, l'ordre défini sur les points est simplement l'ordre lexicographique sur leurs coordonnées.

TAB. A.3: *Spécifications des points de* $\mathbb{R}^2$

Spéc *POINT-2D* **étend** *x-FLOAT* **avec**

Sortes *Point*

Opérateurs

$gp : Float\ Float \longrightarrow Point$	(*générateur de points*)
$eqp : Point\ Point \longrightarrow Bool$	(*égalité de deux points*)
$<_p : Point\ Point \longrightarrow Bool$	(*relation d'ordre sur les points*)

Axiomes $(x, y, x', y' : Float)$

$eqp(gp(x, y), gp(x', y')) = nulln((x'\text{-}x)^2 + (y'\text{-}y)^2)$

$gp(x, y) <_p gp(x', y') = (x < x') \vee (x == x' \wedge y < y')$

Fin

La spécification VECTEUR-2D définit les vecteurs du plan engendrés par deux coordonnées ou par deux points, le test de nullité, la norme d'un vecteur, les produits scalaire et vectoriel, la différence et le produit par un réél. Notons que le produit vectoriel est ici non pas un vecteur, mais un nombre qui est utilisé pour obtenir le sinus d'un angle ou son signe.

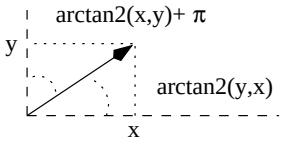


FIG. A.1: *Angle d'un vecteur*

Elle définit également la translation d'un point par un vecteur et l'angle d'un vecteur avec l'axe des y, cet angle étant compris entre 0 et 2π. La fonction $atan2(y, x)$ est la fonction classique donnant l'angle, entre $-\pi$ et π, du point de coordonnées x et y avec l'axe des x dans le plan. La figure A.1 représente ces deux angles.

TAB. A.4: *Spécifications des vecteurs de* $\mathbb{R}^2$

Spéc *VECTEUR-2D* **étend** *POINT-2D* **avec**

Sortes $Vecteur$

Opérateurs

$gv : Float\ Float \longrightarrow Vecteur$	(*générateur de vecteurs*)
$gvp : Point\ Point \longrightarrow Vecteur$	(*générateur de vecteurs*)
$nullv : Vecteur \longrightarrow Bool$	(*nullité d'un vecteur*)
$nv : Vecteur \longrightarrow Float$	(*norme d'un vecteur*)
$- : Vecteur\ Vecteur \longrightarrow Vecteur$	(*différence de deux vecteurs*)
$ps, pv : Vecteur\ Vecteur \longrightarrow Float$	(*produit scalaire, vectoriel*)
$. : Float\ Vecteur \longrightarrow Vecteur$	(*produit par un réél*)
$+ : Point\ Vecteur \longrightarrow Point$	(*translation d'un point*)

Axiomes $(\lambda, x, y, x', y' : Float)$

$gvp(gp(x,y), gp(x',y')) = gv(x'\text{-}x, y'\text{-}y)$
$nullv(gv(x,y)) = nulln(x^2 + y^2)$
$nv(gv(x,y)) = \sqrt{(x^2+y^2)}$
$gv(x,y)\text{-}gv(x',y') = gv(x\text{-}x', y\text{-}y')$
$ps(gv(x,y), gv(x',y')) = xx' + yy'$
$pv(gv(x,y), gv(x',y')) = xy'\text{-}x'y$
$\lambda.gv(x,y) = gv(\lambda x, \lambda y)$
$gp(x,y) + gv(x',y') = gp(x+x', y+y')$
$angle\_vect(gv(x,y)) = atan2(x,y) + \pi$

Fin

La spécification SEGMENT-2D définit les segments du plan engendrés par deux points et l'angle d'un segment qui est l'angle du vecteur formé par ses deux extrémités. De plus, nous définissons ici les opérations propres aux raffinements que sont le test d'incidence d'un point sur un segment, $incident(p, s)$, le test d'intersection de deux segments, $secant(s, t)$, et la recherche du point d'intersection de deux segments sécants, $intersection(s, t)$.

Ces fonctions font appels à des propriétés géométriques simples que nous ne commentons pas plus avant. Notons cependant que le test d'égalité de deux points et les deux derniers tests concernant les segments s'excluent mutuellement deux à deux. Ceci est nécessaire pour la confluence du raffinement et est assuré par l'utilisation systèmatique des fonctions *null* et *nulln*.

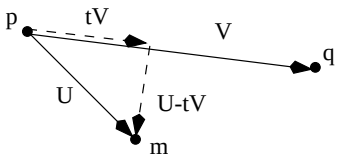


FIG. A.2: *Schématisation des vecteurs utilisisés pour le test d'incidence*

La figure A.2 représente graphiquement diverses variables utilisées dans la spécification de la fonction *incident*.

TAB. A.5: *Spécifications des segments de* $\mathbb{R}^2$

Spéc *SEGMENT-2D* **étend** *VECTEUR-2D* **avec**

Sortes *Segment*

Opérateurs

$gs : Point\ Point \longrightarrow Segment$	(*générateur de segments*)
$angle\_seg : Segment \longrightarrow Float$	(*angle d'un segment*)
$incident : Point\ Seg \longrightarrow Bool$	(*test d'incidence*)
$secant : Seg\ Seg \longrightarrow Bool$	(*test d'intersection*)
$intersection : Seg\ Seg \longrightarrow Point$	(*calcul de l'intersection*)

Axiomes $(m, p, q, p', q' : Point)$

$angle\_seg(gc(p,q)) = angle\_vect(gvp(p,q))$

$incident(m, seg(p,q)) =$ **si** $nullv(V)$ **alors** $faux$
sinon $(t > a) \wedge (t <$ 1-$a) \wedge nullv(U$-$t.V)$

avec $V = gvp(p,q), U = gvp(p,m), t = \dfrac{ps(V,U)}{ps(V,V)}, a = \dfrac{\varepsilon}{nv(V)}$

$secant(seg(p,q), seg(p',q')) =$ **si** $nullv(V) \wedge nullv(V') \wedge d = 0$ **alors** $faux$
sinon $(s > a) \wedge (s <$ 1-$a) \wedge (t > b) \wedge (t <$ 1-$b)$

avec $V = gvp(p,q), V' = gvp(p',q'), U = gvp(p',p), d = pv(V',V)$

$s = \dfrac{ps(U,V)}{d}, t = \dfrac{ps(U,V')}{d}, a = \dfrac{\varepsilon}{nv(V)}, b = \dfrac{\varepsilon}{nv(V')}$

$intersection(seg(p,q), seg(p',q')) = p + t.V$

avec $V = gvp(p,q), V' = gvp(p',q'), U = gvp(p',p), t = \dfrac{pv(U,V')}{pv(V',V)}$

Préconditions $(s, t : Seg)$

prec $intersection(s,t) \equiv secant(s,t)$

Fin

Annexe B

Spécifications des dictionnaires et des files de priorité

B.1 Spécification d'ordres sur les brins

La spécification des files de priorité et des dictionnaires suppose la définition d'un ordre sur les brins d'une carte. Les files de priorité sont utilisées pour parcourir les brins de gauche à droite lors d'un balayage du plan. Elles nécessitent donc un ordre sur les sommets d'une carte. Cet ordre (strict), noté $<_s^C$, compare les 0-plongements des sommets de deux brins. En cas d'égalité des 0-plongements, les angles des arêtes des brins sont comparés. Cela permet, lorsque deux brins ont un sommet commun, de parcourir d'abord le brin dont l'arête se situe à gauche de ce sommet, puis le brin dont l'arête se situe à droite du sommet.

Les dictionnaires sont utilisés pour ordonner de bas en haut les brins actifs lors du balayage. Cet ordre, noté $<_a^C$ ou $>_a^C$, est donc défini sur les segments sur lesquels sont plongés les arêtes des brins. Cet ordre n'est pas détaillé. Le seul impératif est que ce soit un ordre strict, partiel et tel que si deux segments sont sécants, égaux ou si l'extrémité de l'un est incidente à l'autre, alors les segments ne sont pas comparables.

La spécification 2-CARTE-ORDONNEE définit ces trois ordres pour les 2-cartes, en étendant la spécification 2-CARTE-GEO3. Elle est donnée dans la table B.1. Les préconditions précisent que les ordres sont définis pour des brins correctement plongés.

B.2 Spécification des files de priorité

Les files de priorité sont spécifiées sous forme de tas, eux-mêmes définis comme des tableaux. La spécification TABLE de la table B.2 définit les tableaux d'éléments quelconques de sorte Elt. Elle est paramétrée par la spécification triviale TRIV qui définit uniquement la sorte Elt.

Les tableaux de sorte $Table$ sont construits à partir d'un tableau vide v et par insertions successives d'éléments par le générateur $a(T, i, x)$ qui ajoute au tableau T un élément x à l'emplacement d'indice i. Les opérations définies sont classiques pour les tableaux. La fonction $taille(T)$ retourne le nombre d'éléments du tableau. Les fonctions $premier(T)$ et $elt(T, i)$

TAB. B.1: *Spécifications d'ordres sur les sommets et arêtes*

Spéc *2-CARTE-ORDONNEE* **étend** *2-CARTE-GEO3* **avec**

Opérateurs

$\_ <_s^- \_ : 2Carte\ Brin\ Brin \longrightarrow Bool$ (*ordre sur les sommets*)
$\_ <_a^- \_ : 2Carte\ Brin\ Brin \longrightarrow Bool$ (*ordre sur les arêtes*)
$\_ >_a^- \_ : 2Carte\ Brin\ Brin \longrightarrow Bool$ (*ordre sur les arêtes*)

Axiomes $(M : 2Carte\ ;\ x, y : Brin)$

$x <_s^C y = (gem0(C,x) < gem0(C,y))$
$\quad \vee [(gem0(C,x) = gem0(C,y)) \wedge (angle(C,x) < angle(C,y))]$

$x <_a^C y = inf\_seg(gem1(C,x), gem1(C,y))$
$x >_a^C y = sup\_seg(gem1(C,x), gem1(C,y))$

Préconditions

prec $x <_s^C y \equiv$ **prec** $angle(C,x) \wedge$ **prec** $angle(C,y)$
prec $x <_a^C y \equiv$ **prec** $gem1(C,x) \wedge$ **prec** $gem1(C,y)$
prec $x >_a^C y \equiv$ **prec** $gem1(C,x) \wedge$ **prec** $gem1(C,y)$

Fin

donnent respectivement le premier et le i-ème élément du tableau. La fonction $appartient(T,x)$ teste l'existence de l'élément x dans T et $indice(T,x)$ donne l'indice d'un élément x de T.

Ces fonctions servent essentiellement pour définir les préconditions et les deux fonctions principales $modif(T,i,x)$ et $echange(T,i,j)$ qui permettent de modifier l'élément d'indice i de T et d'échanger deux éléments d'indices donnés. La précondition du générateur a assure que toutes les cases du tableau sont remplies successivement en incrémentant d'un les indices correspondants.

La spécification TAS de la table B.3 étend la spécification TABLE en instanciant son paramètre par la spécification des ordres pour les 2-cartes. Les brins joueront le rôle des éléments du tableau. La sorte Tas est une sous-sorte de la sorte $Table$. Cela signifie que les tas sont des tableaux particuliers ce qui permet d'utiliser toutes les opérations définies pour les tableaux sur un tas.

En examinant les profils des fonctions de la table B.3, on peut remarquer que seules trois opérations retournent des tas. Ces opérations sont le générateur v, le tas vide qui surcharge le tableau vide, et les fonctions $insert(T,M,x)$ et $supprime(T,M,x)$ qui décrivent l'insertion et la suppression d'un élément. Ces deux dernières fonctions assurent que leurs résultats sont bien des tas. L'utilisation de toute autre opération est un tableau puisqu'on ne peut plus assurer que le résultat est bien ordonné.

Ces autres opérations sont des modifications locales du tas utilisées lors de l'insertion et de la suppression. La première $echqueue(A,x)$ supprime l'élément x du tableau A et le remplace par le dernier élément. Le résultat est un tableau dont la taille est diminuée de un par rapport à celle du tableau initial. La fonction $minfils(A,M,i)$ donne l'indice du plus petit fils de l'élément d'indice i. Si cet élément n'a pas de fils, c'est-à-dire si c'est une feuille, la valeur retournée est 0. Remarquons la présence du paramètre M nécessaire pour obtenir les plongements des brins lors de leurs comparaisons.

Les fonctions $remonte(A,M,i)$ et $descent(A,M,i)$ sont utilisées pour permuter les élé-

TAB. B.2: *Spécification des tableaux*

Spéc *TABLE[T :TRIV]* **étend** *INT* **avec**

Sortes $Table$

Opérateurs

$v : \longrightarrow Table$	(*arbre vide*)
$a : Table\ Int\ Elt \longrightarrow Table$	(*ajout d'un élément*)
$taille : Table \longrightarrow Int$	(*taille d'un tableau*)
$premier : Table \longrightarrow Elt$	(*premier élément d'un tableau*)
$elt : Table\ Int \longrightarrow Elt$	(*élément d'indice donné*)
$appartient : Table\ Elt \longrightarrow Bool$	(*test d'existence d'un élément*)
$indice : Table\ Elt \longrightarrow Int$	(*indice d'un élément*)
$modif : Table\ Int\ Elt \longrightarrow Table$	(*modification d'un élément*)
$echange : Table\ Int\ Int \longrightarrow Table$	(*échange de deux élément*)

Axiomes $(A : Table\ ;\ i, j : Int\ ;\ x, y : Elt)$

$taille(v) = 0$

$taille(a(A, i, x)) = i$

$premier(a(A, i, x)) =$ **si** $i = 1$ **alors** x **sinon** $premier(A)$

$elt(a(A, i, x), j) =$ **si** $i = j$ **alors** x **sinon** $elt(A, j)$

$appartient(v, x) = false$

$appartient(a(A, i, x), y) =$ **si** $x = y$ **alors** $true$ **sinon** $appartient(A, y)$

$indice(a(A, i, x), y) =$ **si** $x = y$ **alors** i **sinon** $indice(A, y)$

$modif(a(A, i, x), j, y) =$ **si** $i = j$ **alors** $a(A, i, y)$
sinon $a(modif(A, j, y), i, x)$

$echange(A, i, j) = modif(modif(A, i, elt(A, j)), j, elt(A, i))$

Préconditions

prec $a(A, i, x) \equiv (i = 1 + taille(A))$
prec $premier(A) \equiv taille(A) > 0$
prec $elt(A, i) \equiv 1 \leq i \leq taille(A)$
prec $indice(A, x) \equiv appartient(A, x)$
prec $modif(A, i, x) \equiv 1 \leq i \leq taille(A)$
prec $echange(A, i, j) \equiv 1 \leq i \leq taille(A) \wedge 1 \leq j \leq taille(A)$

Fin

ments d'un tableau, afin de les réordonner pour obtenir un tas. La première remonte l'élément d'indice i dans l'arbre tant qu'il est plus petit que son père, c'est-à-dire l'échange avec l'élément d'indice $i/2$ et recommence avec ce dernier élément. La deuxième fonction descend l'élément d'indice i tant qu'il est plus grand qu'un de ses fils et que ce n'est pas une feuille. Pour cela, l'élément est échangé avec le plus petit de ses fils tant que l'indice de ce fils est différent de 0 et que l'élément est plus grand que ce fils.

Les fonctions $insert(T, M, x)$ et $supprime(T, M, x)$, qui ont été nommées i et s précédemment, sont définies uniquement pour des tas et retournent toujours des tas. Un nouvel élément est ajouté à la fin du tableau, puis remonté dans l'arbre pour rétablir l'ordre. Pour supprimer un élément, on l'échange avec le dernier élément, puis on remonte ou descend cet élément suivant qu'il est plus petit ou plus grand que l'élément qu'il remplace.

TAB. B.3: *Spécification des tas*

Spéc *TAS* **étend** *TABLE[2-CARTE-ORDONNEE]* **avec**

Sortes $Tas < Table, Elt = Brin$

Opérateurs

$v : \longrightarrow Tas$	(*tas vide*)
$echqueue : Table\ Elt \longrightarrow Table$	(*échange avec la dernière feuille*)
$minfils : Table\ 2Carte\ Int \longrightarrow Int$	(*plus petit fils, 0 pour une feuille*)
$remonte : Table\ 2Carte\ Int \longrightarrow Table$	(*remonté d'un élément*)
$descent : Table\ 2Carte\ Int \longrightarrow Table$	(*descente d'un élément*)
$insert : Tas\ 2Carte\ Elt \longrightarrow Tas$	(*insertion d'un nouvel élément*)
$supprime : Tas\ 2Carte\ Elt \longrightarrow Tas$	(*suppression d'un élément*)

Axiomes $(T : Tas\ ;\ A : Table\ ;\ M : 2Carte\ ;\ x : Elt)$

$echqueue(a(A,i,y),x) =$ **si** $x = y$ **alors** A **sinon** $modif(A, indice(A,x), y)$

$minfils(A,M,i) =$ **si** $2i+1 \leq taille(A)$
alors si $elt(A,2i) <_s^C elt(A,2i+1)$ **alors** $2i$ **sinon** $2i+1$
sinon si $2i \leq taille(A)$ **alors** $2i$ **sinon** 0

$remonte(A,M,i) =$ **si** $elt(A,i/2) <_s^C elt(A,i)$ **alors** A
sinon $remonte(echange(A,i,i/2),M,i/2)$

$descent(A,M,i) =$ **si** $minfils(A,M,i) = 0$ **alors** A
sinon si $elt(A,i) <_s^C elt(A, minfils(A,M,i))$ **alors** A
sinon $descent(echange(A,i,minfils(A,i)),M,minfils(A,i))$

$insert(T,M,x) = remonte(a(T,taille(T)+1,x),M,taille(T)+1)$

$supprime(T,M,x) =$ **si** $indice(T,x) = taille(T)$ **alors** $echqueue(T,x)$
sinon si $elt(T,taille(T)) <_s^C elt(T,indice(T,x))$
alors $remonte(echqueue(T,x),M,indice(T,x))$
sinon $descent(echqueue(T,x),M,indice(T,x))$

Préconditions

prec $echqueue(S,x) \equiv appartient(S,x)$
prec $minfils(S,M,i) \equiv 1 \leq i \leq taille(S)$
prec $remonte(S,M,i) \equiv 1 \leq i \leq taille(S)$
prec $descent(S,M,i) \equiv 1 \leq i \leq taille(S)$
prec $insert(S,M,x) \equiv$ **prec** $angle(C,x)$
prec $supprime(T,M,x) \equiv appartient(T,x) \wedge$ **prec** $angle(C,x)$

Fin

B.3 Spécification des dictionnaires

Les dictionnaires sont définis comme des arbres binaires ordonnés. La spécification des arbres binaires génériques ARBRE, donnée dans la table B.4, est paramétrée par la spécification TRIV. Elle définit les deux générateurs v, pour l'arbre vide, et $d(G, x, D)$ qui forme par enracinement un arbre avec deux arbres G et D et un élément x.

TAB. B.4: *Spécification des arbres binaires*

Spéc *ARBRE* **étend** *T :TRIV* **avec**

Sortes *Arbre*

Opérateurs

v : $\longrightarrow$ *Arbre*	(*arbre vide*)
d : *Arbre Elt Arbre* $\longrightarrow$ *Arbre*	(*générateur des arbres*)

Fin

La spécification DICO étend la spécification ARBRE dont le paramètre est instancié par la spécification des ordres pour les 2-cartes. Nous définissons un élément particulier *nil* qui est utilisé pour signifier qu'une recherche dans un dictionnaire a échoué. La sorte *Dico* est une sous-sorte de la sorte *Arbre* permettant de distinguer les arbres ordonnés des arbres génériques. Un dictionnaire est donc un arbre créé par les fonctions d'insertion $ins(D, M, x)$ et de suppression $sup(D, M, x)$.

Les fonctions décrites ici sont suffisamment classiques pour ne pas nécessiter d'explications. Notons simplement que la fonction *tinst* teste si l'insertion d'un élément est possible, c'est-à-dire teste si cet élément est comparable aux éléments du dictionnaire. Ceci est nécessaire car l'ordre sur les segments est partiel. La fonction *max* donne le plus grand élément d'un dictionnaire et la fonction *supmax* supprime cet élément.

La spécification RECHERCHE, qui étend DICO, définit les fonctions de recherche dans un dictionnaire. Ces fonctions parcourent le dictionnaire et effectuent les tests géométriques correspondants. Elles permettent de rechercher un segment égal, sécant ou co-incident au segment d'un brin donnée de la carte. Les deux dernières fonctions *gmaxinf* et *gminsup* retournent respectivement le plus grand (petit) brin de l'arbre dont le segment est inférieur (supérieur) à celui du brin donné de la carte. Ces fonctions sont utilisées pour rechercher des brins interagissant avec le brin courant dans l'ensemble des brins actifs, lors d'un balayage du plan.

TAB. B.5: *Spécification des dictionnaires*

Spéc *DICO* **étend** *ARBRE[2-CARTE-ORDONNEE]* **avec**

Sortes $Dico < Arbre, Elt = Brin$

Opérateurs

$nil : \longrightarrow Elt$	(*élément vide*)
$v : \longrightarrow Dico$	(*dictionnaire vide*)
$tins : Dico\ 2Carte\ Elt \longrightarrow Bool$	(*test de possibilité d'insertion*)
$ins : Dico\ 2Carte\ Elt \longrightarrow Dico$	(*insertion d'un élément*)
$max : Dico \longrightarrow Elt$	(*élément maximal d'un dico*)
$supmax : Dico \longrightarrow Dico$	(*suppression de l'élément maximal*)
$sup : Dico\ 2Carte\ Elt \longrightarrow Dico$	(*suppression d'un élément*)

Axiomes $(G, D : Dico\ ;\ M : 2Carte\ ;\ x, y : Elt)$

$tins(v, M, x) = true$

$tins(d(G, y, D), M, x) =$ **si** $x <_a^C y$ **alors** $tins(G, x)$
sinon si $x >_a^C y$ **alors** $tins(D, x)$ **sinon** $false$

$ins(v, M, x) = d(v, x, v)$

$ins(d(G, y, D), M, x) =$ **si** $x <_a^C y$ **alors** $d(ins(G, x), y, D)$
sinon $d(G, y, ins(D, x))$

$max(v) = nil$

$max(d(G, x, v)) = x$

$max(d(G, x, D)) = max(D)$

$supmax(v) = v$

$supmax(d(G, x, v)) = G$

$supmax(d(G, x, D)) = d(G, x, supmax(D))$

$sup(v, M, x) = v$

$sup(d(v, x, D), M, x) = D$

$sup(d(G, x, v), M, x) = G$

$sup(d(G, x, D), M, x) = d(supmax(G), max(G), D)$

$sup(d(G, y, D), M, y) =$ **si** $y <_a^C x$ **alors** $d(G, y, sup(D, x))$
sinon $d(sup(G, x), y, D)$

Préconditions

prec $ins(D, M, x) \equiv$ **prec** $gem1(C, x) \wedge tins(D, M, x) \wedge x \neq nil$

prec $max(G) \equiv \neg(G == v)$

prec $supmax(G) \equiv \neg(G == v)$

Fin

TAB. B.6: *Spécification des opérations de recherche dans un dictionnaire*

Spéc *RECHERCHE* **étend** *DICO* **avec**

Opérateurs

$gequal : Dico\ 2Carte\ Elt \longrightarrow Elt$	(*recherche d'une arête superposée*)
$gsecant : Dico\ 2Carte\ Elt \longrightarrow Elt$	(*recherche d'une arête sécante*)
$gincident : Dico\ 2Carte\ Elt \longrightarrow Elt$	(*recherche d'une arête incidente*)
$gmaxinf : Dico\ 2Carte\ Elt \longrightarrow Elt$	(*recherche le max des inf d'un élément*)
$gminsup : Dico\ 2Carte\ Elt \longrightarrow Elt$	(*recherche le min des sup d'un élément*)

Axiomes $(G, D : Dico\ ;\ M : 2Carte\ ;\ x, y : Elt)$

$gequal(v, M, x) = nil$

$gequal(t(G, y, D), M, x) =$ **si** $eqap(C, x, y)$ **alors** y
sinon si $x >_a^C y$ **alors** $gequal(D, M, x)$
sinon si $x <_a^C y$ **alors** $gequal(G, M, x)$
sinon nil

$gsecant(v, M, x) = nil$

$gsecant(d(G, y, D), x) =$ **si** $secant(C, x, y)$ **alors** y
sinon si $x <_a^C y$ **alors** $gsecant(G, x)$
sinon si $x >_a^C y$ **alors** $gsecant(D, x)$
sinon nil

$gincident(v, M, x) = nil$

$gincident(d(G, y, D), x) =$ **si** $incident2(C, x, y)$ **alors** y
sinon si $x <_a^C y$ **alors** $gincident(G, x)$
sinon si $x >_a^C y$ **alors** $gincident(D, x)$
sinon nil

$gmaxinf(v, M, x) = nil$

$gmaxinf(d(G, y, D), M, x) =$ **si** $x = y$ **alors** $max(G)$
sinon si $x <_a^C y$ **alors** $gmaxinf(G, M, x)$
sinon $gmaxinf(D, M, x)$

Préconditions

prec $gequal(D, M, x) \equiv$ **prec** $gem1(C, x)$
prec $gsecant(D, M, x) \equiv$ **prec** $gem1(C, x)$
prec $gincident(D, M, x) \equiv$ **prec** $gem1(C, x)$

Fin

Bibliographie

[1] J.R. Abrial. *The B-book : Assigning programs to meanings.* Cambridge University Press, 1996.

[2] D. Arquès and P. Koch. Modélisation de solides par les pavages. In *Proc. of Pixim*, pages 47–61, Paris, 1989.

[3] D. Badouel and G. Hégron. An evaluation of CSG trees based on polyhedral solids. In *Proc. of Eurographics*, Nice, 1988.

[4] P. Balbiani, V. Dugat, L. Fariñas del Cerro, and A. Lopez. *Eléments de géométrie mécanique.* Hermès, 1994.

[5] P. Baumann. A formal specification of a boundary representation. In *Proc. of Eurographics*, Nice, France, 1988.

[6] B.G. Baumgart. A polyhedron representation for computer vision. In *Proc. of AFIPS*, volume 44 of *National Computer Conference*, pages 589–596, 1975.

[7] M.O. Benouamer, D. Michelucci, and B. Peroche. Error-free boundary evaluation based on a lazy rational arithmetic : a detailled implementation. *Computer-Aided Design*, 26(6) :403–416, 1994.

[8] J.L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Computer*, 28 :643–647, 1979.

[9] J.A. Bergstra, J. Heering, and P. Klint. *Algebraic specification.* ACM Press, 1989.

[10] J.A. Bergstra and J.W. Klop. Conditional rewrite rules : confluence and termination. *Journal of Computer and System Sciences*, 32 :323–362, 1986.

[11] D. Bert and R. Echahed. Design and implantation of a generic, logic and fonctional programming language. In *Proc. of ESOP'86*, volume 213 of *LNCS*, pages 119–132, 1986.

[12] Y. Bertrand. *Spécification algébrique et réalisation d'un modeleur interactif d'objets géométriques volumiques.* Thèse de doctorat, Université L. Pasteur, Strasbourg, 1992.

[13] Y. Bertrand and J.F. Dufourd. Algebraic specification of a 3D-modeler based on hypermaps. *CVGIP : Graphical Models and Image Processing*, 56(1) :29–60, 1994.

[14] Y. Bertrand, J.F. Dufourd, J.F. Françon, and P. Lienhardt. Algebraic specification and development in geometric modeling. In *EATCS conf. TAPSOFT*, volume 668 of *LNCS*, pages 75–89, Orsay, France, 1993. Springer-Verlag.

[15] E. Bevers and J. Lewi. Proving termination of (conditional) rewrite systems. A semantic approach. *Acta Informatica*, 30 :537–568, 1993.

[16] J.D. Boissonat and M. Yvinec. *Géométrie algorithmique*. Ediscience international, 1995.

[17] B. Brüderlin. Automatizing geometric proofs and construction. *Computational geometry and its applications*, 1988.

[18] B. Brüderlin. Using geometric rewrite rules for solving geometric problems symbolically. *Theoretical Computer Science*, 116 :291–303, 1993.

[19] R.P. Brian and D. Singerman. Foundation of the theory of maps on surfaces with boundary. *Quart. Journal of Math. Oxford*, pages 17–41, 1985.

[20] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of ACM*, 39(1) :1–54, 1992.

[21] A. Corradini and U. Montanari. An algebra of graphs and graph rewriting. In *Proc. of conf. on Category Theory and Computer Science*, volume 530 of *LNCS*, pages 236–260, Paris, France, 1991. Springer-Verlag.

[22] A. Corradini and F. Rossi. Hyperedge replacement jungle rewriting for term-rewriting systems and logic programming. *Theoretical Computer Science*, 109 :7–48, 1993.

[23] B. Courcelle. Graph rewriting : an algebraic and logic approach. In *Formal models and semantics*, Handbook of Theoretical Computer Science, chapter 5, pages 194–241. Elsevier, 1990.

[24] G.A. Croker and W.F. Reinke. An editable nonmanifold boundary representation. *IEEE Computer Graphics & Applications*, pages 39–51, 1991.

[25] N. Dershowitz and J.P. Jouannaud. Rewrite systems. In *Formal models and semantics*, Handbook of Theoretical Computer Science, chapter 6, pages 243–320. Elsevier, 1990.

[26] N. Dershowitz and M. Okada. A rational for conditional equational programming. *Theoretical Computer Science*, 75 :111–138, 1990.

[27] D.P. Dobkin and M.J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. In *Proc. of 3rd ACM Symposium on Computional Geometry*, pages 86–99, 1987.

[28] D.A. Duce, E.V. Fielding, and L.S. Marshall. Formal specification of a small example based on GKS. *ACM Trans. on Graphics*, 7(3) :180–197, 1988.

[29] D.A. Duce and F. Paterno. Lotos description of GKS-R functionality. In Springer-Verlag, editor, *Eurographics workshop on formal methods in computer graphics*, Marina di Carrara, Italy, 1991.

[30] J.F. Dufourd. Algebraic map-based topological kernel for polyhedron modellers : algebraic specification and logic prototyping. In *Proc. of Eurographics*, pages 649–662, 1989.

[31] J.F. Dufourd. Boundary representation revisited with a new premap axiomatics. In Springer-Verlag, editor, *Eurographics workshop on formal methods in computer graphics*, Marina di Carrara, Italy, juin 1991.

[32] J.F. Dufourd. Formal specification of topological subdivisions using hypermaps. *Computer-Aided Design*, 23(2) :99–116, 1991.

[33] J.F. Dufourd. An OBJ3 functional specification for the boundary representation. In ACM Press, editor, *First ACM-SIGGRAPH Symp. on Solid Modeling*, pages 61–72, Austin, Texas, 1991.

[34] J.F. Dufourd. Algebras and formal specifications in geometric modeling. *The Visual Computer*, 1997.

[35] J.F. Dufourd, C. Gross, and J.C. Spehner. A digitization algorithm for the entry of planar maps. In *Proc. of Computer Graphics International Conf.*, pages 649–661, Leeds, U.K., 1989. Springer-Verlag.

[36] H. Ehrig and B. Mahr. *Fundamentals of algebraic specification 1. Equations and initial semantics*, volume 6 of *EATCS Monograph on Theoretical Computer Science*. Springer-Verlag, 1985.

[37] C. Fiorio. *Approche interpixel en analyse d'images, une topologie et des algorithmes de segmentation*. Thèse de doctorat, Université Montpellier II, 1995.

[38] S. Fortune. Polyhedral modelling with multiprecision integer arithmetic. *Computer-Aided Design*, 29(2) :123–133, 1997.

[39] A. Fousse, Y. Bertrand, J.F. Dufourd, and J. Françon. Localisation des points d'un maillage généré en vue de calculs en différences finies, modélisation du sous-sol. In *Colloque BRGM-ENSMP-INRIA*, Paris, 1997.

[40] M. Gangnet, J.C. Hervé, T. Pudet, and J.M. van Thong. Incremental computation of planar maps. In *Proc. of ACM-SIGGRAPH Conf. on Computer Graphics*, volume 23, pages 345–354, Boston, July 1989.

[41] Y. Gardan and E. Perrin. An algorithm reducing 3D boolean operations to a 2D problem : concepts and results. *Computer-Aided Design*, 28(4) :277–287, 1996.

[42] J.A. Goguen, T. Winkler, J. Meseguer, K. Futasugi, and J.P. Jouannaud. *Introducing OBJ*, cambridge university press edition, 1992.

[43] L. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry : building robust algorithms from imprecise computations. In *5th ACM Symposium on Computional Geometry*, Québec, 1989.

[44] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoï diagrams. *ACM Trans. on Graphics*, 4(2) :74–123, April 1985.

[45] E.L. Gursoz, Y. Choi, and F.B. Printz. Boolean set operations on non-manifold boundary representation objects. *Computer-Aided Design*, 23(1) :33–39, 1991.

[46] J.V. Guttag and S.J. Garland. *A guide to LP, the Larch Prover*. Digital, Systems Research Center, 1991.

[47] J.V. Guttag and J.J. Horning. Larch : language and tools for formal specification. In *Texts and Monographs in Computer Science*. Springer-Verlag, 1993.

[48] Y. Halbwachs, G. Courrioux, X. Renaud, and P. Repusseau. Topological and geometric characterization of fault networks using 3-dimensional generalized maps. *Mathématical Geologie*, 28(5) :625–656, 1996.

[49] C.M. Hoffmann, J.E Hopcroft, and M.S. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics & Applications*, 9(6) :50–59, 1989.

[50] Institut National de Recherche en Informatique et en Automatique. *The Coq proof assistant user's guide*, 1993.

[51] A. Jacques. Constellations et graphes topologiques. In *Colloque Math. Soc.*, pages 657–672, 1970.

[52] C.B. Jones. Systematic software development using VDM. In *Texts and Monographs in Computer Science*. Prentice-Hall International, 2nd edition, 1990.

[53] J.P. Jouannaud and P. Lescanne. La réécriture. *Technique et Science Informatiques*, 5(6) :433–452, 1986.

[54] P. Lienhardt. Subdivision of N-dimensional spaces and N-dimensional generalized maps. In *5th ACM Conf. on Computational Geometry*, pages 228–236, Saarbrücken, Germany, 1989.

[55] P. Lienhardt. Topological models for boundary representation : a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1) :59–82, 1991.

[56] W.R. Mallgren. *Formal specification of interactive graphic programming languages*. ACM Dist. Dissertation. MIT Press, USA, 1982.

[57] P. Martin and D. Martin. Les algorithmes de calcul de l'intersection de solides définis par leur bord. *Revue de CFAO*, 2(4) :7–27, 1986.

[58] K. Mehlhorn. *Multi dimensional searching and computational geometry*, volume 3 of *Data structures and algorithms*. Springer-Verlag, 1984.

[59] V.J. Milenkovic. Practical methods for set operations on polygons using exact arithmetic. In *Proc. of Canadian Conf. on Computational Geometry*, Québec, 1995.

[60] M. Mäntylä and R. Sulonen. GWB : a solid modeler with Euler operators. *IEEE Computer Graphics & Applications*, 2(7) :17–31, 1982.

[61] F. Nielsen. *Algorithmes géométriques adaptatifs*. Thèse de doctorat, Université de Nice Sophia-antipolis, 1996.

[62] J. Nievergelt and F.P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Com. of ACM*, 25(10) :739–747, 1982.

[63] F. Puitg and J.F. Dufourd. Spécifications en modélisation géométrique par la théorie des constructions. In *Journées du GDR de Programmation*, Grenoble, 1995.

[64] L.K. Putnam and P.A. Subrahmanyam. Boolean operations on n-dimensional objects. *IEEE Computer Graphics & Applications*, 6(6) :43–51, 1986.

[65] A.A.G. Requicha. Representation for rigid solids : theory, methods and systems. *Computing Survey*, 12(4) :437–463, 1980.

[66] A.A.G. Requicha and H.B. Voelcker. Boolean operations in solid modeling : boundary evaluation and merging algorithms. In *Proc. of IEEE*, volume 73, january 1985.

[67] J.R. Rossignac and M.A O'Connor. SGC : A dimension-independent model for pointsets with internal structures and incomplete boundary. *Computer-Aided Design*, 1991.

[68] J. Rouyer. *Développements d'algorithmes dans le calculs des constructions*. Thèse de doctorat, Institut National Polytechnique de Lorraine, 1994.

[69] E. Sopena. *Réécriture d'hypercartes combinatoires et conception d'un langage pour la programmation d'algorithmes de graphes*. Thèse de doctorat, Université Bordeaux I, 1986.

[70] J.C. Spehner. Merging in maps and in pavings. *Theoretical Computer Science*, pages 205–232, 1991.

[71] J.M. Spivey. *The Z notation – A reference manual.* Prentice-Hall International, 2nd edition, 1992.

[72] R.B. Tilove. Set membership classification : A unified approach to geometric intersection problems. *IEEE Trans. Computer*, 29(10) :874–883, 1980.

[73] W. Tutte. Graph theory. In *Encyclopedia of Mathematics and its Applications*, chapter 21. Cambridge University Press, 1984.

[74] A. Vince. Combinatorial maps. *Journal of Combinatorial Theory*, pages 1–21, 1983.

[75] K.J. Weiler. Polygon comparison using a graph representation. In *Proc. of ACM-SIGGRAPH Conf. on Computer Graphics*, volume 14, pages 10–19, 1980.

[76] K.J. Weiler. The radial edge structure : a topological representation for non-manifold geometric modeling. In *Geometric Modeling for CAD Applications*, pages 37–68. Springer-Verlag, 1986.

[77] M. Wirsing. Algebraic specifications. In *Formal models and semantics*, Handbook of Theoretical Computer Science, chapter 13, pages 675–788. Elsevier, 1990.

[78] J.B. Wordsworth. *Software development with Z : a practical approach to formal methods in software engineering.* Addison-Wesley, Workingham, England, 1992.

[79] C.K. Yap. Towards exact geometric computation. *Computational Geometry, Theory and Applications*, 7 :3–23, 1997.